

PrimeTime® User Guide

Version T-2022.03, March 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

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

Disclaimer

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

Trademarks

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

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

Free and Open-Source Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

New in This Release	35
Related Products, Publications, and Trademarks	35
Conventions	36
Customer Support	37
<hr/>	
1. Introduction to PrimeTime	38
PrimeTime Capabilities	38
PrimeTime Product Tiers and Licenses	39
PrimeTime Add-On Tools	41
Using PrimeTime in the Implementation Flow	42
Compatibility With Synopsys Implementation Tools	44
Overview of Static Timing Analysis	44
Timing Paths	45
Delay Calculation	47
Constraint Checks	48
Example of Setup and Hold Checks for Flip-Flops	48
Example of Setup and Hold Checks for Latches	50
Timing Exceptions	51
<hr/>	
2. Getting Started	53
Installing the PrimeTime Software and Licenses	53
Configuring the PrimeTime Work Environment	53
Starting a PrimeTime Session	54
License Checkouts	55
Entering pt_shell Commands	56
Getting Help on the Command Line	56
The PrimeTime Static Timing Analysis Flow	56
Using Tcl/Tk in PrimeTime	59
Tcl Packages and Autoload	59
Support of the incr Tcl Extension	60

Contents

Checking and Compiling Scripts With the TclPro Toolkit	60
Installing TclPro Tools	60
Checking the Syntax in Scripts With the TclPro Checker	61
Creating Bytecode-Compiled Scripts With the TclPro Compiler	63
Debugging Scripts With the TclPro Debugger	64
Limiting System Messages	64
Saving and Reviewing System Messages	65
Ending a PrimeTime Session	66
Saving a PrimeTime Session	66
Exiting a PrimeTime Session	67
Command Log File	67
<hr/>	
3. Licensing	69
How Licenses Work	69
License Chains	69
Local-Process Licensing	71
License Management in Distributed Analysis	72
Distributed Scenario-Based Licensing	74
Distributed Core-Based Licensing	75
SMVA Licensing	76
Controlling License Behaviors	78
Manually Controlling License Usage	78
Incremental License Handling	79
License Pooling	80
License Queuing	80
License Autoreduction	81
<hr/>	
4. Managing Performance and Capacity	82
High Capacity Mode Options	82
Advanced Data Management	83
Threaded Multicore Analysis	83
Configuring Threaded Multicore Analysis	84
Executing Commands in Parallel	84
The parallel_execute Command	85
The redirect -bg (Background) Command	86
The parallel_foreach_in_collection Command	86

Contents

Threaded Multicore Parasitics Reading	88
Threaded Multicore Path-Based Analysis	88
Distributed Multi-Scenario Analysis	89
Definition of Terms	90
Overview of the DMSA Flow	91
Preparing to Run DMSA	92
DMSA Usage Flow	93
Distributed Processing Setup	97
Starting the Distributed Multi-Scenario Analysis Mode	98
Managing Compute Resources	98
Creating Scenarios	103
Specifying the Current Session and Command Focus	104
Executing Commands Remotely	105
DMSA Batch Mode Script Example	106
Baseline Image Generation and Storage	109
Host Resource Affinity	109
Scenario Variables and Attributes	111
Manager Context Variables	111
Worker Context Variables and Expressions	111
Setting Distributed Variables	112
Getting Distributed Variable Values	112
Merging Distributed Variable Values	113
Synchronizing Object Attributes Across Scenarios	114
Merged Reporting	115
get_timing_paths	116
report_analysis_coverage	116
report_clock_timing	117
report_constraint	118
report_min_pulse_width	120
report_si_bottleneck	120
report_timing	121
Loading Scenario Design Data Into the Manager	124
Loading Netlist and Library Data	124
Loading Scenario-Specific Design Data	127
Saving and Restoring Your Session	127
License Resource Management	128
Limiting License Usage in a DMSA Analysis	128
Worker Process Fault Handling	129
Messages and Log Files	131
Interactive Messages	131
Progress Messages	131

Contents

User Control of Task Execution Status Messages	132
Error Messages	132
Warning Messages	132
Log Files	133
Command Output Redirection	134
DMSA Variables and Commands	135
DMSA Variables	135
DMSA Commands	135
Commands Not Allowed on Worker Processes	137
Limitations of DMSA	137
HyperGrid Distributed Analysis	137
Overview of the HyperGrid Flow	138
Partitioning the Design	138
The Manager Process and the Worker Processes	140
The Distributed Analysis Working Directory	141
Preparing for Distributed Analysis	141
Creating a Wrapper Script	141
Running the Distributed Analysis	142
Querying Attributes in a HyperGrid Analysis	143
Saving a Distributed Session	144
Commands With HyperGrid Distributed Analysis Support	145
Limitations of HyperGrid Distributed Analysis	146
Reporting the CPU Time and Memory Usage	146
Flow Summary Reports	148
The Flow Summary Report Structure	149
Generating a Flow Summary Report	152
Defining Custom Flow Segments	153
Writing to Multiple Flow Summary Reports	154
Limitations	155
Profiling the Performance of Tcl Scripts	155
<hr/>	
5. Working With Design Data	158
Logic Libraries	158
Reading and Linking the Design	159
Identifying and Verifying the Required Design Files	160
Per-Instance Link Library Paths	161
link_path_per_instance	162
set_link_lib_map	162

Contents

Working With Design Objects	163
Setting the Current Design and Current Instance	165
Working With Attributes	165
Saving Attributes	166
Saving and Restoring Sessions	166
Saving Sessions	167
Saving Version-Compatible Sessions	167
Including ECO Physical Data in Saved Sessions	169
Restoring Sessions	169
Restoring Version-Compatible Sessions	170
Creating Test Cases	171
Controlling What Design Logic to Keep	171
Including the Logic Libraries	172
Specifying the Output Directory	172
Creating a Multiple-Instance Module Test Case	172
Limitations	173
<hr/>	
6. Constraining the Design	174
Timing Constraints	174
Input Delays	175
Using Input Ports Simultaneously for Clock and Data	177
Output Delays	177
Drive Characteristics at Input Ports	178
Setting the Port Driving Cell	179
Setting the Port Drive Resistance	180
Setting a Fixed Port Transition Time	180
Displaying Drive Information	180
Removing Drive Information From Ports	180
Port Capacitance	181
Wire Load Models	182
Setting Wire Load Models Manually	182
Automatic Wire Load Model Selection	183
Setting the Wire Load Mode	183
Reporting Wire Load Models	185
Slew Propagation	185

Contents

Design Rule Constraints	186
Maximum Transition Time	186
Multiple Threshold and Derating Values in Libraries	187
Specifying the Maximum Transition Constraint	188
Evaluating the Maximum Transition Constraint	190
Minimum Capacitance	191
Maximum Capacitance	191
Constraining Rise and Fall Maximum Capacitance	192
Frequency-Based Maximum Capacitance Checks	193
Maximum Capacitance Checking With Case Analysis	194
Maximum Fanout Load	194
Fanout Load Values for Output Ports	194
Ideal Networks	195
Propagating Ideal Network Properties	195
Using Ideal Networks	196
Using Ideal Latency	197
Using Ideal Transition	197
Reporting of Ideal Clocks in Full Clock Expanded Mode	198
Checking the Constraints	198
Design Reports	199
Constraint Checking With the <code>check_timing</code> Command	200
Detailed Constraint Checking and Debugging	201
<hr/>	
7. Clocks	204
Clock Overview	204
Specifying Clocks	205
Creating Clocks	205
Creating a Virtual Clock	206
Selecting Clock Objects	207
Applying Commands to All Clocks	207
Removing Clock Objects	207
Specifying Clock Characteristics	207
Setting Clock Latency	208
Setting Propagated Latency	208
Specifying Clock Source Latency	208
Dynamic Effects of Clock Latency	210
Setting Clock Uncertainty	211
Setting the Clock Jitter	213

Contents

Removing the Clock Jitter	213
Reporting the Clock Jitter	214
Estimating Clock Pin Transition Time	214
Checking the Minimum Pulse Width	215
Pulse Width Measurement Thresholds	216
Voltage Swing Checking	217
Minimum Period Checking	218
Using Multiple Clocks	219
Synchronous Clocks	220
Asynchronous Clocks	222
Exclusive Clocks	222
Multiplexed Clock Exclusivity Points	228
Setting Clock Exclusivity Points Explicitly	229
Inferring Clock Exclusivity Points Automatically	229
Reporting Exclusivity Points	230
Alternative Methods	230
Removing Clocks From Analysis	231
Clock Sense	231
Specifying Pulse Clocks	235
Constraining Pulse Widths in the Fanout of Pulse Generator Cells	238
Constraining Transition Times for Pulse Generator Cells	240
Timing PLL-Based Designs	241
Usage for PLL Timing	242
Sequential Cells on a Feedback Path	242
PLL Drift and Jitter	243
CRPR Calculations for PLL Paths	244
Reporting and Timing Checks	245
Requirements for PLL Library Cells	245
Specifying Clock-Gating Setup and Hold Checks	246
Disabling or Restoring Clock-Gating Checks	249
Specifying Internally Generated Clocks	249
Specifying a Divide-by-2 Generated Clock	250
Creating a Generated Clock Based on Edges	251
Creating Clock Senses for Pulse Generators	251
Creating a Divide-by Clock Based on Falling Edges	253
Shifting the Edges of a Generated Clock	255
Multiple Clocks at the Source Pin	256
Selecting Generated Clock Objects	256

Contents

Reporting Clock Information	257
Removing Generated Clock Objects	257
Generated Clock Edge Specific Source Latency Propagation	257
Clock Mesh Analysis	258
Overview of Clock Mesh Analysis	258
Performing Clock Mesh Analysis	261
Clock Network Simulation Commands	263
sim_setup_simulator	263
sim_setup_library	264
sim_setup_spice_deck	264
sim_validate_setup	264
sim_analyze_clock_network	264
<hr/>	
8. Timing Paths and Exceptions	266
Timing Path Groups	266
Path Timing Reports	267
Path Timing Calculation	269
Specifying Timing Paths	270
Multiple Through Arguments	270
Rise/Fall From/To Clock	271
Reporting of Invalid Startpoints or Endpoints	276
Timing Exceptions	276
Overview of Timing Exceptions	277
Single-Cycle (Default) Path Delay Constraints	278
Path Delay for Flip-Flops Using a Single Clock	278
Path Delay for Flip-Flops Using Different Clocks	279
Setting False Paths	283
Setting Maximum and Minimum Path Delays	284
Setting Multicycle Paths	285
Specifying Timing Margins	290
Specifying Exceptions Efficiently	290
Exception Order of Precedence	292
Exception Type Priority	293
Path Specification Priority	293
Reporting Exceptions	294
Reporting Timing Exceptions	296
Reporting Exceptions Source File and Line Number Information	299
Checking Ignored Exceptions	301

Contents

Removing Exceptions	301
Saving and Restoring Timing Path Collections	302
9. Operating Conditions	304
Operating Conditions	304
Interconnect Model Types	305
Setting Operating Conditions	306
Creating Operating Conditions	307
Operating Condition Information	307
Operating Condition Analysis Modes	308
Minimum and Maximum Delay Calculations	309
Minimum-Maximum Cell and Net Delay Values	310
Setup and Hold Checks	312
Path Delay Tracing for Setup and Hold Checks	312
Setup Timing Check for Worst-Case Conditions	313
Hold Timing Check for Best-Case Conditions	313
Path Tracing in the Presence of Delay Variation	314
Specifying the Analysis Mode	315
Single Operating Condition Analysis	315
On-Chip Variation Analysis	316
Using Two Libraries for Analysis	318
Derating Timing Delays	319
Derating Options	320
Conflicting Derating Settings	322
Derating Negative Delays	322
Clock Reconvergence Pessimism Removal	323
On-Chip Variation Example	323
Reconvergent Logic Example	324
Minimum Pulse Width Checking Example	325
CRPR Reporting	327
Derating CRPR for Different Transitions at the Common Point	327
Minimum Pulse Width Example	329
CRPR Merging Threshold	329
CRPR and Crosstalk Analysis	330
CRPR With Dynamic Clock Arrivals	330
Transparent Latch Edge Considerations	331
Reporting CRPR Calculations	331

Contents

Clock On-Chip Variation Pessimism Reduction	332
---	-----

10. Delay Calculation	335
Overview of Delay Calculation	335
Nonlinear Delay Models	337
Composite Current Source Timing Models	338
Pin Capacitance Reporting	340
Guidelines for Characterizing Design Rule Constraints	341
Resolving the CCS Extrapolation Warning Message (RC-011)	342
Cross-Library Voltage and Temperature Scaling	342
Scaling for Multirail Level Shifter Cells	343
Scaling Timing Derates in Voltage Scaling Flows	345
Excluding Rails From Multirail Scaling Library Groups	346
Waveform Propagation	346
Characterization Trip Points	347
Fast Multidrive Delay Analysis	349
Parallel Driver Reduction	349
Invoking Parallel Driver Reduction	352
Working With Reduced Drivers	352
Multi-Input Switching Analysis	353
Multi-Input Switching Analysis Modes	354
Defining Library Timing Arc Coefficients	355
Configuring Multi-Input Switching Analysis	358
Multi-Input Switching Analysis Example	360
Arrival Window Overlap Checking	360
Including or Excluding Specific Cells in MIS Analysis	361
Scaling the Tool-Computed Advanced MIS Derates	362
Unit Delay Analysis	363
11. Back-Annotation	365
SDF Back-Annotation	365
Reading SDF Files	366
Annotating Timing From a Subdesign Timing File	367
Annotating Load Delay	367
Annotating Conditional Delays From SDF	368
Annotating Timing Checks	370

Contents

Reporting Delay Back-Annotation Status	370
Reporting Annotated or Nonannotated Delays	370
Reporting Annotated or Nonannotated Timing Checks	371
Faster Timing Updates in SDF Flows	371
Annotating Delays, Timing Checks, and Transition Times	372
Annotating Delays	372
Annotating Timing Checks	373
Annotating Transition Times	373
Writing an SDF File	374
SDF Constructs	374
SDF Delay Triplets	375
SDF Conditions and Edge Identifiers	375
Reducing SDF for Clock Mesh/Spine Networks	376
PORT Construct	377
Normalizing Multidriven Arcs for Simulation	378
Writing VITAL Compliant SDF Files	380
Writing a Mapped SDF File	381
Specifying Timing Labels in the Library	381
Specifying the min_pulse_width Constraint	382
Using SDF Mapping	383
Supported SDF Mapping Functions	385
SDF Mapping Assumptions	388
Bus Naming Conventions	388
Labeling Bus Arcs	389
SDF Mapping Limitations	390
Mapped SDF File Examples	390
Writing Compressed SDF Files	398
Writing SDF Files Without Setup or Hold Violations	398
Setting Lumped Parasitic Resistance and Capacitance	398
Setting Net Capacitance	399
Setting Net Resistance	399
Detailed Parasitics	400
Reading Parasitic Files	401
Reading Multiple Parasitic Files	402
Applying Location Transformations to Parasitic Data	404
Reading Parasitics With Multiple Physical Pins	405
Reading a Single Corner From Multicorner Parasitic Data	406
Checking the Annotated Nets	407
Scaling Parasitic Values	407

Contents

Global Parasitic Scaling	408
Net-Specific Parasitic Scaling	408
Reporting Scaled Parasitics	409
Resetting Scaled Parasitics	409
Scaling Parasitics in One Block	409
Reading Parasitics for Incremental Timing Analysis	410
Limitations of Parasitic Files	410
Incomplete Annotated Parasitics	411
Selecting a Wire Load Model for Incomplete Nets	412
Completing Missing Segments on the Net	413
Back-Annotation Order of Precedence	414
Reporting Annotated Parasitics	414
Removing Annotated Data	414
GPD Parasitic Explorer	415
Enabling the Parasitic Explorer Feature	416
Parasitic Resistor and Capacitor Collections	416
Querying GPD Data Stored on Disk	417
Reporting GPD Properties	417
Setting GPD Annotation Properties	418
Getting GPD Corners and Layers	419
Parasitic Explorer Without a Netlist	419
12. Case and Mode Analysis	421
Case Analysis	421
Propagation of Constants that Control Logic Values	422
Configuring Case Analysis and Constant Propagation	423
Setting and Removing Case Analysis Values	424
Enabling Case Analysis Propagation Through Sequential Cells	425
Evaluating Conditional Arcs Using Case Analysis	426
Disabling Scan Chains With Case Analysis	426
Performing Case Analysis	427
Basic Case Analysis	428
Detailed Case Analysis	428
Standalone PrimeTime Constraint Consistency Checking	429
Reporting Disabled Arcs and Tracing Constant Propagation	429
Mode Analysis	431
Setting Cell Modes	432
Setting Modes Directly on Cell Instances	432

Contents

Setting Cell Modes Indirectly Using Design Modes	432
Mapping Design Modes	434
Placing Cells Into Modes by Using Case Analysis	435
Reporting Modes	437
Mode Merging for Scenario Reduction	437
Running Mode Merging	438
Debugging Mode Merging Conflicts	439
<hr/>	
13. Variation	441
Advanced On-Chip Variation	441
The AOCV Flow	441
Graph-Based AOCV Analysis	442
Path-Based AOCV Analysis	442
Configuring Advanced On-Chip Variation Analysis	443
Importing AOCV Information	444
Specifying AOCV Derating Tables	444
File Format for AOCV	445
AOCV Table Groups	448
Specifying Depth Coefficients	449
Guard-Banding in AOCV	450
Incremental Timing Derating	450
Using OCV Derating in AOCV Analysis	452
Querying AOCV Derating on Timing Paths	452
Parametric On-Chip Variation (POCV)	453
Variables and Commands for Parametric On-Chip Variation	454
Preparing Input Data for Parametric On-Chip Variation	456
POCV Single Coefficient Specified in a Side File	456
POCV Slew-Load Table in Liberty Variation Format	457
Importing a SPEF File With Physical Locations	459
Enabling Parametric On-Chip Variation Analysis	459
Loading the Parametric On-Chip Variation Input Data	460
Specifying Guard Banding	460
Scaling the Parametric On-Chip Variation Coefficient	461
Enabling Constraint and Slew Variation	461
Enabling Analysis With Moment-Based Modeling	462
Reporting Parametric On-Chip Variation Results	463
Report the POCV Coefficient and Derating	463
Report the POCV Analysis Results	464
Statistical Graph Merging Pessimism	467

Contents

Querying POCV Slack and Arrival Attributes	468
<hr/>	
14. Multivoltage Design Flow	470
Multivoltage Analysis Requirements	470
Specifying Power Supply Connectivity	471
UPF-Specified Supply Connectivity	471
Netlist-Inferred Supply Connectivity	472
Reading Block Boundary Information From UPF	472
UPF Commands	473
UPF Supply Sets	477
UPF Supply Set Handles	477
Identifying the Power and Ground Supply Nets With Attributes	479
Virtual Power Network	480
Setting Voltage and Temperature	481
Library Support for Multivoltage Analysis	483
Scaling Library Groups	483
Cell Alternative Library Mapping	484
Per-Instance Link Library Paths	485
Support for Fine-Grained Switch Cells in Power Analysis	485
Multivoltage Reporting and Checking	486
Collection (get_*) Commands	486
Reporting Power Supply Network Information	488
Querying Power and Ground Pin Attributes	491
Using the check_timing Command	493
Golden UPF Flow	497
Multiple Supply Voltage Analysis	499
Multivoltage Analysis Overview	499
Simultaneous Multivoltage Analysis (SMVA)	500
IR Drop Annotation	501
Signal Level Checking With IR Drop	501
Using Ansys RedHawk-SC With Timing Analysis	502
Writing Ansys RedHawk-SC STA Files	502
Performing Dynamic Voltage Drop Analysis	503
Using Rail Map Files to Apply Scaling/Offset Adjustments	506
<hr/>	
15. SMVA Graph-Based Simultaneous Multivoltage Analysis	508
Overview of SMVA	508
SMVA Requirements	509

Contents

Configuring SMVA Analysis	510
Defining Named SMVA Voltage References	511
Reporting Timing Paths in an SMVA Analysis	512
DVFS Scenarios	514
DVFS Scenario Concepts	514
DVFS Scenario Collection Objects	515
Propagated DVFS Scenarios	516
Using DVFS Scenarios to Control Command Scope	516
Querying DVFS Scenarios	517
Applying DVFS Scenarios to Commands and Attributes	518
Applying DVFS Scenarios to Individual Commands	518
Applying DVFS Scenarios to Attribute Queries	519
Applying DVFS Scenarios to Scripts	519
Setting a DVFS Scenario Context for Command Execution	520
Disabling and Enabling DVFS Scenarios	520
DVFS Scenario Specification Precedence	521
DVFS-Related Object Attributes	521
Using SMVA Analysis With Other PrimeTime Features	522
HyperScale Hierarchical Analysis	522
Context Characterization Flow	525
Voltage-Scaled Timing Derates in SMVA Flows	526
Extracted Timing Models	527
Timing Arc Multiple-Rail Delay Dependency	527
Noise Analysis	529
Usage Examples	530
SMVA Analysis for All Voltage Conditions	530
SMVA Analysis for Specific DVFS Constraints	532
Commands and Attributes With DVFS Scenario Support	534
Commands With DVFS Scenario Support	534
Attributes With DVFS Scenario Support	536
Feature Compatibility	539
<hr/>	
16. Signal Integrity Analysis	540
Overview of Signal Integrity and Crosstalk	540
Crosstalk Delay Effects	541
Delta Delay and Fanout Stage Effect	541
Crosstalk Noise Effects	542

Contents

Aggressor and Victim Nets	543
Timing Windows and Crosstalk Delay Analysis	544
Cross-Coupling Models	544
Crosstalk Delay Analysis	545
Performing Crosstalk Delay Analysis	546
How PrimeTime SI Operates	547
PrimeTime SI Variables	549
Logical Correlation	550
Electrical Filtering	551
Usage Guidelines	553
Preparing to Run Crosstalk Analysis	553
Using check_timing	554
Including or Excluding Specific Nets From Crosstalk Analysis	555
Initial Crosstalk Analysis Run	560
Timing Window Overlap Analysis	561
Clock Groups	568
Composite Aggressors	573
Ideal Aggressors	576
Path-Based Crosstalk Analysis	578
PrimeTime SI Crosstalk Delay Calculation Using CCS Models	578
Waveform Propagation	580
Annotated Delta Delays	581
Iteration Count and Exit	581
Timing Reports	582
Viewing the Crosstalk Analysis Report	582
Bottleneck Reports	584
Crosstalk Net Delay Calculation	585
Reporting Crosstalk Settings	586
Double-Switching Detection	587
Invoking Double-Switching Error Detection	589
How Double-Switching Is Detected	590
Reporting Double-Switching Violations	591
Fixing Double-Switching Violations	591
Static Noise Analysis	591
Static Noise Analysis Overview	592
Noise Bump Characteristics	594
Noise Bump Calculation	596
Noise-Related Logic Failures	597
PrimeTime SI Noise Analysis Flow	599
Noise Analysis Commands	601
Performing Noise Analysis	603

Contents

Reporting Noise Analysis Results	609
Setting Noise Bumps	614
Performing Noise Analysis with Incomplete Library Data	616
Noise Modeling With CCS Noise Data	619
Noise Immunity	620
CCS Noise Analysis for Unbuffered-Output Latches	620
Noise Modeling With Nonlinear Delay Models	622
Steady-State I-V Characteristics	623
Noise Immunity	627
Propagated Noise Characteristics	636
<hr/>	
17. Advanced Analysis Techniques	639
Parallel Arc Path Tracing	639
Support for Retain Arcs	640
Asynchronous Logic Analysis	641
Combinational Feedback Loop Breaking	642
Unrelated Clocks	643
Three-State Bus Analysis	644
Limitations of the Checks	645
Disabling the Checks	645
Bus Contention	645
Floating Buses	646
Three-State Buffers	646
Performing Transient Bus Contention Checks	646
Performing Floating Bus Checks	648
Data-to-Data Checking	649
Data Check Examples	649
Data Checks and Clock Domains	652
Library-Based Data Checks	653
Time Borrowing in Latch-Based Designs	653
Borrowing Time From Logic Stages	653
Latch Timing Reports	655
Maximum Borrow Time Adjustments	658
Time Borrowed and Time Given	663
Limiting Time Borrowing	665
Advanced Latch Analysis	666
Enabling Advanced Latch Analysis	667

Contents

Breaking Loops	668
Specifying Loop-Breaker Latches	668
Finding Loop-Breaker Latches	668
Latch Loop Groups	668
Listing Collections of Latch Loop Groups	669
Reporting Latch Loop Groups	669
Specifying the Maximum Number of Latches Analyzed per Path	670
Timing Exceptions Applied to Latch Paths	670
False Path Exceptions	671
Multicycle Path Exceptions	671
Maximum and Minimum Delay Exceptions	672
Clock Groups	672
Specification of Exceptions on Throughpaths	674
Reporting Paths Through Latches	674
Filtering Paths That Arrive Early at Intermediate Latches	676
Reporting Through Loop-Breaker Latches	677
Tracing Forward Through Loop-Breaker Latches	677
Calculation of the Worst and Total Negative Slack	678
Normalized Slack Analysis	679
Finding Recovered Paths	679
18. Constraint Consistency	681
Constraint Consistency Overview	681
Constraints in the Design Flow	682
Debugging Features	683
Analysis and Debugging Methodology	684
Starting a Constraint Consistency Session	686
Starting a Session	687
License	687
Setup Files	688
Command Log File	688
Supported Constraints	688
Reporting Constraint Acceptance	690
Exception Order of Precedence	690
Exception Type Priority	691
Path Specification Priority	691
Analyzing a Violation	692
Suppressing Violations	696
Overview of Violation Suppression Flow	698
Disabling Rules	699

Contents

Waiving Specific Violations of a Rule	700
create_waiver Command	702
Creating Instance-Level Waivers	705
Usage Guidelines	708
Modifying Waivers	708
Reporting Waivers	708
Removing Waivers	709
Writing Waivers to a File	710
report_constraint_analysis Command	711
Using Attributes	714
Overview	714
Setting, Listing, and Reporting Attributes	714
Attribute Groups	714
Using Arcs to Generate Custom Reports	716
Creating a Collection of Library Timing Arcs	716
Customizing Rules and Reports	717
Enabling and Disabling Rules	717
Modifying Rules	718
Using and Creating Rule Sets	718
Using Tcl Scripts to Report Violation Data	719
Opening the SDC Browser From Custom Rules	720
Creating and Using User-Defined Rules	721
Creating a Tcl Rule-Checking Procedure	721
Registering a User-Defined Rule	723
Using User-Defined Rules	724
Viewing Violations of User-Defined Rules	724
User-Defined Rule Example	724
Rule-Related Commands	728
Reading Designs With Incomplete or Mismatched Netlists	728
Design Consistency Checking	731
Rules and Violations	731
Built-In Rules	732
Hierarchical Consistency Checking	733
Comparing Block- and Top-Level Constraints	734
Methodology	735
Checking Block and Top Constraints in the GUI	736
Constraint Comparison Example	738
Checking Block Versus Top Consistency	739
Checking Multiple Scenarios	741
Checking Multiple Instances of One Block	741
Creating Waivers	742
Creating Text Reports	742

Contents

Built-In Hierarchical Consistency Checking Rules	742
Correlation Consistency Checking	743
Comparing Two Sets of Design Constraints	743
Constraint Set Comparison Overview	744
Methodology	745
Correlation Consistency Violation Browser	746
Rules and Violations When Comparing Constraint Sets	748
Creating Waivers	748
Text Reports	748
Comparing Two Designs With Two Sets of Design Constraints	749
Methodology	750
Linking the Designs and Loading the Constraints Sets	751
Object Mapping With the Name Map File	751
Creating a Custom Clock Mapping File	752
Example Script	753
Additional Analysis Features	753
analyze_paths Command	754
The analyze_paths Default Output Report	755
The analyze_paths Detailed Output Report	755
Summary Output Report for the analyze_paths Command	757
Disabled Paths Output Report for the analyze_paths Command	758
analyze_unclocked_pins Command	759
The analyze_unclocked_pins Verbose Output Report	759
analyze_clock_networks Command	761
Output Reports for the analyze_clock_networks Command	761
report_case_details Command	763
report_clock_crossing Command	765
report_analysis_coverage Command	766
Analysis Coverage Overview	766
report_analysis_coverage in Single-Scenario Runs	767
Narrowing Down Reports	768
report_analysis_coverage in Multi-Scenario Runs	769
report_exceptions Command (Redundant Constraints)	770
Exception Reporting Overview	771
Using report_exceptions	772
Narrowing Down report_exceptions Reports	772
Reporting Ignored Exceptions	773
Reporting Dominant Exceptions	774
Reporting Dominant Exceptions for Ignored Exceptions	775
Graphical User Interface	775
Using the GUI	776

Contents

The Constraint Consistency Window	777
Window Types	778
Toolbars	780
Menu Commands	780
User Message Browser	781
Violation Browser	782
Violation Browser Overview	782
Guidelines for Navigating in the Violation Browser	784
Block-to-Top Violation Browser	785
Correlation Consistency Violation Browser	785
Waiver Configuration Dialog Box	785
Information Pane	787
Console	787
Online Help	787
SDC View	788
Hierarchy Browser and Schematic View	789
Path Schematic	791
Creating a Schematic	791
Select By Name Toolbar	792
Properties Dialog Box	794
Schematic Display Options	795
Collapse and Expand Objects	796
Display Pin and Port Group Attributes	797
Tutorial	801
Overview of Constraint Consistency	801
About the Tutorial Design	804
Starting Constraint Consistency	805
Analyzing the ChipLevel Design	806
Debugging Constraint Problems	810
Investigating the CAS_0003 Violation	810
Investigating the CLK_0003 Violation	816
Investigating the EXC_0004 Violation	822
Ending and Restarting the Tutorial Session	825
19. Reporting and Debugging Analysis Results	827
Global Timing Summary Report	827
Controlling report_global_timing Path Gathering	828
Controlling report_global_timing Report Output	830
Path Timing Report	834

Contents

Using the report_timing Command	834
Reporting Normalized Slack	837
Running Normalized Slack Analysis	837
Setting Limits for Normalized Slack Analysis	838
Using Normalized Slack to Adjust the Clock Period	838
Using the report_timing -exclude Option	839
Cover Design or Cover Through Report	840
Path Tagging	841
Enabling Path Tagging	842
Tagging a Path Collection	842
Excluding Tagged Paths	842
Reporting and Removing Tag Sets	842
Path-Based Analysis of Successively Worse Paths	843
Exhaustive Path-Based Analysis	843
Custom Timing Reports With Attributes	844
Quality of Results Report	845
Constraint Reports	847
Timing Constraints	847
Design Rule Constraints	848
Generating a Default Constraint Report	848
Reporting Violations	849
Maximum Skew Checks	850
No-Change Timing Checks	852
Minimum Pulse Width Report	853
Minimum Period Report	854
Bottleneck Report	855
Global Slack Report	858
Analysis Coverage Report	858
Clock Network Timing Report	860
Latency and Transition Time Reporting	861
Skew Reporting	862
Interclock Skew Reporting	863
Clock Timing Reporting Options	863
Summary Report	864
List Report	866
Verbose Path-Based Report	868
Limitations of Clock Network Reporting	869
Using the get_clock_network_objects Command	869
Clock-Gating and Recovery/Removal Checks	869

Contents

Timing Update Efficiency	870
Status Messages During a Timing Update	871
Path-Based Timing Analysis	872
Path-Based Analysis Modes	873
Exhaustive Path-Based Analysis With Machine Learning	876
Setting Recalculation Limits	876
Exhaustive Path-Based Analysis Settings	877
HyperTrace Accelerated Path-Based Analysis	878
Introduction to HyperTrace Technology	878
Configuring HyperTrace Analysis	881
Using HyperTrace Analysis	884
Unsupported Flows for HyperTrace Analysis	886
CCS Receiver Model for Path-Based Analysis	887
<hr/>	
20. Graphical User Interface	888
Opening and Closing the GUI	888
Opening the GUI	889
Closing the GUI	890
Quick Start Guide	890
GUI Windows	896
View Windows	897
View Settings Panel	898
Toolbars and Panels	899
Hierarchy Browser	899
Console	900
Object Selection	901
Selecting Timing Paths	901
Setting GUI Preferences	902
Analyzing Timing Path Collections	902
Loading Path Collections	905
Categorizing the Timing Paths	906
Saving Path Categories to a File	908
Loading Path Categories From a File	909
Marking Blocks and Applying Block Category Rules	909
Creating Custom Category Rules	911
Selecting Category Attributes for a Custom Category Rule	913
Defining a Filter Expression for a Custom Category Rule	914
Examining Clock Paths in an Abstract Clock Graph	916

Contents

Opening Abstract Clock Graph Views	918
Displaying and Hiding Clock Path Elements	919
Reversing and Reapplying Changes	921
Collapsing Selected Side Branches	921
Viewing Clock Latency Over Time	923
Displaying the Clock Trees Hierarchically	924
Analyzing Clock Domains and Clock-to-Clock Relationships	925
Querying Launch-Capture Clock Pairs	927
Selecting Clocks or Clock Domains	928
Sorting the Clock Tree View	928
Finding Clocks by Name	929
Filtering Clock Domains	930
Saving Clock Attribute or Clock Matrix Constraint Data	932
Clock Matrix Symbols and Colors	932
Examining Timing Paths and Design Logic in a Schematic	935
Examining Hierarchical Cells	937
Moving Down or Up the Design Hierarchy	938
Displaying or Hiding Buffers and Inverters	939
Displaying or Hiding Unconnected Macro Pins	941
Expanding or Collapsing Buses	942
Viewing and Modifying Schematics	944
Schematic Window	945
Selecting or Highlighting Objects By Name	946
Highlighting Schematics By Using Filtering Rules	948
Annotating Schematic Pins and Ports	950
Reversing and Reapplying Schematic Changes	951
Inspecting Timing Path Elements	951
Loading Paths Into a Path Inspector Window	955
Configuring the Path Inspector	955
Viewing Object Attributes	956
Viewing the Current Selection	957
Querying Objects	958
Viewing Object Properties	959
Managing Attribute Groups	961
Saving and Restoring Attribute Groups	963
Creating Custom Attribute Groups	964
Modifying Attribute Groups	964
Copying and Renaming Attribute Groups	965

Contents

Recalculating Timing Paths	966
Comparing Normal and Recalculated Paths	966
Comparing Normal and Recalculated Path Pins	967
Layout View and ECOs in the GUI	968
Interactive Multi-Scenario Analysis of Timing Paths	976
Interactive Multi-Scenario Analysis Flow	976
IMSA Attributes Saved and Restored	978
Using the Path Analyzer Window	979
Shifting the Slack for a Category	982
GUI Tcl Commands	984
<hr/>	
21. ECO Flow	986
ECO Fixing Overview	986
DRC, Crosstalk, and Cell Electromigration Violation Fixing	988
Crosstalk Delta Delay Fixing	989
Cell Electromigration Violation Fixing	990
Timing Violation Fixing	991
Power Recovery Fixing	992
Order of ECO Fixing Steps	994
Setting the ECO Options	995
Physically Aware ECO	999
Performing Physically Aware ECO	1001
Design Data Files	1002
Block-Level LEF Library Data	1004
DEF Files or IC Compiler II Database Files	1005
DEF to LEF Site Name Conversion	1005
Missing LEF Files for Hierarchical Blocks	1005
Physical Constraint File	1006
Advanced Spacing Labels and Rules	1007
Parasitic Data Files From StarRC	1008
Site-Aware Physical ECO Fixing	1008
ECO Fixing Methods	1011
Load Buffering and Load Shielding	1011
Side-Load Cell Sizing	1012
Clock Network ECO Fixing	1013
Fixing Timing Violations by Modifying Clock Networks	1014
Fixing DRC Violations in Clock Networks	1015

Contents

TNS-Driven Clock Network Timing ECO	1015
ECO Hold Fixing Using Load Capacitance Cells	1019
Power Recovery	1021
Power Recovery Based on Library Cell Names	1024
Power Recovery Based on a Library Cell String Attribute	1025
Power Recovery Based on a Library Cell Leakage Attribute	1025
Power Recovery Based on PrimePower Data	1026
Accelerated Power Recovery Using Machine Learning	1028
Excluding I/O Paths From Power Recovery	1033
ECOs With Multiply Instantiated Modules (MIMs)	1034
Reporting Unfixable Violations and Unusable Cells	1036
Reporting Unfixable Timing and DRC Violations	1036
Reporting Unusable Cells for Power Reduction	1038
HyperTrace ECO Fixing	1040
HyperTrace ECO Fixing With fix_eco_timing	1041
HyperTrace ECO Fixing With fix_eco_power	1042
Writing Change Lists	1043
Replaying an ECO Change List in PrimeTime	1043
Replaying Block-Level ECO Changes in a Top-Level Run	1044
Writing ECO Change Lists for Third-Party Place-and-Route Tools	1044
Implementing ECO Changes in Synopsys Layout Tools	1046
Incremental ECO Flow Using Synopsys Tools	1047
Initialize the Incremental ECO Flow	1050
Incremental ECO Iteration	1051
Hierarchical Incremental ECO Flow	1052
ECO Flow Using Reduced Resources	1055
Running Multiple Scenarios in the Same Session	1057
Running Multiple Saved Sessions	1057
Using Negative Timing Margins to Restrict Testing	1060
Reduced Resource Non-DMSA ECO Flow	1060
Freeze Silicon ECO Flow	1061
Preparing for the Freeze Silicon ECO Flow	1063
Programmable Spare Cells	1064
LEF/DEF Descriptions of PSCs	1065
Exporting DEF and Advanced Spacing Rules From IC Compiler II	1066
Freeze Silicon ECO in PrimeTime	1067
Running ECO Scenarios on Fewer Hosts	1068

Contents

Manual Netlist Editing	1069
“What-If” Incremental Analysis	1070
Automatic Uniquifying of Blocks	1071
Resolving Library Cells	1072
Estimating Delay Changes	1073
Sizing Cells	1075
Inserting and Removing Buffers	1076
Sizing Cells and Inserting Buffers in the GUI	1076
Manual Netlist Editing and <code>dont_use</code> Restrictions	1077
Swapping Cells	1077
Renaming Cells or Nets	1078
Library Cell Functional Equivalence	1079
Netlist Editing in Multiple Scenarios	1079
<hr/>	
22. Hierarchical Analysis	1081
Overview of Hierarchical Analysis	1081
HyperScale Analysis	1084
Key HyperScale Technology Features	1087
HyperScale Modeling	1087
HyperScale Block Context	1088
Multiply Instantiated Model (MIM) Analysis	1089
HyperScale Usage Flows	1090
Bottom-Up Analysis Using Block Models Without Context	1092
Top-Down Analysis Using Block Models and Context	1094
Choosing a Top-Down or Bottom-Up Flow	1097
Top-Down HyperScale Flow	1098
Bottom-Up HyperScale Flow	1101
HyperScale Usage Details	1105
HyperScale Configuration	1105
Reading and Linking the Design Data	1107
Block-Level Analysis	1108
Top-Level Analysis	1110
Multiply Instantiated Modules (MIMs)	1111
Hierarchical Constraint Extraction	1117
Performing Manual Clock Mapping	1121
Mismatches Between Top and Block Levels	1122
Block Context Margin	1123
Block Boundary Checking	1126
Clock Mapping Check	1134
HyperScale Session Data	1138

Contents

HyperScale ECO Closure Methods	1143
The PrimeTime ECO Flow With HyperScale	1144
Using Updated Block Contexts for Block-Level ECOs	1145
Context Update Supports Accurate Fixing of Internal Paths	1146
Fixing Block Boundary Paths	1147
ECOs for Multiply Instantiated Modules (MIMs)	1148
Resolving Block Violations in the ECO Flow	1149
Performing ECOs at the Top Level	1151
Running ECOs in Parallel for Runtime and Capacity	1152
HyperScale ECO Interface to IC Compiler and StarRC	1153
Timing Analysis in HyperScale	1157
HyperScale Annotations in Reporting	1157
Block Models	1158
Port Abstraction	1161
Side Inputs	1162
Stub Pins	1163
Transparent Latch Levels	1165
Block Boundary Modeling	1165
Clock Reconvergence Pessimism Removal (CRPR) Modeling	1165
Path-Specific Exception Modeling	1166
External Advanced On-Chip Variation (AOCV) Effects	1167
Clock Latency Modeling	1168
Model-Context Mismatch Handling	1168
Context Override	1169
Clock Mapping for Side-Input and Stub Pins	1170
Controlling the Block Boundary Adjustment	1170
HyperScale Attributes	1172
Attributes for cell objects	1172
Attributes for design objects	1174
Attributes for pin objects	1175
Attributes for port objects	1177
Context Characterization	1177
Characterizing the Context of a Block	1178
Example of Context Characterization	1180
Reporting the Characterized Context	1180
Exporting the Characterized Context	1181
Writing Physical Information	1181
Removing Context Information	1182
Using Context for Synthesis or Optimizing Constraints	1182
Context Characterization Exclusion of Lower-Level Blocks	1182
Mid-Level Context Generation	1183

Contents

Top-Level Context Generation	1183
Using Context for Block-Level Timing Analysis	1184
Context Handling of Cross-Boundary Timing Exceptions	1184
Writing and Reading Context in Binary Format	1187
Binary Context for Multiply Instantiated Modules	1189
Binary Context and Instance Grouping	1189
Reducing Clock Latency Pessimism During Context Merging	1190
Context Budgeting	1190
Overview of Context Budgeting	1191
How Contexts are Adjusted During Budgeting	1192
Performing Context Budgeting	1193
Reporting the Context Budget	1194
Customizing the Context Budget	1195
Applying Slack Margins	1196
Setting the Target Slack of a Block Segment	1197
Setting the Exact Delay of a Block Segment	1197
Setting the Clock Period Percentage of a Block Segment	1198
Setting a Minimum Block Segment Delay	1199
Precedence of Context Budget Specifications	1199
Using Context Budgeting in a HyperScale Flow	1200
Limitations	1201
Extracted Timing Model (ETM)	1201
Timing Model Extraction Overview	1203
Usage Flow of ETM	1205
Block-Level Analysis: Extraction and Validation	1205
Top-Level Analysis	1206
Preparing for Model Extraction	1206
Variables That Affect ETM Generation	1207
Removing Internal Input and Output Delays	1208
Controlling Mode Information in the Extracted Model	1208
Performing Model Extraction	1209
Loading and Linking the Design	1209
Preparing to Extract the Model	1209
Generating the Model	1210
Timing Model Extraction Details	1210
Boundary Nets and Internal Nets	1211
Paths From Inputs to Registers	1211
Paths From Inputs to Outputs	1211
Paths From Registers to Outputs	1212
Paths From Registers to Registers	1212
Clock Paths	1212

Contents

Transparent Latches and Time Borrowing	1212
Minimum Pulse Width and Minimum Period	1213
False Paths	1214
Clock-Gating Checks	1214
Back-Annotated Delays	1215
Noise Characteristics	1216
Other Extracted Information	1217
Model Extraction Capabilities	1218
Model Extraction Options	1220
Extracting Models With CCS Data	1221
Controlling the Extracted Table Sizes	1223
Extracting Clock Latency Arcs	1223
Extracting Constants and Case Values	1224
Extraction of UPF and PG Data	1224
Specifying Model Indexes	1230
Specifying Margins for Delay and Constraint Arcs	1231
Tapped Output Ports	1232
Timing Exceptions	1233
Generated Clocks	1234
Extracting Internal Pins	1235
Multiple Clocks on an Internal Pin	1237
Different Operating Corners	1238
Extracting Clock-Gating Checks	1238
Restricting the Types of Arcs Extracted	1241
Merging Extracted Models	1242
Model Merging Overview	1242
Merging Multiple Modes Into a Mode Group	1244
Merging Multiple Modes Into Multiple Mode Groups	1245
Merging Models Across Block-Internal Power Supplies	1245
Controlling the Model Merging Process	1249
Best Practices for ETM Generation	1249
Hierarchical Design Style Considerations	1249
Conservative Block Timing Analysis and Model Extraction	1250
Validation of ETM Models	1256
Model Validation Commands	1256
Automatic Model Validation	1257
Manual Model Validation Flow	1258
Troubleshooting Model Validation Mismatches	1259
Effects of Wrapper Design	1260
Quick Timing Model (QTM)	1262
Defining a Quick Timing Model	1263
Specifying Constraint and Delay Arcs as Path Types	1264

Contents

Script Example to Create a Quick Timing Model	1265
Instantiating a Quick Timing Model in a Design	1267
<hr/>	
23. Using PrimeTime With SPICE	1268
Simulation Link to Correlate PrimeTime and SPICE Results	1268
Summary of Simulation Link Commands	1269
Correlating Path-Based Uncoupled PrimeTime and SPICE Analysis	1269
Correlating Arc-Based Coupled PrimeTime SI and SPICE Analysis	1271
Correlating PrimeTime SI and SPICE Noise Analysis	1274
Distributing Simulations Across Multiple Machines	1274
Generating a SPICE Deck With the write_spice_deck Command	1275
Writing a SPICE Deck for a Timing Path	1276
Writing a SPICE Deck for a Timing Arc	1277
Library Sensitization in write_spice_deck	1280
Library Driver Waveform	1282
Additional Required Information for SPICE Simulation	1282
Example of write_spice_deck Output	1283
Limitations of Using write_spice_deck for SPICE Correlation	1284
<hr/>	
24. Using PrimeTime With Design Compiler	1286
Features Specific to PrimeTime	1286
Timing Analysis Differences From Design Compiler	1286
Sharing Design Compiler and PrimeTime Scripts	1288
Synopsys Design Constraints Format (SDC) Script Files	1289
Checking the Syntax of the SDC File	1289
Checking the Consistency of Units	1289
Path Groups in Design Compiler and PrimeTime	1289
<hr/>	
A. Deprecated Features	1291
HyperScale Distributed Analysis	1291
From Full-Chip Flat to HyperScale Distributed Analysis	1291
Flat-Like Top-Level Reporting Using Restored Sessions	1297
From Distributed Analysis to Block-Level Analysis	1298
MIM Merging in Distributed HyperScale Analysis	1299
HyperScale Attributes	1301
Legacy Path-Based SMVA	1301

Contents

Glossary	1308
-----------------------	------

About This User Guide

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

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

The PrimeTime Suite documentation consists of the following documents:

- *PrimeTime User Guide* (this user guide) – Timing analysis using the PrimeTime, PrimeTime SI, PrimeTime ADV, and PrimeTime ADVP tools.
- *PrimeECO User Guide* – ECO timing, power, and DRC fixing with signoff-quality timing.
- *PrimePower User Guide* – Static and dynamic full-chip power analysis.
- *PrimeShield User Guide* – Parametric timing robustness analysis, cell robustness analysis, and critical path fast Monte Carlo HSPICE simulation.

This preface includes the following sections:

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

New in This Release

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

Related Products, Publications, and Trademarks

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

<https://solvnetplus.synopsys.com>

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

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

Conventions

The following conventions are used in Synopsys documentation.

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

Convention	Description
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy .
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

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

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

<https://solvnetplus.synopsys.com>

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

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

Contacting Customer Support

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

1

Introduction to PrimeTime

The PrimeTime Suite performs full-chip, gate-level static timing analysis, 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. To learn more about the basics of the PrimeTime tool, see

- [PrimeTime Capabilities](#)
 - [PrimeTime Product Tiers and Licenses](#)
 - [PrimeTime Add-On Tools](#)
 - [Using PrimeTime in the Implementation Flow](#)
 - [Compatibility With Synopsys Implementation Tools](#)
 - [Overview of Static Timing Analysis](#)
-

PrimeTime Capabilities

In PrimeTime, you can perform many types of design checks and analysis:

- **Design Checks**
 - 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**
 - Multiple clocks and clock frequencies
 - False path and multicycle 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
- Efficient hierarchical analysis using HyperScale technology
- Comprehensive constraint consistency checking
- Engineering change order (ECO) analysis that corrects violations and optimizes power by inserting buffers, resizing cells, and swapping cells

PrimeTime Product Tiers and Licenses

The PrimeTime product is available in three tiers: PrimeTime Base, PrimeTime Elite, and PrimeTime Apex.

[Table 1](#) lists the licenses included in each product tier. Newer features use the PrimeTime-ELT and PrimeTime-APX licenses. Older features use the other licenses.

Table 1 Licenses Included in PrimeTime Product Packaging Tiers

PrimeTime Base	PrimeTime Elite	PrimeTime Apex
PrimeTime	PrimeTime	PrimeTime x 2
PrimeTime-SI	PrimeTime-SI	PrimeTime-SI x 2
	PrimeTime-ADV	PrimeTime-ADV x 2
	PrimeTime-ADV-PLUS	PrimeTime-ADV-PLUS x 2
	PrimeTime-ELT	PrimeTime-ELT x 2
		PrimeTime-APX

[Table 2](#) lists the features included in each PrimeTime product tier, along with the license that enables the feature.

Table 2 PrimeTime Product Tiers and Included Features

Feature (and license)	PrimeTime Base	PrimeTime Elite	PrimeTime Apex
Multicore processing	X (16 cores)	X (32 cores)	X (64 cores)

Table 2 *PrimeTime Product Tiers and Included Features (Continued)*

Feature (and license)	PrimeTime Base	PrimeTime Elite	PrimeTime Apex
Signal Integrity Analysis (PrimeTime-SI)	X	X	X
Static Noise Analysis (PrimeTime-SI)	X	X	X
Constraint Consistency (PrimeTime-SI)	X	X	X
Writing and Reading Context in Binary Format (PrimeTime-SI)	X	X	X
IR Drop Annotation (PrimeTime-SI)	X	X	X
Generating a SPICE Deck With the write_spice_deck Command (PrimeTime-SI)	X	X	X
Parametric On-Chip Variation (POCV) (PrimeTime-ADV)		X	X
Multi-Input Switching Analysis (PrimeTime-ADV)		X	X
Physically Aware ECO (PrimeTime-ADV)		X	X
Distributed Core-Based Licensing (PrimeTime-ADV)		X	X
Replaying an ECO Change List in PrimeTime (PrimeTime-ADV)		X	X
Exhaustive Path-Based Analysis With Machine Learning (PrimeTime-ADV-PLUS)		X	X
HyperTrace Accelerated Path-Based Analysis (PrimeTime-ADV-PLUS)		X	X
Simultaneous Multivoltage Analysis (SMVA) (PrimeTime-ADV-PLUS)		X	X
POCV With Moment-Based Modeling (PrimeTime-ADV-PLUS)		X	X
POCV With Via Variation (PrimeTime-ADV-PLUS)		X	X
Multi-Input Switching Analysis - lib_arc and advanced modes (PrimeTime-ADV-PLUS)		X	X
Accelerated Power Recovery Using Machine Learning (PrimeTime-ADV-PLUS)		X	X
Layout View and ECOs in the GUI (PrimeTime-ADV-PLUS)		X	X

Table 2 PrimeTime Product Tiers and Included Features (Continued)

Feature (and license)	PrimeTime Base	PrimeTime Elite	PrimeTime Apex
GPD Parasitic Explorer (PrimeTime-ADV-PLUS)	X	X	
Using Ansys RedHawk-SC With Timing Analysis (PrimeTime-ADV-PLUS)	X	X	
HyperGrid Distributed Analysis (PrimeTime-ADV-PLUS)	X	X	
Advanced Data Management (PrimeTime-ELT)	X	X	
Context Budgeting (PrimeTime-APX)			X

PrimeTime Add-On Tools

The PrimeTime Suite supports the following add-on features.

- **PrimeECO** – Supports engineering change order implementation and analysis:
 - The PrimeECO add-on tool combines the timing analysis, physical implementation, and parasitic extraction functions of PrimeTime, IC Compiler II, and StarRC into a single integrated tool. It performs multiple ECO fixing iterations without the need for pausing between iterations.

For more information, see the *PrimeECO User Guide*.
- **PrimePower** – Supports advanced power analysis:
 - The PrimePower add-on tool analyzes full-chip power dissipation of cell-based designs. It supports both vector-free and vector-based peak power and average power analysis.

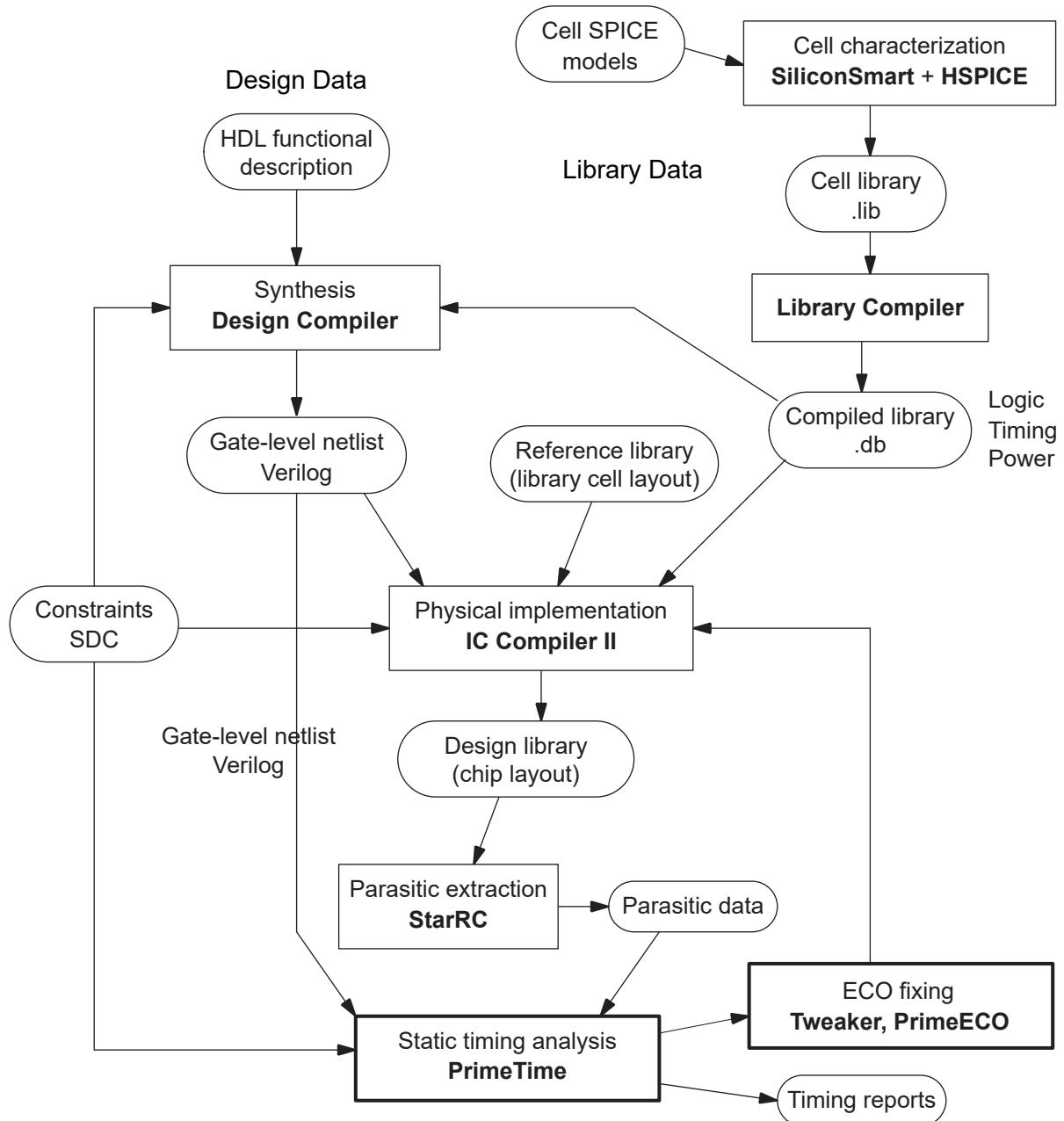
For more information, see the *PrimePower User Guide*.
- **PrimeShield** – Supports parametric robustness analysis:
 - PrimeShield provides design robustness analysis and fixing in the presence of increased process and voltage variability at advanced nodes, and enables designers to minimize pessimism, over-margining, and over-design to effectively reduce power and boost maximum operating frequency.

For more information, see the *PrimeShield User Guide*.

Using PrimeTime in the Implementation Flow

The PrimeTime tool works well with other Synopsys tools such as Design Compiler, IC Compiler II, Fusion Compiler, and StarRC. These tools share many of the same libraries, databases, and commands, as shown in the following figure.

Figure 1 Implementation Flow With PrimeTime Analysis



Starting from an RTL design, the Design Compiler tool generates a gate-level design. From the gate-level netlist and physical library information, the physical implementation tool (IC Compiler II) performs placement and routing. The StarRC parasitic extraction tool extracts the parasitic RC data from the chip layout database.

The PrimeTime tool reads the gate-level netlist and parasitic data, and verifies the design timing using information provided in the logic (.db) library. If the tool finds any violations, it can generate engineering change orders (ECOs) that guide the physical implementation tool to fix the violations and optimize power.

PrimeTime can also operate as a standalone static timing analyzer in other design flows.

Compatibility With Synopsys Implementation Tools

The PrimeTime static timing analysis tool is designed to work well with the Design Compiler synthesis tool and the IC Compiler II 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.
- They support the Synopsys Design Constraints (SDC) format for specifying design constraints, including the timing and area constraints.
- They share many commands (such as `create_clock`, `set_input_delay`, and `report_timing`) that are identical or similar in operation.
- They share the same delay calculation algorithms and produce similar delay results.
- They generate similar timing reports.
- PrimeTime can provide engineering change order (ECO) guidance to IC Compiler II by generating change lists that fix timing violations and optimize power.

Although the Design Compiler and IC Compiler II tools have their own built-in static timing analysis capabilities, the PrimeTime tool has better speed, accuracy, capacity, and flexibility for static timing analysis. The PrimeTime tool also provides 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. PrimeTime breaks a design down into 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 to dynamic

simulation, static timing analysis is much faster because it is not necessary to simulate the logical operation of the circuit. Static timing analysis 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.

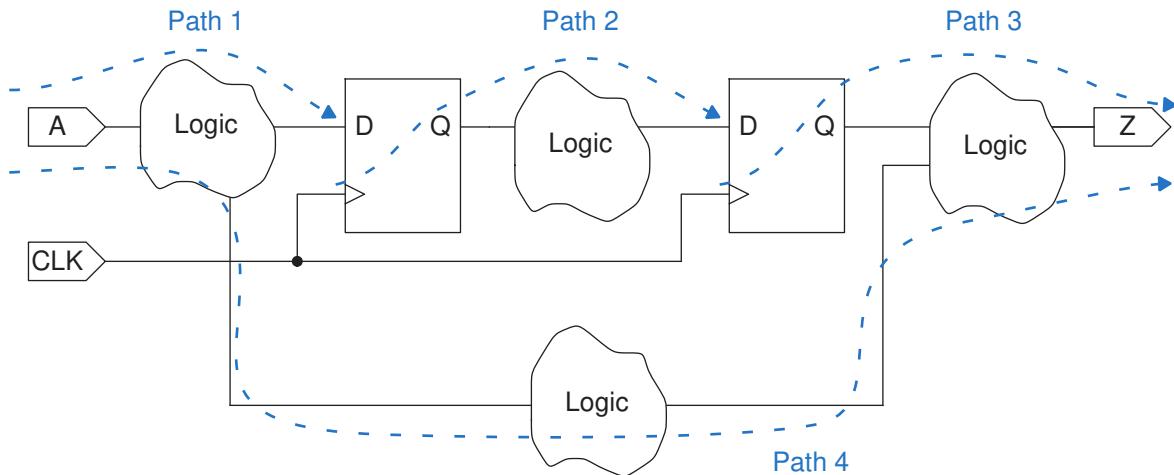
Timing Paths

When performing timing analysis, PrimeTime first breaks down the design into timing paths. Each timing path consists of the following elements:

- *Startpoint* – The start of a timing path where data is launched by a clock edge or where the data must be available at a specific time. Every startpoint must be either an input port or a register clock pin.
- *Combinational logic network* – 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.
- *Endpoint* – The end of a timing path where data is 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.

The following figure shows the timing paths in a simple design example.

Figure 2 Timing paths

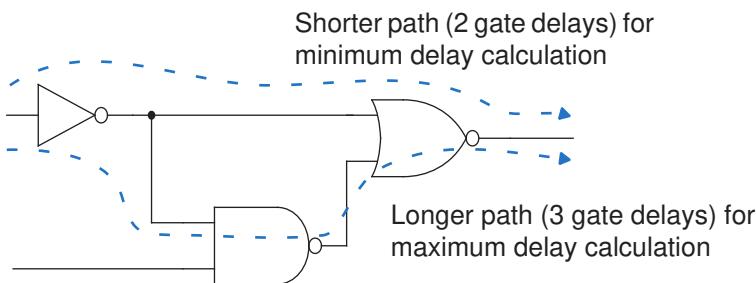


In the example, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point.

Path	Startpoint	Endpoint
Path 1	Input port	Data input of sequential element
Path 2	Clock pin of a sequential element	Data input of a sequential element
Path 3	Clock pin of a sequential element	Output port
Path 4	Input port	Output port

A combinational logic cloud might contain multiple paths, as shown in the following figure. PrimeTime uses the longest path to calculate a maximum delay and the shortest path to calculate a minimum delay.

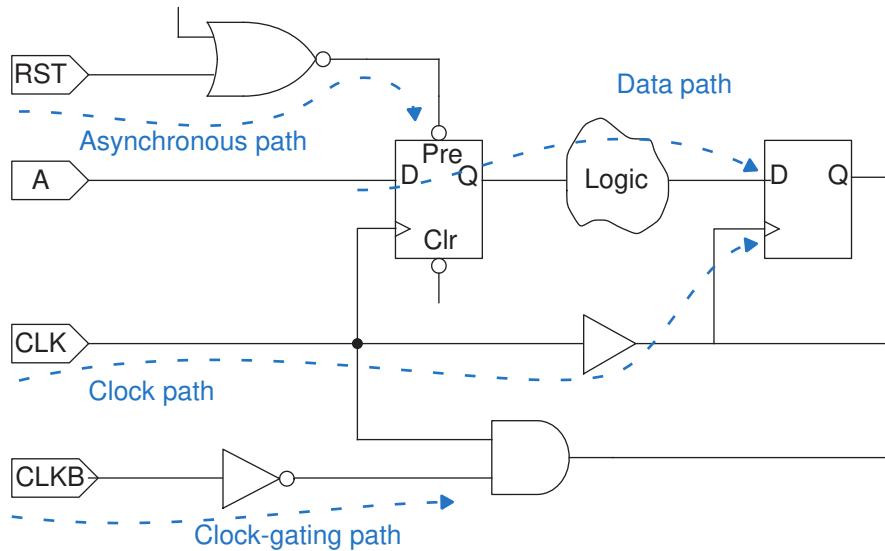
Figure 3 Multiple paths through combinational logic



PrimeTime also considers the following types of paths for timing analysis:

- 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 4 Types of paths considered for timing analysis



Delay Calculation

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

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, the tool 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. From these table entries, the tool 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.

The PrimeTime tool calculates net delays with the following methods:

- Estimating delays from a wire load model; this method is used before layout, when the chip topography is unknown
- Using specific time values back-annotated from a Standard Delay Format (SDF) file
- Using detailed parasitic resistance and capacitance data back-annotated from a Galaxy Parasitic Database (GPD), Standard Parasitic Exchange Format (SPEF), Detailed Standard Parasitic Format (DSPF), Reduced Standard Parasitic Format (RSPF) file

Constraint Checks

After PrimeTime determines the timing paths and calculates the path delays, it checks 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 edge.
- 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 edge.

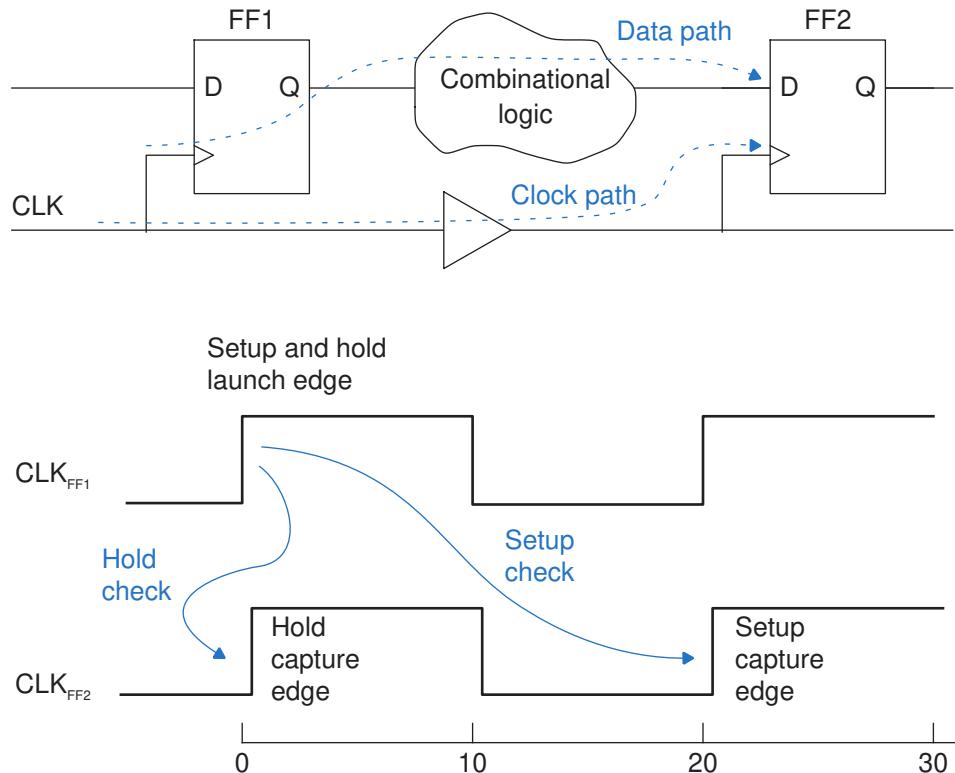
In addition to setup and hold constraints, PrimeTime can also check recovery and removal constraints, data-to-data constraints, clock-gating setup and hold constraints, and the minimum pulse width for clock signals.

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.

Example of Setup and Hold Checks for Flip-Flops

The following example shows how PrimeTime checks setup and hold constraints for a flip-flop.

Figure 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 the tool to be 10 time units. The time unit size, such as ns or ps, is specified in the logic library.

By default, the tool assumes that signals are propagated through each data path in one clock cycle. Therefore, when the tool performs a setup check, it verifies 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, the tool considers the longest possible delay along the data path and the shortest possible delay along the clock path between FF1 and FF2.

When the tool 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, the tool considers the shortest possible delay along the data path and the longest possible delay

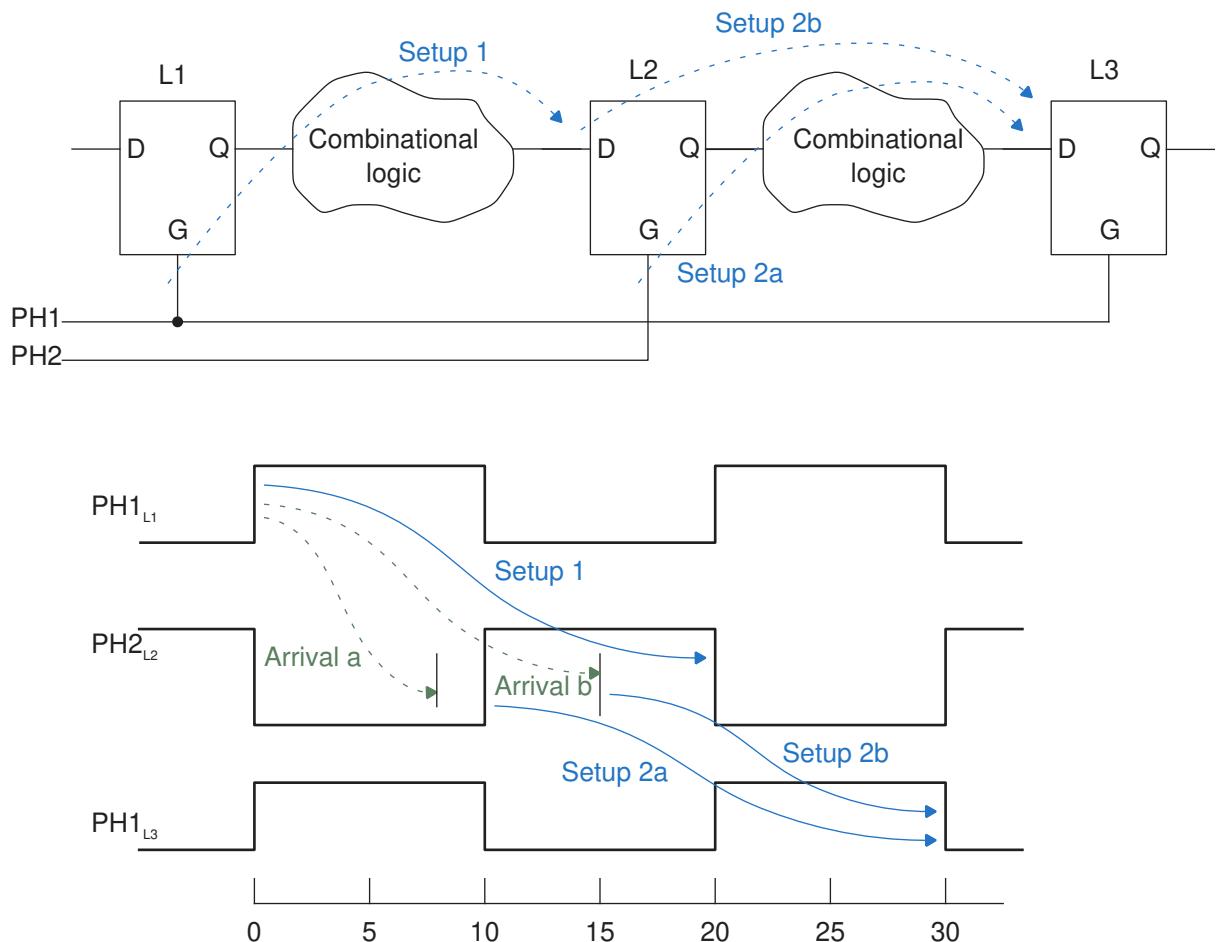
along the clock path between FF1 and FF2. A hold violation can occur if the clock path has a long delay.

Example of Setup and Hold Checks 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 reduce the constraints on successive paths.

For example, consider the two-phase, latch-based path shown in the following figure. All three latches are level-sensitive, with the gate active when the G input is high. The L1 and L3 latches are controlled by PH1, and the L2 latch is controlled by PH2. A rising edge launches data from the latch output, and a falling edge captures data at the latch input. In this example, consider the latch setup and delay times to be zero.

Figure 6 Latch-based paths



For the path from L1 to L2, the rising edge of PH1 launches the data. The data must arrive at L2 before the falling 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 rising 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.

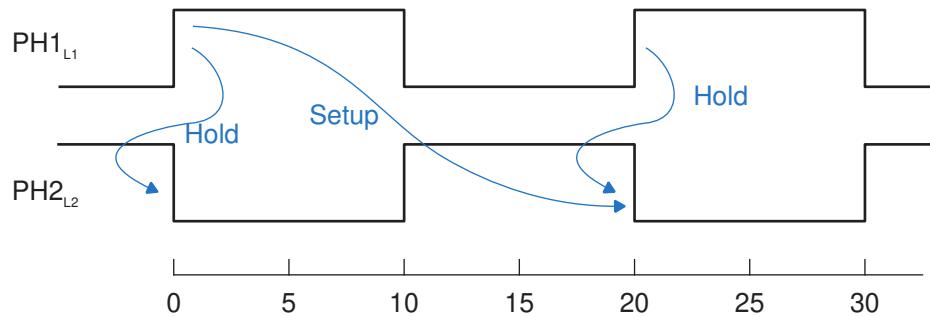
If the data arrives at L2 before the rising edge of PH2 at time = 10 (Arrival a), the data for the next path from L2 to L3 gets launched by the rising edge of PH2 at time = 10, just as a synchronous flip-flop operates. In this case, no time is borrowed from the second path. This timing requirement for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

If the data arrives after the rising 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 rising edge, but at the data arrival time at L2, at some time between the rising and falling 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 rising edge at time=10, or negative (a violation) if the data arrives after the falling edge at time=20.

To perform hold checking, PrimeTime considers the launch and capture edges relative to the setup check, as shown in the following figure. 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.

Figure 7 Hold checks in latch-based paths



Timing Exceptions

If certain paths are not intended to operate according to the default setup and hold behavior assumed by the PrimeTime tool, you need to specify those paths as timing

exceptions. Otherwise, the tool might incorrectly report those paths as having timing violations.

The PrimeTime tool lets you specify the following types of 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 explicitly specify as a time value.

For more information, see [Chapter 8, Timing Paths and Exceptions](#).

2

Getting Started

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

- [Installing the PrimeTime Software and Licenses](#)
 - [Configuring the PrimeTime Work Environment](#)
 - [Starting a PrimeTime Session](#)
 - [License Checkouts](#)
 - [Entering pt_shell Commands](#)
 - [Getting Help on the Command Line](#)
 - [The PrimeTime Static Timing Analysis Flow](#)
 - [Using Tcl/Tk in PrimeTime](#)
 - [Checking and Compiling Scripts With the TclPro Toolkit](#)
 - [Limiting System Messages](#)
 - [Saving and Reviewing System Messages](#)
 - [Ending a PrimeTime Session](#)
-

Installing the PrimeTime Software and Licenses

Before you can use the PrimeTime tool, you must install the software and licenses for your site. For information, see the following guides:

- *Synopsys Installation Guide* at <http://www.synopsys.com/install>
 - *Synopsys Licensing Quickstart Guide* at <http://www.synopsys.com/licensing>
-

Configuring the PrimeTime Work Environment

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

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

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

This system-wide setup file contains

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

Only the system administrator can modify this file.

2. Your home directory

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

3. The current working directory from which you start PrimeTime

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

To suppress the execution of all `.synopsys_pt.setup` files, use the `-no_init` option when you start the tool with the `pt_shell` command.

Starting a PrimeTime Session

You can start a PrimeTime session in either the command-line interface or the graphical user interface:

- For the command-line interface, enter the following command at the Linux shell prompt:

```
% pt_shell
```

```
PrimeTime (R)
```

```
Version M-2018.06 for linux64 - May 16, 2018
```

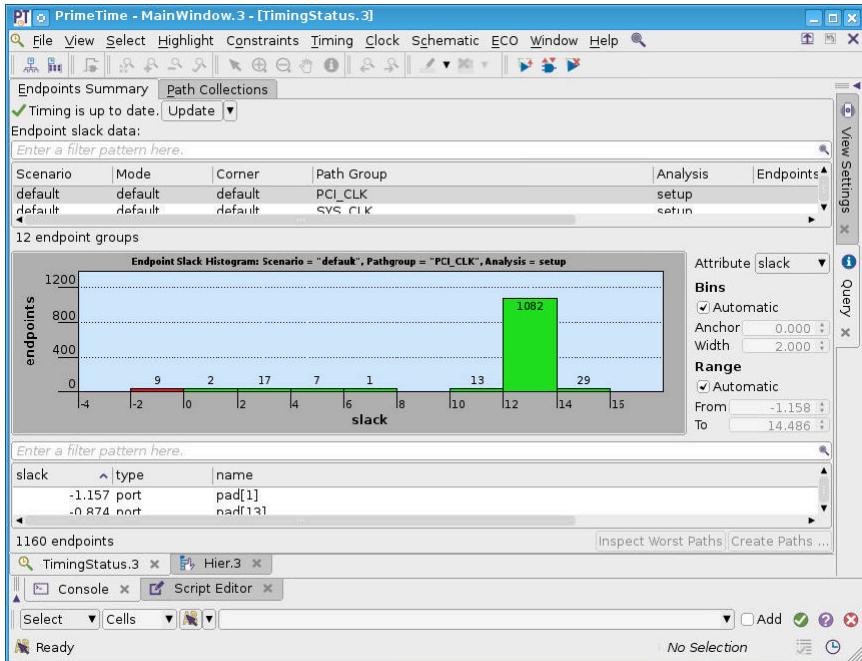
```
Copyright (c) 1988 - 201 Synopsys, Inc.  
This software and the associated documentation are proprietary ...
```

```
pt_shell>
```

- For the graphical user interface (GUI), enter this command:

```
% pt_shell -gui
```

Figure 8 PrimeTime GUI window



See Also

- [Graphical User Interface](#)

License Checkouts

When you start a PrimeTime session, the tool automatically checks out a PrimeTime license.

During analysis, the tool automatically checks out additional licenses as needed, depending on the tool features used. An information message indicates when this occurs:

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

To view the number of licenses currently checked out, use the `list_licenses` command:

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

Many additional features are available to control how licenses are used. For details, see [Controlling License Behaviors](#).

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 Linux 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

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

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> help -verbose command_name
```
- Man pages

To display the man page of a command, variable, or message, enter

```
pt_shell> man command_variable_or_message_name
```

To open the man page in an HTML Web browser, use the `-html` option of the `man` command.

The PrimeTime Static Timing Analysis Flow

To perform PrimeTime static timing analysis, follow the typical flow outlined in [Table 3](#).

Table 3 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_verilog</code> <code>link_design</code>	Working With Design Data
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>	Constraining the Design
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	Apply variation (optional)	<code>read_ocvm</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 3 Typical PrimeTime Static Timing Analysis Flow (Continued)

Step	Task	Typical commands	Related topics
9	Specify power information	load_upf create_power_domain create_supply_net create_supply_set create_supply_port connect_supply_net set_voltage	Multivoltage Design Flow
10	Specify options and data for signal integrity analysis	set_si_enable_analysis true read_parasitics -keep_capacitive_coupling	Signal Integrity Analysis
11	Apply options for specific design techniques	set_latch_loop_breaker set_multi_input_switching_coefficient define_scaling_lib_group ...	Advanced Analysis Techniques , Advanced Latch Analysis , Multi-Input Switching Analysis , Scaling for Multirail Level Shifter Cells , Fast Multidrive Delay Analysis , Parallel Driver Reduction
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 Constraints

Table 3 Typical PrimeTime Static Timing Analysis Flow (Continued)

Step	Task	Typical commands	Related topics
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	Generate engineering change orders (ECOs) to fix timing violations or 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

For scripts that you can use as a starting point to implement the PrimeTime static timing analysis flow, see the Synopsys Reference Methodology System at <https://solvnet.synopsys.com/rmgen>.

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 [SolvNetPlus](#).

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

```
pt_shell> printvar tcl_version
tcl_version = "8.6"
```

Tcl Packages and Autoload

The PrimeTime tool supports the standard Tcl package and autoload facilities. However, the load command is not supported, so packages that require shared libraries cannot be used. The tool 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 Synopsys installation root directory in the auxx/tcllib/lib/tcl8.6 directory.

To add a new Tcl package to PrimeTime, do one of the following actions:

- Install the package into the Synopsys tree.
- In the `.synopsys_pt.setup` setup file, add a new directory to the `auto_path` variable.

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
```

Support of the incr Tcl Extension

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

For information about incr Tcl, see the Tcl Developer Xchange website at <http://tcl.tk>

Checking and Compiling Scripts With the TclPro Toolkit

TclPro is an open-source toolkit for Tcl programming that supports the following:

- [Checking the Syntax in Scripts With the TclPro Checker](#)
- [Creating Bytecode-Compiled Scripts With the TclPro Compiler](#)
- [Debugging Scripts With the TclPro Debugger](#)

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

Installing TclPro Tools

Before using the TclPro tools, you need to install TclPro on your system. After you do this, 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. The PrimeTime tool uses this variable as a base for launching some of the TclPro tools. In addition, other Synopsys applications use this variable to link to the TclPro tools.

Checking the Syntax 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 the PrimeTime tool 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 the linux64 platform, you can find the script in the PrimeTime installation directory as linux64/syn/bin/ptprocheck.

To launch snps_checker from PrimeTime, you need to load a package provided with the installation 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]

```

The script loads the extensions shipped with your release. The following output example shows the loaded Synopsys extensions. Each line in the output shows where a syntax or semantic error was detected. The report indicates invalid syntax with a caret (^) below the character that begins the invalid syntax.

Example 1 snps_checker Output

```

% /synopsys/linux64/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
^
```

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. For example, some cases where one option requires another option are not checked.
- It cannot check extremely large scripts.
- 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.

Creating 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

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.

Debugging Scripts With the TclPro Debugger

The TclPro debugger (prodebug) 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, 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:

```
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 the PrimeTime tool. 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.

Limiting System Messages

A condition that triggers a warning message can occur many times in a single operation, resulting in hundreds of repeated messages. To keep the size of the session log reasonable, the tool automatically limits the number of messages generated for each type of condition.

You can control the message limits by the following methods:

- To set a limit for a specific message type in the whole session, use the `set_message_info` command:

```
pt_shell> set_message_info -id PARA-020 -limit 200
1
```

- To specify a limit on messages generated by the `read_parasitics`, `read_sdf`, `report_annotated_parasitics -check`, and `update_timing` commands, set the `sh_message_limit` variable:

```
pt_shell> set_app_var sh_message_limit 50
50
pt_shell> printvar sh_limited_messages
sh_limited_messages = "DES-002 RC-002 RC-004 RC-005 RC-006 RC-009 ..."
```

This limit applies separately for each type of message and for each time a command is run (for example, `read_sdf`). The default is 100 messages of each type per command execution.

- To specify the general limit that applies to all message types, set the `sh_global_per_message_limit` variable. The default is 10,000 of each type in the PrimeTime session.

```
pt_shell> set_app_var sh_global_per_message_limit 800
800
```

The first method has priority over the other two methods. Between the limits specified by the second and third methods, the lower limit applies.

Saving and Reviewing System Messages

You can save system messages reported by the tool, review the messages at any time, and query the numeric and string values reported inside the messages. For example, the PTE-064 information message reports path-related clocks:

```
Information: Related clock set 0 includes clock 'SDRAM_CLK' with period
7.500. (PTE-064)
```

To save PTE-064 messages generated in the PrimeTime session, use the `-save_limit` option of the `set_message_info` command:

```
pt_shell> set_message_info -id PTE-064 -save_limit 100
```

This tells the tool to save the next 100 PTE-064 messages generated by the tool. The default `-save_limit` value is -1, which means that no messages are saved. Setting a value of 0 means no limit.

To query the message format and save limit, use the `get_message_info` command:

```
pt_shell> get_message_info -id PTE-064
id PTE-064 severity Information limit 10000 save_limit 100 occurrences
22 suppressed 0 message {Related clock set %d includes clock '%s' with
period %.3f.}
```

To create a collection of the saved messages and query the content of the messages, use the `get_message_instances` command and the `get_attribute` command:

```
pt_shell> set my_msgs [get_message_instances PTE-064]
...
```

The attributes of the `message_instance` object class let you query the arguments (values and strings reported in the message), full message body, and message ID:

```
pt_shell> get_attribute -class message_instance $my_msgs arguments
0,SDRAM_CLK,7.500 0,SD_DDR_CLK,7.500 1,SYS_CLK,8.000 ...
pt_shell> get_attribute -class message_instance $my_msgs message_body
{Related clock set 0 includes clock 'SDRAM_CLK' with period 7.500.}
{Related clock set 0 includes clock 'SD_DDR_CLK' with period 7.500.}
...
pt_shell> get_attribute -class message_instance $my_msgs message_id
PTE-064 PTE-064 PTE-064 PTE-064 PTE-064 PTE-064 ...
```

To retrieve the second attribute from each message in a collection, you can use a script similar to this:

```
foreach_in_collection x [get_message_instances PTE-064] {
    # To replay the message
    echo [get_attribute $x message_body]
    # To fetch arguments and process them
    # Get arguments string delimited by ','
    set y [get_attribute $x arguments]
    set z [split $y ',']
    # The second argument (with index 1) is the clock
    set my_clock [get_clock [lindex $z 1]]
    # This helps report the clock fetched from the second argument
    # in each case
    query_objects $my_clock
}
```

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](#)
- [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 by using 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.

For more information, see [Saving and Restoring Sessions](#).

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`!

In the PrimeTime GUI main window, the File > Exit menu command has the same effect; it shuts down the PrimeTime tool. To close the GUI window but keep the `pt_shell` window open, use the File > Close GUI command or click the X in the corner of the GUI window.

Command Log File

When you end a PrimeTime session, the tool saves the session history into a file called the command log file. This file contains all commands executed during the session and serves as a record of your work. Later, you can 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, 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

Licensing

The following topics describe how licenses are checked out and managed in the PrimeTime tool:

- [How Licenses Work](#)
 - [Controlling License Behaviors](#)
-

How Licenses Work

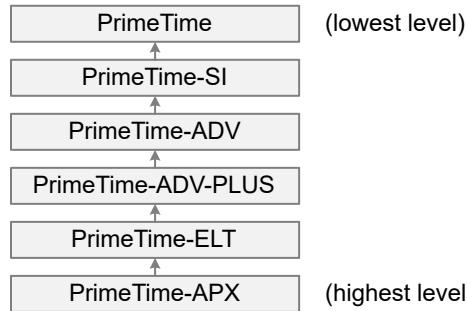
The following topics describe how licenses are checked out and how license requirements are computed in the PrimeTime tool:

- [License Chains](#)
 - [Local-Process Licensing](#)
 - [License Management in Distributed Analysis](#)
 - [Distributed Scenario-Based Licensing](#)
 - [Distributed Core-Based Licensing](#)
 - [SMVA Licensing](#)
-

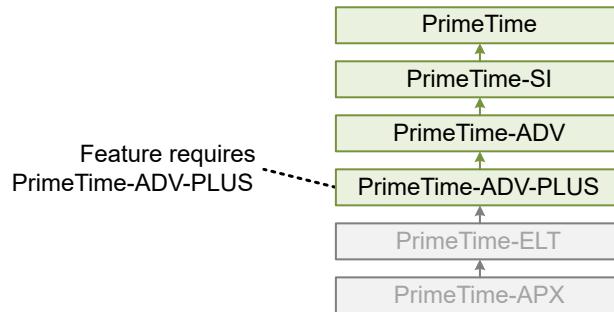
License Chains

A *license chain* is an ordered sequence of licenses. Each subsequent higher-level license is a superset of the lower-level licenses that precede it.

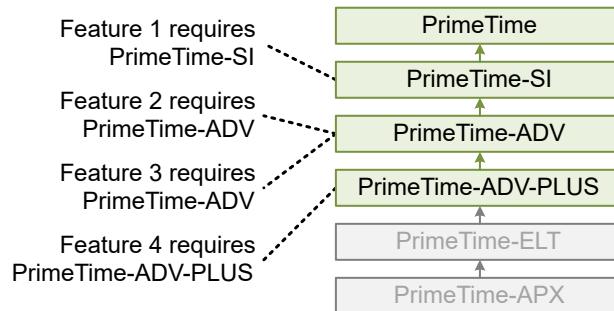
The full license chain for the PrimeTime Suite products is:



To check out a particular license, all preceding lower-level licenses must be checked out first (and in the same quantity). For example, if a feature requires PrimeTime-ADV-PLUS, the following licenses must be checked out:



When multiple features that require different licenses are used, the license chain is checked out up to the highest-level license required by a feature. If multiple features require the same license, that same license satisfies those features.



After a feature completes operation, its licenses remain checked out by the tool. This ensures that subsequent uses of the feature have the licenses needed to use it. For information on manually checking licenses in and out, see [Manually Controlling License Usage](#).

For details on what licenses are included in the different PrimeTime tiers, see [PrimeTime Product Tiers and Licenses](#).

Local-Process Licensing

In *local-process licensing*, the license requirements of a local (non-distributed) PrimeTime process are determined by the features used by that process. This is the default licensing behavior.

When multicore analysis is used:

```
pt_shell> set_host_options -max_cores N
```

then the following behaviors apply:

- Each purchased PrimeTime Base product enables (up to) 16 cores.
- Each purchased PrimeTime Elite product enables (up to) 32 cores.
- Each purchased PrimeTime Apex product enables (up to) 64 cores.

To implement these behaviors, the required license count is computed from N as follows:

- 1 to 16 cores

No additional license is required. No adjustment to license count is needed.

- 17 to 32 cores

A PrimeTime-ADV license (provided by the PrimeTime Elite product tier) is required.
No adjustment to license count is needed.

- 33 or more cores

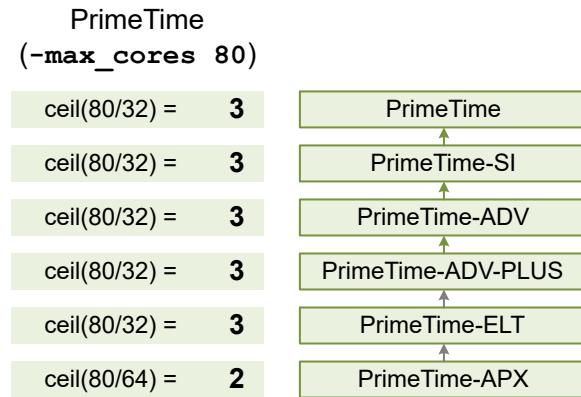
A PrimeTime-APX license (provided by the PrimeTime Apex product tier) is required.
The required license counts are computed using the following divisor rules (the ceil() function rounds up to the next nearest integer):

License	License count computed from N
PrimeTime	ceil($N/32$)
PrimeTime-SI	ceil($N/32$)
PrimeTime-ADV	ceil($N/32$)
PrimeTime-ADV-PLUS	ceil($N/32$)
PrimeTime-ELT	ceil($N/32$)
PrimeTime-APX	ceil($N/64$)

Note:

Although the divisors for the lower-level licenses under PrimeTime-APX are 32 instead of 64, each PrimeTime-APX license comes with two of each of these lower-level licenses to compensate. For details, see [PrimeTime Product Tiers and Licenses](#).

For example, the license requirements for a local PrimeTime analysis using `-max_cores 80` are computed as follows:



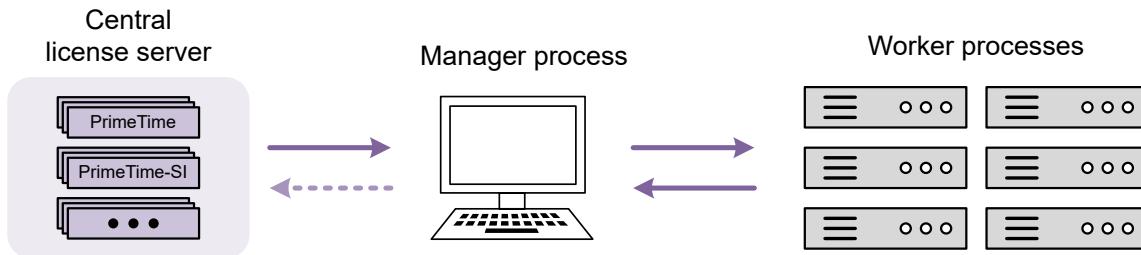
License Management in Distributed Analysis

Distributed analysis allows analysis work to be distributed across worker processes that are controlled by a manager process. PrimeTime provides two types of distributed analysis:

- [Distributed multi-scenario analysis \(DMSA\)](#)
- [HyperGrid distributed analysis](#)

In a distributed analysis, the manager process performs all license management. It acquires any licenses needed by the worker processes from the central license server, then dynamically distributes them to the worker processes.

Figure 9 Manager Process License Management in Distributed Analysis



The manager process shares its licenses with the worker processes. As a result, the manager process is effectively unlicensed.

By default, the manager keeps the licenses that it checks out from the central license server. This ensures that the worker processes have the licenses needed to perform continued work. However, the manager can return licenses to the central license server if:

- License autoreduction is enabled
- The license count is manually reduced

Distributed Multi-Scenario Analysis (DMSA)

In DMSA analysis, the manager process dynamically allocates licenses to worker processes.

A worker process uses a license only when actively executing a task. If you have more worker processes than licenses, some worker processes execute tasks (using licenses allocated to it by the manager) while others remain idle (waiting for available licenses). As worker processes complete their tasks, their licenses become available for other worker processes to use.

Each time the manager process gives a worker process a task to complete,

- If the manager has the required licenses available, it allocates them to the worker so it can begin the task.
- If not, the manager attempts to acquire the required licenses from the central license server. If successful, it allocates them to the worker so it can begin the task.
- If not, the worker remains idle until another worker completes its task and its licenses become available.

This dynamic license allocation can minimize expensive scenario image-swapping in license-limited situations.

HyperGrid Distributed Analysis

In HyperGrid distributed analysis, the manager process acquires the licenses required by the partition worker processes at the beginning of the analysis.

If the required licenses cannot be obtained, the manager process exits with an error message:

```
Error: Insufficient license resources available to analyze 8 partitions.  
(DA-020)
```

Distributed Scenario-Based Licensing

In *distributed scenario-based licensing*,

- Each scenario's license requirements are determined separately, as described in [Local-Process Licensing](#).
- The total license requirements are the sum of the scenario license requirements.

The scenario-based licensing mode applies only to distributed multi-scenario analysis (DMSA), and it is the default licensing behavior for it.

For example, the scenario-based license requirements for two DMSA scenarios, each using `-max_cores 80`, are computed as follows:

Scenario A (<code>-max_cores 80</code>)	$\text{ceil}(80/32) = 3$	+	$\text{ceil}(80/32) = 3$	=	6	PrimeTime
Scenario B (<code>-max_cores 80</code>)						
	$\text{ceil}(80/32) = 3$	+	$\text{ceil}(80/32) = 3$	=	6	PrimeTime-SI
	$\text{ceil}(80/32) = 3$	+	$\text{ceil}(80/32) = 3$	=	6	PrimeTime-ADV
	$\text{ceil}(80/32) = 3$	+	$\text{ceil}(80/32) = 3$	=	6	PrimeTime-ADV-PLUS
	$\text{ceil}(80/32) = 3$	+	$\text{ceil}(80/32) = 3$	=	6	PrimeTime-ELT
	$\text{ceil}(80/64) = 2$	+	$\text{ceil}(80/64) = 2$	=	4	PrimeTime-APX

Distributed Core-Based Licensing

In *distributed core-based licensing*,

- The total number of cores requiring each license are summed across the distributed scenarios or worker processes.
- For each license, the requirement is computed from the *total* number of cores using that license (N), using the following divisor rules (the `ceil()` function rounds up to the next nearest integer):

License	License count computed from N
PrimeTime	$\text{ceil}(N/32)$
PrimeTime-SI	$\text{ceil}(N/32)$
PrimeTime-ADV	$\text{ceil}(N/32)$
PrimeTime-ADV-PLUS	$\text{ceil}(N/32)$
PrimeTime-ELT	$\text{ceil}(N/32)$
PrimeTime-APX	$\text{ceil}(N/64)$

- The PrimeTime-ADV license is required to enable core-based licensing. If the PrimeTime-ADV license is not already required by other features used in the analysis, it is checked out with a count computed from N as above.

The core-based licensing mode is always used for HyperGrid distributed analysis.

The core-based licensing mode is optional for distributed multi-scenario analysis (DMSA). To enable it, set the following variable at the manager process:

```
pt_shell> set_app_var multi_scenario_license_mode core
```

The following example shows the core-based license requirements for two DMSA scenarios, each using `-max_cores 80`, are computed as follows:

Scenario A (<code>-max_cores 80</code>)	Scenario B (<code>-max_cores 80</code>)	
$\text{ceil}((80 + 80) /32)$	= 5	PrimeTime
$\text{ceil}((80 + 80) /32)$	= 5	PrimeTime-SI
$\text{ceil}((80 + 80) /32)$	= 5	PrimeTime-ADV
$\text{ceil}((80 + 80) /32)$	= 5	PrimeTime-ADV-PLUS
$\text{ceil}((80 + 80) /32)$	= 5	PrimeTime-ELT
$\text{ceil}((80 + 80) /64)$	= 3	PrimeTime-APX

SMVA Licensing

Most features in the PrimeTime tool are enabled by a single license, multiplied by the core count adjustment as needed.

However, [simultaneous multi-voltage analysis](#) (SMVA) requires the PrimeTime-ADV-PLUS license with a license count dependent on the analysis characteristics. The SMVA license requirement is computed as follows:

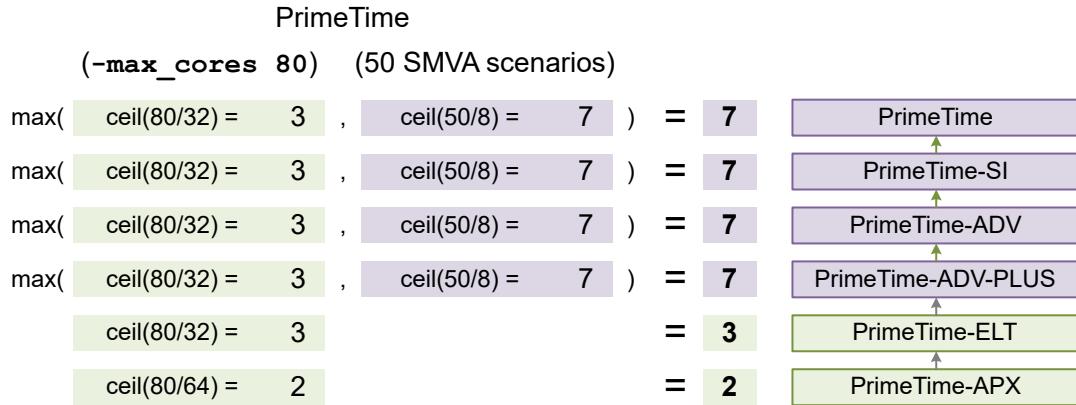
$$\text{PrimeTime-ADV-PLUS license count} = \text{ceiling}\left(\frac{\max\left(\begin{array}{l} \text{propagated DVFS scenario count,} \\ \text{supply group 1 voltage level count,} \\ \vdots \\ \text{supply group N voltage level count} \end{array}\right)}{8}\right)$$

where:

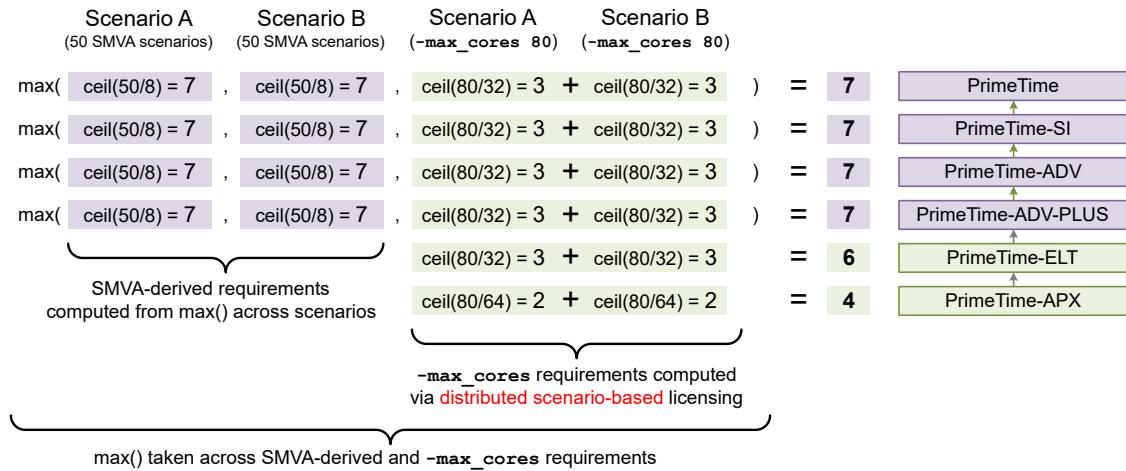
- “Propagated DVFS scenario count” is the number of DVFS scenarios actually found when propagating timing information across power domains. See [Propagated DVFS Scenarios](#) for details.
- “Supply group voltage level count” is the number of voltage levels defined in a supply group.

In other words, the license requirement is determined by the number of DVFS scenarios or the number of voltage levels in any supply group (whichever is higher), divided by 8, with fractional values always rounded up to the next integer value.

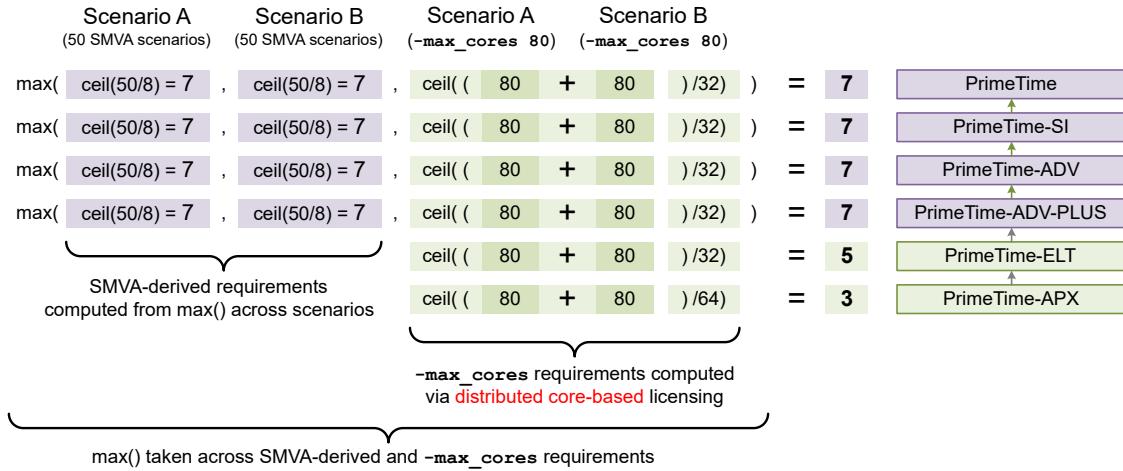
This SMVA-derived requirement applies to the PrimeTime-ADV-PLUS license (and thus also to the lower-level licenses in the chain). For each license, the final requirement is computed from the `max()` of the `-max_cores`-derived and SMVA-derived requirements:



In a DMSA run with SMVA, the overall SMVA-derived requirement is always computed from the `max()` of all scenario-specific SMVA requirements. This is true for both distributed scenario-based licensing:



and for distributed core-based licensing:



Controlling License Behaviors

The following topics describe how you can manage how the PrimeTime tool checks in and out licenses during analysis:

- [Manually Controlling License Usage](#)
- [Incremental License Handling](#)
- [License Pooling](#)
- [License Queuing](#)
- [License Autoreduction](#)

Manually Controlling License Usage

By default, the tool automatically checks out as many licenses as it needs and checks the licenses back in when you exit the tool. If there is a shortage of available licenses at your site, you can control license usage by using the commands in the following table.

To do this	Use this command
Check out one (or optionally more) licenses	<code>get_license [-quantity number] feature_list</code>
Show the licenses currently held by the tool	<code>list_licenses</code>

To do this	Use this command
Check in licenses back to the license server (optionally keeping a certain quantity)	<code>remove_license [-keep number] feature_list</code>
Limit the maximum checkout count for specific licenses	<code>set_license_limit -quantity number feature_list</code>
Remove the license limit for specific licenses	<code>remove_license_limit feature_list</code>
Report the license limit for specific licenses	<code>report_license_limit feature_list</code>

The following example manually checks out, reports, then checks in some licenses:

```
pt_shell> # check out four PrimeTime and PrimeTime SI licenses
pt_shell> get_license -quantity 4 {PrimeTime PrimeTime-SI}
1
pt_shell> # view the licenses currently checked out
pt_shell> list_licenses
Licenses in use:
    PrimeTime      (4)
    PrimeTime-SI   (4)
1
pt_shell> # check in all PrimeTime SI licenses except one
pt_shell> remove_license -keep 1 {PrimeTime-SI}
1
pt_shell> # verify that the PrimeTime SI licenses were returned
pt_shell> list_licenses
Licenses in use:
    PrimeTime      (4)
    PrimeTime-SI   (1)
1
```

Incremental License Handling

Incremental licensing handling allows the number of licenses of a feature to be changed during analysis. This feature is always enabled.

Licenses can be incrementally checked in and out as follows:

- You can increase or decrease the number of checked out licenses for a feature by using the `get_license` or `remove_license` command, respectively.
- You can use the `set_license_limit` command to change the maximum number of licenses checked out. If you reduce the limit, PrimeTime checks in enough licenses of the feature to meet the new target limit.

License Pooling

License pooling allows you to check out licenses of different features from different license servers. This feature is always enabled.

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

License Queuing

License queueing allows you to wait for available licenses if all licenses are currently in use. It is disabled by default. To enable it, set the `SNPSLMD_QUEUE` environment variable to true in the user environment before the tool is invoked. When you start the tool, you get this 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, also set the following environment variables:

- `SNPS_MAX_WAITTIME` – This variable specifies the maximum time that a process waits to acquire the initial license to start the PrimeTime session. 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 feature licenses within the same `pt_shell` process, after you have successfully checked out the initial 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)

Note:

`SNPS_MAX_QUEUEETIME` is not recommended in a distributed multi-scenario analysis (DMSA) flow. The DMSA manager already performs dynamic license acquisition in license-limited situations. If this variable is also set, each unsuccessful attempt by the DMSA manager to obtain a new license would block for the timeout period, thus preventing worker processes from starting new jobs until the timeout completes.

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)
```

License Autoreduction

License autoreduction is a distributed multi-scenario analysis (DMSA) feature that allows idle scenario worker processes to temporarily return their licenses to the central server.

The feature is disabled by default. You can enable or disable license autoreduction at any time from within the DMSA manager 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 for the following reasons:

- Although the feature 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 manager 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.
- 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}
set enable_license_auto_reduction false

# generate merged reports
report_timing ...
report_constraint ...
```

4

Managing Performance and Capacity

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

- [High Capacity Mode Options](#)
 - [Advanced Data Management](#)
 - [Threaded Multicore Analysis](#)
 - [Distributed Multi-Scenario Analysis](#)
 - [HyperGrid Distributed Analysis](#)
 - [Reporting the CPU Time and Memory Usage](#)
 - [Flow Summary Reports](#)
 - [Profiling the Performance of Tcl Scripts](#)
-

High Capacity Mode Options

By default, PrimeTime runs in high capacity mode, which reduces the peak memory footprint and makes runtime trade-offs between performance and capacity, without affecting the final results. 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.

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 should be as large as the physical RAM. However, actual disk usage is dependent on the design, constraints, and tool settings.

To specify the tradeoff between capacity and performance in high capacity mode, set the `sh_high_capacity_effort` variable to `low`, `medium`, `high`, or `default` (where `default` is equivalent to `medium`). A higher effort setting results in lower peak memory usage but longer runtime. You must set this variable before using the `set_program_options` command.

To disable high capacity mode, run the `set_program_options` command with the `-disable_high_capacity` option. The `sh_high_capacity_enabled` read-only variable shows whether high capacity mode is enabled:

```
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"
```

Advanced Data Management

The PrimeTime tool provides a more efficient method of storing design data, called *advanced data management*, that reduces in-memory consumption and allows larger designs to be analyzed.

This feature is recommended for designs with over 100 million instances, particularly those using parametric on-chip variation (POCV) analysis with moment-based modeling. There is a small increase in runtime and disk space usage when this feature is enabled.

To use this feature, set the following variable before any design data is loaded:

```
pt_shell> set_app_var timing_enable_advanced_data_management true
```

As a convenience, when you set this variable to `true`, the tool also sets the `sh_high_capacity_effort` variable to `high`. (The default high capacity effort level is `medium`).

This feature requires a PrimeTime-ELT license.

Threaded Multicore Analysis

PrimeTime can perform threaded multicore analysis, which improves performance on a shared memory server by running multiple threads on the available cores. The tool executes multiple tasks in parallel to yield faster turnaround for the following commands:

```
get_timing_paths
read_parasitics
read_verilog
report_analysis_coverage
report_attribute
report_constraint
report_timing
save_session
update_timing
update_noise
write_sdf
```

To learn how to use threaded multicore analysis, see

- Configuring Threaded Multicore Analysis
 - Executing Commands in Parallel
 - Threaded Multicore Parasitics Reading
 - Threaded Multicore Path-Based Analysis
-

Configuring Threaded Multicore Analysis

By default, the tool uses *threaded multicore analysis* to improve performance. This feature allows multiple simultaneous execution threads to perform work in parallel.

By default, the maximum number of CPU cores used per process is four. The `-max_cores` option of the `set_host_options` command allows you to specify this limit. For example,

```
pt_shell> set_host_options -max_cores 32
Information: Checked out license 'PrimeTime-ADV' (PT-019)
```

Values greater than 16 require a PrimeTime-ADV license, and values greater than 32 require one or more PrimeTime-APX licenses. The tool retains the licenses even if the maximum number of cores is later reduced below the threshold. For details, see [Local-Process Licensing](#).

If the limit exceeds the number of CPU cores provided by the current machine, it is reduced to the number of available cores. (This adjustment considers only the presence of the cores; machine loading is not taken into account.)

Overthreading allows the number of simultaneous threads to temporarily exceed the maximum core count for short intervals. It is enabled by default. If overthreading is not permitted in your compute environment, disable it by setting the following variable:

```
pt_shell> set_app_var multi_core_allow_overthreading false
```

Disabling overthreading can result in reduced multicore performance.

Executing Commands in Parallel

You can use parallel command execution for Tcl procedures and for post-update reporting and analysis, but not for commands that update the design such as ECOs, timing updates, or noise updates.

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.

Parallel command execution is most effective when you launch the longest running command first.

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

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

The parallel_execute Command

To run a set of commands in parallel, use the `parallel_execute` command. For example:

```
parallel_execute {
    {save_session design_dir/my_session} /dev/null
    {report_timing -max 10000 -nworst 10} report_timing.log
    {report_constraints -all_violators} report_constraints.log
    {report_qor} report_qor.log
}
```

After all commands running in parallel are complete, the tool returns to the `pt_shell` prompt.

The `parallel_execute` command dynamically allocates cores to the commands as they run. This improves processing efficiency when a mix of command types is given. More specifically,

- When a mix of multicore and non-multicore commands is specified, processing cores are dynamically allocated to the commands that can use them.
- When a command completes, its processing cores can be reallocated to other commands that are still running.

You can include command blocks that contain multiple commands. This is useful for commands that have dependencies on each other, or to parallelize more complex code structures. For example,

```
parallel_execute {
    {
        set paths [get_timing_paths ...]
        foreach_in_collection path $paths {
            ...
        }
    } paths.txt
    ...
}
```

If you specify `set_host_options -max_cores 1`, the tool runs in single-core analysis mode and executes all commands within the `parallel_execute` command in series in the specified order.

The redirect -bg (Background) Command

To run a post-update command in the background, use the `redirect -bg` command:

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

The main `pt_shell` keeps running while the `redirect -bg` command queues the jobs and waits for available cores.

If you specify `set_host_options -max_cores 1`, the tool runs in single-core analysis mode; the `redirect` command disregards the `-bg` option and runs the commands in the foreground.

If you issue an `exit` command while there is at least one active background `redirect -bg` command, the tool blocks the `exit` request 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 in sequence by the manager process, as shown in the following syntax:

```
parallel_foreach_in_collection iterator_variable collections {
    body_script_executed_by_workers_in_parallel
    ...
    post_eval {
        script_executed_by_manager_in_sequence
        ...
    }
}
```

The following example shows how to use the `parallel_foreach_in_collection` command to find and report the endpoint with the most violating paths.

Figure 10 Example of parallel_for_each_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_for_each_in_collection endpoint $endpoints {
    set paths [get_timing_paths -to $endpoint -nw 1000
               -slack_lesser_than 0]
    set num_violators [sizeof_collection $paths]
}
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"

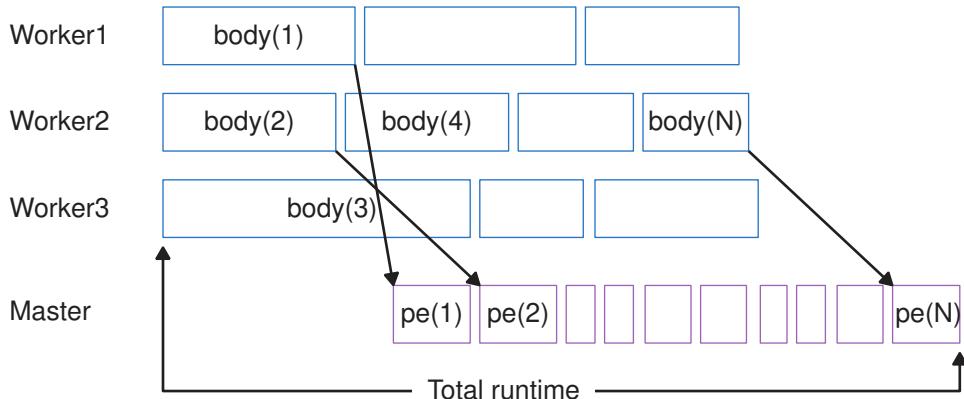
```

Body script, executed by workers in parallel

post_eval (PE) script, executed by the master in sequence

The following figure shows the `parallel_for_each_in_collection` command execution timing, where the processing time proceeds from left to right. The blue boxes represent iterations of the body script executed in parallel by worker processes.

Figure 11 Execution of parallel_for_each_in_collection



The `parallel_for_each_in_collection` command automatically schedules and balances the execution of N iterations to optimize the utilization of available cores.

While executing the body script, the worker processes submit the script specified by the `post_eval` command to the manager process (represented by the purple boxes in the preceding figure). Then each worker proceeds with its next loop iteration without waiting.

The manager 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, especially when dealing with large lists and arrays.

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

Threaded Multicore Parasitics Reading

By default, the `read_parasitics` command runs in a separate process launched by the tool, performed in parallel with other commands such as `read_sdc`. To get the maximum runtime benefit, use the `read_parasitics` command immediately after linking the design. You can disable parallel parasitic data reading by using the `set_host_options -max_cores 1` 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 the tool at the end of the session. By default, the tool writes the log of the parasitic commands to the `parasitics_command.log` file in the current working directory; to specify a different log file, set the `parasitics_log_file` variable.

Threaded Multicore Path-Based Analysis

PrimeTime runs threaded multicore analysis for path-based analysis when you run the `get_timing_paths` or `report_timing` commands with the `-pba_mode path` or `-pba_mode exhaustive` option. This feature is supported for all command variations of path-specific and exhaustive recalculation. For example:

- `get_timing_paths -pba_mode exhaustive ...`
- `get_timing_paths -pba_mode path ...`
- `get_timing_paths -pba_mode path $path_collection`
- `report_timing -pba_mode exhaustive ...`
- `report_timing -pba_mode path ...`
- `report_timing -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 and different operating modes. 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

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

The PrimeTime tool can analyze several scenarios in parallel with *distributed multi-scenario analysis* (DMSA). Instead of analyzing each scenario in sequence, DMSA uses a manager PrimeTime process that sets up, executes, and controls multiple worker processes—one for each scenario. You can distribute the processing of scenarios onto different hosts running in parallel, reducing the overall turnaround time. 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 modes for a given design and to distribute those analysis tasks onto different hosts.

To learn how to use DMSA, see

- [Definition of Terms](#)
- [Overview of the DMSA Flow](#)
- [Distributed Processing Setup](#)
- [DMSA Batch Mode Script Example](#)
- [Baseline Image Generation and Storage](#)
- [Host Resource Affinity](#)
- [Scenario Variables and Attributes](#)
- [Merged Reporting](#)
- [Loading Scenario Design Data Into the Manager](#)
- [Saving and Restoring Your Session](#)
- [License Resource Management](#)
- [Worker Process Fault Handling](#)
- [Messages and Log Files](#)

- [DMSA Variables and Commands](#)
 - [Limitations of DMSA](#)
-

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 worker process must switch to work on another scenario.

manager

Process that manages the distribution of scenario analysis processes.

scenario

Specific combination of operating conditions and operating modes.

session

Current set of scenarios selected for analysis.

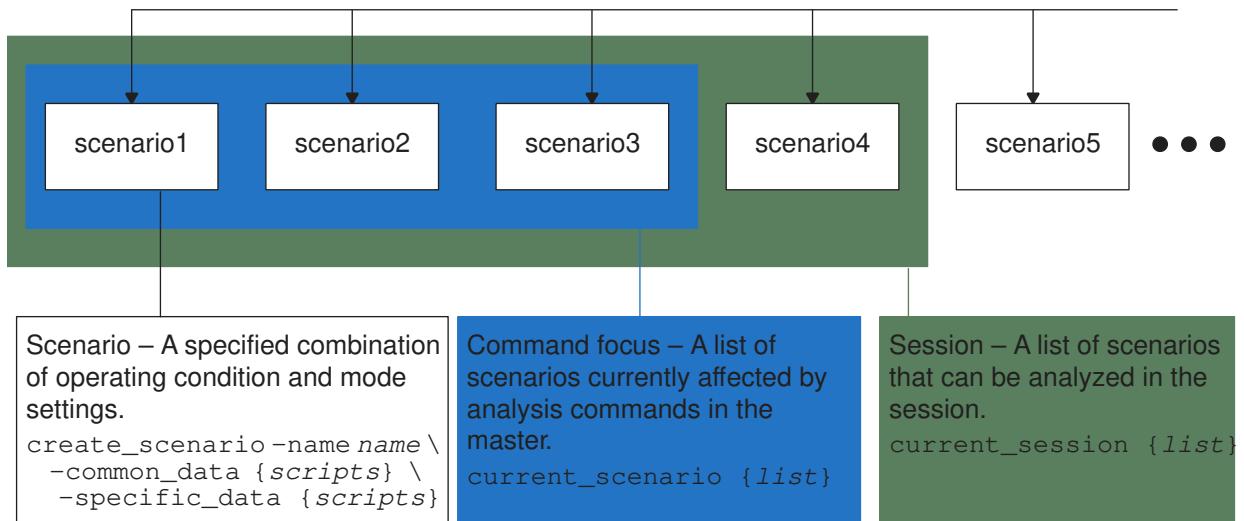
task

Self-contained piece of work defined by the manager for a worker to execute.

worker

Process started and controlled by the manager to perform timing analysis for one scenario; also called a *worker* process.

Figure 12 Relationships Between 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 manager). To create a scenario, use the `create_scenario` command, which 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 manager 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.

The **command focus** is the set of scenarios affected by PrimeTime analysis commands entered at the PrimeTime prompt in the manager process. The command focus can consist of all scenarios in the current session or just a subset specified by the `current_scenario` command. By default, all scenarios of the current session are in the command focus.

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

Setting the Search Path

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

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

The manager automatically sets the search path of the worker 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 worker context

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

.synopsys_pt.setup File

The manager and workers 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 manager launch directory at
`$sh_launch_dir/.synopsys_pt.setup`

To conditionally execute commands in certain scalar or distributed analysis modes only, check the value of the `sh_host_mode` variable to see whether it is set to `scalar`, `manager`, or `worker`. For example:

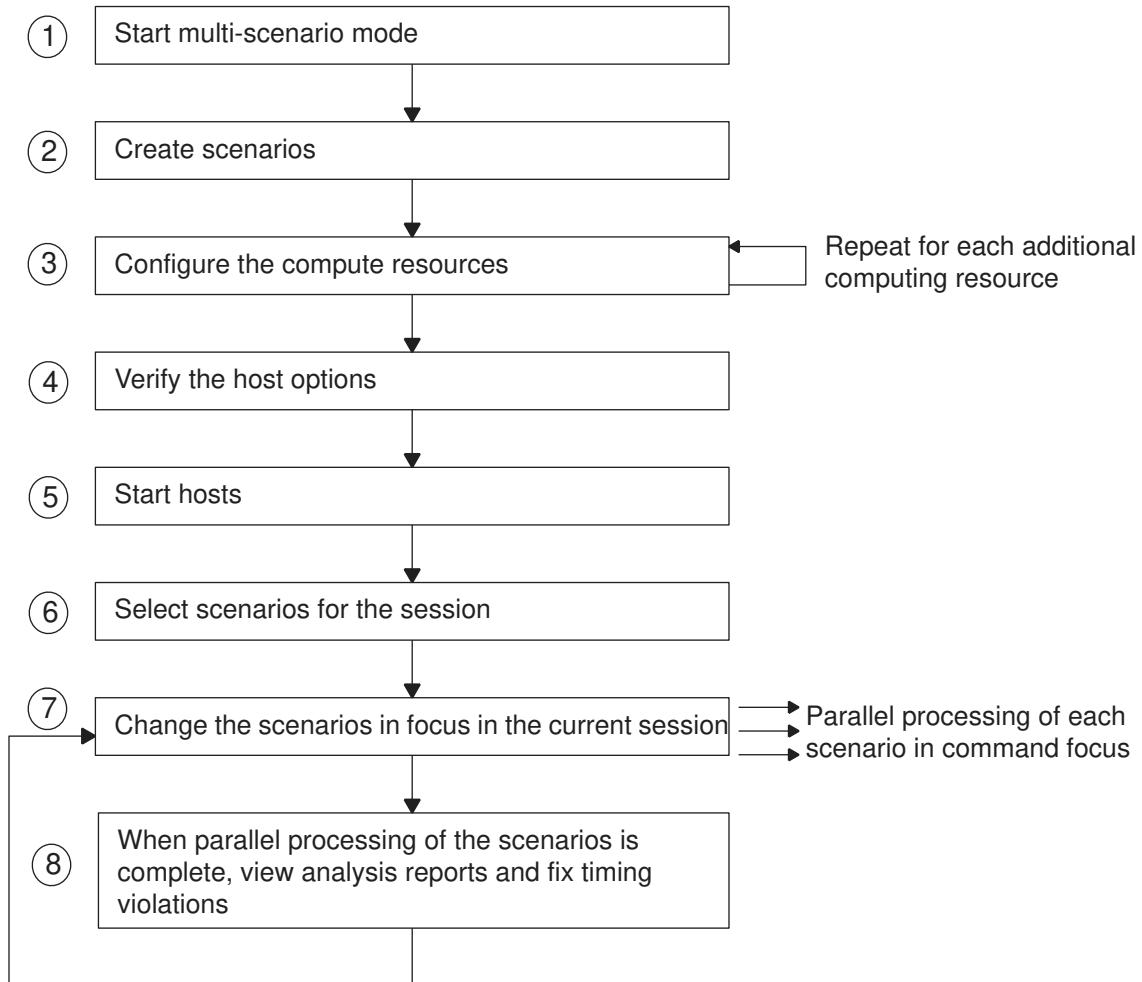
```
if { $sh_host_mode eq "manager" } {  
    set multi_scenario_working_directory "./work"  
}
```

DMSA Usage Flow

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

From the `pt_shell` prompt of the manager PrimeTime process, you can initiate and control up to 256 worker processes (see [Limitations of DMSA](#)).

Figure 13 Typical Multi-Scenario Analysis Flow



A typical multi-scenario analysis has the following steps:

1. Start PrimeTime in the multi-scenario analysis mode by running the `pt_shell` command with the `-multi_scenario` option. Alternatively, from a normal PrimeTime session, set the `multi_scenario_enable_analysis` variable to `true`.
2. 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.
3. Configure the compute resources that you want to use for the timing update and reporting by running the `set_host_options` command. This command does not start the host, but it 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
```

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

- Verify the host options by running the `report_host_usage` command. 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.

```
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
```

- Request compute resources and bring the hosts online by running the `start_hosts` command.

Note:

If you set the `multi_scenario_working_directory` and `multi_scenario_merged_error_log` variables, do so before you start the compute resources.

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
...
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

Note:

You must complete steps 1 through 5 before you can run the `current_session` command in the next step.

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.
 - a. Start processing the scenarios by executing the `remote_execute` command or performing a merged report command at the manager. For more information about merged reports, see [Merged Reporting](#).
 - b. When the processing of all scenarios by the worker 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 manager. The output of all merged reporting commands is displayed directly in the console at the manager.
 - c. 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 the PrimeTime tool 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 the Distributed Multi-Scenario Analysis Mode](#)
- [Managing Compute Resources](#)
- [Creating Scenarios](#)

- Specifying the Current Session and Command Focus
- Executing Commands Remotely

Starting the Distributed Multi-Scenario Analysis Mode

To start the PrimeTime tool in DMSA mode, use the `-multi_scenario` option when you start the PrimeTime shell:

```
% 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 manager PrimeTime process and multiple PrimeTime worker processes. The manager and worker processes interact with each other using full-duplex network communications.

The manager process generates analysis tasks and manages the distribution of those tasks to the worker 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 manager is therefore restricted to commands that perform manager functions.

To specify the directory in which to store working files for distributed processing, use the `multi_scenario_working_directory` variable.

To specify the file in which to write all error, warning, and information messages issued by the workers, set the `multi_scenario_merged_error_log` variable. Any message issued by more than one worker is merged into a single entry in the merged error log. For details, see [Merged Error Log](#).

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.

Upon startup, every PrimeTime process, both manager and worker, 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

In distributed analysis, the manager process uses one or more worker processes to perform the work. These worker processes can be configured, launched in a variety of compute environments, queried for their status, and terminated as needed.

The manager generates *tasks* for the worker processes to execute. Any one task can apply to only one scenario and any one worker process can be configured for only one scenario at a time. If there are more scenarios than worker processes to run them, some workers need to swap out one scenario for another so that all are executed.

It is important to avoid specifying more processes on a host than the number of available CPUs. Otherwise, much time is spent swapping worker processes in and out. For optimal

system performance of DMSA, match the number of worker processes, scenarios in command focus, and available CPUs so that every scenario is executed in its own worker process on its own CPU.

Setting Up Distributed Host Options

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

To specify worker processes, use the `set_host_options -num_processes` command:

```
pt_shell> set_host_options -num_processes process_count ...
```

The `-num_processes` option indicates that the specification applies to worker processes instead of the local process.

The `set_host_options` command provides additional options to configure the worker processes. For example, the `-protocol` option specifies the compute environment to use for the processes:

-protocol value	Type	Description
auto (default)	Automatic	Infers protocol from the <code>-submit_command</code> value
rsh	Unmanaged (direct login)	Remote shell login
ssh	Unmanaged (direct login)	Secure Shell login
lsf	Managed (compute farm)	Load Sharing Facility
netbatch	Managed (compute farm)	NetBatch
pbs	Managed (compute farm)	PBS Professional
rtda	Managed (compute farm)	RunTime Design Automation
sge	Managed (compute farm)	Grid Engine
slurm	Managed (compute farm)	Slurm Workload Manager
custom	Custom	Custom job submission command or script

Managed compute resources enables the tool to allocate pooled resources as they become available.

The `set_host_options` command provides a variety of additional options to configure the job submission command, define a name for the worker processes, and define load-

sharing characteristics of the worker processes. For details, including command examples, see the `set_host_options` man page.

You can use multiple `set_host_options -num_processes` commands to create multiple sets of processes to launch, each with its own host options, and optionally assign a name to each set by using the `-name` option.

The following commands configure four worker processes, two using SSH to a specific machine, and two more on an LSF compute farm:

```
set_host_options \
  -num_processes 2 \
  -name my_local \
  -max_cores 2 \
  -protocol ssh \
  workstation23.internal.mycompany.com

set_host_options \
  -num_processes 2 \
  -name my_farm \
  -max_cores 2 \
  -protocol lsf \
  -submit_command "[sh which bsub] -n 2 -R rusage" \
  -terminate_command "[sh which bkill]"
```

You can specify the maximum number of cores each worker process can use for threaded multicore analysis by specifying the `-max_cores` option along with the `-num_processes` option.

You can additionally configure the manager and worker processes by setting the following variables.

Variable	Description
<code>distributed_cleanup_variables_for_swap</code>	Controls cleaning up the state of variables when scenarios are swapped.
<code>distributed_custom_protocol_error_detecti on_timeout</code>	Specifies the maximum time that the distributed manager waits for custom protocol worker processes to come online.
<code>distributed_farm_check_pending_workers_in terval</code>	Specifies the maximum time that the distributed manager waits for a worker processes launched on a compute farm to come online.
<code>distributed_farm_protocol_error_detection _timeout</code>	Specifies the maximum time that the distributed manager waits for a worker processes launched on a compute farm to come online.
<code>distributed_heartbeat_period</code>	Specifies the interval at which the distributed workers send heartbeats to the manager process.

Variable	Description
distributed_heartbeat_timeout	Specifies the heartbeat timeout (in missed heartbeats). If the manager misses these many heartbeats (consecutively) from a worker, then the worker is assumed to have failed.
distributed_logging	Specifies the verbosity of logging during distributed analysis.
distributed_sh_protocol_error_detection_timeout	Specifies the maximum time that the distributed manager waits for sh, rsh, and ssh protocol worker processes to come online.
distributed_worker_exit_action	Controls the behavior of the exit command at distributed worker processes.

For details, see their man pages.

To remove host options and terminate their associated processes, use the `remove_host_options` command. For example, to remove and stop host1 and host3:

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

Configuring the Distributed Environment

To configure the distributed environment (remote launch script, worker startup script, and maximum levels of collection attributes), use the `set_distributed_parameters` command. Set the configuration before you run the `start_hosts` command.

Starting the Compute Resources

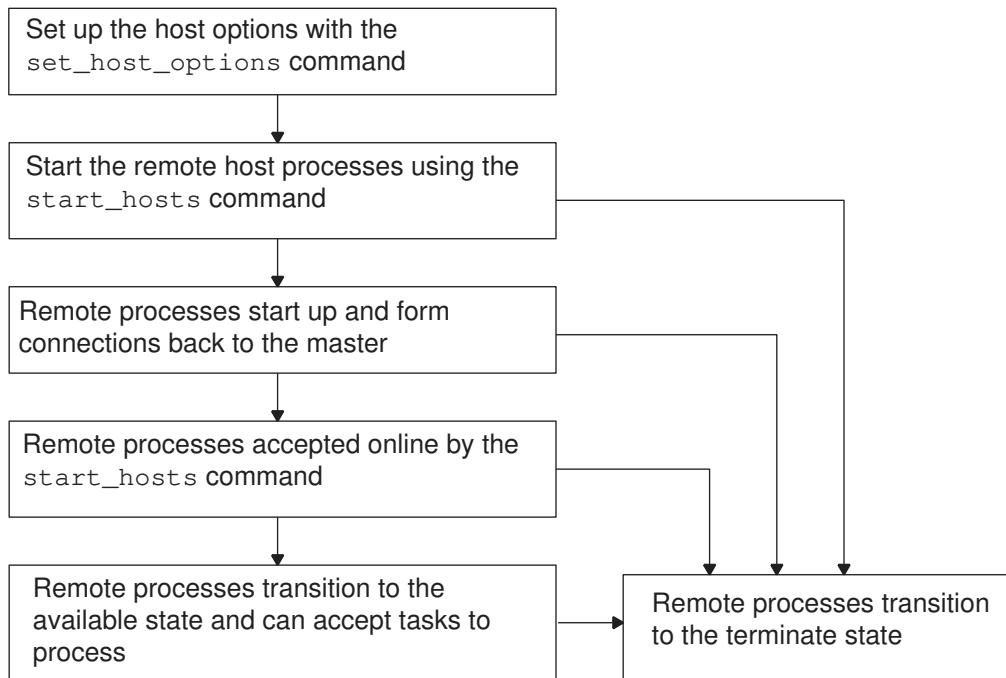
To start the hosts specified by the `set_host_options` command, use the `start_hosts` command, which requests compute resources and brings the hosts online. The `start_hosts` command continues to run until one of the following events occurs:

- All requested remote host processes become available
- The number of remote host processes that become available reaches the `-min_hosts` value specified in the `start_hosts` command (if specified)
- The number of seconds reaches the `-timeout` value specified in the `start_hosts` command (default 21600)

The number of available hosts directly affects the ability of the PrimeTime tool to simultaneously analyze scenarios. When there are too few hosts available, expensive image swapping occurs until enough hosts become available.

When you run the `start_hosts` command, any previously added hosts begin to transition through the states of their life cycle, as shown in the following diagram.

Figure 14 Life cycle states



DMSA Virtual Workers

When the number of scenarios exceeds the number of available hosts, at least two scenarios must be assigned to run on a host. If multiple commands are executed in those scenarios, the tool must perform save and restore operations to swap designs in and out of memory for executing each command, which can consume significant runtime and network resources.

In this situation, you can avoid the additional runtime and delay by setting a “load factor” in the `set_host_options` command:

```
pt_shell> set_host_options -load_factor 2 ...
```

The default load factor is 1, which disables the feature. Setting a value of 2 reduces save and restore operations at the cost of more memory.

The feature works by creating virtual workers in memory that can each handle one scenario. A setting of 2 doubles the number of workers by creating one virtual worker in memory for each real worker. If the real and virtual workers can accept all the scenarios at the same time, there is no need for save and restore operations.

The following `set_host_options` command examples demonstrate this feature. The `-submit_command` option shows the syntax for submitting jobs to an LSF farm with a specific memory allocation; use the appropriate syntax for your installation.

- **2 scenarios, multiple commands per scenario:**

```
set_host_options -num_processes 2 \
    -submit_command {bsub -n 16 -R "rusage[mem=16384]"}
```

The number of processes equals the number of scenarios, so there is no benefit from increasing the load factor.

- **4 scenarios, multiple commands per scenario:**

```
set_host_options -num_processes 2 -load_factor 2 \
    -submit_command {bsub -n 16 -R "rusage[mem=32768]"}
```

This command doubles the number of workers from 2 to 4 by creating a virtual worker for each real worker. It also doubles the memory allocation to accommodate the virtual workers. No save and restore operations are needed because the 4 workers can accept the 4 scenarios at the same time.

- **6 scenarios, multiple commands per scenario:**

```
set_host_options -num_processes 2 -load_factor 2 \
    -submit_command {bsub -n 16 -R "rusage[mem=32768]"}
```

This command doubles the number of workers from 2 to 4, which reduces the need for save and restore operations. However, it does not eliminate them completely because the 4 workers cannot accept all 6 scenarios at the same time.

To optimize the time benefit, make the total number of workers equal to the number of scenarios (possibly using multiple `set_host_options` commands) and allocate enough memory on the hosts for the virtual workers.

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:

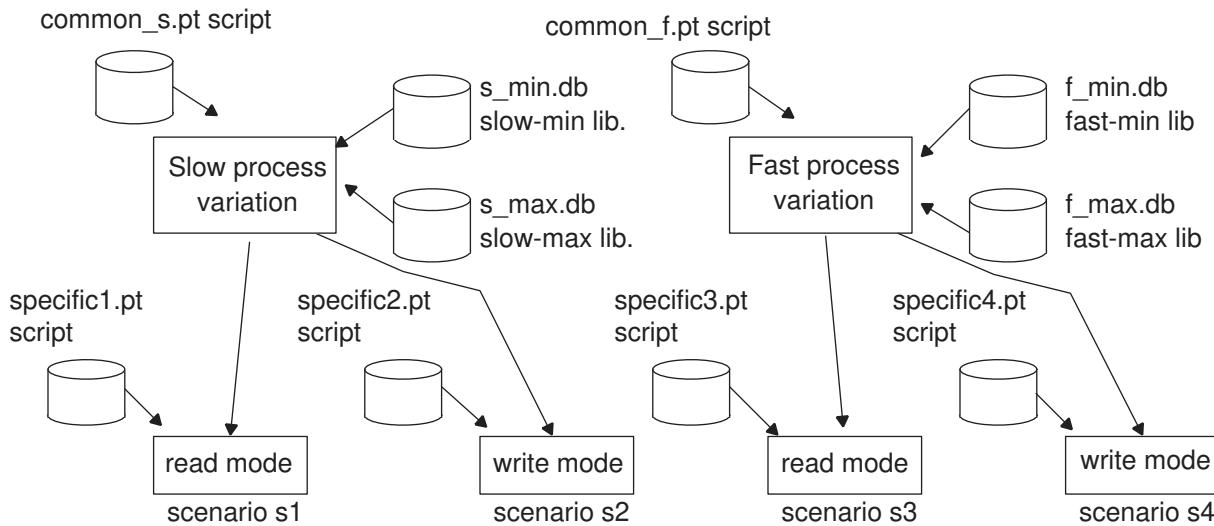
```
pt_shell> create_scenario -name scen1 \
    -common_data {common_s.pt} \
    -specific_data {specific1.pt}
```

There are two types of scripts associated with the scenario:

- Scripts that handle common data shared by multiple scenarios and included in the baseline image generation process (see [Baseline Image Generation and Storage](#))
- Scripts that handle specific data applied only to the specified scenario

The manager shares common data between groups of scenarios where appropriate, thereby reducing the total runtime for all scenarios. For example, consider the following figure.

Figure 15 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, the four scenarios to be analyzed are slow-read, slow-write, fast-read, and fast-write.

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.

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 manager, or a session saved from the current image generated during a DMSA session.

If you need to create voltage scaling groups, do so before you use the `link_design` command in one of the scripts used to set up a scenario.

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

To specify the set of scenarios analyze, use the `current_session` command:

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

The command lists the scenarios to be part of the current session. By default, all the listed scenarios are in *command focus*, the scenarios currently affected by analysis commands in the manager. To further narrow the command focus, use the `current_scenario` command:

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

This command is useful during interactive analysis. For example, you might want to modify the load on a particular net and then recheck the path timing in just that scenario or a subset of all scenarios in the current session.

You can restore the command focus to all scenarios in the session as follows:

```
pt_shell> current_scenario -all
```

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

Executing Commands Remotely

To explicitly run commands in the worker context, use the `remote_execute` command. Enclose the list of commands as a Tcl string. Separate the command with semicolons so that they execute one at a time.

To evaluate subexpressions and variables remotely, generate the command string using curly braces. For example:

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

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

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

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

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

The following example shows how to use the `remote_execute` command.

```

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 workers

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

```

If you use the `-pre_commands` option, the `remote_execute` command executes the specified commands before the remote execution command.

```

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

```

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

```

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

```

You can use the `remote_execute` command with the `-verbose` option to return all worker data to the manager 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` (worker only)
- Explicit `restore_session`
- `remove_design`

DMSA Batch Mode Script Example

In batch mode, PrimeTime commands are contained in a script invoked when the tool is started. For example,

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

The following script is a typical multi-scenario manager 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 s1 \
    -common_data common_data_scenarios_s1_s3.tcl \
    -specific_data specific_data_scenario_s1.tcl
create_scenario \
    -name s2 \
    -common_data common_data_scenarios_s2_s4.tcl \
    -specific_data specific_data_scenario_s2.tcl

create_scenario \
    -name s3 \
    -common_data common_data_scenarios_s1_s3.tcl \
    -specific_data specific_data_scenario_s3.tcl

create_scenario \
    -name s4 \
    -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 {s1 s2}

# cmd1,cmd2,cmd3, are placeholders for typical PrimeTime commands or
# scripts that need to be executed on s1 and s2
#
# 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 workers are inactivated and licenses are returned
```

A task is a self-contained block of work that a worker process executes to perform some function. A worker process starts execution of a task for the specified scenario as soon as it gets the licenses appropriate to that task.

This script performs the following functions:

1. The `initial_set` command sets the search path at the manager (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 have been set.
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 s1 and s2 for the current session, which is also the command focus by default. (You could 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 s1 and s2. 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 commands that alter the netlist or save or restore the session.

8. The `quit` command terminates the worker processes and manager processes in an orderly manner and releases the checked-out PrimeTime licenses.

The script writes out the report and log files in the following directory paths (assuming the manager `pt_shell` was launched from the directory `/mypath`).

- Manager command log:

`/mypath/pt_shell_command.log`

- Errors generated by the workers and returned to the manager and merged for reporting:

`/mypath/ms_session_1/merged_errors.log`

- Worker output log for work done for the session:

```
/mypath/ms_session_1/default_session/out.log
```

- Worker output logs generated for s1 and s2:

```
/mypath/ms_session_1/s1/out.log  
/mypath/ms_session_1/s2/out.log
```

- Worker command logs:

```
/mypath/ms_session_1/pt_shell_command_log/  
platinum1_pt_shell_command.log  
/mypath/ms_session_1/pt_shell_command_log/  
platinum2_pt_shell_command.log
```

Baseline Image Generation and Storage

The first task that the manager generates and delegates to a worker process is baseline image generation. The baseline image consists of a netlist representation and the common data files for that scenario. In many cases, the same image can be shared by multiple scenarios.

Before any other types of tasks can proceed, baseline image generation must be completed successfully. Failure of image generation also results in failure of the multi-scenario analysis for those scenarios that depend on the image.

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.

After baseline image generation, the manager generates and delegates to worker processes the execution of the user-specified commands in the scenarios that are in command focus.

When there are more scenarios than processes available to execute the scenario tasks, the worker process saves a current image for one scenario while it proceeds with task execution for a different scenario. The worker process saves the image in the scenario_name/current_image directory under the multi-scenario working directory.

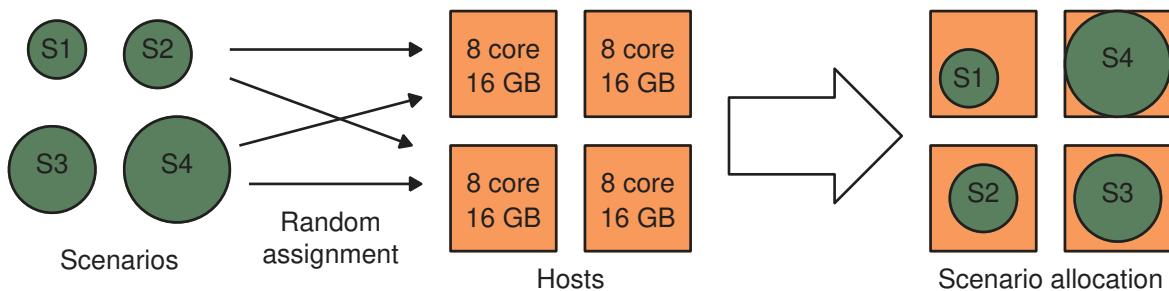
Host Resource Affinity

Automatic reconfiguring of worker processes to run different tasks in different scenarios can be an expensive operation. The multi-scenario process tries 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 worker processes added. A significant imbalance of scenarios to worker processes can degrade system performance.

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

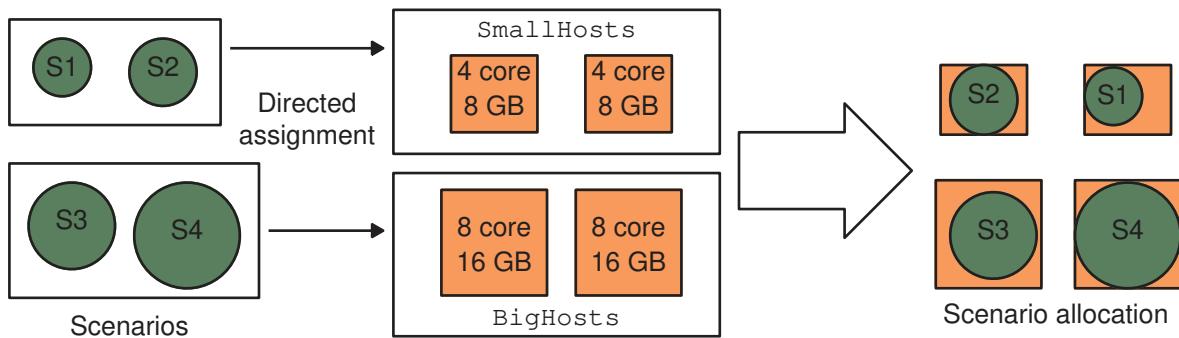
You can optionally assign scenarios an “affinity” for execution on a specified hosts, allowing more efficient allocation of limited computing resources. For example, suppose that you need to run four scenarios, S1 through S4, having different memory and core processing requirements. By default, you need to make four hosts available, each with enough CPU power and memory to accommodate the largest job, as shown in the following figure.

Figure 16 Random Scenario-to-Host Distribution (Default)



You can optionally assign smaller jobs to smaller hosts, and thereby use fewer resources while achieving the same turnaround time, as shown in the following figure.

Figure 17 Directed Scenario-to-Host Distribution Using Affinity



To specify the host resource affinity for scenarios, use commands similar to the following:

```
set_host_options -name SmallHosts -max_cores 8 -num_processes 2 \
    -submit_command {bsub -n 8 -R "rusage[mem=4000] span[ptile=1]"}

set_host_options -name BigHosts -max_cores 16 -num_processes 2 \
```

```
-submit_command {bsub -n 16 -R "rusage[mem=8000] span[ptile=1]"}
...
create_scenario -name S1 -affinity SmallHosts ...
create_scenario -name S2 -affinity SmallHosts ...
create_scenario -name S3 -affinity BigHosts ...
create_scenario -name S4 -affinity BigHosts ...
```

To report the scenario affinity settings, use the `report_multi_scenario_design -scenario` command.

Scenario Variables and Attributes

You can set variables in either the manager or any of its workers during DMSA, as described in the following sections:

- [Manager Context Variables](#)
- [Worker Context Variables and Expressions](#)
- [Setting Distributed Variables](#)
- [Getting Distributed Variable Values](#)
- [Merging Distributed Variable Values](#)
- [Synchronizing Object Attributes Across Scenarios](#)

Manager Context Variables

Here is an example of a manager context variable.

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

Notice how `x` is forced to be a string.

Worker Context Variables and Expressions

The manager has no knowledge of variables residing on the worker, but can pass variable or expression strings to the worker for remote evaluation. To pass a token to the worker for remote evaluation, use curly braces.

Suppose you have 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

To set variables at the worker level from the manager, use the `set_distributed_variables` command. The variable can be a Tcl variable, collection, array, or list.

To set variables in multiple scenarios from the manager, first create a Tcl array of the required values for those scenarios, then use the `set_distributed_variables` command to distribute the settings to the scenarios.

For example, suppose that you want to set variables `x` and `y` at the worker level to different values in two different scenarios. Create an array, indexed by scenario name, containing the values to set in the workers. From the manager, 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 manager to the workers, and creates the variables `x` and `y` at the workers if they do not already exist, and sets them to the specified values.

To create and set a single variable, there is no need to create an array. For example,

```
pt_shell> set my_var 2.5
2.5
pt_shell> set_distributed_variables my_var
```

Getting Distributed Variable Values

To get the values of variables set in scenarios in command focus, use the `get_distributed_variables` command. The retrieved values are returned to the manager 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 need to specify which attributes to retrieve from the collection. For example, 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.

Merging Distributed Variable Values

By default, the `get_distributed_variables` command brings back a scenario variable as an array. With the `-merge_type` option, the values are merged back into a variable instead.

For example, to keep the minimum (most negative) value of the `worst_slack` variable from all scenarios:

```
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, you have the following array settings in three scenarios:

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

scenario typ_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 report the most negative values, use these commands:

```
pt_shell> get_distributed_variables {slacks} -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 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 `get_distributed_variables` command supports the following merge types:

- `-merge_type min`
- `-merge_type max`
- `-merge_type average`
- `-merge_type sum`
- `-merge_type list`
- `-merge_type unique_list`
- `-merge_type none`

You can use the `-null_merge_method` option to specify what happens when merging null (empty) entries: ignore the null entry, allow the null value to override other entries, or issue an error message. For details, see the man page for the `get_distributed_variables` command.

Note that a null data value is different from an undefined value. If array entries are undefined, the `-null_merge_method` option does not apply; the merging process operates normally on the remaining data values.

Synchronizing Object Attributes Across Scenarios

You can easily set an object attribute to the same value across all scenarios for a specified object, or for each member of a collection of objects.

For example, to synchronize the `dont_touch` attribute of cell U72 to `true` across all scenarios in the current command focus, use the following command:

```
pt_shell> synchronize_attribute -class cell {U72} dont_touch
```

This command works in the DMSA manager and can be used to synchronize the settings for a user-defined attribute or the `dont_touch` application attribute across all scenarios in the current command focus. It cannot be used on application attributes other than `dont_touch`.

The `-merge_type` option can be set to `min` or `max` to specify the synchronizing action. The default is `max`, which synchronizes the attributes to the maximum value found for the

attribute across the scenarios, or the last string in alphanumeric order for a string attribute, or `true` for a Boolean attribute.

Merged Reporting

Running reporting commands in multi-scenario analysis can generate large amounts of data from each scenario, making it difficult to identify critical information. To help manage these large data sets, the tool supports merged reporting.

Merged reporting automatically eliminates redundant data and sorts the data in numeric order, allowing you to treat all the scenarios as a single virtual PrimeTime instance. This feature works with these commands:

```
get_timing_paths
report_analysis_coverage
report_clock_timing
report_constraint
report_min_pulse_width
report_si_bottleneck
report_timing
```

To get a merged timing report, issue the `report_timing` command at the manager as you would in an ordinary single-scenario session. This is called executing the command in the manager context. The manager and worker processes work together to execute the `report_timing` command in all the scenarios in command focus, producing a final merged report displayed at the manager console. Each report shows the scenario name from which it came.

You can optionally specify one or more commands to run in the worker context before or after the merged reporting command. For example,

```
pt_shell> remote_execute -pre_commands {source my_constraints.tcl} \
           -post_commands {source my_data_proc.tcl} {report_timing}
```

This runs the `my_constraints.tcl` script before and the `my_data_proc.tcl` script after the `report_timing` command, all in the worker context.

In situations where there are more scenarios than worker processes, using these options can reduce the amount of swapping scenarios in and out of hosts because all of the scenario-specific tasks (pre-commands, reporting command, and post-commands) are performed together in a single loading of the image onto the host.

To learn about the reporting commands in DMSA, 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 manager 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 name for a particular timing path object, query the `scenario_name` attribute of the timing path object. To specify the attributes to be included in the timing path collection, use the `-attributes` option with the `get_timing_paths` command. By default, only the `full_name`, `scenario_name`, and `object_class` attributes are included.

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 arrival time does not meet the timing requirements
- *Met* if the timing requirements are satisfied
- *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, it is reported as either Violated or Met, even if untested in some scenarios. This allows meaningful reporting with significantly different behavior, such as functional versus test modes.

The merged report includes the scenario names, as shown in the following example.

Example 2 Summary merged analysis coverage report

```
pt_shell> report_analysis_coverage
...
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. The following example shows a report where some scenarios are met and others are untested.

Example 3 Detailed merged analysis coverage report

```
pt_shell> report_analysis_coverage -status_details {untested met} \
    -sort_by slack
```

...

Scenarios: scen1, scen2, scen3

Type of Check	Total	Met	Violated	Untested
<hr/>				
setup	15	2 (13%)	12 (80%)	1 (7%)
hold	15	12 (80%)	2 (13%)	1 (7%)
<hr/>				
All Checks	30	14 (47%)	14 (47%)	2 (7%)
<hr/>				
Constrained Pin	Related Pin	Clock	Check Type	Reason
ffd/CR	CP(rise)		hold	scen2 scen1
ffd/CR	CP(rise)		setup	scen2 scen1
ffa/D	CP(rise)	clk2	hold	scen2
ffa/D	CP(rise)	CLK	hold	scen2
ffb/D	CP(rise)	CLK	hold	scen2
ffb/D	CP(rise)	clk2	hold	scen2
ffd/D	CP(rise)	CLK	hold	scen2
ffd/D	CP(rise)	clk2	hold	scen2
ffa/D	CP(rise)	clk3	hold	scen1
ffb/D	CP(rise)	clk3	hold	scen1
ffc/D	CP(rise)	clk3	hold	scen1
ffc/D	CP(rise)	CLK	hold	scen1
ffd/D	CP(rise)	clk3	hold	scen1
ffd/CR	CP(rise)	CLK	setup	scen3
ffd/CR	CP(rise)	clk2	setup	scen3
ffc/D	CP(rise)	clk2	hold	scen2

report_clock_timing

The `report_clock_timing` command executed by the manager 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 4 Merged report_clock_timing report

```
pt_shell> report_clock_timing -type latency -nworst 3
...
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 manager process returns a merged report that displays constraint-based information for all scenarios in command focus. It reports the size of the worst violation and the design object that caused the violation.

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 for consistent results. When you specify the `-verbose` or `-all_violators` option, PrimeTime reports the scenario name for the constraint violators.

Example 5 DMSA merged constraint report

```
pt_shell> report_constraint -all_violators -path_type slack_only
...
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
-----
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	
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 6](#) shows the summary output for the merged DMSA constraint report shown in [Example 7](#).

Example 6 DMSA merged summary report

```
pt_shell> report_constraint
...
Weighted
Group (max_delay/setup) Cost Weight Cost
-----
default 1.80 - 1.80
CLK 4.12 - 4.12
-----
max_delay/setup 5.92

Weighted
Group (min_delay/hold) Cost Weight 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)
```

The following example shows the output of an all-violators constraint report with the `max_capacitance` option for a multi-scenario analysis:

Example 7 DMSA merged constraint report

```
pt_shell> report_constraint -all_violators -path_type slack_only \
    -max_capacitance
...
Scenarios: scen1 scen2
max_capacitance
Required Actual
Pin Scenario Capacitance Capacitance 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_min_pulse_width

To locate the pins with the most critical minimum pulse width violations, use the `report_min_pulse_width` command. The tool prunes, sorts, and reports the pins with the worst pulse width violations across all scenarios in focus, as shown in the following example.

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

```
pt_shell> report_min_pulse_width
```

```
...
```

```
Scenarios: scen1 scen2
```

clock_tree_pulse_width		Required	Actual	
Pin	Scenario	pulse width	pulse width	Slack
<hr/>				
u2/i (high)	scen1	10.00	4.99	-5.01
u1/zn (high)	scen1	10.00	4.99	-5.01
p1/i (high)	scen2	10.00	4.99	-5.01
u2/zn (high)	scen2	10.00	4.99	-5.01

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 manager.

```
pt_shell> report_si_bottleneck \
    -cost_type total_victim_delay_bump -max \
    -slack_lesser_than 2 -significant_digits 6
```

```
...
```

```
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 manager 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 focused on one portion of the design that is critical in all scenarios. To prevent merging of paths, use the `-dont_merge_duplicates` option of the `report_timing` command.

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 manager 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 Manager/Worker 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 Manager/Worker Task Processing

Startpoint: txethmac1/WillTransmit_reg
(rising edge-triggered flip-flop clocked by mtx_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                               Incr      Path
-----
clock mtx_clk_pad_i (rise edge)     0.00      0.00
library hold time -                0.06      0.67
data required time                 0.67
-----                                0.67
data required time                 -0.76
-----                                -0.76
slack (MET)                         0.10

Startpoint: txethmac1/WillTransmit_reg
(rising edge-triggered flip-flop clocked by mtx_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                               Incr      Path
-----
clock mtx_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 time 1.49

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
-----                                1.37
data required time                 -1.49
-----                                -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](#)

Standard Option Handling

All of the options of the `report_timing` command are available for merged reporting, with a few exceptions (see [Limitations of DMSA](#)). Provided collections are not being passed to the options, they can be specified at the manager 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 worker and generates a merged report at the manager. The `all_in` variable and the `all_outputs` expression are evaluated in a worker context.

To evaluate expressions and variables locally at the manager, enclose the command string in quotation marks. All manager 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 manager then refers to the explicit collection using the name you specified.

Example 8 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 manager using curly braces around the worker expression. At the worker, 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 manager and workers variables and expressions from the multi-scenario manager, the following options have been altered at the multi-scenario manager:

-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.
```

Loading Scenario Design Data Into the Manager

By default, the manager in a DMSA analysis does not load the design data. This reduces the memory and runtime requirements and is sufficient for most merged reporting and ECO tasks.

For more complex tasks, you can use the `load_distributed_design` and `cache_distributed_attribute_data` commands to load design and library data at the manager:

- [Loading Netlist and Library Data](#)
- [Loading Scenario-Specific Design Data](#)

Loading Netlist and Library Data

The `load_distributed_design` command loads the netlist and library data that is the same across all scenarios, such as pin directions and timing arcs. It loads the netlist and libraries from the first scenario in the `current_scenarios` list. Data that is not guaranteed to be identical across scenarios, such as timing or slack data, is not loaded.

Once the netlist and library data is loaded,

- Netlist objects can be obtained with the `get_cells`, `get_pins`, `get_timing_arcs`, and similar commands.
- Library objects can be obtained with the `get_lib_cells`, `get_lib_pins`, and `get_lib_timing_arcs` commands.

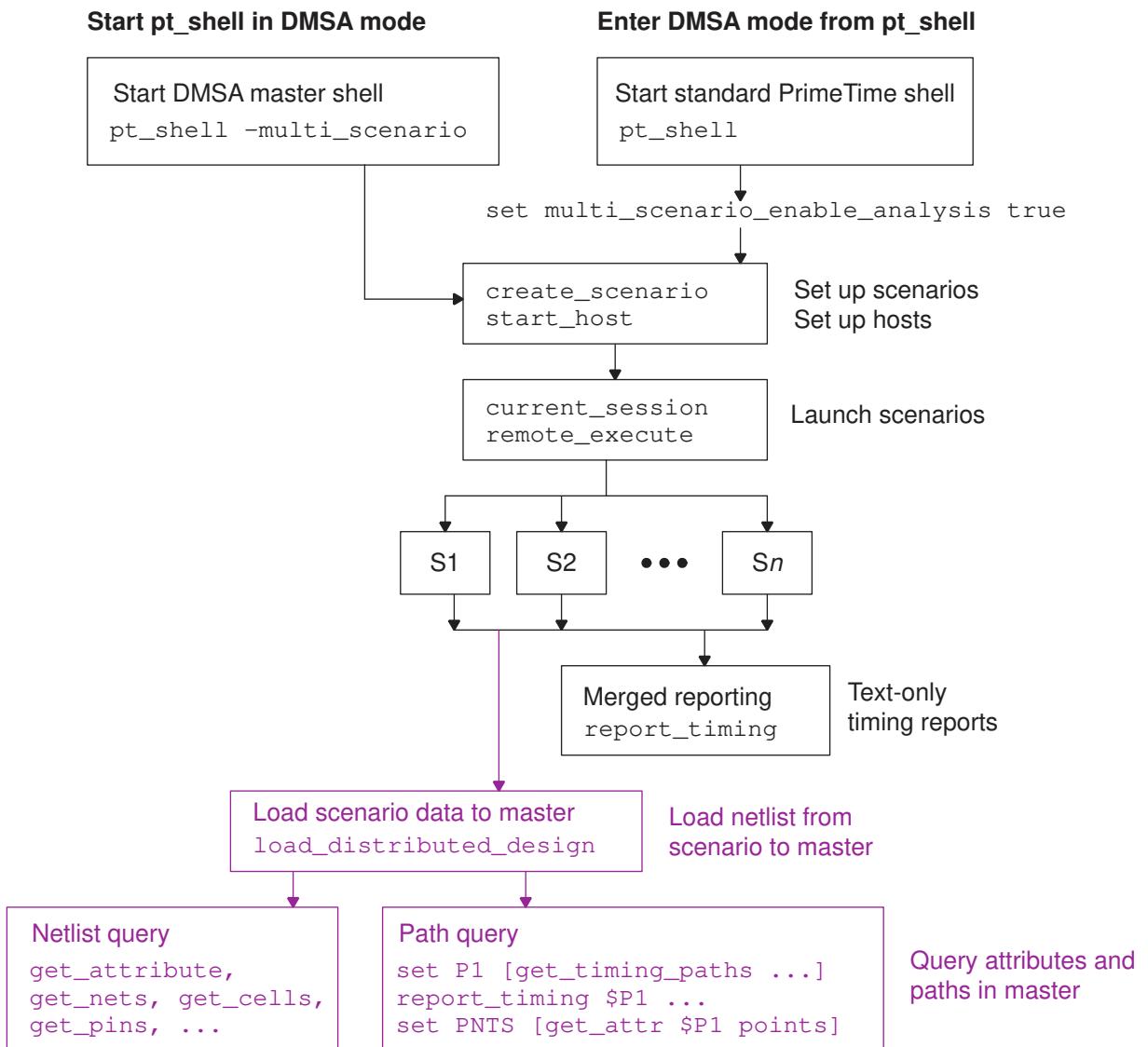
- The `get_timing_paths` command returned true `timing_path` collection objects (instead of abstracted timing path objects stored internally as text), and most attributes of the paths and paths points can be queried as in non-DMSA analysis.
- The netlist topology can be explored with the `all_fanin` and `all_fanout` commands.
- In the PrimeTime GUI, the ECO and Path Analyzer features can be used.

For example,

```
pt_shell> load_distributed_design
...
pt_shell> set mypaths [get_timing_paths -max_paths 50 ...]
...
pt_shell> report_timing $mypaths -path_type full_clock_expanded -voltage
...
pt_shell> get_nets -hierarchical -of_objects [get_cells ...]
...
pt_shell> get_attribute [get_cells buf1] number_of_pins
...
```

The following diagram shows the additional netlist and path query capabilities when you load netlist and library data into the manager.

Figure 18 Netlist and Path Query After Loading Scenario Data Into Distributed Manager



All scenario netlists must be identical (but the constraints and operating conditions can vary). If there are any netlist differences, or if there is no current scenario at the manager, the tool issues an error:

Error: Restore session at manager failed due to: Scenario has a different signature compared to known signatures. (MS-037)

Loading a netlist at the manager reserves a PrimeTime license for the DMSA manager, meaning that this license (or part of it in the case of core-based licensing) is not shared

with any worker processes running DMSA scenarios. In contrast, without loading the netlist, a PrimeTime license checked out by the manager at startup is shared with remote workers.

To remove loaded distributed data, use the `remove_distributed_design` command.

Loading Scenario-Specific Design Data

The `cache_distribute_attribute_data` command loads scenario-specific design data from the currently active scenarios into the manager. This command requires that the design data already be loaded using the `load_distributed_design` command.

To conserve memory, the command loads only the attributes you specify into the manager. For example,

```
pt_shell> current_scenario {scen0 scen1 scen2}

pt_shell> cache_distribute_attribute_data \
           -attributes max_rise_slack -class pin
Start of Manager/Worker Task Processing
...
End of Manager/Worker Task Processing
```

The supported `-class` values are `port`, `cell`, `pin`, and `net`.

After loading the attribute data into the manager, you can query it normally to obtain values for all scenarios in focus, or use subscripted querying to obtain per-scenario values:

```
pt_shell> get_attribute $my_pin max_rise_slack
{scen0 48.816082 scen1 45.250504 scen2 46.121669}

pt_shell> get_attribute $my_pin max_rise_slack(scen1)
45.250504
```

Multiple calls to the `cache_distribute_attribute_data` command are cumulative.

To remove some or all of the attribute data from memory at the manager, use the `purge_distributed_attribute_data` command. For details, see the man page.

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

In a DMSA analysis, the manager performs all license management. It acquires any licenses needed by the worker processes from the central license server, then dynamically distributes them to the worker processes.

You can use the following licensing features in a DMSA analysis:

- [Distributed Core-Based Licensing](#) – Each PrimeTime-ADV or PrimeTime-APX license enables 32 or 64 cores respectively across the scenarios, regardless of how the cores are distributed across the scenarios. This often reduces the overall license requirements of the analysis.
- [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 manager 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.

Limiting License Usage in a DMSA Analysis

To limit the maximum number of licenses the manager can acquire for use by the worker processes, use the `set_license_limit` command at the manager:

```
pt_shell> set_license_limit PrimeTime -quantity 5
```

By default, licenses are checked out only when the workers begin processing tasks. To check out licenses immediately, use the `get_license` command at the manager:

```
pt_shell> get_license PrimeTime-SI -quantity 5
```

For best DMSA analysis throughput, match the license limit to the number of worker processes. This ensures that all worker processes can execute concurrently.

See Also

- [Manually Controlling License Usage](#) for more information on manually controlling license usage

Worker Process Fault Handling

When a fault occurs in a worker process that causes an abnormal termination, the tool takes action as determined by the `multi_scenario_fault_handling` variable, which you can set to `ignore` or `exit`.

When the variable is set to `ignore` (the default) and a critical fault occurs in a worker 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 manager 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 manager 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.

When the variable is set to `exit` and a critical fault occurs in a worker process, the active command completes its task and then the manager process terminates the entire analysis. An information message is displayed explaining what occurred.

Merged Reporting and Worker Fault Handling

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 worker 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. It 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 and Worker Fault Handling

Before a netlist change is committed to any scenario from the manager, 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.

remote_execute Command and Worker Fault Handling

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 and Worker Fault Handling

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 manager, all data is written to a set of log files. A limited set of interactive messages are displayed at the manager.

If you use the `-verbose` option, the `remote_execute` command sends all data to the manager 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 workers, the workers send messages back to the manager to indicate their progress.

Progress Messages

As the workers progress through each analysis stage, the workers 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 Manager/Worker 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 Manager/Worker 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 Manager/Worker 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 you need more control over the output. 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 worker for a given scenario, the error is reported in full at the manager together with the name of that scenario. For example:

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

Start of Manager/Worker 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 worker for a given scenario, the warning is reported at the manager 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 Manager/Worker 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 workers 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 `multi_scenario_working_directory` variable. If you do not explicitly set this variable, it is set to the current directory from which the PrimeTime manager was invoked. You must have write permission to the working directory, and the directory must be accessible by the manager and to all workers.

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 managers 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 manager. 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 worker processes attempt to write to the same file, which could lead to read/write conflicts and cause the worker 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 manager process.

Example 2: Script-based redirection

The `multi_scenario_working_directory` variable has been set to `/mypath`, and there are two scenarios in command focus, `s3` and `s4`. The manager 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:

```
/mypath/s3/rt.log
/mypath/s4/rt.log
```

DMSA Variables and Commands

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

- [DMSA Variables](#)
- [DMSA Commands](#)
- [Commands Not Allowed on Worker Processes](#)

DMSA Variables

To customize multi-scenario reporting, set the following variables.

Variable	Description
multi_scenario_merged_error_limit	Specifies the maximum number of errors of a particular type to be written to the command log on a per-task basis. Default is 100.
multi_scenario_merged_error_log	Specifies a file location where error, warning, and informational messages are stored for data produced by the workers. Default is "". Set this variable before running the <code>start_hosts</code> command.
multi_scenario_working_directory	Specifies a working directory that can be used by all workers and the manager for which you have write permissions. The default is the current working directory. Set this variable before running the <code>start_hosts</code> command.
sh_host_mode	Specifies the working mode: <code>scalar</code> , <code>manager</code> , or <code>worker</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 manager or worker. For example, to set the working directory from only the manager, you could use the following: <pre>if {\$sh_host_mode eq "manager"} { set multi_scenario_working_directory "/home/wd" }</pre>

DMSA Commands

The following table lists the commands that you use throughout the DMSA flow.

Table 4 DMSA commands

Category	Command	Description
Scenario setup	create_scenario	Creates a single scenario. This command is required.
	remove_scenario	Removes one or more specified scenarios.
	remove_multi_scenario_design	Removes all multi-scenario data specified, including the session, all scenarios, and the configuration.
	report_multi_scenario_design	Reports the current user-defined multi-scenario analysis.
Host and license configuration	set_host_options	Define the compute resources. This command is required.
	remove_host_options	Removes all distributed hosts added to the farm's pool.
	set_license_limit	Limits the number of licenses the manager process acquires for worker usage.
	start_hosts	Starts the worker processes on the hosts specified by the <code>set_host_options</code> command. This command is required.
	report_host_usage	Reports host and resource usage.
Analysis focus	current_scenario	Changes the command focus to the selected scenarios within the current session; returns a list of all scenarios in command focus.
	current_session	Puts a particular group of scenarios in focus for analysis. The scenarios in focus receive and interpret commands from the manager; 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 <code>remove_scenario</code> or <code>quit</code> command.
Worker context	remote_execute	Batch or interactive mode; from the manager, creates a command buffer to be executed in the worker context.

Commands Not Allowed on Worker Processes

You cannot run the following commands on worker processes.

Command	Description
exit	Exits <code>pt_shell</code> .
quit	Quits <code>pt_shell</code> .
remove_design	Removes one or more designs from memory.
remove_lib	Removes one or more libraries from memory.
rename_design	Renames a design.
restore_session	Restores a <code>pt_shell</code> session previously created with the <code>save_session</code> command.

Limitations of DMSA

The DMSA feature has the following limitations:

- The maximum number of worker processes that the manager process can control is 256 concurrent hosts.
- Usage of the GUI is not supported.

HyperGrid Distributed Analysis

Timing analysis for very large designs is challenging. If the design contains physical design blocks, you can use [hierarchical timing analysis](#) to first verify the blocks individually, then verify the top-level design with block models.

HyperGrid distributed analysis can analyze very large designs in a single run by parallelizing the analysis across multiple worker processes running on different hosts. The entire design can be explored and reported with the same commands used in a regular PrimeTime analysis.

HyperGrid distributed analysis can be used for designs that are not suitable for hierarchical timing analysis flows. It can also be used as an additional final verification in hierarchical flows.

This feature requires a PrimeTime-ADV-PLUS license.

The following topics provide more information about HyperGrid distributed analysis:

- [Overview of the HyperGrid Flow](#)
 - [Preparing for Distributed Analysis](#)
 - [Running the Distributed Analysis](#)
 - [Querying Attributes in a HyperGrid Analysis](#)
 - [Saving a Distributed Session](#)
 - [Commands With HyperGrid Distributed Analysis Support](#)
 - [Limitations of HyperGrid Distributed Analysis](#)
-

Overview of the HyperGrid Flow

HyperGrid distributes the analysis of a very large design across multiple machines, using the same analysis script and generating the same reports as full-flat analysis.

The following topics describe how HyperGrid performs distributed analysis:

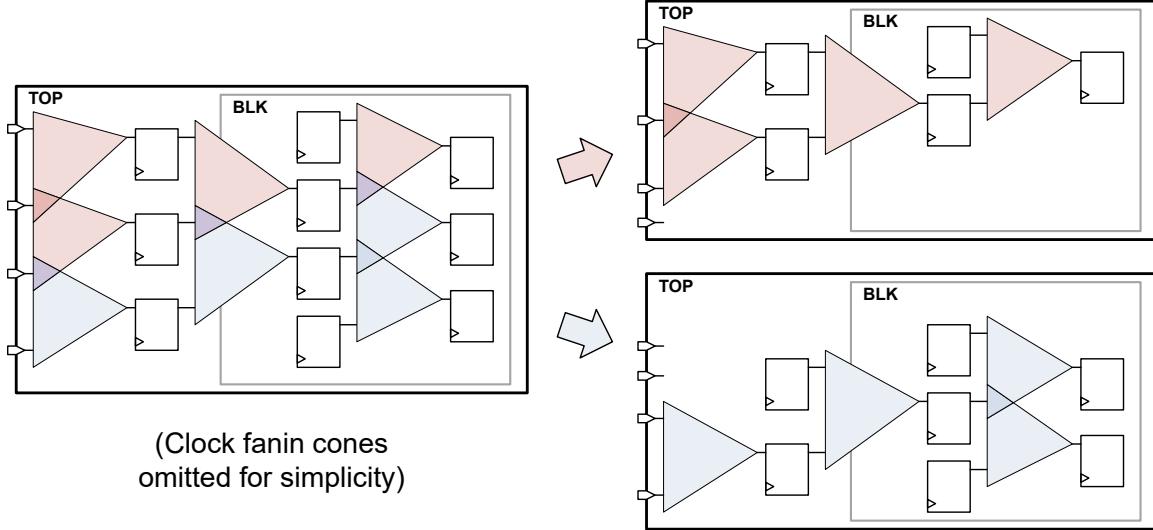
- [Partitioning the Design](#)
- [The Manager Process and the Worker Processes](#)
- [The Distributed Analysis Working Directory](#)

Partitioning the Design

In a HyperGrid distributed analysis, the process of splitting up the design is called *partitioning*.

HyperGrid partitioning is not restricted by logical design hierarchy. Instead, it partitions by fanin logic cones to clock and data endpoints. Fanin cones can traverse through hierarchical blocks. Different partitions can contain different portions of the same block.

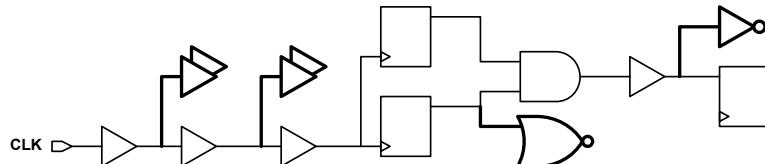
Figure 19 HyperGrid Partitioning, Using Fanin Logic Cones to Endpoints



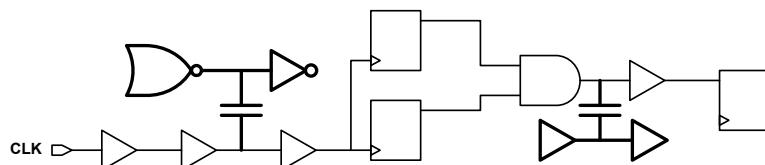
Each partition includes the fanin logic cones to a set of clock/data endpoints. The partitioning process uses netlist, detailed parasitics, and constraints data to ensure the full timing context of the design logic is considered.

In addition, the fanin logic cones include the following:

- *Side load stages*, which model the receiver load of paths that diverge to endpoints outside the current partition:



- *Aggressor stages*, which model cross-coupled timing and noise interactions between partitions:



HyperGrid synchronizes signal integrity effects between partitions as needed.

Different endpoints in different partitions can share common startpoints. In this case, some portion of the startpoint fanout is included in both partitions.

By their nature, clock fanin cones tend to have fewer startpoints and more side load pins than data fanin cones.

The Manager Process and the Worker Processes

HyperGrid distributed analysis uses a single *manager* process that invokes and controls multiple *worker* processes.

The manager process is the process invoked by the user. It does the following:

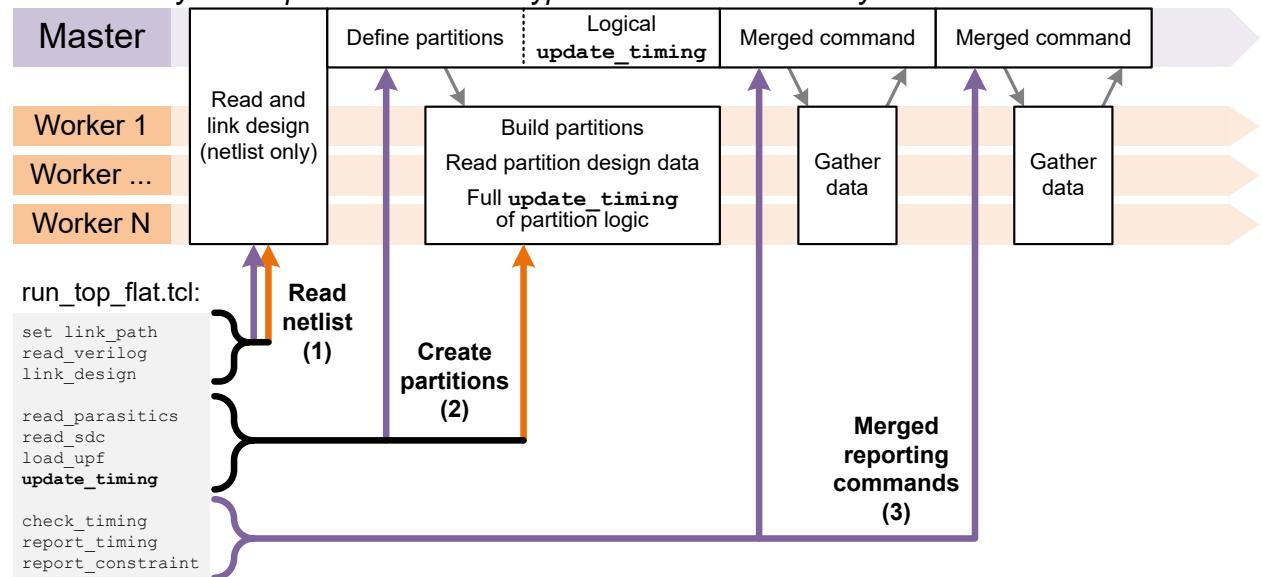
- Determines how to partition the design
- Invokes the worker processes (one for each partition)
- Generates full-design reports by querying the workers for information and merging it together

The worker processes are invoked by the manager process. A worker process does the following:

- Reads the design data for a specific partition
- Performs timing and noise updates
- Gathers and returns results for queries sent by the manager process

[Figure 20](#) shows how a full-flat analysis script is run in a HyperGrid distributed analysis.

Figure 20 Analysis Script Execution in a HyperGrid Distributed Analysis

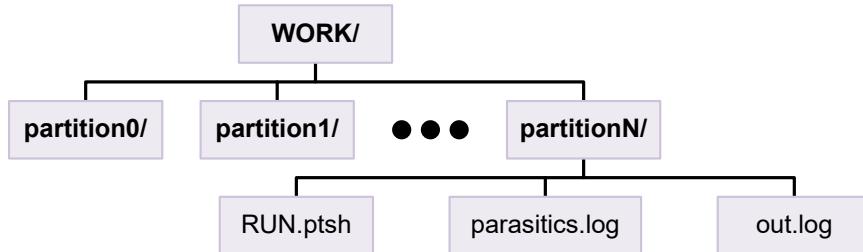


HyperGrid distributed analysis always uses core-based licensing. See [Distributed Core-Based Licensing](#) for details.

The Distributed Analysis Working Directory

In HyperGrid distributed analysis, the distributed analysis working directory contains tool-generated scripts and log files data for the partitions.

The directory structure in the working directory is as follows:



After the design is linked, the `out.log` file in each partition directory shows the partition's characteristics. For example,

```

Partitioning for DSTA Worker
Design: bit_coin
  Leaf Pins and Ports: 999296
  Hierarchical Pins: 163677
  Leaf Cells: 234424
  Hierarchical Cells: 5739
  Nets: 254436
  Number of Partitions: 4
-----
Partition#: 2
  Leaf Pins: 258846      25.90%
  Topo Pins: 237420      23.75%
  Receiver Pins: 2963      0.29%
  Dangling Pins: 15426     1.54%
  Pre Dangling Pins: 4253     0.42%
  Leaf Nets: 64281       25.26%
  Topo Nets: 55087        21.65%
  Receiver Nets: 744       0.29%
  Dangling Nets: 6866       2.69%
  Pre Dangling Nets: 1584     0.62%
  Leaf Cells: 63052       26.89%
  
```

Preparing for Distributed Analysis

The following topics describe how to prepare for HyperGrid distributed analysis:

- [Creating a Wrapper Script](#)

Creating a Wrapper Script

HyperGrid distributed analysis requires a *wrapper script* that enables HyperGrid, configures and starts the worker processes, then runs the actual analysis script in distributed mode.

An example wrapper script is as follows:

```
# enable Hypergrid analysis
set_app_var distributed_enable_analysis true
set_app_var distributed_working_directory ./MY_WORK_DIR

# start worker processes
set_host_options \
    -num_process 4 \
    -max_cores 4 \
    -submit {...compute farm submission command...}
start_hosts

# run the timing analysis script, distributed across the workers
start_dstas \
    -script run_top_flat.tcl
```

The important commands in the wrapper script are as follows:

- The `distributed_enable_analysis` variable enables HyperGrid distributed analysis.
- The `distributed_working_directory` variable sets the location of the working directory. The default directory name is `dsta_working_dir`.
- The `set_host_options` command configures the remote worker processes, and the `start_hosts` command starts them. For details on how to configure worker processes, see [Setting Up Distributed Host Options](#).
- The `start_dstas` command begins distributed analysis. Specifically, it does the following:
 - Partitions the design (at the master)
 - Loads partitioned design data (at the workers)
 - Runs the specified analysis script (at the master, communicating with the workers)

In the wrapper script, any commands that follow the `start_dstas` command are executed as distributed reporting commands when the specified analysis script completes.

Running the Distributed Analysis

To run a HyperGrid distributed analysis,

1. Create a wrapper script, as described in [Preparing for Distributed Analysis](#).
2. Run the wrapper script in a new PrimeTime session:

```
% pt_shell -f wrapper.tcl
```

If the analysis script (called by the wrapper script) redirects command output to files before the `update_timing` command, the target file names must be relative paths. Otherwise, conflicts will occur as multiple worker processes attempt to write to the same file while reading and constraining the design.

If the analysis script runs the `exit` or `quit` command, the distributed analysis session is ended.

Querying Attributes in a HyperGrid Analysis

In a distributed analysis, the manager process maintains a logical netlist representation of the full design, but it does not compute any timing information.

When an attribute is queried at the manager process, the tool behavior depends on where the attribute data is stored:

- If the attribute data is available from the logical netlist representation at the manager process (such as object names or port/pin directions), then the query returns immediately.
- If the attribute data is available only from partition-level analysis (such as timing or noise information), then the data must be obtained by requesting it from the worker processes.

To improve performance for partition-sourced attributes, HyperGrid provides *attribute caching* functionality that minimizes the amount of data transfer needed from worker processes.

Automatic Attribute Caching in `foreach_in_collection`

When an attribute query is performed inside a `foreach_in_collection` loop at the manager process, HyperGrid detects the queried attribute and caches its values from the partitions for faster subsequent loop iterations. For example,

```
foreach_in_collection my_pin $all_pins_in_design {
    ...
    # the 'max_rise_slack' attribute is automatically cached
    set slack [get_attribute [get_pin $my_pin] max_rise_slack]
}
```

The attribute data is cached only for the loop collection objects, and only for the duration of the loop. When the loop completes, the cache is released. This minimizes memory overhead while still providing a performance improvement.

Explicit Attribute Caching

Scripts can explicitly cache attribute data at the manager process by using the `cache_distributed_attribute_data` command. For example,

For example,

```
# explicitly cache pin slack attributes
cache_distributed_attribute_data \
    -class pin \
    -attributes {min_rise_slack max_rise_slack}

while {...} {
    ...
    set slack [get_attribute [get_pin $my_pin] max_rise_slack]
}
```

These explicitly cached attributes are available to all subsequent commands until explicitly released by the `purge_distributed_attribute_data` command.

For details, see the man pages.

Saving a Distributed Session

You can use the `save_session` command to save a HyperGrid distributed analysis session to disk.

The resulting directory contains the data from the manager process and all worker processes. Due to overhead in the distributed design data, the size of a distributed analysis session directory is larger than an equivalent full-flat session directory.

When you restore a distributed session, the tool re-creates the original distributed analysis with the same partitions, starting worker processes with the same `set_host_options` specifications as in the original distributed analysis.

To restore a saved distributed session,

1. (Optional) Set the location of the distributed analysis working directory:

```
pt_shell> set_app_var distributed_working_directory ./MY_WORK_DIR
```

2. Use the `restore_session` command in the usual way:

```
pt_shell> restore_session my_distributed_session
```

Commands With HyperGrid Distributed Analysis Support

The following tool commands support HyperGrid distributed analysis:

```
check_noise
check_timing
get_attribute
get_clock_relationship
get_timing_paths
report_analysis_coverage
report_annotated_parasitics -list_annotated
report_attribute
report_clock_gating_check
report_constraint
report_crpr
report_delay_calculation
report_global_slack
report_global_timing
report_min_period
report_min_pulse_width
report_net
report_noise
report_noise_calculation
report_qor
report_timing
restore_session
save_session
write_rh_file
```

The following commands are supported for querying and exploring multivoltage information at the manager process:

```
add_power_state
add_pst_state
create_pst
create_supply_set
get_supply_groups
get_supply_nets
get_supply_ports
get_supply_sets
report_power_domain
report_power_pin_info
report_power_switch
report_pst
report_supply_group
report_supply_net
report_supply_set
report_timing -supply_net_group
set_retention
set_retention_control
set_retention_elements
```

Limitations of HyperGrid Distributed Analysis

HyperGrid distributed analysis does not support the following PrimeTime flows and features:

- AOCV depth adjustment (distance adjustment is supported)
- ECO fixing (automatic or manual), netlist editing commands
- ETM models
- HyperScale hierarchical analysis
- HyperTrace accelerated PBA
- SMVA/DVFS simultaneous multivoltage analysis

Reporting the CPU Time and Memory Usage

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 use single-core analysis, multicore analysis, or distributed multi-scenario analysis, 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
...
```

Chapter 4: Managing Performance and Capacity

Reporting the CPU Time and Memory 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)	
<hr/>			
(local process)	-	3	25
my_opts1	1	1	14
	2	2	15

When you exit PrimeTime, the tool reports the peak memory, CPU usage, and the elapsed time for the session:

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

When exiting a DMSA session, the tool reports information about the distributed processes:

```
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
```

Flow Summary Reports

Modern-day analysis flows are complicated. The full analysis flow is split across many scripts: tool configuration, design read and link, design constraints, and reporting. Many times, scripts are nested and call more scripts. Determining the runtime consumed by

major steps in the flow is hard; tracking down where an unexpected implicit update occurs is even harder.

Fortunately, the PrimeTime tool provides a *flow summary* feature that

- Generates a structured summary of script execution throughout the analysis flow
- Highlights notable tool events (design link, tool updates, and so on)
- Reports the runtime used by scripts and notable commands within them
- Highlights repeated commands that consume nontrivial runtime as a group
- Incurs little performance overhead (less than 5 percent)

The Flow Summary Report Structure

A flow summary is a textual representation of script execution, with annotations added for notable and time-consuming events.

Flow Summary Context Levels

The opening START marker represents where the flow summary feature was first enabled, and subsequent Indentation levels correspond to nested script execution:

```
START {
| source read_design.pt {
| | <<< Event: Invoking link_design; File: read_design.pt; Line: 3; "link"
| } 0.6s
| source constraints.pt {
| | source multicycles.pt {
| | | <<< Event: Invoking logical_update_timing; File: multicycles.pt; Line: 3;
| | | "all_fanout -flat -endpoints_only -from _sel17"
| | } 0.1s
| } 0.5s
| <<< Event: Invoking update_timing; File: run.pt; Line: 8; "update_timing"
| update_timing: 14.9s
| uncaptured: 0.2s
} 16.3s
```

You can use a text editor that supports finding matching opening/closing braces to easily navigate along context levels in the summary.

Event Markers

“<<< Event” markers indicate notable tool events, along with the command (including script and line number) that caused them:

```
START {
| source read_design.pt {
| | <<< Event: Invoking link_design; File: read_design.pt; Line: 3; "link"
| } 0.6s
| source constraints.pt {
| | source multicycles.pt {
```

```

| | | <<< Event: Invoking logical_update_timing; File: multicycles.pt; Line: 3;
| | | "all_fanout -flat -endpoints_only -from _sel17"
| | } 0.1s
| | } 0.5s
| <<< Event: Invoking update_timing; File: run.pt; Line: 8; "update_timing"
| update_timing: 14.9s
| uncaptured: 0.2s
} 16.3s

```

The possible event types are:

- Invoking link_design
- Invoking update_timing
- Halting background parasitics read
- Invalidating update (logical)
- Invalidating update (full)
- Invoking logical update_timing
- Invoking update_noise
- Forcing full update
- *helper_command* > 30s

Notable Command Markers

Commands that consume nontrivial amounts of runtime (10 seconds or more) are explicitly noted with their runtime:

```

START {
| source read_design.pt {
| | <<< Event: Invoking link_design; File: read_design.pt; Line: 3; "link"
| } 0.6s
| source constraints.pt {
| | source multicycles.pt {
| | | <<< Event: Invoking logical_update_timing; File: multicycles.pt; Line: 3;
| | | "all_fanout -flat -endpoints_only -from _sel17"
| | } 0.1s
| | } 0.5s
| <<< Event: Invoking update_timing; File: run.pt; Line: 8; "update_timing"
| update_timing: 14.9s
| uncaptured: 0.2s
} 16.3s

```

In this example, note that the `update_timing` command produced both a timing-update event and a notable command entry.

Uncaptured Command Markers

In general, fast commands are not explicitly listed. Instead, their total runtime is summarized by an “uncaptured” runtime marker:

```
START {
| source read_design.pt {
| | <<< Event: Invoking link_design; File: read_design.pt; Line: 3; "link"
| } 0.6s
| source constraints.pt {
| | source multicycles.pt {
| | | <<< Event: Invoking logical_update_timing; File: multicycles.pt; Line: 3;
| | | "all_fanout -flat -endpoints_only -from _sel17"
| | } 0.1s
| } 0.5s
| <<< Event: Invoking update_timing; File: run.pt; Line: 8; "update_timing"
| update_timing: 14.9s
| uncaptured: 0.2s
} 16.5s
```

Command Bundles

If there are multiple fast commands whose cumulative runtime becomes notable (greater than 10 seconds), they are grouped into *command bundles*:

<pre># commands in thisfile.tcl: get_pins x → 3s report_timing → 8s report_timing → 4s report_timing → 15s report_timing → 9s get_pins y → 2s report_timing → 5s get_pins z → 4s</pre>		<pre>source thisfile.tcl { report_timing x 2: 12s <<< Event: Invoking report_timing File: ...; Line: ... report_timing x 2: 14s --uncaptured -- : 9s }</pre>
---	--	---

Commands are not bundled across notable command entries. This keeps the report aligned with the actual order of command execution.

Helper Commands

A *helper* command is a command that is typically used to provide information to other commands. They are: `get_pins`, `get_nets`, `get_ports`, `get_cells`, `all_fanin`, and `all_fanout`.

Helper commands are normally not interesting by themselves, except when they contribute unexpectedly to runtime. For this reason, only helper commands of 30 seconds or more are considered notable, and they are reported using a special “notable helper command” event marker.

Helper command bundles contain helper commands of less than 30 seconds. Only helper command bundles that are 30 seconds or more are considered notable and are explicitly reported.

If a helper command has both notable (30 seconds or more) and fast command executions, the notable commands are explicitly reported, and the surrounding fast commands are grouped and processed in the usual way:

<pre># commands in thisfile.tcl: get_pins → 3s get_pins → 4s get_pins → 35s get_pins → 8s get_pins → 8s get_pins → 8s get_pins → 8s</pre>		<pre>source thisfile.tcl { <<< Event: get_pins > 30s; File: ...; Line: ...; "get_pins -hier -filter ..." get_pins: 35s get_pins x 4: 32s --uncaptured -- : 7s }</pre>
---	---	--

Runtimes

The flow summary shows runtime information for commands (notable and uncaptured) and closing context levels:

```
START {
| source read_design.pt {
| | <<< Event: Invoking link_design; File: read_design.pt; Line: 3; "link"
| } 0.6s
| source constraints.pt {
| | source multicycles.pt {
| | | <<< Event: Invoking logical_update_timing; File: multicycles.pt; Line: 3;
| | | "all_fanout -flat -endpoints_only -from _sel17"
| | } 0.1s
| } 0.5s
| <<< Event: Invoking update_timing; File: run.pt; Line: 8; "update_timing"
| update_timing: 14.9s
| uncaptured: 0.2s
} 16.3s
```

All runtime values are elapsed wall clock time measurements.

Generating a Flow Summary Report

To start a new flow summary report, set the `sh_flow_summary_file` variable to the file name where the summary should be written to:

```
pt_shell> # start a new flow summary file
pt_shell> file delete ./summary.txt
pt_shell> set_app_var sh_flow_summary_file {./summary.txt}
Information: Writing flow summary to summary.txt.
```

By default, this variable is set to an empty string value, which disables the feature.

If the specified file already exists, subsequent flow summary information is appended to the end. This allows you to turn the feature on and off throughout your flow to include only the areas of interest. But as a result, any previous summary file must be deleted to start a new file.

The flow summary is written out incrementally during tool execution. This allows you to examine the flow summary of an in-progress analysis.

To stop writing information to a flow summary report, set the `sh_flow_summary_file` variable to an empty string:

```
pt_shell> # stop the current flow summary
pt_shell> set_app_var sh_flow_summary_file {}
Information: Closing flow summary file summary.txt.
```

When you stop a flow summary report, the tool adds a final line to the output file with the total overhead of the flow summary feature itself:

```
Runtime overhead of flow summary utility is: 3.1 s
```

Defining Custom Flow Segments

Custom *flow segments* allow you to group part of a script into an additional named context level of structure. This allows you to separate parts of a long script, or gain insight into where uncaptured runtime is occurring.

To do this, use the `define_custom_flow_segment` command to define named start and end points in a script:

```
# ...preceding commands...
define_custom_flow_segment \
    -start -name drc_max_tran
get_pins -hier -filter "..."      → 750s
foreach_in_collection {
    report_timing
}
...write to file...                → 500s
define_custom_flow_segment \
    -end -name drc_max_tran
# ...following commands...
```

```
START {
| ...preceding commands...
| CUSTOM_SEGMENT_drc_max_tran
{
| | get_pins: 750s
| | report_timing x 21567:
| | 500s
| | --uncaptured -- : 70s
| } 1320s
| ...following commands...
} 2000s
```

Custom flow segments can contain nested flow segments and calls to other scripts:

```
CUSTOM_SEGMENT_drc {
| source max_transition.pt {
| | CUSTOM_SEGMENT_drc_max_transition {
| | |
| | | ...
| | } 100s
| } 100s
| source max_capacitance.pt {
| | CUSTOM_SEGMENT_drc_max_capacitance {
| | |
| | | ...
| | } 50s
```

```
| } 50s
} 150s
```

A flow segment should start and end in the same script file context. For nested flow segments, flow segments must be closed in order (lower-level before higher-level).

Writing to Multiple Flow Summary Reports

To stop writing information to the current flow summary report and start writing to a new one, set the `sh_flow_summary_file` variable to the new file name:

```
pt_shell> # switch to a new flow summary report
pt_shell> set_app_var sh_flow_summary_file {multicycles.txt}
Information: Closing flow summary file summary.txt.
Information: Writing flow summary to multicycles.txt.
```

If an existing flow summary report is in progress, it is automatically closed.

When stopping and starting flow summary reports from context levels below the top level, note the following:

- If a flow summary report is stopped in a lower context level, the pending levels are closed and written to the report.
- If a flow summary report is started from a lower context level, the currently pending context levels are *not* written as opening levels in the report. Instead, the current context is treated as the top level, and “`<end of source preceding flow summary>`” messages indicate where existing contexts are exited.

For example,

<pre># commands in run.tcl: set_app_var \ sh_flow_summary_file {top.txt} source constraints.tcl ... </pre> <hr/> <pre># commands in constraints.tcl ... source multicycles.tcl </pre> <hr/> <pre># commands in multicycles.tcl set_app_var \ sh_flow_summary_file {mcp.txt} set_multicycle ... set_multicycle ... set_app_var \ sh_flow_summary_file {top.txt}</pre>	 <pre>START { ... source constraints.pt { ... source multicycles.pt { ... } 14.0s } 25.1s } 25.7s Runtime overhead of flow summary is: 0.0 s START { <end of source preceding flow summary> 0.0s <end of source preceding flow summary> 0.0s ... } 14.3s Runtime overhead of flow summary is: 0.0 s</pre>
--	--

Limitations

Note the following limitations:

- Time values are rounded to the nearest 0.1 second for printing. As a result, individual values might not sum precisely to total values.
- Tcl procedures do not introduce explicit levels in the flow summary, although the activity within them is summarized normally.
- The `parallel_execute` and `parallel_foreach_in_collection` commands are listed as single commands without details about the worker threads

Profiling the Performance of Tcl Scripts

To identify runtime and memory bottlenecks in your Tcl scripts, you can perform Tcl profiling during your timing runs. This capability can help you fix performance and capacity issues.

To perform Tcl profiling:

1. Start collecting profiling data by running the `start_profile` command.

By default, `start_profile` tracks commands that take more than 0.1 second to run or use more than 100 KB of memory. If you use the `-verbose` option, `start_profile` tracks all commands.

2. Run the Tcl scripts that you want to analyze.
3. Stop collecting profiling data by using the `stop_profile`, `exit`, or `quit` command.
4. Evaluate the automatically generated Tcl profile and stopwatch reports. By default, the tool writes these files to the profile subdirectory of the current working directory.

Table 5 Tcl profile and stopwatch reports

File name	Description
<code>tcl_profile_summary_latest.html</code>	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.
<code>tcl_profile_summary_latest.txt</code>	Summary page in plain text that shows the start and stop times of the profiling and paths to the report files.
<code>tcl_profile_sorted_by_cpu_time*.txt</code>	Tcl profile report sorted by CPU time.

File name	Description
tcl_profile_sorted_by_delta_mem_*.txt	Tcl profile report sorted by memory usage.
tcl_profile_sorted_by_elapsed_time_*.txt	Tcl profile report sorted by elapsed time.
tcl_profile_sorted_by_num_calls_*.txt	Tcl profile report sorted by number of calls.
tcl_stopwatch_*.txt	Stopwatch report in plain text.

Here is an example of the Tcl script profile report summary. The `get_timing_paths` command was called one time from the `gtp_only.tcl` script and took 8 seconds to execute, which is shown in the Elapsed column.

Example 9 *Tcl script profile report summary*

Procedure/Command	Calls	Elapsed	CPU	Delta Memory
<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_verilog	1	0	0	44
source write_constraints.pt	1	0	0	0

The stopwatch report (`tcl_stopwatch_*.txt`), is an ASCII text file that shows the time, performance, and memory metrics of each command. Here is an example of a stopwatch report.

Figure 21 Stopwatch Report Example

```

####EVNT-1:args=
STOPWATCH[START]: CUR_datetime      = Wed Apr 01 03:11:21 2015
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 01 03:11:46 2015
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 01 03:11:46 2015
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 01 03:11:54 2015
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 01 03:11:54 2015
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 01 03:11:54 2015
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 01 03:11:54 2015
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 01 03:11:59 2015
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

```

Begin profiling

Stopwatch entry when command begins execution

Stopwatch entry when command completes execution

Command

Command options

Current metrics and differences compared to previous stopwatch entry

End profiling

5

Working With Design Data

Before you perform timing analysis, you need to read and link the design and associated logic libraries. To learn about working with design data, see these topics:

- [Logic Libraries](#)
 - [Reading and Linking the Design](#)
 - [Working With Design Objects](#)
 - [Setting the Current Design and Current Instance](#)
 - [Working With Attributes](#)
 - [Saving Attributes](#)
 - [Saving and Restoring Sessions](#)
 - [Creating Test Cases](#)
-

Logic Libraries

A logic library describes the timing and function of macro cells in an ASIC technology by using the Library Compiler tool. Other Synopsys tools, such as the Design Compiler synthesis tool and the IC Compiler place-and-route tool, also use these logic libraries.

A logic library contains library cell descriptions that include

- Cell, bus, and pin structure that describes each cell's connection to the outside world
- Logic function of output pins of cells
- Timing analysis and design optimization information, such as the pin-to-pin timing relationships, delay parameters, and timing constraints for sequential cells
- Other parameters that describe area, power, and design rule constraints

PrimeTime can read logic libraries in the .db and .lib formats. The libraries can have different units of time, capacitance, and voltage.

For more information about logic libraries, see the Library Compiler documentation.

Reading and Linking the Design

Before you perform timing analysis, you need to read and link the design and logic libraries. PrimeTime can read the following file formats:

Input data	Supported file formats
Design data	Binary database (.db) Milkyway Synopsys logical database (.ddc) Verilog VHDL
Logic libraries	Binary database (.db) Synopsys Library Compiler format (.lib)

To read and link the design data, follow these steps:

1. Specify the directories in which PrimeTime searches for designs, logic libraries, and other design data such as timing models. To do this, set the `search_path` variable. For example:

```
pt_shell> set_app_var search_path ". /abc/design /abc/libs"
```

PrimeTime searches the directories in the order that you specify.

2. Specify the libraries in which PrimeTime finds elements in the design hierarchy by setting the `link_path` variable. For example:

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

The variable can contain an asterisk (*), library names, and file names. The asterisk instructs PrimeTime to search for a design in memory. PrimeTime searches libraries in the order that you specify. The main library is the first library in the link path.

3. Read the design into memory:

```
pt_shell> read_verilog TOP.v
```

If the search path includes files that contain the subdesigns, you need to read only the top-level design. The search path entry must include the full path to the subdesign file, not just the directory containing the file.

4. Link the design to resolve references to library cells and subdesigns:

```
pt_shell> link_design TOP
```

During design linking, the tool automatically loads the subdesigns if the subdesign names match the file names.

5. Verify the loaded designs and libraries by using the `list_designs` and `list_libs` commands.

Identifying and Verifying the Required Design Files

It can be useful to quickly identify all design files required by your script, and verify that they exist, before running full timing analysis.

To do this, set the following variable before running your script:

```
pt_shell> set_app_var sh_program_mode file_identification
```

In this mode, the tool opens and immediately closes any accessed files, but no design data is read. Tool commands use the modified behaviors, described in [Table 6](#), designed to allow the script run to completion without design data.

Table 6 Modified Command Behaviors

Command type	Modified behavior
Commands that explicitly or implicitly read libraries	Opens the libraries to confirm read access, then immediately closes them. Libraries are gathered from the configuration settings (the <code>link_path</code> variable, scaling groups, and so on) in the usual way.
Commands that read files	Opens files to confirm read access, then immediately closes them. The <code>search_path</code> variable is used in the usual way.
Commands that return collections	Returns a special <code>=mock_collection=</code> collection handle.
The <code>get_attribute</code> command	For a <code>=mock_collection=</code> object, returns a string value of <code>=mock_collection=attr=</code> .
The <code>sizeof_collection</code> command	For a <code>=mock_collection=</code> object, returns a value of 1.
Conditional and iterating (looping) commands	If the condition or iteration list contains the special <code>=mock_collection=</code> collection handle, then the command body is executed once. Otherwise, the loop or condition is evaluated in the usual way.
The <code>restore_session</code> command	Opens all files in the session directory, then immediately closes them. No data is read or restored.
The <code>echo</code> command	Prints output in the usual way.
All other commands	Returns a value of 1.

You can use the `-x` option of the `pt_shell` invocation command to set the variable before your script runs. Here are some usage examples:

- Running a local script:

```
% $PT_PATH/pt_shell \
-x 'set sh_program_mode file_identification' \
-f my_script.pt
```

- Using the [Synopsys Testcase Packager \(STP\) utility](#):

```
% stp -r \
"$PT_PATH/pt_shell \
-x 'set sh_program_mode file_identification' \
-f my_script.pt"
```

- Using the [Synopsys Cloud Transfer Packager \(SCTP\) utility](#):

```
% sctp -r \
"$PT_PATH/pt_shell \
-x 'set sh_program_mode file_identification' \
-f my_script.pt"
```

To identify the full list of files used by your script, you can use the Synopsys Cloud Transfer Packager. This utility creates a list of required files for review; you do not need to package your test case to generate this list.

Note the following limitations:

- DMSA analysis is not supported.
- Distributed analysis is not supported.
- Scripts that derive file names from design object names (such as creating SDC file names from design names) are not supported.

Per-Instance Link Library Paths

The `link_path` variable provides a link path specification that is used for the entire design.

If needed, you can apply link path specifications to individual cell instances. These *per-instance* link paths take precedence over the global `link_path` specification.

Per-instance link specifications can be used to model process and layout effects, such as body bias and proximity interactions. They can also be used to assign voltage-specific libraries to design blocks (although scaling library groups are recommended for this).

There are two ways to specify per-instance link paths:

- The `link_path_per_instance` variable, set to the list of all per-instance link paths
- The `set_link_lib_map` command, specifying a file containing per-instance link paths

Both methods are functionally identical. The `link_path_per_instance` variable requires no additional files but the specification is stored in memory; it is suitable for smaller link path lists. The `set_link_lib_map` command requires one or more files, read only during link; It is suitable for larger link path lists. They cannot be used together.

link_path_per_instance

The `link_path_per_instance` variable 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. The variable must be set before the design is linked. For example,

```
set_app_var link_path {* lib1.db}
set_app_var link_path_per_instance [list
    [list {ucore1 ucore2}      {* lib2.db}]
    [list {ucore1/usubblk}     {* lib3.db}]
    [list {ucore1/U12 ucore1/U34} {* lib4.db}]]]

read_verilog my_design.v
link_design MY_DESIGN
```

The listed instances can be hierarchical cells (blocks) or leaf cells. All instance paths are relative to the top-level design. The precedence rules are as follows:

- Specifications propagate downward into hierarchy.
- Lower-level specifications take precedence over higher-level specifications.
- Leaf cell specifications take precedence over block specifications.

set_link_lib_map

The `set_link_lib_map` command uses a file as input. Each line of the file contains a set of instances, a colon separator character (:) with spaces on both sides, and a `link_path` specification to be used for the instances. For example,

```
% cat ./link_paths.txt
ucore1 ucore2      : * lib2.db
ucore1/usubblk     : * lib3.db
ucore1/U12 ucore1/U34 : * lib4.db
```

The `set_link_lib_map` takes this file as input before the design is linked. For example,

```
set_app_var link_path {* lib1.db}
set_link_lib_map ./link_paths.txt
```

```
read_verilog my_design.v
link_design MY_DESIGN
```

The listed instances can be hierarchical cells (blocks) or leaf cells. By default, the instance paths are relative to the top-level design. However, the `set_link_lib_map` command provides two options that allow files to be applied relative to cell instances and design references:

```
set_link_lib_map -instance {ucore1/usubblk1} ./link_paths_subcore.txt
set_link_lib_map -reference {subblk}      ./link_paths_subcore.txt
```

The `-instance` option applies the file relative to the specified cell instance. The `-reference` option applies the file relative to *all* cell instances of the specified design name. These options allow long instance names to be shortened in input files. Both options accept a single name.

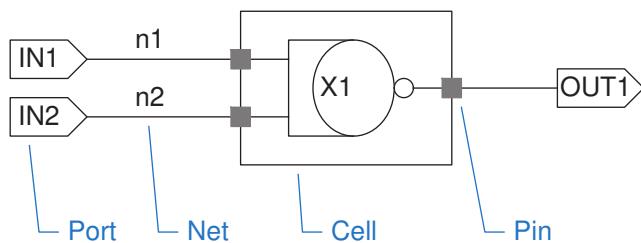
The precedence rules are as follows:

- Specifications propagate downward into hierarchy.
- Lower-level specifications take precedence over higher-level specifications.
- Leaf cell specifications take precedence over block specifications.
- `-instance` specifications take precedence over `-reference` specifications.

Working With Design Objects

Designs are hierarchical entities composed of objects such as cells, ports, and nets.

Figure 22 Typical design objects



In the PrimeTime tool, a design contains the objects in the following table.

Table 7 Design objects in PrimeTime

Object class	Description	Command to create object collection
cell	Instance in the design; can be a hierarchical block or primitive library cells	get_cells
clock	Clock	get_clocks
design	Design	get_designs
lib	Library	get_libs
lib_cell	Cell in a logic library	get_lib_cells
lib_pin	Pin on a library cell	get_lib_pins
lib_timing_arc	Timing arc on a library cell	get_lib_timing_arcs
net	Net in the current design	get_nets
path_group	Group of paths for cost-function calculations and timing reports	get_path_groups
pin	Pin of a lower-level cell in the design; can be input, output, or inout (bidirectional)	get_pins
port	Port of the current design; can be input, output, or inout (bidirectional)	get_ports
timing_arc	Timing arc	get_timing_arcs
timing_path	Timing path	get_timing_paths

To constrain the design, perform detailed timing analysis, and locate the source of timing problems, you need to access the design objects. You can do this by creating collections of objects with the appropriate “get” command. For example, the `get_ports` command creates a collection of ports.

You can nest the result of a “get” command within another command that operates on the objects. For example:

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

For more information, see the “Searching for Design Objects” section in *Using Tcl With Synopsys Tools*, available on [SolvNetPlus](#).

Setting the Current Design and Current Instance

The current design and current instance define the focus of many PrimeTime commands. To set or return the current design or instance, use the following commands.

Table 8 Current design objects

Object	Description	Command to set or return current object
Current design	The top level of the current design; most objects are referenced relative to the current design	<code>current_design</code>
Current instance	The instance (hierarchical cell) that is the current scope within the current design; you traverse the hierarchy by changing the current instance	<code>current_instance</code>

Working With Attributes

An attribute is a string or value associated with an object that carries some information about that object. You can write programs in Tcl to get attribute information from the design database and generate custom reports on the design.

PrimeTime provides the following commands for setting, reporting, listing, and creating attributes.

Table 9 Commands for Working With Attributes

Attribute command	Description
<code>define_user_attribute</code>	Creates a new attribute for one or more object classes
<code>get_attribute</code>	Retrieves the value of any attribute from a single object
<code>list_attributes</code>	Shows the attributes defined for each object class or a specified object class; optionally shows application attributes
<code>remove_user_attribute</code>	Removes a user-defined attribute from one or more objects
<code>report_attribute</code>	Displays the value of all attributes on one or more objects; optionally shows application attributes
<code>set_user_attribute</code>	Sets a user-defined attribute on one or more objects

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
```

Saving Attributes

The `save_session` command does not save attributes. To save attributes that you applied during a session, create a script in the pt_shell, dc_shell, or dctcl format by using the `write_script` command. When you restore the PrimeTime session, you can use the script to re-create the attributes on the design.

Saving and Restoring Sessions

You can use the `save_session` and `restore_session` commands to save the design data from one session, then restore it into a later session for continued use.

A *session* is a tool-created directory that contains 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 design database archiving tool. 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)

Saving Sessions

To save a session, use the `save_session` command and specify the directory name for the session (required):

```
pt_shell> save_session my_session
Saving environmental constraints.....
Saving netlist information.....
Saving miscellaneous application information.....
Saving timing information.....
Saving variable information.....
Information: Executed with 4 workers
Information: At least 6 MB of free disk space in pt_tmp_dir will be
            required to restore this session. (SR-044)
```

If the directory already exists, the tool issues an information message and overwrites the directory contents:

```
pt_shell> save_session my_session
Information: Cleaning and overwriting all data in the existing directory
            '/remote/cael1180/chrispy/hist/my'. (SR-002)
Saving environmental constraints.....
...
```

Saving Version-Compatible Sessions

By default, the `save_session` command saves a version-specific session that can only be restored into the same tool version.

However, the `-version` option allows you to save a session that is version-specific (the default), version-compatible, or both:

- `save_session -version compatible`

Saves a version-compatible session that can be restored into the same or *later* version of the PrimeTime tool. A timing update is always needed when restoring the session.

- `save_session -version specific`

Saves a version-specific session. A timing update is not needed when restoring the session. Although this is the default, the keyword provides an explicit way to specify the default.

- `save_session -version both`

Saves a combined version-compatible and version-specific session. A timing update is only needed when restoring the session into a different version.

To identify a saved session as version-compatible, the tool includes the following note in the README file inside the session directory:

```
***** This session is saved in version compatible mode and
it can be restored in newer versions of PrimeTime *****
```

This session format cannot store the in-memory data from a timing update; thus, a timing update is needed to perform analysis after restoring the session. This can result in slight QoR differences, particularly when restoring into a newer version of the tool.

Similarly, in-memory physical data for ECO fixing is not stored and will be reloaded when the `check_eco` command or a `fix_eco_*` command is run.

A typical version-compatible session is restored as follows:

```
# restore the session
restore_session session_name

# (optional) apply any new settings not stored in the session
set_app_var ...
set_app_var ...

# recompute timing
update_timing

# perform any additional analysis operations here (reporting, etc.)
...
```

In an ECO flow, the preceding script is continued as follows:

```
# (optional) apply any new settings not stored in the session
set_eco_options ...

# ensure everything is ready to go
check_eco

# perform ECO fixing
fix_eco_*

# perform any additional ECO operations here
```

Including ECO Physical Data in Saved Sessions

When you save a version-compatible session in a physically aware ECO flow, you can include a copy of the physical data files by using the `physical_data` keyword with the `-include` option:

```
pt_shell> save_session my_session \
           -version compatible \
           -include {physical_data}
```

As described in [Physically Aware ECO](#), the physical data files are configured by the `set_eco_options` command. This includes:

- LEF technology and cell definition files
- DEF design data files
- Physical library constraints and rules
- Via ladder definitions and associations

When you use this feature to include physical data in a saved session file, the ECO physical data directory is copied into the saved session directory. (By default, the ECO configuration data is still stored, but it points to files outside the session directory.)

Note:

The IC Compiler II reference libraries, specified with the `set_eco_option -physical_icc2_lib` option, are not copied to the session directory. Their original location is stored in the `session_dir/eco_config` file. If needed, edit these paths before restoring the session.

This feature is supported only for version-compatible sessions.

Restoring Sessions

The `restore_session` command reads a directory that was written by the `save_session` command, restoring a PrimeTime session to the same state as the session where the `save_session` command was issued.

By default, PrimeTime requires that you restore into the same PrimeTime version as used for the original session. Locate the version used to save the session from the README file located in the session directory.

If you use the feature described in [Saving Version-Compatible Sessions](#), then the session can be restored into the same or later PrimeTime version. In this case, the tool reports the original tool version when the session is restored:

```
pt_shell> restore_session my_vc_session
...
Information: Restoring the version compatible session saved with version
'Q-2019.12-SP4'. (SR-048)
```

If there is existing design or library data in memory, it is removed before restoring the session.

The command first tries to grab all licenses that were present at the time of the `save_session` command. If this fails, restoring the session fails.

The restore directory contains a `lib_map` ASCII file name. This file contains the path and leaf name for each logic library needed to restore the session. If necessary, you can edit the path names to give the `restore_session` command the updated locations of the needed logic library files. The leaf names of those files cannot be changed. The logic libraries are assumed to be the same as those used when the `save_session` command was issued.

Restoring Version-Compatible Sessions

PrimeTime saves two types of settings in a session:

- Application variables – the default and user-specified settings that are reported by the `report_app_var` command
- Program options – internal program settings that affect the analysis and sometimes changes across releases, but are not reported by the `report_app_var` command

When you restore a version-compatible session into a different version, the `restore_session` command allows you to control how the internal program options are handled:

- `restore_session -program_options save_version`

This setting uses the program options stored in the saved session, which helps the analysis in the new version match the original session. This is the default.

- `restore_session -program_options restore_version`

This option uses the program option defaults of the current (restoring) version, which allows updated program options in the current release to be used.

Creating Test Cases

In some cases, you might want to extract part of the design logic into a standalone runnable test case. For example, you might want to experiment with settings or constraints to see how the analysis is affected. Or, you might need to send a test case to Synopsys Support for further investigation.

You can use the `create testcase` command to do this. It creates a directory with the specified portion of the design, a constraints file which applies to that portion of the design, any detailed parasitics used, and a run script for the test case.

The syntax of the `create testcase` command is as follows:

```
Usage:
  create testcase      # Create a testcase for specified design objects
  [-paths list]        (create testcase for the given paths)
  [-ports list]        (create testcase for the given ports)
  [-endpoints list]    (create testcase for the given endpoints)
  [-cells list]        (create testcase for the given cells)
  [-directory string]  (Output directory name)
  [-include_libs]      (Include libraries)
  [-mim]               (Include multi-instance modules)
```

These options are described in the following sections.

Controlling What Design Logic to Keep

The `create testcase` command keeps only the design logic that you specify. [Table 10](#) lists the different ways you can specify what to keep.

Table 10 Specifying the Design Logic for `create testcase` to Keep

Option name	Design logic that is kept	Clock paths to registers included?
<code>-paths path_collection</code>	All cells along the timing paths, including startpoints and endpoints	Yes
<code>-ports port_list</code>	Fanout logic from input ports, including endpoints Fanin logic to output ports, including startpoints	Yes
<code>-endpoints pin_port_list</code>	Fanin logic to endpoint pins/ports, including startpoints	Yes
<code>-cells cell_list</code>	The specified leaf cells The nets driven by the cells The leaf cells and ports connected to the driven nets	No

A single design logic option (`-paths`, `-ports`, `-endpoints`, or `-cells`) is required. Multiple options cannot be combined.

In a PrimeTime SI analysis, the aggressor stages with annotated aggressor timing windows are included.

A timing-updated design is not required when using the `-ports` or `-endpoints` option. In this case, the aggressors use zero input pin slews and infinite windows. This is useful for reproducing crashes in the initial timing update.

Including the Logic Libraries

By default, the `create_testcase` command creates a test case that references the required logic libraries in their current location. This avoids using unnecessary disk space for test cases that are used in the current design environment.

If the test case must run in another design environment, specify the `-include_libs` option to copy the required logic libraries into the test case directory:

```
pt_shell> create testcase ... -include_libs
```

Specifying the Output Directory

By default, the test case is created in a directory named after the current design:

```
pt_shell> create testcase ...
...
Testcase is generated in the TOP directory.
1
```

To specify a particular test case output directory, use the `-directory` option:

```
pt_shell> create testcase ... -directory ./my_testcase_FLAT
...
Testcase is generated in the my_testcase_FLAT directory.
1
```

Creating a Multiple-Instance Module Test Case

For designs with multiple-instance modules (MIMs), the `create testcase` command provides a feature to keep the same logic in parallel across the MIM instances.

When you keep logic to an endpoint inside an MIM block, you can specify the `-mim` option to keep the same endpoint-fanin logic across all instances of that MIM block:

```
pt_shell> create testcase \
          -endpoints $endpoint_in_one_MIM \
          -mim
```

This feature requires that the `-endpoints` option be used.

Limitations

The `create testcase` command does not support the following tool features:

- HyperScale data (block models or context data)
- SMVA/DVFS analysis

6

Constraining the Design

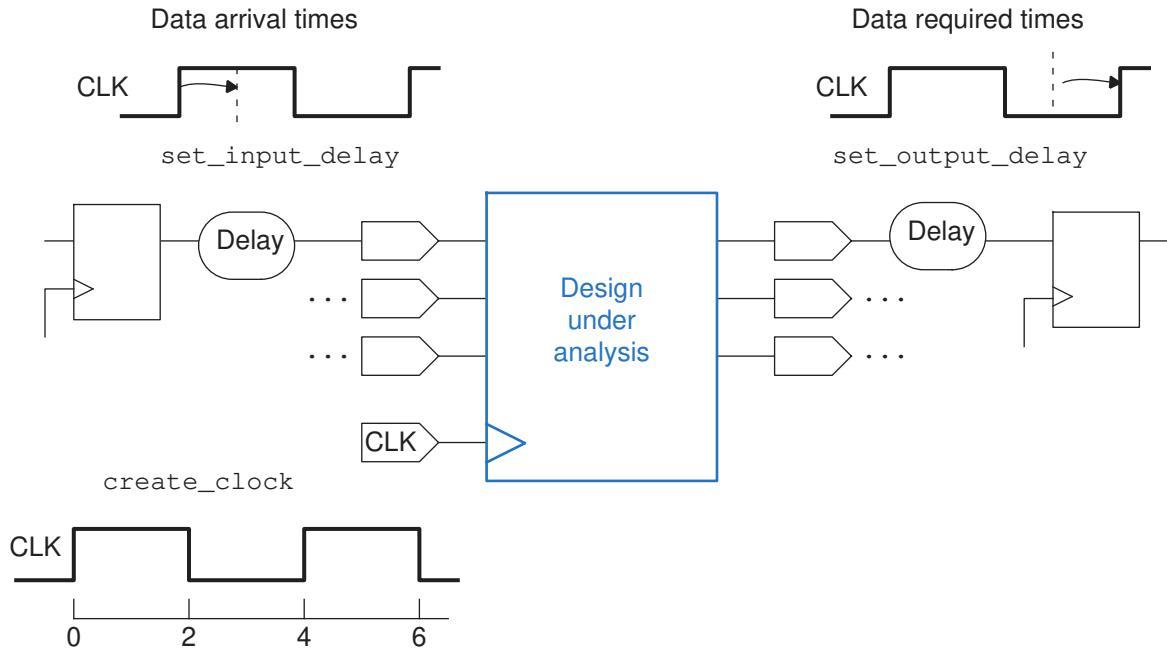
To learn about constraining the inputs and outputs of the design, see these topics:

- [Timing Constraints](#)
 - [Input Delays](#)
 - [Output Delays](#)
 - [Drive Characteristics at Input Ports](#)
 - [Port Capacitance](#)
 - [Wire Load Models](#)
 - [Slew Propagation](#)
 - [Design Rule Constraints](#)
 - [Ideal Networks](#)
 - [Checking the Constraints](#)
-

Timing Constraints

Before you begin timing analysis, you need to specify the timing constraints on the design. A timing constraint (also called a *timing assertion*) limits the allowed range of time that a signal can arrive at a device input or be valid at a device output.

Figure 23 Design-level timing constraints



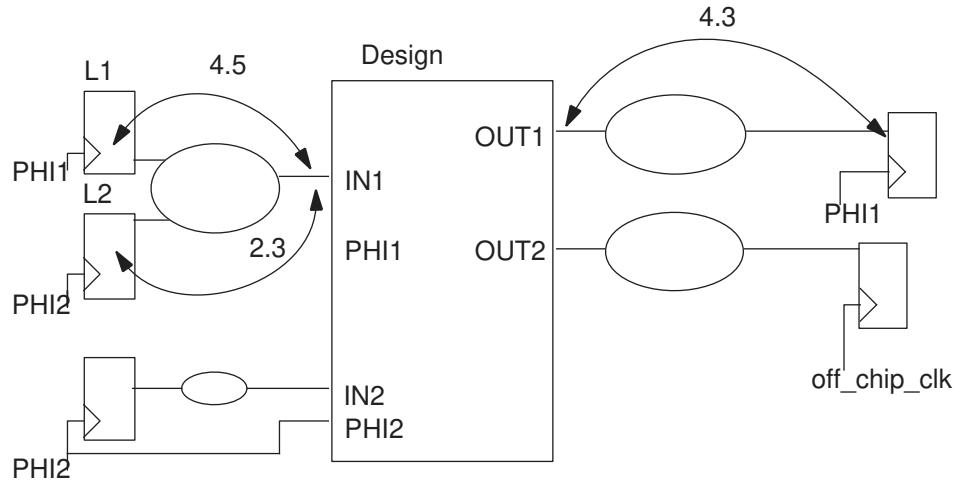
Input Delays

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. The following example shows the external path for the L1 clock port to port IN1.

Figure 24 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.

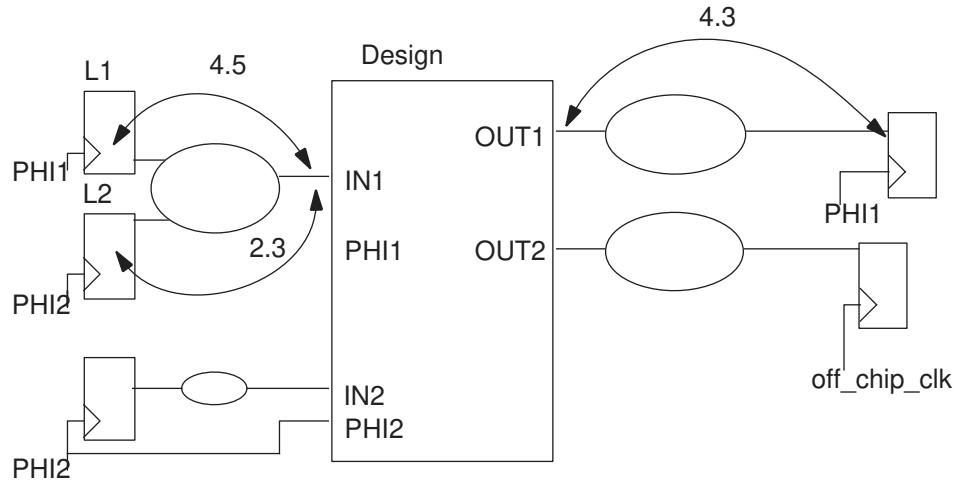
Output Delays

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:

- $\text{Maximum_output_delay} = \text{length_of_longest_path_to_register_data_pin} + \text{setup_time_of_the_register}$
- $\text{Minimum_output_delay} = \text{length_of_shortest_path_to_register_data_pin} - \text{hold_time}$

Figure 25 Two-phase clocking example for setting port delays



In the preceding example, this command sets an output delay of 4.3 relative to the rising edge of clock PHI1 on port OUT1:

```
pt_shell> set_output_delay 4.3 -clock PHI1 {OUT1}
```

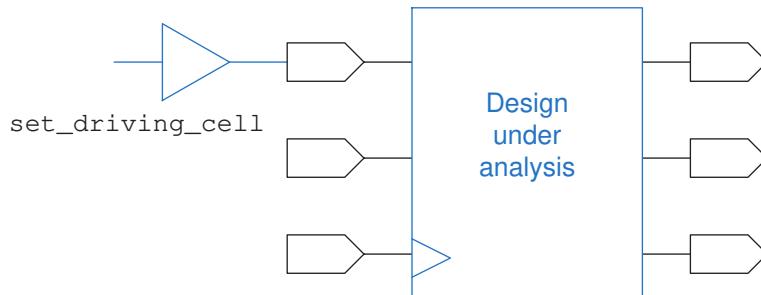
To show output delays associated with ports, use the `report_port -output_delay` command.

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.

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. 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 26 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

To remove drive information from ports, use the following commands.

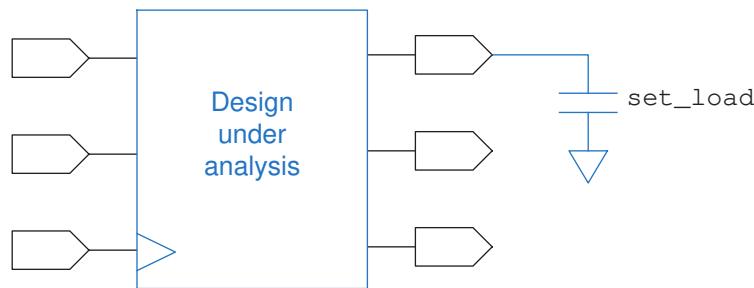
Command	Description
<code>remove_driving_cell</code>	Removes driving cell information from specified ports
<code>reset_design</code>	Removes drive data and all user-specified data, such as clocks, input and output delays

Command	Description
set_drive 0.0	Removes drive resistance
set_input_transition 0.0	Removes input transition

Port Capacitance

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

Figure 27 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:

```
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.

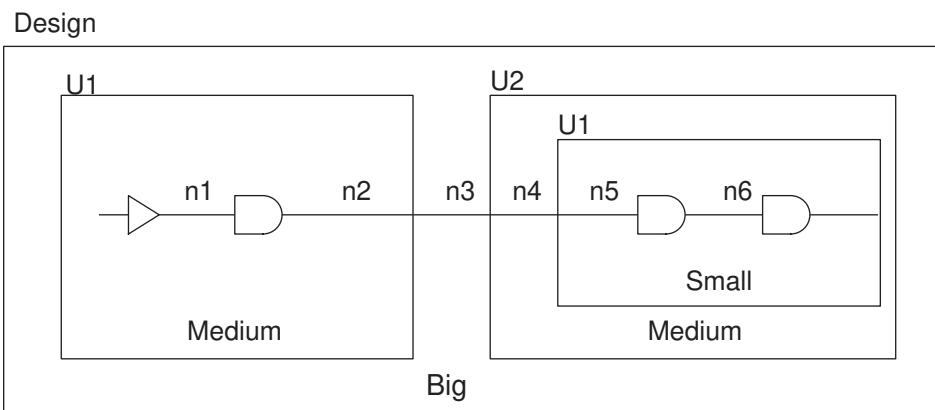
To set the wire load mode to `enclosed` on the current design, enter

```
pt_shell> set_wire_load_mode enclosed
```

This design example sets the following wire load models:

```
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 28 Design example for wire load mode settings



The following table lists the resulting wire load modes and models that apply to the nets in the design example.

Table 11 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

Table 11 Wire load models (Continued)

Wire load setting	Wire load model	Applies to these nets
segmented	Medium	U1/n1
	Small	U2/U1/n6
	Big	n3
	Medium	U1/n1, U1/n2, U2/n4
	Small	U2/U1/n5, U2/U1/n6

Reporting Wire Load Models

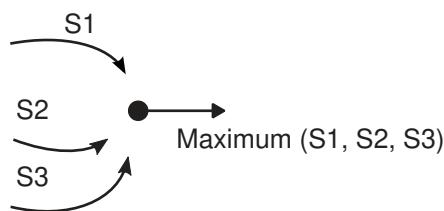
To obtain wire load reports from PrimeTime, use these commands:

- `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. 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 29 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.

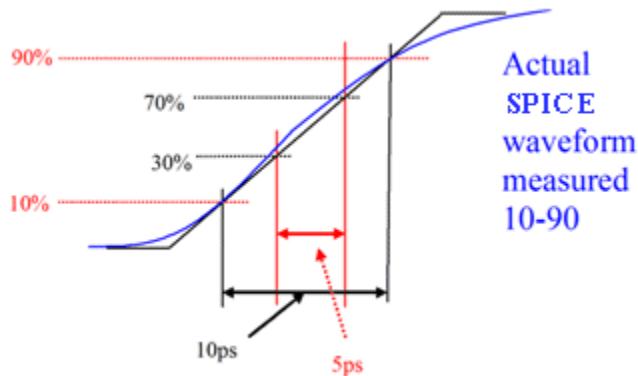
Maximum Transition Time

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. In the following figure,

- 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 $12.5 \text{ ps} = 10 * (100-0) / (90-10)$
- Representation as single float with slew threshold 30-70 is $5 \text{ ps} = 12.5 * (70-30) / (100-0)$

Figure 30 SPICE waveform



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.

Multiple Threshold and Derating Values in Libraries

Liberty allows an arbitrary slew threshold to minimize the error due to slew linearization for various process technologies. Therefore, whenever slew information from a 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.

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 [Specifying the Maximum Transition Constraint](#).

Specifying the Maximum Transition Constraint

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.

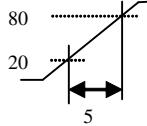
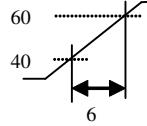
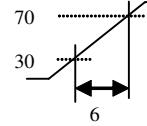
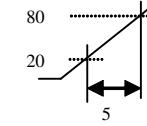
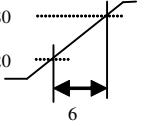
By default, during constraint checking on a pin or port, the most restrictive constraint specified on a design, pin, port, clock (if the pin or port is in that clock domain), or library is considered. This is also true where multiple clocks launch the same path.

To change the default behavior, set the `timing_enable_max_slew_precedence` to `true`. In that case, where the constraint is set determines the priority, irrespective of the relative constraint values, as follows:

1. Pin or port level
2. Library-derived constraint
3. Clock or design level

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 12 Converting slews between different slew thresholds and derating factors

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)
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	$10 * (0.6 / (0.6 - 0.4)) * ((0.8 - 0.3) / (0.8 - 0.2) / 0.5)$	$10 * (0.6 / (0.7 - 0.3)) * ((0.8 - 0.2) / 0.5)$	$10 * 0.5 / (0.8 - 0.2) * ((0.8 - 0.2) / 0.5)$	$10 * 0.6 / (0.7 - 0.3) * ((0.8 - 0.2) / 0.5)$
PrimeTime or Liberty value (slew or limit)	10 ps	-	-	-	-
Main library limit expressed in local threshold and derate	10 ps	$10 * (0.5 / (0.8 - 0.2)) * ((0.6 - 0.4) / 0.6)$	$10 * (0.5 / (0.8 - 0.2)) * ((0.7 - 0.3) / 0.6)$	$10 * (0.5 / (0.8 - 0.2)) * ((0.8 - 0.2) / 0.5)$	$10 * (0.5 / (0.8 - 0.2)) * ((0.7 - 0.3) / 0.6)$

To apply a global derating factor to `max_transition` values, set the `timing_max_transition_derate` variable.

To see the capacitance constraint evaluations, run the `report_constraint -max_capacitance` command. To see port capacitance limits, run the `report_port_design_rule` command. To see the default capacitance settings for the current design, run the `report_design` command. To remove user-specified capacitance limits, run the `remove_max_capacitance` command.

Evaluating the 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
FF1/D	0.6000	0.4000	0.2000

Minimum Capacitance

To specify a minimum capacitance limit on specific ports or on the entire design, run the `set_min_capacitance` command. 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.

The `set_min_capacitance` command applies the `min_capacitance` attribute (a design rule constraint) on the specified objects. Capacitance constraint checks are only applicable for output pins. During a constraint check, the most restrictive constraint is considered.

To see the capacitance constraint evaluations, run the `report_constraint -min_capacitance` command. To see port capacitance limits, run the `report_port -design_rule` command. To see the default capacitance settings for the current design, run the `report_design` command. To remove user-specified capacitance limits, run the `remove_min_capacitance` command.

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]
```

Maximum Capacitance

To specify a maximum capacitance limit, run the `set_max_capacitance` command. 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 a maximum 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_max_capacitance` command applies the `max_capacitance` attribute (a design rule constraint) on the specified objects. Capacitance constraint checks are only applicable for output pins. During a constraint check, the most restrictive constraint is considered.

To apply a global derating factor to `max_capacitance` values, set the `timing_max_capacitance_derate` variable.

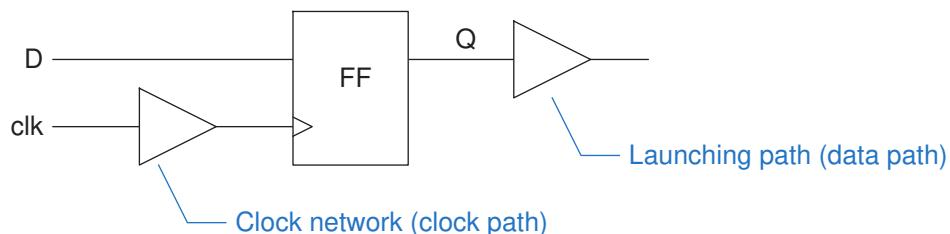
To see the capacitance constraint evaluations, run the `report_constraint -max_capacitance` command. To see port capacitance limits, run the `report_port_design_rule` command. To see the default capacitance settings for the current design, run the `report_design` command. To remove user-specified capacitance limits, run the `remove_max_capacitance` command.

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.

Figure 31 Restrict constraints



If the list of objects is a clock list, you can optionally restrict the constraint to only certain path types or transition types by using the `-clock_path`, `-data_path`, `-rise`, or `-fall` options.

If a pin (port) is subject to different maximum capacitance constraints specified on the design, library, clock (and port), the most restrictive constraint has priority. The tool reports the slack at a pin as the difference between the most restrictive constraint and the total capacitance at the pin (port).

This feature is applicable to generated clocks. To report the constraint violations, use the `report_constraint` command.

Frequency-Based Maximum Capacitance Checks

A library can contain lookup tables that specify frequency-based `max_capacitance` values for each driver pin. The pin frequency is the maximum frequency of all clocks that launch signals arriving at the pin.

By default, the tool considers the `max_capacitance` values from frequency-based lookup tables if available. Design rule constraint (DRC) checking for attribute reporting, the `report_constraint` command, extracted timing model (ETM) generation, and ECO fixing use the smallest of the following `max_capacitance` values:

- The `max_capacitance` value from the lookup table in the library
- The library-cell `max_capacitance` value
- All user-specified `max_capacitance` values defined at the pin, port, clock, and design levels

To completely ignore any `max_capacitance` values from the lookup table, set the `timing_enable_library_max_cap_lookup_table` variable to `false`.

To give precedence to the `max_capacitance` value from the lookup table over the library-cell `max_capacitance` value, set the `timing_library_max_cap_from_lookup_table` variable to `true`.

The following table shows how the variable settings affect the selection of the `max_capacitance` values.

Table 13 Variables that affect the `max_capacitance` value selection

	<code>timing_library_max_cap_from_lookup_table = false</code> (the default)	<code>timing_library_max_cap_from_lookup_table = true</code>
<code>timing_enable_library_max_cap_lookup_table = true</code> (default)	DRC checking uses the minimum of <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 	DRC checking uses the minimum of <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; if this value does not exist, the cell-level <code>max_capacitance</code> value defined in the library is considered instead
<code>timing_enable_library_max_cap_lookup_table = false</code>	DRC checking uses the minimum of <ul style="list-style-type: none"> • All user-specified <code>max_capacitance</code> values defined at the pin, port, clock, and design levels • The cell-level <code>max_capacitance</code> value defined in the library 	DRC checking uses the minimum of <ul style="list-style-type: none"> • All user-specified <code>max_capacitance</code> values defined at the pin, port, clock, and design levels • The cell-level <code>max_capacitance</code> value defined in the library

Maximum Capacitance Checking With Case Analysis

By default, maximum capacitance checking is not performed on pins in the path of constant propagation. To enable maximum capacitance checking on the constant propagation path, set the `timing_enable_max_capacitance_set_case_analysis` variable to `true`.

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](#)
 - [Reporting of Ideal Clocks in Full Clock Expanded Mode](#)
-

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
...
Source ports      Latency                      Transition
and pins          Rise            Fall           Rise            Fall
                  min    max        min    max       min    max     min    max
-----
middle/p_in       --    --    --    --
                           Rise            Fall           Rise            Fall
                           min    max        min    max       min    max
-----
Internal pins      Latency                      Transition
with ideal timing  Rise            Fall           Rise            Fall
                  min    max        min    max       min    max
```

n3_i/B	5.62	5.62	--	--	--	--	--	--
n0_i/z	1.34	--	1.34	--	--	--	--	--
<hr/>								
Boundary pins	Latency		Transition					
	Rise	Fall	min	max	Rise	Fall	min	max
	min	max	min	max	min	max	min	max
<hr/>								
middle/p_out	--	--	--	--	--	--	--	--
<hr/>								
Nets								
p_in								
p_out								
n2								
n1								
n0								
<hr/>								
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.

Reporting of Ideal Clocks in Full Clock Expanded Mode

By default, when you use the `report_timing -path_type full_clock_expanded` command, the report does not show the clock path through an ideal clock network. Instead, it shows a single line with the ideal clock latency at the clock arrival point.

You might want to know the topological path through the ideal clock network. For example, you might want to find out which ideal clock path is reaching a register where you did not expect it.

To report clock paths through ideal clock networks, set the following control variable:

```
pt_shell> set_app_var timing_report_trace_ideal_clocks true
true
pt_shell> report_timing -path_type full_clock_expanded...
...
          Point                               Incr      Path
-----
clock CLK0' (rise edge)                  4.00      4.00
clock source latency                   0.00      4.00
CLK0 (in)                            0.00 &    4.00 f
C_D950/C_CLK/U142/z (AN2P)           0.61 &    4.61 f
C_D950/C_CLK/C_DCLKA_GEN_DMA_CLK/I16/z (DLY6) 0.57 &    5.18 f
C_D950/C_CLK/C_DCLKA_GEN_DMA_CLK/I20/z (IV)   0.08 &    5.26 r
... full clock path ...
C_DMA/i0/i2/U5xI_7_00/z (ITSENX4)       0.68 &    8.43 r
C_DMA/i0/i2/U5xCBGF_GCKB_EN/G0 (CBUF01X8) 0.35 &    8.78 r
C_DMA/i0/i2/GCKB_EN_I/U5xCBGF_GCKB_EN_L1/z (BF2T8) 0.02 &    8.80 r
propagated network delay removal        -4.80      4.00
clock network delay (ideal)            1.00      5.00
C_DMA/i0/i9/IOE2_reg/I6/CP (FD1S)       0.00 &    5.00 r
C_DMA/i0/i9/IOE2_reg/I6/QN (FD1S)       0.37 &    5.37 f
... full data path ...
```

With the `timing_report_trace_ideal_clocks` variable set to `true`, the report shows the full ideal clock path and potential incremental delay along the path. At the clock arrival point, the report applies an adjustment to negate the propagated delay calculation and obtain the ideal clock edge time. It then applies the ideal clock network delay.

This option similarly controls reporting of paths by the `report_clock_timing -verbose` and `report_min_pulse_width -path_type full_clock_expanded` commands.

Checking the Constraints

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.

Design Reports

Before running a full timing analysis, you can check the design with the commands in the following table. After running timing analysis, these commands can also help you to debug violations in the design.

Table 14 Design reporting commands

Command	Description
report_bus	Reports information about buses (pins or ports) in the current instance or current design.
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_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_design	Lists the attributes of the design, including the chosen operating conditions, wire load information, design rules, and derating factors.
report_disable_timing	Reports disabled timing arcs.
report_hierarchy	Generates a hierarchical list of submodules and leaf-level cells in the current design.
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_net	Lists the nets and shows net information such as the fanin, fanout, capacitance, wire resistance, number of pins, net attributes, and connections.
report_path_group	Reports the path groups. PrimeTime organizes timing paths into groups based on the conditions at the path endpoints.
report_port	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.
report_reference	Lists hierarchical references, showing for each submodule the reference name, unit area, number of occurrences, total area, and cell attributes.
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.

Table 14 Design reporting commands (Continued)

Command	Description
report_units	Shows the units of measurement for current, capacitance, resistance, time, and voltage used in the current design.
report_wire_load	Shows the wire load models set on the current design or on specified cells.

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. This command also 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 32 Multiple clock fanin

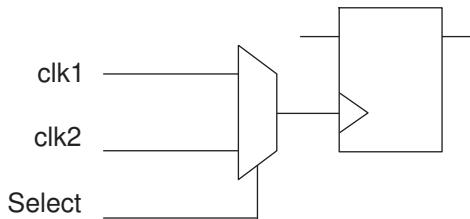


Figure 33 Self-looping latch

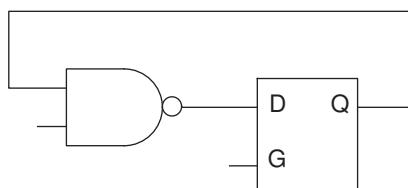
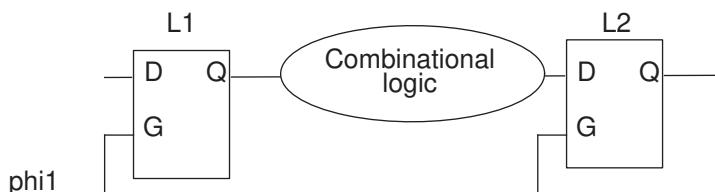


Figure 34 Latch fanout to another latch of the same clock



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.

Detailed Constraint Checking and Debugging

To report incorrect constraints or other potential problems, use the detailed constraint checking and debugging capabilities of the tool. This constraint checking capability 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 constraint checking from the PrimeTime shell,

1. (Optional) Specify the constraint checking setup in the `.synopsys_gca.setup` file.
2. Specify the current design by using the `current_design` command. For example:

```
pt_shell> current_design top
```

3. (Optional) Specify a constraint checking rule file that contains user-defined violation waivers and custom rules by setting the `gca_setup_file` variable. For example:

```
pt_shell> set_app_var gca_setup_file ./top_rules.tcl
```

4. Perform constraint analysis by using the `check_constraints` command. For example:

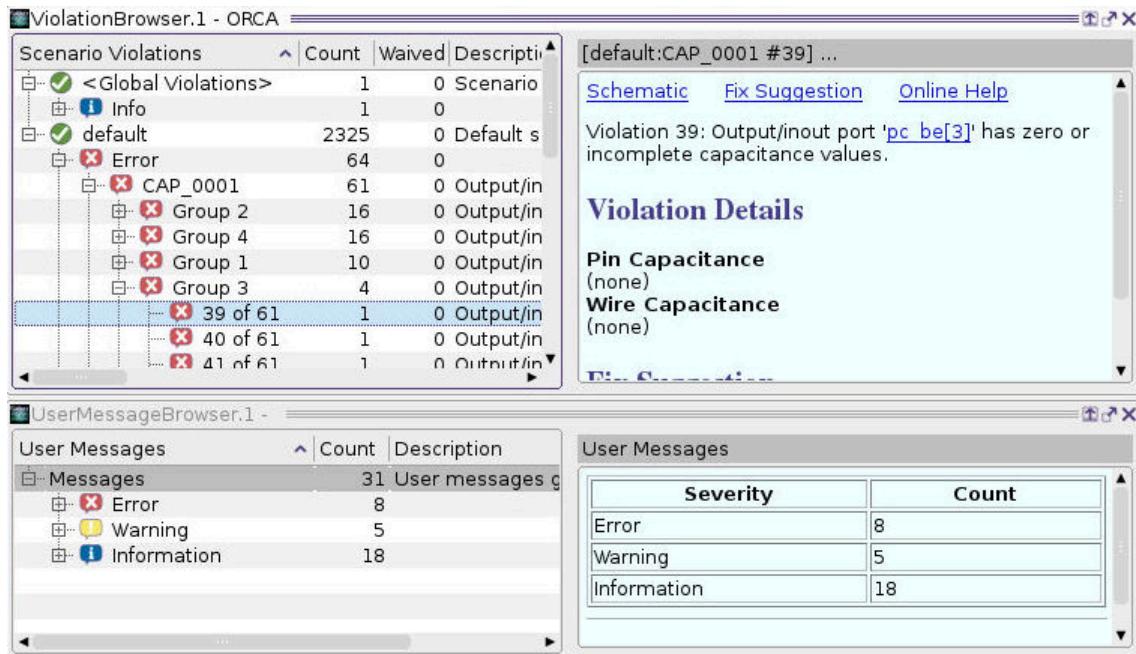
```
pt_shell> check_constraints
```

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
...
*****
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
```

To view the constraint checking results in the GUI, choose Constraints > View Constraint Checking Results.

Figure 35 Constraint Checking Results Displayed in the GUI



Alternatively, you can perform constraint analysis in a separate constraint checking shell, ptc_shell. For details, see [Constraint Consistency](#).

7

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](#)
 - [Specifying 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 [Timing 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 following waveform.

Figure 36 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

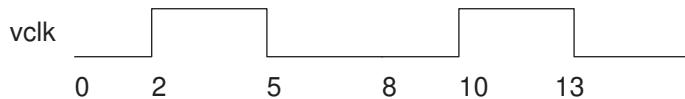
You can use the `create_clock` command to define virtual clocks for signals that interface to external (off-chip) clocked devices. A virtual clock has no actual source in the current design, but you can use it for setting input or output delays.

To create a virtual clock named vclk, enter

```
pt_shell> create_clock -period 8 -name vclk -waveform {2 5}
```

The vclk clock has the following waveform.

Figure 37 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 that 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

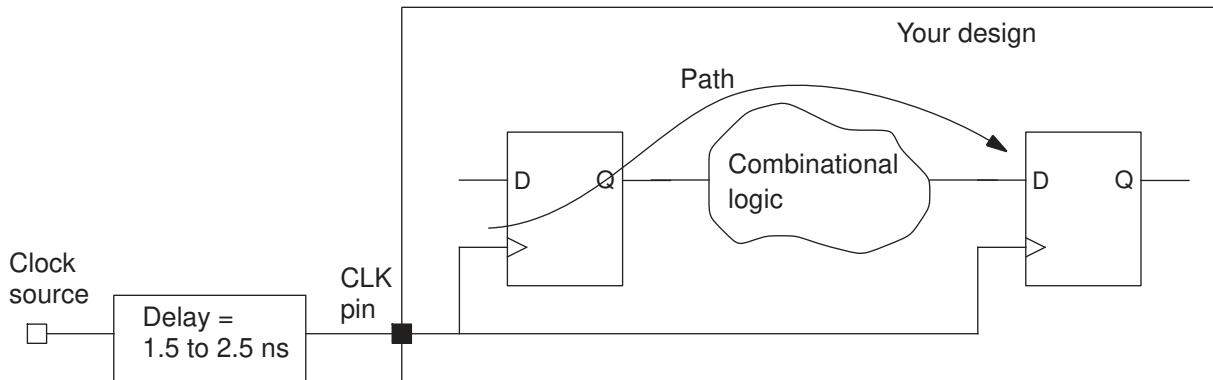
You can specify clock source latency for ideal or propagated clocks. Source latency is the latency from the ideal waveform to the source pin or port. The latency at a register clock pin is the sum of clock source latency and clock network latency.

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.

Figure 38 External source latency

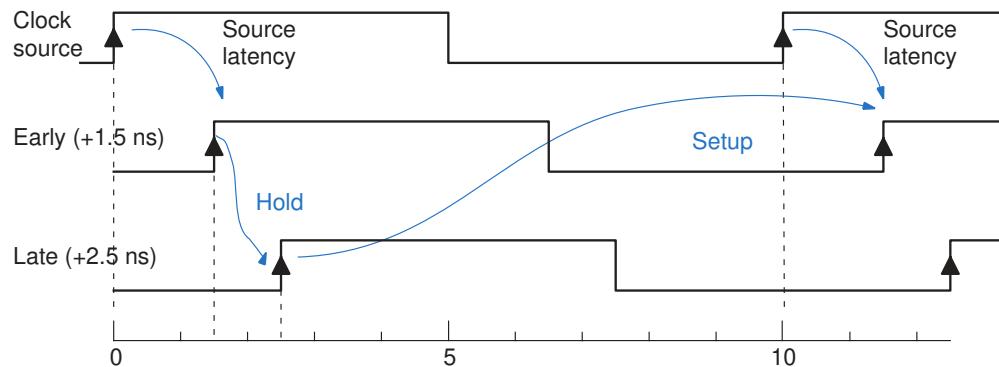


To specify this type of source latency, you can use commands such as the following:

```
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. The following figure 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 39 Early and 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.

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](#).

Setting Clock Uncertainty

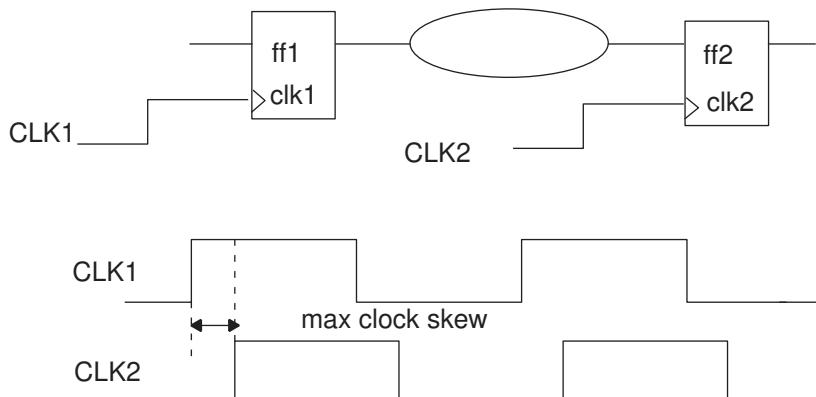
You can model the expected uncertainty (skew) for a prelayout design with setup or hold and rise or fall uncertainty values. PrimeTime subtracts a setup uncertainty value from the data required time when it checks setup time (maximum paths). PrimeTime adds a hold uncertainty value to the data required 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.

Figure 40 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]
```

If you set simple uncertainty on a clock port or pin, the uncertainty applies to capturing latches whose clock pins are in the transitive fanout of the specified object. If different uncertainty values are propagated from multiple ports or pins to inputs of a clock MUX, the worst uncertainty setting applies to the entire fanout of the MUX, even when the associated clock is disabled at the MUX. To ensure usage of the proper uncertainty value in this situation, set the uncertainty on the clock object rather than on a clock port or clock pin.

To remove clock uncertainty information from clocks, ports, pins, or cells, or between specified clocks, use the `remove_clock_uncertainty` command removes uncertainty.

Setting the Clock Jitter

To set clock jitter on a master clock, run the `set_clock_jitter` command. This command sets the same clock jitter properties on all clocks generated from the specified master clock. If you do not specify a master clock, the command sets the jitter on all clocks.

Removing the Clock Jitter

To remove the clock jitter, run the `remove_clock_jitter` command. This command automatically removes the clock jitter properties from the generated clocks as well. If you do not specify a master clock, the command removes the jitter from all clocks.

Reporting the Clock Jitter

To report the clock jitter, run the `report_clock_jitter` command. The report shows the cycle jitter, duty-cycle jitter, and the master clock with jitter that is used by the generated clocks:

```
pt_shell> report_clock_jitter
*****
Report : clock jitter
...
*****
Clock      cycle jitter    duty-cycle jitter    master clock with jitter
-----
mclk        2                  3                  mclk
gclk        2                  3                  mclk
```

To see the effects of clock jitter, run the `report_timing` command:

```
pt_shell> set_clock_uncertainty 0.3 CLK -rise
pt_shell> set_clock_jitter -cycle 0.4 CLK
pt_shell> report_timing -from ff1 -to ff2 -path_type full_clock
```

Point	Incr	Path
clock CLK (rise edge)	0.00	0.00
clock network delay (ideal)	2.00	2.00
ff1/clk	0.00	2.00 r
data arrival time		2.00
...		
clock CLK (rise edge)	10.00	10.00
clock network delay (ideal)	1.75	11.75
clock uncertainty	-0.30	11.35
clock jitter	-0.40	10.95
ff2/clk		10.95 r
library setup time	-0.05	10.90
data required time		10.90
data required time		10.90

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.

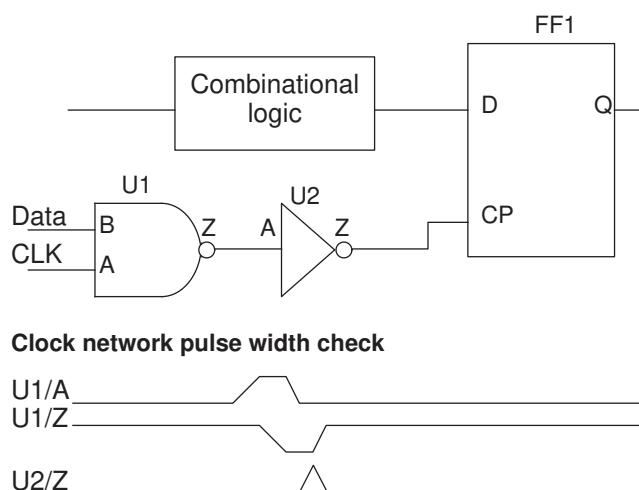
To remove the estimated clock pin transition time, use the `remove_clock_transition` command.

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. PrimeTime can check the entire clock network to ensure that the pulse does not shrink below a certain threshold.

Figure 41 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.

By default, the clock uncertainty specified by the `set_clock_uncertainty` command applies to all types of timing constraints, including pulse width checking. You can optionally specify whether to apply clock uncertainty to pulse width checking, and if so, whether to apply only setup or only hold uncertainty, or both. To use this option, set the following variable:

```
set_app_var timing_include_uncertainty_for_pulse_checks \
    none | setup_only | hold_only | setup_hold
```

The default is `setup_hold`, which applies both setup and hold uncertainty to pulse width checking.

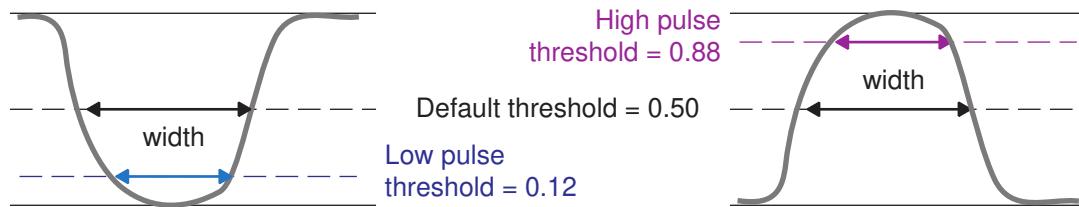
Pulse Width Measurement Thresholds

By default, the tool measures the width of a pulse where the signal waveform crosses the voltage threshold at 50 percent of the supply voltage. You can set the measurement threshold to any fraction of the supply voltage between 0.0 and 1.0 for clock network pins.

For more conservative minimum pulse width checking, set the threshold to something less than 0.5 for low pulses and something more than 0.5 for high pulses. For example,

```
set delay_calc_waveform_analysis full_design      # Required for analysis
set timing_enable_min_pulse_width_waveform_adjustment true # Enable
set timing_min_pulse_width_low_pulse_threshold 0.12      # Low threshold
set timing_min_pulse_width_high_pulse_threshold 0.88      # High threshold
report_min_pulse_width -path_type short ...           # Perform check
```

Figure 42 Minimum Pulse Width Checking Thresholds



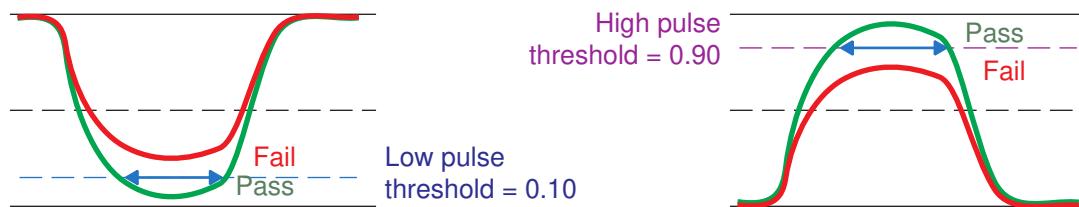
A report generated by the `report_min_pulse_width -path_type ...` command shows path adjustments for nondefault thresholds as “open edge threshold voltage adjustment” and “close edge threshold voltage adjustment.”

Voltage Swing Checking

You can perform voltage swing checking on clock network pins. This check determines whether a high pulse gets close enough to the power rail voltage, and whether a low pulse gets close enough to the ground rail voltage.

```
set delay_calc_waveform_analysis full_design # Required for analysis
set timing_enable_voltage_swing true          # Enable voltage swing check
set timing_voltage_swing_low_pulse_threshold 0.10 # Low pulse threshold
set timing_voltage_swing_high_pulse_threshold 0.90 # High pulse threshold
report_min_pulse_width -voltage_swing ...      # Perform check
```

Figure 43 Voltage Swing Checking Thresholds



Voltage swing checking is performed as a special case of a minimum pulse width check, with the minimum width set to 0.0 and the width measurement made at a specified fraction of the supply voltage. A pulse of any width that reaches or crosses the threshold passes the voltage swing check. A pulse that fails to cross the threshold is reported as a violation.

For a pulse that passes the check, the positive slack is reported as the width of the pulse, measured where the waveform crosses the threshold voltage.

Minimum Period Checking

Minimum period checking looks for violations of the `min_period` constraint for a cell pin, as defined in the cell library. This constraint requires the time between successive arrivals of clock edges of the same type (rising or falling) to be at least a specified value.

To perform minimum period checking, use one of these commands:

- `report_min_period [-verbose] ...`
- `report_constraint -min_period [-verbose] ...`
- `report_analysis_coverage -check_type min_period`

The following example shows how the tool calculates and reports the minimum period check with the `report_min_period` command.

```
pt_shell> report_min_period -path_type full_clock uff2/CP
...
Pin: uff2/CP
Related clock: clk
Check: sequential_clock_min_period

Point                                Incr      Path
-----
clock clk' (fall edge)                0.00     0.00
clock source latency                 0.20     0.20
clk (in)                            0.00     0.20 r
ckbf1/Y (IBUFFX2_HVT)               1.11 H   1.31 f
ckbf2/Y (INVX1_HVT)                 0.08     1.39 r
ckbf3/Y (INVX1_HVT)                 0.03     1.41 f
uff2/CP (FD1)                       -0.05    1.37 f
open edge clock latency           1.37

clock clk' (fall edge)                3.50     3.50
clock source latency                 0.20     3.70
clk (in)                            0.00     3.70 r
ckbf1/Y (IBUFFX2_HVT)               0.81 H   4.51 f
ckbf2/Y (INVX1_HVT)                 0.06     4.57 r
ckbf3/Y (INVX1_HVT)                 0.02     4.59 f
uff2/CP (FD1)                       -0.04    4.55 f
clock reconvergence pessimism       0.32     4.87
clock uncertainty                   -0.02    4.85
close edge clock latency          4.85

required min period              3.50
actual period                    3.48

slack (VIOLATED)                -0.02
```

In this example, the `report_min_period` command reports a minimum period violation at the CP pin of cell `uff2`. It shows two successive falling edges arriving at the pin with a spacing of less than 3.50 time units, the `min_period` constraint for that pin.

The “open edge clock latency” is the latest arrival time of the first edge, taking into account the maximum delays through the cells in the clock network.

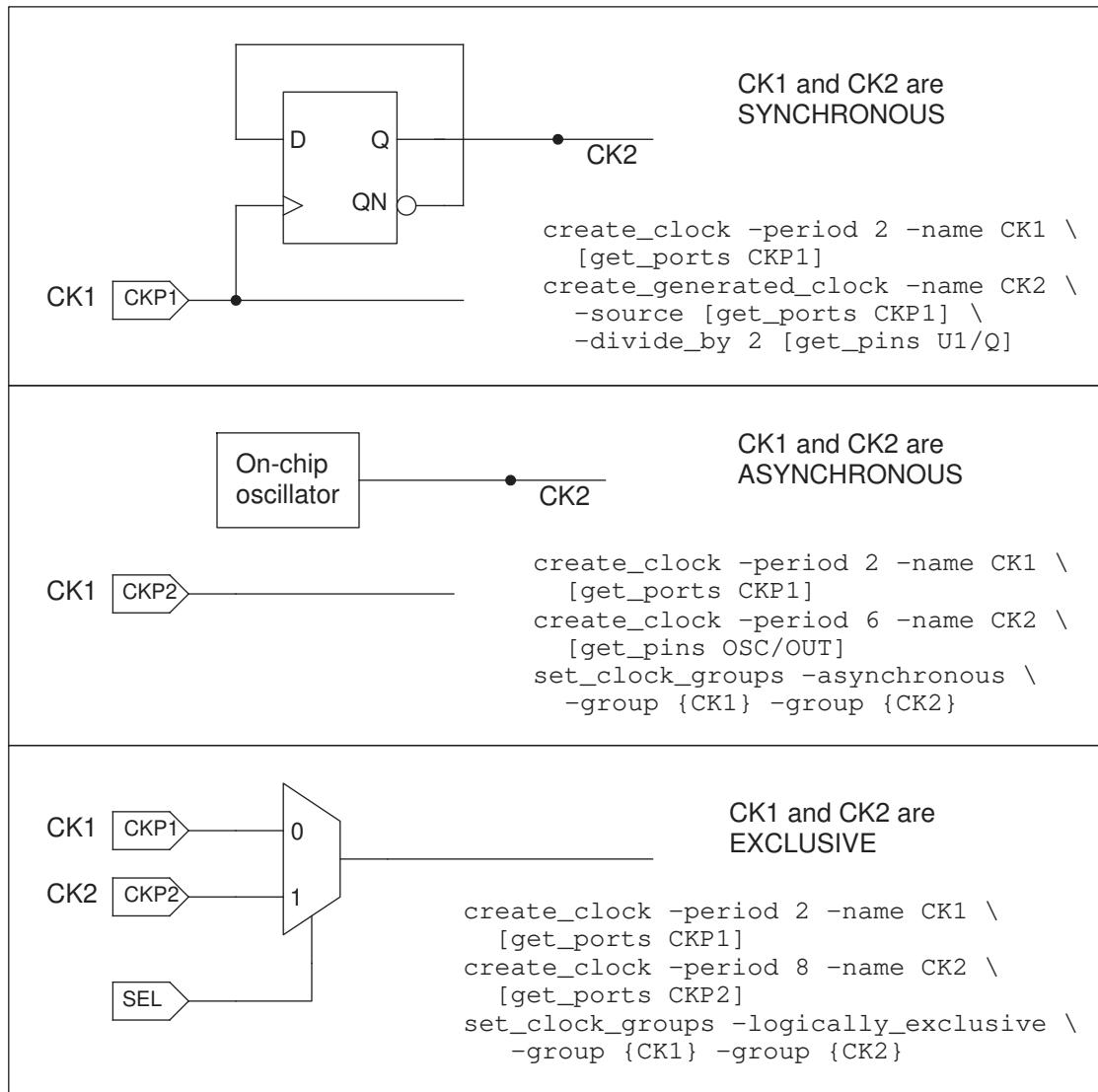
The “close edge clock latency” is the earliest arrival time of the second edge, taking into account the minimum delays through the same cells in the clock network. It also adds back clock reconvergence pessimism, which lessens the calculated violation; and subtracts clock uncertainty, which contributes to the calculated violation.

The “actual period” is the “close edge clock latency” minus the “open edge clock latency.” In this example, the actual period is less than the required period, resulting in a reported slack of –0.02.

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 the following figure.

Figure 44 Synchronous, asynchronous, and exclusive clocks



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.

Synchronous Clocks

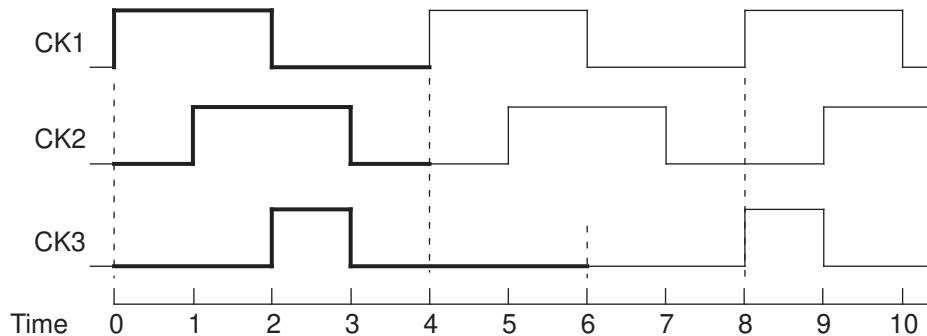
Two clocks are synchronous with respect to each other if they share a common source and have a fixed phase relationship. Unless you specify otherwise, PrimeTime assumes that two clocks are synchronous if there is any path with data launched by one clock

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}
```

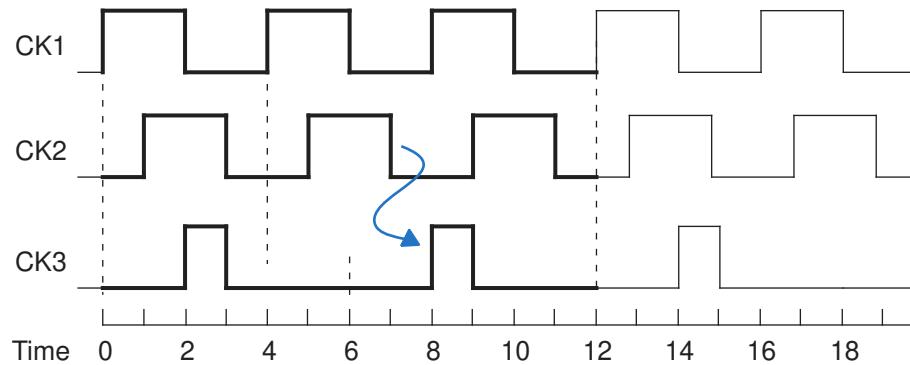
PrimeTime creates the clocks as specified in the commands, with the waveforms synchronized. PrimeTime adjusts the timing relationships further for any specified or calculated latency or uncertainty.

Figure 45 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 the following figure. Each expanded clock has a period of 12.

Figure 46 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 blue arrow.

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 needed, 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}
```

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](#).

To report the relationship between clocks, use the `get_clock_relationship` command:

```
pt_shell> get_clock_relationship {CK1 CK2}
```

To remove a clock grouping declaration, use the `remove_clock_groups` command:

```
pt_shell> remove_clock_groups -logically_exclusive -name EX1
```

To remove all exclusive clock grouping declarations made with the `set_clock_groups` command:

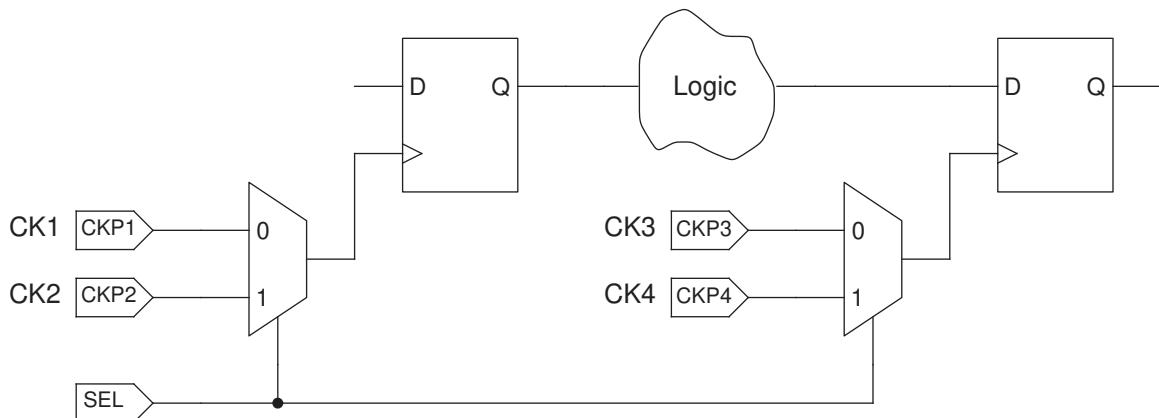
```
pt_shell> remove_clock_groups -logically_exclusive -all
```

The following examples demonstrate some of the ways to specify exclusive clocks.

Example 1: Four Clocks and One Selection Signal

Consider a circuit example with 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 47 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 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:

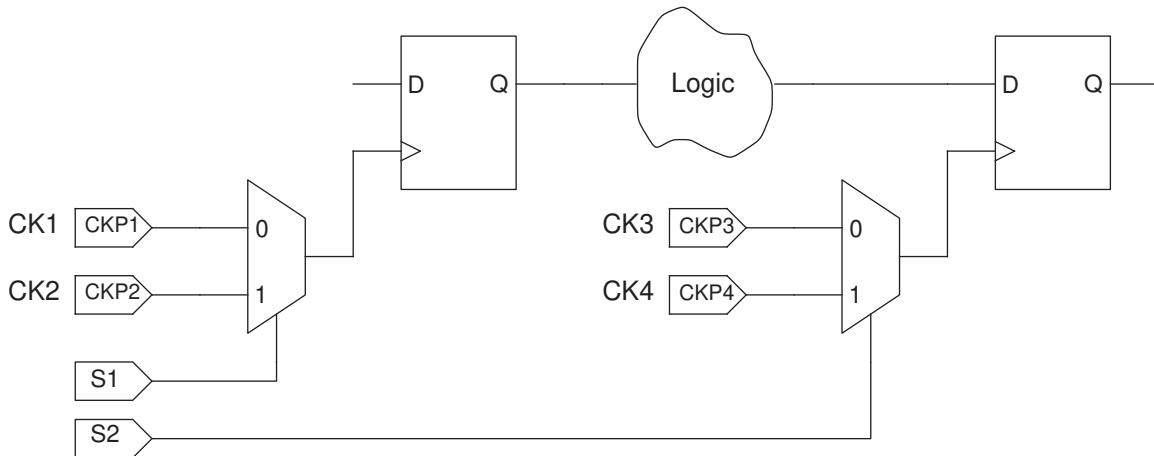
```
pt_shell> set_clock_groups -logically_exclusive -name E1 \
           -group {CK1 CK3} -group {CK2 CK4}
pt_shell> set_active_clocks {CK1,CK3}
```

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 2: Four Clocks and Two Selection Signals

Consider a circuit example with two separate clock selection inputs, S1 and S2, for the two multiplexers.

Figure 48 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.

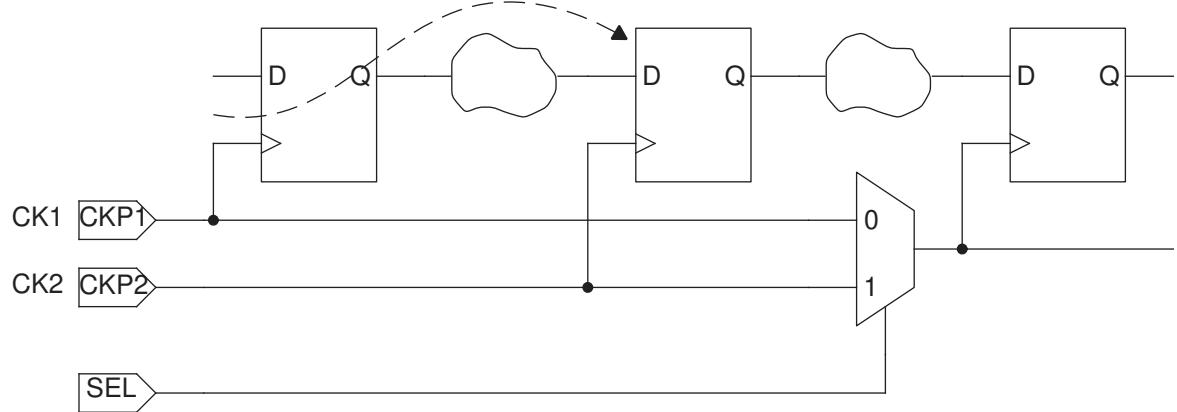
If you only want to consider the case where S1-S2=00, you can do it easily by using a different `set_active_clocks` command:

```
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 3: Multiplexed Clocks and Case Analysis

Consider a circuit example with the CK1 and CK2 clocks at the startpoint and endpoint of one path. These clocks are also multiplexed onto a shared clock line.

Figure 49 Multiplexed Clocks Specified With Case Analysis

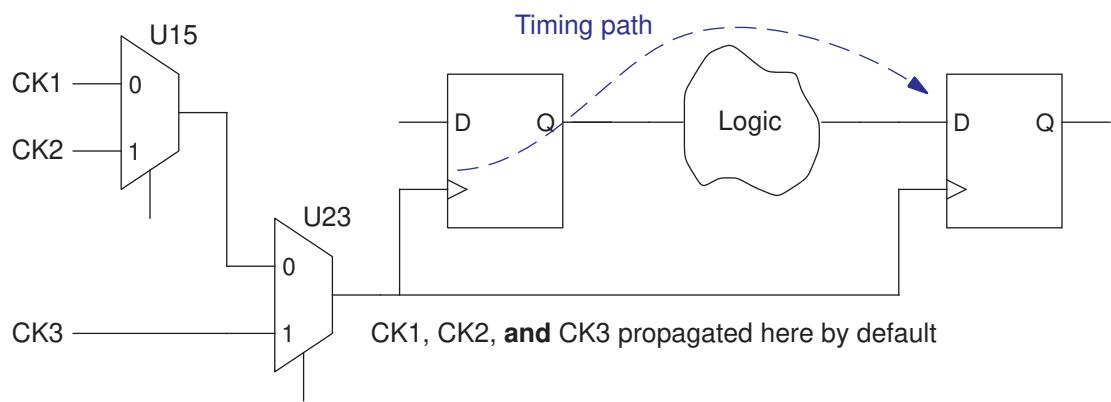


If you want PrimeTime to check the valid path between CK1 and CK2, but avoid checking invalid paths between the two clock domains downstream from the multiplexer, the best way to do it is with case analysis.

Multiplexed Clock Exclusivity Points

Modern chip designs often use multiple clocks that can be applied to a given circuit at different times, for example, to operate at a higher frequency during periods of high workload and lower frequency at other times to save power. Multiplexer (MUX) cells are often used to select these clocks for operation, as shown in the following figure.

Figure 50 Multiplexed Clocks

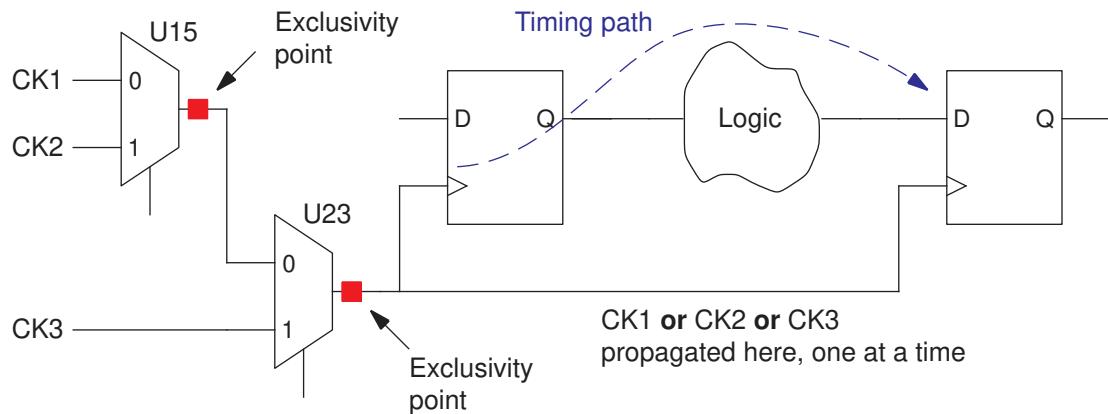


By default, the PrimeTime tool considers all combinations of launch and capture clocks, such as data launched by CK1 and captured by CK2. However, for clocks multiplexed as

shown in the figure, only one of the three clocks can operate on the path at any given time; they are exclusive clocks downstream from the MUX cells as long as the MUX control signals remain static between launch and capture.

One method of specifying exclusive clocks is to set “exclusivity points” to mark the pins in the design where only one clock at a time can be propagated, as shown in the following figure. For complex clock MUX configurations, this method typically offers better simplicity, accuracy, completeness, and turnaround time than other methods.

Figure 51 Multiplexed Clocks With Exclusivity Points



You can set clock exclusivity points at specific pins in the design, or you can have the tool automatically find and place these points at clock MUX output pins. The exclusivity setting applies to checking of delay, minimum pulse width, and minimum period constraints.

Setting Clock Exclusivity Points Explicitly

To set exclusivity points manually, use the `set_clock_exclusivity` command:

```
pt_shell> set_clock_exclusivity -type mux -output "U15/z"
pt_shell> set_clock_exclusivity -type mux -output "U23/z"
```

Setting a clock exclusivity point allows only one clock at a time to be propagated through the pin on which it is set. This enforces a physically exclusive relationship between the different clocks in the transitive fanout of the pins. The clock exclusivity effect works whether the `timing_enable_auto_mux_clock_exclusivity` variable is set to `true` or `false`.

Inferring Clock Exclusivity Points Automatically

To have the tool automatically find and place exclusivity points at the outputs of clock MUX cells, set the `timing_enable_auto_mux_clock_exclusivity` variable to `true`:

```
pt_shell> set_app_var timing_enable_auto_mux_clock_exclusivity true
```

With this variable set to `true`, the tool automatically finds multiple clock signals entering a MUX cell and places exclusivity points at the output of the MUX. This enforces a physically exclusive relationship between the clocks in the transitive *fanout* of the MUX cell, without affecting possible clock interactions in the transitive *fanin* to the MUX cell.

To disable this automatic inference at selected MUX cells in the design, use the following command:

```
pt_shell> set_disable_auto_mux_clock_exclusivity MUX32/z
```

Automatic detection of multiplexed clocks works only for MUX cells, as determined by the cell's `is_mux_cell` attribute. A multiplexing function built out of random logic is not automatically detected, but you can still manually define an exclusivity point in such logic by using the `set_clock_exclusivity` command. For example,

```
pt_shell> set_clock_exclusivity -type user_defined \  
-inputs "OR1/A OR1/B" -output OR1/z
```

For a non-MUX cell, you must specify the clock input pins and exclusivity output pin.

Reporting Exclusivity Points

To report the exclusivity points, use the `-exclusivity` option of the `report_clock` command:

```
pt_shell> report_clock -exclusivity  
...  
clock exclusivity points:  
  
-type mux  
-output MUX88/z
```

This reports all exclusivity points, whether inferred globally by using the `timing_enable_auto_mux_clock_exclusivity` variable or explicitly by using the `set_clock_exclusivity` command.

To query the exclusivity of a pin:

```
pt_shell> get_attribute -class pin [get_pins U53/z] is_clock_exclusivity  
true
```

Alternative Methods

Aside from using exclusivity points, you can use the following methods to prevent an overly conservative analysis that considers the interactions between exclusive clocks:

- Declare a false path between the clocks (`set_false_path`)
- Declare the clocks to be exclusive (`set_clock_groups -logically_exclusive`)

- Use case analysis to analyze only a particular operating mode (`set_case_analysis`)
- Disable checking of all paths leading from the different clock input pins (`set_disable_timing`)

Each of these methods has advantages and disadvantages with respect to simplicity, flexibility, turnaround time, and completeness. To define clocks as exclusive downstream (but not upstream) from the MUX outputs, you can define new clocks on the MUX outputs. For more about using these methods, see [Exclusive Clocks](#) and [SolvNetPlus article 000005025, "Specifying MUXed Clocks in PrimeTime."](#)

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

The tool 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. You do not need to

specify 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*.

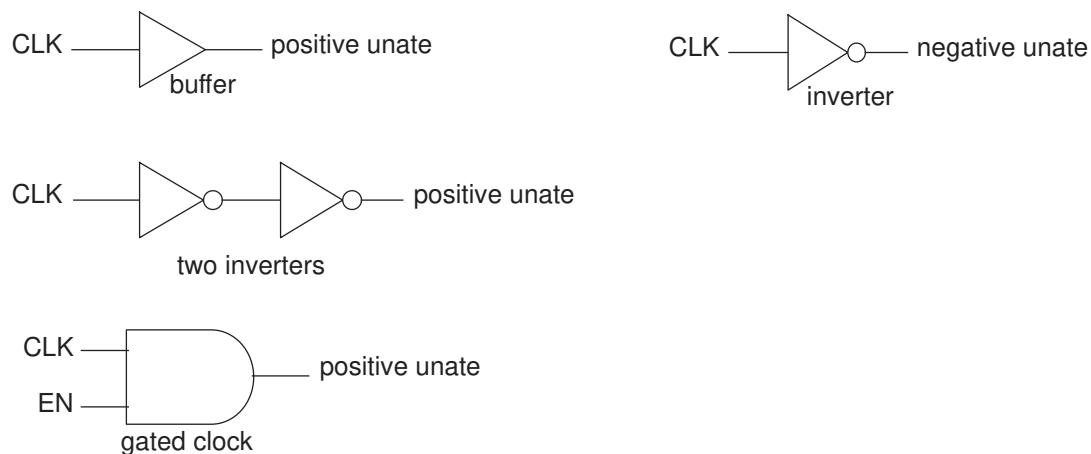
A clock signal is *positive unate* if

- A rising edge at the clock source causes a rising edge at the register clock pin.
- A falling edge at the clock source causes a falling edge at the register clock pin.

A clock signal is *negative unate* if

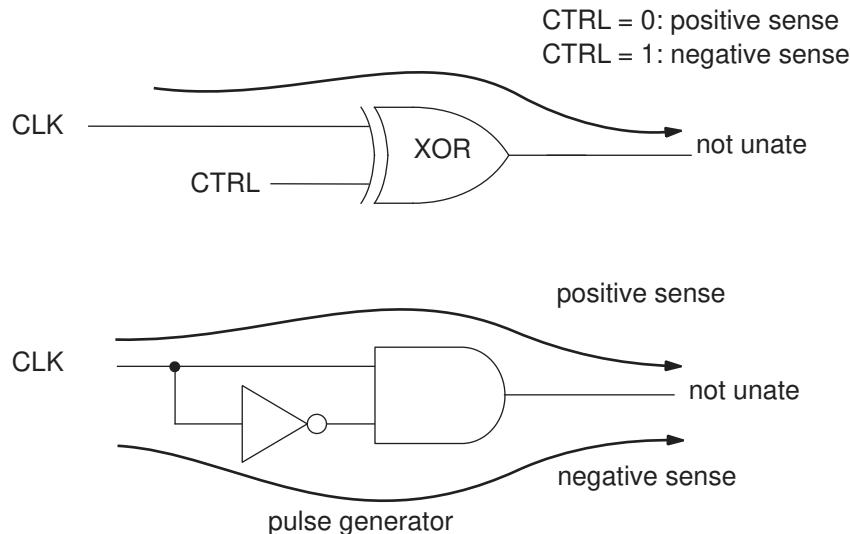
- A rising edge at the clock source causes a falling edge at the register clock pin.
- A falling edge at the clock source causes a rising edge at the register clock pin.

Figure 52 Positive and negative unate clock signals



A clock signal is *non-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.

Figure 53 Non-unate clock signals



The tool considers the output of the pulse generator at the bottom of the preceding figure 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 resolve this ambiguity for the tool, specify either a positive or negative clock sense to be propagated forward from a pin within a non-unate part of the clock network by using the `set_sense` command with the following syntax:

```
set_sense -type clock
    -positive | -negative
    object_list
```

For example, the following command propagates 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.

```
pt_shell> set_sense -type clock -positive [get_pins xor1.z]
```

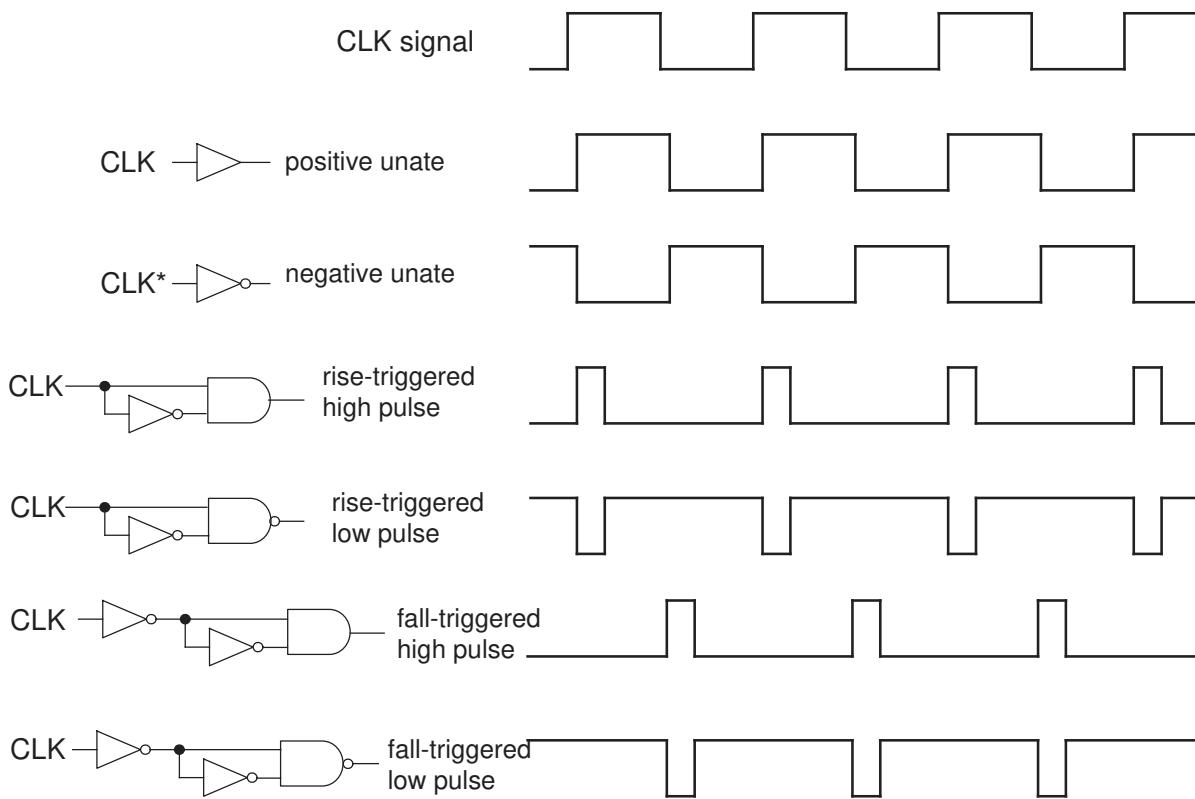
The `set_sense -type clock` command affects only the non-unate part of a clock network. If you specify this command for a pin in a unate part of the clock network, and the requested sense disagrees with the actual sense, PrimeTime issues an error message and disregards the command.

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 specific 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 provides options that support pulse clocks. The following examples show clock-modifying circuits and the resulting clock senses.

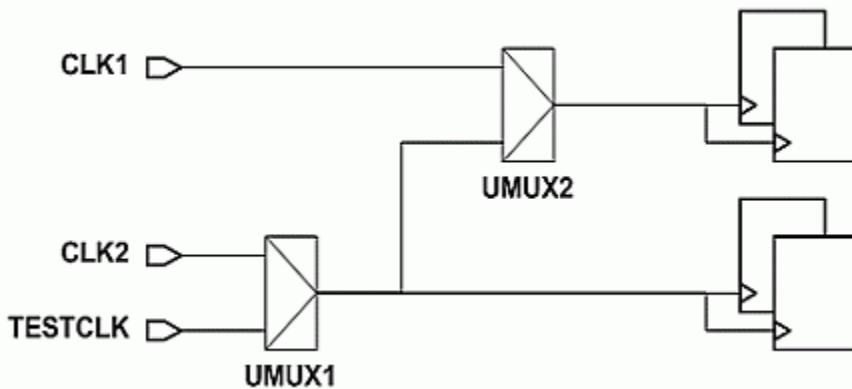
Figure 54 Clock sense examples



To stop clock propagation from a specific pin or cell timing arc, use the `set_sense -type clock -stop_propagation` command. This is appropriate in cases where the clock physically does not propagate past a specific point.

In the following example of physical clock stopping, 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.

Figure 55 Physical Clock Stopping



To model this in PrimeTime, use this command:

```
pt_shell> set_sense -type clock -stop_propagation -clocks TESTCLK UMUX2/z
```

This tells the tool that the clock signal is not used as a clock at the specified pin, which prevents clock gating checks on the clock network leading up to the pin.

To stop propagation of a clock signal at a leaf pin of a clock network, use the `-clock_leaf` option instead of the `-stop_propagation` option:

```
pt_shell> set_sense -type clock -clock_leaf -clocks CLK3 UMUX3/z
...
```

This tells the tool that the clock signal is used as a clock at the specified pin but is not propagated beyond that pin. The tool still performs clock gating checks on the clock network leading up to the pin.

To remove the sense that was previously specified by the `set_sense -type clock`, use the `remove_sense -type clock` command.

Specifying 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.

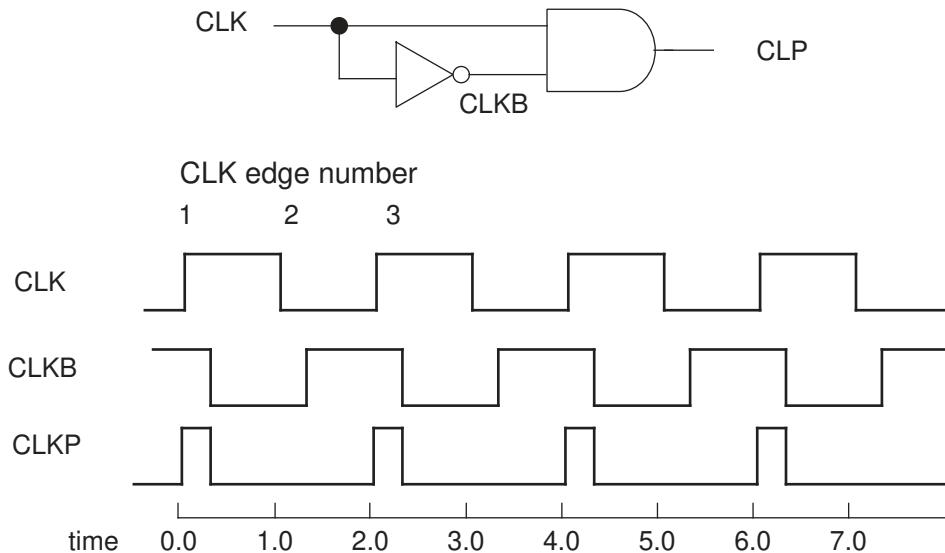
To analyze the timing of a circuit containing pulse clocks, PrimeTime needs information about the timing characteristics of the clock. There are three ways to provide this information:

- 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. In the following pulse clock circuit, each rising edge of CLK generates a pulse on CLKP.

Figure 56 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 (instead of 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 an existing clock.

For example, the following command specifies 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.

```
pt_shell> set_sense -type clock -pulse rise_triggered_high_pulse \
    [get_pins and2/z]
```

In general, to specify the clock sense of a pulse clock, use the following syntax:

```
set_sense
    -type clock
    -pulse rise_triggered_high_pulse | rise_triggered_low_pulse |
        fall_triggered_high_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:

- (high pulse width) = (fall network latency) – (rise network latency)
- (low pulse width) = (rise network latency) – (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, the following commands 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:

```
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

The transitive fanout of a pulse generator is a *pulse clock network*. The clock propagating through the pulse generator in the pulse clock network is 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.

To constrain a pulse clock network, use these commands:

- `set_pulse_clock_min_width -transitive_fanout value object_list`

The `set_pulse_clock_min_width` command constrains 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.

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.

- `set_pulse_clock_max_width -transitive_fanout value object_list`

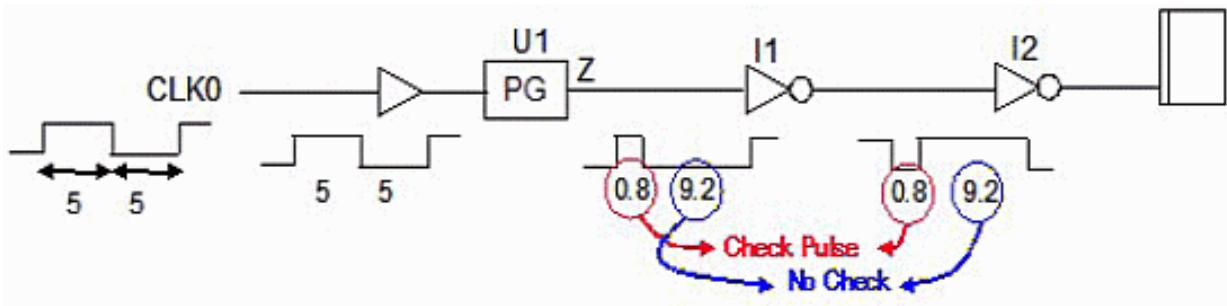
The `set_pulse_clock_max_width` command constrains the maximum pulse width in the fanout of pulse generator instances by pulse generator instance name or pulse generator library cell.

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 the pulse width 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. In the following example,

- Before the inverter, the minimum pulse width of the high pulse is constrained.
- After the inverter, the minimum pulse width of the low pulse is constrained.

Figure 57 Sense propagation affects high and low pulse width propagation



To remove the minimum or maximum pulse width constraint from the pulse generator network, use the `remove_pulse_clock_min_width` or `remove_pulse_clock_max_width` command, respectively. When these commands, you must explicitly specify all of the sources of the constraint.

To report the pulse width of all pulse generator networks, use these commands:

- `report_pulse_clock_min_width` or `report_constraint -pulse_clock_min_width`
- `report_pulse_clock_max_width` or `report_constraint -pulse_clock_max_width`

By default, the tool applies the general minimum pulse width constraints everywhere except the pulse clock network and applies the more specific pulse clock constraints in the pulse clock network. To override this default behavior, set the `timing_enable_pulse_clock_constraints` variable to `false`; with this setting, the tool ignores the more specific pulse clock constraints and applies the general minimum pulse width constraints everywhere, including pulse clock networks.

Note:

During the `report_analysis_coverage` command, the tool temporarily sets the `timing_enable_pulse_clock_constraints` variable to `false` to properly obtain design information. An information message is also printed.

Applying Clock Uncertainty to Pulse Width Checking

By default, the clock uncertainty specified by the `set_clock_uncertainty` command applies to all types of timing constraints, including pulse width checking. You can optionally specify whether to apply clock uncertainty to pulse width checking, and if so, whether to apply only setup or only hold uncertainty, or both. To use this option, set the following variable:

```
set_app_var timing_include_uncertainty_for_pulse_checks \
    none | setup_only | hold_only | setup_hold
```

The default is `setup_hold`, which applies both setup and hold uncertainty to pulse width checking.

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.

To remove the constraint from the input of pulse generator cells or from the pulse generator, use the `remove_pulse_clock_min_transition` or `remove_pulse_clock_max_transition -transitive_fanout` command, respectively.

Note:

When removing the constraint from the input of pulse generators, do not use the `-transitive_fanout` option.

To report the maximum transition computation, 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. 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 a single-output PLL, the `-pll_output` pin is the same as the generated clock source pin. For a multiple-output PLL 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:

- 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.

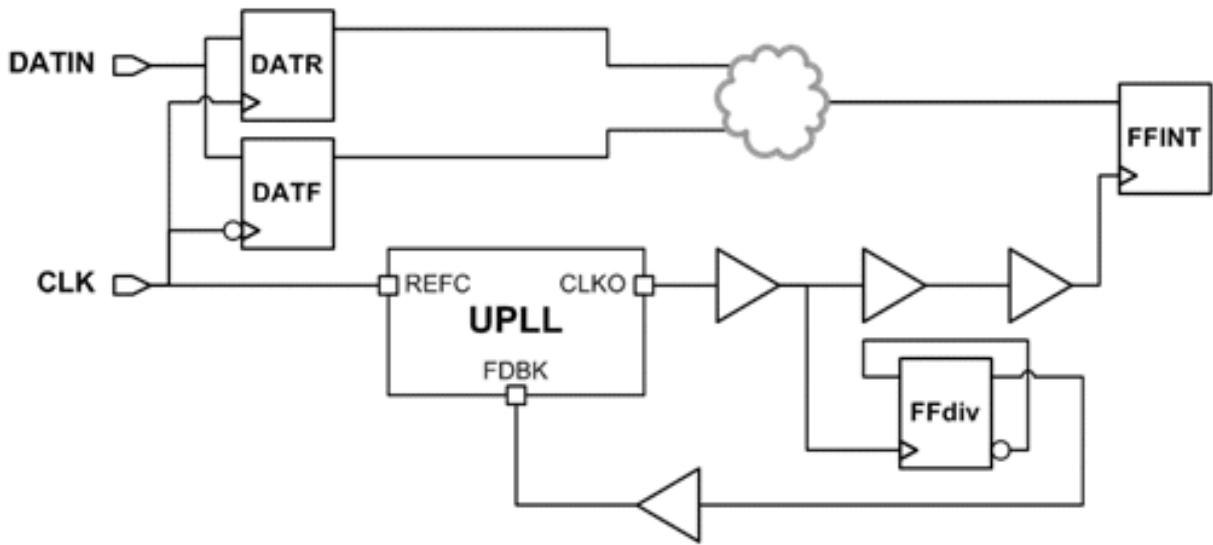
For the PLL adjustment to occur correctly, set the PLL output clock and reference clock arriving at the reference clock pin of the PLL as propagated clocks.

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 conditions:

- The fanout of the output pin is connected to the feedback pin.
- The pin lies on the feedback path.

Figure 58 Circuit example



In the preceding circuit, the PLL output clock at pin CLKO is twice the frequency of the PLL reference clock at pin REFC. 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/REFC] \
  -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.

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` 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")
        }
    }
}
}

```

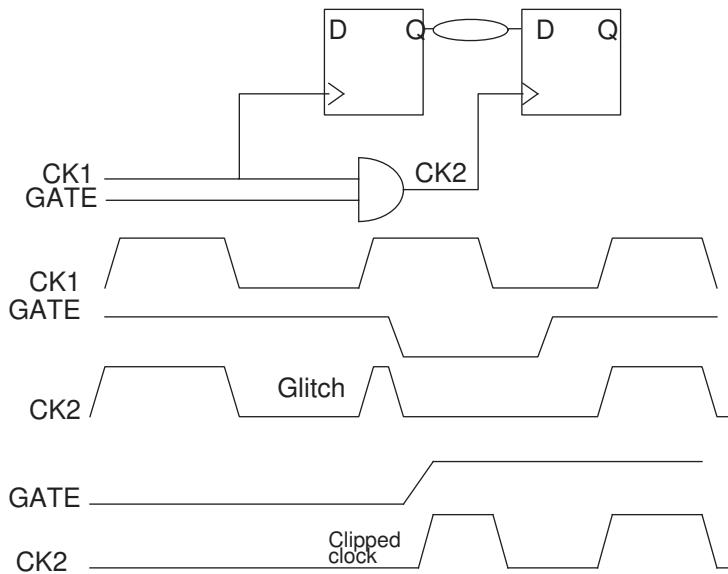
Specifying Clock-Gating Setup and Hold Checks

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.

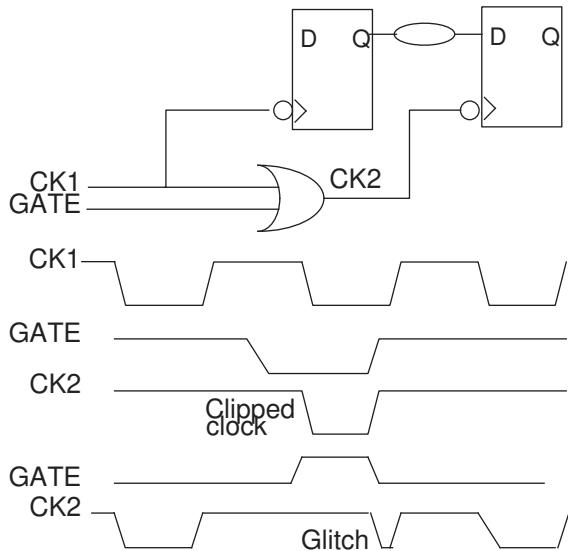
The clock-gating setup check ensures that the control data signal enables the gate before the clock becomes active. The arrival time of the leading edge of the clock pin is checked against both edges of any data signal feeding the data pins to prevent a glitch at the leading edge of the clock pulse or clipped clock pulse. The clock-gating setup violations are shown in the following figure.

Figure 59 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 60.

Figure 60 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 restrict the reporting of clock gating violations to only those gates inferred automatically by the PrimeTime tool, use the `report_clock_gating_check` command with the `-auto_inferred_only` option. This prevents the reporting of checks inserted by the Power Compiler tool or defined by clock-gating library cells.

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:

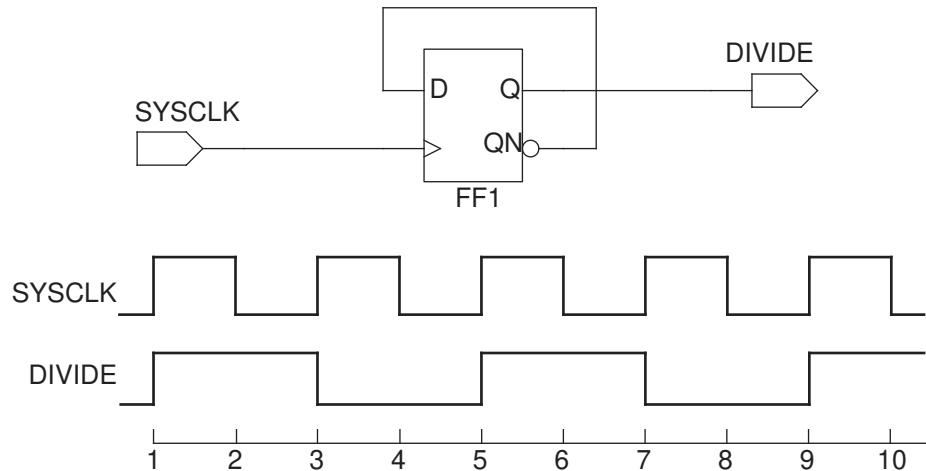
```
pt_shell> set_disable_clock_gating_check U44/z
```

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.

The following figure 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.

Figure 61 Divide-by-2 clock generator



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.

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 61](#), 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]
```

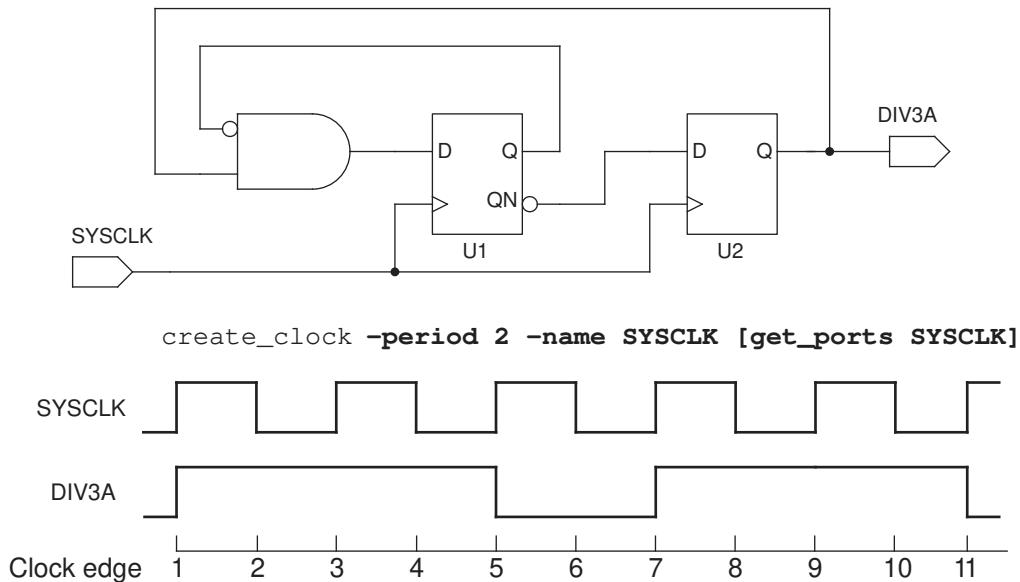
Specify a port or a pin (not a clock name) as the master source from which the new clock is generated. Specify a pin as the creation point for the new generated clock.

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 following clock generator circuit.

Figure 62 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

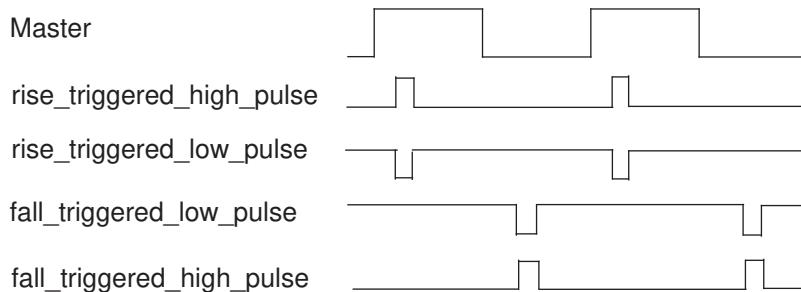
Pulse generators can improve circuit performance and save power consumption. Because of clock gating, skew management, and preserving generated pulse integrity, it might

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 63 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
```

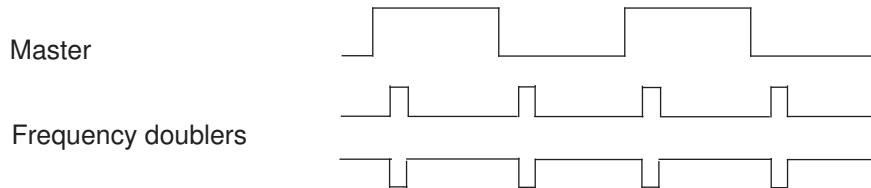
If your waveform is non-monotonic, yet has increasing values, use the `create_clock -waveform` command. This instructs PrimeTime to create the correct clock for those clocks driven by a pulse generator. For example,

```
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.

Figure 64 Pulse clock types that change frequency

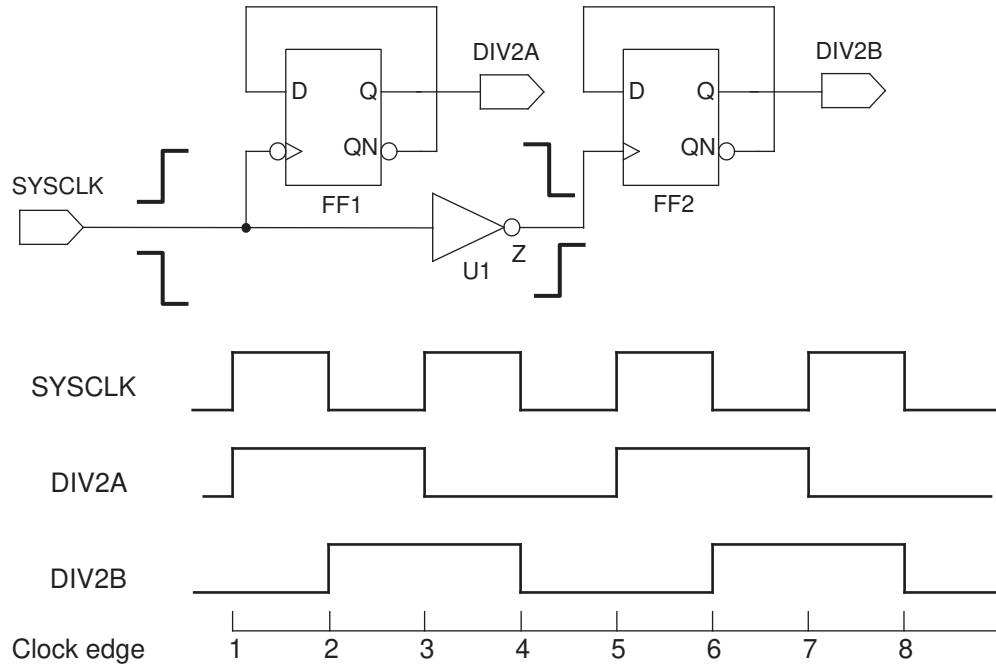


If the requested sense of clock does not exist, PrimeTime issues a warning. For example, if there was only a positive unate path from the clock source to a pin with `set_sense -type clock -pulse rise_triggered_high_pulse`, PrimeTime does not run the command as no rising at the clock source to failing at the clock sense pin path is found. However, if there are both a positive and a 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 following generated clock examples show how to do this.

Figure 65 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]
```

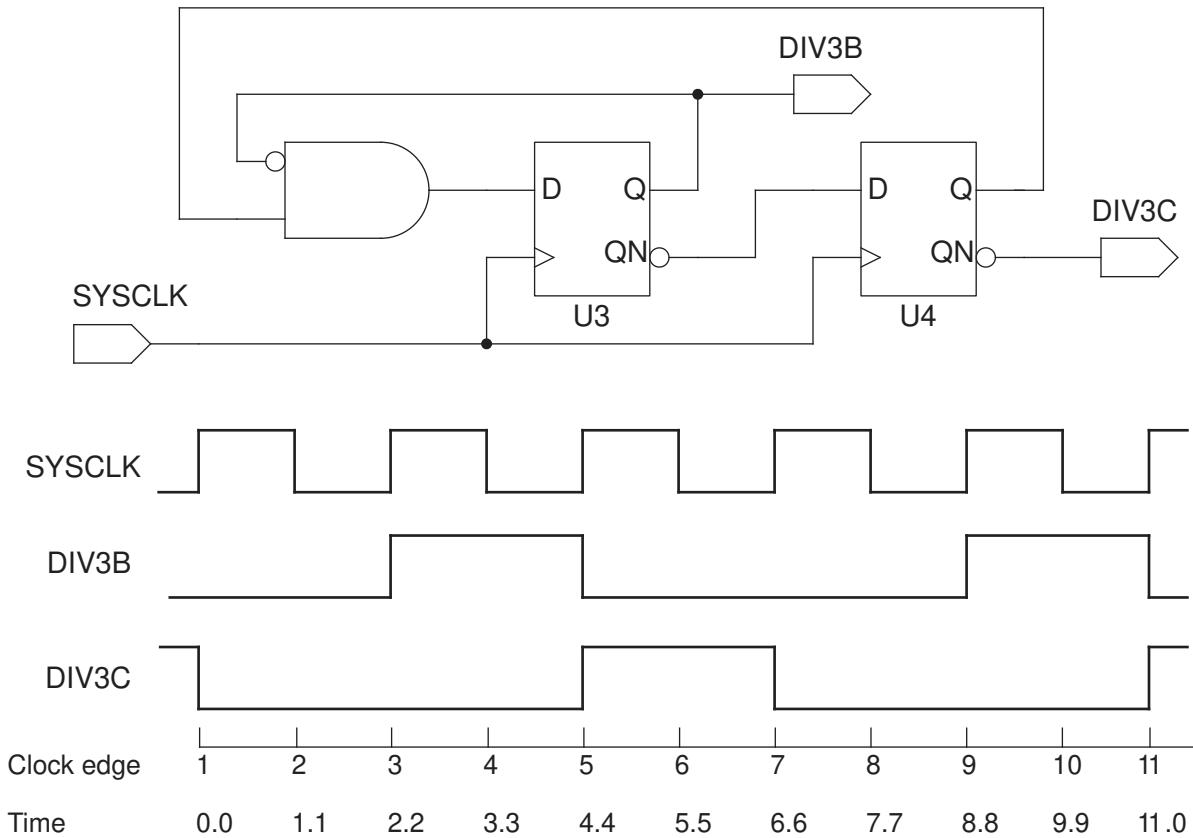
Another way to specify DIV2B is to use a different source pin:

```
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 following clock generator circuit.

Figure 66 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

For information about clocks and generated clocks in the current design, use the `report_clock` command. For a detailed report about a clock network, use the `report_clock_timing` command as described in [Clock Network Timing Report](#).

To show the clock networks in your design, use the `report_transitive_fanout_clock_tree` command.

Removing Generated Clock Objects

To remove generated clocks, use the `remove_generated_clock` command.

For example, the following command removes the CLK_DIV2 generated clock (and its corresponding clock object):

```
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

```

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

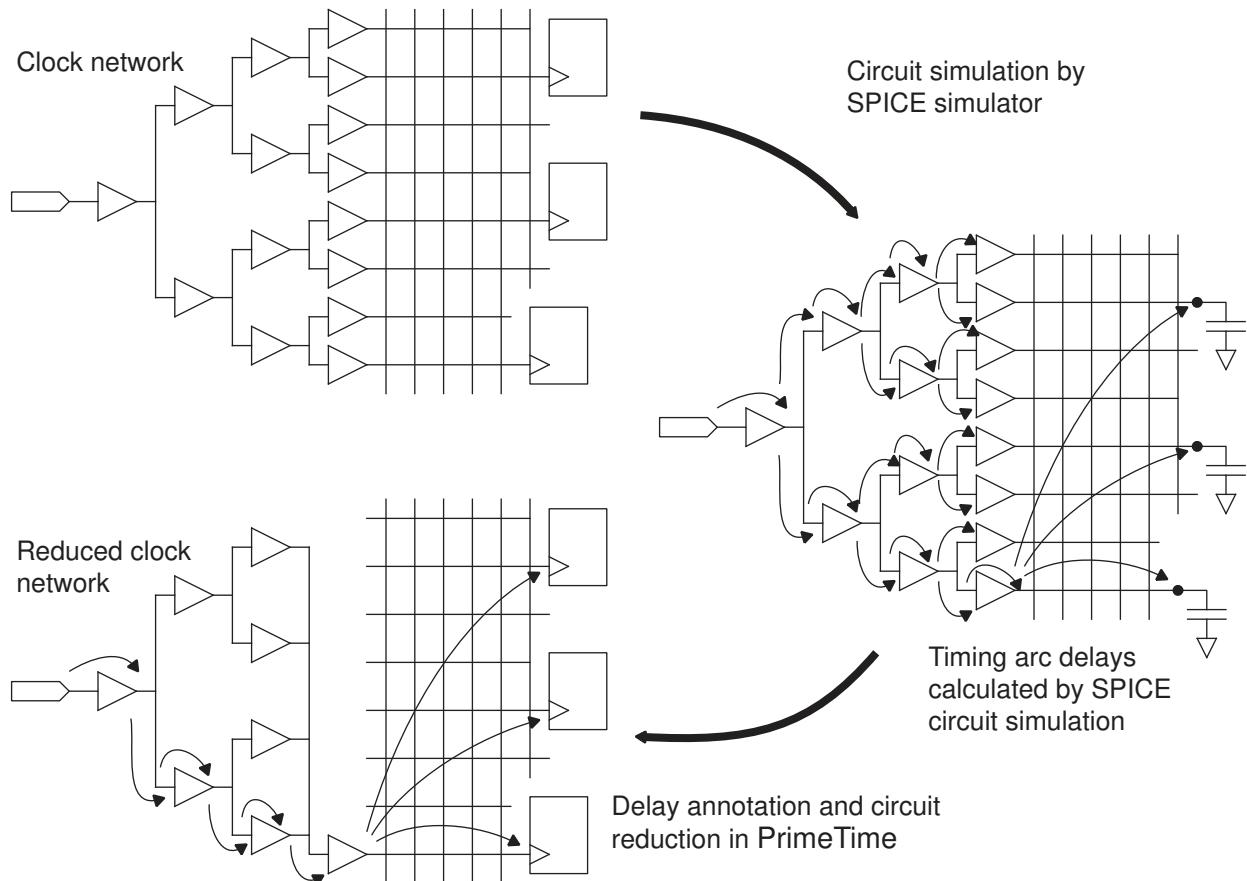
- [Overview of Clock Mesh Analysis](#)
 - [Performing Clock Mesh Analysis](#)
 - [Clock Network Simulation Commands](#)
-

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 a single mesh driver 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.

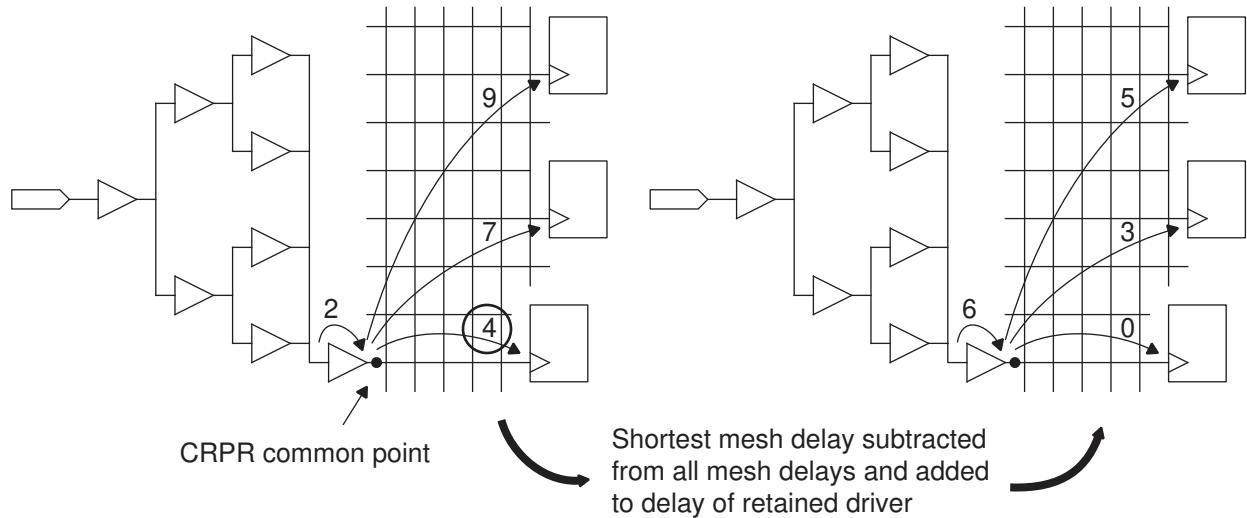
Figure 67 Clock mesh analysis and circuit reduction



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. During CRPR calculations, PrimeTime SI makes the following adjustment.

Figure 68 Clock Reconvergence Pessimism Removal From Mesh



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.

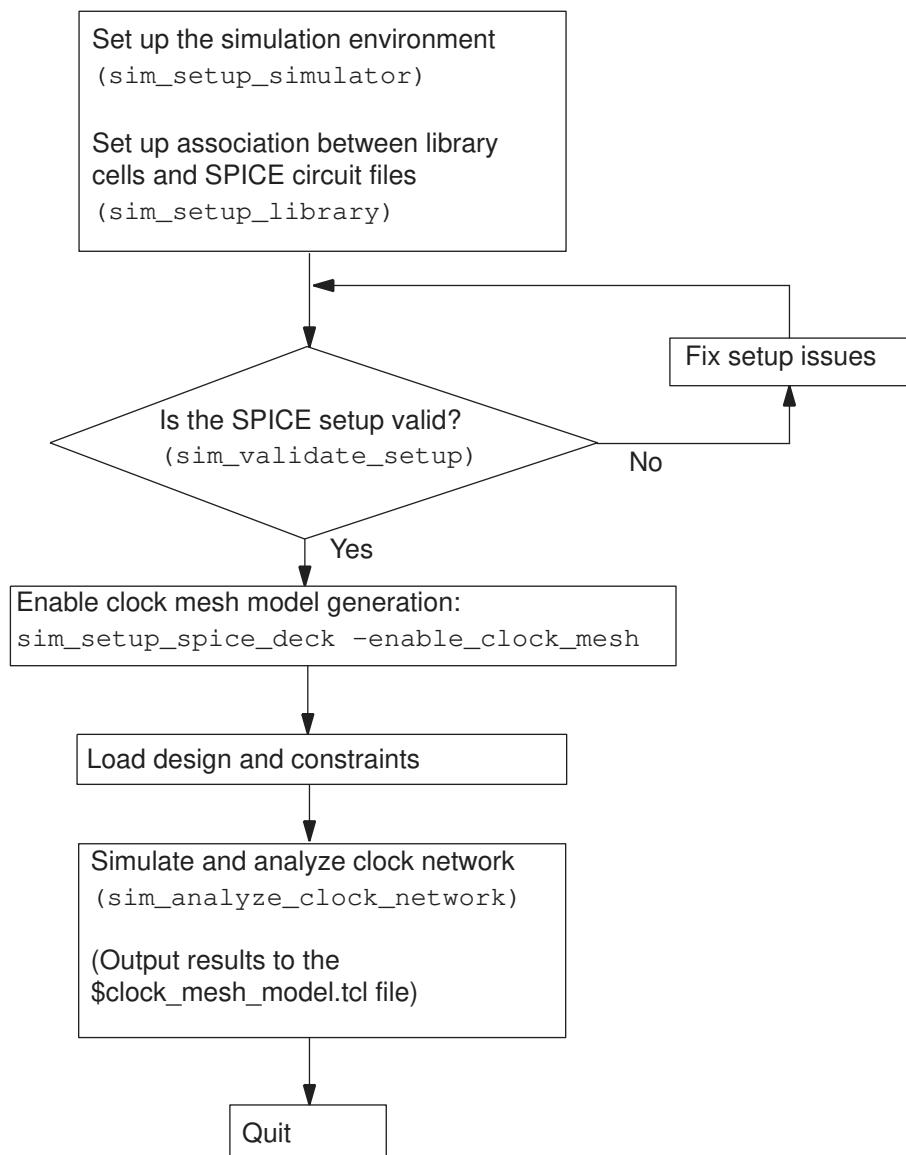
If you use the `report_timing` command to report a timing path that goes through the mesh, the report shows the timing through the single retained mesh driver and reflects the results obtained from the SPICE simulations.

Performing Clock Mesh Analysis

To perform clock mesh analysis:

1. Generate the clock mesh model using clock network simulation commands. Perform this step one time.

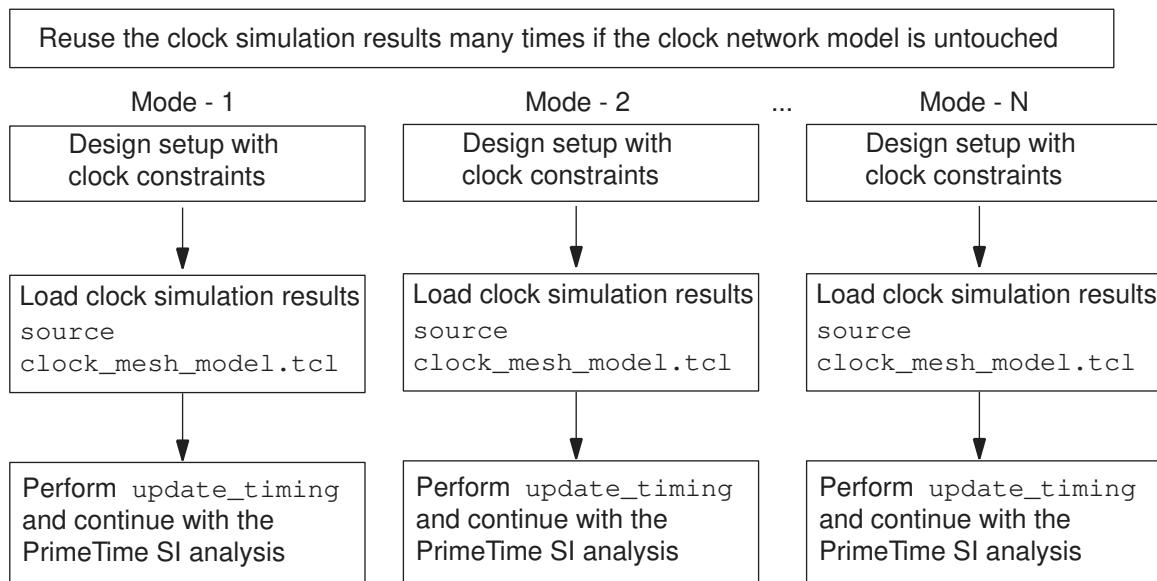
Figure 69 Generating the Clock Mesh Model



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 70 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_in_collection 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 ..."
```

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}
```

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
```

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
```

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
```

8

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

- [Timing Path Groups](#)
 - [Path Timing Reports](#)
 - [Path Timing Calculation](#)
 - [Specifying Timing Paths](#)
 - [Timing Exceptions](#)
 - [Saving and Restoring Timing Path Collections](#)
-

Timing 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 create a new clock with the `create_clock` command. 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.

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. Note that the `group_path` command is not compatible with hierarchical model extraction.

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*]
```

Unconstrained paths do not belong to any path group. To report unconstrained paths, set the `timing_report_unconstrained_paths` variable to `true`. The `report_timing` command reports unconstrained paths as belonging to a path group called “(none)”.

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 specific paths. You can specify the details provided in the output. For example:

- 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

Use the options of the `report_timing` command 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
```

...

```
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
ffd/D (DTN10)		0.46	0.00	9.15 r
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 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.

Specifying Timing Paths

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
```

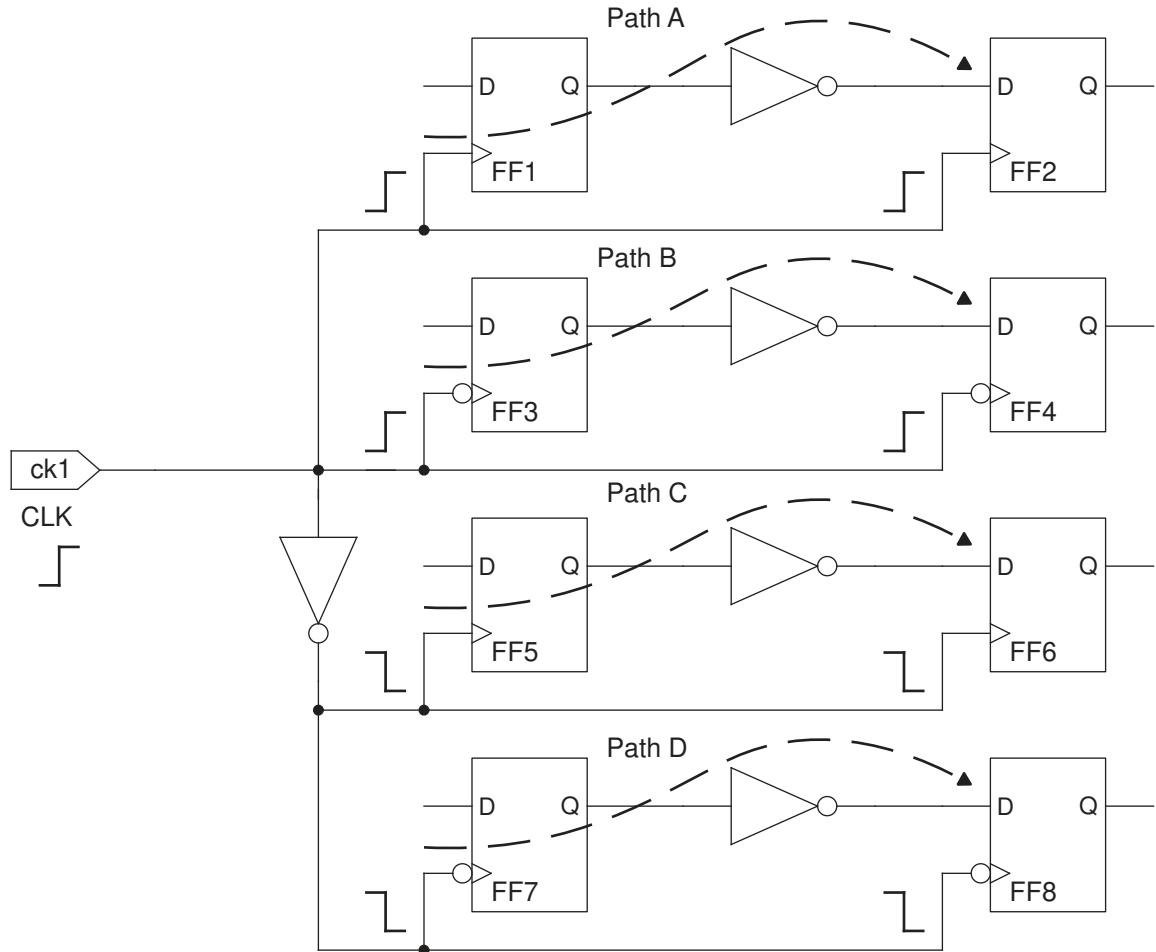
This means any path that starts at A1, passes through either B1 or B2, then passes through either C1 or C2, and ends at D1.

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 71](#). 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 71 Circuit for path specification in examples 1 to 3



Example 1

```
pt_shell> set_false_path -to [get_clocks CLK]
```

In Figure 71, 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 71, 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

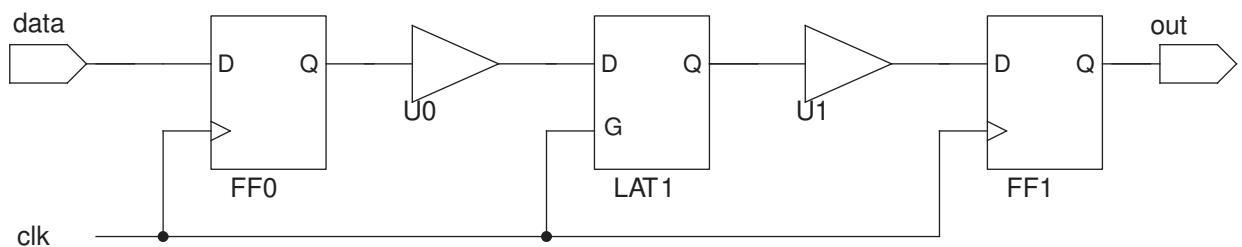
```
pt_shell> set_false_path -fall_to [get_clocks CLK]
```

In Figure 71, 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 72. 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 72 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/D` to `FF1`, from `LAT1/G` to `FF1`, and from `FF1` to `out`:

```
pt_shell> report_timing -rise_from [get_clocks clk] -nworst 100
```

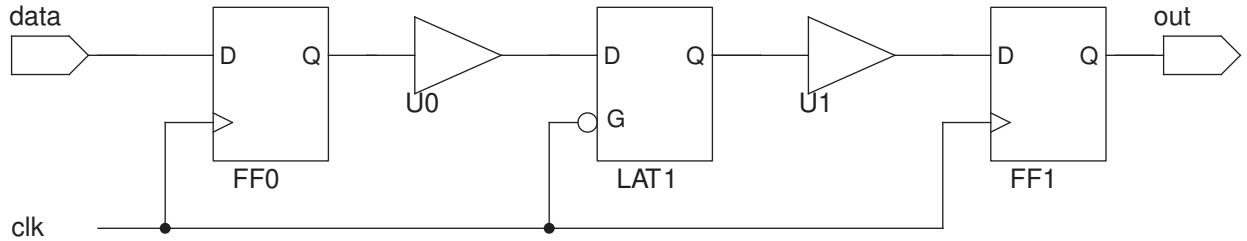
Using the `-fall_to` option of the command instead reports the path with data captured by a falling edge of the clock, which is from `FF0` to `LAT1`:

```
pt_shell> report_timing -fall_to [get_clocks clk] -nworst 100
```

Example 5

The circuit shown in Figure 73 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 73 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
```

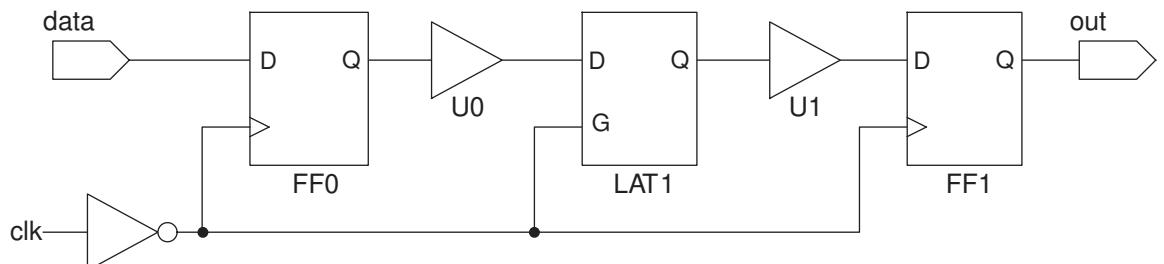
Using the `-rise_to` option of the command reports the paths from `data` to `FF0`, from `FF0` to `LAT1`, from `LAT1` to `FF1`, and from `FF1` to `out`:

```
pt_shell> report_timing -rise_to [get_clocks clk] -nworst 100
```

Example 6

The circuit shown in Figure 74 is the same as in Example 4 (Figure 72), except that the clock signal is inverted.

Figure 74 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
```

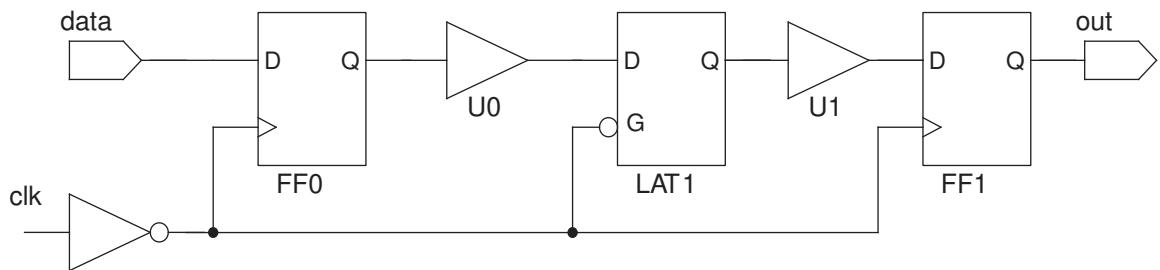
Using the `-rise_to` option of the command reports the path with data captured by a rising edge of the clock, which is from `FF0` to `LAT1` and from `FF1` to `out`:

```
pt_shell> report_timing -rise_to [get_clocks clk] -nworst 100
```

Example 7

The circuit shown in Figure 75 is the same as in Example 5 (Figure 73), except that the clock signal is inverted.

Figure 75 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
```

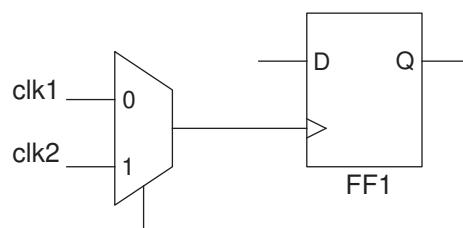
Using the `-fall_to` option of the command reports the paths from `data` to `FF0`, from `FF0` to `LAT1`, from `LAT1/D` to `FF1`, and from `LAT1/G` to `FF1`:

```
pt_shell> report_timing -fall_to [get_clocks clk] -nworst 100
```

Example 8

In the circuit shown in Figure 76, multiple clocks reach the flip-flop.

Figure 76 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

- [Overview of Timing Exceptions](#)
- [Single-Cycle \(Default\) Path Delay Constraints](#)
- [Setting False Paths](#)

- [Setting Maximum and Minimum Path Delays](#)
 - [Setting Multicycle Paths](#)
 - [Specifying Timing Margins](#)
 - [Specifying Exceptions Efficiently](#)
 - [Exception Order of Precedence](#)
 - [Reporting Exceptions](#)
 - [Reporting Timing Exceptions](#)
 - [Reporting Exceptions Source File and Line Number Information](#)
 - [Checking Ignored Exceptions](#)
 - [Removing Exceptions](#)
-

Overview of Timing Exceptions

You can set the following timing exceptions:

Timing exception	Command	Description
False paths	<code>set_false_path</code>	Prevents analysis of the specified path; removes timing constraints on the path
Minimum and maximum path delays	<code>set_max_delay</code> and <code>set_min_delay</code>	Overrides the default setup and hold constraints with specific maximum and minimum time values
Multicycle paths	<code>set_multicycle_path</code>	Specifies the number of clock cycles required to propagate data from the start to the end of the path

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 [Specifying Timing Paths](#).

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.

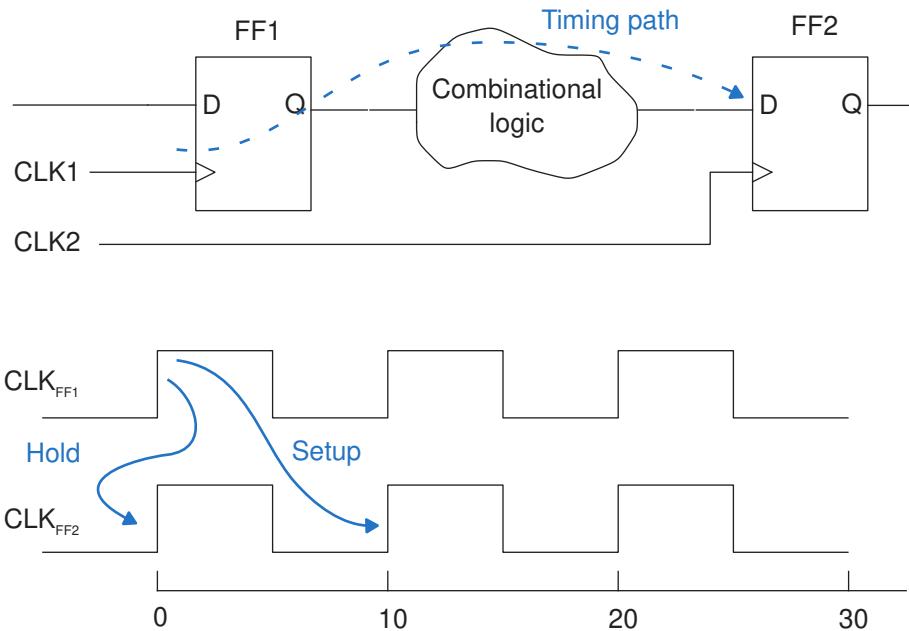
Single-Cycle (Default) Path Delay Constraints

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

The following figure 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.

Figure 77 Single-Cycle Setup and Hold for Flip-Flops



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.

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.

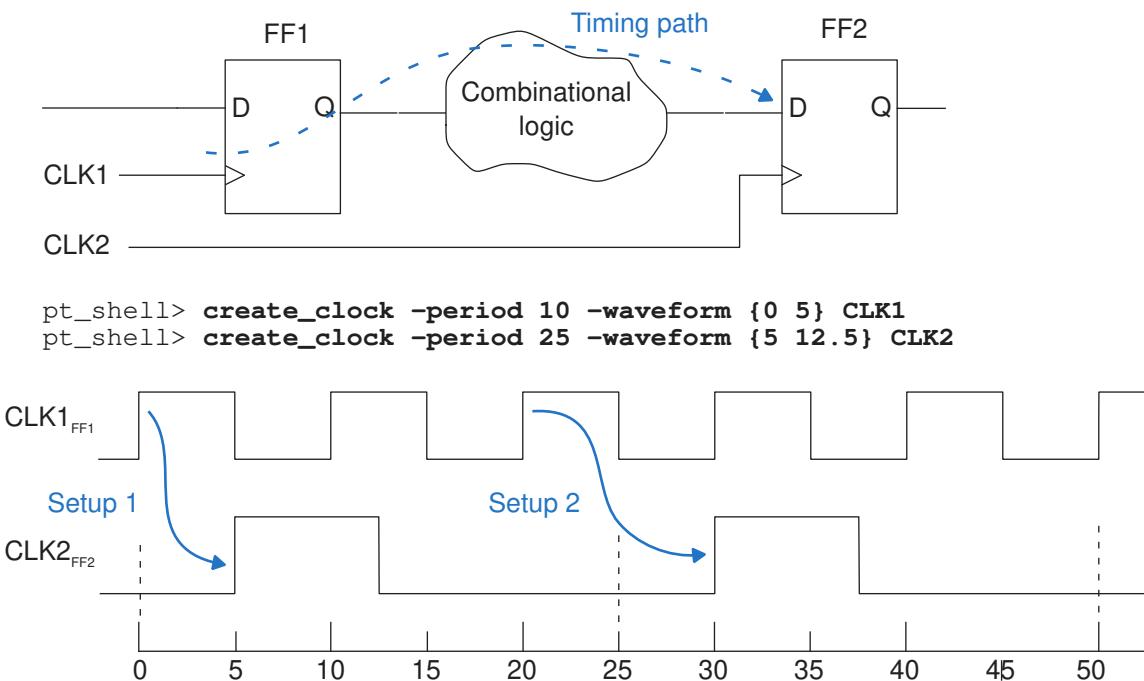
The setup and hold requirements determined by PrimeTime for sequential elements take into account all relevant parameters such as the delay of the combinational logic elements in the path, the setup and hold requirements of the flip-flop elements as defined in the logic library, and the net delays between 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.

In the following figure, the CLK1 and CLK2 clocks control the flip-flops at the beginning and end of the timing path. The `create_clock` commands define the two clocks.

Figure 78 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 78](#), 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 78](#).

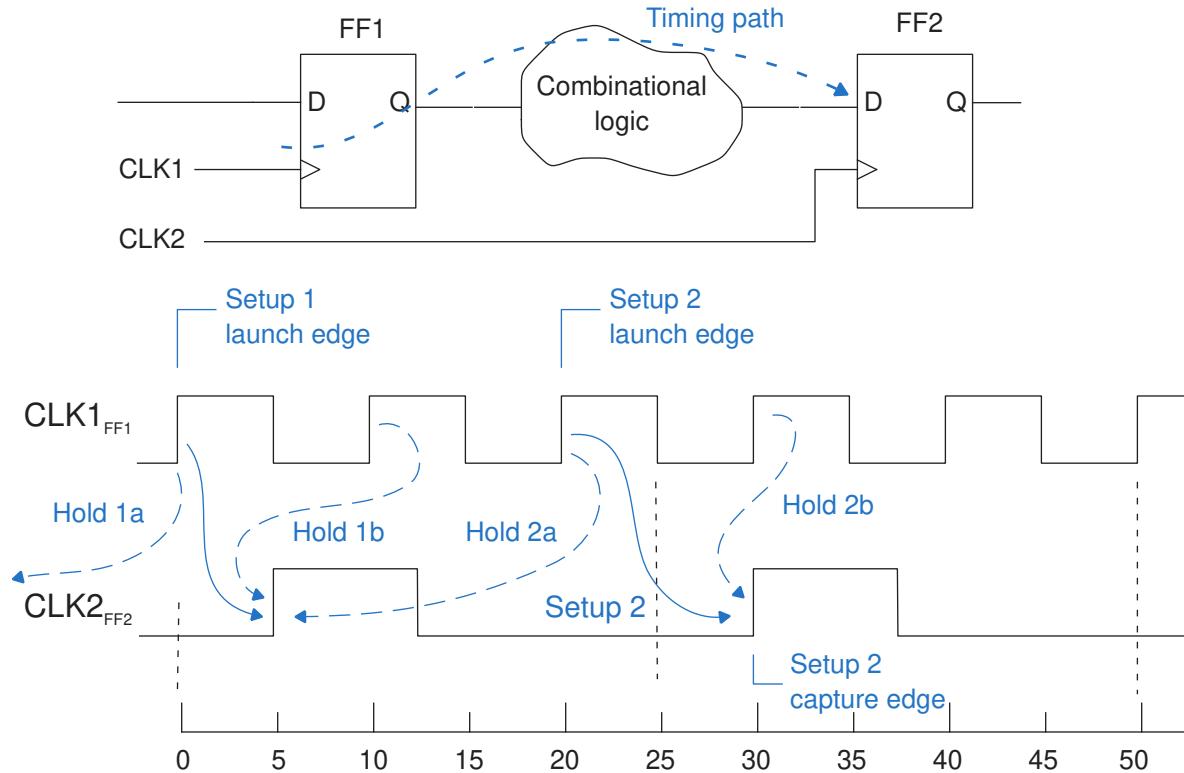
For each setup relationship, PrimeTime performs two different hold checks:

- 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.

In the following example, PrimeTime performs these hold checks:

- Hold 2a – PrimeTime checks that the data launched by the Setup 2 launch edge is not captured by the previous capture edge.
- Hold 2b – PrimeTime checks that the data launched by the next clock edge at the source is not captured by the setup capture edge.

Figure 79 Hold Constraints for Flip-Flops With Different Clocks



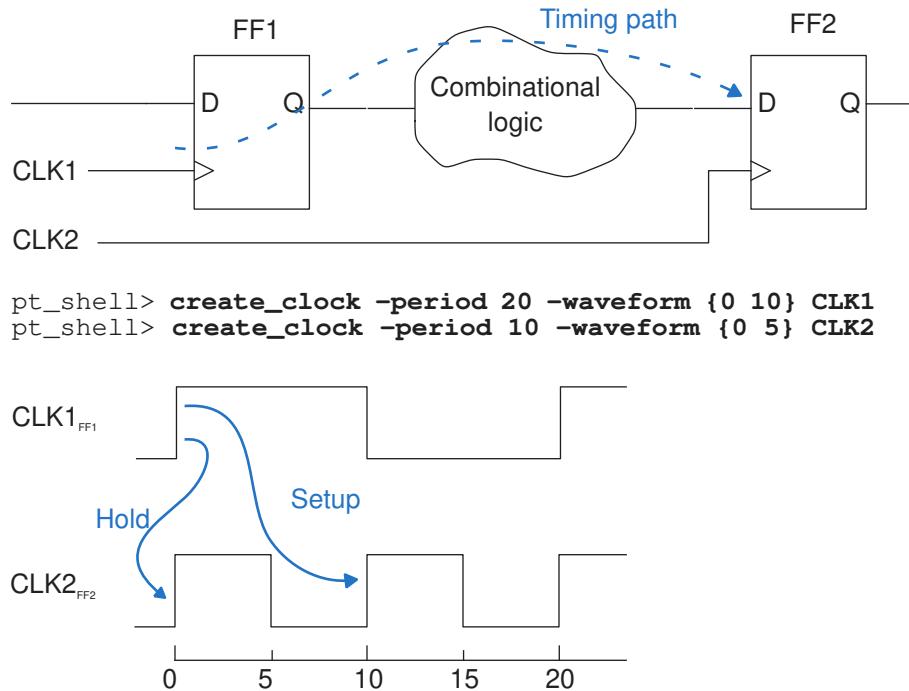
PrimeTime similarly checks the hold relationships relative to the clock edges of Setup 1, as indicated in the figure. The most restrictive hold check is the one where the capture edge occurs latest relative to the launch edge, which is Hold 2b in this example. Therefore, Hold 2b determines the minimum 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.

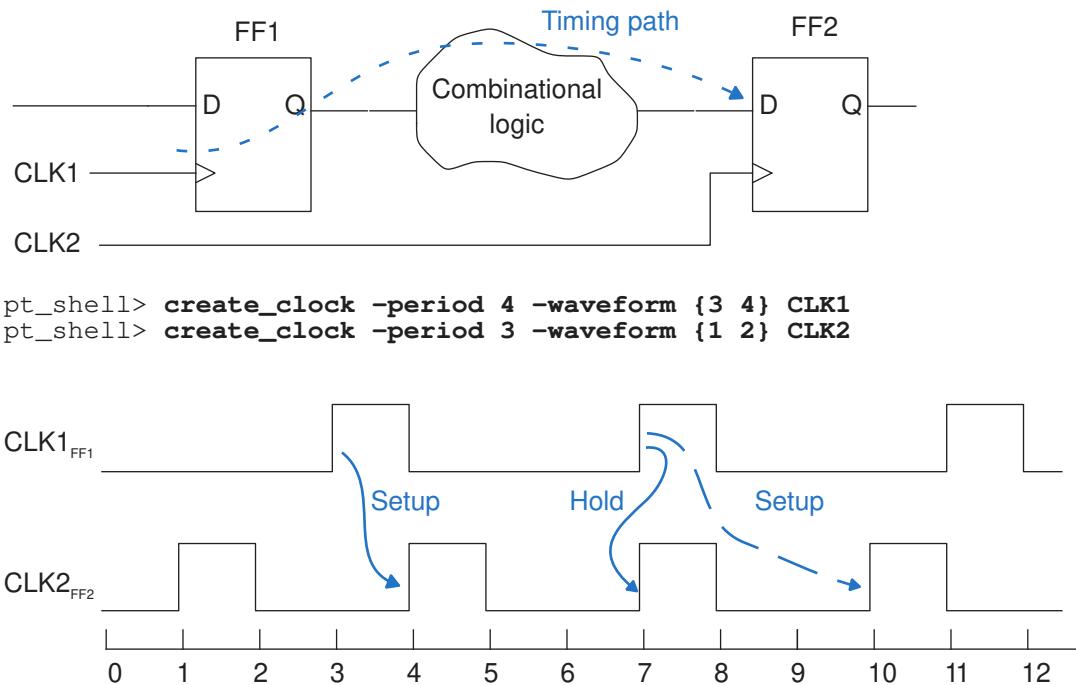
In the following figure, CLK1 has a period of 20 ns and CLK2 has a period of 10 ns. The most restrictive setup relationship is the launch edge at time=0 to the capture edge at time=10. The most restrictive hold relationship is the launch edge at time=0 to the capture edge at time=0.

Figure 80 Delay Requirements With 20ns/10ns Clocks



In the following figure, 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 81 Delay Requirements With 4ns/3ns Clocks



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 12.0 -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 for all timing paths. PrimeTime issues a UITE-217 warning message when you specify an invalid startpoint or endpoint.

To set a timing endpoint for a maximum delay constraint without forcing an endpoint for other timing paths, use the `-probe` option:

```
pt_shell> set_max_delay 4.5 -to I_TOP/MEM/U133/Z -probe
Information: Forcing pin 'I_TOP/MEM/U133/Z' to be a timing endpoint only
for this probe-type timing exception. Existing timing paths are not
affected. (UITE-618)
```

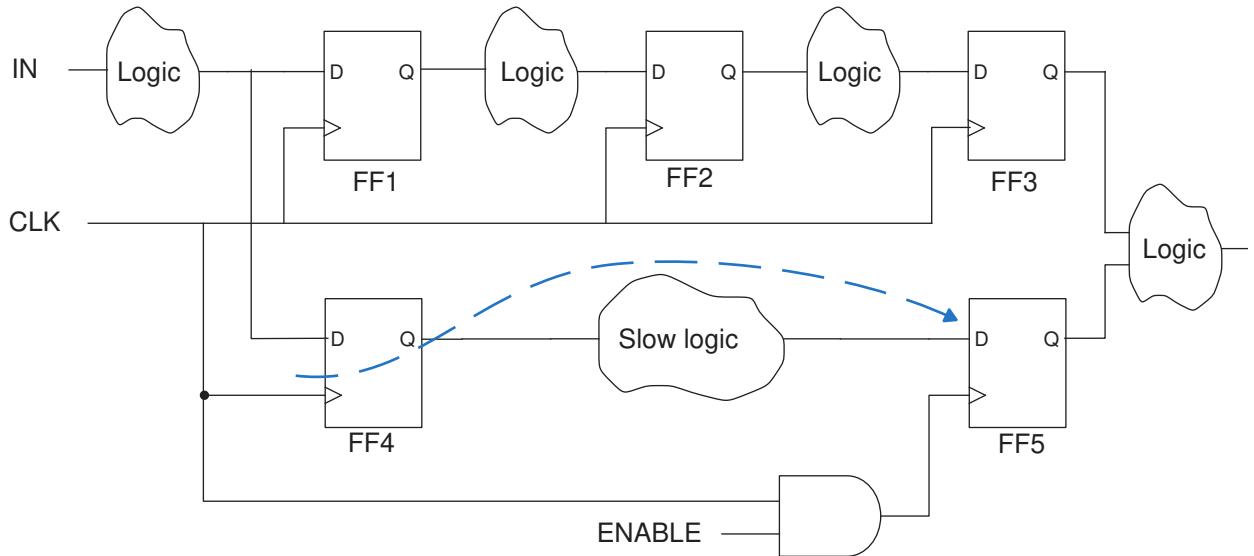
If you specify a sequential cell as the argument of the `-from` option, the command is automatically expanded to apply to each clock input pin of the cell. Similarly, if you specify a sequential cell as the argument of the `-to` option, the command is automatically expanded to apply to each data input pin of the cell. You can see the expanded pin-by-pin exceptions by using the `report_exceptions` or `write_sdc` command. The expansion of an exception set on a cell to the individual pins of the cell does not occur in some other tools such as IC Compiler II. In those tools, the exception is maintained at the cell level. This can cause different tools to resolve overlapping exceptions in different ways.

Setting Multicycle Paths

To specify the number of clock cycles required to propagate data from the start of a path to the end of the path, use the `set_multicycle_path` command. PrimeTime calculates the setup or hold constraint according to the specified number of cycles.

In the following example, 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 82 Multicycle Path Example



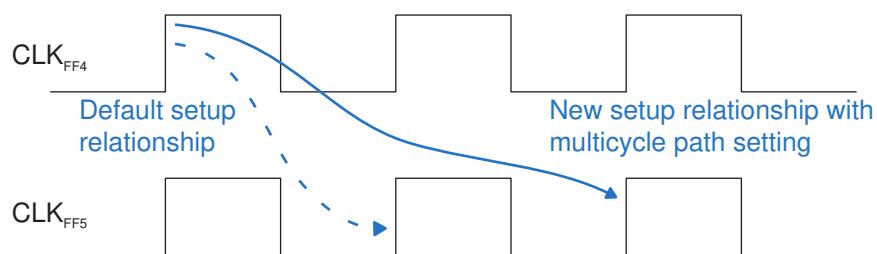
Although you could specify an explicit maximum delay value with the `set_max_delay` command, it is better to use the `set_multicycle_path` command, which automatically adjusts the maximum delay value if you change the clock period.

To set the multicycle path for the preceding example, use this command:

```
pt_shell> set_multicycle_path -setup 2 \
    -from [get_cells FF4] -to [get_cells FF5]
```

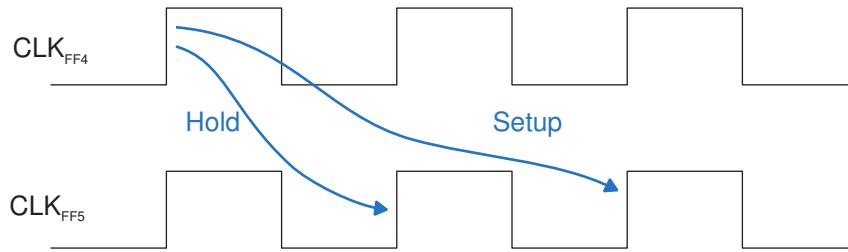
This command specifies that the path takes two clock cycles rather than one, establishing the new setup relationship shown in [Figure 83](#). The second capture edge (rather than the first) following the launch edge becomes the applicable edge for the end of the path.

Figure 83 Multicycle path setup relationship



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 the following figure.

Figure 84 Multicycle path hold relationship based on new setup



The new hold relationship 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.

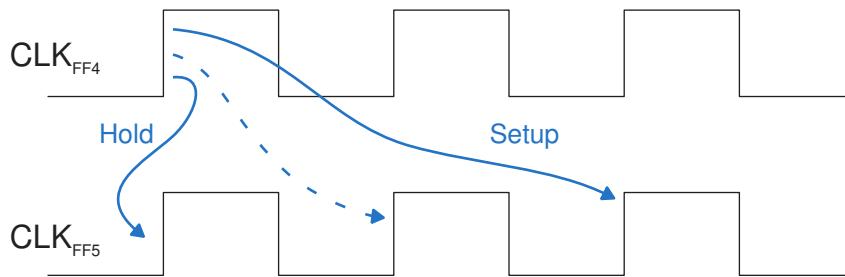
Although you could use the `set_min_delay` command to specify an explicit hold time, it is better to use another `set_multicycle_path` command to move the capture edge for the hold relationship backward by one clock cycle. For example:

```
pt_shell> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]

pt_shell> set_multicycle_path -hold 1 \
           -from [get_cells FF4] -to [get_cells FF5]
```

[Figure 85](#) 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 85 Multicycle path hold set correctly

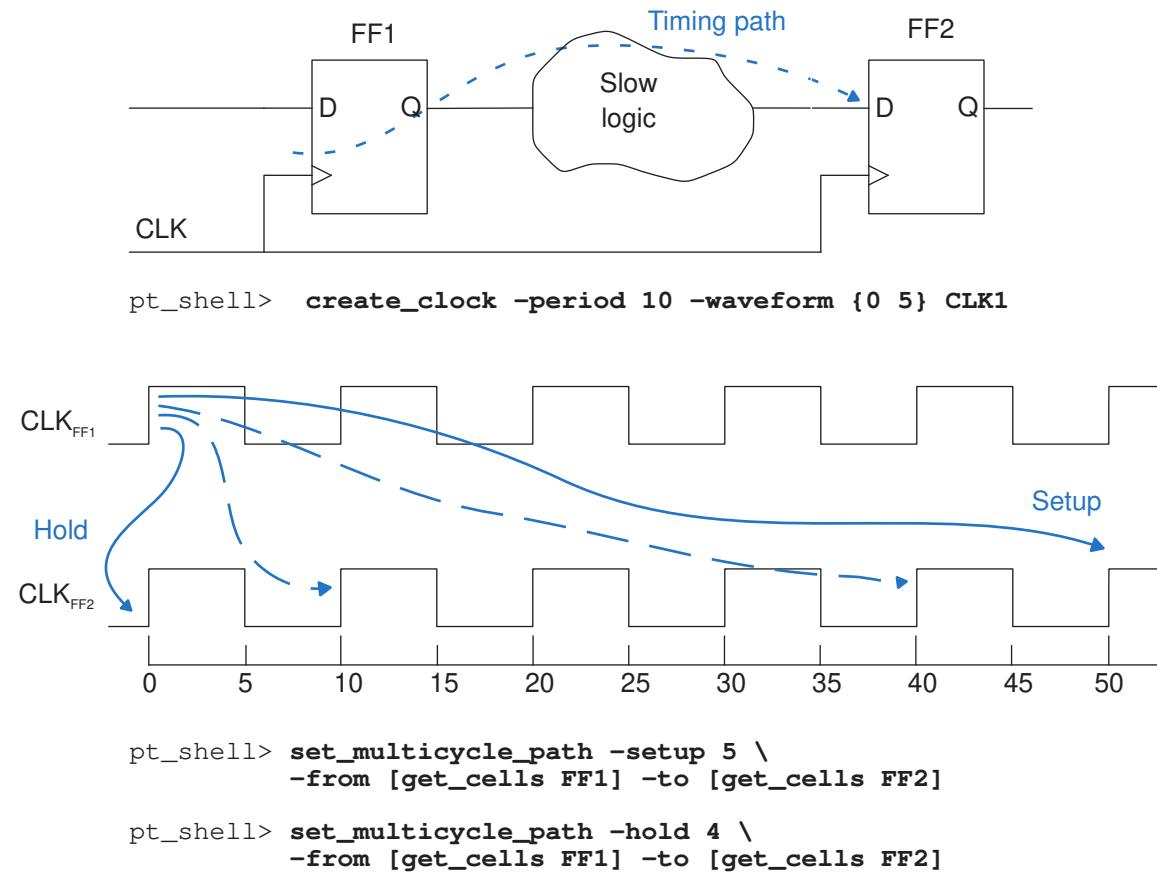


PrimeTime interprets the `-setup` and `-hold` options in the `set_multicycle_path` command differently:

- The integer value for the `-setup` option 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` option 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 is 0.

The following figure 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 86 Multicycle path taking five clock cycles



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 85, hold cycles = $2 - 1 - 1 = 0$. For Figure 86, hold cycles = $5 - 1 - 4 = 0$.

You can optionally specify that the multicycle path exception apply only to rising edges or only to 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 Timing Margins

You can make any timing check more or less restrictive by a specified amount of time by using the `set_path_margin` command. For example,

```
pt_shell> set_path_margin -setup 1.2 -to I_BL/reg30a/D
```

This example makes all setup checks arriving at a specified register pin more restrictive by 1.2 time units. The reported setup slack is 1.2 time units less (more negative) than without the path margin. In a report generated by the `report_timing` command, the margin appears as a “path margin” adjustment in the “data required time” calculation.

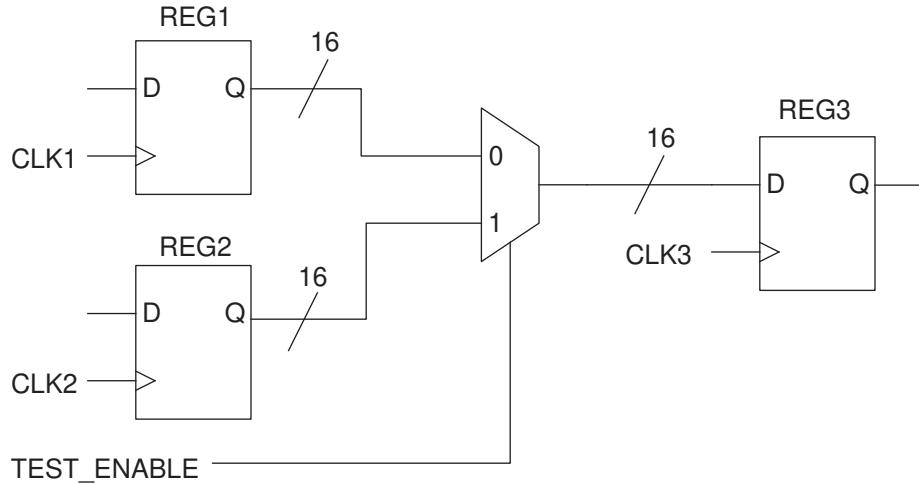
For all types of timing paths, specifying a positive margin results in a more restrictive or tighter check. Conversely, specifying a negative margin results in a less restrictive or looser check.

The path margin setting is a timing exception, much like an exception set with the `set_max_delay` or `set_multicycle_path` command. The specified margin applies to a timing check in addition to any minimum-delay, maximum-delay, and multicycle path exceptions that apply to the same 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. In the following figure, 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 87 Multiplexed register paths



To prevent timing analysis from REG2 to REG3, use one 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.

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.

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.

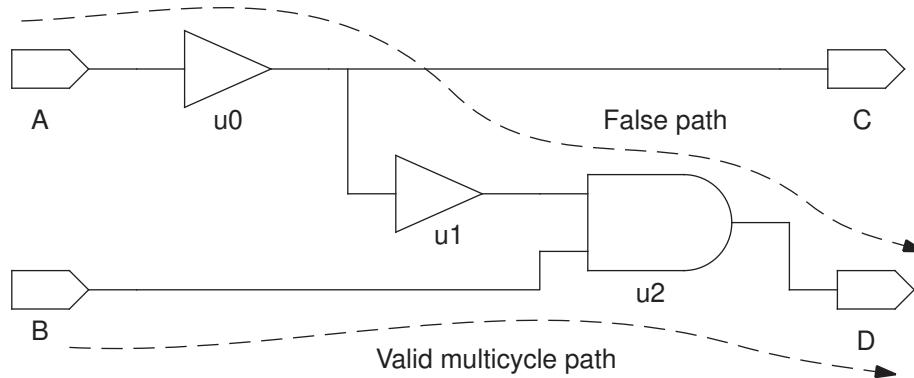
The `report_exceptions` command causes a complete timing update, so be sure to use it only after you have set up all timing assertions and you are ready to receive the report.

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 88](#) and the following exception-setting and exception-reporting commands.

Figure 88 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` to 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
*		*	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. The following table shows the timing path attributes along with their possible values, in relation to unconstrained paths:

Table 15 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.
fanin_of_disabled		Path ending at a fanin of a disabled timing arc or component. The internal pin is either without constraints, 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.

Table 15 Timing path attributes and values (Continued)

Timing path attribute	Reason code	Reason code description
startpoint_unconstrained_reason	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_path` 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
-----
lr6_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
```

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.

The following examples show exception reports that contain the source file and line number information.

Example 10 Exceptions Report

```
pt_shell> 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 11 Timing Report With the -exceptions Option

```
pt_shell> report_timing -exceptions all -from a[1] -to lar5_ff1/D
...
The dominant exceptions are:
From      To          Setup      Hold     Ignored
-----
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 specified but is not accepted by PrimeTime is called an *ignored exception*. PrimeTime ignores an exception for the following reasons:

- 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.

To report exceptions that are fully and partially valid, use the `report_exceptions` command. 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 `report_exceptions -ignored` command.

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
```

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.

To control the scope of exception removal, use the `reset_path` command with options such as `-from`, `-to`, `-through`, `-rise_from`, or `-fall_to`. 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 the preceding 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.

To reset all exceptions set on a path before applying a new exception, use the exception-setting command with the `-reset_path` option. 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, 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).

Saving and Restoring Timing Path Collections

You can save a timing path collection to a disk directory and restore the collection in a new tool session. Saving and restoring a path collection is much faster than regenerating the collection with a new `get_timing_paths` command.

For example,

```
pt_shell> set my_paths [get_timing_paths -nworst 20 -max_paths 500]
...
pt_shell> save_timing_paths $my_paths -output paths1
...
```

The `save_timing_paths` command must specify a timing path collection and the name of a directory in which to store the collection.

You can quickly restore the collection in another PrimeTime session by using the `restore_timing_paths` command:

```
pt_shell> set my_paths [restore_timing_paths paths1]
...
pt_shell> report_timing $my_paths
```

You must specify the name of the directory where the path collection was stored. The design, constraints, and PrimeTime tool version must be the same as when the collection was stored.

For information about analyzing the paths in a collection using the GUI, see [Analyzing Timing Path Collections](#).

9

Operating Conditions

Semiconductor device parameters vary with process, voltage, and temperature (PVT) conditions. The ASIC vendor typically characterizes the device parameters 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](#)
 - [Derating Timing Delays](#)
 - [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.

Operating condition	Description
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 documentation.

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 documentation.

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;
```

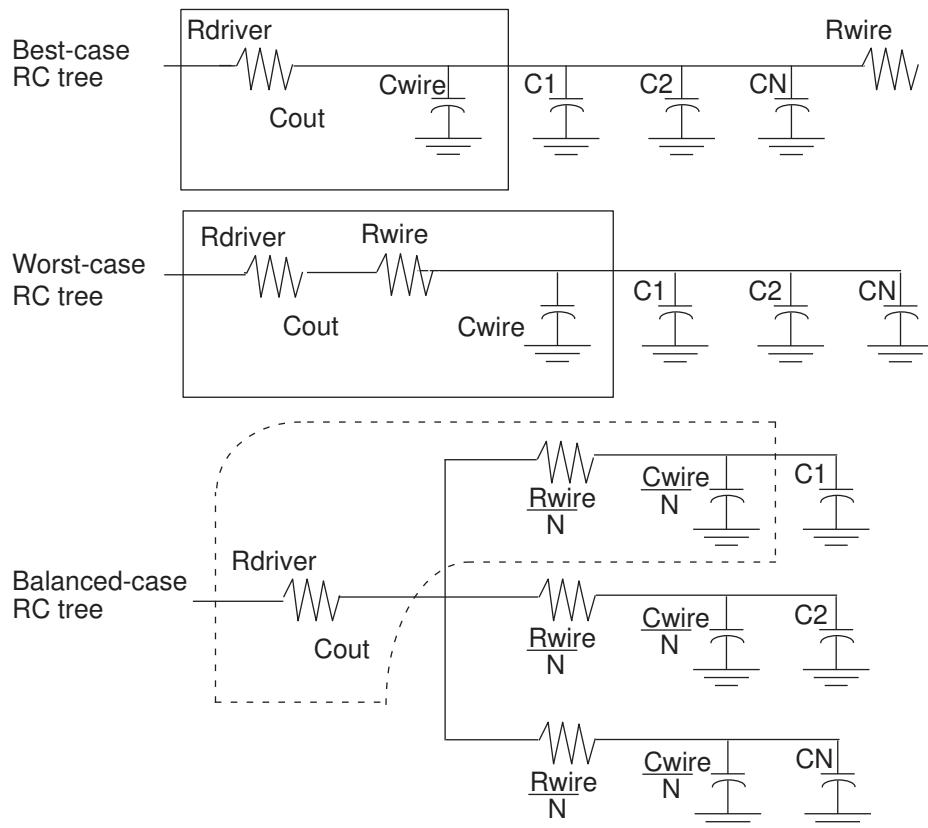
```

        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. The following figure shows the tree type model networks.

Figure 89 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

Command	Action
<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.
- Parametric on-chip variation (POCV) mode – Calculates delay, required time, and slack values as statistical distributions.

The following table 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 16 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)

Table 16 Timing Parameters Used for Setup and Hold Checks (Continued)

Analysis mode	Timing check	Launch clock path	Data path	Capture clock path
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)

The following table shows the usage of minimum and maximum delays from SDF triplet data.

Table 17 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

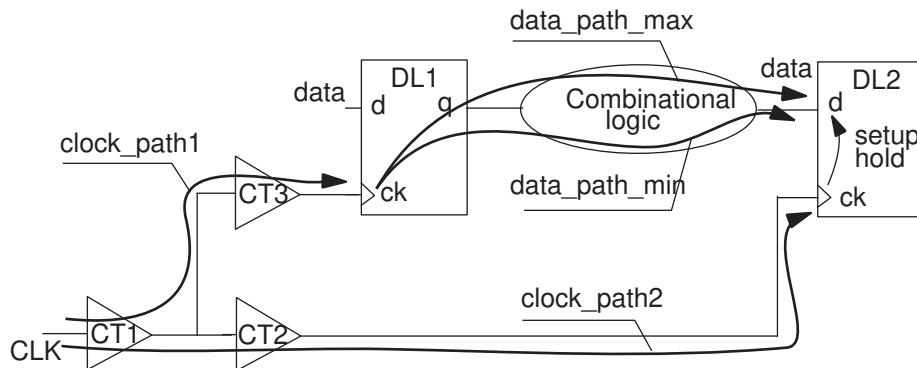
For examples of how setup and hold timing checks are done for a single operating condition and for OCV, see

- [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

The following figure shows how PrimeTime performs setup and hold checks.

Figure 90 Design Example



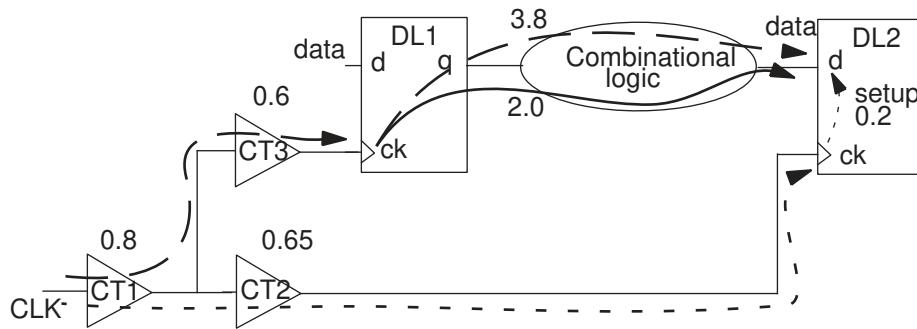
- The setup timing check from pin DL2/ck to DL2/d considers
 - Maximum delay for clock_path1
 - Maximum delay for data path (data_path_max)
 - Minimum delay for clock_path2
- The hold timing check from pin DL2/ck to DL2/d considers
 - Minimum delay for clock_path1
 - Minimum delay for data path (data_path_min)
 - 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

The following figure shows how cell delays are computed for worst-case conditions. To simplify the example, the net delays are ignored.

Figure 91 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

$$\text{clockpath1} = 0.8 + 0.6 = 1.4$$

$$\text{datapathmax} = 3.8$$

$$\text{clockpath2} = 0.8 + 0.65 = 1.45$$

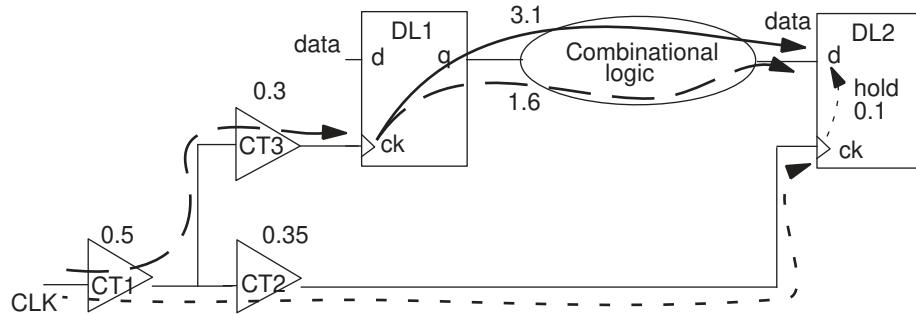
$$\text{setup} = 0.2$$

The clock period must be at least $1.4 + 3.8 - 1.45 + 0.2 = 3.95$.

Hold Timing Check for Best-Case Conditions

The following figure shows how cell delays are computed for best-case conditions.

Figure 92 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

$$\text{clockpath1} = 0.5 + 0.3 = 0.8$$

$$\text{datapathmax} = 1.6$$

$$\text{clockpath2} = 0.5 + 0.35 = 0.85$$

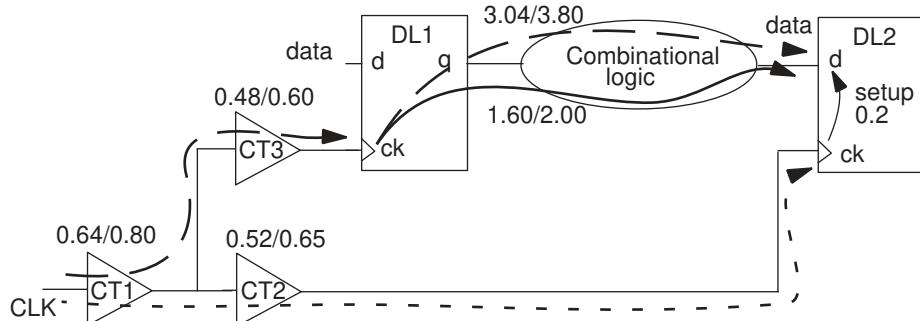
$$\text{hold} = 0.1$$

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 the following figure, 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 93 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

$$\text{clockpath1} = 0.80 + 0.60 = 1.40 \text{ (at 100% of worst case)}$$

$$\text{datapath_max} = 3.80 \text{ (at 100% of worst case)}$$

$$\text{clockpath2} = 0.64 + 0.52 = 1.16 \text{ (at 80% of worst case)}$$

$$\text{setup} = 0.2$$

$$\text{The clock period must be at least } 1.40 + 3.80 - 1.16 + 0.2 = 4.24$$

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

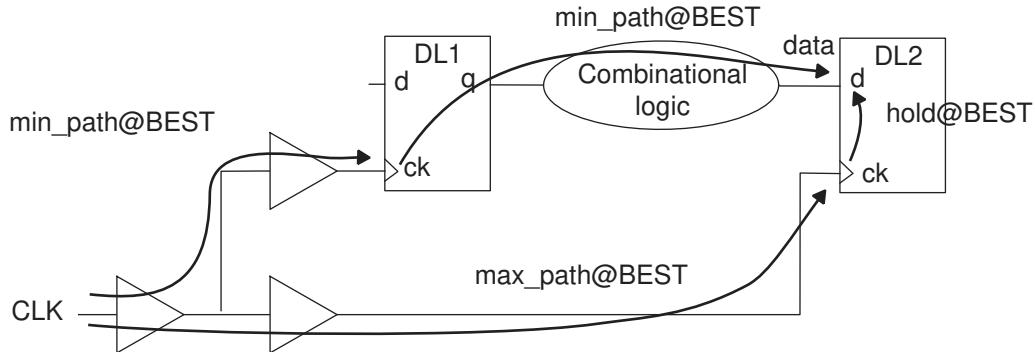
- [Single Operating Condition Analysis](#)
- [On-Chip Variation Analysis](#)

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
```

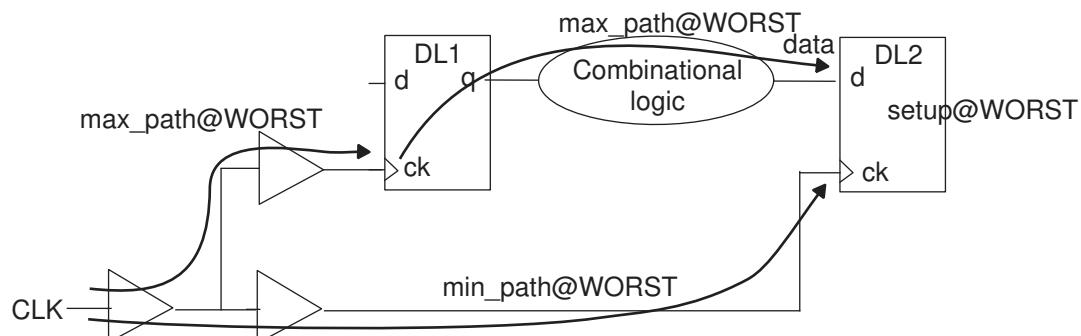
Figure 94 Timing path for one operating condition – best case



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 95 Timing path for one operating condition – worst case



On-Chip Variation Analysis

To perform OCV analysis,

1. Specify the operating conditions:

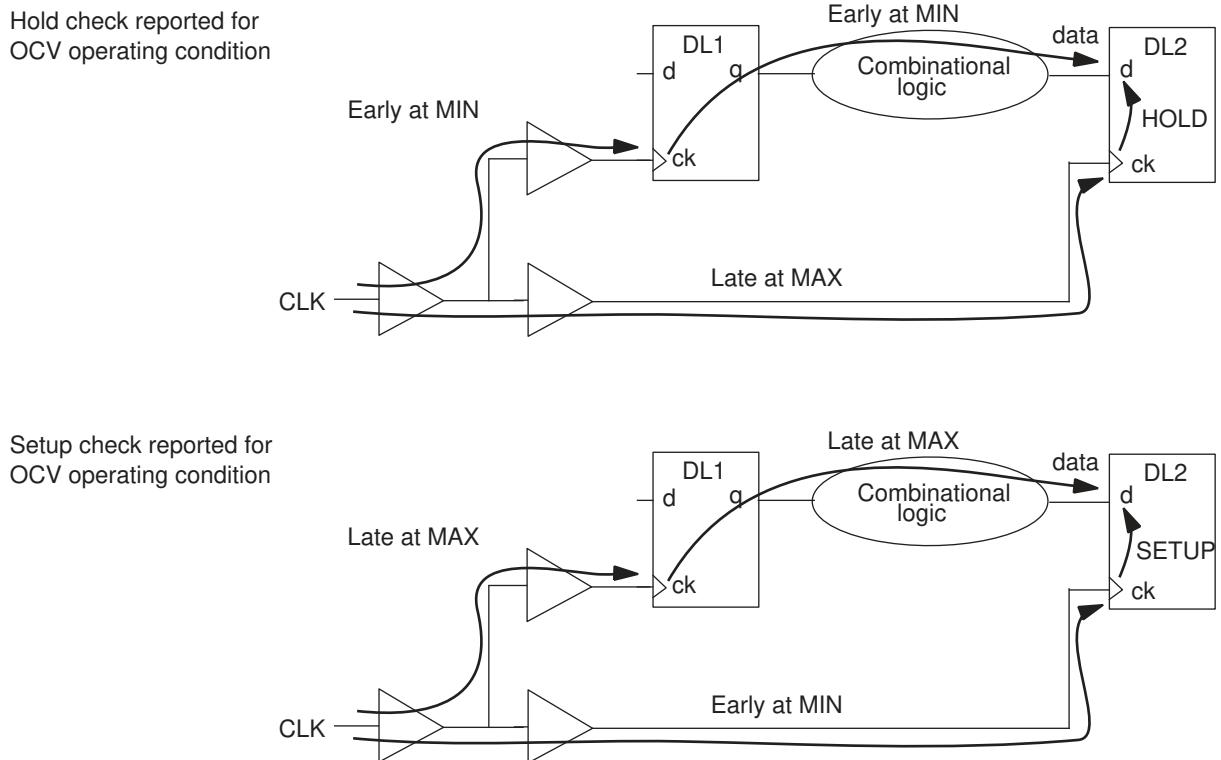
```
pt_shell> set_operating_conditions \
           -analysis_type on_chip_variation \
           -min MIN -max MAX
```

2. Report the timing for setup and hold analysis:

```
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

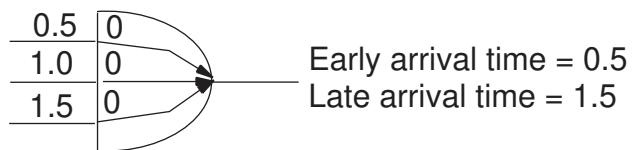
The following figure shows the paths reported for OCV analysis.

Figure 96 Timing path reported during OCV analysis



In the following figure, 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 97 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.

Derating Timing Delays

You can derate calculated delays to model the effects of on-chip variation. Derating multiplies the calculated delays by a factor that you specify, which changes the delay and slack values reported by the `report_timing` command and other reporting commands.

The `set_timing_derate` command specifies the early or late delay adjustment factor and optionally the scope of the design affected by derating. For example,

```
pt_shell> set_timing_derate -early 0.9
pt_shell> set_timing_derate -late 1.2
```

The first command decreases all early (shortest-path) cell and net delays by 10 percent, such as those in the data path of a hold check. The second command increases late (longest-path) cell and net delays by 20 percent, such as those in the data path of a setup check. These adjustments result in a more conservative analysis.

Using the `set_timing_derate` command implicitly changes the tool to on-chip variation mode, if not already in that mode, like using the following command:

```
pt_shell> set_operating_conditions -analysis_type on_chip_variation
```

In on-chip variation mode, the tool applies worst-case delay adjustments, both early and late, at the same time. For example, for a setup check, it applies late derating on the launch clock path and data path, and early derating on the capture clock path. For a hold check, it does the opposite.

In the `set_timing_derate` command, you must use either `-early` or `-late` to specify either shortest-path or longest-path delays for applying the derating. To set both early and late derating, use two separate commands.

You must also specify the derating factor, a floating-point number. To get a more conservative analysis that considers the effects of on-chip variation, use a derating factor less than 1.0 for early derating or greater than 1.0 for late derating.

To report derating that has been set, use the `report_timing_derate` command.

To report the derating factor used for each incremental delay in a timing report, use the `-derate` option in the `report_timing` command:

```
pt_shell> report_timing -derate
...
Path Type: max
```

Point	Derate	Incr	Path
clock PCI_CLK (rise edge)	0.000	0.000	
clock network delay (propagated)	1.802	1.802	
I_ORCA_TOP/I_PCI_CORE/pad_en_reg/CP (sdcrql)	0.000	1.802 r	
I_ORCA_TOP/I_PCI_CORE/pad_en_reg/Q (sdcrql)	1.200	0.485 &	2.287 r
U7/I (inv0d1)	1.200	0.010 &	2.297 r
U7/ZN (inv0d1)	1.100	0.063 &	2.360 f
U62/I (inv0d1)	1.100	0.005 &	2.365 f
U62/ZN (inv0d1)	1.200	0.046 &	2.410 r
U63/A2 (or02d7)	1.200	0.005 &	2.415 r
U63/Z (or02d7)	1.200	0.318 &	2.733 r
...			

To cancel all derating, use the `reset_timing_derate` command.

Derating Options

By default, the `set_timing_derate` command applies the specified early or late derating factor to all early or late timing paths in the whole design. You can optionally modify the scope of the command in several ways:

- To derate only clock paths or only data paths, use the `-clock` or `-data` option.
- To derate only net delays or only cell delays, use the `-net_delay` or `-cell_delay` option.
- To derate only rising-edge delays, use the `-rise` option. The derating applies only to cell and net delays that end with a rising transition. Similarly, to derate only falling-edge delays, use the `-fall` option.
- To derate only certain nets, cell instances, or library cells, enter an object list in the command. You can use an embedded `get_nets`, `get_cells`, or `get_lib_cells` command to generate the list.
- To derate only the static or dynamic component of net delay, use the `-net_delay` option together with the `-static` or `-dynamic` option. By default, both the static and dynamic components are derated. The static component is the net delay without considering crosstalk. The dynamic component is the change in net delay (delta delay) caused by crosstalk.

- To derate cell timing check constraints instead of delays, use the `-cell_check` option. Cell timing check constraints are the cell hold and removal time requirements for early derating, or the cell setup and recovery time requirements for late derating. By default, cell timing checks are not derated.
- To apply derating to advanced on-chip variation (AOCV) analysis or parametric on-chip variation (POCV) analysis, use the `-aocvm_guardband`, `-pocvm_guardband`, or `-pocvm_coefficient_scale_factor` option. By default, the `set_timing_derate` command affects ordinary cell and net delays, not AOCV or POCV analysis.

To report the derating options that have been set, use the `report_timing_derate` command. For example,

```
pt_shell> report_timing_derate -significant_digits 2
...
----- Clock -----      ----- Data -----
          Rise     Fall      Rise     Fall
          Early    Late    Early    Late   Early   Late
-----
```

design: MYDESIGN								
Net delay static	0.90	1.20	0.90	1.10	0.90	1.20	0.90	1.10
Net delay dynamic	0.90	1.20	0.90	1.10	0.90	1.20	0.90	1.10
Cell delay	0.90	1.20	0.90	1.10	0.90	1.20	0.90	1.10
Cell check	--	--	--	--	--	--	--	--
<hr/>								
net: net198								
Net delay static	0.90	1.30	0.90	1.30	0.90	1.30	0.90	1.30
Net delay dynamic	0.90	1.30	0.90	1.30	0.90	1.30	0.90	1.30

If you set derating on a net or leaf-level cell, the command implicitly matches the derating type to the object type: net derating for a net, or cell derating for a leaf-level cell. Derating set on a net applies to the whole net, even as it crosses into a lower level of hierarchy.

To set derating on a hierarchical cell, you must explicitly specify the type of derating by using the `-net_delay` or `-cell_delay` option (or both), or the `-cell_check` option.

To enable derating of minimum pulse width and minimum period constraints, set the `timing_use_constraint_derates_for_pulse_checks` variable to `true`. In that case, the tool applies the derating factors set with the `-cell_check` and `-late` options of the `set_timing_derate` command.

Another way to derate minimum pulse width and minimum period constraints is to use the `-min_pulse_width` and `-min_period` options of the `set_timing_derate` command. In that case, the `timing_use_constraint_derates_for_pulse_checks` variable must be left set to `false` (the default).

```
# Set min pulse width and min period derating
set_timing_derate -min_pulse_width rise 1.2
set_timing_derate -min_period fall 1.1

# Report derating settings
report_timing_derate -min_pulse_width
```

```
report_timing_derate -min_period

# Report min pulse width and min period
report_min_pulse_width ...
report_min_period ...

# Reset min pulse width and min period derating
reset_timing_derate -min_pulse_width
reset_timing_derate -min_period
```

This feature works with POCV analysis. In the `set_timing_derate` command, use the new options together with the `-pocvm_guardband` and `-pocvm_coefficient_scale_factor` options.

Conflicting Derating Settings

A new `set_timing_derate` command overrides any derating value set previously on the same scope of the design. In case of conflicting settings that overlap in scope, the command with the narrower scope has priority. In the following script example, the earlier commands are more specific in scope than the later ones, so the earlier ones have priority.

```
# Derate a collection of cell instances
set_timing_derate -late -cell_delay 1.4 [get_cells U2*]

# Derate a collection of library cells
set_timing_derate -late -cell_delay 1.3 [get_lib_cells libAZ/ND*]

# Derate all cell delays
set_timing_derate -late -cell_delay 1.2

# Derate the design (all cell and net delays)
set_timing_derate -late -1.1
```

Derating Negative Delays

Delays can be negative in some unusual situations. For example, if a cell's input transition is slow and its output transition is fast, and if the input-to-output delay is very short, the output signal can reach the 50 percent trip point before the input signal, resulting in a negative delay for the cell. A similar situation can occur for a change in net delay caused by crosstalk.

In general, 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:

$$\text{delay_new} = \text{old_delay} * (2.0 - \text{derating_factor})$$

When both the static and dynamic components of a net delay are derated, the two components are derated separately first, and then combined, so that a negative delta delay is properly derated before it is combined with a positive static delay.

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. Automated correction of this inaccuracy is called clock reconvergence pessimism removal (CRPR).

CRPR is enabled by default. You can disable CRPR to save runtime and memory at the cost of accuracy (increased pessimism). Applying CRPR reduces pessimism, so it can only increase the reported slack. Therefore, if the design has no violations with CRPR disabled, it will also have no violations with CRPR enabled.

To disable CRPR, set the `timing_remove_clock_reconvergence_pessimism` variable to `false` before you begin timing analysis. Note that any change in this variable causes a complete timing update.

The tool performs CRPR at the same time as regular timing analysis. The following examples show how pessimism removal works.

On-Chip Variation Example

Consider the following command sequence for running OCV analysis and the corresponding design in the following figure:

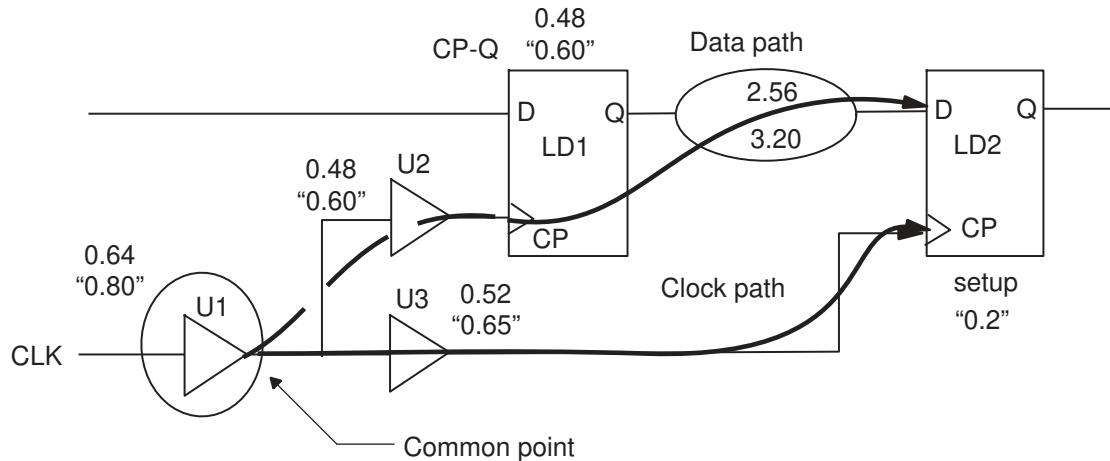
```
pt_shell> set_operating_conditions -analysis_type \
           on_chip_variation -min MIN -max MAX

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 98 Clock reconvergence pessimism example



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.

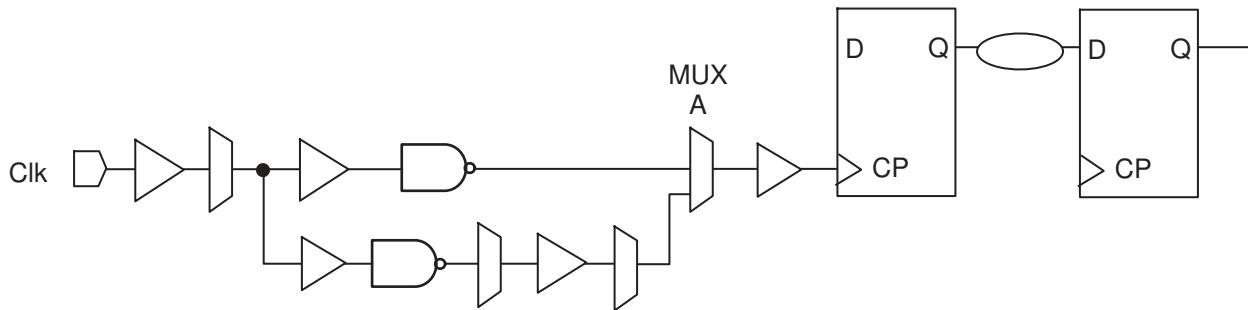
The setup check considers that cell U1 simultaneously has two different delays, 0.64 and 0.80, resulting in a pessimistic analysis in the amount of 0.16. This amount, obtained by subtracting the earliest arrival time from the latest arrival time at the common point, is called the *clock reconvergence pessimism*.

This inaccuracy also occurs in an analogous way for the hold test at the LD2 latch.

Reconvergent Logic Example

The following figure 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 99 Reconvengent 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 increases the accuracy of this checking.

For example, consider the circuit shown in [Figure 100](#). 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 100](#), 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 $+1.4$.

With CRPR enabled, the tool 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

$$\text{crp} = \text{clock reconvergence pessimism}$$

$$\text{Mr} = \text{cumulative maximum rise delay}$$

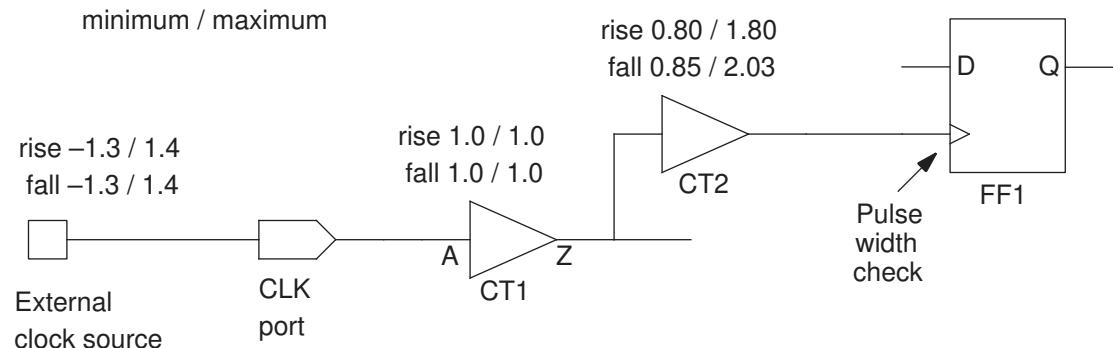
mr = cumulative minimum rise delay

Mf = cumulative maximum fall delay

mf = cumulative minimum fall delay

For an example of this calculation applied to pulse width checking, see the following figure.

Figure 100 Minimum pulse width analysis



$$\text{Minimum rise} = -1.3 + 1.0 + 0.80 = 0.50$$

$$\text{Maximum rise} = 1.4 + 1.0 + 1.80 = 4.20$$

$$\text{Rise range} = 4.20 - 0.50 = 3.70$$

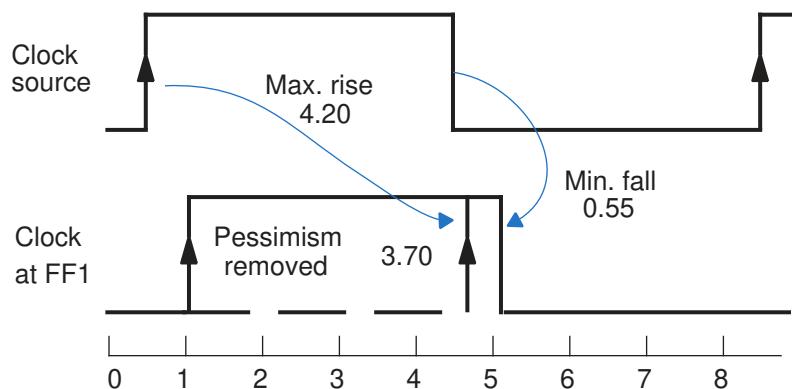
$$\text{Minimum fall} = -1.3 + 1.0 + 0.85 = 0.55$$

$$\text{Maximum fall} = 1.4 + 1.0 + 2.03 = 4.43$$

$$\text{Fall range} = 4.43 - 0.55 = 3.88$$

Clock reconvergence pessimism = smaller of rise range or fall range = 3.70

Minimum pulse width check: maximum rise = 4.20, minimum fall = 0.55



CRPR Reporting

Clock reconvergence pessimism adjustments are reported by the `report_timing`, `report_constraint`, `report_analysis_coverage`, and `report_bottleneck` detailed reports. The following example shows a timing report for the design in [On-Chip Variation Example](#):

```
pt_shell> report_timing -delay_type max
...
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)                         0.00      7.32 r
library setup time                  -0.20      7.12
data required time                  7.12
data arrival time                   -5.20
                                         -5.20

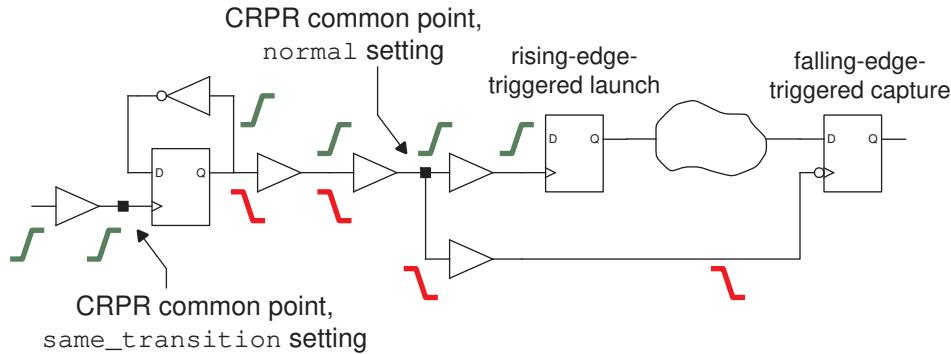
-----  

slack (MET)                          1.92
```

Derating CRPR for Different Transitions at the Common Point

The `timing_clock_reconvergence_pessimism` variable specifies how to find the latest common point in the launch and capture clock paths with respect to mismatching transitions. With the `normal` (default) setting, the tool finds the latest topological common point in the paths, even if the launch and capture transitions are different at that point (one rising, one falling). With the `same_transition` setting, the tool finds the latest point that also shares the same transition type. The following figure shows an example.

Figure 101 Common Point for Different timing_clock_reconvergence_pessimism Settings



The `normal` setting is appropriate when the rising and falling transition arrival times are highly correlated. In situations where they are not highly correlated, the `same_transition` setting ensures a conservative (but possibly pessimistic) analysis.

Different-Transition Derating of CRP

You can use the `normal` (default) common point search method and derate the calculated clock reconvergence pessimism (CRP) when the transitions are different at the common point. This adjusts the pessimism correction for uncorrelated rising and falling transitions, while still allowing usage of a later common point and therefore more pessimism removal.

To use this feature, set the controlling variable as shown in the following example:

```
set_app_var timing_crpr_different_transition_derate 0.90
```

If the launch and capture transitions are different at the common point, this setting causes the CRP value to be reduced to 90 percent of its original calculated value before it is added back to the calculated slack. This results in a more conservative analysis than the default behavior of adding 100 percent of the CRP value.

Different-Transition Derating of POCV CRP Variation

In parametric on-chip variation (POCV) analysis, to derate the variation of the CRP as well as the nominal value, set both the nominal and variation-related control variables:

```
set_app_var timing_crpr_different_transition_derate 0.90
set_app_var timing_crpr_different_transition_variation_derate 0.90
```

These variables have no effect when the launch and capture transitions are of the same type (both rising or both falling) at the common point. They also have no effect when the `timing_clock_reconvergence_pessimism` variable is set to `same_transition`.

The default for the first variable is 1.0 (no derating of the nominal) and for the second variable is 0.0 (full derating or zeroing of the CRP variation). Therefore, by default, random

variation does not contribute to pessimism removal, resulting in a conservative approach for mixed edge types at the common point (for example, in a minimum pulse width check).

Minimum Pulse Width Example

For the minimum pulse width checking example shown in [Minimum Pulse Width Checking Example](#), 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

CRPR Merging Threshold

For computational efficiency, the tool merges multiple points in a path when the CRP differences between adjacent points are smaller than a certain threshold. The `timing_crpr_threshold_ps` variable specifies the threshold in picoseconds. The default is 5, which causes adjacent nodes to be merged when the difference is 5 ps or less.

For a good balance between performance and accuracy, set 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 can use a larger value during the design phase for faster analysis and a smaller value for signoff 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, which are 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]

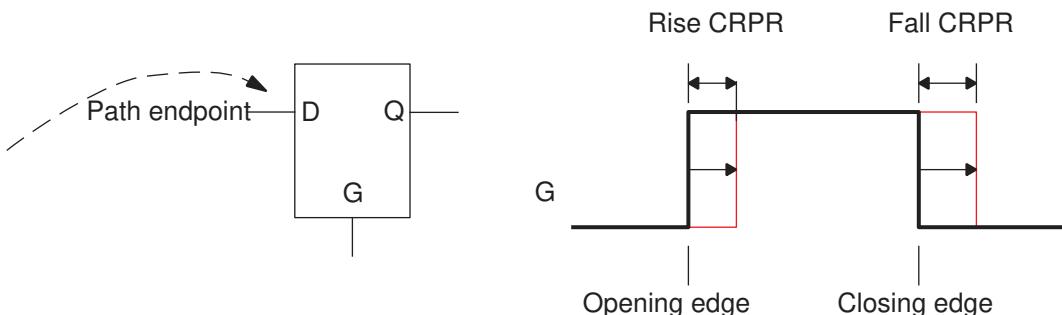
pt_shell> set_clock_latency -source -late 5.5 \
           -dynamic 0.5 [get_clocks CLK]
```

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.

Figure 102 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.

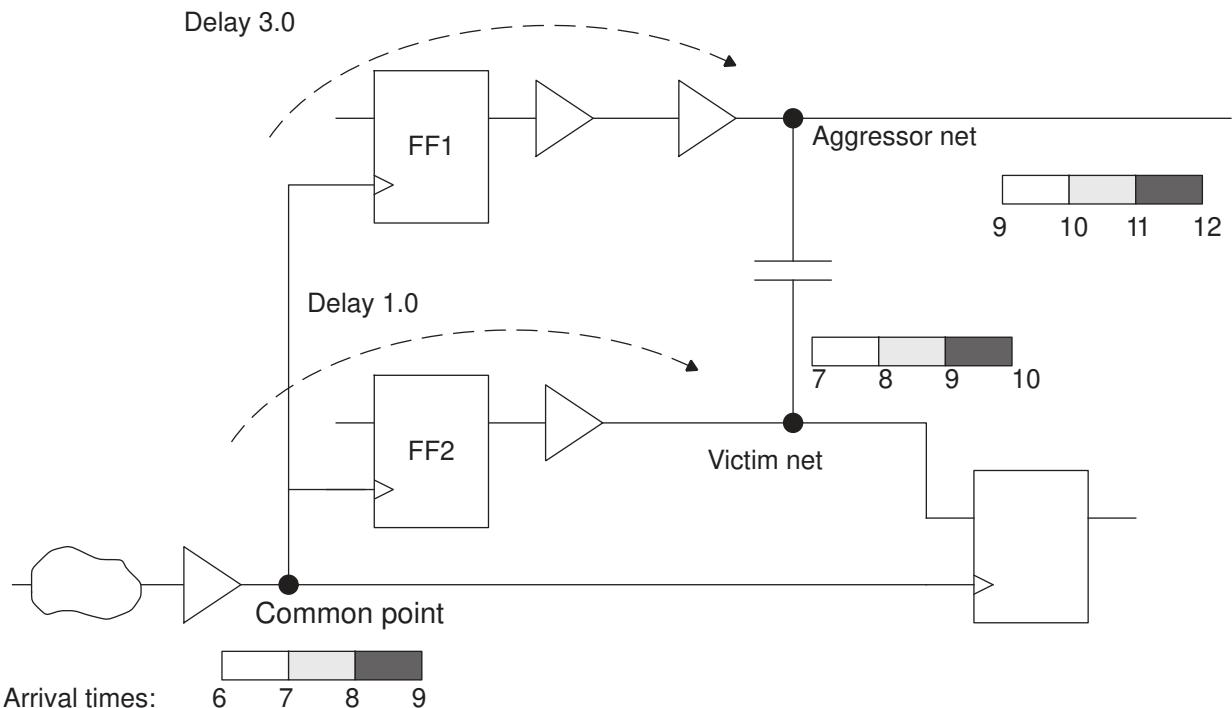
The amount of CRP reported by the `report_crpr` command can be slightly different from the amount reported by the `report_timing` command. This is because the `report_timing` command, for computational efficiency, merges multiple points for CRPR calculations when the CRP differences between adjacent points are too small to be significant. The `timing_crpr_threshold_ps` variable sets the time threshold for merging, which is 5 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.

The following example shows pessimism caused by derating a long clock path. There is cross-coupling capacitance between two wires in the fanout of FF1 and FF2, both of which are clocked by the same clock signal.

Figure 103 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`.

10

Delay Calculation

To perform delay calculation accurately and efficiently, PrimeTime can use 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

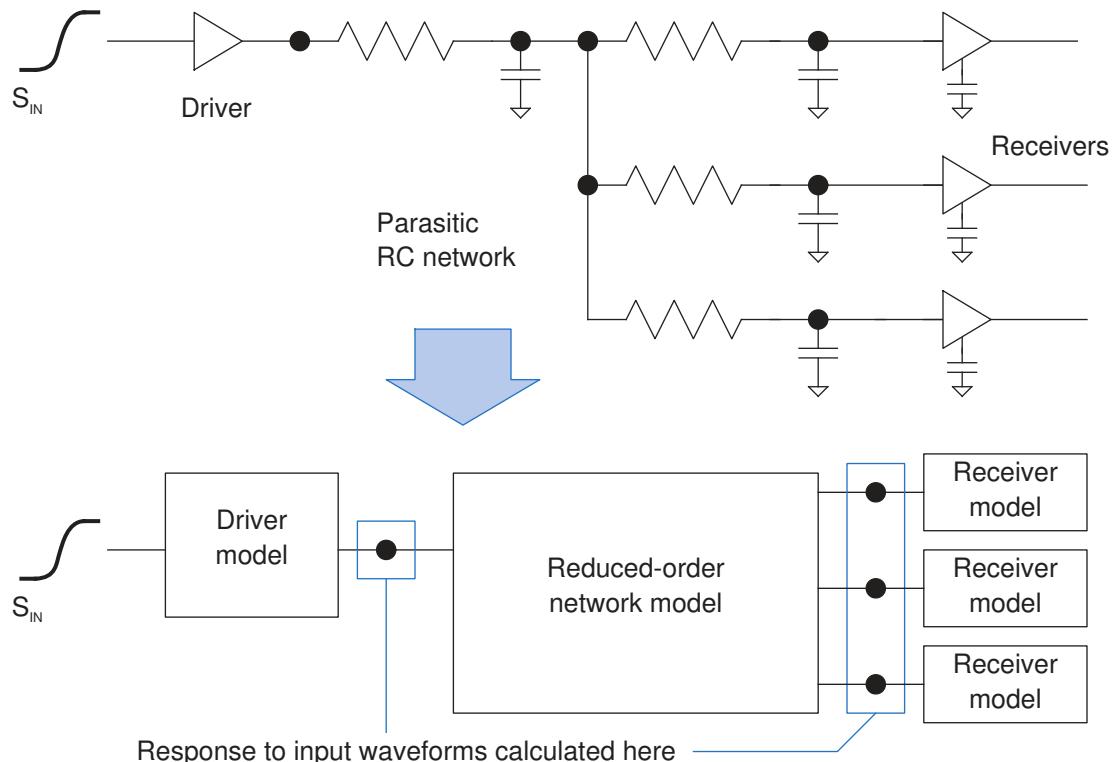
- [Overview of Delay Calculation](#)
 - [Nonlinear Delay Models](#)
 - [Composite Current Source Timing Models](#)
 - [Cross-Library Voltage and Temperature Scaling](#)
 - [Waveform Propagation](#)
 - [Characterization Trip Points](#)
 - [Fast Multidrive Delay Analysis](#)
 - [Parallel Driver Reduction](#)
 - [Multi-Input Switching Analysis](#)
 - [Unit Delay 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. 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 104 Models used to calculate stage delays and slews



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:

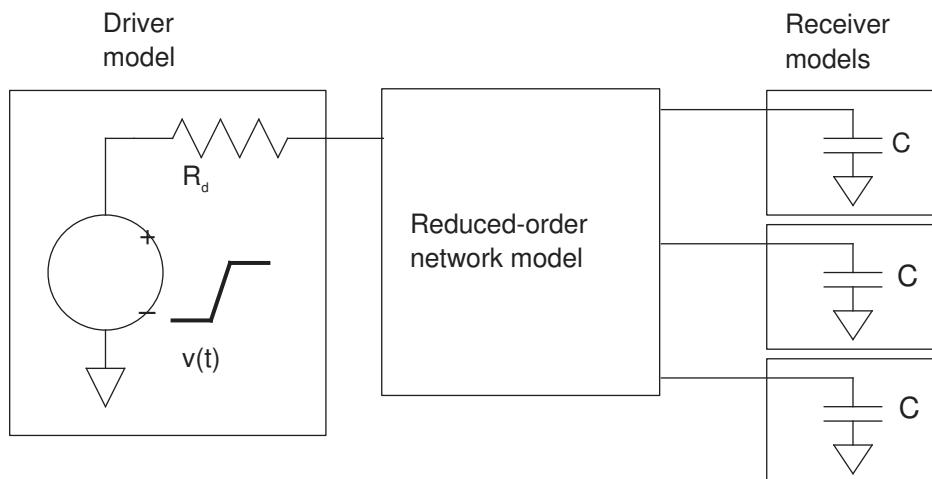
- `basic` – RC delay calculation uses the basic nonlinear delay model (NLD) present in the cell libraries.
- `advanced` (default) – RC delay calculation uses the more advanced Composite Current Source (CCS) timing model, if CCS data is present in the cell libraries; otherwise, the NLD model is used.

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 the following figure. 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 105 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.

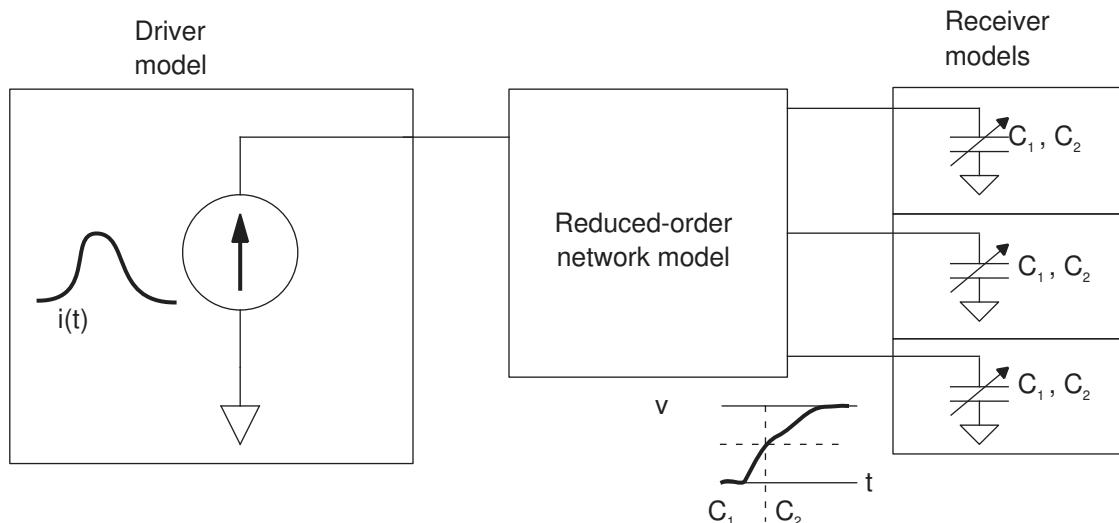
Note:

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 the following figure. The advantage of this driver model is its ability to handle high-impedance nets and other nonmonotonic behavior accurately.

Figure 106 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 "C1C2" 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 the following library information, in the order of highest to lowest precedence, 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, you can determine whether CCS timing data was used for delay calculation by running 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 PrimePower, 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.

By default, the tool handles the extrapolation of CCS timing data by clipping at 10 percent above the maximum library index and at 80 percent of the minimum library index.

To increase the extrapolation range, set the following variable:

```
pt_shell> set_app_var rc_ccs_extrapolation_range_compatibility false
```

With this setting, the tool extrapolates by clipping at 50 percent above the maximum library index and at 50 percent below the minimum library index. This behavior reduces the number of RC-011 warning messages.

Cross-Library Voltage and Temperature Scaling

PrimeTime performs voltage and temperature scaling by interpolating data between libraries 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 those 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 PrimePower
- Nonlinear delay model (NLDM) delay and slew data

Scaling reduces the number of libraries required for the analysis of multivoltage designs and therefore reduces the library characterization effort.

Library groups specify the scaling relationships between multiple libraries. To define a library group for scaling, use the `define_scaling_lib_group` command:

```
pt_shell> define_scaling_lib_group \
           {lib_0.9V.db lib_1.05V.db lib_1.3V.db}
```

There is no limit on the number of libraries in a group. To cover different portions of the design, use this command multiple times to define multiple library groups. However, each library can be part of no more than one library group.

To report the defined scaling library groups, use the `report_lib_groups` command.

To report point-to-point delay calculations, including the libraries used for scaling, use the `report_delay_calculation` command.

To define a library group for the exact-match flow, 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.db lib_1.0V.db lib_1.1V.db}
```

In the exact-match flow, operating conditions must exactly match one of the libraries, and no scaling between libraries is performed.

The following commands invoke library scaling and exact matching:

```
set_operating_conditions
set_voltage
set_temperature
set_rail_voltage
```

For more information on library scaling, see [SolvNetPlus article 000024652, “PrimeTime Multivoltage Scaling Application Note.”](#)

Scaling for Multirail Level Shifter Cells

PrimeTime can perform scaling for multirail cells such as level shifters, which connect driver and load pins of cells belonging to different power domains. Accurate multirail scaling enables PrimeTime to analyze the dependencies on multiple variables, including multiple rail voltages and temperature.

By default, multirail scaling is enabled for timing, noise, and power analysis. The tool automatically applies multirail scaling when there are enough libraries in the scaling library group to support the required scaling formation for the number of rails.

To set up multirail scaling:

1. Set up scaling library groups using the `define_scaling_lib_group` command.

Ensure that there is at least one scaling library group for each scaling situation. 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:

```
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 both voltage and temperature scaling if you are using the on-the-grid formation. All other one-rail cells can be put in one library, and only four libraries are needed to perform both voltage and temperature scaling of these one-rail cells.
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:

```
pt_shell> report_lib_groups -scaling -show {voltage temp process}
...
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
```

Scaling Timing Derates in Voltage Scaling Flows

The `set_timing_derate` command allows you to specify early or late timing adjustment values on different object scopes. In voltage scaling flows, when you apply timing derates on library cells across libraries in a scaling group, the tool can compute scaled derates for cell instances based on their voltage.

For example,

```
define_scaling_lib_group {lib_1.1V.db lib_1.0V.db lib_0.9V.db}

set_timing_derate -late [get_lib_cell lib_1.1V/INV1] 1.07
set_timing_derate -late [get_lib_cell lib_1.0V/INV1] 1.10
set_timing_derate -late [get_lib_cell lib_0.9V/INV1] 1.15
```

To enable voltage scaling of timing derates, set the following variable:

```
pt_shell> set_app_var \
    timing_enable_derate_scaling_for_library_cells_compatibility false
```

For best results, also set the following variable to enable computation of fully scaled (interpolated) derate values:

```
pt_shell> set_app_var \
    timing_enable_derate_scaling_interpolation_for_library_cells true
```

With these settings applied,

- Scaling is performed for both `set_timing_derate` derates and table-based timing derates read in by the `read_ocvm` command.
- If a cell's voltage configuration exactly matches that of a scaling library, that library's derate data is used.
- If a cell's voltage configuration does not exactly match a scaling library, a fully scaled (interpolated) derate computed from nearby scaling libraries is used.
- If a cell has multiple rails, then the worst-case (non-interpolated) derate from nearby scaling libraries is used.

By default, both of these features (derate scaling and interpolation-based derate scaling) are disabled by default. For details, see the man pages.

If you set derating on a library cell in one scaling library, you should also set derating on the same library cell in the other scaling libraries as well. Otherwise, you can have “missing” derates at some voltage conditions. If a voltage-scaled derate cannot be computed for a library cell because of this, the tool uses the next derate setting found in the object scope precedence list (link library derate, then hierarchical cell, then global).

See Also

- [Derating Timing Delays](#) for more information about the `set_timing_derate` command

Excluding Rails From Multirail Scaling Library Groups

The `define_scaling_lib_group` command provides an `-excluded_rail_names` option that allows specified library voltage rails to be ignored by voltage scaling.

These rails are ignored when validating scaling group formations and when performing the scaling calculations themselves, thus reducing N -dimensional multirail scaling to a lower dimension.

This option can be used in true scaling flows to remove problematic rails that prevent valid formations. It can also be used in exact-match flows to remove problematic rails that do not match cell instances.

In the following example, a 2-D scaling group cannot be created with only two libraries (at least three are required), but a 1-D scaling group can be created by ignoring the VDD2 rail:

```
define_scaling_lib_group { \
    slow__VDD1_1.0__VDD2_1.5.db \
    slow__VDD1_2.0__VDD2_1.6.db \
} \
-excluded_rail_names {VDD2}
```

The `-excluded_rail_names` option should be used with care, as it disables consistency checks and scaling algorithms that are normally applied to the analysis.

This option affects all cells in the scaling group libraries. To exclude rails for specific library cells, use the `set_disable_pg_pins` command instead.

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 advanced waveform propagation, which uses a combination of CCS timing and CCS noise models during delay calculation. Waveform propagation supports both graph-based and path-based analysis 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 standard delay calculation.

To use waveform propagation during setup and hold constraint analysis, set the `delay_calc_waveform_analysis_constraint_arcs_compatibility` variable to `false`. By default, the variable is set to `true`, which causes the tool to perform setup and hold constraint analysis using only nonlinear delay model (NLDM) lookup tables.

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 thresholds are defined on specific library pins, the application tools use those. Library pin thresholds override library-level thresholds of the same name.
2. If library pin thresholds are not defined, but library-level thresholds are, the application tools use the library-level thresholds.
3. If neither library 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 107](#) and [Figure 108](#) 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 107 Slew transition points

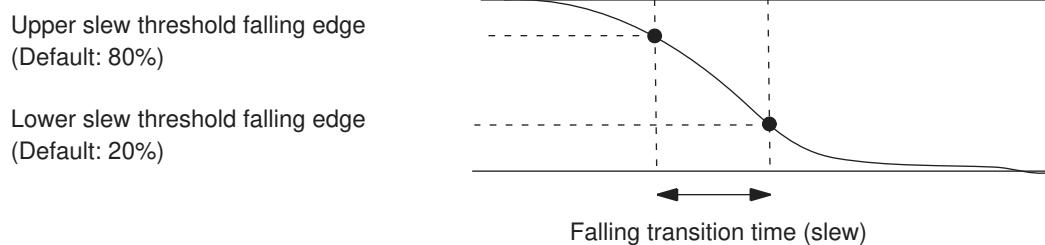
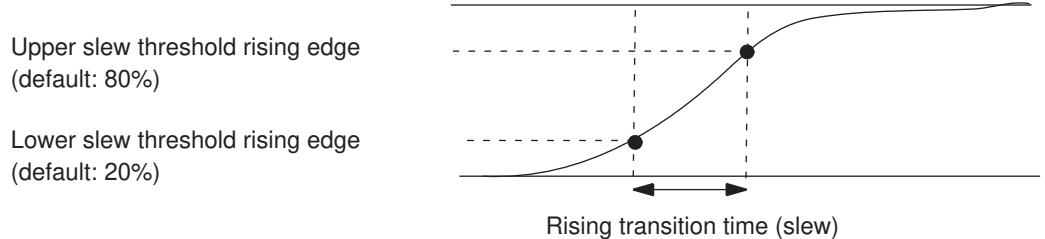
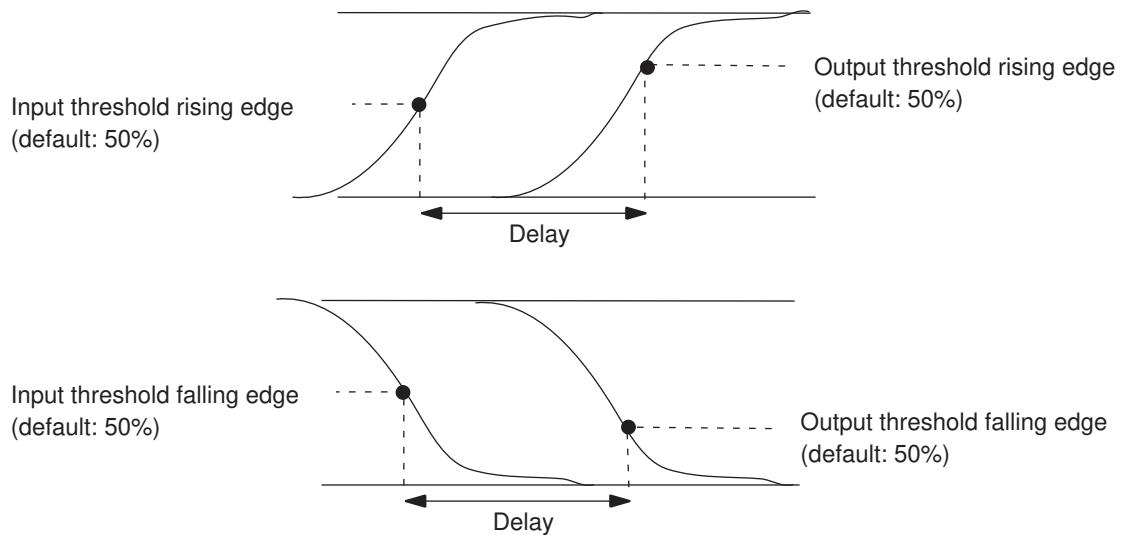


Figure 108 Input/output transition points



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.

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](#)

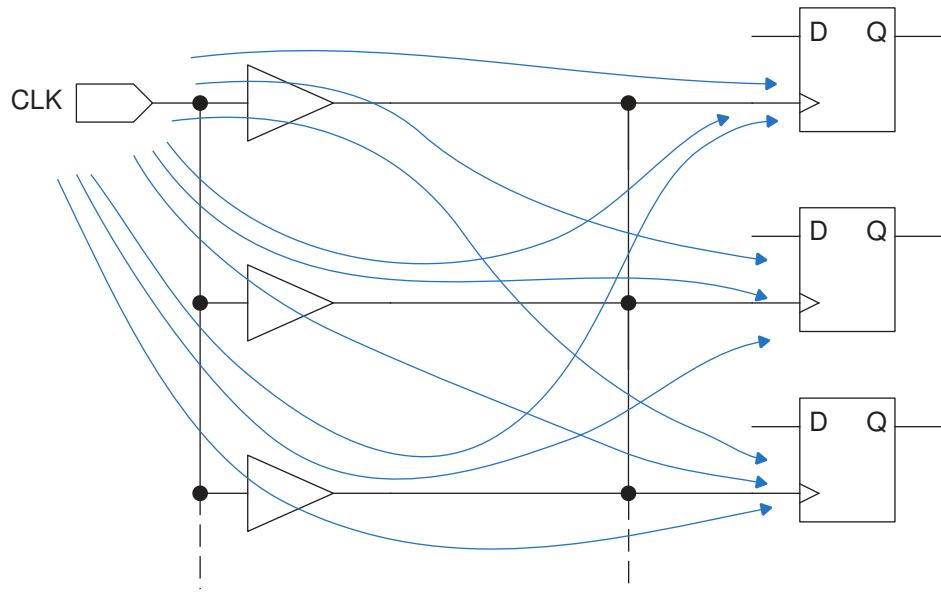
Parallel Driver Reduction

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.

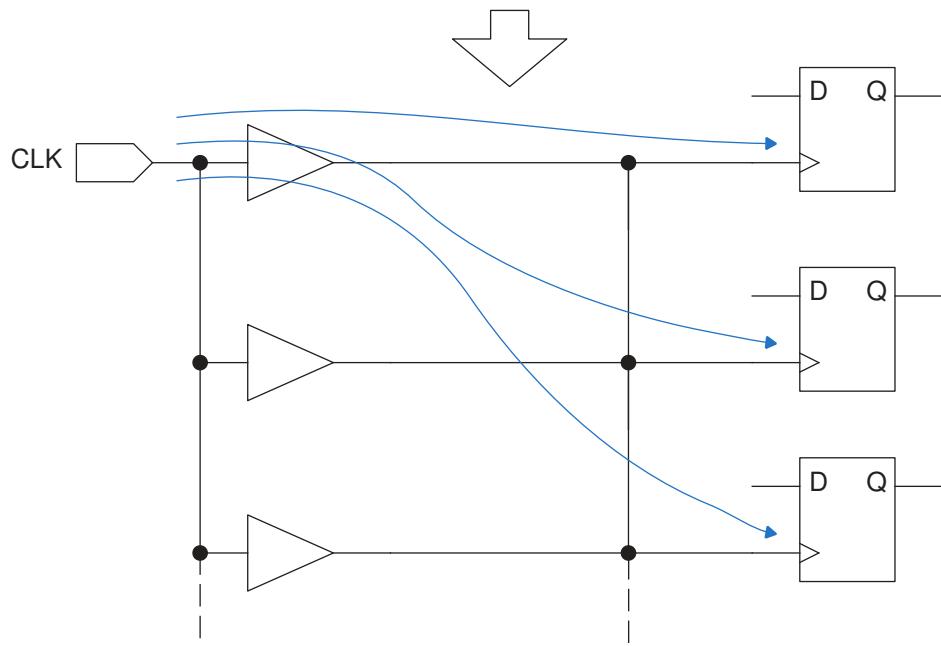
If a design has 1,000 drivers in parallel driving 1,000 loads, PrimeTime must keep track of one million different timing arcs between the drivers and loads. Timing analysis of such a network can consume a large amount of CPU and memory resources.

For better performance, PrimeTime can reduce the number of timing arcs that must be analyzed, as shown in the following figure. If you enable parallel driver reduction, PrimeTime selects one driver in a parallel network and analyzes the timing arcs through that driver only.

Figure 109 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.

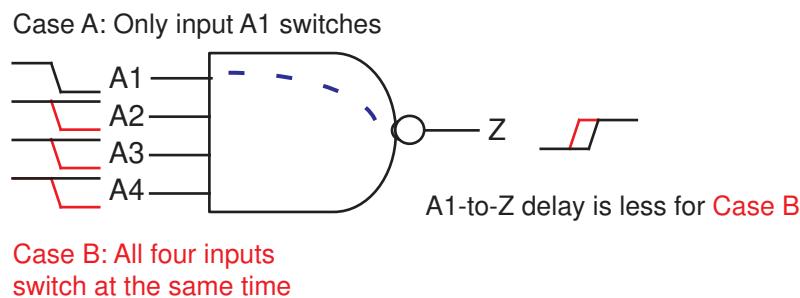
Reducing parallel drivers only affects the timing arcs analyzed by PrimeTime, not the netlist. Therefore, it does not affect the output of the `write_changes` command. For information about RC delay calculation with massively multidriven networks, see [Fast Multidrive Delay Analysis](#).

Multi-Input Switching Analysis

To characterize the input-to-output delay of a combinational logic gate for a library, the characterization tools consider the switching of one input at a time among multiple inputs. However, when multiple inputs switch at the same time, the delay can be different. To increase the timing accuracy in these situations, you can invoke multi-input switching analysis. Using this feature requires a PrimeTime-ADV license.

In the 4-input NAND gate shown in the following figure, an output transition from low to high can be triggered by a single input going low while the other inputs are held high (Case A) or by all inputs going low simultaneously (Case B). In Case A, only one active transistor in the NAND gate pulls up the output, whereas in Case B, four active transistors do the same thing in parallel, so Case B has the shorter input-to-output delay.

Figure 110 Multi-Input Switching Example: 4-Input NAND Gate



The logic library contains only the delay value for single-input switching. However, simultaneous switching can occur in circuits such as memory decoders. If a NAND gate is in a data path of such a circuit, a hold check is optimistic for the simultaneous switching case because the true delay is less. Multi-input switching analysis can account for this effect.

For the opposite situation where the output of the NAND gate goes from high to low, multi-input switching results in longer delays than single-input switching, which can affect setup checking. However, setup-critical data paths are usually long, with many gates, so a small increase in one gate delay has a negligible overall effect. Conversely, hold-critical

data paths are usually short, where a small decrease in delay can be important. For this reason, multi-input switching analysis applies only to minimum-delay (hold) checking.

Multi-Input Switching Analysis Modes

Multi-input switching analysis provides three analysis modes: `lib_cell`, `lib_arc`, and `advanced`.

Library Cell User-Defined Coefficients

The `lib_cell` analysis mode allows you to define delay-scaling coefficients on library cells. These coefficients are defined by the `set_multi_input_switching_coefficient` command.

For example,

```
pt_shell> set_multi_input_switching_coefficient \
           [get_lib_cell lib1/NAND4gx] -rise -delay 0.683
```

To determine the correct delay adjustment factor for each multi-input combinational cell and output transition type, use the HSPICE tool or other simulation tool to find the actual reduced delays when all inputs switch simultaneously.

Library Timing Arc User-Defined Coefficients

The `lib_arc` analysis mode allows you to define delay-scaling coefficients on individual library cell timing arcs. These coefficients can be provided in a library or applied by a script file.

This mode allows more detailed coefficient behaviors to be defined, including switching pin group information. For example,

```
cell("AND4X1") {
    pin(Z) {
        direction : output;
        function : "(A1|A2) & (B1|B2)";
        timing() {
            related_pin : A1;
            timing_sense : negative_unate;
            ...
            sdf_cond : "A2 == 1'b1 && B1 == 1'b0 && B2 == 1'b0";
            when : "A2&!B1&!B2";
            mis_factor_rise : 0.85;
            mis_pin_set_rise : "A1 A2";
            mis_factor_fall : 0.8;
            mis_pin_set_fall : "A1 B1 B2";
        }
    }
}
```

For details on these library attributes, see [Defining Library Timing Arc Coefficients](#).

The `lib_arc` analysis mode requires a PrimeTime-ADV-PLUS license.

Advanced Multi-Input Switching Analysis

The `advanced` analysis mode analyzes library cell data, along with actual pin slew, arrival, load, and waveform information, to automatically estimate multi-input switching effects.

User-defined coefficients are not required, but the following library data is required:

- CCS timing models
- CCS noise models (arc-based)
- All `when` conditions characterized

[Table 18](#) shows the supported library cell types.

Table 18 Supported Library Cell Types for Advanced Multi-Input Switching Analysis

AND2	OR2	NAND2	NOR2	AO211	AOI211	OA211	OAI211
AND3	OR3	NAND3	NOR3	AO21	AOI21	OA21	OAI21
AND4	OR4	NAND4	NOR4	AO22	AOI22	OA22	OAI22
				AO31	AOI31	OA31	OAI31

An output pin of a multi-output cell qualifies if (1) its logic function and library data qualify, and (2) its logic function shares no input pins with other output pins.

For library cells that do not meet the requirements, you can apply user-defined coefficients to library cells or library timing arcs with the other two analysis methods.

The `advanced` analysis mode requires a PrimeTime-ADV-PLUS license.

Defining Library Timing Arc Coefficients

The `lib_arc` analysis mode allows you to define delay-scaling coefficients on individual library cell timing arcs. These coefficients can be provided in a library or applied by a script file.

The coefficients are provided using the following attributes defined on library cell timing arcs:

```
mis_factor_rise
mis_factor_fall
```

```
mis_pin_set_rise
mis_pin_set_fall
```

These attributes are not standard Liberty attributes. Thus, to include them in a library, they must be declared as user attributes at the top of the library file:

```
library(mis_arc_based_lib) {
    define(mis_factor_rise, timing, float);
    define(mis_factor_fall, timing, float);
    define(mis_pin_set_rise, timing, string);
    define(mis_pin_set_fall, timing, string);
```

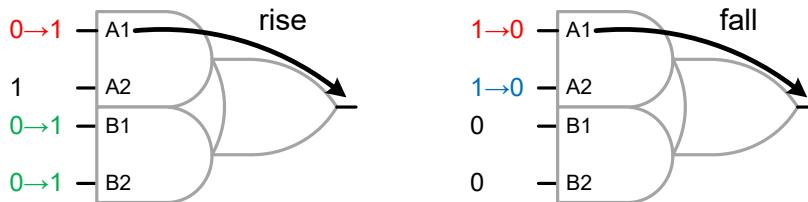
Then in each timing() group, their values can be defined as needed:

```
cell("NAND2X1") {
    pin(ZN) {
        direction : output;
        function : "!A | !B";
        timing() {
            related_pin : A;
            timing_sense : negative_unate;
            ...
            mis_factor_rise : 0.55;
            mis_pin_set_rise : "A B";
        }
        timing() {
            related_pin : B;
            timing_sense : negative_unate;
            ...
            mis_factor_rise : 0.45;
            mis_pin_set_rise : "A B";
        }
    }
}
```

The mis_factor_rise and mis_factor_fall attributes specify the delay-scaling coefficients to use when all pins in the mis_pin_set_rise and mis_pin_set_fall pin set switch, respectively. If fewer pins switch, an intermediate factor is computed as described in [Arrival Window Overlap Checking](#).

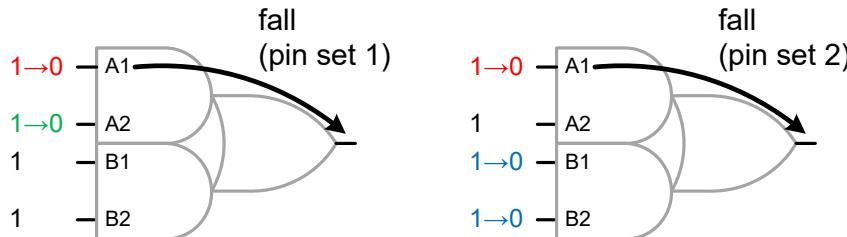
Cells can have both “rise” and “fall” MIS attributes relevant to the same timing arc. For example, an AO22 gate can have different pin sets (and coefficients) for the rising and falling transitions of an arc:

```
when : "A2 & !B1 & !B2"; /* A2==1, B1==0, B2==0 */
mis_pin_set_rise : "A1 B1 B2";
mis_pin_set_fall : "A1 A2";
```



Cells can have multiple pin sets that sensitize MIS effects. For example, the falling $A1 \rightarrow Z$ arc of an AO22 gate might be influenced by side pins from either AND gate also switching. To specify multiple pin sets, separate them with “+” in the attribute value:

```
when : "A2 & B1 & B2"; /* A2==1, B1==1, B2==1 */
mis_pin_set_fall : "A1 A2 + A1 B1 B2";
```



When all pins in any pin set switch together, the full coefficient value applies.

Note:

For MIS analysis, PrimeTime conservatively assumes that `when` conditions could represent pre-transition or post-transition side pin values. The convention of how to assign MIS effects to conditional arcs is dependent on the library characterization tool.

The attributes must be declared as imported user attributes in PrimeTime before the library is loaded, so that they are retained in memory. To do this, run the following commands before the library is loaded:

```
define_user_attribute \
    -type float -class lib_timing_arc -import mis_factor_rise
define_user_attribute \
    -type float -class lib_timing_arc -import mis_factor_fall
define_user_attribute \
```

```
-type string -class lib_timing_arc -import mis_pin_set_rise
define_user_attribute \
-type string -class lib_timing_arc -import mis_pin_set_fall
```

If your library does not contain these attributes, you can define them in a Tcl script file. The script should first define the attributes with the `define_user_attribute` `-import` commands above, then define the attributes on library timing arcs using the `set_user_attribute` and `get_lib_timing_arcs` commands as follows:

```
# include the define_user_attribute commands here
# define_user_attribute ...

# apply per-library-arc coefficient values
set_user_attribute \
    [get_lib_timing_arcs -from NAND2/a -to NAND2/ZN] \
    mis_factor_rise 0.35
set_user_attribute \
    [get_lib_timing_arcs -from NAND2/a -to NAND2/ZN] \
    mis_pin_set_rise {a b}
set_user_attribute \
    [get_lib_timing_arcs -from NAND2/b -to NAND2/ZN] \
    mis_factor_fall 0.40
set_user_attribute \
    [get_lib_timing_arcs -from NAND2/b -to NAND2/ZN] \
    mis_pin_set_fall {a b}
```

For complex cells, you can use the `-filter` option of the `get_lib_timing_arcs` command to filter arcs by `sense`, `when`, or other attributes.

Configuring Multi-Input Switching Analysis

To use multi-input switching analysis,

1. Enable the feature by setting the following variable to `true`:

```
pt_shell> set_app_var \
           si_enable_multi_input_switching_analysis true
```

2. Specify the ordered list of analysis modes to use by setting the `si_multi_input_switching_analysis_mode` variable. For example,

```
pt_shell> # use all analysis modes, preferring advanced where possible
pt_shell> set_app_var \
           si_multi_input_switching_analysis_mode \
           {advanced lib_arc lib_cell}
```

This variable specifies which analysis modes to use, and in what order of precedence (highest first). Valid values are `lib_cell`, `lib_arc`, and `advanced`. If an analysis mode is not in the list, it is not used.

3. If you are using the `lib_cell` analysis mode,

- a. Define scaling coefficients on library cells:

```
pt_shell> set_multi_input_switching_coefficient \
           [get_lib_cell lib1/NAND4gx] -rise -delay 0.683
```

- b. (Optional) Report the applied library cell coefficients:

```
pt_shell> report_multi_input_switching_coefficient
```

4. If you are using the `lib_arc` analysis mode,

- If your library contains `lib_arc` attribute information, define the user-defined attributes in PrimeTime before the libraries are loaded:

```
set_app_var link_path {* lib_with_MIS.db}

define_user_attribute \
    -type float -class lib_timing_arc -import mis_factor_rise
define_user_attribute \
    -type float -class lib_timing_arc -import mis_factor_fall
define_user_attribute \
    -type string -class lib_timing_arc -import mis_pin_set_rise
define_user_attribute \
    -type string -class lib_timing_arc -import mis_pin_set_fall

read_verilog ...
link_design TOP ;# libraries (and attributes) are loaded here
```

- If you are defining `lib_arc` attribute information with a Tcl script file, source the file after the libraries are loaded:

```
set_app_var link_path {* lib_with_MIS.db}

read_verilog ...
link_design TOP ;# libraries (and attributes) are loaded here

source lib_arc_coefficients.tcl
```

For details on how to provide `lib_arc` coefficient information, see [Defining Library Timing Arc Coefficients](#).

5. If you are using the advanced analysis mode,

- ▶ (Optional) Report the library cells that qualify for automatic multi-input switching analysis:

```
pt_shell> report_multi_input_switching_lib_cells
```

Multi-Input Switching Analysis Example

The following script demonstrates library cell multi-input switching analysis.

```
set search_path { . my_lib.db }
set link_path {my_lib.db}
read_verilog design.v
link_design
read_parasitics design.spf
source design.sdc

# Enable multi-input switching analysis
set_app_var si_enable_multi_input_switching_analysis true

# Use the lib_cell analysis mode
set_app_var si_multi_input_switching_analysis_mode {lib_cell}

# Specify delay change factors for library cells
set_multi_input_switching_coefficient \
[get_lib_cell lib1/NAND4gx] -rise -delay 0.683

# Report multi-input switching delay factors applied
report_multi_input_switching_coefficient

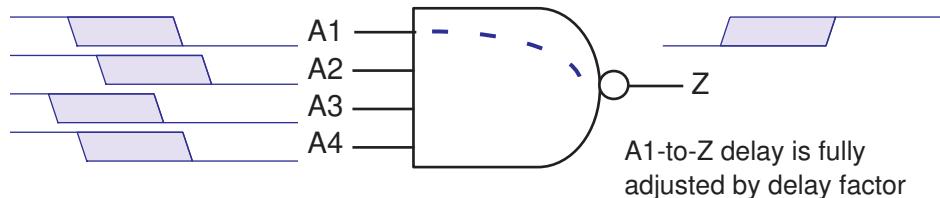
update_timing -full
report_timing -delay_type min -input_pins
report_delay_calculation -min -from Inst1/A -to Inst1/Z
```

In this example, the tool looks for situations in which a rising transition occurs at the output of each instance of the library cell NAND4gx, and multiple inputs of the cell can switch at the same time. In these situations, the tool multiplies the library-specified delay value by a factor between 0.683 and 1.0 (depending on the number of potential switching inputs), resulting in a shorter minimum delay from input to output.

Arrival Window Overlap Checking

By default, when you enable multi-input switching analysis, the tool considers arrival windows for the signal transitions at the multiple inputs. When considering a transition on one input, it looks for overlapping arrival windows of the other inputs, as shown in the following figure.

Figure 111 Arrival Windows for Multi-Input Switching Analysis



When considering the delay from input A1 to output Z, if the other input arrival windows overlap the input A1 arrival window, the full multi-input delay factor applies to the library delay value. Conversely, if there is no overlap, a factor of 1.0 applies. If some input arrival windows overlap and others do not, an intermediate factor applies (computed using an inverse proportional model).

You can disable the arrival window analysis and always fully apply the multi-input delay factor when multiple inputs can switch in the same clock cycle. This results in a more pessimistic analysis but saves runtime compared to the default multi-input switching analysis. To invoke this runtime-saving option, set the following additional variable:

```
# Enable multi-input switching analysis
set_app_var si_enable_multi_input_switching_analysis true

# Disable input arrival window timing analysis
set_app_var si_enable_multi_input_switching_timing_window_filter false
```

Including or Excluding Specific Cells in MIS Analysis

The `set_multi_input_switching_analysis` command allows you to include or exclude specific cells or library cells from MIS analysis:

```
set_multi_input_switching_analysis # Set multi-input switching analysis
                                    information for cells
      [-exclude_only]      (Exclude cells for MIS)
      [-include_only]      (Include cells for MIS)
      [-reset]             (Reset previously applied set command)
      [-analysis_mode one-of-string]
                            (Apply to specified analysis mode:
                             Values: lib_cell, lib_arc, advanced)
      object_list          (List of cell or lib_cell instances with MIS)
```

where:

- `object_list` can contain leaf cells or library cells
- The `-include_only` option includes only the specified objects in MIS analysis

- The `-exclude_only` option excludes the specified objects from MIS analysis (but allows it for everything else)
- The `-analysis_mode` option restricts the specification to only the specified analysis modes

For details, see the man page.

Scaling the Tool-Computed Advanced MIS Derates

The `set_advanced_multi_input_switching_factor` command allows you to scale the tool-computed advanced MIS delay-scaling coefficients:

```
set_advanced_multi_input_switching_factor
  -scale_factor scale_factor
    (Factor used for scaling advanced MIS coefficient:
     Range: 0 to 1)
  [-all]      (Apply factor to entire design)
  -reset      (Reset previously applied set command)
  [-inverse]  (Apply inverse factor)
  [object_list] (List of libcells to scale advanced MIS factor)
```

By default, the scaling factor *relaxes* the MIS adjustment by being multiplied against the speedup value. A value of 1 keeps the speedup as-is, a value of 0 relaxes (removes) the speedup entirely, and values in between relax the speedup partially:

```
# do not modify the MIS-adjusted cell delay at all
set_advanced_multi_input_switching_factor -scale_factor 1

# keep 70% of the tool-computed speedup
set_advanced_multi_input_switching_factor -scale_factor 0.7

# discard the tool-computed speedup entirely
# (revert to the original non-MIS cell delay)
set_advanced_multi_input_switching_factor -scale_factor 0
```

When the `-inverse` option is specified, the scaling factor *strengthens* the speedup by being multiplied directly against the MIS-adjusted cell delay:

```
# do not modify the MIS-adjusted cell delay at all
set_advanced_multi_input_switching_factor -scale_factor 1 -inverse

# multiply the MIS-adjusted cell delay by 0.7
set_advanced_multi_input_switching_factor -scale_factor 0.7 -inverse

# multiply the MIS-adjusted cell delay by 0
# (the final cell delay becomes zero)
set_advanced_multi_input_switching_factor -scale_factor 0 -inverse
```

The command applies to the specified objects (if `object_list` is used) or to the entire design (if `-all` is used).

For details, see the man page.

Unit Delay Analysis

When performing flow testing and constraint validation, the focus is typically on looking for warning and error messages rather than on the numerical timing results.

The PrimeTime tool provides a *unit delay* calculation mode that computes a 1.0 unit delay value (in library units) for every cell delay, net delay, pin transition, and coupled-net delta delay and transition. For example,

Point	DTrans	Trans	Delta	Incr	Path
<hr/>					
clock CLK1 (rise edge)			0.0	0.0	
clock network delay (propagated)			8.0	8.0	
uv21/CP (DFD1)	1.0		0.0	8.0	r
uv21/Q (DFD1)	1.0		1.0	&	9.0 r
uv12/A2 (ND2D2)	0.0	1.0	0.0	1.0	& 10.0 r
uv12/ZN (ND2D2)		1.0		1.0	& 11.0 f
uv13/I (BUFFD6)	1.0	1.0	1.0	2.0	& 13.0 f
uv13/Z (BUFFD6)		1.0		1.0	14.0 f
uv14/I (INVD8)	0.0	1.0	0.0	1.0	15.0 f
uv14/ZN (INVD8)		1.0		1.0	16.0 r
uv10/D (DFD1)	0.0	1.0	0.0	1.0	17.0 r
data arrival time					17.0
<hr/>					
clock CLK1 (rise edge)			10.0	10.0	
clock network delay (propagated)			8.0	18.0	
clock reconvergence pessimism			0.0	18.0	
uv10/CP (DFD1)				18.0	r
library setup time			-1.0	17.0	
data required time				17.0	
<hr/>					
data required time			17.0		
data arrival time			-17.0		
<hr/>					
slack (MET)				0.0	

This feature improves runtime. There is no effect on memory consumption.

To enable this feature, set the following variable:

```
pt_shell> set_app_var timing_enable_unit_delay true
```

Also disable the following features—CRPR, SI, AOCVM, POCVM, and advanced latch analysis—if you are using them:

```
pt_shell> set_app_var timing_remove_clock_reconvergence_pessimism false
pt_shell> set_app_var si_enable_analysis false
pt_shell> set_app_var timing_aocvm_enable_analysis false
```

```
pt_shell> set_app_var timing_pocvm_enable_analysis false
pt_shell> set_app_var timing_enable_through_paths false
```

11

Back-Annotation

Back-annotation is the process of reading delay or parasitic resistance and capacitance values from an external file into the tool for timing analysis. Using back-annotation, you can accurately analyze the circuit timing in the tool after each phase of physical design. To learn about back-annotation, see

- [SDF Back-Annotation](#)
- [Annotating Delays, Timing Checks, and Transition Times](#)
- [Writing an SDF File](#)
- [Setting Lumped Parasitic Resistance and Capacitance](#)
- [Detailed Parasitics](#)
- [Reading Parasitic Files](#)
- [Incomplete Annotated Parasitics](#)
- [Back-Annotation Order of Precedence](#)
- [Reporting Annotated Parasitics](#)
- [Removing Annotated Data](#)
- [GPD Parasitic Explorer](#)

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 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, SKEWCONSTRAINT

It also supports the following subset of SDF v3.0 constructs:

- RETAIN
- RECREM
- REMOVAL
- CONDENSE

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_annotated_delay` and `report_annotated_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

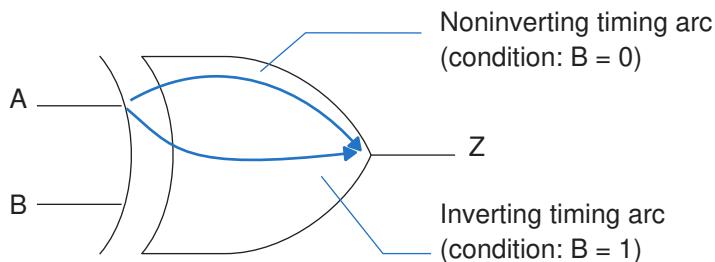
Delays and timing checks specified in an SDF file can be conditional. The SDF condition is usually an expression based on the value of some inputs of the annotated cell. The way PrimeTime annotates these conditional delays depends on whether the Synopsys library specifies conditional delays.

- 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 required conditional arc values. Consider the example 2-input XOR gate in Figure 112.

Figure 112 Example of state-dependent timing arcs



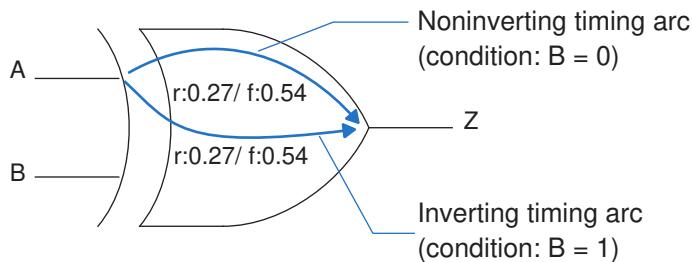
The SDF for the delay from A to Z looks like this:

```
(INSTANCE U1)
(DELAY
  (ABSOLUTE
    (COND B (IOPATH A Z (0.21) (0.54) ) )
    (COND ~B (IOPATH A Z (0.27) (0.34) ) )
```

)
)

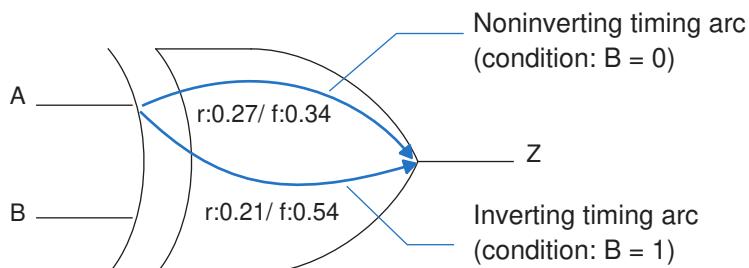
If the Synopsys library contains no conditions, the annotation from SDF uses the worst-case delay for all timing arcs from A to Z. Mapping from the library the timing arc that corresponds to a given condition is not possible. In this case, the annotated delays are as shown in [Figure 113](#).

Figure 113 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 114](#).

Figure 114 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:

```
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 19](#), they are used to annotate the current design.

Table 19 SDF constructs for timing checks

For check	SDF construct is
Setup and hold	SETUP, HOLD, and SETUPHOLD
Recovery	RECOVERY
Removal	REMOVAL
Minimum pulse width	WIDTH
Minimum period	PERIOD
Maximum skew	SKEW
No change	NOCHANGE

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, run 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 forgo 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`.

Annotating Delays, Timing Checks, and Transition Times

You can annotate delays, timing checks, and transition times 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

To set cell or net delays, run the `set_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.

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
```

To list annotated delay values, run the `report_annotated_delay` command. To remove the annotated cell or net delay values from a design, run the `remove_annotated_delay` or `reset_design` command.

Annotating Timing Checks

The `set_annotated_check` command sets the setup, hold, recovery, removal, or no-change timing check value between two pins.

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
```

To list annotated timing check values, use the `report_annotated_check` command. To see the effect of `set_annotated_check` for a specific instance, use the `report_timing` command. To remove annotated timing check values from a design, use the `remove_annotated_check` or `reset_design` command.

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]
```

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](#)
 - [Writing a Mapped SDF File](#)
 - [Writing Compressed SDF Files](#)
 - [Writing SDF Files Without Setup or Hold Violations](#)
-

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, CELLCODE, INSTANCE
- ABSOLUTE, COND, CONDELS, 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: CONDELSE, 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.

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.

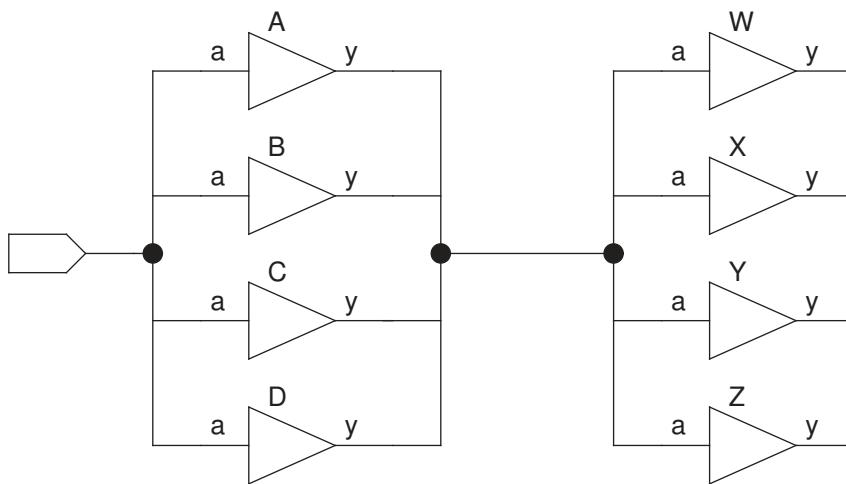
See Also

- [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 115.

Figure 115 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

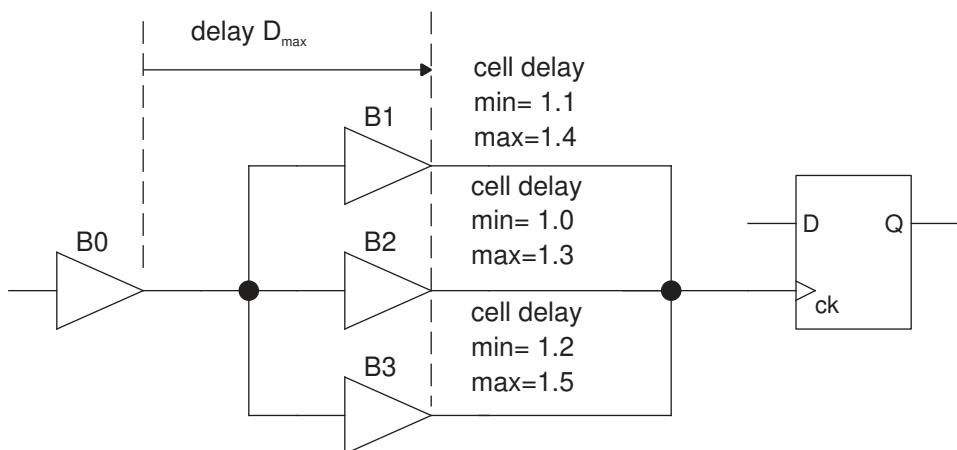
When the `sdf_align_multi_drive_cell_arcs` and `sdf_enable_port_construct` variables are enabled, PrimeTime aligns similar delay values of cell timing arcs where multiple drivers drive a common net.

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 116](#).

Figure 116 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 are 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 B0 to cells B1 through B3, 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](#)
- [Bus Naming Conventions](#)
- [Labeling Bus Arcs](#)
- [SDF Mapping Limitations](#)
- [Mapped SDF File Examples](#)

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 21](#).

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 21](#).

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        "Wed Apr 01 08:18:28 2019" )
  (VENDOR      "abc"     )
  (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 21](#).

SDF Mapping Variables

[Table 20](#) lists the variables used in the SDF mapping language.

Table 20 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 21 .
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
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 21](#) lists the functions supported when writing mapped SDF files.

Table 21 SDF mapping functions

Function	Return value	Comment
<code>in(float)</code>	Float	Natural logarithm.
<code>exp(float)</code>	Float	Exponentiation.
<code>sqrt(float)</code>	Float	Square root.
<code>max(float, float)</code>	Float	Two-argument maximum function.
<code>min(float, float)</code>	Float	Two-argument minimum function.
<code>bus(string)</code>	String	Returns a transformed string by replacing the bus delimiter characters in the string with the appropriate bus delimiter characters as specified in the <code>write_sdf</code> command. The <code>-context verilog</code> option causes the string to be transformed to the <code>%s[%d]</code> format. The <code>-context vhdl</code> option causes the string to be transformed to the <code>%s(%d)</code> format.
<code>pin(string)</code>	String	Returns the string argument passed to it.
<code>max_fall_delay(label)</code>	Float	Maximum fall delay value associated with that timing label.
<code>max_fall_delay_bus(label, from_pin, to_pin)</code>	Float	Maximum fall delay value to be used when the timing label is associated with a bus arc.
<code>max_fall_retain_delay(label, from_pin, to_pin)</code>	Float	Maximum fall delay value to be used when the timing label is associated with a retain arc.
<code>max_rise_delay(label)</code>	Float	Maximum rise delay value associated with that timing label.
<code>max_rise_delay_bus(label, from_pin, to_pin)</code>	Float	Maximum rise delay value is used when the timing label is associated with a bus arc.
<code>max_rise_retain_delay(label, from_pin, to_pin)</code>	Float	Maximum rise delay value to be used when the timing label is associated with a retain arc.
<code>min_fall_delay(label)</code>	Float	Minimum fall delay value associated with that timing label.
<code>min_fall_delay_bus(label, from_pin, to_pin)</code>	Float	Minimum fall delay function to be used when the timing label is associated with a bus arc.
<code>min_fall_retain_delay(label, from_pin, to_pin)</code>	Float	Minimum fall delay value to be used when the timing label is associated with a retain arc.

Table 21 SDF mapping functions (Continued)

Function	Return value	Comment
<code>min_period(pin)</code>	Float	Returns the minimum period value associated with the pin name “pin” of the current cell.
<code>min_rise_delay(label)</code>	Float	Minimum rise delay value associated with that timing label.
<code>min_rise_delay_bus(label, from_pin, to_pin)</code>	Float	Minimum rise delay value to be used when the timing label is associated with a bus arc.
<code>min_rise_retain_delay(label, from_pin, to_pin)</code>	Float	Minimum rise delay value to be used when the timing label is associated with a retain arc.

SDF Mapping File Syntax

[Example 12](#) shows the complete syntax of a mapped SDF file.

Example 12 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
busDelimiter      ::= $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 13 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 13](#) (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 14](#) and [Example 15](#).

Example 14 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 15 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 17](#). The `min_pulse_width` arcs are labeled by using Style 2.

[Example 16](#) shows a library file, `example.lib`.

Example 16 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 17](#) shows a mapped SDF file, `example.map`.

Example 17 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 18](#)
- Mapped SDF file – [Example 19](#)

Example 18 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 19 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)
###
```

Writing Compressed SDF Files

To compress large SDF files to a gzip file, use the `write_sdf` command with the `-compress` option:

```
pt_shell> write_sdf -compress gzip 1.sdf.gz
```

Writing SDF Files Without Setup or Hold Violations

The `write_sdf` command allows you to write an SDF file that masks setup and hold violations. This is useful in the middle of the design flow, when the timing still has some violations but gate-level simulation must be run.

To do this, use the `-mask_violations` option of the `write_sdf` command. Valid values are `setup`, `hold`, and `both`. For example,

```
pt_shell> write_sdf \
    -significant_digits 5 \
    -maskViolations both \
    masked.sdf
Adjusting SDF to mask 16 setup violations.
Adjusting SDF to mask 131 hold violations.
```

The `write_sdf` command masks violations by adjusting the delay of the last timing arc driving the violating endpoint, adding delay to mask hold violations and subtracting delay to mask setup violations.

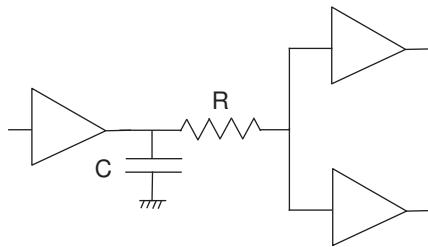
Note the following limitation:

- You might need to use a higher `-significant_digits` value to avoid rounding issues that cause slight violations.

Setting Lumped Parasitic Resistance and Capacitance

You can annotate resistance and capacitance (RC) on nets, as shown in [Figure 117](#). 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 117 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_design` 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`.

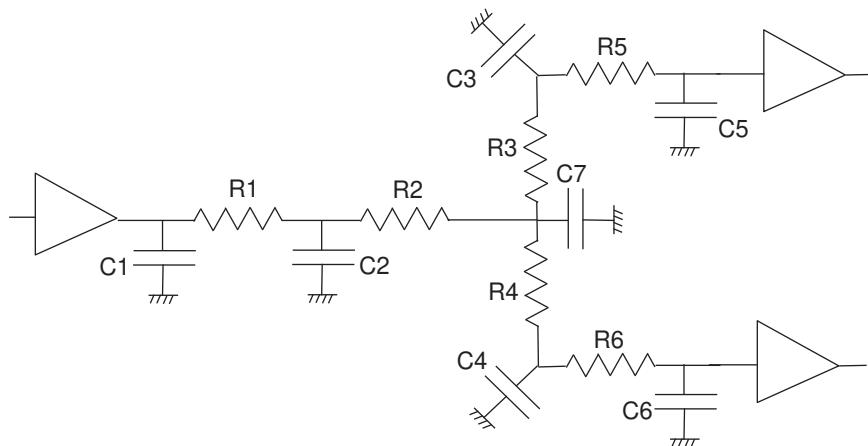
To view resistance values, use the `report_net` command. To remove resistance values annotated on specified nets, use the `remove_resistance` command. To remove resistance annotated on the entire design, use the `reset_design` command.

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 118](#)). 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.

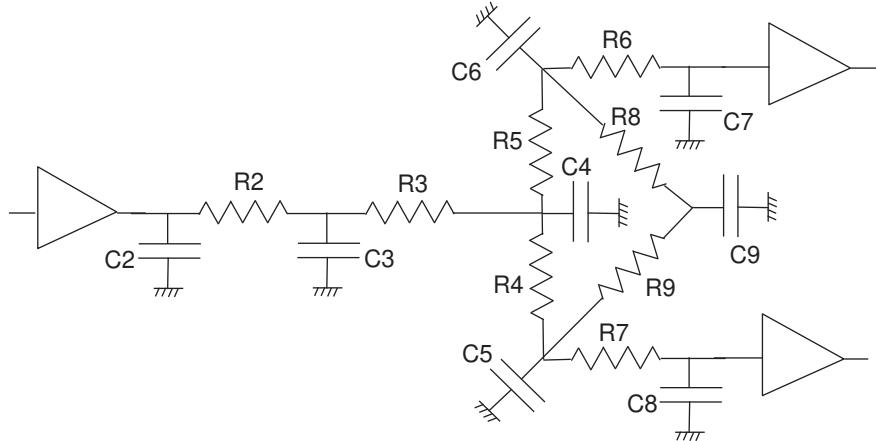
This RC network is used to compute effective capacitance ($C_{\text{effective}}$), slew, and delays at each subnode of the net. PrimeTime can read detailed RC in SPEF format.

Figure 118 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 119](#).

Figure 119 Meshed RC



Reading Parasitic Files

The `read_parasitics` command can read parasitic data file in the following formats:

- Galaxy Parasitic Database (GPD)
- Standard Parasitic Exchange Format (SPEF)
- Detailed Standard Parasitic Format (DSPF)
- Reduced Standard Parasitic Format (RSPF), version IEEE 1481-1999
- Milkyway (PARA)

The `read_parasitics` command annotates the current design with the parasitic information. A 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 parasitic file. For example, if you create the parasitic 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 a SPEF file using gzip can improve overall processing time because the file is much smaller.

For more information, see the following topics:

- [Reading Multiple Parasitic Files](#)
- [Applying Location Transformations to Parasitic Data](#)
- [Reading Parasitics With Multiple Physical Pins](#)
- [Reading a Single Corner From Multicorner Parasitic Data](#)
- [Checking the Annotated Nets](#)
- [Scaling Parasitic Values](#)
- [Reading Parasitics for Incremental Timing Analysis](#)
- [Limitations of Parasitic Files](#)

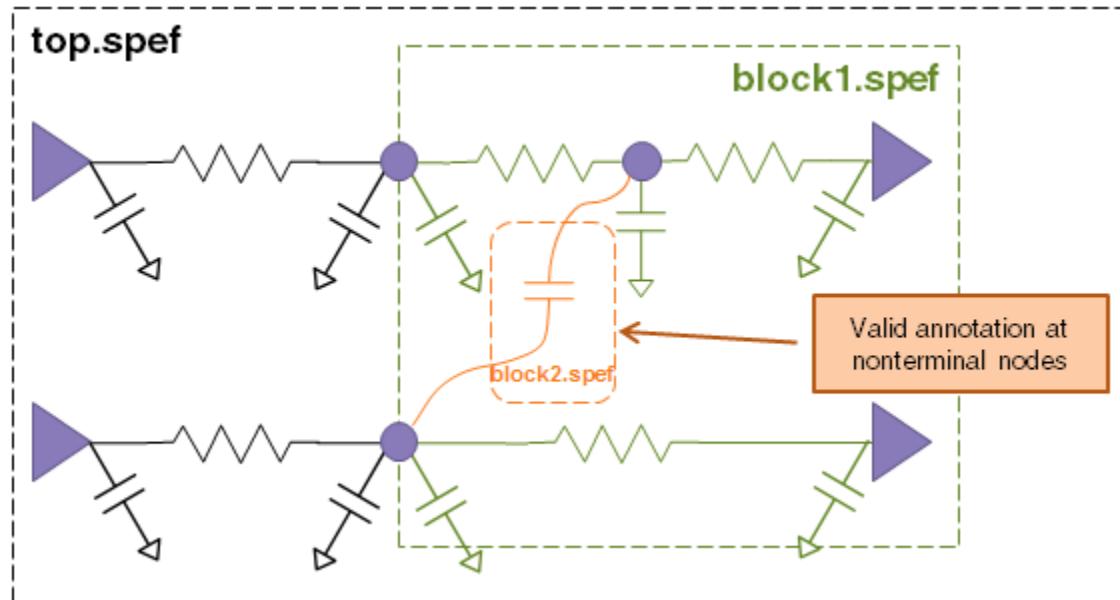
Reading Multiple Parasitic Files

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.spef -path [all_instances -hierarchy BLKA]
read_parasitics B.spef
read_parasitics C.spef
read_parasitics D.spef
read_parasitics chip_file_name
report_annotated_parasitics -check
```

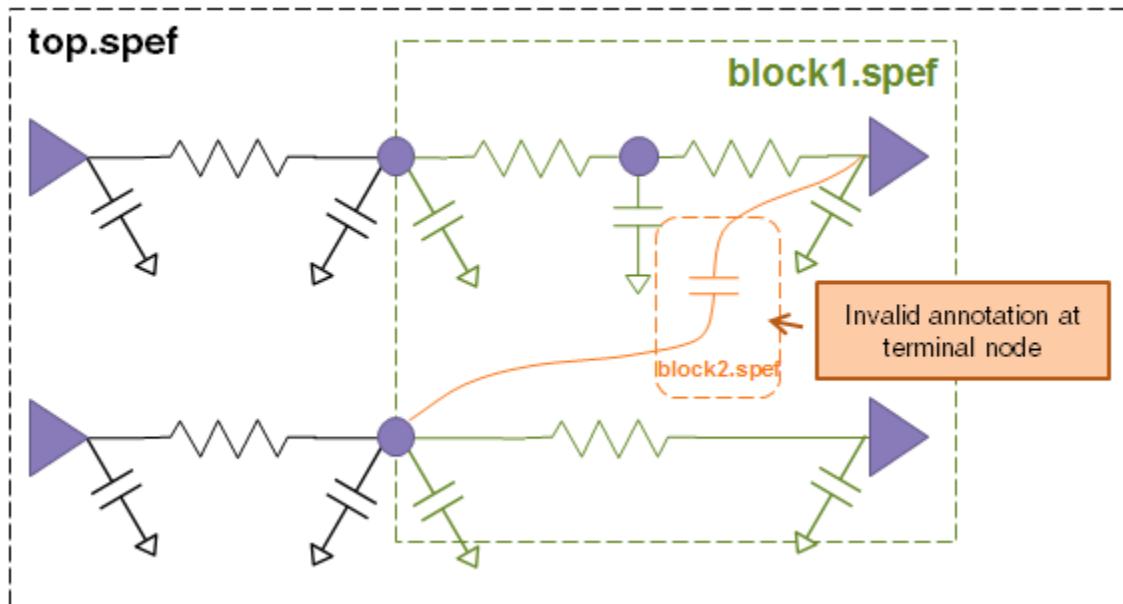
When reading multiple parasitic files, PrimeTime stitches subsequent parasitic annotations that occur only on nonterminal nodes. For example, in [Figure 120](#), the annotation specified by the block2.spf file is valid because it does not occur at a terminal node.

Figure 120 Second annotation is valid



However, in [Figure 121](#), the annotation specified by block2.spf occurs at a terminal node and is therefore invalid. PrimeTime ignores this annotation and issues a PARA-114 error message.

Figure 121 Second annotation is invalid



Applying Location Transformations to Parasitic Data

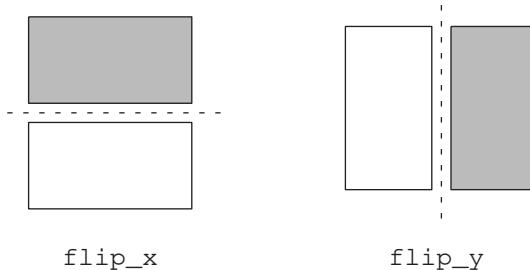
By default, the `read_parasitics` command gets information about the location and orientation of macros from the GPD or SPEF parasitic data file, if that information is available.

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 122](#) shows the transformations specified by the `-axis_flip flip_x` and `-axis_flip flip_y` options.

Figure 122 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.

Reading Parasitics With Multiple Physical Pins

In some cases, a single logical port might be associated with multiple physical shapes. For example, for a port-connected net, the wire might reach the block boundary at multiple physical locations. Such ports are described as having *multiple physical pins* in the detailed parasitics file.

In StarRC, multiple physical pin extraction is enabled by setting the following extraction option:

```
MULTI_PHYSICAL_PINS_PREFIX: YES
```

Additional options are required. For details, see the “`MULTI_PHYSICAL_PINS_PREFIX`” section in the *StarRC User Guide and Command Reference*.

To read SPEF or GPD parasitics with multiple physical pins into PrimeTime, set the following variable before reading them in:

```
pt_shell> set_app_var parasitics_enable_multiple_physical_pins true
```

In the detailed parasitics information, equivalent physical pins use a special SNPS_EEQ (“electrically equivalent”) prefix in their port (*P) definitions. For example,

```
*D_NET in 4.0
*CONN
*P in           I *C 0 0
*P SNPS_EEQ_in_1 I *C 0 0
```

In some cases, the detailed parasitics information includes a mapping table from port name to equivalent pin name at the beginning of the file:

```
// comments
// start_MPIN_table
// TOP in SNPS_EEQ_irr_0
// BLKA in SNPS_EEQ_irr10 SNPS_EEQ_irr20
// end_MPIN_table

...
*D_NET w 4.0
*CONN
*P blkal:in           I *C 0 0
*P blkal:SNPS_EEQ_in_1 I *C 0 0
*P blka1:SNPS_EEQ_irr10 I *C 0 0
*P blka1:SNPS_EEQ_irr20 I *C 0 0
```

When SPEF or GPD parasitics are read in, the tool recognizes the equivalent physical pins and maps them to the same logical netlist port. When multiple parasitics files are read in and stitched together, the equivalent physical pins are connected correctly.

When these parasitics are subsequently written out (such as by the `write_parasitics` command or when a HyperScale block model is written), the tool includes the equivalent physical pin information.

This feature is not supported in the incremental SPEF reading flow, which uses the `read_parasitics -eco` command.

Reading a Single Corner From Multicorner Parasitic Data

You can annotate your design with the parasitic data from a single corner of a multicorner GPD or 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 : ...
```

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

Use the `scale_parasitics` command to scale parasitic resistor, ground capacitor, and cross-coupling capacitor values by specified factors. You can perform a single parasitic extraction and then use the `scale_parasitics` command in the PrimeTime tool to scale the parasitic data for variation effects.

The `scale_parasitics` command works for all types of back-annotated parasitic data, including SPEF, GPD, DSPF, and RSPF. Delay calculations use the scaled parasitics

instead of the original parasitics. If you write out parasitics using the `write_parasitics` command, the scaled parasitic values are written out.

You can scale the parasitics for the whole design or just a specific list of nets. If you scale the whole design and then a specific net, the net-specific scaling applies to the original value, not the globally scaled value.

If you use the `scale_parasitics` command multiple times, each command scales the original values, not the previously scaled values. If you read in parasitics again for previously scaled nets, the global scaling factors still apply, but net-specific scaling factors are discarded.

To apply global scaling factors only to parasitic data read in by future `read_parasitics` commands (without affecting existing parasitic data), use the `scale_parasitics` command with the `-dont_apply_to_annotated` option.

To report or cancel scaling factors, use the `report_scale_parasitics` or `reset_scale_parasitics` command.

Global Parasitic Scaling

To scale all resistor, ground capacitor, and coupling capacitor values in the design, use the `scale_parasitics` command without specifying any nets. For example,

```
pt_shell> scale_parasitics \
    -resistance_factor 1.10 \
    -ground_capacitance_factor 1.10 \
    -coupling_capacitance_factor 1.25
```

Scaling parasitic values globally triggers a full timing update. To minimize the runtime cost, set the resistor, ground capacitor, and coupling capacitor scaling factors in a single `scale_parasitics` command.

Net-Specific Parasitic Scaling

You can apply scaling factors to specific nets by using the `scale_parasitics` command with a list of nets. Net-specific parasitic scaling usually triggers only an incremental timing update, not a full update.

To scale the ground capacitor and resistor values on a net n2:

```
pt_shell> scale_parasitics [get_nets n2] \
    -ground_capacitance_factor 1.3 \
    -resistance_factor 1.22
```

To inspect the changes to the scaled parasitics, use the `report_annotated_parasitics -list_annotated net_list` command.

Reporting Scaled Parasitics

To query the current global and net-specific scaling factors, use the `report_scale_parasitics` command:

```
pt_shell> scale_parasitics -coupling_capacitance_factor 1.1
...
pt_shell> scale_parasitics -coupling_capacitance_factor 1.3 [get_nets n2]
...
pt_shell> report_scale_parasitics
Global factors:
Coupling factor : 1.100000
Net             Cap factor     Res factor     Coupling factor
-----
n2              1.300000
```

Resetting Scaled Parasitics

To reset the scaled nets to their original values, use the `reset_scale_parasitics` command. You can reset all scaled nets or just specific nets:

```
pt_shell> reset_scale_parasitics [get_nets n1]
```

The `reset_scale_parasitics` command returns the values to the original values, not previously scaled values.

Scaling Parasitics in One Block

This example shows how to scale each net in a physical block in your design. Foundries can provide different scaling values for different operating conditions for a block, and you can model these variation effects using parasitic scaling.

To use this methodology, link the top level of the design, named Chip in this example. You can then switch to the physical block of interest by using a script like the following:

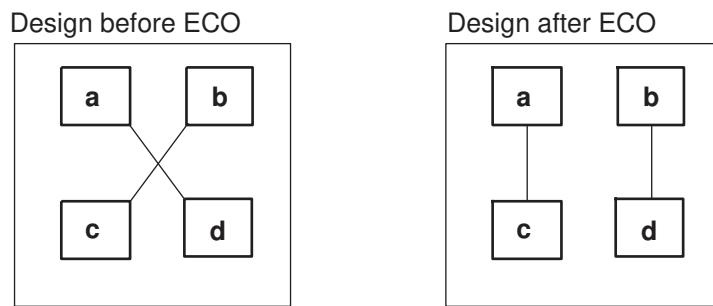
```
current_instance A
set gnets [get_nets *]      # Collection of nets for block A
scale_parasitics -ground_capacitance_factor 1.1 \
    -resistance_factor 1.1 -coupling_capacitance_factor 1.1 $gnets
current_instance Chip
# Proceed with the analysis
```

Each net in block A is scaled as specified by the `scale_parasitics` command. You can inspect the changes to the parasitics using the `report_annotated_parasitics -list_annotated` command.

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 123](#).

Figure 123 Incremental update before and after engineering change



To set the annotation on most or all nets, enter

```
pt_shell> read_parasitics full_chip.spef
pt_shell> report_timing
```

To override the annotations on a small number of nets, enter

```
pt_shell> read_parasitics eco.spef
pt_shell> report_timing
```

Limitations of Parasitic Files

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:

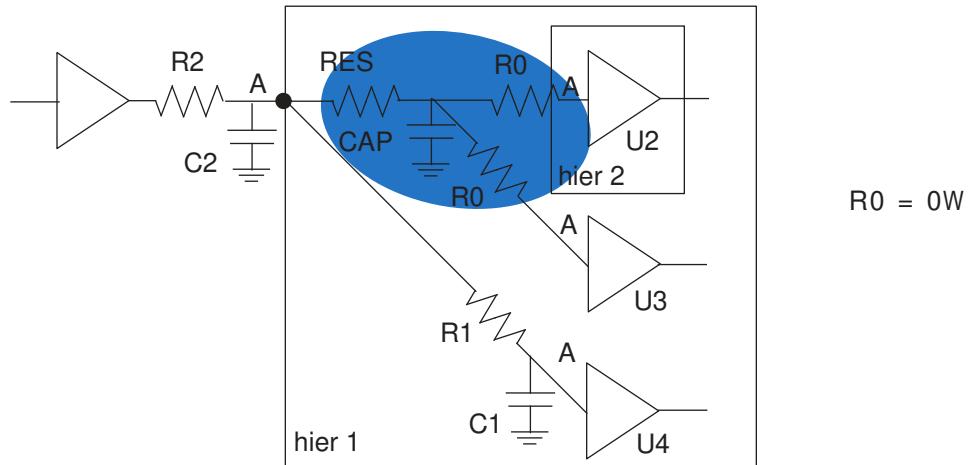
- Inductors are ignored.
- Poles and residue descriptions on reduced nets are not supported.
- Resistors cannot be connected to ground. PrimeTime ignores resistors connected to ground and displays a warning.
- Cross-coupling capacitors are split to ground unless you are using PrimeTime SI and crosstalk analysis is enabled.

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 a 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 124](#), the missing segment is completed with a fanout of two.

Figure 124 Missing segment completed using the fanout of two



The shaded area shown in Figure 124 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

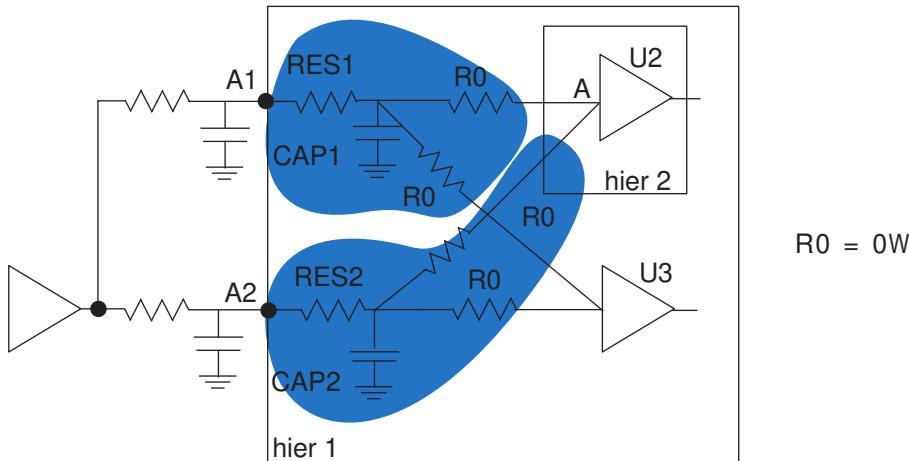
PrimeTime completes multiple pin to pin segments with a single RC pair based on the value of the option `-complete_with`, which can be `zero` or `wlm` (wire load model).

Here is a command sequence that completes the missing segments:

```
pt_shell> read_verilog design.v
pt_shell> read_parasitics incomplete_parasitics.spef
pt_shell> complete_net_parasitics -complete_with wlm
```

As shown in [Figure 125](#), 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 125 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.

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.

Reporting Annotated Parasitics

Parasitic data files can be large (100 MB or larger) and can contain many parasitics. To verify that the tool back-annotated all cell drivers, run the `report_annotated_parasitics` command.

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 Data

To remove annotations, use the commands in [Table 22](#).

Table 22 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_transition</code>
Parasitics	<code>remove_annotated_parasitics</code>
All annotations	<code>reset_design</code>

To remove all timing constraints and annotations but keep the parasitic data, use the `reset_design` command with the `-keep_parasitics` option.

GPD Parasitic Explorer

The Parasitic Explorer is an option that lets you query parasitic resistors and capacitors in GPD format that have been back-annotated on the design by the `read_parasitics` command. This feature supports the following capabilities:

- `get_resistors` – Creates a collection of parasitic resistors from one or more nets.
- `get_ground_capacitors` – Creates a collection of ground capacitors from one or more nets.
- `get_coupling_capacitors` – Creates a collection of coupling capacitors from one or more net pairs.
- Ability to query parasitic element attributes such as resistance, capacitance, subnode name, layer name, layer number, and physical location

In addition, the Parasitic Explorer provides commands to query data in GPD directories and control the reading and annotation of that data in the PrimeTime tool:

- `report_gpd_properties` – Reports the properties of the parasitic data such as completeness, the presence or absence of specific types of data, and the number of nets, cells, and ports
- `set_gpd_config` – Specifies the parasitic corners to be read and the thresholds for filtering coupling capacitors during reading
- `report_gpd_config` – Reports the option settings for reading the GPD data
- `reset_gpd_config` – Resets the settings made by the `set_gpd_config` command

- `get_gpd_corners` – Reports the parasitic corner names defined in the GPD directory
- `get_gpd_layers` – Reports the layer names defined in the GPD directory

Enabling the Parasitic Explorer Feature

To enable the Parasitic Explorer feature, set the `parasitic_explorer_enable_analysis` variable to `true`:

```
pt_shell> set_app_var parasitic_explorer_enable_analysis true
true
pt_shell> read_parasitics \
    -format gpd my_design_dir.gpd -keep_capacitive_coupling ...
```

By default, the variable is set to `false` and the feature is disabled.

If you set the variable to `true` *before* you read in the GPD parasitics, the PrimeTime tool reads in the data from all parasitic corners and you can query the data from all corners.

If you set the variable to `true` *after* you read in the GPD parasitics, the PrimeTime tool still enables the feature, but you can query parasitic data only from the current parasitic corner.

Parasitic Resistor and Capacitor Collections

To create a collection of parasitic resistors or capacitors, use the `get_resistors`, `get_ground_capacitors`, or `get_coupling_capacitors` command. These commands have options to do the following:

- Specify the net or nets from which to get the collection
- Specify a path from a node or to another node from which to get the collection
- Specify the parasitic corner or corners of the data from which to get the collection
- Filter the collection using a logical condition such as an attribute value test

For example,

```
get_resistors -of_objects [get_nets {net_rx*}] -parasitic_corners vhi85c
get_ground_capacitors -from_node U235/z -to_node n121:4
get_coupling_capacitors -of_objects n2 -filter "capacitance_max > 0.5e-3"
```

You must use either the `-of_objects` option or the `-from_node` and `-to_node` options to specify the part of the design from which to get the parasitic resistors or capacitors.

The argument of the `-of_objects` option is one or more nets. The arguments of the `-from_node` and `-to_node` options are pins, ports, or internal nodes of nets specified in the form of `net_name:index_number`.

Each command reports the contents of the collection in the same manner as the `query_objects` command. By default, a maximum of 100 objects are displayed. You can change the maximum by setting the `collection_result_display_limit` variable.

To query an attribute of an object, use the `get_attribute` command. For example, to get the resistance value for a resistor:

```
pt_shell> get_attribute -class resistor [get_resistors ...] resistance
```

To list the available attributes of an object type, use the `list_attributes` command. For example,

```
pt_shell> list_attributes -application -class resistor
```

Querying GPD Data Stored on Disk

You can query GPD data stored on disk, including parasitic data that has not yet been read in and annotated on the design. These are the supported commands:

```
report_gpd_properties
set_gpd_config
report_gpd_config
reset_gpd_config
get_gpd_corners
get_gpd_layers
```

Reporting GPD Properties

The `report_gpd_properties` command reports general information about the data in a specified GPD directory:

```
pt_shell> report_gpd_properties -gpd MyDesignA.gpd
...
GPD Summary:
Properties           Value
-----
design_name          MyDesignA
vendor_name          Synopsys Inc.
program_name         StarRC
program_version      0-2017.06
program_timestamp    July 1 2018 21:02:19
gpd_timestamp        Tue Apr 10 18:26:45 2018
gpd_version          2.6
number_of_nets       288930
number_of_cells      234730
...

pt_shell> report_gpd_properties -layers -gpd MyDesignA.gpd
...
Layer information:
Name                Properties           Value
```

```
-----
SUBSTRATE      id          0
SUBSTRATE      is_via     No
poly           id          1
poly           is_via     No
M1             id          2
M1             is_via     No
...
pt_shell> report_gpd_properties -parasitic_corners -gpd MyDesignA.gpd
...
Corner information:
-----  

Name          Properties      Value
-----  

CMINW125     process_name  /mydata/mypara/grd.min
CMINW125     temperature   125
CMINW125     global_temperature 25
CMINB40      process_name  /mydata/mypara/grd.min
CMINB40      temperature   -40
CMINB40      global_temperature 25
...

```

Setting GPD Annotation Properties

The `set_gpd_config` command lets you override certain parameters for reading parasitic data in GPD format using the `read_parasitics -format gpd` command. The default parameters are defined in a file called the GPD configuration file. This file is generated by the `StarXtract -set_gpd_config` command in the StarRC tool.

For example, the following command sets both absolute and relative thresholds for filtering coupling capacitors:

```
pt_shell> set_gpd_config -gpd my_design1.gpd \
    -absolute_coupling_threshold 3.0e-3 \
    -relative_coupling_threshold 0.03
```

To report the GPD configuration that has been set, use the `report_gpd_config` command:

```
pt_shell> report_gpd_config -gpd my_design.gpd
...
-----  

Property        Value
-----  

absolute_coupling_threshold 0.003000
relative_coupling_threshold 0.030000
coupling_threshold_operation and
netlist_select_nets *
netlist_type     {RCC *}
selected_parasitic_corners TYP25 CWORST110 CBEST0
...
```

To include reporting of options that were set in the StarRC tool during parasitic extraction, use the `-include_starrc_options` option:

Property	Value	StarRC
absolute_coupling_threshold	0.003000	N
relative_coupling_threshold	0.030000	N
coupling_threshold_operation	and	N
netlist_select_nets	*	N
netlist_type	{RCC *}	N
selected_parasitic_corners	TYP25 CWORST110 CBEST0	N
netlist_compress	true	Y
dp_string	true	Y
netlist_connect_section	false	Y
pin_delimiter	/	Y
netlist_name_map	true	Y
netlist_incremental	false	Y

To reset options previously set by the `set_gpd_config` command, use the `reset_gpd_config` command:

```
pt_shell> reset_gpd_config -gpd my_design.gpd
```

Getting GPD Corners and Layers

To report the parasitic corners or layers in a GPD directory, use the `get_gpd_corners` or `get_gpd_layers` command:

```
pt_shell> get_gpd_corners -gpd my_design1.gpd
CWORST110 TYP25 CBEST0
```

```
pt_shell> get_gpd_layers -gpd my_design1.gpd
M1 M2 M3 M4 VIA1 VIA2 VIA3
```

Parasitic Explorer Without a Netlist

The Parasitic Explorer supports reading parasitics in GPD format without a Verilog netlist or logic library. You can query the parasitic resistors and capacitors in the network. The tool derives the network connectivity from the GPD database itself instead of using the Verilog netlist.

This is the no-netlist Parasitic Explorer flow:

```
# Enable Parasitic Explorer (without reading netlist)
set_app_var parasitic_explorer_enable_analysis true

# Read parasitics in GPD format
read_parasitics -format gpd my_gpd
```

```
# Set current design to top-level module
current_design TOP
```

The `read_parasitics` command prepares for reading the parasitics but does not actually do the reading. When you run the `current_design TOP` command, the tool links the design and reads in the GPD parasitic data from the specified directory. Then you can query the parasitics using `get_resistors`, `get_ground_capacitors`, `report_gpd_config`, and so on.

12

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:

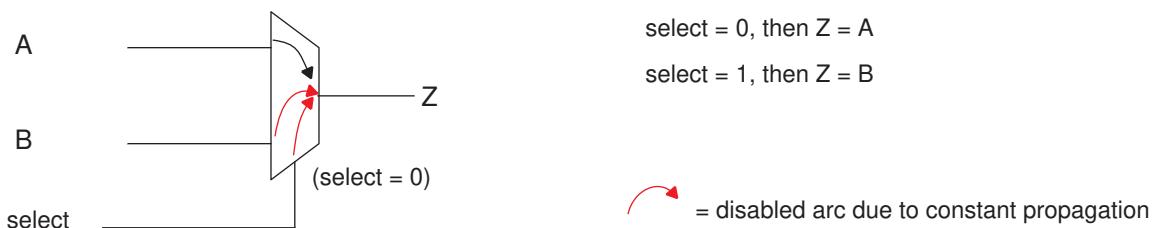
- Case Analysis
 - Mode Analysis
 - Mode Merging for Scenario Reduction
-

Case Analysis

Case analysis performs 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 and disables paths on which the constant is propagated. Case analysis does not propagate backward through the design.

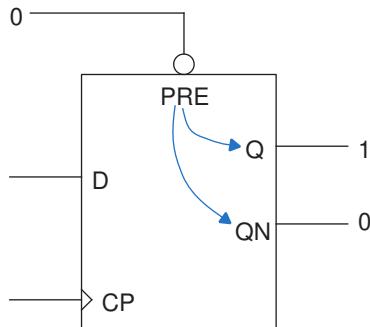
Setting a case analysis with a logic transition (rising or falling) eliminates certain paths by limiting the transitions considered during analysis. For example, the following figure shows a multiplexer. Setting the select signal to logic 0 blocks the data from B to Z and therefore disables the timing arc from B to Z.

Figure 126 Constant propagation disables timing arcs



If 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 the following figure.

Figure 127 Constant propagation through an asynchronous input



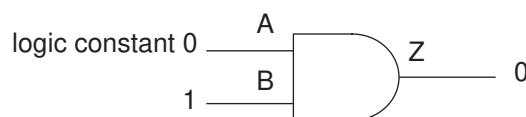
To learn more about case analysis, see

- Propagation of Constants that Control Logic Values
- Configuring Case Analysis and Constant Propagation
- Setting and Removing Case Analysis Values
- Enabling Case Analysis Propagation Through Sequential Cells
- Evaluating Conditional Arcs Using Case Analysis
- Disabling Scan Chains With Case Analysis
- Performing Case Analysis
- Reporting Disabled Arcs and Tracing Constant Propagation

Propagation of Constants that Control Logic Values

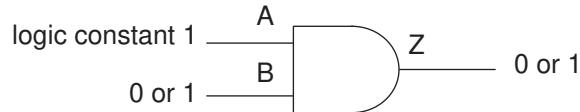
If there are logic constants and case analysis set on the inputs of a cell, and the logic constants are set to a controlling logic value of that cell, then the logic constant is propagated from the output; otherwise case analysis is propagated.

Example 1: AND Gate With Constant Logic 0 on Input A



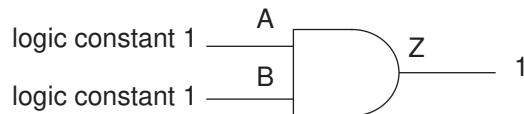
The logic 0 is propagated from the output pin Z, irrespective of the value on B.

Example 2: An AND Gate With Constant Logic 1 on Input A



The logic constant 1 on pin A is not a controlling logic value. Therefore, the logic constant does not propagate to output pin Z.

Example 3: An AND Gate With Constant Logic 1 on Inputs A and B



The logic constant 1 is propagated from output pin Z.

Configuring Case Analysis and Constant Propagation

To configure case analysis and how the tool propagates constants, set the following variables.

Variable	Description
case_analysis_sequential_propagation	Controls the propagation of logic constants through sequential cells.
case_analysis_propagate_through_icg	Controls the propagation of logic constants through integrated clock-gating cells.
disable_case_analysis	Disables the propagation of both user-specified case analysis logic values and logic constants from the netlist; this variable setting overrides the setting of the disable_case_analysis_ti_hi_lo variable.
disable_case_analysis_ti_hi_lo	Disables the propagation of logic constants from the netlist.

Setting and Removing Case Analysis Values

To set case analysis values on specified pins or ports, run the `set_case_analysis` command. When setting case analysis, you can specify either of the following:

- **Constant logic 0, 1, or static**

This command sets case analysis for the test port to constant logic 0:

```
pt_shell> set_case_analysis 0 [get_ports test]
```

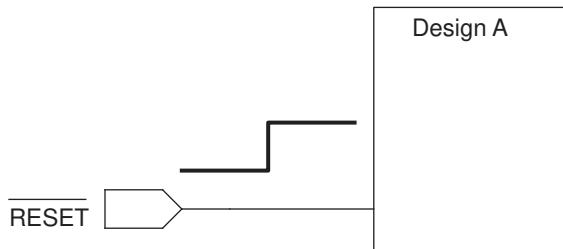
The logic value `static` means constant 0 or 1, without any transitions, so that the connected net cannot act as a crosstalk aggressor or experience a change in delay as a crosstalk victim.

- **Rising or falling transition**

This command sets case analysis for the RESET port to a rising transition:

```
pt_shell> set_case_analysis rising [get_ports RESET]
```

Figure 128 Setting Case Analysis With a Rising Transition



Note:

PrimePower calculation ignores the rising and falling transition values.

In case of conflict, the following rules apply:

- A `set_case_analysis` setting has priority over a built-in constant value (for example, in the Verilog netlist).
- A newer `set_case_analysis` setting has priority over an older setting on the same port or pin.
- A `set_case_analysis` value set directly on a port or pin has priority over a conflicting case analysis value propagated to that port or pin.
- Where propagated case analysis values are in conflict, logic 0 has the highest priority, then the `static` case setting, then logic 1.

You can use case analysis in combination with mode analysis. For example, use case analysis to specify that certain internal logic is a constant value because the RAM block is in read mode:

```
pt_shell> set_case_analysis 1 U1/U2/select_read
```

Removing Case Analysis Values

To remove case analysis values, run the `remove_case_analysis` command:

```
pt_shell> remove_case_analysis test
```

To suppress the propagation of all 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 to write scripts that Design Compiler can synthesize correctly.

Enabling Case Analysis Propagation Through Sequential Cells

By default, PrimeTime does not propagate case analysis values through sequential cells. You can override the default behavior in the following ways:

- To propagate case analysis values through all sequential cells, set the `case_analysis_sequential_propagation` variable to `always`.
- To propagate case analysis values through specific sequential cells:
 1. Ensure that the `case_analysis_sequential_propagation` variable is set to `never` (the default).
 2. Specify the cells through which to propagate case analysis values by running the `set_case_sequential_propagation` command:

```
pt_shell> set_case_sequential_propagation cell_or_libcell_list
```
 3. Verify the sequential cells enabled for case analysis propagation with one of these methods:
 - Run the `report_case_analysis` command with the `-sequential_propagation` option.
 - Query the `is_case_sequential_propagation` attribute, which returns `true` for sequential cells and library cells that are enabled for case analysis propagation.

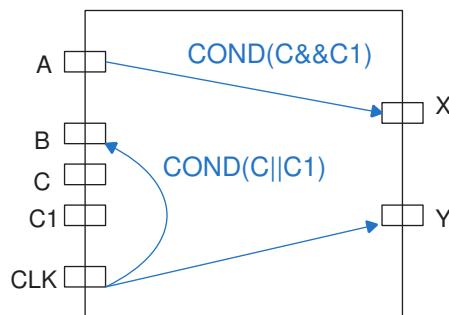
To disable case analysis propagation through specific sequential cells, run the `remove_case_sequential_propagation` command.

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.

In the following example, a delay arc exists from A to X with condition $\text{COND}(C \& C1)$. PrimeTime disables the arc when the Boolean expression ($C \& C1$) evaluates to false. A setup arc also exists from CLK to B with condition $\text{COND}(C \mid\mid C1)$. PrimeTime disables the setup arc when the Boolean expression ($C \mid\mid C1$) evaluates to false. The arc from CLK to Y is a nonconditional arc.

Figure 129 Conditional and nonconditional arcs in a Liberty Model



These commands disable the delay arc from A to X because case analysis evaluates the condition to be false:

```
pt_shell> set_case_analysis 0 C
pt_shell> set_case_analysis 1 C1
```

To enable the delay arc from A to X and the setup arc from CLK to B:

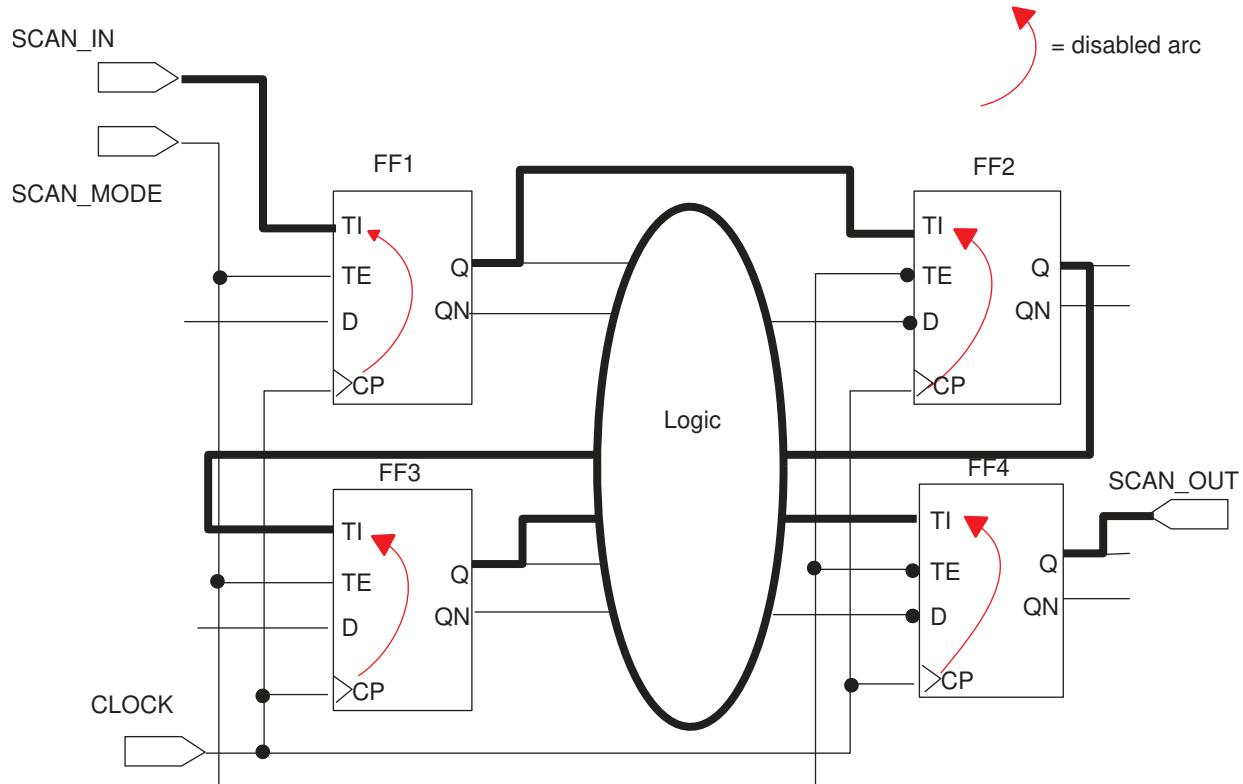
```
pt_shell> set_case_analysis 1 {C C1}
```

For more information about specifying conditional arcs in the Liberty modeling language, see the *Library Compiler User Guide*.

Disabling Scan Chains With Case Analysis

By using case analysis, you can disable a scan chain and therefore omit the scan chain from timing analysis.

Figure 130 Scan chain that starts at SCAN_IN and ends at SCAN_OUT



To disable the preceding scan chain, set the **SCAN_MODE** port to logic 0:

```
pt_shell> set_case_analysis 0 [get_ports "SCAN_MODE"]
```

PrimeTime propagates the constant value to the **TE** pin of each scan flip-flop. The sequential case analysis on the **FF1**, **FF2**, **FF3**, and **FF4** cells disables the setup and hold arcs from **CP** to **TI**.

Performing Case Analysis

You can perform case analysis with these methods:

- Basic Case Analysis
- Detailed Case Analysis
- Standalone PrimeTime Constraint Consistency Checking

Basic Case Analysis

To generate a basic case analysis report, run the `report_case_analysis` command:

```
pt_shell> report_case_analysis
*****
Report : case_analysis
...
*****
Pin name          Case analysis value
-----
test_port          0
U1/U2/core/WR     1
```

Detailed Case Analysis

To perform a detailed case analysis, run the `report_case_analysis` command with the `-from` or `-to` option and a list of pins or ports. This invokes constraint consistency checking, which requires a PrimeTime SI license.

```
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
-----
2            1       F()=A B & C &        y/z
3            1       y/A
4            1       y/B
4            1       y/C

Path number: 2
Path Ref #  Value  Properties           Pin/Port
-----
1            1       user case          y/A
1            1       user case          P

Path number: 3
Path Ref #  Value  Properties           Pin/Port
-----
1            1       user case          y/B
1            1       user case          T
```

Path Ref #	Value	Properties	Pin/Port
1			y/c
1	user case		L

Alternatively, you can view case analysis in the PrimeTime GUI by choosing Constraints > View Case Debugging Results. The results are shown in a separate window in which you can view the case propagation details.

Standalone PrimeTime Constraint Consistency Checking

To access extensive case analysis features, run the PrimeTime constraint consistency shell. For details, see [Starting a Constraint Consistency Session](#).

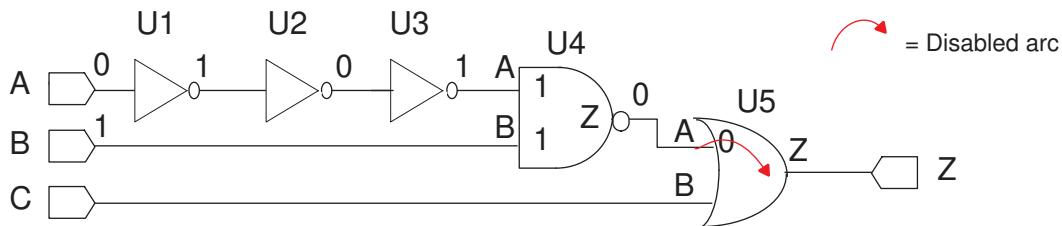
Reporting Disabled Arcs and Tracing Constant Propagation

When PrimeTime performs constant propagation, it disables timing arcs at the boundary of the domain where a constant propagates. To trace the cause of a disabled timing arc due to constant propagation:

1. Enable the generation of log file that lists nets and pins that propagate constants; to do this, set the `case_analysis_log_file` variable to a file name.
2. View the timing arcs disabled after constant propagation by running the `report_disable_timing` command.

In the following example, the tool disables the arc from U5/A to U5/Z because U5/A is at constant value 0, and no constant is propagated to U5/Z.

Figure 131 Arc From U5/A to U5/Z is Disabled



These commands set the case analysis, generate a constant propagation log file, and report the disabled timing arcs:

```
pt_shell> set_case_analysis 0 {get_ports "A"}
pt_shell> set_case_analysis 1 {get_ports "B"}
pt_shell> set_app_var case_analysis_log_file my_design_cnst.log
```

```
pt_shell> report_disable_timing
Cell or Port      From      To      Flag  Reason
-----  
U5                A         Z       C     A = 0
```

The constant propagation log file my_design_cnst.log shows this information:

```
*****
Report : case_analysis propagation
...
*****  
  
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'  
...  
  
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
```

Mode Analysis

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 when 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 types of modes:

- **Cell 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.

- **Design modes**

You can map a design mode 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 mode are enabled. When a design mode is inactive (due to selection of a different design mode), all cell modes mapped to that design mode are made inactive and all paths mapped to that design mode are set to false paths.

Mode Groups

Modes are organized into groups. Within each group, there are two possible states: all modes 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. 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.

To learn about working with modes, see

- [Setting Cell Modes](#)
 - [Reporting Modes](#)
-

Setting Cell Modes

To set a cell operating mode, use one of these methods (shown in order of highest to lowest precedence):

- [Setting Modes Directly on Cell Instances](#)
 - [Setting Cell Modes Indirectly Using Design Modes](#)
 - [Placing Cells Into Modes by Using Case Analysis](#)
-

Setting Modes Directly on Cell Instances

By default, all cell modes are enabled in each cell instance. You can enable only one cell mode and disable all other modes. When a mode is disabled, all timing arcs in that cell mapped to that mode are disabled (not analyzed for timing).

To enable a cell mode on cell instances, use the `set_mode -type cell` command:

```
pt_shell> set_mode -type cell READ U1/U2/core
```

This enables the read mode and disables all other modes in the mode group for the U1/U2/core cell.

To cancel the mode selection and make all modes active for the U1/U2/core cell:

```
pt_shell> reset_mode U1/U2/core
```

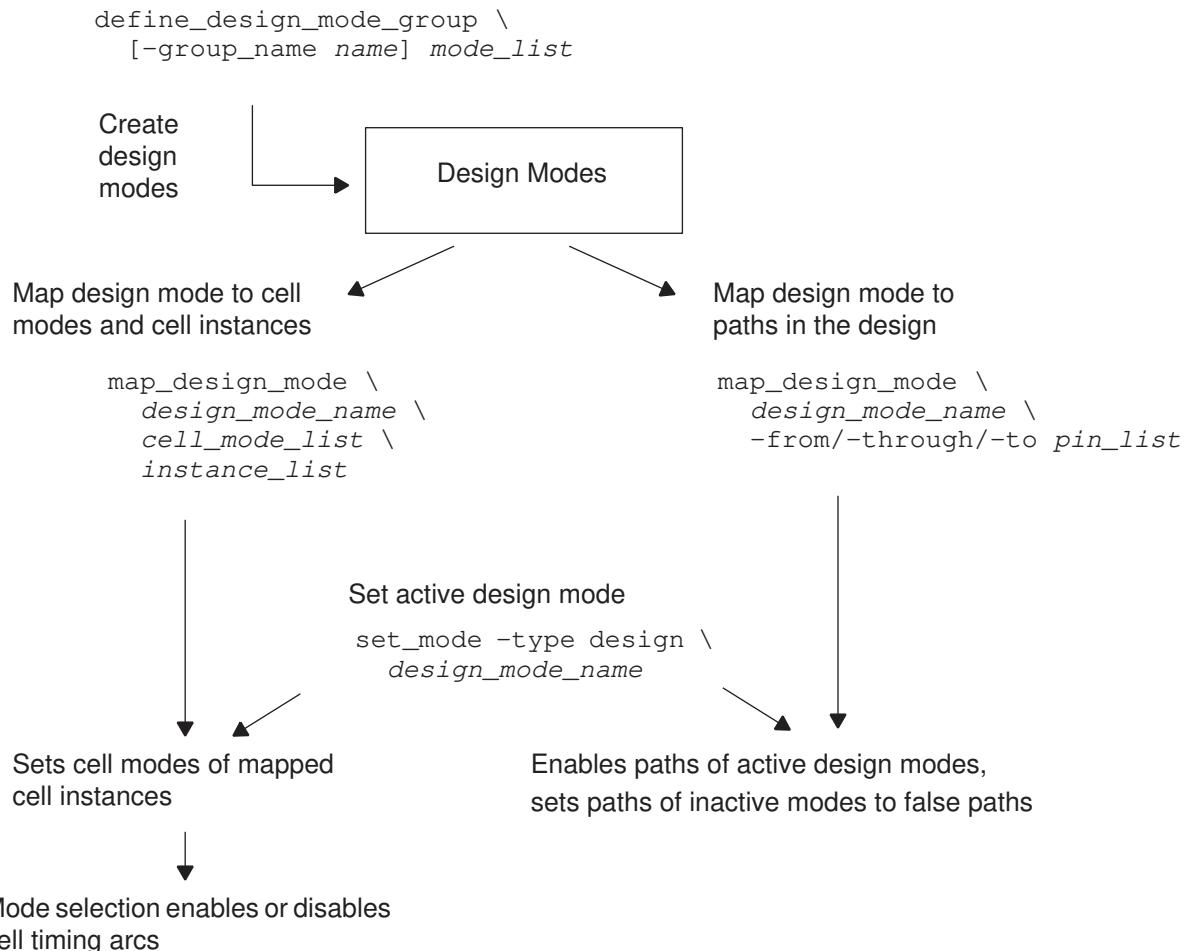
Setting Cell Modes Indirectly Using Design Modes

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 set cell modes indirectly using design modes:

1. Create a group of modes for the current design with the `define_design_mode_group` command. To create more than one design mode group, use multiple `define_design_mode_group` commands.
2. Map each design mode to the associated cell modes or paths with the `map_design_mode` command.
3. Set the current design 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.

Figure 132 Defining, mapping, and setting design modes



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 with these methods:

- [Mapping a Design Mode to Cell Modes and Instances](#)
- [Mapping a Design Mode 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.

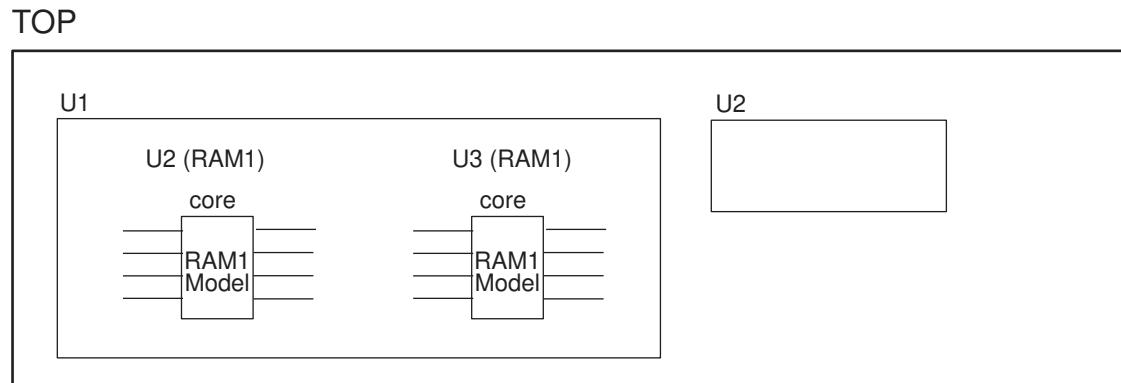
To map a design mode to a cell mode and cell instances, run the `map_design_mode` command with this syntax:

```
map_design_mode design_mode_name \
    cell_mode_list instance_list
```

For example, this command maps the execute design mode to invoke the `read_ram` cell mode in the U1/U2/core cell instance:

```
pt_shell> map_design_mode execute read_ram U1/U2/core
```

Figure 133 Mapping a design mode to a cell mode and cell instance



After this mapping, setting the execute design mode 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. Therefore, these false paths are removed from the timing analysis.

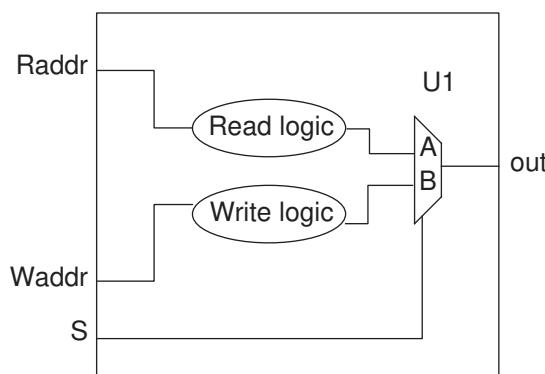
To map a design mode to a set of paths, run the `map_design_mode` command with this syntax:

```
map_design_mode design_mode_name \
    [-from pin_list] [-through pin_list] [-to pin_list]
```

For example, the following commands map the read mode to the paths through the read logic and the write mode to the paths through the write logic:

```
pt_shell> map_design_mode read -from Raddr -through U1/A
pt_shell> map_design_mode write -from Waddr -through U1/B
```

Figure 134 Mapping a design mode to a set of paths



Unmapping a Design Mode

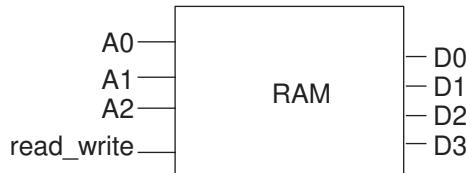
To cancel or change the mapping of a design mode, run the `remove_design_mode` command. Removing the mapping of a design mode reverses the effects of any cell mode settings or false paths applied as a result of that mapping.

Placing Cells Into Modes by 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.

Figure 135 RAM Liberty cell can operate in read and write modes



In the preceding example, 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";
        }
    }
    ...
}
  
```

This command enables the read mode in PrimeTime:

```
pt_shell> set_case_analysis 0 [get_ports read_write]
```

This command enables the write mode in PrimeTime:

```
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.

Reporting Modes

To report cell and design modes that have been defined or set, run the `report_mode` command:

```
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

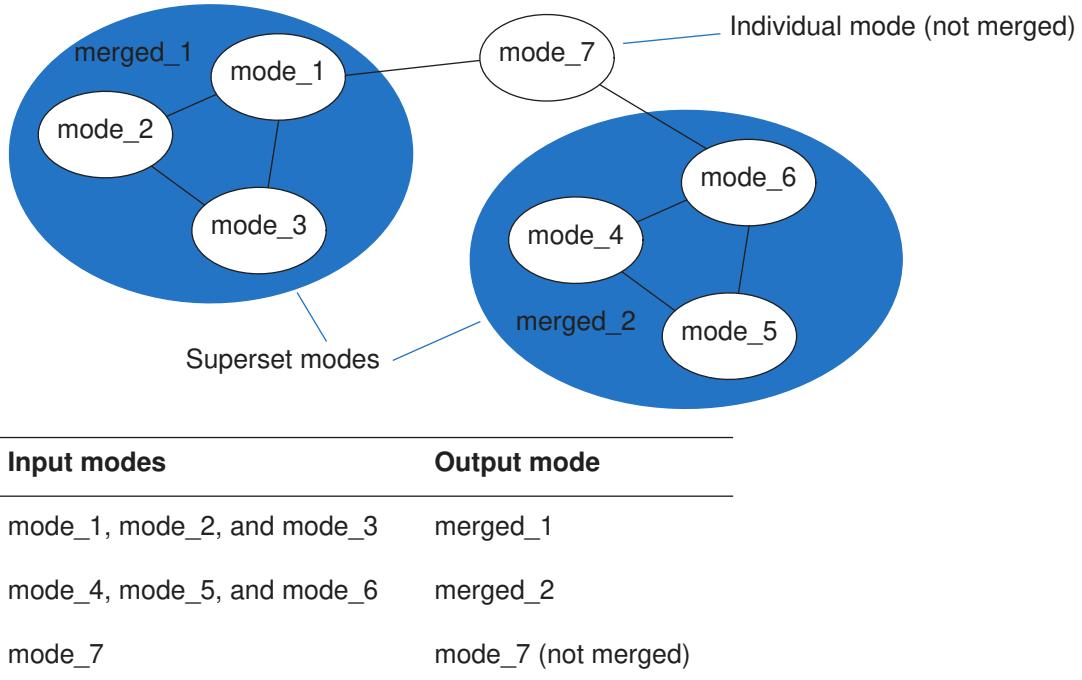
Mode Merging for Scenario Reduction

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 PrimeTime constraint consistency mode merging capabilities.

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

Figure 136 Mode merging automatically determines which input modes to merge and creates output modes



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. Run mode merging:

```
create_merged_modes
```

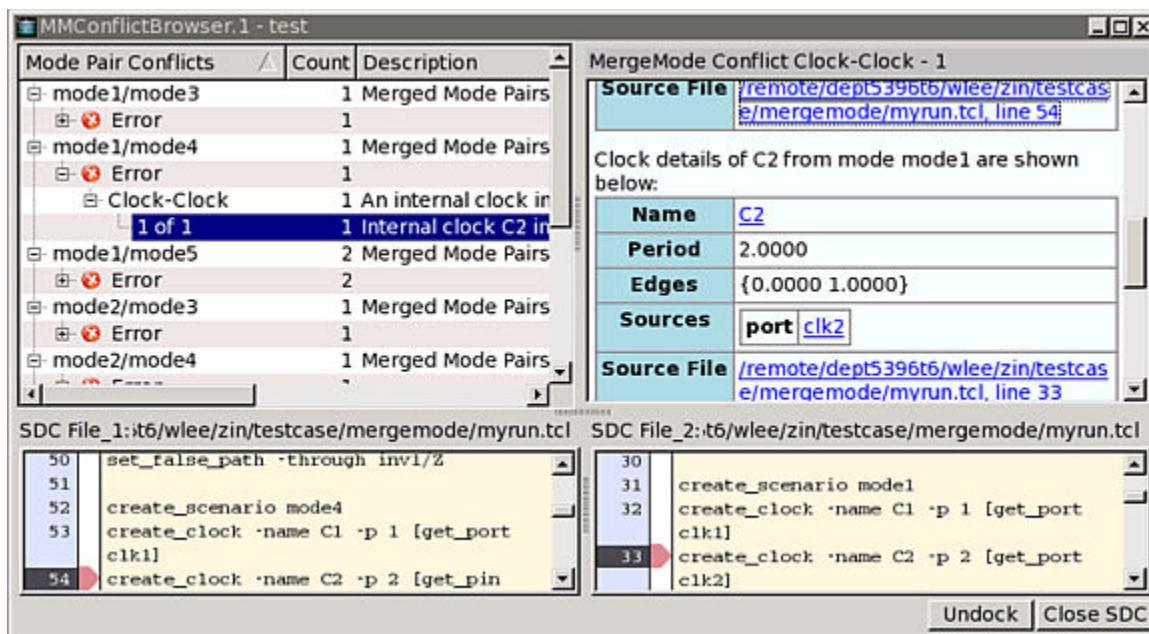
5. Check the merged constraints and mode merging analysis report in the merged_constraints directory.

Mode merging considers corner conditions. The merged constraints are valid only for corners if the corresponding input modes are valid for those corners. The valid corners are specified by the `snps_corner_name` DMSA host variable.

Debugging Mode Merging Conflicts

To visually debug mode merging conflicts in a GUI, run the `create_merged_modes` command with the `-interactive` option. This performs 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.

Figure 137 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 [SolvNetPlus article 000024378](#),
[“PrimeTime Mode Merging Application Note.”](#)

13

Variation

On-chip variation analysis applies different operating conditions to minimum paths and maximum paths. To improve the accuracy, you can apply the following variation methods:

- [Advanced On-Chip Variation](#)
 - [Parametric On-Chip Variation \(POCV\)](#)
-

Advanced On-Chip Variation

Advanced on-chip variation (AOCV) analysis reduces pessimism by applying less derating to paths with less variation, based on the number of gates in the path and the physical distance spanned by the path. AOCV is less pessimistic than a traditional OCV analysis, which relies on constant derating factors that do not take path-specific metrics into account.

A longer path that has more gates tends to have less total variation because the random variations from gate to gate cancel each other out. A path that spans a larger physical distance across the chip tends to have larger systematic variation. AOCV analysis determines path-depth and location-based bounding box metrics to calculate a path-specific AOCV derating factor.

AOCV analysis works with all other PrimeTime features and affects all reporting commands. It is compatible with distributed multi-scenario analysis and multicore analysis.

To learn about using AOCV, see

- [The AOCV Flow](#)
 - [Configuring Advanced On-Chip Variation Analysis](#)
 - [Importing AOCV Information](#)
 - [Querying AOCV Derating on Timing Paths](#)
-

The 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_ocvm` command.

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 `update_timing` or `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. Signoff 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. You can exploit reduced derating pessimism across the entire design to save area and improve performance.

By construction, graph-based analysis 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, 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 signoff.

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`:

```
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`.

Configuring Advanced On-Chip Variation Analysis

To configure advanced on-chip variation (AOCV) analysis, set the variables in the following table.

Variable	Description
<code>timing_aocvm_analysis_mode</code>	Specifies the calculation of depth metrics
<code>timing_aocvm_enable_clock_network_only</code>	Applies AOCV analysis to the clock network only
<code>timing_aocvm_enable_single_path_metrics</code>	Specifies whether AOCV analysis uses separate depth and distance values for nets and cells
<code>timing_ocvm_enable_distance_analysis</code>	Specifies whether advanced or parametric on-chip variation analysis performs distance-based analysis
<code>timing_ocvm_precedence_compatibility</code>	Controls the fallback to on-chip variation (OCV) derating for advanced or parametric OCV

Importing AOCV Information

To perform AOCV analysis, you must specify AOCV information in derating tables. To optionally model other effects not related to process variation, use guard-band timing derating.

To learn how to specify AOCV information, see

- [Specifying AOCV Derating Tables](#)
- [File Format for AOCV](#)
- [AOCV Table Groups](#)
- [Specifying Depth Coefficients](#)
- [Guard-Banding in AOCV](#)
- [Incremental Timing Derating](#)
- [Using OCV Derating in AOCV Analysis](#)

Specifying AOCV Derating Tables

Use the `read_ocvm` 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-format AOCV files. This command is used to protect sensitive process-related information. The `read_ocvm` 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 (//).

The following table shows the syntax definition for the AOCV file format.

Table 23 AOCV file format syntax

Specifier	Description
version	The AOCV version number.

Table 23 AOCV file format syntax (Continued)

Specifier	Description
group_name	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.
object_type	design lib_cell cell
rf_type	rise fall rise fall
delay_type	cell net cell net
derate_type	early late
path_type	clock data clock data
object_spec	A pattern describing objects affected by the table.
voltage	The supply voltage, in volts. The table applies only to cells operating at that voltage. This is an optional parameter.
depth	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.
distance	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	<ul style="list-style-type: none"> • 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_ocvm` 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 derating, so the `-increment` and `-aocvm_guardband` options of the `set_timing_derate` command are mutually exclusive.

Incremental timing derating follows to the same precedence and override rules as regular timing derating. For more information about the precedence rules, see [File Format for AOCV](#).

By default, a specified incremental derating factor replaces any previous incremental derating factor set for an object, such as a cell or net.

To accumulate rather than overwrite successive incremental derating settings, set the `timing_enable_cumulative_incremental_drate` variable to `true`.

If no timing derating factor exists, the value of 0.0 is used for incremental derating, and 1.0 is used for regular derating. 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.

Here are some examples of incremental timing derating calculation.

- 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 Derating in AOCV Analysis

PrimeTime supports a unified framework so that AOCV can use OCV derating under certain conditions. By default, both OCV and AOCV derating 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 Derating on Timing Paths

To get detailed information about AOCV derating applied to a timing path, query the following AOCV attributes:

- `aocvm_coefficient`
- `applied_derate`
- `depth`
- `derate_factor_depth_distance`
- `distance`

- guardband
- incremental

Although AOCV derating is 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 (POCV)

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
- Reduced pessimism gap between graph-based analysis and path-based analysis
- Less overhead for incremental timing analysis

Using this feature requires a PrimeTime-ADV license. To learn about performing POCV analysis, see the following topics:

- [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](#)
- [Enabling Analysis With Moment-Based Modeling](#)
- [Reporting Parametric On-Chip Variation Results](#)
- [Statistical Graph Merging Pessimism](#)
- [Querying POCV Slack and Arrival Attributes](#)

Variables and Commands for Parametric On-Chip Variation

The following tables summarize the variables and commands for POCV.

Table 24 Variables for POCV analysis

Variable	Description
parasitics_enable_tail_annotation	Enables reading of tail annotation data from parasitic data files for via variation analysis
timing_enable_constraint_variation	Enables constraint variation for setup and hold constraints
timing_enable_slew_variation	Enables slew variation for cell delays
timing_enable_via_variation	Enables physical via variation analysis
timing_enable_via_variation	Enables via variation analysis using data read with the <code>read_ivm</code> command
timing_pocvm_corner_sigma	Specifies the corner sigma for POCV delay constraint analysis
timing_pocvm_enable_analysis	Enables POCV analysis

Table 24 Variables for POCV analysis (Continued)

Variable	Description
timing_pocvm_enable_extended_moments	Enables usage of asymmetric moment-based modeling data
timing_pocvm_max_transition_sigma	Specifies the corner sigma for POCV maximum transition time analysis
timing_pocvm_precedence	Specifies the order of priority between file-based and library-based POCV coefficients
timing_pocvm_report_sigma	Specifies the sigma for reporting
timing_use_slew_variation_in_constraint_arcs	Enables use of slew variation in constraint arc variation computation (requires that timing_enable_constraint_variation be set to true)

Table 25 Commands for POCV Analysis

Command	Description
read_ocvm pocvm_file	Reads POCV tables
read_ivm via_file	Reads via variation tables
report_delay_calculation -derate	Reports details about derating for delay calculation
report_ocvm -type pocvm	Displays POCV information, including POCV coefficient and distance-based derating table data
report_ivm	Displays via variation table data
report_timing -derate	Reports POCV information in timing report
report_timing_derate -pocvm_coefficient_scale_factor	Reports POCV scaling
report_timing_derate -pocvm_guardband	Reports POCV guard banding
reset_timing_derate -pocvm_coefficient_scale_factor	Removes POCV scaling
reset_timing_derate -pocvm_guardband	Removes POCV guard banding
set_timing_derate -pocvm_coefficient_scale_factor	Specifies POCV coefficient scaling

Table 25 Commands for POCV Analysis (Continued)

Command	Description
<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 26 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:

$$\text{POCV coefficient} = \frac{s \text{ (delay variation)}}{m \text{ (nominal delay)}}$$

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). The following example shows a POCV slew-load table in the library.

Example 20 POCV LVF Slew-Load Table

```
ocv_sigma_cell_rise ("delay_template_7x7") {
    sigma_type : "early";
    index_1("0.008800, 0.026400, 0.060800, 0.129600, 0.267200, 0.542400, 1.093600");
    index_2("0.001000, 0.002400, 0.005200, 0.010800, 0.022100, 0.044500, 0.089500");
    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", \
    "
```

Chapter 13: Variation
Parametric On-Chip Variation (POCV)

```

        "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("0.008800, 0.026400, 0.060800, 0.129600, 0.267200, 0.542400, 1.093600");
    index_2("0.001000, 0.002400, 0.005200, 0.010800, 0.022100, 0.044500, 0.089500");
    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");
}

```

Using LVF improves the overall accuracy of POCV analysis by applying coefficients that vary with different slew and load conditions, instead of using a fixed coefficient for all conditions. The tool supports POCV LVF data for delay timing arcs, setup and hold constraints, recovery and removal constraints, and clock-gating checks.

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 the following example. While linking the design, PrimeTime automatically reads the distance-based derating table if it exists in the library.

Example 21 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;

```

Extraction of LVF Coefficients in ETMs

When you generate an extracted timing model (ETM) using the `extract_model` command with POCV enabled, by default the tool generates Liberty Variation Format (LVF) tables to model the POCV effects. With LVF tables in the ETM, the tool that consumes the ETM can perform timing analysis accurately at various corners.

The `extract_model` command creates delay sigma tables for both combinational and sequential delay arcs. It can also create constraint sigma tables for constraints arcs (setup, hold, minimum pulse width, clock-gating, removal, and recovery) when constraint variation is enabled.

For best accuracy at the top level, generating LVF tables is recommended. However, LVF might not be fully supported by all downstream tools. You can specify the types of LVF data generated for the ETM by using the `extract_model_create_variation_tables` variable.

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 SPEF or GPD parasitic data 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_ocvm` command (similar to loading AOCV tables in the AOCV flow):

```
pt_shell> read_ocvm pocv_coefficient_file_name
pt_shell> read_ocvm 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_path` variables. 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 additionally include slew variation effects in the constraint arc variation computation, set the `timing_use_slew_variation_in_constraint_arcs` variable to `setup_hold` (or `setup` or `hold` only). The default is `none`.
- To enable slew variation for cell delays, set the `timing_enable_slew_variation` variable to `true`.

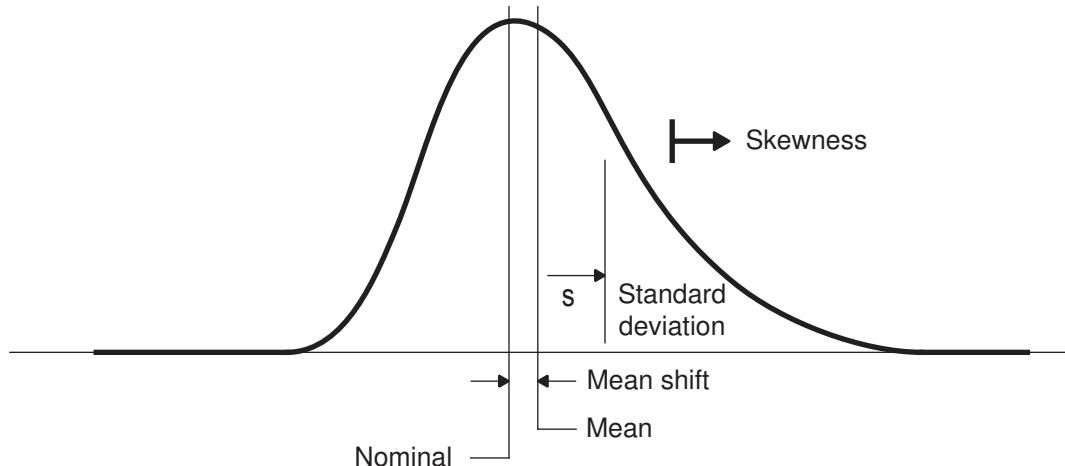
The tool reads the constraint and slew variation data provided in the Liberty Variation Format (LVF) in the `.lib` or `.db` file.

To report the constraint and slew variation, use the `report_timing -variation` or `report_ocvm` command.

Enabling Analysis With Moment-Based Modeling

The tool supports analysis using cell libraries containing moment-based Liberty Variation Format (LVF) delay and constraint data. This type of data more accurately models the non-Gaussian statistical variation characteristics of advanced process nodes and very low supply voltages. The following diagram shows a typical asymmetric cell delay distribution.

Figure 138 Moment-Based Asymmetric Delay Distribution



The statistical parameters for this distribution are the mean shift, standard deviation, and skewness. In LVF modeling syntax, these parameters are specified by the following attributes:

```
ocv_std_dev_*
ocv_mean_shift_*
ocv_skewness_*
```

For example, an `ocv_std_dev_cell_rise` table in the LVF model specifies the standard deviation of the cell delay distribution as a function of input transition and output load.

To enable timing analysis using moment-based modeling data, set the control variable:

```
pt_shell> set_app_var timing_pocvm_enable_extended_moments true
```

In variation reports, the mean value reported for a parameter includes both the nominal and the mean shift adjustment.

Using this feature requires a PrimeTime-ADV-PLUS license.

Reporting Parametric On-Chip Variation Results

To see the results of parametric on-chip variation (POCV) analysis,

1. [Report the POCV Coefficient and Derating](#)
2. [Report the POCV Analysis Results](#)

Report the POCV Coefficient and Derating

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 the following example.

```
pt_shell> report_ocvm -type pocvm [get_cells 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 the following example.

Example 22 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. Here is an example:

Point	Incr			Path			Value
	Mean	Sensit	Corner	Value	Mean	Sensit	
clock clk (rise edge)				0.00			0.00
clock network delay (ideal)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
input external delay				0.00			0.00 r
in1 (in)	0.00	0.00	0.00	0.00	0.00	0.00	0.00 r
b1/Z (B1I)	1.00	3.00	10.00	10.00	1.00	3.00 H	10.00 r
b3/Z (B1I)	1.00	2.00	7.00	2.82	2.00	3.61 H	12.82 r
out1 (out)	0.00	0.00	0.00	0.00	2.00	3.61	12.82 r
data arrival time							12.82
clock clk (rise edge)				2.00			2.00
clock network delay (ideal)	0.00	0.00	0.00	0.00	0.00	0.00	2.00
clock reconvergence pessimism	0.00	0.00	0.00	0.00	0.00	0.00	2.00
clock uncertainty				-0.20			1.80
output external delay				-10.00			-8.20
data required time					-8.20	0.00	-8.20
data required time					-8.20	0.00	-8.20
data arrival time					-2.00	3.61	-12.82
statistical adjustment				0.00			-21.02
slack (VIOLATED)					-10.20	3.61	-21.02

In the `Incr` columns,

- The `Mean`, `Sensit`, and `Corner` numbers provide mean, standard deviation, and corner information about each incremental delay distribution by itself, independent of other incremental delay distributions. `Corner` is equal to $\text{Mean} \pm K^* \text{Sensit}$, where K is equal to `timing_pocvm_report_sigma`.
- The `Incr Value` number is not derived from the incremental delay distribution. Instead, it is the simple numerical difference between the previous and current `Path Value` (corner arrival) numbers, described below.

In the `Path` columns,

- The `Mean`, `Sensit`, and `Value` numbers provide provide mean, standard deviation, and corner information about the cumulative arrival distribution at that pin. `Value` is equal to $\text{Mean} \pm K^* \text{Sensit}$, where K is equal to `timing_pocvm_report_sigma`.
- A pin's cumulative arrival distribution is the statistical combination (in the distribution domain) of all incremental distributions up to that pin.

For details on how PrimeTime calculates the final derating factor from all derating factors applied in POCV analysis, run the `report_delay_calculation -derate` command. It reports in details how the final derating values are derived for cell delay and cell delay sigma as shown in the following example.

Example 23 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 the preceding example, the `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. The following example shows the output of `report_delay_calculation -derate` when the POCV slew-load table is applied.

Example 24 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" )

```

Statistical Graph Merging Pessimism

In POCV analysis, delays are statistical distributions rather than fixed numbers. Therefore, when comparing multiple delay values, such as converging timing arcs through a multiple-input gate, the maximum delay can be larger than any of the individual delays when considered by themselves. This effect is called graph merging pessimism.

POCV analysis accounts for graph merging pessimism by removing the pessimism as a timing point adjustment in the `report_timing -variation` timing report, as shown in the following example.

Point Value	Incr				Path	
	Mean	Sensit	Corner	Value	Mean	Sensit
<hr/>						
clock clk (rise edge) 0.00	0.00				0.00	0.00
clock network delay (ideal) 0.00	0.00	0.00	0.00	0.00	0.00	0.00
input external delay 0.00 r	0.00				0.00	0.00
in1 (in) 0.00 r	0.00	0.00	0.00	0.00	0.00	0.00
b1/Z (B1I) 10.00 r	1.00	3.00	10.00	10.00	1.00	3.00 H
statistical graph pessimism 10.00	3.00	-2.24			0.00	4.00 2.00
b3/Z (B1I) 13.49 r	1.00	2.00	7.00	3.49	5.00	2.83 H
statistical graph pessimism 13.49	5.00	-2.58			0.00	10.00 1.16
out1 (out) 13.49 r	0.00	0.00	0.00	0.00	10.00	1.16
data arrival time 13.49						

```

clock clk (rise edge)      2.00          2.00      2.00
2.00
clock network delay (ideal) 0.00  0.00  0.00  0.00  0.00  0.00
2.00
clock reconvergence pessimism 0.00  0.00  0.00  0.00  0.00  0.00
2.00
output external delay           0.00
2.00
data required time            2.00      0.00
2.00
-----
--- data required time          2.00      0.00
2.00
data arrival time             -10.00    1.16
-13.49
-----
--- statistical adjustment     0.00
-11.49
slack (VIOLATED)              -8.00    1.16
-11.49

```

A `report_timing` command that uses both `-from` and `-to` can potentially have less graph merging pessimism than a timing report using `-to` the same endpoint alone (without using `-from`). Therefore, you can expect that the results of `report_timing -to xyz` to bound, but not necessarily match, the results of `report_timing -from abc -to xyz`.

Querying POCV Slack and Arrival Attributes

POCV analysis uses the following slack and arrival attributes on pins, timing paths, and timing points to cover the mean and variation of the cell delays.

Table 27 POCV slack and arrival attributes

Attribute	Object class
<code>max_fall_variation_arrival</code>	pin
<code>max_fall_variation_slack</code>	pin
<code>max_rise_variation_arrival</code>	pin
<code>max_rise_variation_slack</code>	pin
<code>min_fall_variation_arrival</code>	pin
<code>min_fall_variation_slack</code>	pin
<code>min_rise_variation_arrival</code>	pin
<code>min_rise_variation_slack</code>	pin

Table 27 POCV slack and arrival attributes (Continued)

Attribute	Object class
statistical_adjustment	timing_path
variation_arrival	timing_path, timing_point
variation_slack	timing_path, timing_point

POCV slack and arrival attributes are statistical quantities; 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. For example:

```
pt_shell> get_attribute [get_pins I/ZN] max_fall_variation_slack
{}
pt_shell> get_attribute [get_attribute [get_pins I/ZN] \
max_fall_variation_slack] mean
0.990180
pt_shell> get_attribute [get_attribute [get_pins 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.mean
0.990180
pt_shell> get_attribute $path variation_slack.std_dev
0.000488
```

14

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 Requirements](#)
 - [Specifying Power Supply Connectivity](#)
 - [UPF Commands](#)
 - [Golden UPF Flow](#)
 - [Multiple Supply Voltage Analysis](#)
-

Multivoltage Analysis Requirements

The Prime Suite tools support *multivoltage* analysis (which is analysis with different power supply voltages on different cells):

- PrimeTime calculates voltage-correct delays and slews.
- PrimeTime SI calculates voltage-correct crosstalk delay and noise effects.
- PrimePower calculates power consumption based on the actual supply voltage on each supply pin of each cell.

To perform a multivoltage analysis, you need to specify the *power intent* of the design (which logic operates at what voltages). This can be done using either method shown in [Table 28](#).

Table 28 Providing Multivoltage Power Intent for a Design

When power intent is	Verilog netlist requirements	Library requirements
Read from UPF file (or equivalent Tcl commands)	PG-pin netlist or non-PG-pin netlist	PG-pin library
Inferred from PG-pin netlist	PG-pin netlist	PG-pin library

The library requirements for multivoltage analysis are:

- The libraries 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 User Guide*.

If any cells in a PG pin library are missing power pins or ground pins, PrimeTime removes these cells from the design at link time. No timing or power information is reported for these types of cells.

- The Liberty library data must 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 [Cross-Library Voltage and Temperature Scaling](#).

Specifying Power Supply Connectivity

The PrimeTime tool supports two ways to specify power supply connectivity:

- [UPF-Specified Supply Connectivity](#)
- [Netlist-Inferred Supply Connectivity](#)

UPF-Specified Supply Connectivity

By default, for multivoltage designs with UPF-specified power intent, you get the PG connectivity by loading the UPF files associated with the design. For example,

```
set_app_var search_path "/path1/mydata /path2/mydata"
set_app_var link_path "* mylib_1V.db"
read_verilog my_design.v
link_design
load_upf my_design.UPF
read_sdc my_constraints.sdc
set_voltage 0.9 -min 0.6 -object_list [get_supply_nets VDD1]
set_voltage 0.8 -min 0.5 -object_list [get_supply_nets VDD2]
set_voltage 0.0 -min 0.0 -object_list [get_supply_nets VSS]
...
```

See Also

- [UPF Commands](#)

Netlist-Inferred Supply Connectivity

To have the PrimeTime tool derive the supply connectivity information from a PG pin Verilog netlist, set the `link_keep_pg_connectivity` variable to `true` before you read in any design information. For example,

```
set_app_var search_path "/path1/mydata /path2/mydata"
set_app_var link_path "* mylib_1V.db"
set_app_var link_keep_pg_connectivity true
read_verilog my_design.v
link_design
read_sdc my_constraints.sdc    # no need to load UPF file
set_voltage 0.9 -min 0.6 -object_list [get_supply_nets VDD1]
set_voltage 0.8 -min 0.5 -object_list [get_supply_nets VDD2]
set_voltage 0.0 -min 0.0 -object_list [get_supply_nets VSS]
...
```

With this variable set to `true`, the tool derives the UPF power intent from the PG netlist data and implicitly updates the design database as if it were running UPF commands like `create_power_domain`, `create_supply_port`, `create_supply_net`, and `connect_supply_net`. After you link the design, you can set the supply voltages by using the `set_voltage` command.

This PG netlist flow works with design data generated in either the UPF-prime flow or the golden UPF flow. It supports PrimeTime features such as timing analysis, voltage scaling, power switches, and ECOs. Note that the PG netlist flow requires all the netlists to have PG information; it does not accept mixture of PG and non-PG netlists.

In the netlist-inferred flow, you can read in a UPF file to apply boundary information (port supply and voltage information), but internal block information is ignored. For details, see [Reading Block Boundary Information From UPF](#).

The tool creates UPF power switch objects specified in a PG Verilog file. You can query `upf_power_switch` objects using the following commands:

```
get_power_switches
report_power_switch
get_supply_nets -of_objects [get_object {upf_power_switch}]
```

Reading Block Boundary Information From UPF

In the netlist-inferred flow, you can read in a UPF file to apply boundary information, such as port supply and voltage information. This is particularly useful when using the `extract_model` command, to ensure that the resulting model has the correct port attributes.

[Table 29](#) lists the boundary UPF commands that are applied.

Table 29 UPF Commands Applied in the Netlist-Inferred Supply Connectivity Flow

UPF command	Behavior
add_pst_state	Always applied
connect_supply_net	Ignored by default, but applied if pg_allow_pg_pin_reconnection is set to true
set_port_attributes	Always applied (but used only by ETM extraction)
set_related_supply_net	Always applied

All other commands, such as those that define internal block information, are ignored when the link_keep_pg_connectivity variable is set to true. In particular, note that the following significant UPF commands are ignored:

- create_supply_net
- set_level_shifter
- set_isolation
- create_power_switch

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 PrimePower 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.lib
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_elements
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](#)
- [UPF Supply Set Handles](#)

- [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:

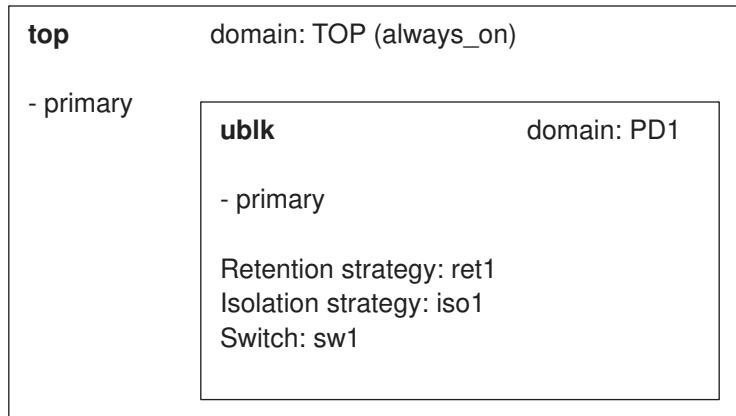
- primary
- default_isolation
- default_retention

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 139](#) has two power domains, TOP and PD1. [Example 25](#) shows the use of supply set handles for this design example.

Figure 139 Design example using supply set handles



Example 25 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 26](#). This script collects and reports the power and ground supply nets associated with the primary supply sets of each power domain.

Example 26 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 [Collection \(get_*\) Commands](#).

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`, `set_retention_elements`, 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 specified 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. PrimePower 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 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

PrimeTime uses library and 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`.

Scaling Library Groups

The recommended way to provide voltage-specific library information to PrimeTime is using *scaling library groups*.

With this feature, you provide a set of libraries characterized across a range of supply voltage and temperature values:

```
pt_shell> define_scaling_lib_group \
    {lib_0.9V.db \
     lib_1.05V.db \
     lib_1.3V.db}
```

Then during analysis, PrimeTime automatically interpolates between the specified libraries to compute cell behaviors at the exact voltage and temperature conditions of each cell instance.

For more information on scaling library groups, see [Cross-Library Voltage and Temperature Scaling](#).

Cell Alternative Library Mapping

The cell alternative library mapping method lets you specify or change the mapping of cell instances to libraries. This can be useful when [UPF-specified](#) or [netlist-inferred](#) power intent information is not available.

For example,

```
set link_path {* slow_0.7v.db}
read_verilog text.v
link_design MY_DESIGN

define_scaling_lib_group -exact_match_only { \
    slow_0.7v.db \
    slow_1.1v.db}
define_cell_alternative_lib_mapping {slow_0.7v.db} -cells {BLK1}
define_cell_alternative_lib_mapping {slow_1.1v.db} -cells {BLK2}

update_timing
report_timing
```

The `define_cell_alternative_lib_mapping` command specifies an alternative library to use for the analysis of specific cell instances. Cells can be hierarchical or leaf cells. For hierarchical cells, a set of libraries can be specified; for each leaf cell, the first library in the set that satisfies the cell reference is used.

Each library must be part of a scaling library group. The scaling library group definition can include the `-exact_match_only` option to indicate the per-cell exact match intent. However, this option can be omitted if other areas of the design require true voltage scaling analysis.

The library mapping relationships can be specified after the design is linked; they are evaluated during timing analysis. The design does not need to be relinked if the library mappings are updated (although the timing would need to be updated).

The precedence rules are as follows:

- Leaf cell specifications take precedence over hierarchical cell specifications.
- `define_cell_alternative_lib_mapping` settings take precedence over `link_path_per_instance` settings, `set_link_lib_map` settings, and instance-specific `set_voltage` specifications.

You can determine the assignment of a cell to an alternative library by querying the cell's `exact_match_library` attribute:

```
pt_shell> get_attribute [get_cells U01] exact_match_library
{"slow_1.1v"}
```

Per-Instance Link Library Paths

You can use per-instance link library paths to control how cell instances link to library files when the design is being linked. These *per-instance* link paths take precedence over the global `link_path` specification.

Per-instance link library paths can be applied in two ways:

- The `link_path_per_instance` variable
- The `set_link_lib_map` command

For more information, see [Per-Instance Link Library Paths](#).

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.

PrimePower 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 both internal and external PG pin information such as the name, type, and voltage value.

Note:

The `report_power` command reports the power of internal PG pins and corresponding external PG pins. It 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 User Guide*. For more information about power analysis, see the “Multivoltage Power Analysis” chapter in the *PrimePower 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 30](#) lists the attributes of UPF power domain, supply net, supply port, and power switch objects.

Table 30 UPF collection objects

Command and object class	Attribute	Attribute type	Comment
<code>get_power_domains</code>	<code>name</code>	String	Name as entered
<code>upf_power_domain</code>	<code>full_name</code>	String	Hierarchical name
	<code>elements</code>	Collection	
	<code>scope</code>	String	

Table 30 UPF collection objects (Continued)

Command and object class	Attribute	Attribute type	Comment
	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)
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

Table 30 UPF collection objects (Continued)

Command and object class	Attribute	Attribute type	Comment
	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 report 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 ted	Power Pin	Type	Voltage	Power Net Connec
	Name		Max	Min
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

You can use `pg_pin_info` and `lib_pg_pin_info` collection objects to query the power and ground pins associated with cells, ports, and library cells.

These are informational collection objects that represent PG pins. They can be queried for information, but they are not PG pin netlist objects themselves.

To obtain these informational objects, query the `pg_pin_info` attribute of a cell or port or the `lib_pg_pin_info` attribute of a library cell. For example,

```
pt_shell> get_attribute [get_cells FF1] pg_pin_info
{"FF1/VDD", "FF1/VSS"}
pt_shell> get_attribute [get_ports CLK] pg_pin_info
{"CLK/<port_power>", "CLK/<port_ground>"}
pt_shell> get_attribute [get_lib_cells */DFFX1] lib_pg_pin_info
{"DFFX1/VDD", "DFFX1/VSS"}
```

The following example shows the attributes available for `pg_pin_info` objects associated with a cell (ports are similar):

```
pt_shell> report_attribute -application
          [get_attribute [get_cells FF1] pg_pin_info]
...
top      FF1/VDD      string      full_name      FF1/VDD
top      FF1/VDD      string      pin_name      VDD
```

Chapter 14: Multivoltage Design Flow

UPF Commands

```

top      FF1/VDD      string    supply_connection      VDD_TOP_NET
top      FF1/VDD      string    type                  primary_power
top      FF1/VDD      float     voltage_for_max_delay 0.800000
top      FF1/VDD      float     voltage_for_min_delay 1.200000
top      FF1/VDD      string    voltage_source

top      FF1/VSS      string    full_name           FF1/VSS
top      FF1/VSS      string    pin_name            VSS
top      FF1/VSS      string    supply_connection    VSS_NET
top      FF1/VSS      string    type                primary_ground
top      FF1/VSS      float    voltage_for_max_delay 0.000000
top      FF1/VSS      float    voltage_for_min_delay 0.000000
top      FF1/VSS      string    voltage_source       LIBRARY_VOLTAGE_MAP

```

The following example shows the attributes available for `lib_pg_pin_info` objects associated with a library cell:

```

pt_shell> report_attribute -application
          [get_attribute [get_lib_cells */DFFX1] lib_pg_pin_info]
...
top      DFCND1LVT/VDD   string    pin_name           VDD
top      DFCND1LVT/VDD   string    type                primary_power
top      DFCND1LVT/VDD   float     voltage            0.864000

top      DFCND1LVT/VSS   string    pin_name           VSS
top      DFCND1LVT/VSS   string    type                primary_ground
top      DFCND1LVT/VSS   float     voltage            0.000000

```

For more information about these attributes and what they represent, see the `pg_pin_info_attributes` and `lib_pg_pin_info_attributes` man pages.

The following example uses these attributes to print the supply names and voltages for a collection of cells:

```

set my_cells [get_cells ...]
foreach_in_collection cell $my_cells {
    # print cell name
    echo [get_object_name $cell]

    # print supply pin names and voltages
    foreach_in_collection pg_pin_info [get_attribute $cell pg_pin_info] {
        echo [format {%8s %6.3f %6.3f} \
               [get_attribute $pg_pin_info pin_name] \
               [get_attribute $pg_pin_info voltage_for_max_delay] \
               [get_attribute $pg_pin_info voltage_for_min_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).

To check for supply nets without an associated supply port, use the following command:

```
pt_shell> check_timing \
    -include supply_net_without_associated_supply_port
```

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:

- $VIL = 0.3 * VDD$, $VIH = 0.7 * VDD$
- $VOL = 0.4$, $VOH = 2.4$
- $VLmin = -0.5$, $VLmax = VDD + 0.5$
- $VOmin = -0.3$, $VOmax = VDD + 0.3$

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 31](#) summarizes the effects of the threshold and strategy settings.

Table 31 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 VOH < Load VIH	None	None	None	None	
Driver VOL > Load VIL	None	None	None	None	
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.

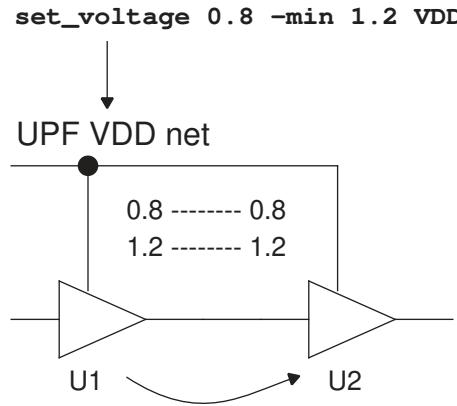
Effect of Correlated Supplies on Signal Level Checking

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 140](#).

Figure 140 Signal Level Checking With Supply Correlation



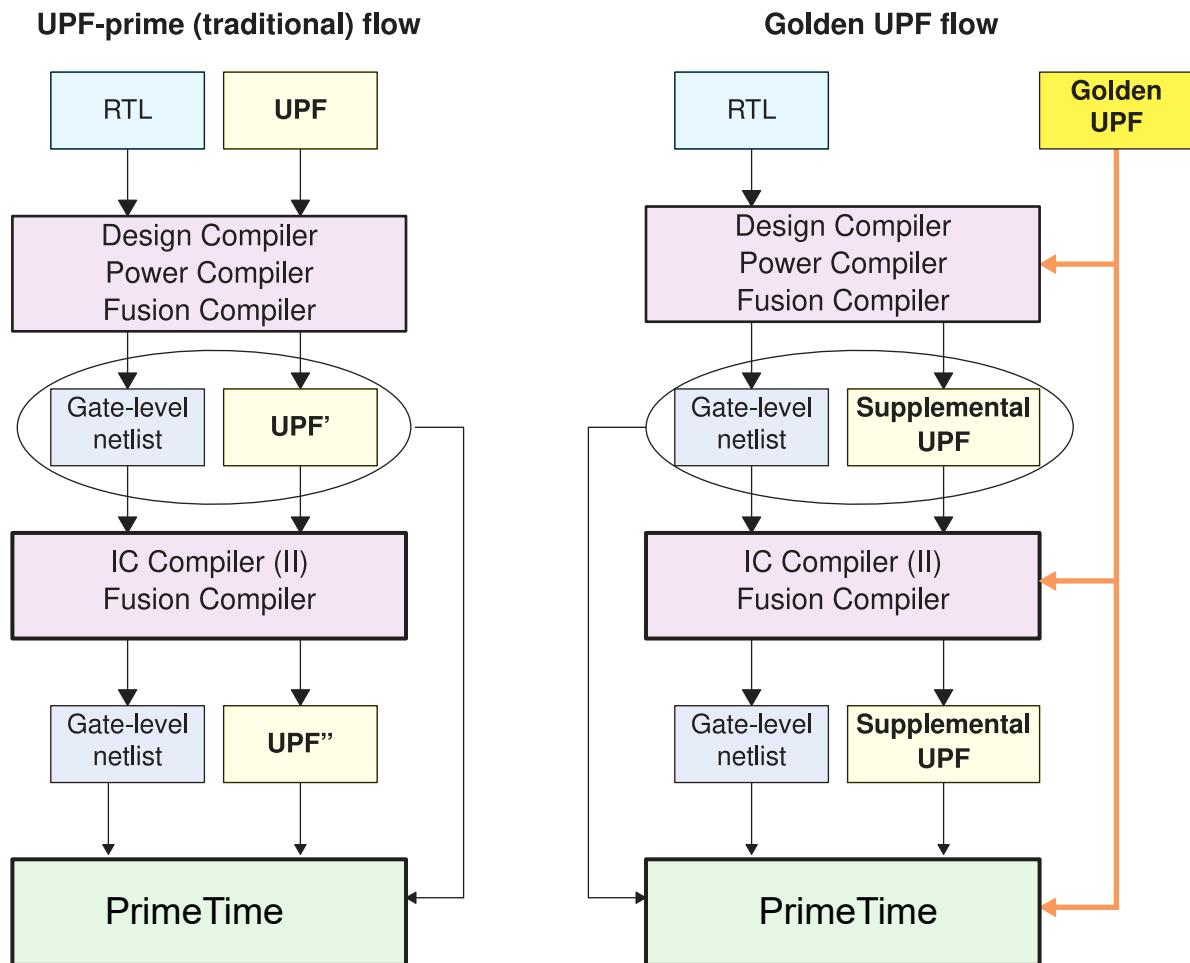
PrimeTime considers the fact that the supply voltages of the driver and receiver are correlated. When the supply voltage is low for U1, it is also low for U2, so there is no mismatch. The same is true for the case where the supply voltage is high.

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 synthesis and physical implementation tools.

[Figure 141](#) compares the traditional UPF-prime flow with the golden UPF flow.

Figure 141 UPF-Prime and Golden UPF Flows



The golden UPF flow maintains and uses the same, original “golden” UPF file throughout the flow. The synthesis and physical implementation 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 in PrimeTime SI, you must enable it by setting a variable:

```
pt_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 [SolvNetPlus article 000024504, "Golden UPF Flow Application Note."](#)

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 \(SMVA\)](#)
- [IR Drop Annotation](#)
- [Using Ansys RedHawk-SC With Timing Analysis](#)

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.
- Voltage values on PG nets specified with the `set_voltage` command.
- 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` variable and `define_scaling_lib_group` command in [Library Support for Multivoltage Analysis](#).

Simultaneous Multivoltage Analysis (SMVA)

In multivoltage designs, timing paths can cross from one power domain to another. When the power domains can operate at multiple supply voltages, traditional analysis cannot accurately analyze how the supply voltage combinations affect each path. If a single voltage is used for each domain, then the voltage combinations are not bounded. If min/max (on-chip variation) voltages are used for each domain, then the analysis is pessimistic because different cells in the same domain cannot operate at different voltages at the same time.

Graph-based simultaneous multivoltage analysis (SMVA) is an analysis method that removes this pessimism. Within-domain paths are analyzed at all voltage levels of the domain they belong to, and cross-domain paths are analyzed at all voltage level combinations of the voltage domains they traverse, all in a single run.

For more information on SMVA analysis, see [Chapter 15, SMVA Graph-Based Simultaneous Multivoltage Analysis](#).

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:

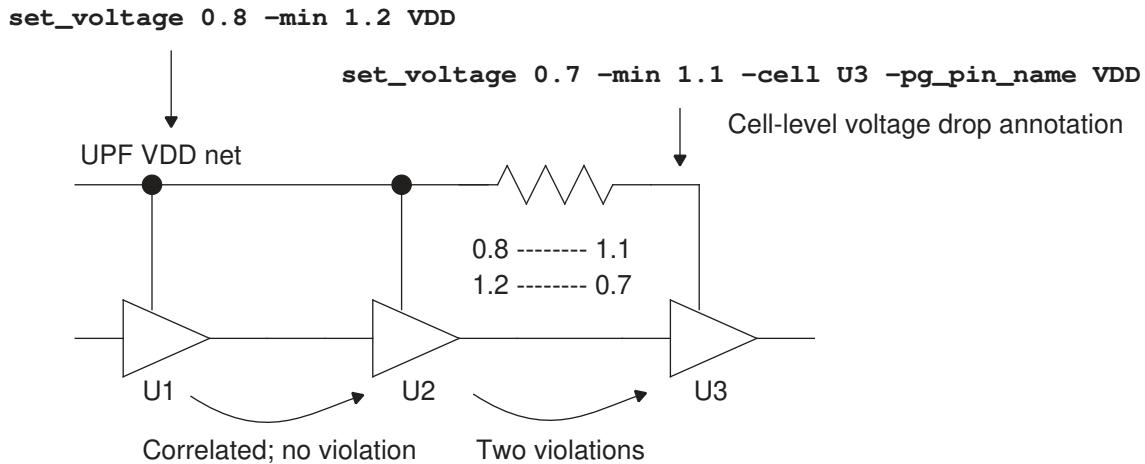
```
pt_shell> check_timing -include signal_level
```

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 142](#).

Figure 142 Signal Level Checking With Cell-Level Voltage Annotation



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.

Using Ansys RedHawk-SC With Timing Analysis

The RedHawk-SC power and reliability analysis tool can work together with PrimeTime to analyze IR drop and clock jitter effects on timing analysis.

Writing Ansys RedHawk-SC STA Files

The RedHawk-SC tool requires static timing analysis (STA) data from PrimeTime to perform IR drop, clock jitter, and dynamic voltage drop analysis.

To write this data,

1. Write design pin information by running the following command:

```

write_rh_file
-output string
[-filetype jitter | irdrop]
[-significant_digits digits]
  
```

The command performs a timing update if needed. It writes out the static timing analysis data in compressed (.gz) format, which is accepted directly by the RedHawk tool.

For dynamic voltage drop analysis, use the `-filetype irdrop` option.

Writing this data requires a PrimePower or PrimeTime-ADV-PLUS license.

2. (Optional) If you are using the critical-path scenario approach in RedHawk-SC, write path information for the desired timing paths using the `write_dvd_timing_paths` command:

```
set target_paths [get_timing_paths ...]
write_dvd_timing_paths -out input.json -paths $target_paths
```

For best results, use the `-vdd_slack_lesser_than` feature of the PrimeShield tool to find violating or near-violating paths that are sensitive to voltage variation. The following settings are recommended for good coverage:

```
set target_paths \
[get_timing_paths \
-vdd_slack_lesser_than 0.1 \
-path_type full_clock_expanded \
-pba_mode exhaustive \
-max_paths 100000]
```

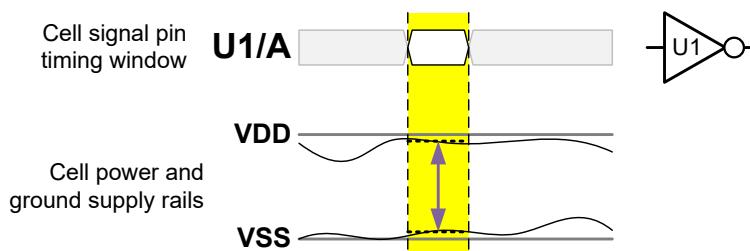
Note:

In the S-2021.06 release, the `write_dvd_timing_paths` command is provided as a Tcl procedure. To obtain it, contact your Synopsys support representative.

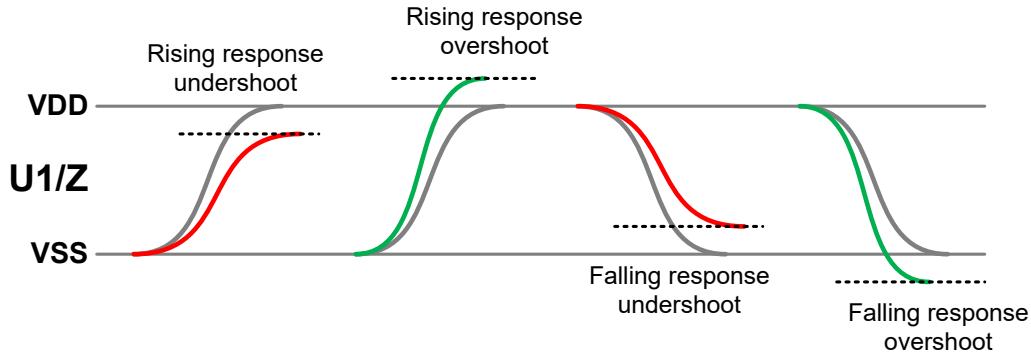
Performing Dynamic Voltage Drop Analysis

PrimeTime can read dynamic voltage drop data from the RedHawk-SC tool, then use it to analyze how dynamic variations in power and ground rail voltage affect design timing.

This feature considers only power and ground variations that occur within cell signal pin switching windows:



These power and ground variations are then used when computing cell signal pin responses, both for undershoot (slowdown) and overshoot (speedup) effects:



The effects are computed during path-based analysis, starting at the CRPR common point of each path.

This feature requires a PrimeTime-ADV-PLUS license.

To perform this analysis,

1. Configure voltage scaling library groups for the design, as described in [Cross-Library Voltage and Temperature Scaling](#).

The libraries must have CCS noise data.

2. Load, link, and constrain the design.
3. Enable the dynamic voltage drop feature by setting the following variable:

```
pt_shell> set_app_var timing_enable_dvd_analysis true
```

4. Read the dynamic voltage drop file from Ansys RedHawk-SC into PrimeTime:

```
pt_shell> read_dvd -mode 0|1 ./redhawk_data.dvd
```

Specify `-mode 0` for power rail variation and `-mode 1` for simultaneous power/ground rail variation.

5. Update the design timing:

```
pt_shell> update_timing
```

6. Use path-based analysis to analyze timing paths of interest with dynamic voltage variation effects included.

You must use the `-path full_clock_expanded` option of the `report_timing` or `get_timing_paths` command when doing this.

When the `timing_enable_dvd_analysis` variable is set to true, the `report_timing` output includes additional dynamic voltage effect information. For example,

```
pt_shell> report_timing \
    -pba_mode exhaustive \
    -path full_clock_expanded \
    -voltage \
    -input_pins
...
Startpoint: ff1 (rising edge-triggered flip-flop clocked by CLK)
Endpoint: ff2 (rising edge-triggered flip-flop clocked by CLK)
Last common pin: buf1/Y
Path Group: CLK
Path Type: max (recalculated)

Point           Incr      Path     Voltage   Delta_V   DvD_Delay
-----
clock CLK (rise edge) 0.000    0.000
clock source latency 0.000    0.000
clk (in)          0.000 & 0.000 r  0.570    0.000    0.000
buf1/A (BUF_X4)  -0.000 & 0.000 r  0.570    -0.022    0.000
buf1/Y (BUF_X4)  0.054 & 0.054 r  0.570    -0.022    0.005
ff1/CK (SDFFQ_X1) 0.001 & 0.055 r  0.570    -0.009    0.000
ff1/Q (SDFFQ_X1)  0.188    0.243 r  0.570    -0.009    0.017
inv2/A (INV_X2)  -0.000    0.243 r  0.570    0.000    0.000
inv2/Y (INV_X2)  0.037    0.279 f  0.570    0.000    0.002
and/B (AND2_X2)  -0.000    0.279 f  0.570    -0.046    0.000
and/Y (AND2_X2)  0.145 & 0.424 f  0.570    -0.046    0.012
or/B (OR2_X2)    0.010 & 0.435 f  0.570    -0.075    0.006
or/Y (OR2_X2)    0.192    0.627 f  0.570    -0.075    0.054
ff2/D (SDFFQ_X1) 0.000    0.627 f  0.570    -0.059    0.000
data arrival time           0.627

clock CLK (rise edge) 0.500    0.500
clock source latency 0.000    0.500
clk (in)          0.000 & 0.500 r  0.570    0.000    0.000
buf1/A (BUF_X4)  0.000 & 0.500 r  0.570    0.000    0.000
buf1/Y (BUF_X4)  0.050 & 0.550 r  0.570    0.000    0.000
buf2/A (BUF_X4)  0.000 & 0.550 r  0.570    0.000    0.000
buf2/Y (BUF_X4)  0.044 & 0.594 r  0.570    0.000    0.000
ff2/CK (SDFFQ_X1) 0.000 & 0.594 r  0.570    0.000    0.000
clock reconvergence pessimism
                           0.004    0.598
library setup time     -0.169    0.429
data required time    0.429

data required time    0.429
data arrival time     -0.627

slack (VIOLATED)      -0.198
DvD slack shift      -0.167
```

The `Delta_V` column indicates the reduced voltage distance between the power and ground rails, and the `DvD_Delay` indicates the resulting change in the cell delay. The `DvD slack shift` line at the end indicates the overall path slack change due to dynamic voltage drop effects.

To remove the dynamic voltage drop data and revert to regular timing analysis, use the following command:

```
pt_shell> remove_dvd
```

Using Rail Map Files to Apply Scaling/Offset Adjustments

In some cases, PrimeTime analysis might be run at a voltage corner that does not have dynamic voltage drop data available. In this case, you can use a *rail map file* to remap dynamic voltage drop data from one voltage corner to another.

A rail map file is a text file in the following format:

```
version: 1.0

object_type: supply_net
object_spec: supply_net1 [supply_net2 ... ]
scaling: value_list
offset: value_list
```

where the fields are as follows:

object_type

Specifies the object type being scaled. Only `supply_net` objects are supported.

object_spec

Specifies a list of one or more supply nets to scale.

scaling and offset

A list of one or two values, interpreted as follows:

- `value` (used for both clock and data)
- `data_value clock_value` (separate values for clock and data)

For example,

```
version: 1.0

object_type: supply_net
object_spec: VDD1
scaling: 0.85 0.80
offset: 0.02

object_type: supply_net
object_spec: VDD2
scaling: 0.85
offset: 0.021 0.019
```

For each original voltage value in the voltage drop input file, a remapped value is computed as follows:

$$V_{\text{new}} = (V_{\text{input}} * \text{value}_{\text{scaling}}) - \text{value}_{\text{offset}}$$

A positive offset value reduces the adjusted voltage value. If no remap rule applies to a supply net, no adjustment is applied (equivalent to a scaling value of 1 and an offset value of 0).

To read a rail map file in, use the `-rail_map` option of the `read_dvd` command:

```
set timing_enable_dvd_analysis true
read_dvd -mode 1 ir_drop.dvd
read_dvd -rail_map high2low_set1.map
```

Because the `read_dvd` command accepts only one file name at a time, the dynamic voltage drop file and the rail map file must be read in using separate commands.

To apply new rail map data in place of the existing rail map data, use the `-rail_map_only` option of the `remove_dvd` command to remove only the rail map data but not the voltage drop data:

```
remove_dvd -rail_map_only
read_dvd -rail_map high2low_set2.map
```

15

SMVA Graph-Based Simultaneous Multivoltage Analysis

SMVA graph-based simultaneous multivoltage analysis considers all combinations of supply voltages for each path simultaneously, for all paths in the whole design, in a single analysis run. To learn about SMVA analysis, see:

- [Overview of SMVA](#)
 - [Configuring SMVA Analysis](#)
 - [Reporting Timing Paths in an SMVA Analysis](#)
 - [DVFS Scenarios](#)
 - [Using SMVA Analysis With Other PrimeTime Features](#)
 - [Usage Examples](#)
 - [Commands and Attributes With DVFS Scenario Support](#)
 - [Feature Compatibility](#)
-

Overview of SMVA

In multivoltage designs, timing paths can cross from one power domain to another. When the power domains can operate at multiple supply voltages, traditional analysis cannot accurately analyze how the supply voltage combinations affect each path. If a single voltage is used for each domain, then the voltage combinations are not bounded. If min/max (on-chip variation) voltages are used for each domain, then the analysis is pessimistic because different cells in the same domain cannot operate at different voltages at the same time.

Graph-based simultaneous multivoltage analysis (SMVA) is an analysis method that removes this pessimism. Within-domain paths are analyzed at all voltage levels of the domain they belong to, and cross-domain paths are analyzed at all voltage level combinations of the voltage domains they traverse, all in a single run.

SMVA analysis is particularly useful for designs that use dynamic voltage and frequency scaling (DVFS) techniques to adjust the supply voltage and frequency during chip

operation, using more power when required for demanding tasks and less power at other times. You can get accurate timing results in these situations by providing scaling library groups and specifying the particular supply voltages of the power domains in the design.

SMVA Requirements

Graph-based SMVA analysis requires the following:

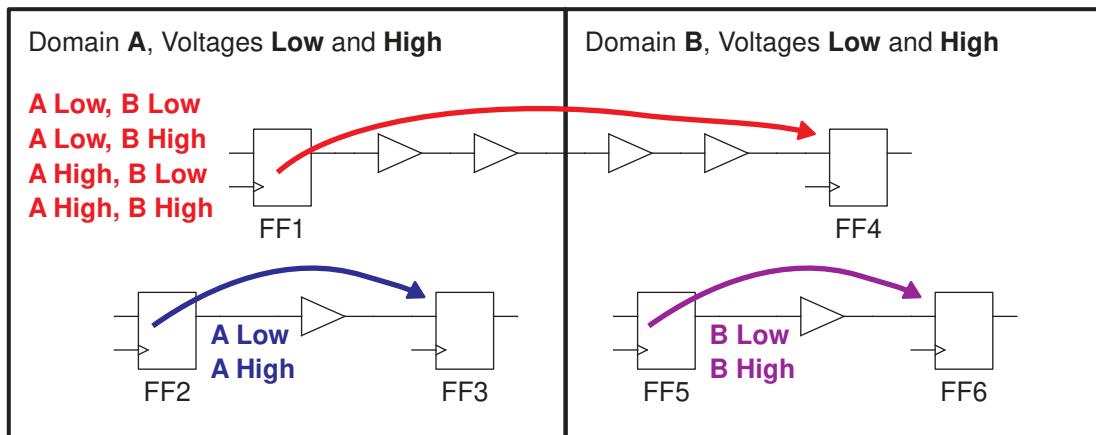
- The design must have multivoltage power intent information. For details, see [Multivoltage Analysis Requirements](#).
- The SMVA feature is enabled by the PrimeTime-ADV-PLUS license. The number of required licenses depends on the number of voltage levels and DVFS scenarios used in the analysis. For details, see [SMVA Licensing](#).

SMVA Power Domain Analysis

When the SMVA feature is enabled, graph-based analysis simultaneously considers all combinations of supply voltages for each path, for all paths in the design (both cross-domain and within-domain), excluding impossible combinations for each path and path segment, in a single analysis run.

For example, consider the paths shown in the following figure. The power supply for domains A and B can be independently set to either low or high voltage levels.

Figure 143 Supply Combinations for Cross-Domain and Within-Domain Paths



The within-domain timing paths starting at FF2 (blue) and FF5 (purple) are analyzed at both the low and high supply voltages of their respective domains. On the other hand,

the timing path starting at FF1 (red) is a cross-domain path, which is analyzed under four possible combinations of supply voltages for the two domains crossed by the path.

Each path operating under a given supply condition is treated as a separate path for collection and reporting purposes. For example, the physical path from FF1 to FF4 (red) can be reported up to four times by the `report_timing -nworst` command, once for each of the four possible supply combinations, for each type of timing constraint (setup and hold).

A timing path is identified as a cross-domain path if any timing arc in the data path or clock path crosses a power domain boundary. This can be a net arc, when the driving cell belongs to one domain and the receiver belongs to a different domain. It can also be a cell arc in the case of a level shifter.

The UPF infrastructure determines the power domain membership. Every cell must have its PG pins connected to a supply net, and each port should have a defined supply net. The tool recognizes supply net connections specified both by the Verilog PG netlist and by UPF commands (`connect_supply_net`). Specifying the supply connections using a Verilog PG netlist is the preferred method because it is more direct, concise, and reliable.

For any timing path that contains an object without supply net information, the tool finds the most likely supply from the design and library information and uses the single voltage value of that supply. To report missing information, you can use the following `check_timing` commands:

```
check_timing -include unconnected_pg_pins
check_timing -include supply_net_voltage
```

Configuring SMVA Analysis

To configure SMVA analysis, do the following before the first timing update:

1. Configure your design for multivoltage analysis, as described in [Chapter 14, Multivoltage Design Flow](#).
2. Enable SMVA analysis by setting the following variable:

```
set_app_var timing_enable_cross_voltage_domain_analysis true
```

3. Define reference voltage names and voltage values for the power domains, as described in [Defining Named SMVA Voltage References](#).

This information allows the tool to collect and combine multiple voltage corner configurations runs into a single run.

4. (Optional) Control or restrict the scope of SMVA analysis by setting the `timing_cross_voltage_domain_analysis_mode` variable to one of the following values:
 - `full` (the default) – The tool performs timing analysis and calculates slack values for all paths in the design, including cross-domain and within-domain paths. By default, all reporting and ECO commands consider all paths in the design.
 - `only_crossing` – The tool performs timing analysis on all paths in the design, but it calculates slack only for cross-domain paths. Reporting and ECO commands consider only cross-domain paths.
 - `exclude_crossing` – The tool performs timing analysis on all paths in the design, but it calculates slack only for within-domain paths. Reporting and ECO commands consider only within-domain paths.
 - `legacy` – The tool performs path-based analysis of cross-domain paths and ignores within-domain paths.

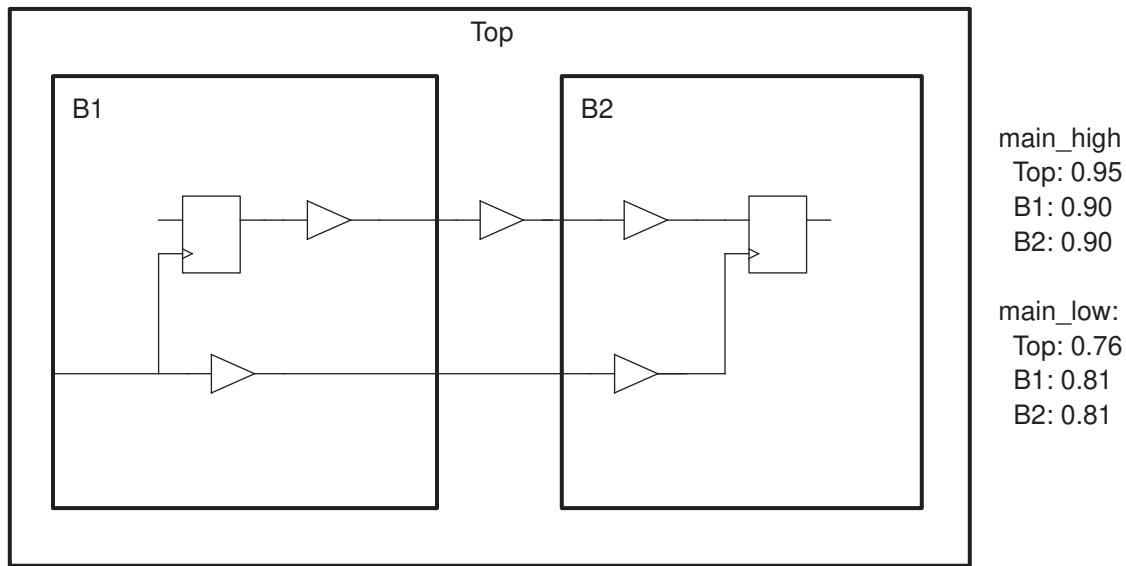
Defining Named SMVA Voltage References

To specify the allowed combinations of voltages in each domain for SMVA analysis, use the following commands:

- `get_supply_groups {supplies}`
 Creates a collection of supply groups that contain the specified supplies. To determine the membership of supply nets to supply groups, the command traces the supply connections using both the Verilog PG netlist and the UPF `connect_supply_net` commands.
- `set_voltage_levels -reference_names {names} supply_group_collection`
 Defines a set of named voltage references for a supply group collection, such as high and low, and assigns a name to each reference voltage.
- `set_voltage -reference_name name \
 -object_list supply_group_collection voltage`
 Associates specific voltage values with the reference names previously defined by the `set_voltage_levels` command.

For example, suppose that you have lower-level blocks named B1 and B2 in a top-level design named TOP. You have two reference voltage conditions, named `main_low` and `main_high`, corresponding to TOP supply voltages of 0.95 and 0.76, and block-level voltages of 0.90 and 0.81, as shown in the following figure.

Figure 144 Reference Voltage Conditions



To define these conditions for graph-based DVFS analysis, use the following commands:

```
set sg_block [get_supply_groups {B1/VDD B2/VDD}]
set_voltage_levels -reference_names {main_low main_high} $sg_block
set_voltage -reference_name main_high -object_list $sg_block 0.90
set_voltage -reference_name main_low -object_list $sg_block 0.81

set sg_top [get_supply_groups VDDTOP]
set_voltage_levels -reference_names {main_low main_high} $sg_top
set_voltage -reference_name main_high -object_list $sg_top 0.95
set_voltage -reference_name main_low -object_list $sg_top 0.76
```

The highlighted commands and options are available only for SMVA analysis.

Reporting Timing Paths in an SMVA Analysis

In an SMVA analysis, the `report_timing` command includes an additional line in each path header indicating the voltage configuration.

```
pt_shell> report_timing -through BLK2/C
...
Startpoint: BLK1/FF1 (rising edge-triggered flip-flop clocked by CLKB1[0])
Endpoint: BLK2/FF3 (rising edge-triggered flip-flop clocked by CLKB2[0])
Path Group: CLKB2[0]
Path Type: max
Voltage Config: VDDT:lo, VDD1:lo, VDD2:lo
...
```

In addition, you can use the `-supply_net_group` option to see supply group information along the timing path. For example,

```
pt_shell> report_timing -through BLK2/C -supply_net_group
...
Startpoint: BLK1/FF1 (rising edge-triggered flip-flop clocked by CLKB1[0])
Endpoint: BLK2/FF3 (rising edge-triggered flip-flop clocked by CLKB2[0])
Path Group: CLKB2[0]
Path Type: max
Voltage Config: VDDT:lo, VDD1:lo, VDD2:lo

Point           Incr      Path   Supply-Net-Group
-----
clock CLKB1[0] (rise edge)      0.00    0.00
clock network delay (propagated) 0.00    0.00
BLK1/FF1/CP (DFCND1LVT)        0.00    0.00 r  VDD1
BLK1/FF1/Q (DFCND1LVT)         0.21    0.21 r  VDD1
U1/Z (BUFFD2LVT)               0.07    0.28 r  VDDT
BLK2/C (block2) <-             0.00    0.28 r
BLK2/FF3/D (DFCND1LVT)         0.01    0.29 r  VDD2
data arrival time                  0.29

clock CLKB2[0] (rise edge)      10.00   10.00
clock network delay (propagated) 0.00    10.00
clock reconvergence pessimism   0.00    10.00
BLK2/FF3/CP (DFCND1LVT)        10.00   10.00 r
library setup time              -0.06   9.94
data required time                9.94
data arrival time                  9.94
-----
data required time                9.94
data arrival time                  -0.29
-----
slack (MET)                      9.65
```

Similarly, `timing_path` collection objects include a `dvfs_scenario` attribute whose `description` attribute can be queried:

```
pt_shell> get_attribute $path dvfs_scenario.description
{VDD2:lo, VDD1:lo, VDDT:lo}
```

As described in [SMVA Power Domain Analysis](#), each voltage condition of a timing path is treated as a separate path for collection and reporting purposes. For example, the `-nworst` option can return multiple voltage condition versions of the same path.

By default, SMVA analysis considers both within-domain and cross-domain paths. You can globally restrict this by setting the `timing_cross_voltage_domain_analysis_mode` variable.

You can also use the `-domain_crossing` option of the `report_timing` and `get_timing_paths` commands to restrict this on a per-command basis:

- `-domain_crossing all` – The command gathers or reports all paths, including both cross-domain and within-domain paths. This is the default behavior.
- `-domain_crossing only_crossing` – The command gathers or reports only cross-domain paths.
- `-domain_crossing exclude_crossing` – The command gathers or reports only within-domain paths.

DVFS Scenarios

For designs that use dynamic voltage and frequency scaling (DVFS), you can use *DVFS scenarios* to analyze the design under all DVFS conditions simultaneously. For details, see the following topics:

- [DVFS Scenario Concepts](#)
- [Querying DVFS Scenarios](#)
- [Applying DVFS Scenarios to Commands and Attributes](#)
- [DVFS-Related Object Attributes](#)

Note:

DVFS scenarios are voltage/frequency scenarios used in SMVA analysis. They are not related to the operating condition and constraint scenarios used in distributed multi-scenario analysis (DMSA).

DVFS Scenario Concepts

Simultaneous multivoltage analysis (SMVA) allows you to define multiple supply voltage groups in the design, then analyze all voltage combinations simultaneously in a single run. By itself, SMVA analysis varies only the supply voltages.

Dynamic voltage and frequency scaling (DVFS) is a design technique that adjusts supply voltage and frequency during chip operation, using more power when required for demanding tasks and less power at other times. In these designs, additional timing parameters (such as clock frequency) must also vary with voltage.

To properly analyze these designs, SMVA analysis provides a *DVFS scenario* feature that allows you to

- Apply design constraints to specific voltage conditions
- Run reporting and analysis commands under specific voltage conditions

DVFS Scenario Collection Objects

A DVFS scenario is represented by a `dvfs_scenario` collection object that captures a description of voltage conditions in the design. It can include a single supply group or multiple supply groups, and each supply group can include one or more named voltage references.

For example,

{VDDT:lo}	A single supply group at a single named voltage reference
{VDD1:lo sleep, VDDT:hi, VDD2:lo sleep}	Multiple supply groups, some with multiple allowable named voltage references

You can use the `get_dvfs_scenarios` command to construct a `dvfs_scenario` collection object. For example,

```
set my_dvfs_scenario1 [get_dvfs_scenarios \
    -supply_group VDDT -reference_names {lo}]

set my_dvfs_scenario2 [get_dvfs_scenarios \
    -supply_group VDD1 -reference_names {lo sleep} \
    -supply_group VDDT -reference_names {hi} \
    -supply_group VDD2 -reference_names {lo sleep}]
```

Both commands return a single `dvfs_scenario` collection object that describes the specified voltage condition requirements:

```
pt_shell> sizeof_collection $my_dvfs_scenario1
1
pt_shell> sizeof_collection $my_dvfs_scenario2
1
pt_shell> get_attribute $my_dvfs_scenario1 description
{VDDT:lo}
pt_shell> get_attribute $my_dvfs_scenario2 description
{VDD1:lo sleep, VDDT:hi, VDD2:lo sleep}
```

The order of supply groups and named voltage references is not important.

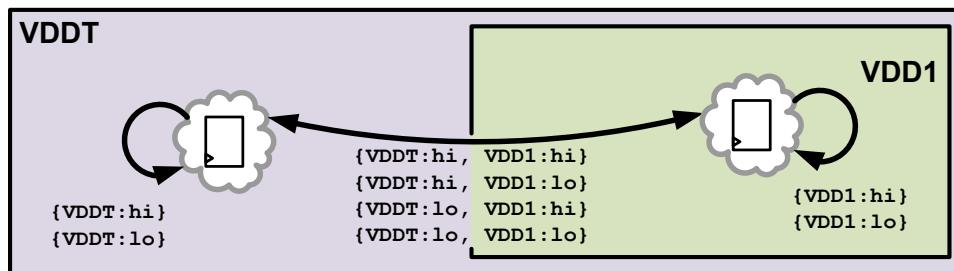
It is important to realize that `dvfs_scenario` collection objects are *constructed and used as needed*. The tool does not fully enumerate every possible partial and full voltage condition description in memory, as this would be computationally infeasible.

For simplicity, documentation outside this topic refers to “DVFS scenarios” instead of `dvfs_scenario` collection objects.

Propagated DVFS Scenarios

Each timing path has a set of DVFS scenarios that fully describe its possible voltage conditions. For example, a within-domain and cross-domain timing path might have the following fully describing DVFS scenarios:

Figure 145 Fully Describing DVFS Scenarios for Individual Timing Paths

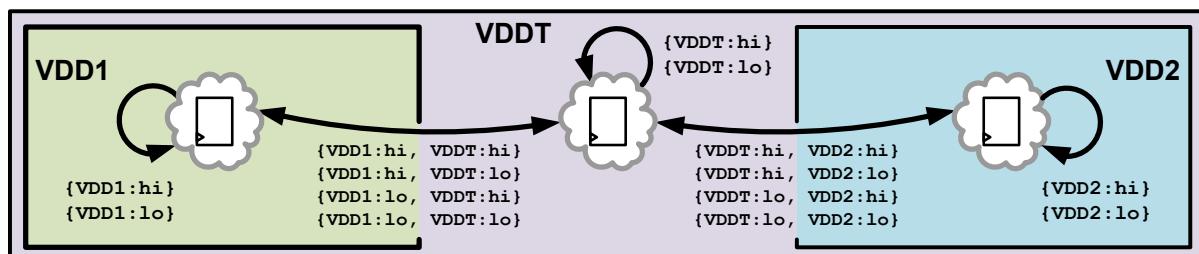


A DVFS scenario that is fully describing for one path might be partially describing for another. For example, {VDDT:hi} is fully describing for the path on the left but partially describing for the path on the right.

If you obtain the fully describing DVFS scenarios for every path in the design, you get the *propagated DVFS scenarios* for that design.

For example, a design with three voltage supply groups (VDDT, VDD1, and VDD2) with various within-domain and cross-domain paths might have the following propagated DVFS scenarios:

Figure 146 The Propagated DVFS Scenarios for a Design



A propagated DVFS scenario always has a single named voltage reference per supply group.

Using DVFS Scenarios to Control Command Scope

When you use a DVFS scenario with a constraint or reporting command, the command applies to the propagated DVFS scenarios that are *compatible* with the command-specified DVFS scenario.

A propagated DVFS scenario is compatible with a command-specified DVFS scenario if *there is no conflict in the supply group named voltage references between them*.

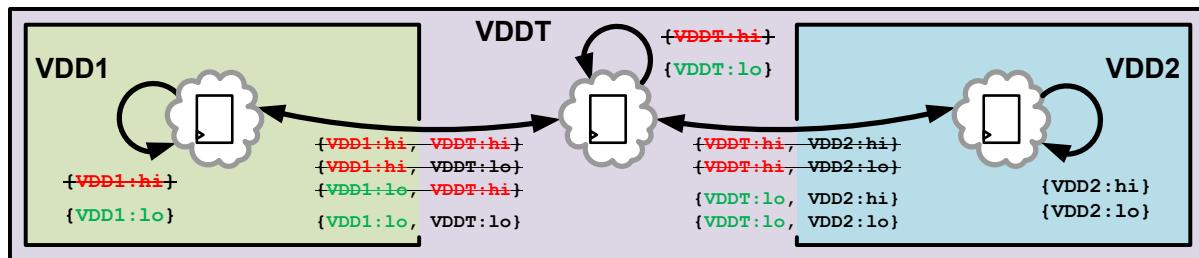
In the following example, a DVFS scenario of `{VDD1:lo VDDT:lo}` is given to the `report_timing` command, causing the command to exclude propagated DVFS scenarios with named voltage references other than those explicitly allowed:

Figure 147 Applying a DVFS Scenario to a Command

```
report_timing -dvfs_scenarios \
  [get_dvfs_scenarios \
    -supply_group VDD1 -reference_names {lo} \
    -supply_group VDDT -reference_names {lo}]]
```

Excludes paths with `{VDD1:hi}`

Excludes paths with `{VDDT:hi}`



In this example, note that paths entirely within VDD2 are not excluded. When you specify a DVFS scenario for a command, it *excludes* what is incompatible with the specified DVFS scenario; it does not limit inclusion to only the specified scenario conditions.

Querying DVFS Scenarios

DVFS scenarios are referenced as collection objects. You can use the `get_dvfs_scenarios` command to query and return them.

Constructing a DVFS Scenarios for Supply Group Conditions

To construct a DVFS scenario for particular supply group voltage conditions, specify one or more `-supply_group` and `-reference_names` option pairs:

```
pt_shell> get_dvfs_scenarios \
  -supply_group $sg_top -reference_names {low}

pt_shell> get_dvfs_scenarios \
  -supply_group $sg_blk1 -reference_names {low sleep} \
  -supply_group $sg_top -reference_names {high} \
  -supply_group $sg_blk2 -reference_names {low sleep}
```

This returns a DVFS scenario collection object with the specified conditions. See [DVFS Scenario Collection Objects](#) for details.

Use this method to construct a DVFS scenario to be passed to a constraint or reporting command.

Obtaining All Propagated DVFS Scenarios

To obtain the full set of propagated DVFS scenarios for the current design, use the `-propagated` option of the `get_dvfs_scenarios` command:

```
pt_shell> get_dvfs_scenarios -propagated
```

A logical design update is performed if the design is not currently updated.

The result is a collection of multiple `dvfs_scenario` collection objects, each corresponding to one or more timing paths in the design. See [Propagated DVFS Scenarios](#) for details.

This collection of propagated DVFS scenarios can be used for design exploration. Note that not all commands with the `-dvfs_scenarios` option support multiple DVFS scenario collection objects as input.

Applying DVFS Scenarios to Commands and Attributes

By default, commands and attributes that support DVFS scenarios consider all DVFS scenarios. The following topics describe how to limit these commands and attributes to a subset of DVFS scenarios:

- [Applying DVFS Scenarios to Individual Commands](#)
- [Applying DVFS Scenarios to Attribute Queries](#)
- [Applying DVFS Scenarios to Scripts](#)
- [Setting a DVFS Scenario Context for Command Execution](#)
- [Disabling and Enabling DVFS Scenarios](#)
- [DVFS Scenario Specification Precedence](#)

For details on which commands and attributes support DVFS scenarios, see [Commands and Attributes With DVFS Scenario Support](#).

Applying DVFS Scenarios to Individual Commands

Commands that have a `-dvfs_scenarios` option allow you to explicitly specify the DVFS scenarios for the command to consider:

```
pt_shell> report_timing -dvfs_scenarios $my_dvfs_scenarios
```

Explicitly specified DVFS scenarios take precedence over any script-based or context-based DVFS scenario settings.

Applying DVFS Scenarios to Attribute Queries

To query the value of an attribute in specific DVFS scenarios, include a `dvfs_scenarios` collection object in parentheses after the attribute name. For example,

```
set low_low [get_dvfs_scenarios \
    -supply VDD_top -reference_names {low} \
    -supply VDD_block -reference_names {low}]

get_attribute $pin pin_capacitance_max($low_low)
get_attribute $pin net.pin_capacitance_max($low_low)
report_attribute -attribute {pin_capacitance_max($low_low)} $pin
get_pins -filter {pin_capacitance_max($low_low) > 0.1}
```

If the collection contains multiple DVFS scenarios, the worst-case value is returned, which is the minimum or maximum value according to the attribute type. If the per-scenario values differ but cannot be merged (such as for string attributes), the tool issues an ATTR-11 warning.

If a queried attribute does not support DVFS scenarios, the tool issues an ATTR-3 error.

Attribute subscripts take precedence over any script-based or context-based DVFS scenario settings.

Applying DVFS Scenarios to Scripts

To apply DVFS scenarios to an entire SDC constraint file or PrimeTime script, use the `-dvfs_scenarios` option of the `read_sdc` or `source` command, respectively:

```
pt_shell> source -dvfs_scenarios [get_dvfs_scenarios ...]
pt_shell> read_sdc -dvfs_scenarios [get_dvfs_scenarios ...]
```

These commands are equivalent to using the `-dvfs_scenarios` option with each command that supports the option, and subscripting each attribute query that supports such access.

When you use this method, consider the following:

Not all SDC commands support the `-dvfs_scenarios` option. If such a command is encountered by the `read_sdc` or `source` command, the latest command has priority over earlier ones. To avoid ambiguous settings, it is recommended to use only the commands that support the `-dvfs_scenarios` option in the constraint script.

Commands that do not currently support the `-dvfs_scenarios` option might be enhanced in future releases to support it, resulting in changes to results. This is another reason to use only commands that support the `-dvfs_scenarios` option.

Setting a DVFS Scenario Context for Command Execution

To apply DVFS scenarios to an arbitrary sequence of commands, including scripts and Tcl procedures, use the `push_dvfs_scenario` and `pop_dvfs_scenario` commands:

```
pt_shell> push_dvfs_scenario -dvfs_scenario $low_low
pt_shell> # ...analysis commands for low_low DVFS scenario...
pt_shell> pop_dvfs_scenario
```

The `push_dvfs_scenario` command creates a DVFS scenario context that applies to subsequent commands and attribute queries that support DVFS scenario use. The `pop_dvfs_scenario` command removes the most recently applied context.

DVFS scenario contexts can be nested. In this case, there is no interaction between the inner and outer DVFS contexts; the inner context only is used until it is popped.

Timing updates within DVFS contexts are not supported.

Disabling and Enabling DVFS Scenarios

You can use the `-disable` and `-enable` options of the `configure_dvfs_scenarios` command to control which DVFS scenarios are included in the analysis.

For example,

```
pt_shell> configure_dvfs_scenarios -disable $low_low
pt_shell> # ...analysis with $low_low omitted...
pt_shell> configure_dvfs_scenarios -enable $low_low
```

The `report_dvfs_scenarios -disabled` command reports any currently disabled scenarios:

```
pt_shell> report_dvfs_scenarios -disabled
...
    Current Ignored Scenarios:
-----
B1/VDD:low VDDTOP:low
```

Disabled DVFS scenarios are omitted entirely from the analysis. Timing updates do not compute their data, and reporting does not consider them.

Any enabling or disabling of DVFS scenarios requires a full timing update (`update_timing` with the `-full` option). By default, the `update_timing` command performs an incremental timing update, which is not sufficient to update the DVFS scenario changes.

An alternative method is to set a false path to everything in the DVFS scenarios to be disabled. For example,

```
pt_shell> set top_high_block_low [get_dvfs_scenarios ...]
pt_shell> set_false_path -to * -dvfs_scenarios $top_high_block_low
```

DVFS Scenario Specification Precedence

When multiple DVFS scenario reference methods are used, the most specific method (“closest” to the command) takes precedence.

Explicit DVFS scenario references (the `-dvfs_scenarios` option of constraint and reporting commands, scenario-specific attribute queries) have the highest precedence. For example,

- `report_timing -dvfs_scenarios $dvfs_scenario`
- `get_attribute $pin power_rail_voltage_max($dvfs_scenario)`

Implicit references via DVFS scenario contexts have a lower precedence:

- `read_sdc -dvfs_scenarios $dvfs_scenario`
- `source -dvfs_scenarios $dvfs_scenario`
- `push_dvfs_scenario $dvfs_scenario`

All three implicit reference methods have equal precedence; the last applied implicit context wins.

The `get_current_dvfs_scenario` command returns the current DVFS scenario context, or 0 if no context exists.

Lower-precedence DVFS scenario references are completely ignored when a higher-precedence reference applies.

DVFS-Related Object Attributes

You can query the attributes in the following table to get SMVA information about timing paths and DVFS scenarios.

Attribute name	Class	Type	Description
<code>dvfs_scenario</code>	<code>timing_path</code>	<code>dvfs_scenario object</code>	<code>dvfs_scenario</code> associated with the timing path

Attribute name	Class	Type	Description
has_voltage_domain_crossing	timing_path	Boolean	true for a cross-domain path (a path reported by report_timing -domain_crossing only_crossing)
supply_groups	dvfs_scenario	supply_group object collection	The collection of supply groups relevant to the dvfs_scenario

For example, to iterate through a path collection and filter the cross-domain paths, use a script similar to the following:

```
set my_path_collection [get_timing_paths ...]
foreach_in_collection path $my_path_collection {
    if {[get_attribute $path has_voltage_domain_crossing]} {
        // Take action regarding voltage-domain-crossing $path here
        ...
    }
}
```

Using SMVA Analysis With Other PrimeTime Features

The following topics describe how SMVA analysis interacts with other PrimeTime features:

- [HyperScale Hierarchical Analysis](#)
- [Voltage-Scaled Timing Derates in SMVA Flows](#)
- [Extracted Timing Models](#)
- [Noise Analysis](#)

HyperScale Hierarchical Analysis

The SMVA feature is compatible with HyperScale hierarchical analysis. The flow uses accurate aggressor modeling and context annotations (such as arrival times and latency values) at the top-to-block hierarchical boundaries for all voltage configurations:

- For HyperScale block models written by the `write_hier_data` command during block-level analysis, data for all DVFS scenarios is captured. For example, arrival times and cross-talk aggressor behavior are tracked and captured for each voltage configuration. The block-level model is used during top-level analysis and allows you to analyze all DVFS scenarios as if the run is flat.

- For HyperScale context information written by the `write_hier_data` command during top-level analysis, the context data accurately represents all DVFS scenarios. The merged context data from the top level is used during block-level analysis, and allows you to analyze all DVFS scenarios as if the run is flat.

In other words, HyperScale SMVA allows you to analyze both the block and top levels for all DVFS scenarios, providing the same timing as seen in a flat run, while also allowing the reduction in memory and runtime consumption of HyperScale technology.

The HyperScale flow requires consistent SMVA setup between the block level and top level, including enabling SMVA, voltage domains, voltage levels, DVFS scenarios, and so on, in addition to the consistency requirements for non-SMVA runs such as clocks, libraries, and timing exceptions. Note that the tool does not perform SMVA setup consistency checking between the top and block levels, so inconsistent SMVA setup may result in unpredictable errors.

The following script examples demonstrate HyperScale block-level and top-level analyses with the SMVA feature.

```
set hier_enable_analysis true
# HyperScale block-level session

set link_path {* *.db}
# Without UPF 2.0 commands impcicit supply_sets are not needed
set upf_create_implicit_supply_sets false
read_verilog block.v.gz
link_design block

set timing_enable_cross_voltage_domain_analysis true

read_parasitics -format spef block.spef
# Power Supply Definitions
create_power_domain B -elements [get_cells {"ff1" "iv1" ... }]
create_power_domain T #virtual top domain

create_supply_net sn_T -domain T
create_supply_net gn_T -domain T

create_supply_net sn_B -domain B
create_supply_net gn_B -domain B

set_domain_supply_net -primary_power_net sn_T -primary_ground_net gn_T T
set_domain_supply_net -primary_power_net sn_B -primary_ground_net gn_B B

set_voltage_levels sn_T -reference_names {hi lo}
set_voltage_levels sn_B -reference_names {hi lo}

set_voltage -object_list sn_T -reference_name lo 0.80 -min 0.90
set_voltage -object_list sn_T -reference_name hi 1.10 -min 1.20
set_voltage -object_list sn_B -reference_name lo 0.70 -min 0.80
```

```

set_voltage -object_list sn_B -reference_name hi 1.00 -min 1.10

create_clock -name CK1 -period 10 clk1 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_B -reference_names hi]
create_clock -name CK1 -period 12 clk1 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_B -reference_names lo] -add

create_clock -name CK2 -period 10 clk2 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_T -reference_names hi]
create_clock -name CK2 -period 12 clk2 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_T -reference_names lo] -add

# different cycles for specific scenarios
set_multicycle_path 3 -from CK1 -to CK2 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_B -reference_names hi \
-supply_group sn_T -reference_names lo ]
set_multicycle_path 4 -through iv1/Z -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_B -reference_names hi \
-supply_group sn_T -reference_names lo ]

# specify non DVFS timing constraints
read_sdc block.sdc

# read_context and apply clock mapping if any:
read_context ./top
set_clock_map CK1 -parent_clocks CK1
set_clock_map CK2 -parent_clocks CK2

# run "full" SMVA timing analysis
update_timing

# Write out the model data
write_hier_data ./block

-----
# HyperScale top-level session

set link_path {* *.db}
# without UPF 2.0 commands impcicit supply_sets are not needed
set upf_create_implicit_supply_sets false

set_hier_config -block block -path ./block

read_verilog top.v.gz
link_design top

set timing_enable_cross_voltage_domain_analysis true
set timing_cross_voltage_domain_analysis_mode full

read_parasitics -format spef top.spef

```

```

# Power Supply Definitions

create_power_domain T -elements [get_cells {"fft" "a" ... }]

create_supply_net sn_T -domain T
create_supply_net gn_T -domain T

set_domain_supply_net -primary_power_net sn_T -primary_ground_net gn_T T
set_voltage_levels sn_T -reference_names {hi lo}
# UPF for block will be loaded from HS model as B/sn_B
set_voltage_levels B/sn_B -reference_names {hi lo}

set_voltage -object_list sn_T -reference_name lo 0.80 -min 0.90
set_voltage -object_list sn_T -reference_name hi 1.10 -min 1.20
set_voltage -object_list B/sn_B -reference_name lo 0.70 -min 0.80
set_voltage -object_list B/sn_B -reference_name hi 1.00 -min 1.10

create_clock -name CK1 -period 10 clk1 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group B/sn_B -reference_names hi]
create_clock -name CK1 -period 12 clk1 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group B/sn_B -reference_names lo] -add

create_clock -name CK2 -period 10 clk2 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_T -reference_names hi]
create_clock -name CK2 -period 12 clk2 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group sn_T -reference_names lo] -add

# different cycles for specific scenarios
set_multicycle_path 3 -from CK1 -to CK2 -dvfs_scenarios \
[get_dvfs_scenarios -supply_group B/sn_B -reference_names hi \
-supply_group sn_T -reference_names lo ]
set_multicycle_path 4 -through B/iv1/Z -dvfs_scenarios \
[get_dvfs_scenarios -supply_group B/sn_B -reference_names lo \
-supply_group sn_T -reference_names hi ]

# specify non DVFS timing constraints
read_sdc design.sdc
set_clock_map CK1 -block_clocks CK1
set_clock_map CK2 -block_clocks CK2

update_timing

# write block context
write_hier_data ./top

```

Context Characterization Flow

The SMVA feature is compatible with the context characterization flow using the `characterize_context` and `write_context` commands. The generated context uses accurate delay and latency across all DVFS scenarios. The output format of the `write_context` command must be in binary context format (`-format gbc`).

Also, the `hier_characterize_context_mode` variable must be set to `full_context` (the default), not `constraints_only`.

Voltage-Scaled Timing Derates in SMVA Flows

The `set_timing_derate` command allows you to specify early or late timing adjustment values on different object scopes. In SMVA flows, you can apply voltage-specific timing derates, and the tool dynamically scales them as needed during the SMVA analysis.

To enable this feature, set the following variables:

```
pt_shell> set_app_var \
    timing_enable_derate_scaling_for_library_cells_compatibility false

pt_shell> set_app_var \
    timing_enable_derate_scaling_interpolation_for_library_cells true
```

For details, see [Scaling Timing Derates in Voltage Scaling Flows](#).

With these settings applied, if you set timing derates on library cells in scaling group libraries, the tool scales the derate values as needed according to the instance-specific voltages computed per DVFS scenario. For example,

```
define_scaling_lib_group {lib_1.1V.db lib_1.0V.db lib_0.9V.db}

# define timing derates on scaling library cells
set_timing_derate -late [get_lib_cell lib_1.1V/INV1] 1.07
set_timing_derate -late [get_lib_cell lib_1.0V/INV1] 1.10
set_timing_derate -late [get_lib_cell lib_0.9V/INV1] 1.15

# define voltages for SMVA DVFS scenarios
set_voltage_levels sg1 -reference_names {hi lo}
set_voltage -object_list sg1 -reference_name lo 0.90 ;# exact derate
set_voltage -object_list sg1 -reference_name hi 1.05 ;# scaled derate
```

In this example, the following derating values apply to INV1 cells in DVFS scenarios:

- sg1:lo – The low voltage 0.90 is an exact match with the 0.9V library, so the derating value 1.15 set on the INV1 cell in that library cell applies.
- sg1:hi – The high voltage 1.05 is not an exact match with any library. The derating values set on INV1 library cells in the two nearest libraries (1.0V and 1.1V) are 1.07 and 1.10. The worse value is used, 1.10.

The `set_timing_derate` command also has the `-dvfs_scenarios` option, allowing you to directly specify the DVFS scenarios to which cell derating applies. This is supported only for leaf cell instances, and the DVFS scenario can only reference supplies for that cell. As with ordinary derating, a `set_timing_derate` setting applied to a cell instance overrides a more general setting on the corresponding library cell.

Extracted Timing Models

The SMVA feature is compatible with extracted timing model (ETM) blocks used in a scaling library group. To generate a timing model, you need to provide characterization data for the ETM at each voltage corner.

The `extract_model` command in SMVA mode automatically produces one ETM file for each active DVFS scenario (or voltage combination) relevant to the design. When `-dvfs_scenarios` is used, the `extract_model` command automatically extracts one model for each DVFS scenario compatible with the scenario specified with `-dvfs_scenarios` option.

The names of extracted models are formed by combining the output model name specified in the `extract_model` command, the voltage domain names, and voltage level names. For example, suppose that you have voltage domains SN1 and SN2, and each domain has two configurations named low and high. The command `extract_model -output test` generates the following library model files:

```
test_SN1_low_SN2_low.lib
test_SN1_low_SN2_high.lib
test_SN1_high_SN2_low.lib
test_SN1_high_SN2_high.lib
```

The tool performs delay calculation for the ETM using the delay tables from the appropriate voltage corner based on the DVFS scenarios explored during SMVA analysis.

The ETM blocks extracted at different voltages must pass all library group formation criteria, including sharing the same number and types of timing arcs. Only the values in the timing tables can be different.

In some cases, extraction at different supply voltages can result in different timing arcs. For example, a transparent latch might be characterized as transparent in one model and not in the other, due to different arrival times within the block. In that case, attempting to create a scaling library group with these blocks triggers an error, and using the models in this manner is not supported.

Timing Arc Multiple-Rail Delay Dependency

With the SMVA feature enabled, the PrimeTime tool supports complex macro cells in which the timing arcs of a library cell depend on PG pins other than the PG pins related to the “from” and “to” pins of the arc. For example, a low-power memory cell might have constraint arcs that depend on the states of both the main and backup power supplies. An extra PG supply on which the timing arc depends is called a virtual PG pin.

The mapping of constraint arcs to virtual PG pins is not directly supported in Liberty syntax. Instead, the PrimeTime tool supports usage of a user-defined `lib_timing_arc` attribute called `related_power_pins_user`. The PrimeTime tool imports the related

power pin information from a user-defined attribute in the Liberty description of the cell. You can also manually set this attribute per library timing arc.

For example, the following .lib description of a memory cell defines the related power pin attribute and sets the attribute to the pin name VDD_SRAM.

```
# Modifications to Liberty .lib file
library (MEM_1_0.81) {
    define (related_power_pins_user, timing, string);
    ...
    cell("dual_rail_ram") {
        ...
        pin (D) {
            direction : input;
            related_power_pin : VDD;
            timing() {
                related_pin : "CLK"
                timing_type : setup_rising;
                related_power_pins_user : "VDD_SRAM";
            }
        }
    }
}
```

The following PrimeTime script defines the attribute and imports the PG pin mapping information.

```
# Import user-defined attribute
define_user_attribute related_power_pins_user \
    -classes lib_timing_arc -type string -import

# Add one of the SRAM libraries to the link path
set_link_path {MEM_1_0.81 <other libs needed in design>}

read_verilog top.v

# Set up scaling library group for SRAM
define_scaling_lib_group \
    { MEM_1_0.81 MEM_1_0.72 MEM_0.9_0.81 MEM_0.9_0.72 }

# Other scaling groups in design go here ...

# Enable SMVA graph-based analysis
set timing_enable_cross_voltage_domain_analysis true

load_upf chip_connectivity.upf

# Define voltage levels to explore in SMVA analysis
set_voltage_levels {VDD1 VDD2} -reference_names {low high}
set_voltage -supply_group VDD1 -reference_name low 0.9
set_voltage -supply_group VDD1 -reference_name high 1
set_voltage -supply_group VDD2 -reference_name low 0.72
set_voltage -supply_group VDD2 -reference_name high 0.81
```

```
read_parasitics top.spf
read chip.sdc
update_timing -full
```

The `lib_timing_arc` attribute named `has_arc_related_power_pins` returns true when the PG pin dependence of this timing arc has been augmented with the specification of a `related_power_pins_user` user-defined attribute.

The `lib_timing_arc` attributes named `related_power_pins`, `unrelated_power_pins`, and `has_arc_related_power_pins` provide information about the related power pins that affect (or do not affect) the timing arc.

Noise Analysis

You can analyze noise in an SMVA analysis, provided that

- You define DVFS scenarios for the SMVA analysis.
- You analyze noise in a single DVFS scenario at a time.

The `update_noise` command provides a `-dvfs_scenarios` option that accepts a single scenario. The scenario must be fully describing (specifies a voltage name for all supply groups in the design).

The `report_noise` command reports the noise results from the last DVFS scenario updated with the `update_noise -dvfs_scenarios` command.

You can use these commands as follows to analyze noise across multiple DVFS scenarios:

```
# apply noise settings across all DVFS scenarios here
set_input_noise ...
set_noise_lib_pin ...
set_noise_margin ...

foreach_in_collection dvfs_scenario [get_dvfs_scenarios ...] {
    # apply per-DVFS-scenario noise settings here
    set_input_noise ...
    set_noise_lib_pin ...
    set_noise_margin ...

    # update noise information, report the results
    update_noise -dvfs_scenarios $dvfs_scenario
    report_noise ...
}
```

Noise settings that are common across all DVFS scenarios can be specified once before any noise analysis is run. Noise settings that vary by DVFS scenario must be specified just before analyzing that specific scenario.

The `update_noise` command also considers DVFS contexts applied by the following commands:

- `push_dvfs_scenario`
- `source -dvfs_scenario`
- `read_sdc -dvfs_scenario`

Usage Examples

The following usage examples demonstrate the SMVA flow:

- [SMVA Analysis for All Voltage Conditions](#)
- [SMVA Analysis for Specific DVFS Constraints](#)

In the following script examples, the text highlighted in red shows the commands, command options, and variables used in the SMVA flow. These features can be used only when the `timing_enable_cross_voltage_domain_analysis` variable is set to true.

SMVA Analysis for All Voltage Conditions

To perform SMVA analysis of all combinations of multivoltage conditions, use the following procedure:

1. Enable cross-domain analysis and SMVA analysis.
2. Define the scaling library groups.
3. Load the UPF specification.
4. Specify the supply groups and reference voltage levels.
5. Specify voltage values for each reference voltage level.
6. Run the timing analysis.
7. (Optional) Generate additional reports for specific voltage configurations.

The following example demonstrates the flow.

```
set_app_var link_path {* lib_1.0V_pg.db \
lib_1.0V_lvt_pg.db lib_1.0V_hvt_pg.db}
```

```

read_verilog mydesign.v.gz
current_design mydesign
link

# 1. Enable cross-domain analysis and define analysis type
set_app_var timing_enable_cross_voltage_domain_analysis true

# 2. Define scaling library groups
define_scaling_lib_group \
    "lib_1.0V_pg.db lib_0.9V_pg.db lib_0.8V_pg.db"
define_scaling_lib_group \
    "lib_1.0V_lvt_pg.db lib_0.9V_lvt_pg.db lib_0.8V_lvt_pg.db"
define_scaling_lib_group \
    "lib_1.0V_hvt_pg.db lib_0.9V_hvt_pg.db lib_0.8V_hvt_pg.db"

# 3. Read UPF specification
load_upf mydesign.upf
# Ensure that all ports have a related supply net assigned as needed
set_related_supply_net -object_list [get_ports *] -power VDD -ground VSS

# 4. Specify supply groups and reference voltage levels
set sg [get_supply_groups VDDTOP]
set_voltage_levels -reference_names { main_low main_high } $sg

# 5. Specify voltage values for each reference voltage level
set_voltage -reference_name main_high -object_list $sg 1.0
set_voltage -reference_name main_low -object_list $sg 0.86

# Optional instance-specific set_voltage commands go here

# Additional timing checks for PG pins and supply nets
check_timing -override_defaults unconnected_pg_pins
check_timing -override_defaults supply_net_voltage

# Specify timing constraints (other than DVFS)
read_sdc mydesign.sdc

# 6. Run a "full" SMVA timing analysis
update_timing

# Report worst path across all possible voltage configurations
report_timing

# 7. Generate additional reports for specific voltage configurations
#     [Optional]
set vdd_main_low [get_dvfs_scenarios \
    -supply_group [get_supply_groups VDDTOP] -reference_names main_low]
report_timing -dvfs_scenarios $vdd_main_low

```

SMVA Analysis for Specific DVFS Constraints

To perform SMVA analysis of specific combinations of multivoltage conditions, use the following procedure:

1. Enable cross-domain analysis and SMVA analysis.
2. Define the scaling library groups.
3. Load the UPF specification.
4. Specify the supply groups and reference voltage levels.
5. Specify voltage values for each reference voltage level.
6. *Specify the DVFS scenarios and the related design constraints.*
7. Run the timing analysis.
8. (Optional) Generate additional reports for specific voltage configurations.

Each combination of voltage supply values for the power domains is called a dynamic voltage and frequency scaling (DVFS) scenario.

Note:

DVFS scenarios are not related to operating condition scenarios analyzed in distributed multi-scenario analysis (DMSA). DVFS multivoltage scenarios are analyzed in a single run, whereas DMSA operating condition scenarios are analyzed in distributed runs.

The following example demonstrates the flow.

```
set_app_var link_path {* lib_1.0V_pg.db \
    lib_1.0V_lvt_pg.db lib_1.0V_hvt_pg.db}
read_verilog mydesign.v.gz
current_design mydesign
link

# 1. Enable cross-domain analysis and define analysis type
set_app_var timing_enable_cross_voltage_domain_analysis true

# 2. Define scaling library groups
define_scaling_lib_group \
    "lib_1.0V_pg.db lib_0.9V_pg.db lib_0.8V_pg.db"
define_scaling_lib_group \
    "lib_1.0V_lvt_pg.db lib_0.9V_lvt_pg.db lib_0.8V_lvt_pg.db"
define_scaling_lib_group \
    "lib_1.0V_hvt_pg.db lib_0.9V_hvt_pg.db lib_0.8V_hvt_pg.db"

# 3. Read UPF specification
```

```

load_upf mydesign.upf
# Ensure that all ports have a related supply net assigned as needed
set_related_supply_net -object_list [get_ports *] -power VDD -ground VSS

# 4. Specify supply groups and reference voltage levels
set vdd_top [get_supply_groups VDDTOP]
set vdd_block1 [get_supply_groups B1/VDD]
set vdd_block2 [get_supply_groups B2/VDD]
set_voltage_levels -reference_names {low high} $vdd_top
set_voltage_levels -reference_names {low high} $vdd_block1
set_voltage_levels -reference_names {low high} $vdd_block2

# 5. Specify voltage values for each reference voltage level
set_voltage -reference_name high -object_list $vdd_top 1.0
set_voltage -reference_name low -object_list $vdd_top 0.86
set_voltage -reference_name high -object_list $vdd_block1 1.0
set_voltage -reference_name low -object_list $vdd_block1 0.86
set_voltage -reference_name high -object_list $vdd_block2 1.0
set_voltage -reference_name low -object_list $vdd_block2 0.86

# Optional instance-specific set_voltage commands

# 6. Define DVFS scenarios and DVFS-specific design constraints

# -----
# Create DVFS scenarios
set vdd_top [get_supply_groups VDDTOP]
set vdd1 [get_supply_groups B1/VDD]
set vdd2 [get_supply_groups B2/VDD]
set vdd_mission [get_dvfs_scenarios \
    -supply_group $vdd_top -reference_names high]
set vdd_sleep [get_dvfs_scenarios \
    -supply_group $vdd_top -reference_names low]
set vdd1_low_vdd2_high [get_dvfs_scenarios \
    -supply_group $vdd1 -reference_names low
    -supply_group $vdd2 -reference_names high]
set vdd1_high_vdd2_low [get_dvfs_scenarios \
    -supply_group $vdd1 -reference_names high
    -supply_group $vdd2 -reference_names low]
# Define clocks that depend on the DVFS scenarios
create_clock -name CK1 -period 200 \
    -dvfs_scenarios $vdd_mission [get_pin pll/o]
create_clock -name CK1 -period 300 \
    -dvfs_scenarios $vdd_sleep [get_pin pll/o]

# Specify timing exceptions that depend on the DVFS scenarios
set_false_path -from CK2 -to CK1 \
    -dvfs_scenarios { $vdd1_high_vdd2_low $vdd1_low_vdd2_high}
set_multicycle_path 3 -from CK1 -to CK2
set_multicycle_path 4 -from CK1 -to CK2 \
    -dvfs_scenarios $vdd1_high_vdd2_low
set_multicycle_path 5 -from CK1 -to CK2 \

```

```

-dvfs_scenarios $vdd1_low_vdd2_high

# Additional timing checks for PG pins and supply nets
check_timing -override_defaults unconnected_pg_pins
check_timing -override_defaults supply_net_voltage

# Specify timing constraints (other than DVFS)
read_sdc mydesign.sdc
# ----

# 7. Run a "full" graph-based DVFS timing analysis
update_timing

# Report worst path across all possible DVFS scenarios
report_timing

# 8. Generate additional reports for specific voltage configurations
#     [Optional]
set vdd_main_low [get_dvfs_scenarios \
    -supply_group [get_supply_groups VDDTOP] -reference_names low]
report_timing -dvfs_scenarios $vdd_main_low

```

Commands and Attributes With DVFS Scenario Support

The following topics list the commands and attributes with DVFS scenario support:

- [Commands With DVFS Scenario Support](#)
- [Attributes With DVFS Scenario Support](#)

Commands With DVFS Scenario Support

The following commands support usage of the `-dvfs_scenarios` option in SMVA analysis to restrict the scope of a constraint or a report to specific DVFS scenarios:

```

create_clock
create_generated_clock
define_cell_alternative_lib_mapping
extract_model
get_timing_paths
read_sdc
remove_clock_gating_check
remove_clock_latency
remove_clock_uncertainty
remove_fanout_load
remove_input_delay
remove_max_capacitance
remove_max_fanout
remove_max_transition
remove_min_capacitance

```

```

remove_output_delay
remove_waveform_integrity_analysis
report_analysis_coverage
report_clock
report_clock_timing
report_clock_gating_check
report_constraint
report_crpr
report_delay_calculation
report_etm_arc
report_global_slack
report_global_timing
report_min_period
report_min_pulse_width
report_noise          ;# single DVFS scenario at a time
report_port
report_pulse_clock_max_transition
report_pulse_clock_max_width
report_pulse_clock_min_transition
report_pulse_clock_min_width
report_qor
report_si_bottleneck
report_timing
report_timing_derate
report_waveform_integrity_analysis
set_annotated_transition
set_clock_gating_check
set_clock_latency
set_clock_uncertainty
set_driving_cell      ;# only -input_transition_rise/fall
set_false_path
set_fanout_load
set_input_delay
set_load              ;# only -pin_load
set_max_capacitance
set_max_fanout
set_min_capacitance
set_output_delay
set_false_path
set_max_delay
set_max_transition
set_min_capacitance
set_min_delay
set_multicycle_path
set_output_delay
set_path_margin
set_timing_derate
set_waveform_integrity_analysis
source
update_noise          ;# single DVFS scenario at a time
write_interface_timing
write_sdf

```

Note the following:

- The `check_timing` and `report_exceptions` commands support SMVA by applying the checking or reporting to all the possible scenario and voltage combinations.
- The `update_noise` command has some limitations. See [Noise Analysis](#) for details.

Attributes With DVFS Scenario Support

The following attributes can be queried for their value in a particular DVFS scenario or set of scenarios. In the case of multiple scenarios, the worst-case value is returned.

Table 32 Attributes That Support Querying by DVFS Scenario

Attribute name	Object class
<code>actual_fall_transition_max</code>	pin, port
<code>actual_fall_transition_min</code>	pin, port
<code>actual_rise_transition_max</code>	pin, port
<code>actual_rise_transition_min</code>	pin, port
<code>actual_transition_max</code>	pin, port
<code>actual_transition_min</code>	pin, port
<code>cached_c1_max_fall</code>	pin, port
<code>cached_c1_max_rise</code>	pin, port
<code>cached_c1_min_fall</code>	pin, port
<code>cached_c1_min_rise</code>	pin, port
<code>cached_c2_max_fall</code>	pin, port
<code>cached_c2_max_rise</code>	pin, port
<code>cached_c2_min_fall</code>	pin, port
<code>cached_c2_min_rise</code>	pin, port
<code>cached_ceff_max_fall</code>	pin, port
<code>cached_ceff_max_rise</code>	pin, port
<code>cached_ceff_min_fall</code>	pin, port

**Table 32 Attributes That Support Querying by DVFS Scenario
 (Continued)**

Attribute name	Object class
cached_ceff_min_rise	pin, port
ceff_params_max	pin, port
ceff_params_min	pin, port
constraining_max_transition	pin, port
drc_actual_max_capacitance	pin, port
drc_actual_max_transition	pin, port
drc_constraining_max_capacitance	pin, port
drc_max_capacitance_slack	pin, port
drc_max_transition_slack	pin, port
driver_model_scaling_libs_max	pin, port
driver_model_scaling_libs_min	pin, port
driver_model_type_max_fall	pin, port
driver_model_type_max_rise	pin, port
driver_model_type_min_fall	pin, port
driver_model_type_min_rise	pin, port
effective_capacitance_max	pin, port
effective_capacitance_min	pin, port
exact_match_lib_cell_max_fall	cell, port
exact_match_lib_cell_max_rise	cell, port
exact_match_lib_cell_min_fall	cell, port
exact_match_lib_cell_min_rise	cell, port
fanout_load	lib_pin, pin, port
is_driver_scaled_max_fall	pin, port
is_driver_scaled_max_rise	pin, port

**Table 32 Attributes That Support Querying by DVFS Scenario
 (Continued)**

Attribute name	Object class
is_driver_scaled_min_fall	pin, port
is_driver_scaled_min_rise	pin, port
is_receiver_scaled_max_fall	pin, port
is_receiver_scaled_max_rise	pin, port
is_receiver_scaled_min_fall	pin, port
is_receiver_scaled_min_rise	pin, port
max_capacitance	lib_pin, pin, port
max_capacitance_clock_path_fall	clock
max_capacitance_clock_path_rise	clock
max_capacitance_data_path_fall	clock
max_capacitance_data_path_rise	clock
max_fanout	lib_pin, pin, port
max_transition	lib_pin, pin, port
max_transition_clock_path_fall	clock
max_transition_clock_path_rise	clock
max_transition_data_path_fall	clock
max_transition_data_path_rise	clock
min_capacitance	lib_pin, pin, port
min_fanout	lib_pin, pin, port
min_transition	lib_pin, pin, port
receiver_model_scaling_libs_max	pin, port
pin_capacitance_max	pin, port, net
pin_capacitance_max_fall	pin, port, net
pin_capacitance_max_rise	pin, port, net

Table 32 Attributes That Support Querying by DVFS Scenario (Continued)

Attribute name	Object class
pin_capacitance_min	pin, port, net
pin_capacitance_min_fall	pin, port, net
pin_capacitance_min_rise	pin, port, net
receiver_model_scaling_libs_min	pin, port
power_rail_voltage_max	pin, port
power_rail_voltage_min	pin, port
receiver_model_type_max_fall	pin, port
receiver_model_type_max_rise	pin, port
receiver_model_type_min_fall	pin, port
receiver_model_type_min_rise	pin, port
voltage_for_max_delay	pg_pin_info
voltage_for_min_delay	pg_pin_info

Feature Compatibility

SMVA analysis is not compatible with the following features:

- Noise analysis on multiple DVFS scenarios at once
- Extracted timing model (ETM) generation of a single SMVA model
- Reduced-resource ECO
- Different SDF annotated delay data per DVFS scenario
- Phase-locked loop generated clocks (`create_generated_clock -pll_feedback and -pll_output`)
- Certain types of advanced attribute usage

16

Signal Integrity Analysis

PrimeTime SI is an option 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. 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.

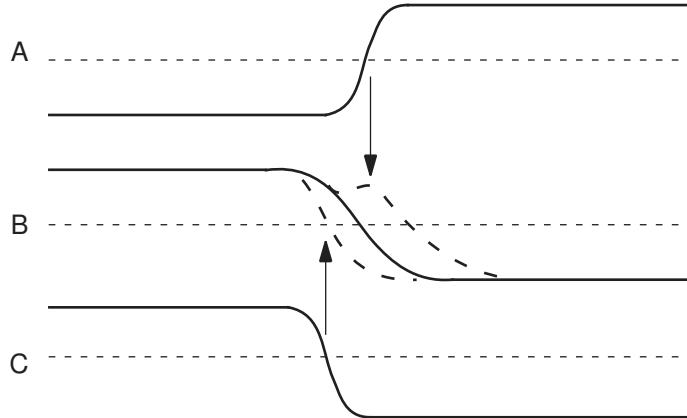
As circuit geometries become smaller, wire interconnections become closer together and taller, thus increasing the cross-coupling capacitance between nets. At the same time, parasitic capacitance to the substrate becomes less as interconnections become narrower, and cell delays are reduced as transistors become smaller.

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 SI 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 148](#) shows the signal waveforms on cross-coupled nets A, B, and C.

Figure 148 Transition slowdown or speedup caused by crosstalk



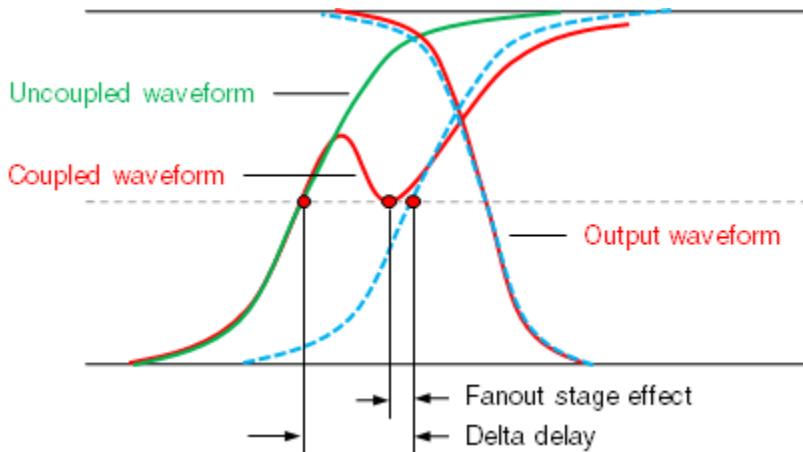
Because of capacitive cross-coupling, the transitions on net A and net C can affect the time at which the transition occurs on net B. A rising-edge transition on net A at the time shown in [Figure 148](#) can cause the transition to occur later on net B, possibly contributing to a setup violation for 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 149](#).

Figure 149 Delta delay includes the fanout stage effect

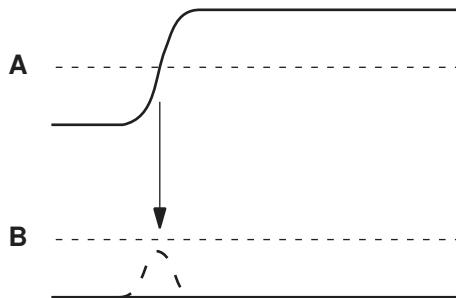


PrimeTime SI calculates the *delta delay*, which is the amount of additional delay induced by crosstalk on a switching net. The *fanout stage effect* represents the delay propagated to the fanout stages. PrimeTime SI includes this fanout stage effect as part of the delta delay.

Crosstalk Noise Effects

PrimeTime SI also determines the logic effects of crosstalk noise on steady-state nets. Figure 150 shows an example of a noise bump due to crosstalk on cross-coupled nets A and B.

Figure 150 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.

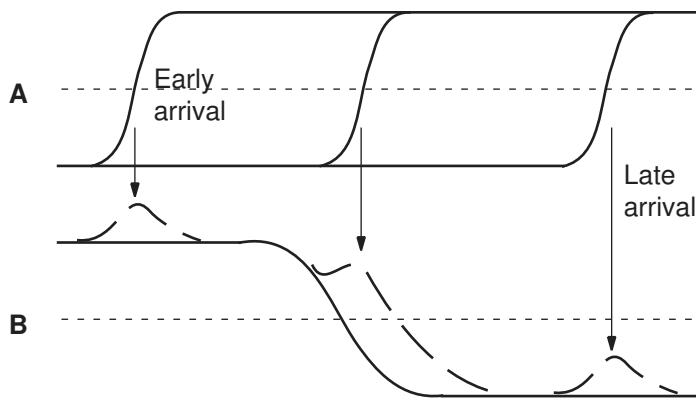
The timing effect of an aggressor net on a victim net depends on several factors:

- 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 151](#) 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 151 Effects of crosstalk at different arrival times



As shown in [Figure 151](#), 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.

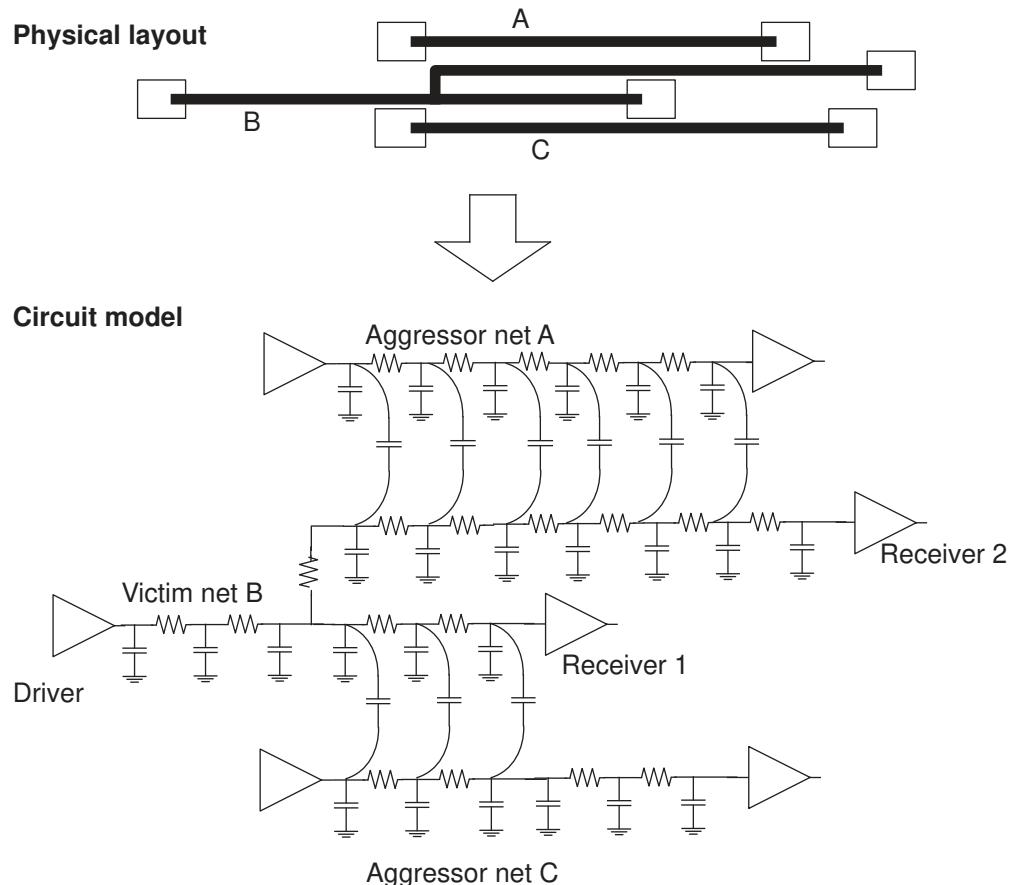
By default, PrimeTime SI performs crosstalk analysis using two iterations. During the first iteration, it ignores the timing windows and assumes that all transitions can occur at any time. This results in pessimistic crosstalk delay values. During the second and subsequent iterations, PrimeTime SI considers 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 152 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 152 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 SPEF or GPD file:

```
pt_shell> read_parasitics -keep_capacitive_coupling file_name.spf
```

or

```
pt_shell> read_parasitics -keep_capacitive_coupling \  
-format GPD file_name.gpd
```

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_verilog ./test1.v
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
```

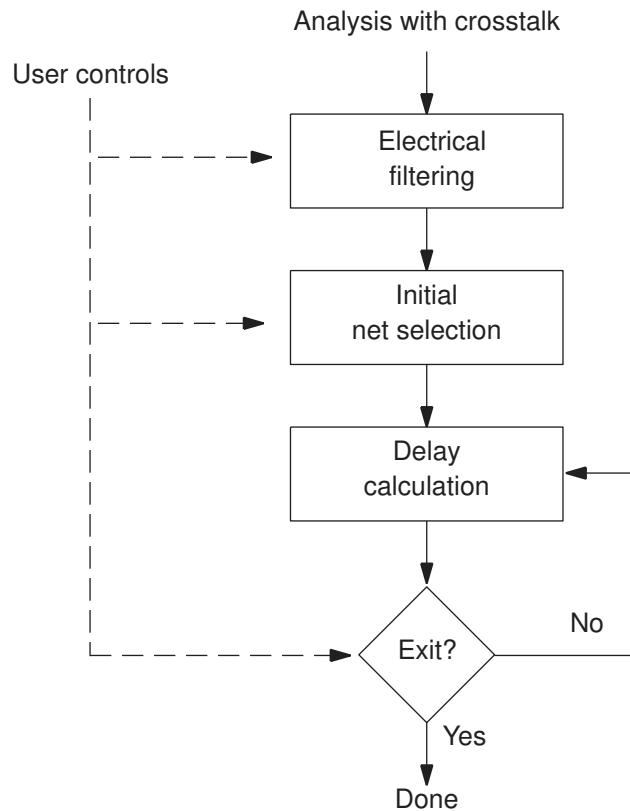
The `set_operating_conditions` command sets the analysis type to `on_chip_variation`, which is necessary to allow PrimeTime SI to correctly handle the min/max timing window relationships. If you do not specify this analysis type explicitly, PrimeTime SI automatically switches to that mode.

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 153](#).

Figure 153 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 33](#) lists the variables that control crosstalk analysis. To enable crosstalk analysis, set the `si_enable_analysis` variable to `true`.

Table 33 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_filter_accum_aggr_noise_peak_ratio</code>	0.03
<code>si_filter_per_aggr_noise_peak_ratio</code>	0.01
<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>

Table 33 PrimeTime SI Variables (Continued)

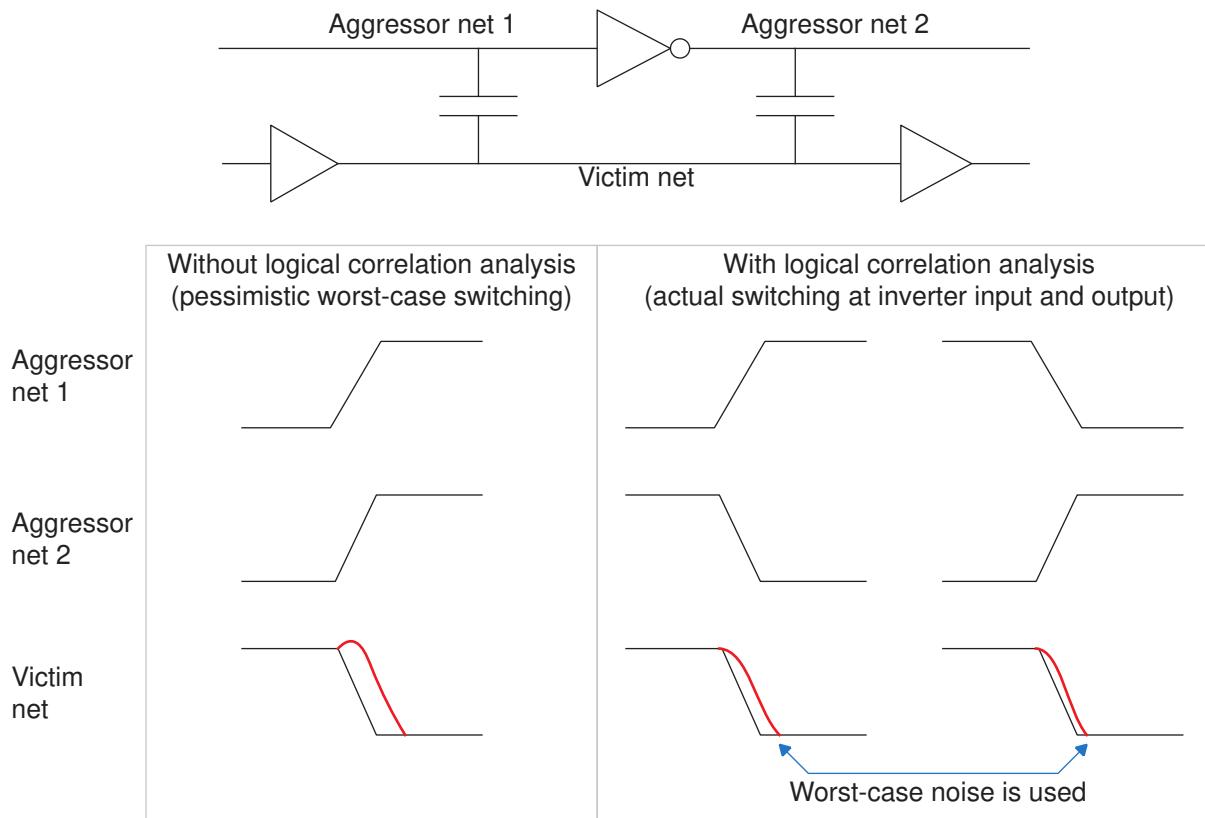
Variable	Default setting
si_xtalk_composite_aggr_noise_peak_ratio	0.01
si_xtalk_composite_aggr_quantile_high_pct	99.73
si_xtalk_delay_analysis_mode	all_paths
si_xtalk_double_switching_mode	disabled
si_xtalk_exit_on_max_iteration_count	2
si_xtalk_max_transition_mode	uncoupled

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 154](#) 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 154 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.

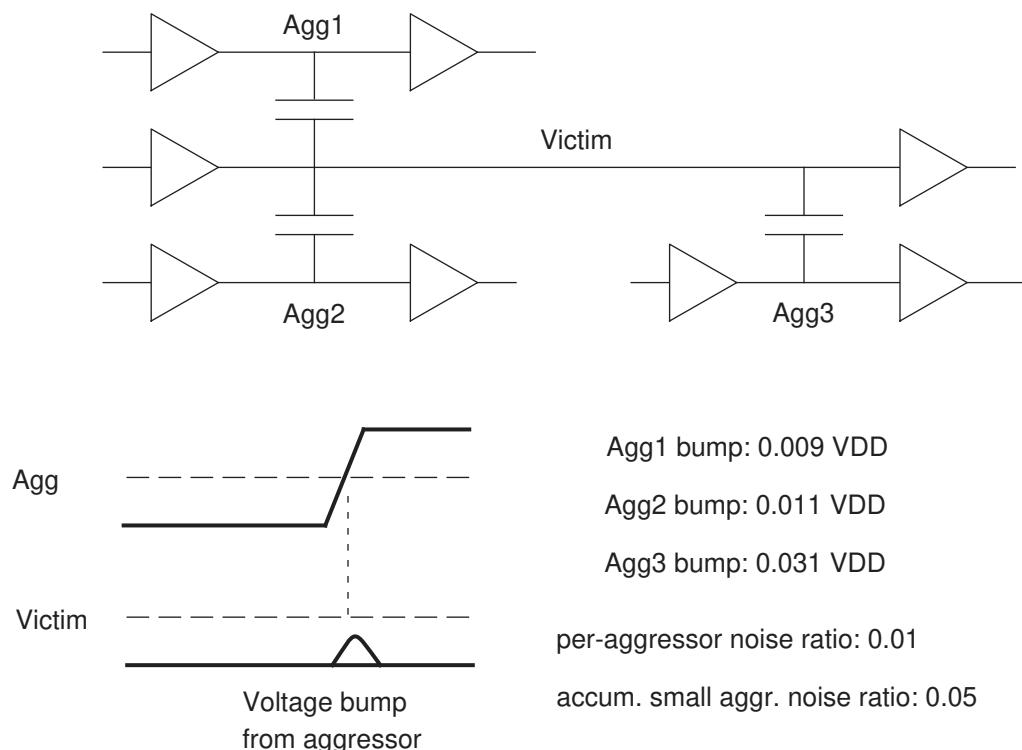
Filtering eliminates aggressors based on the size of the voltage bump induced on the victim net by the aggressor net. The bump sizes depend on the cross-coupling capacitance values, drive strengths, and resistance values in the nets. An aggressor net is filtered if the peak voltage of the noise bump 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 155 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 155 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.

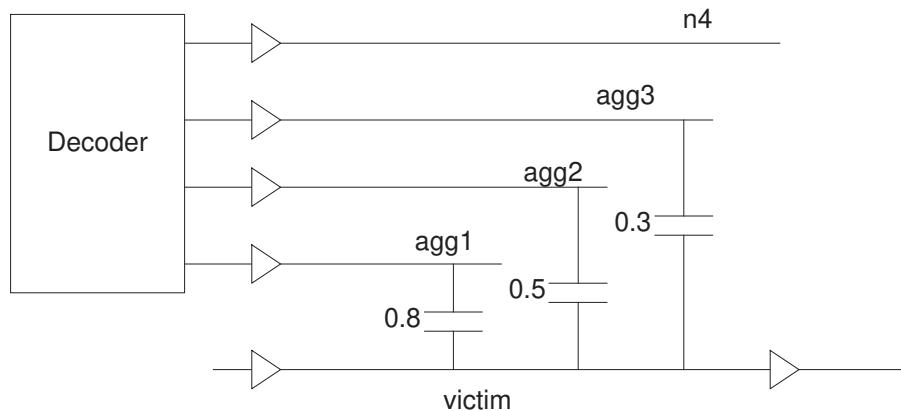
Excluding Aggressor-to-Aggressor Nets

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 156](#) 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 156 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 157](#). Table 34 describes how the aggressors are considered during crosstalk analysis.

Figure 157 Multiple aggressor arrival windows at decoder outputs

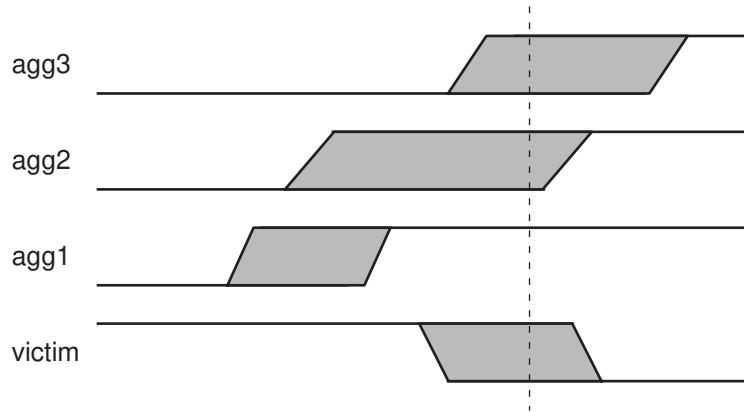


Table 34 Crosstalk analysis with and without exclusive group

	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 specify neither 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 35](#).

Table 35 Commands 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>

Table 35 Commands for removing exclusions (Continued)

To remove exclusions set by...	Use this command
set_coupling_separation	remove_coupling_separation

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.

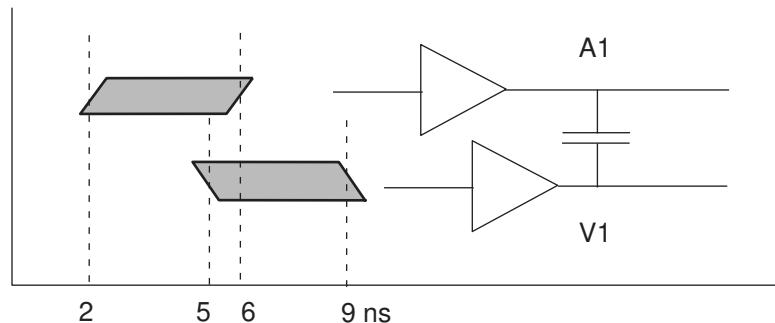
Timing Window Overlap Analysis

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 158](#) shows how timing windows can overlap.

Figure 158 Victim-aggressor switching time alignment



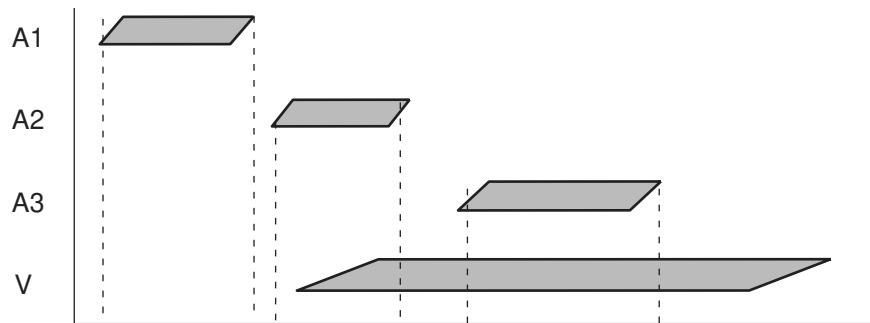
In this example, victim net V1 is coupled to aggressor net A1. The timing arrival windows are 2 ns to 6 ns for V1, and 5 ns to 9 ns for A1. Since the timing window of V1 overlaps with the timing window of A1, PrimeTime SI calculates the crosstalk delta delay of V1 due to A1.

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 159](#) shows a victim net V with three aggressors, A1, A2, and A3.

Figure 159 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:

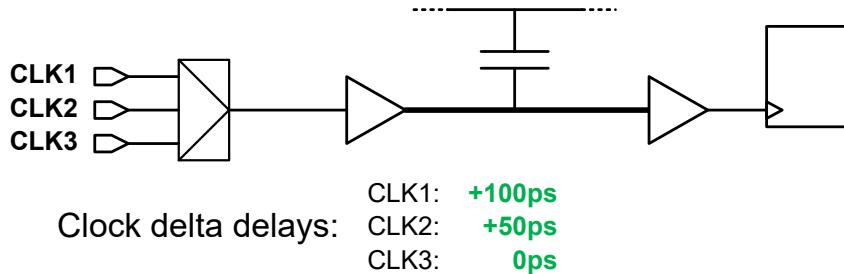
N – aggressor does not overlap for the worst case alignment

Victim Nets in Multiple Clock Domains

When a victim net has arrival windows from multiple clock domains, PrimeTime SI must determine how to compute and store their delta delays.

Clock Network Victim Nets

For clock network delay calculation, PrimeTime SI always computes a separate delta delay for each victim net clock domain:



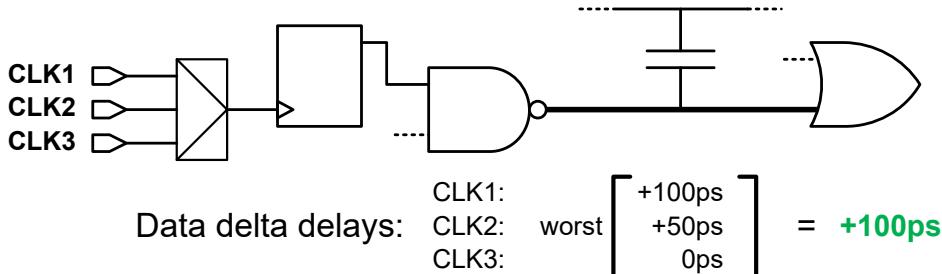
This is because clock networks typically have narrow, tightly controlled arrival windows. A clock net might have a delta delay in one clock domain but not another. In addition, crosstalk effects on a clock net affect the timing of all downstream launching and capturing sequential cells.

Data Network Victim Nets

For data network delay calculation (including clocks used as data), the behavior is configurable by setting the following variable:

```
pt_shell> set_app_var \
    timing_enable_independent_crosstalk_in_data_network \
    none | all_clocks | physically_exclusive_clocks
```

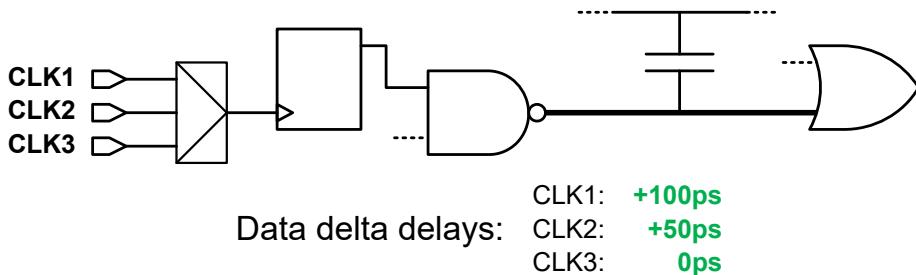
With the default setting of `none`, PrimeTime SI computes the worst delta delay across all victim net clock domains:



This is the default because data networks tend to have wider arrival windows due to their reconvergent nature, and thus tend to have less delta delay variation across clock domains. It is fast, efficient, and suitable for most analysis.

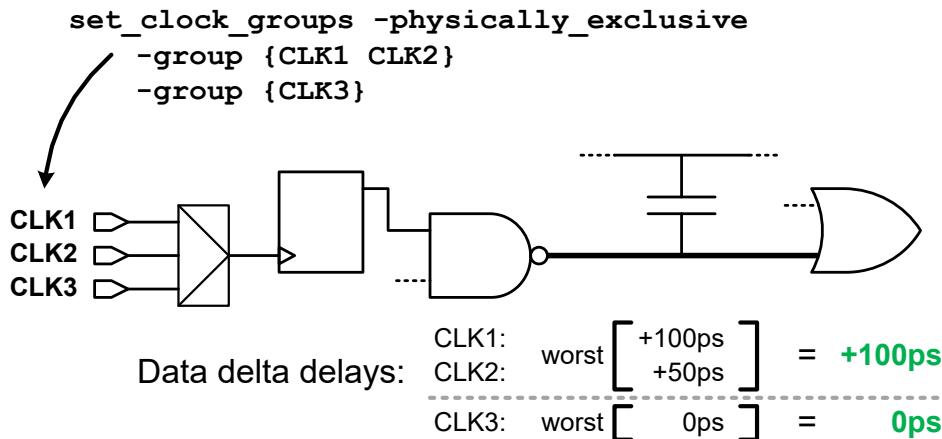
However, pessimism in data network delta delays can still occur, particularly when analyzing multiple operating modes with significantly different clock characteristics in the same analysis run.

When the variable is set to `all_clocks`, the tool computes a separate delta delay for each victim net clock domain (just as with clock nets):



Note that this setting can result in increased memory and runtime for multimode designs with many overlapping clocks.

When the variable is set to `physically_exclusive_clocks`, the tool computes a separate delta delay for each group of victim net clock domains that can physically coexist at the same time:



For multimode analysis that uses physically exclusive clock groups, this corresponds to merging delta delays within each mode, but keeping delta delays separate across modes. This setting can still provide some pessimism reduction, but with less memory and runtime than the `all_clocks` setting. For details on physically exclusive clock groups, see [Logically and Physically Exclusive Clocks](#).

The `physically_exclusive_clocks` mode considers only clock groups explicitly defined by the `set_clock_groups -physically_exclusive` command. It does not consider physically exclusive clocks found by the `set_clock_exclusivity` command.

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

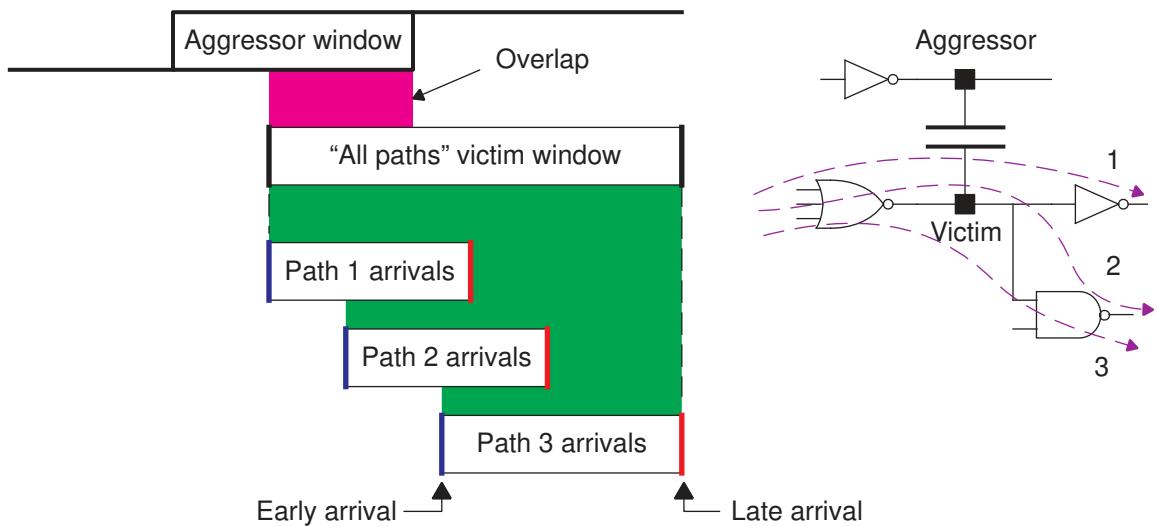
For multiple aggressors, if the aggressor clocks are synchronous with each other, but asynchronous with the victim, the timing relationships between the aggressors are respected, but they are still treated as infinite windows with respect to the victim.

“All Paths” Window Versus “All Path Edges” Overlap Modes

By default, the tool considers the full range of possible arrival times for all paths through the victim net, from the earliest minimum-delay path to the latest maximum-delay path. Any overlap between an aggressor arrival window and the “all paths” victim window triggers a delta delay calculation.

The “all paths” window overlap mode is simple and fast, but it is also conservative because the actual victim transitions might not overlap the aggressor window. For example, suppose that you have three different paths through the victim net, each with an arrival range shown in the following diagram. The tool merges the path-specific victim arrival ranges into a single all-paths window, triggering a delta delay calculation for all the early and late arrival edges.

Figure 160 “All Paths” (Default) Window Overlap Mode



In this example, for a *maximum-delay* calculation, the *late* arrival edges (red lines) do not actually overlap the aggressor window, so there is no real possibility for them to be delayed further by the aggressor.

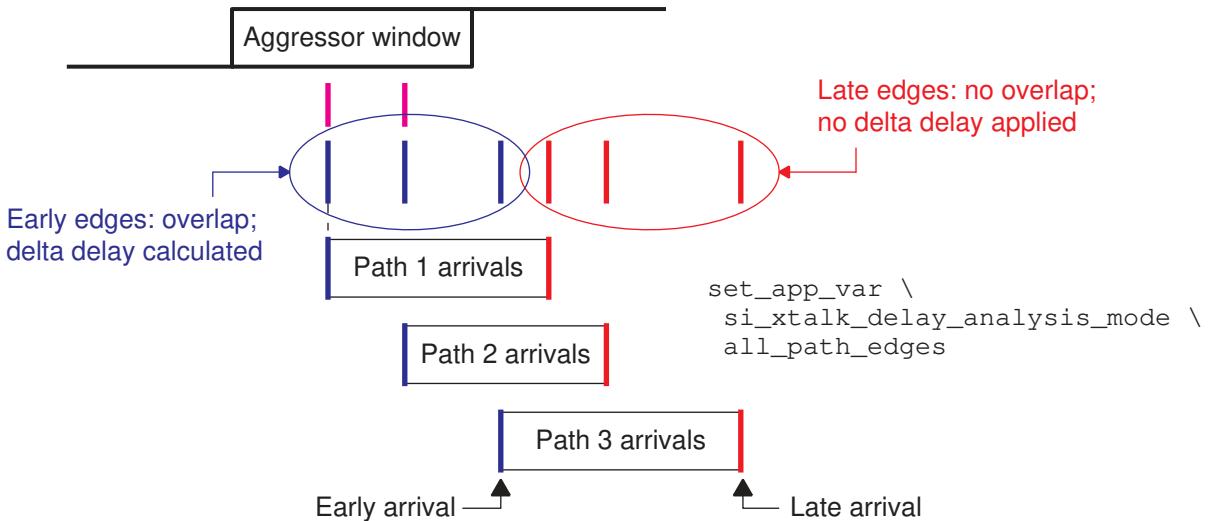
For critical-path analysis, you can remove this pessimism by either of the following methods:

- Perform path-based analysis, which considers each path in isolation from the others:
`report_timing -pba_mode path or exhaustive ...`
- Set the “all path edges” overlap analysis mode:
`set_app_var si_xtalk_delay_analysis_mode all_path_edges`

The “all path edges” mode considers the actual early or late arrival edge times, without combining them into a continuous window. In the following example, for maximum-delay analysis, the “all path edges” mode does not apply delta delays because there is no overlap of the late edges with the aggressor window. For minimum-delay analysis, it

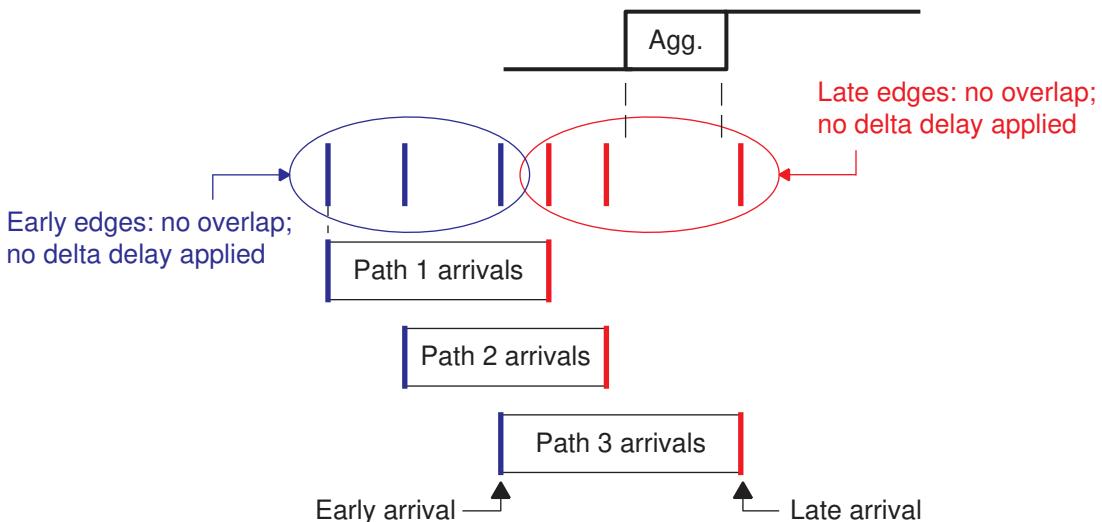
applies delta delays for all the early arrival edges because there is at least one early-edge overlap.

Figure 161 “All Path Edges” Overlap Mode



The “all path edges” method even excludes aggressor windows that fall in between edges, as shown in the following example.

Figure 162 “All Path Edges” Overlap Mode, Aggressor Window Between Edges



In summary, to control crosstalk window alignment speed and accuracy, use

- “All paths” (default) mode for a fast, conservative analysis
- “All path edges” mode for reduced pessimism at some runtime cost:

```
set_app_var si_xtalk_delay_analysis_mode all_path_edges
```

- Path-based analysis for maximum accuracy at the largest runtime cost:

```
report_timing -pba_mode path or exhaustive ...
```

For more information, see [SolvNetPlus article 000006197, “How does the PrimeTime SI all_path_edges Alignment Mode Work?”](#)

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.

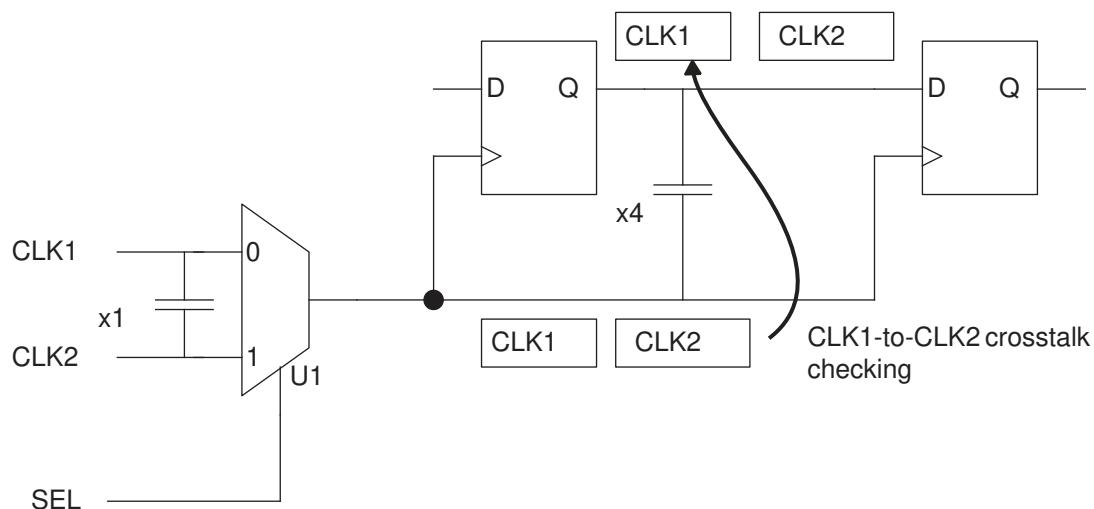
The `-allow_paths` option can be used with asynchronous clocks to restore analysis of the timing paths between clock groups, but still using infinite alignment windows for crosstalk analysis.

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 163](#). 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 163 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:

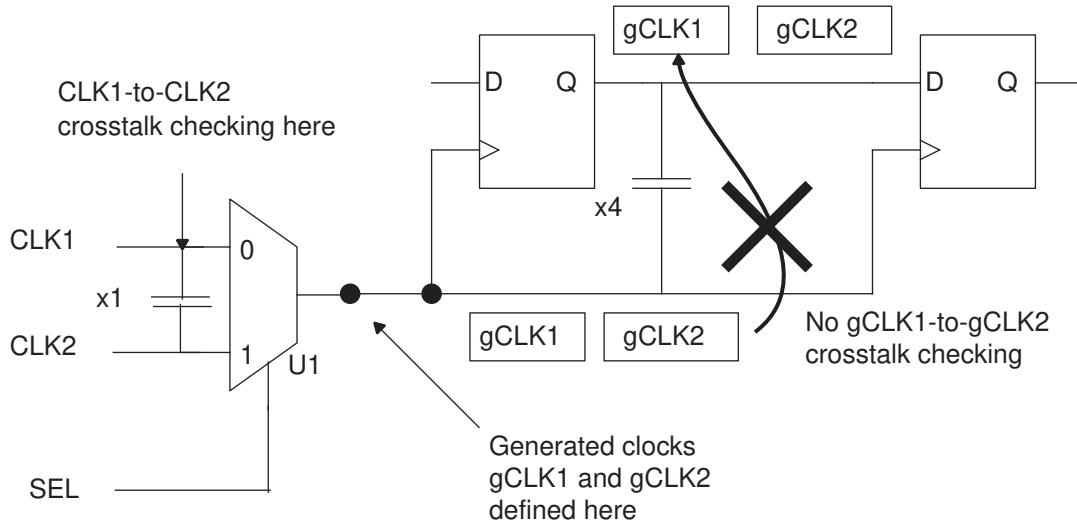
```
pt_shell> create_generated_clock -name gCLK1 \
    -source [get_ports CLK1] -divide_by 1 \
    -add -master_clock [get_clocks CLK1] \
    [get_pins U1/z]

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 164](#).

Figure 164 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.

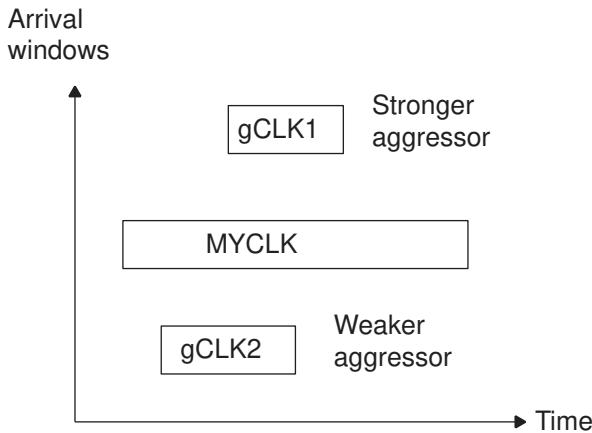
PrimeTime detects only group-to-group conflicts, not implied conflicts. For example, if you declare CLK1 and CLK2 to be asynchronous to each other, they are both still synchronous to CLK3 by default. This is an implied three-way conflict that PrimeTime does not detect. You are responsible for resolving 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 165.

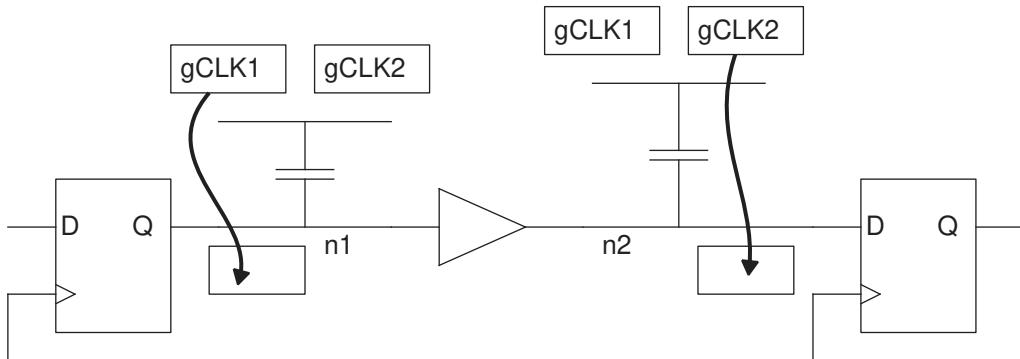
Figure 165 Crosstalk alignment diagram



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 166. 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 166 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.

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 reduces 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 167](#) and [Figure 168](#).

Figure 167 Composite aggressor mode disabled

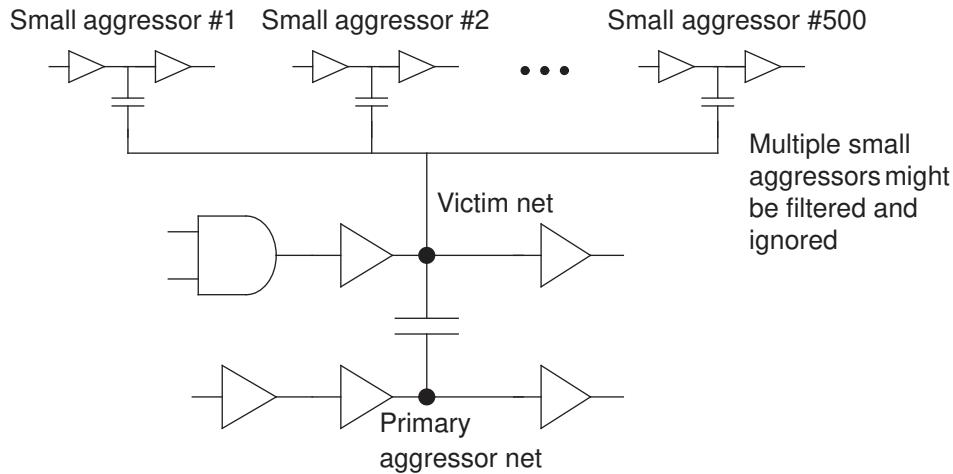
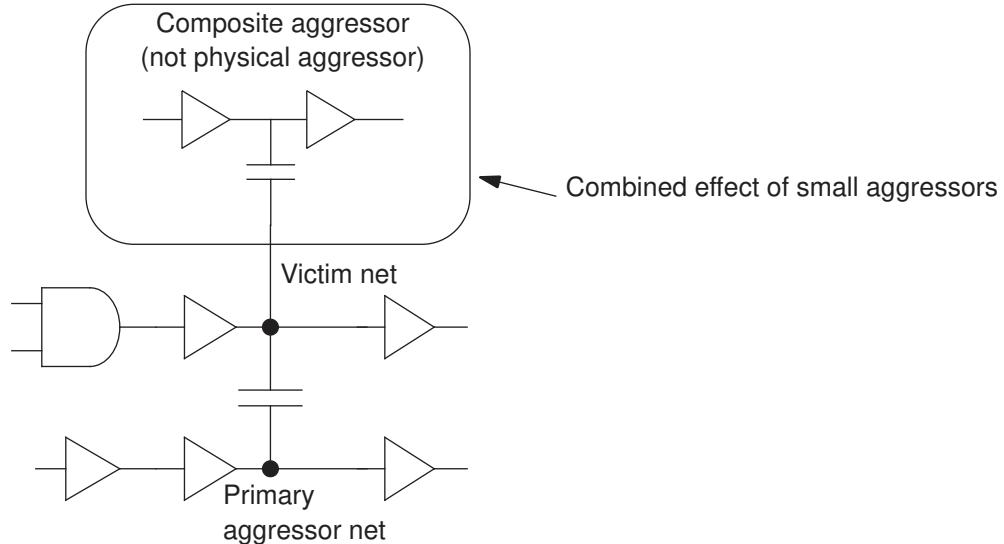


Figure 168 Composite aggressor mode enabled



In [Figure 167](#), the circuit has 500 small aggressors. Composite aggressor mode replaces these aggressors with a single composite aggressor waveform, as shown in [Figure 168](#).

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 the 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 required 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:

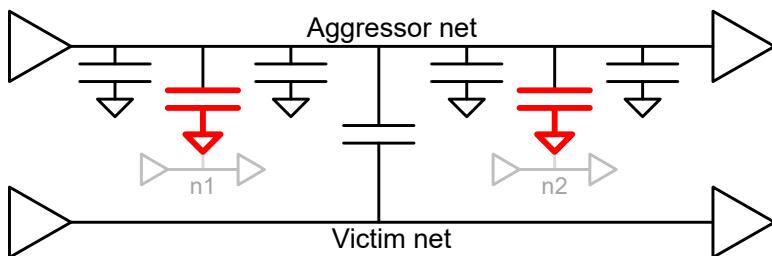
- `si_xtalk_composite_aggr_min_rise`
- `si_xtalk_composite_aggr_min_fall`
- `si_xtalk_composite_aggr_max_rise`
- `si_xtalk_composite_aggr_max_fall`

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"}
```

Ideal Aggressors

When an aggressor net couples to multiple nets, only aggressor coupling capacitors to the *current* victim net are considered. By default, aggressor coupling capacitors to other nets are converted to ground capacitors:

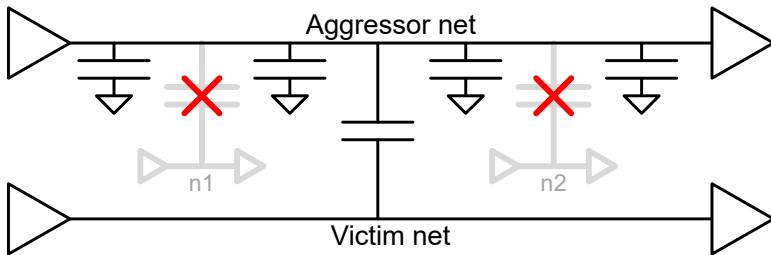


However, some flows have special “placeholder” nets that can couple to many other nets. With the default behavior, large numbers of inactive-coupling ground capacitors can reduce an aggressor net’s effective strength during the calculation.

To remedy this, you can use the `set_ideal_aggressor` command to apply an `si_ideal_aggressor` attribute to these placeholder nets:

```
pt_shell> set_ideal_aggressor $nets
```

When the `si_ideal_aggressor` attribute exists on an aggressor net, its coupling capacitors to other nets (besides the current victim net) are omitted entirely, thus strengthening it against the current victim net:



The `si_ideal_aggressor` attribute affects aggressor net behaviors for both crosstalk delay and noise. It does not affect the following:

- Aggressor transition times
- Aggressor timing windows
- Coupling capacitances of victim nets

The `report_delay_calculation` command indicates ideal aggressors with a 'P' annotation:

```
pt_shell> report_delay_calculation ...
...
```

Aggressor Net	Coupling Cap	Driver Lib Cell	Clocks	Attributes	Switching Bump (ratio of VDD)
n232	0.200000	BF1T4_D	CLK1	P	-

Ideal aggressors can be found and filtered by their `si_ideal_aggressor` attribute:

```
pt_shell> get_nets ... -filter {si_ideal_aggressor == true}
```

To remove the `si_ideal_aggressor` attribute, use the `remove_ideal_aggressor` command:

```
pt_shell> remove_ideal_aggressor $nets
```

```
pt_shell> remove_ideal_aggressor -all
```

For more information, see the man pages.

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.

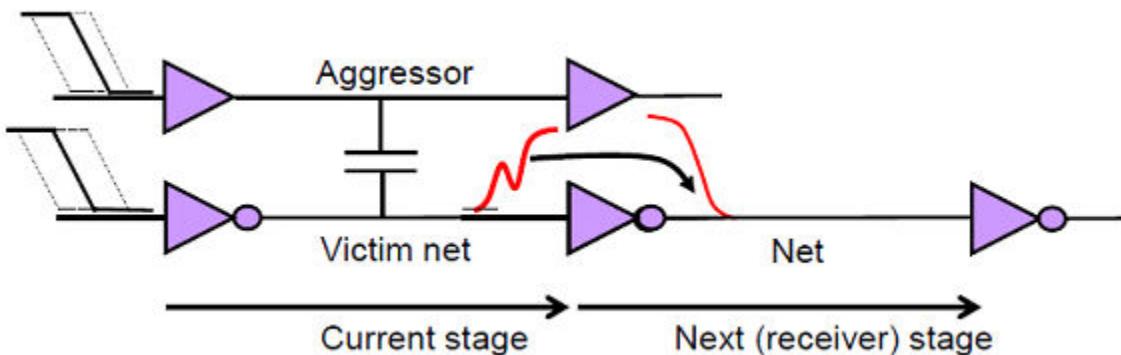
For more information, see [Path-Based Timing Analysis](#).

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.

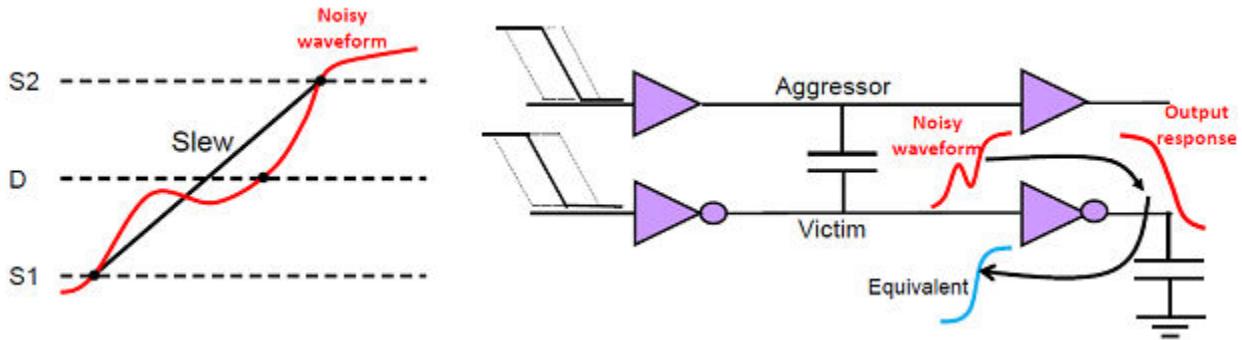
Crosstalk causes distortions in the switching waveforms and affects the delay of the victim stage and its fanouts. The circuit example in [Figure 169](#) has a victim net with a single aggressor. Because of cross-coupling between the aggressor net and the victim net, the switching waveform at the input 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 169 Crosstalk effect modeled as delta delay for current stage and fanout



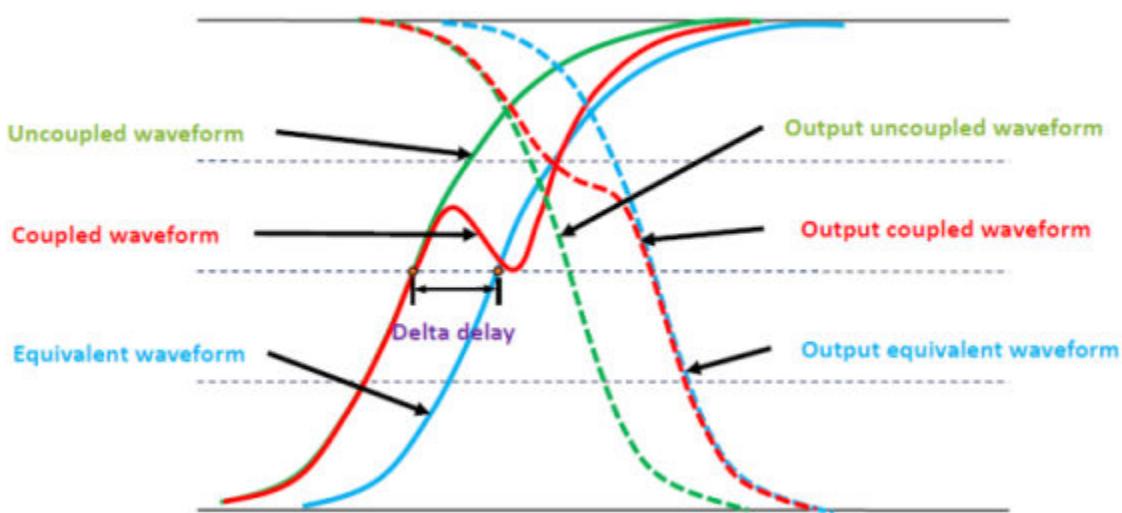
PrimeTime SI uses an equivalent waveform approach to model the distorted coupled waveform. PrimeTime delay calculation is stage-based, and the transition time calculated for the current stage is propagated to the next stage for delay calculation. In presence of severely distorted waveform as shown [Figure 170](#), using the traditional approach of measuring the slew at slew trip points (S1, S2) would be pessimistic.

Figure 170 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 171](#). This ensures a conservative analysis irrespective of the distorted coupled waveform's transition time.

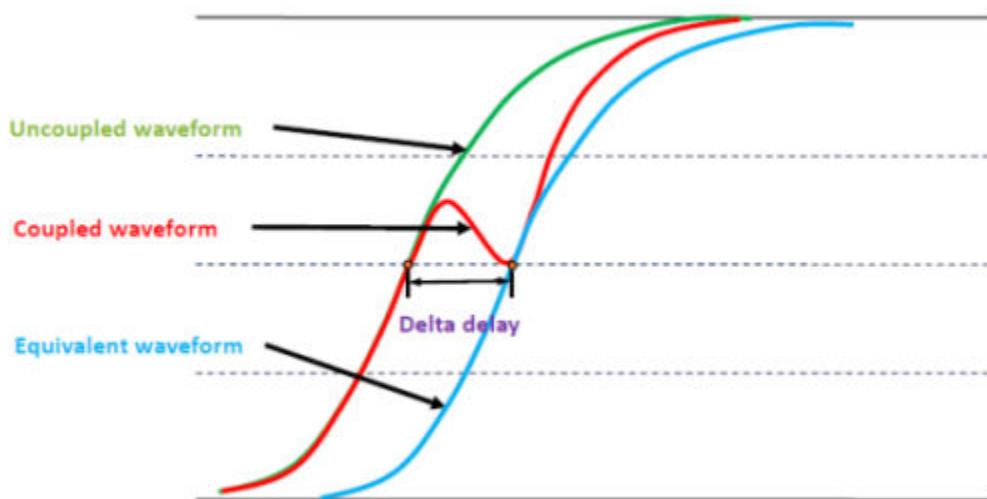
Figure 171 Signal integrity effect with CCS noise library or forward timing arc



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 172](#). This ensures a conservative analysis.

Figure 172 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

To improve accuracy in the presence of waveform distortions, enable waveform propagation, set the `delay_calc_waveform_analysis_mode` variable to `full_design`.

For more information, see [Waveform Propagation](#).

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 27](#) shows a timing report with crosstalk effects.

Example 27 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_regex28x
    (recovery check against rising-edge clock clock)
Path Group: **async_default**
Path Type: max
```

Chapter 16: Signal Integrity Analysis

Crosstalk Delay Analysis

Point	DTrans	Trans	Delta	Incr	Path
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	0.0066	0.0028	&	0.0028 f
U26/Z (BF1T2)	1.0368		0.7040	&	0.7068 f
hostif_0/reset (hostif_mstest_1)	0.0000		0.0000		0.7068 f
hostif_0/U1836/A (IV)	-0.0023	1.0414	0.0087	0.0388 &	0.7456 f
hostif_0/U1836/Z (IV)	0.7593		0.5008	&	1.2463 r
hostif_0/U1835/A (BF1T4)	0.0000	0.7593	0.0000	0.0002 &	1.2466 r
hostif_0/U1835/Z (BF1T4)	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 arrival time					-2.0960
slack (MET)					2.6571

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
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 minimum-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_aggressor` 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 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 needed, 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}
```

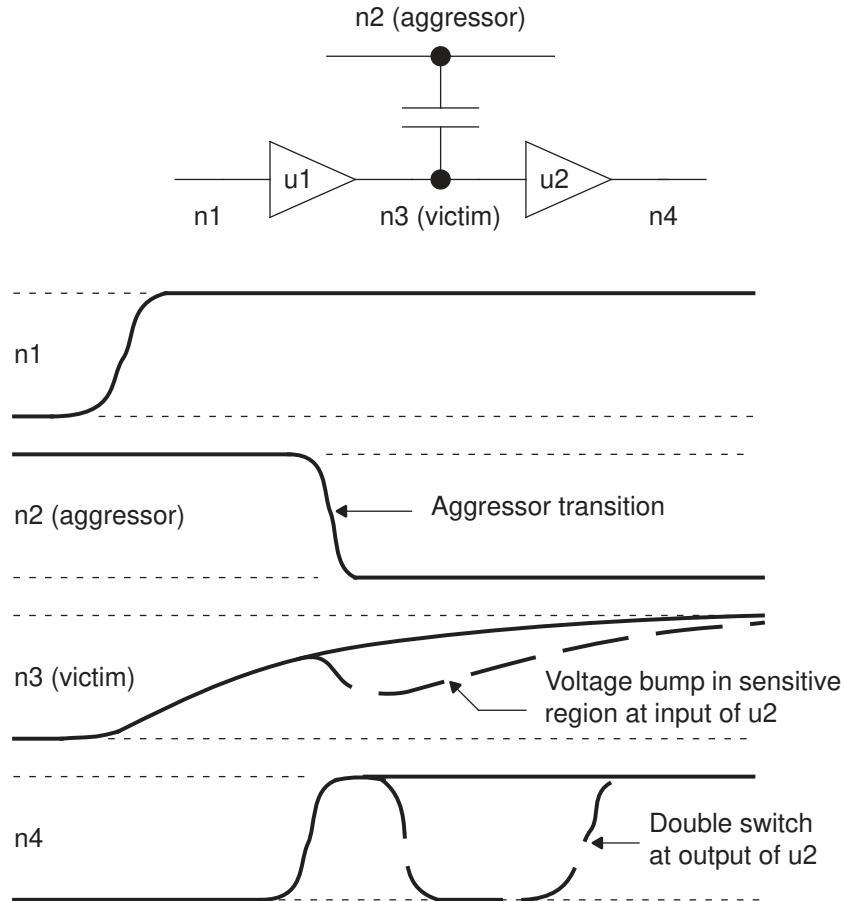
The `report_si_aggressor_exclusion` command reports the crosstalk settings made with the `set_si_aggressor_exclusion` command. Net-specific reporting for the `report_si_aggressor_exclusion` command reports all exclusive groups to which the specific net belongs. To find out which of the aggressors were 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 173](#).

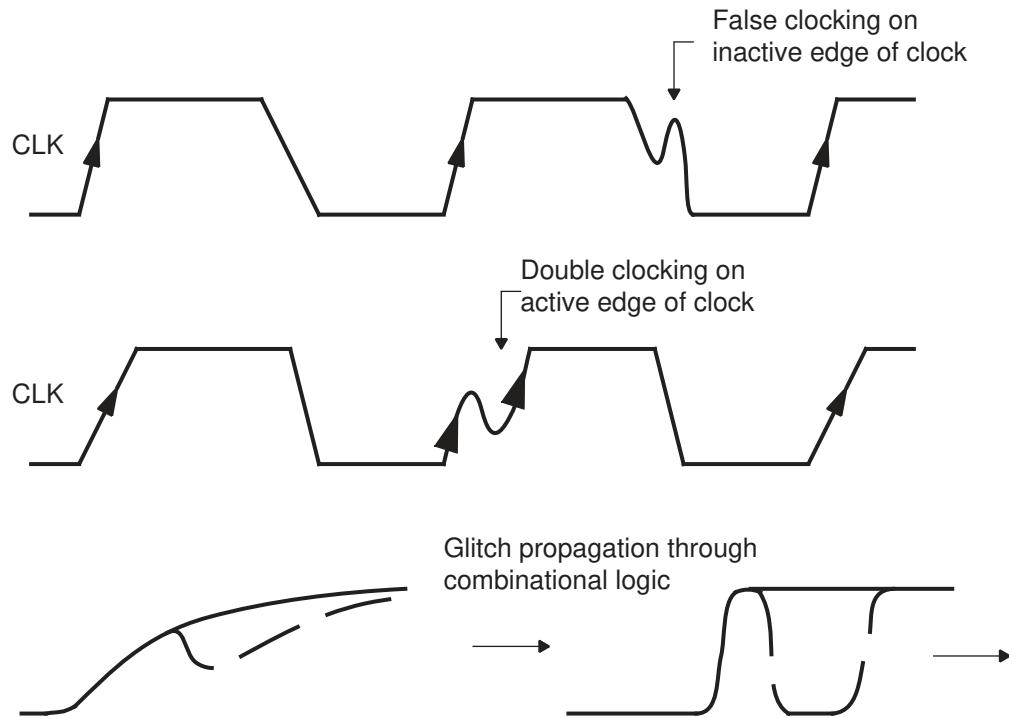
Figure 173 Double-switching error example



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 173](#), 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 174](#). The false clocking and double clocking examples cause incorrect clocking of rising-edge-sensitive registers.

Figure 174 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.

To fix timing violations, use the `fix_eco_timing` command. For more information, see [Timing Violation Fixing](#).

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](#)
-

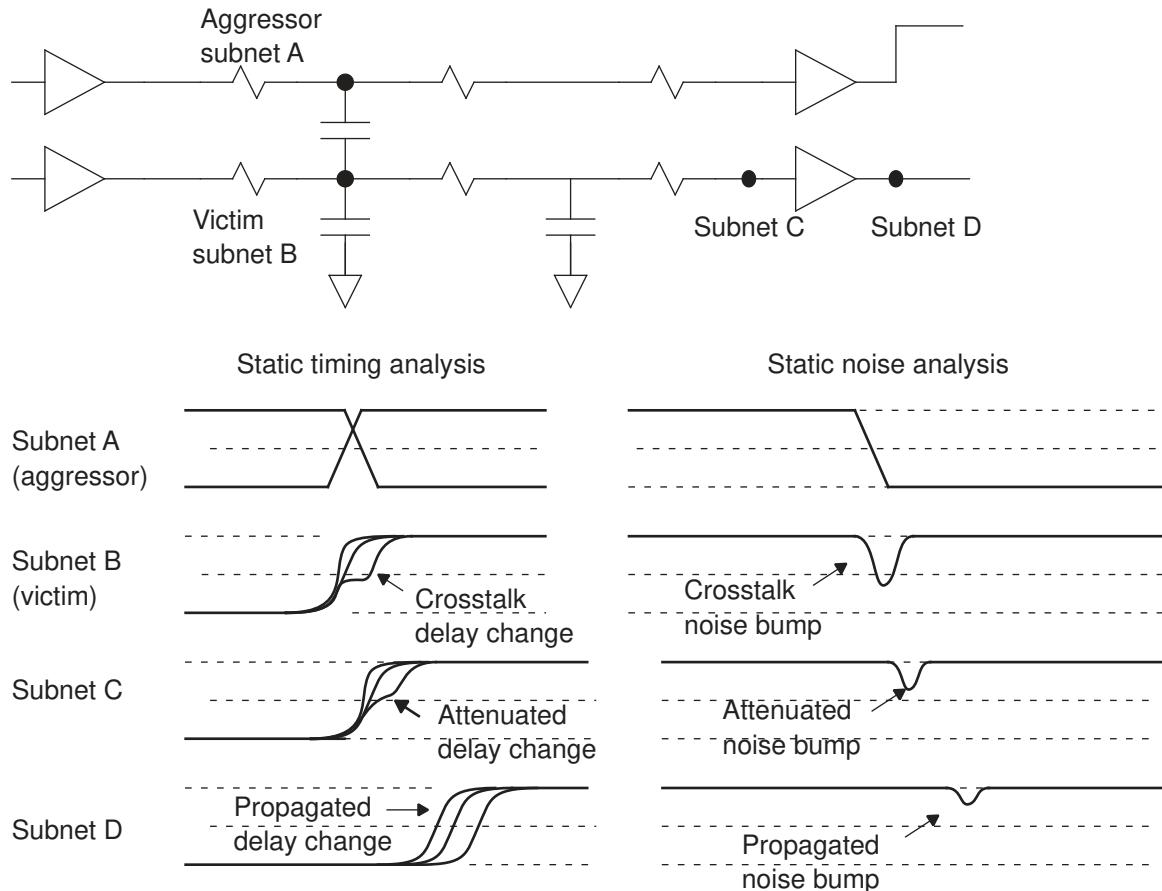
Static Noise Analysis Overview

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 175](#) 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.

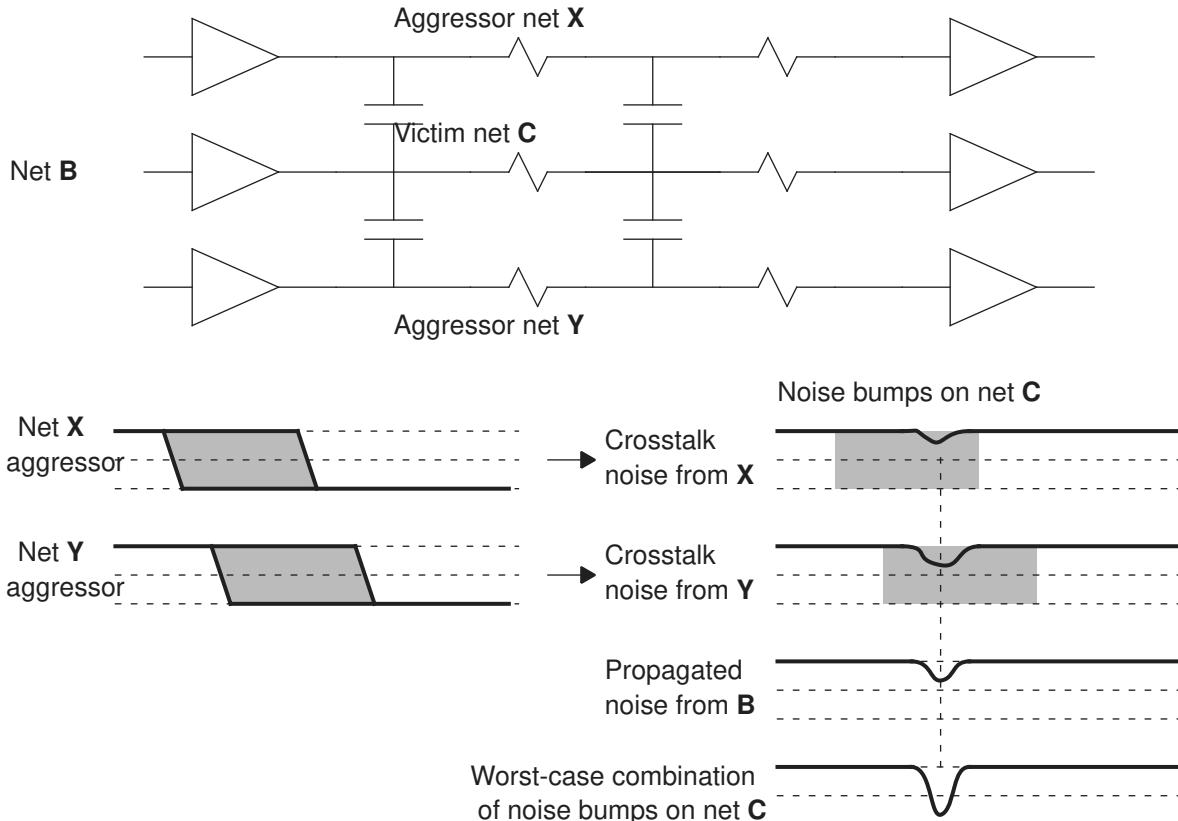
Figure 175 Crosstalk effects on timing and steady-state voltage



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 176 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 176 Combined effects of crosstalk and propagated noise



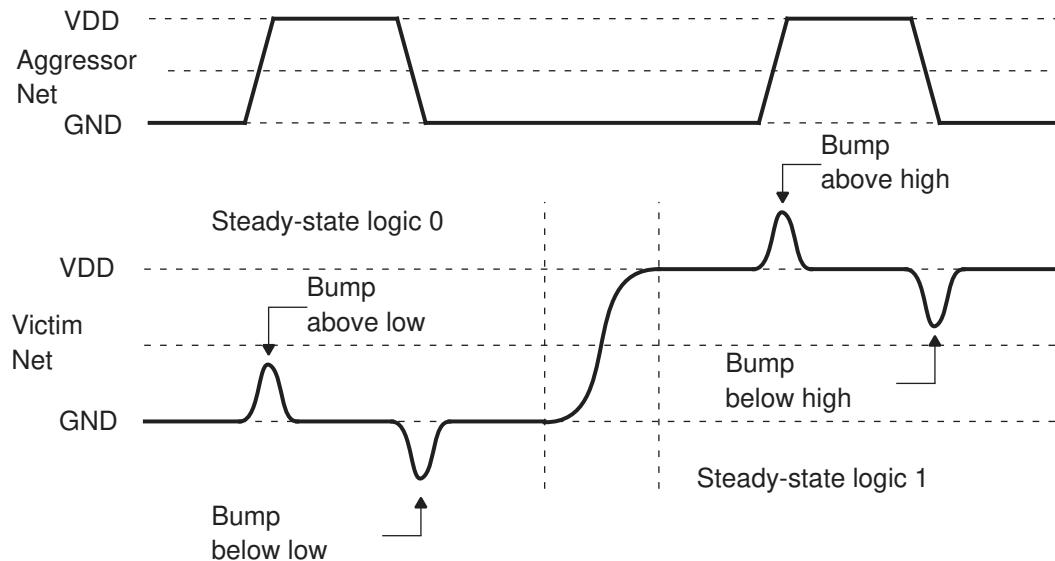
Noise Bump Characteristics

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 177](#).

Figure 177 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.

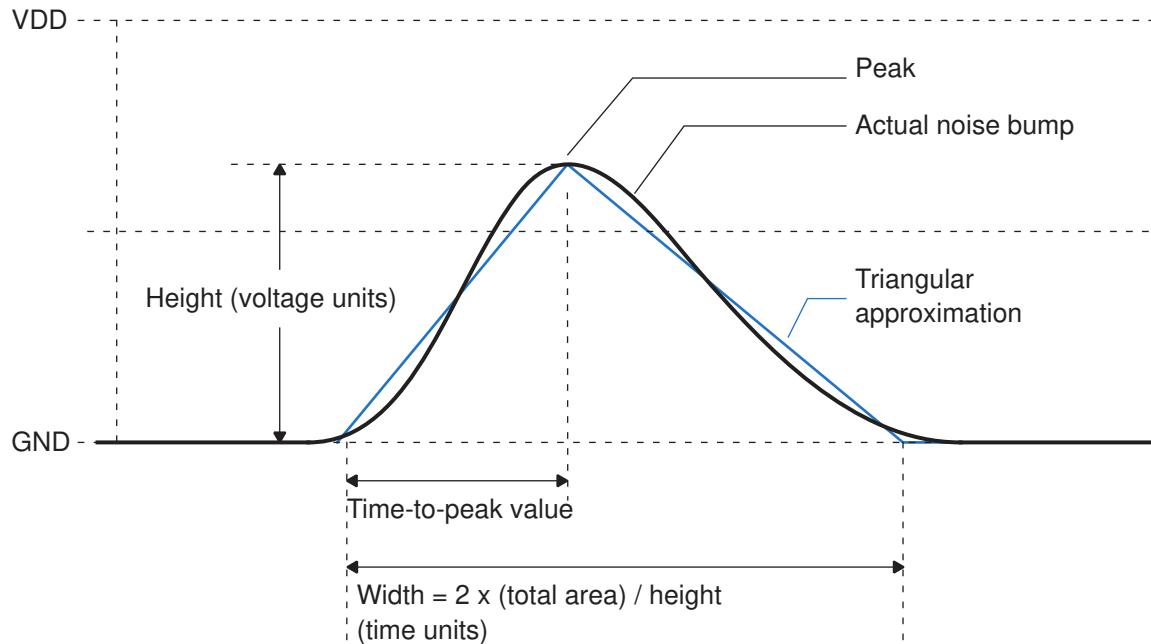
The following numbers specify the characteristics of a noise bump:

- 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 178](#), 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 178 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

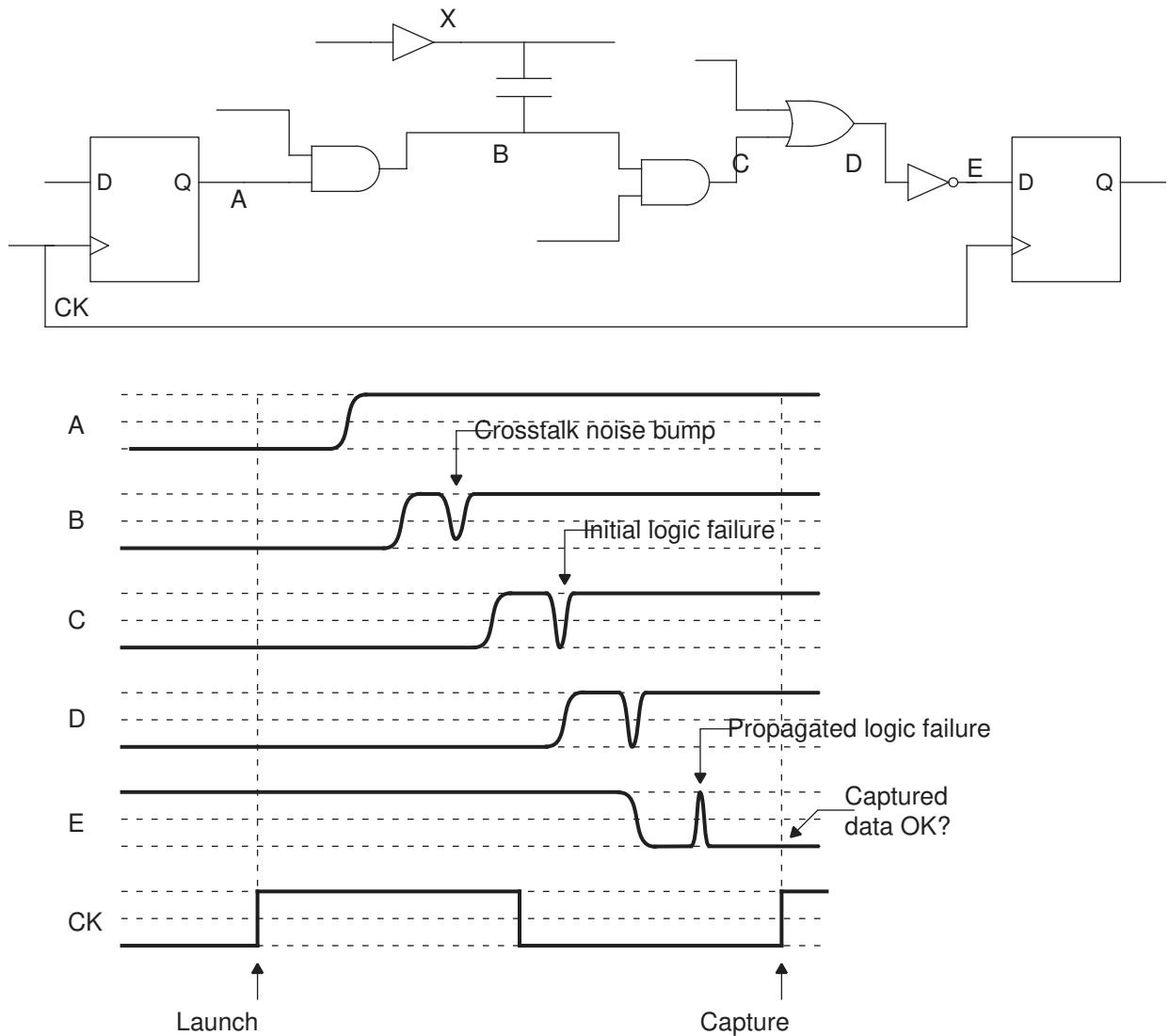
In some cases, propagated noise exists but cannot be calculated. For example if the cell is an extracted timing model or a black-box model, you can specify a noise bump explicitly on any pin or port by using the `set_input_noise` command. This is called *user-defined noise*. User-defined noise can either 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 179](#). 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 179 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.

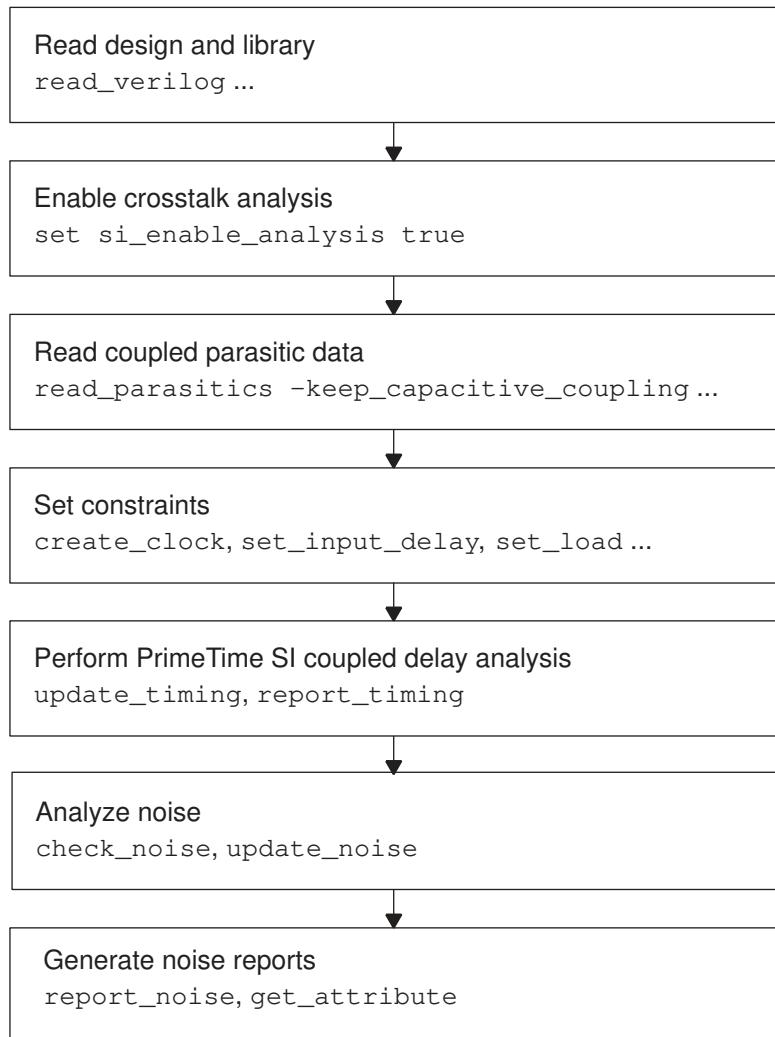
There are several ways to fix a crosstalk noise problem, such as changing the physical routing to avoid cross-coupling, inserting a buffer at the victim net, or increasing the strength of the victim net driver. To confirm the fix and to check for any new crosstalk problems, analyze the fixed design again.

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 180](#).

Figure 180 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 36](#).

Table 36 Noise analysis commands

Command	Purpose
<code>check_noise</code>	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.
<code>get_attribute si_noise_*</code>	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.
<code>get_noiseViolationSources</code>	Creates a collection of noise violation sources that are propagated to failing noise endpoints; used only in the “report at endpoint” analysis mode.
<code>remove_input_noise</code>	Removes noise bump annotations previously set on ports or pins with the <code>set_input_noise</code> command.
<code>remove_noiseImmunityCurve</code>	Removes noise immunity information previously set on ports or pins with the <code>set_noise_immunity_curve</code> command.
<code>remove_noiseLibPin</code>	Reverses the effects of the <code>set_noise_lib_pin</code> command.
<code>remove_noiseMargin</code>	Removes noise margins previously set on ports or pins with the <code>set_noise_margin</code> command.
<code>remove_si_noise_analysis</code>	Reverses the effects of the <code>set_si_noise_analysis</code> command.
<code>remove_si_noise_disable_statistical</code>	Reverses the effects of the <code>set_si_noise_disable_statistical</code> command.
<code>remove_steady_state_resistance</code>	Removes steady-state resistance information previously set on cells with the <code>set_noise_margin</code> command.
<code>report_noise</code>	Reports noise effects, including the width, height, and slack of worst-case noise bumps.
<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.

Table 36 Noise analysis commands (Continued)

Command	Purpose
report_noise_parameters	Reports the settings made with the set_noise_parameters command.
report_noiseViolationSources	Reports the noise violation sources that are propagated to failing noise endpoints; used only in the “report at endpoint” analysis mode.
reset_noise_parameters	Resets all the set_noise_parameters parameters to their default settings.
set_input_noise	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.
set_noise_derate	Derates (modifies) the calculated noise height or width by a specified fraction or absolute amount.
set_noise_ immunity_curve	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.
set_noise_lib_pin	Sets the noise characteristics of an input pin or output pin to match the noise characteristics of a specified library pin.
set_noise_margin	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.
set_noise_parameters	Specifies options for noise analysis: effort level, aggressor arrival times, beyond-rail analysis, noise propagation, and source/endpoint noise reporting.
set_si_noise_analysis	Includes or excludes specified nets for crosstalk noise analysis.
set_si_noise_disable_statistical	Disables statistical analysis for the specified nets when using composite aggressor analysis.
set_steady_state_resistance	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.

Table 36 Noise analysis commands (Continued)

Command	Purpose
<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 181](#) and [Figure 182](#) show the two noise reporting modes.

Figure 181 Report at source

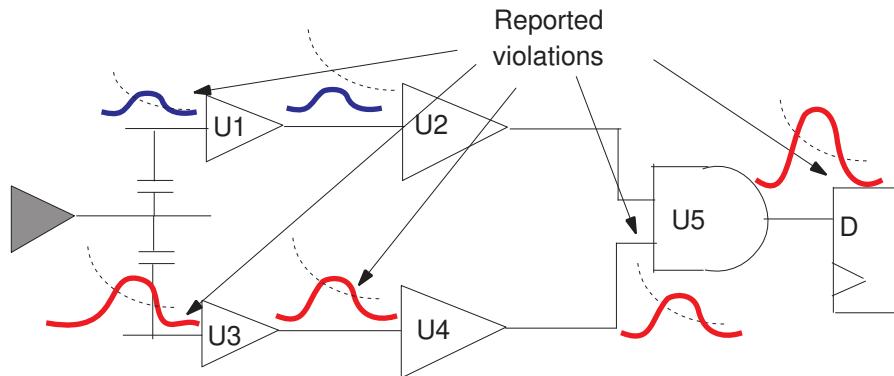
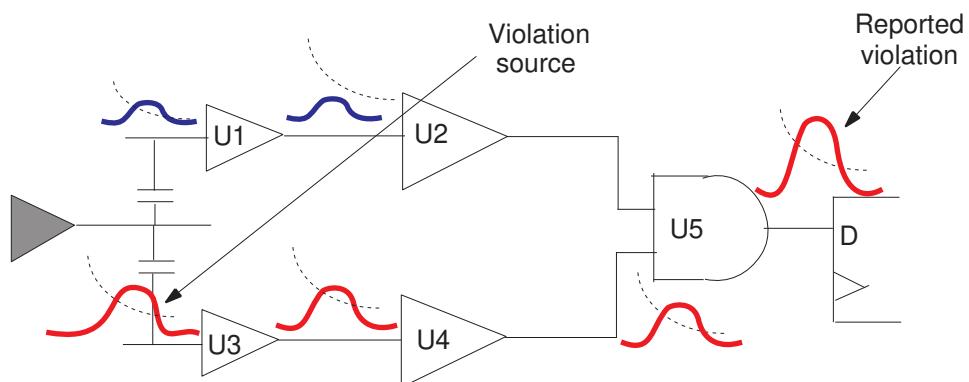


Figure 182 Report at endpoint



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 181](#), 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 182](#), 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 analysis mode requires a full timing update. To save time, set the analysis 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.

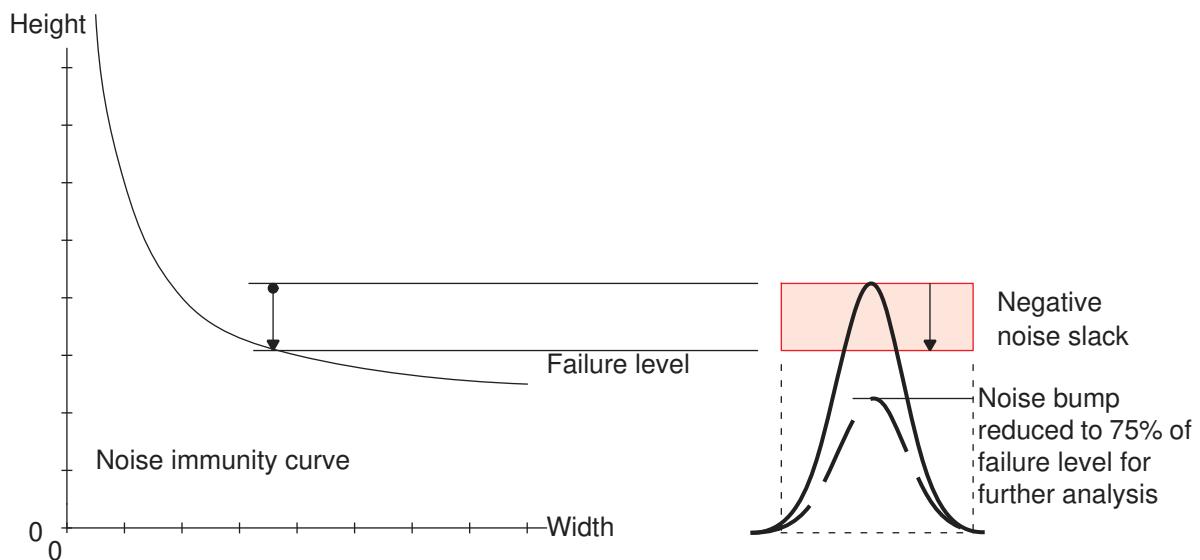
Derating Noise Results

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 183](#).

Figure 183 Input noise bump reduction for propagation analysis



To specify a different reduction factor, set the `si_noise_limit_propagation_ratio` variable to the required ratio, which must be a value between 0.0 and 1.0. 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 [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
<hr/>			
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. The specified pin should be a load pin (a cell input pin or bidirectional pin, not an output 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 `-async_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
<hr/>				
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/INVDS3
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.
--------	-------	------	-----------------

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
Required	0.581570	(0.581570 * 0.919737)
Actual	0.497124	(0.497124 * 0.919737)
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 [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 37](#).

Table 37 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 37](#) 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

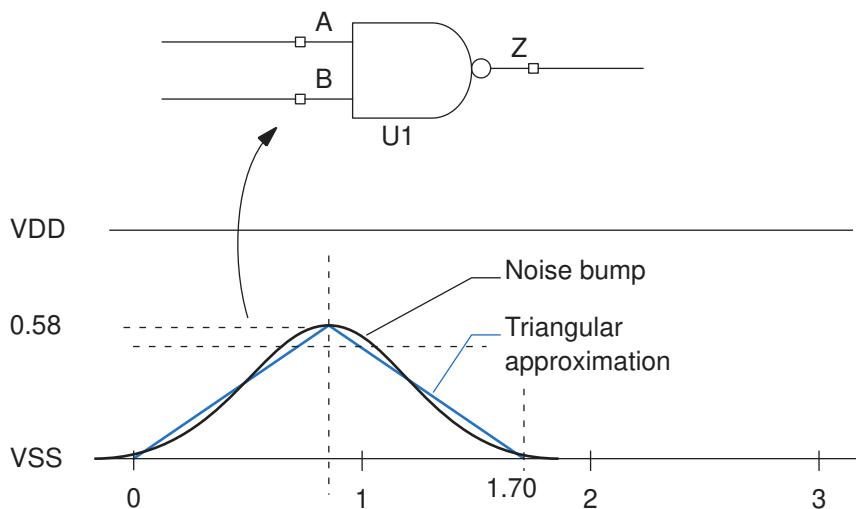
Rather than allow the tool to calculate propagated noise bumps, you can explicitly specify a noise bump at any port or pin. To do so, use the `set_input_noise` command. In the command, you specify the port or pin on which to apply the noise and the characteristics

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 184](#).

Figure 184 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

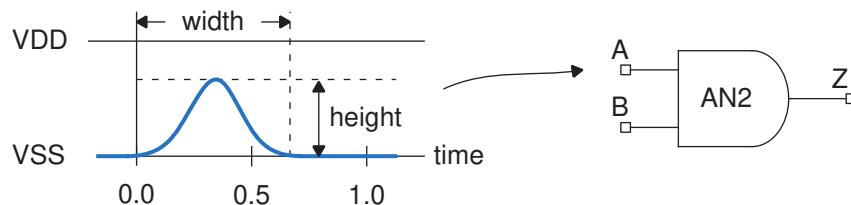
- The type of noise bump: above-high, below-high, above-low, or below-low. To fully specify the noise immunity characteristics of a cell or design, you need one `set_noise_immunity_curve` command for each type of noise bump.
- The three positive coefficient values that define the curve
- The input pin 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
```

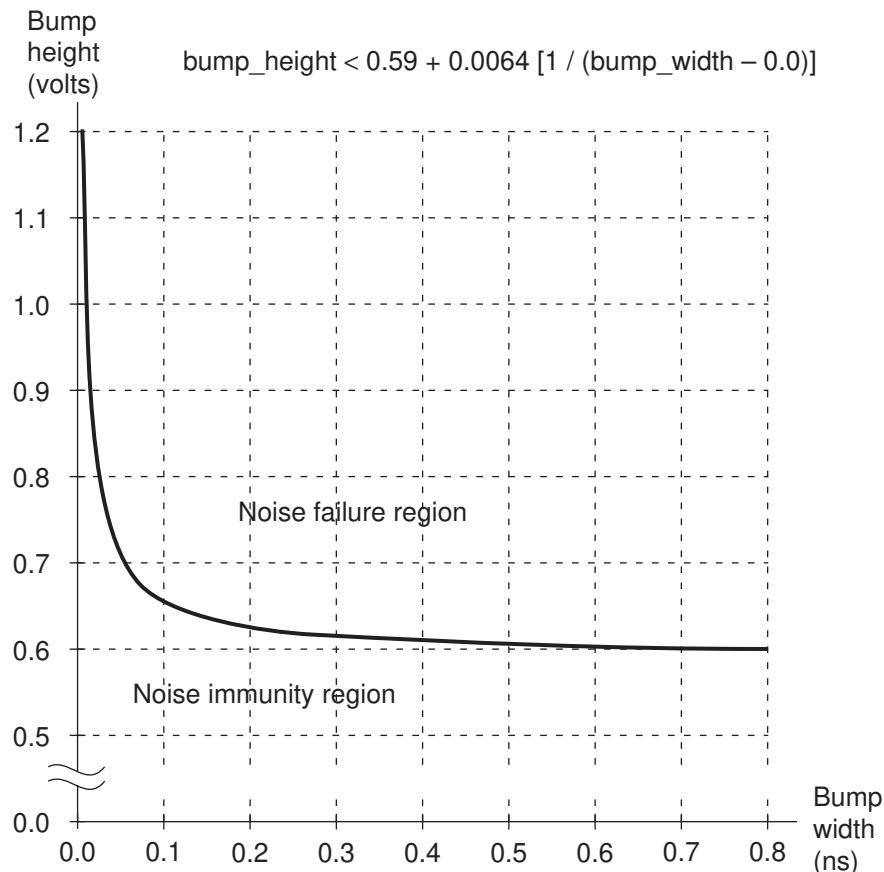
This command defines the above-low noise immunity curve for input A of library cell AN2, as shown in [Figure 185](#). [Figure 186](#) shows the resulting noise immunity curve.

Figure 185 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 186 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:

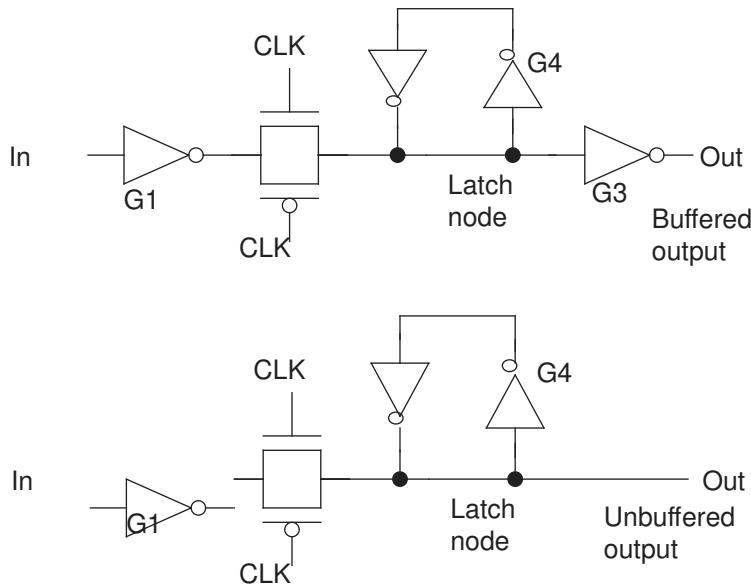
1. Static noise immunity curve annotated using the `set_noise_immunity_curve` command
2. DC noise margin annotated using the `set_noise_margin` command
3. Arc-specific noise immunity curve from library
4. Pin-specific noise immunity curve from library
5. CCS noise model from library
6. DC noise margin 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 187](#) 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 187 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 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 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, you can use a circuit simulator such as HSPICE 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.

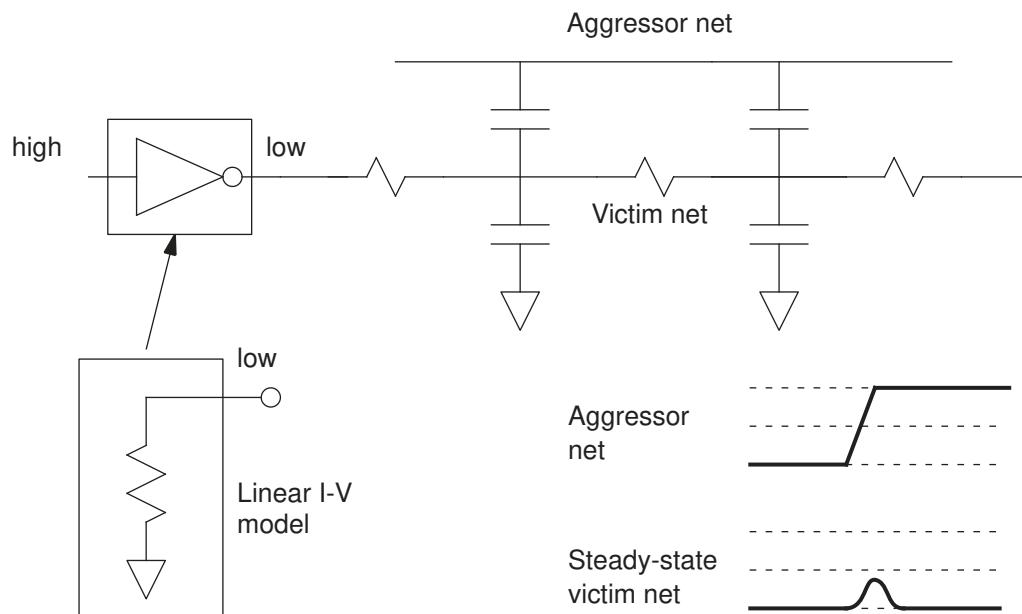
See Also

- [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 188](#). 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 188 Linear I-V model for a cell with steady-state low output

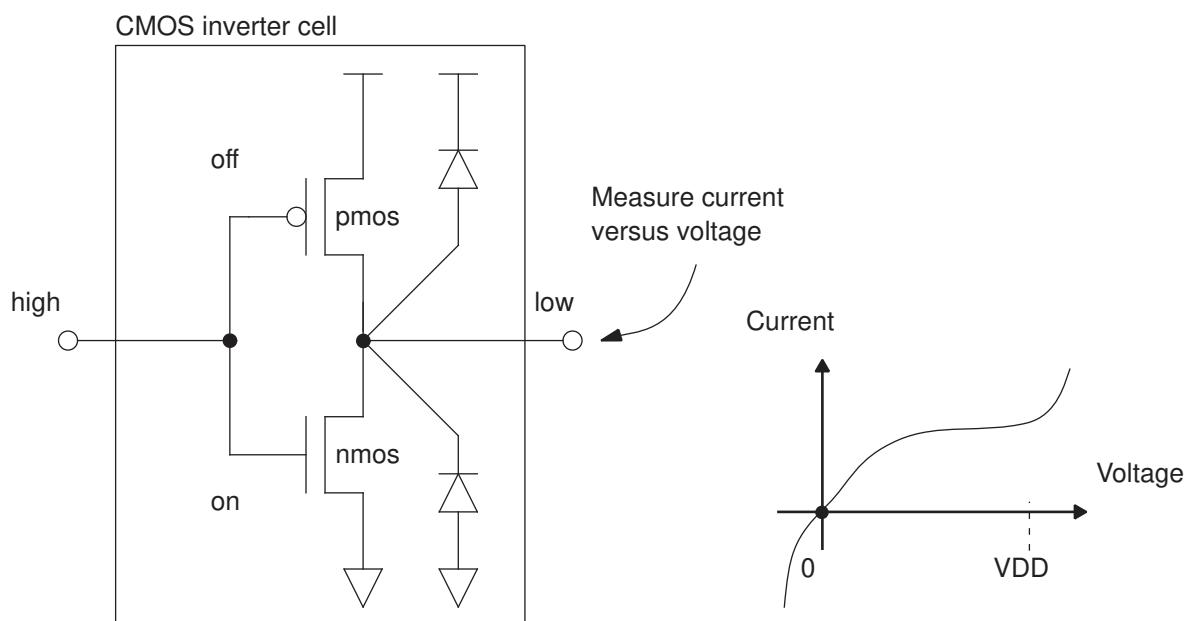


The I-V characteristics of a cell can be determined by placing the cell output into either the low or high state, and then measuring the current at different voltages at the output, as shown for the CMOS inverter in [Figure 189](#). The circuit diagram includes the diodes at the output that exist because 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 189 I-V characterization of a steady-state low output

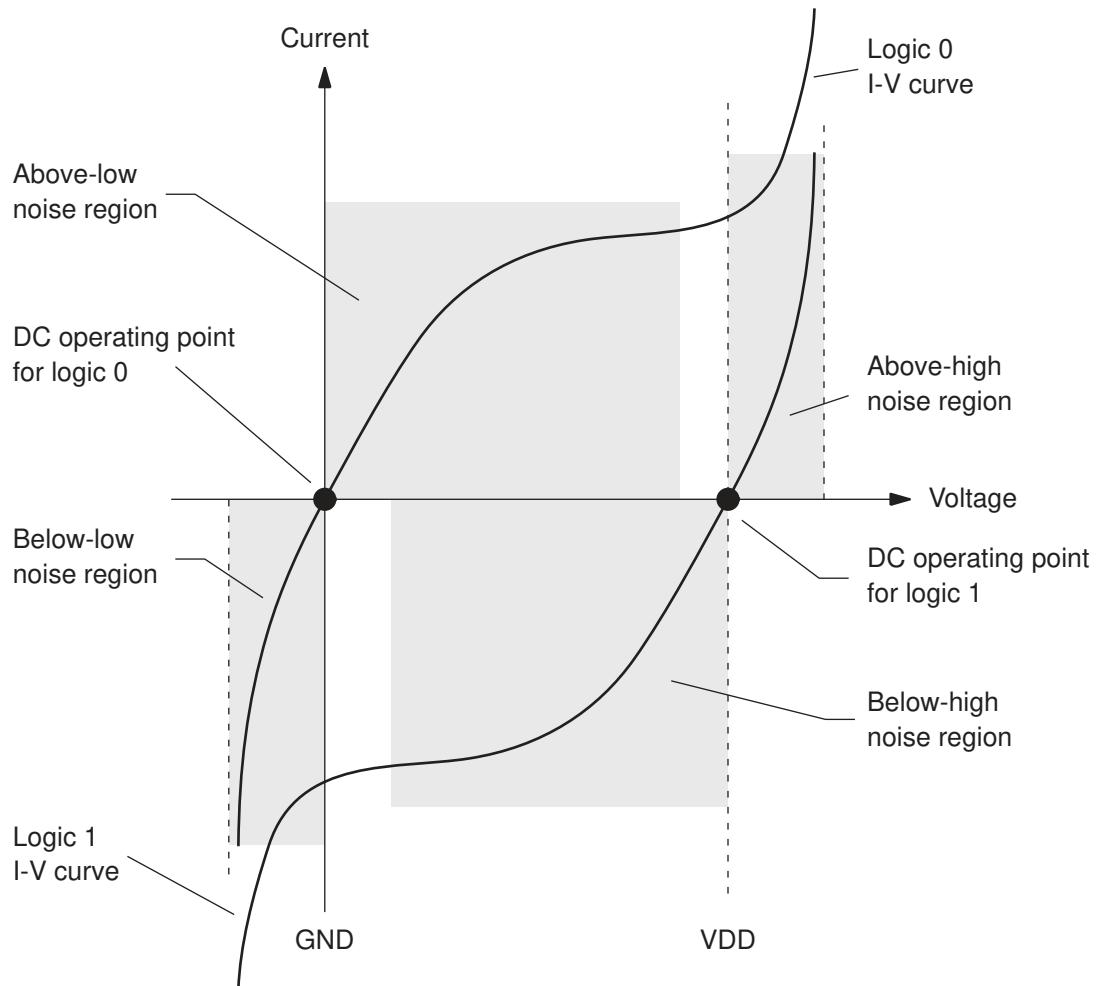


A similar I-V plot can be obtained by placing the CMOS output at logic 1. Its steady-state operating point is at (VDD, 0.0).

[Figure 190](#) 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.

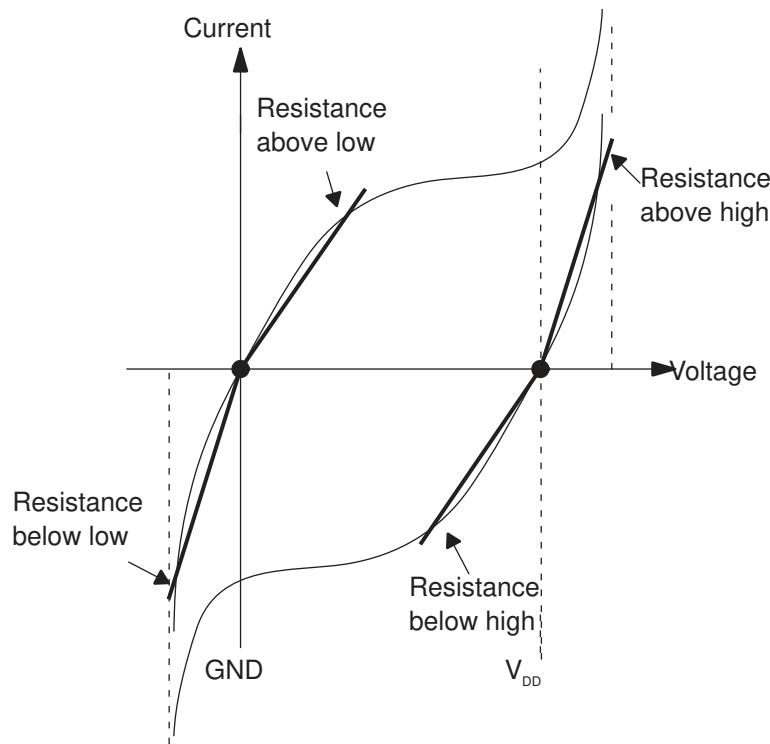
The four shaded portions indicate the operating regions when noise bumps occur: below low, above low, below high, and above high. The library syntax allows the I-V characteristics to be specified as two resistors (one each for above low and below low), as a piecewise linear model, or as a polynomial function.

Figure 190 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 191. 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 191 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 38](#) describes the methods for specifying 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 38 Library specification methods for output I-V characteristics

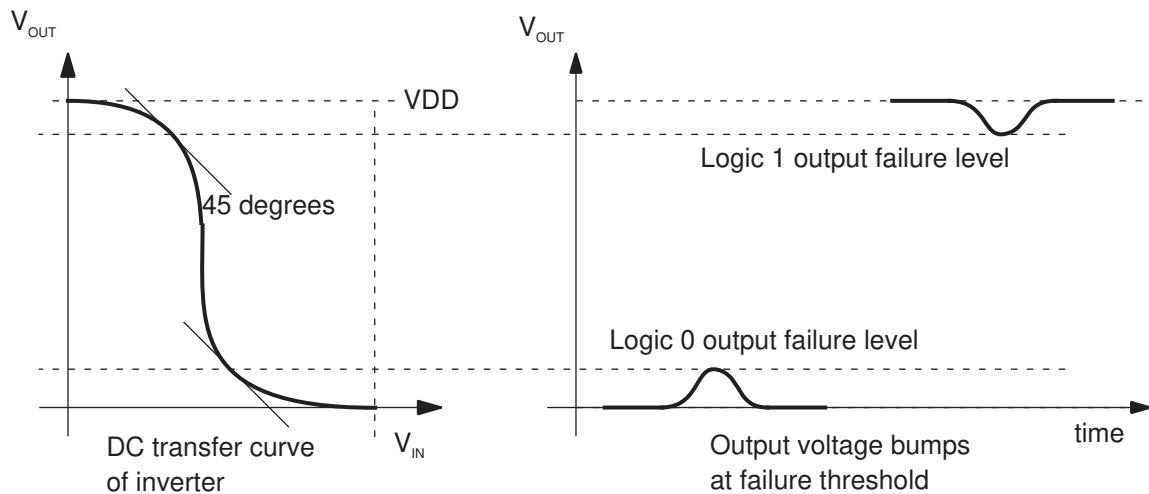
Specification method	Library type	How specified in library
Lookup tables	NLDM	Two lookup tables per timing arc for steady-state low and steady-state high. Tables describe current as a function of voltage.
Steady-state resistance	NLDM	Four floating-point resistance values per timing arc for above low, 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 192](#). However, the creator of the library might use some other failure criteria.

Figure 192 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 39](#) 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 39 Library specification methods for noise immunity

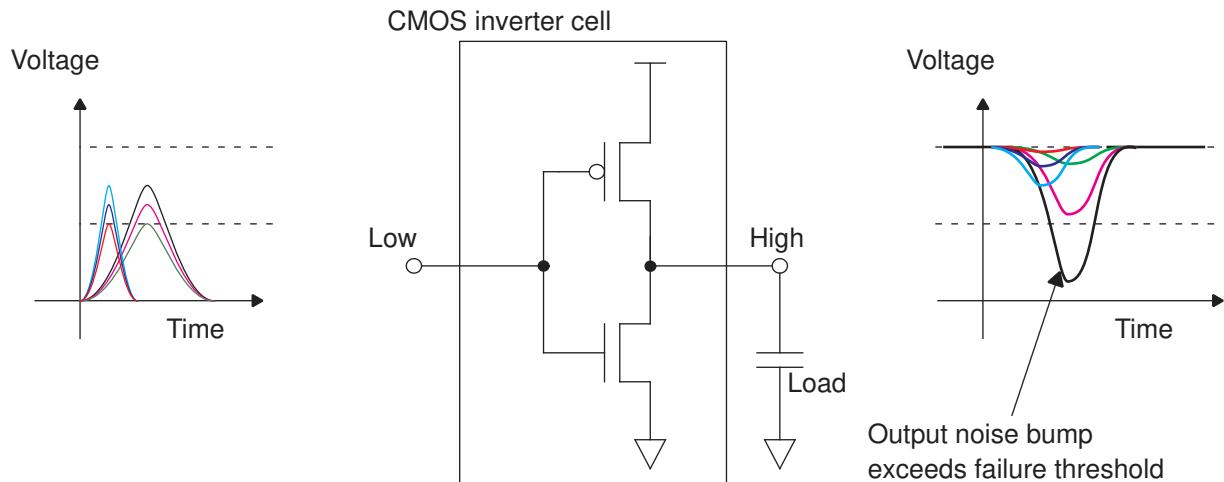
Specification method	Library type	How specified in library
Hyperbolic noise immunity curves (per input)	NLDM	Four hyperbolas per input specify the maximum noise bump height as a function of noise bump width: one hyperbola each for input noise bumps above low, below low, above high, and below high. Each hyperbola is specified with three floating-point coefficients. Can also be specified with the <code>set_noise_immunity_curve</code> command.
Noise immunity 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	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 by applying noise bumps of varying heights and widths to the input, as shown for the CMOS inverter in [Figure 193](#). 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.

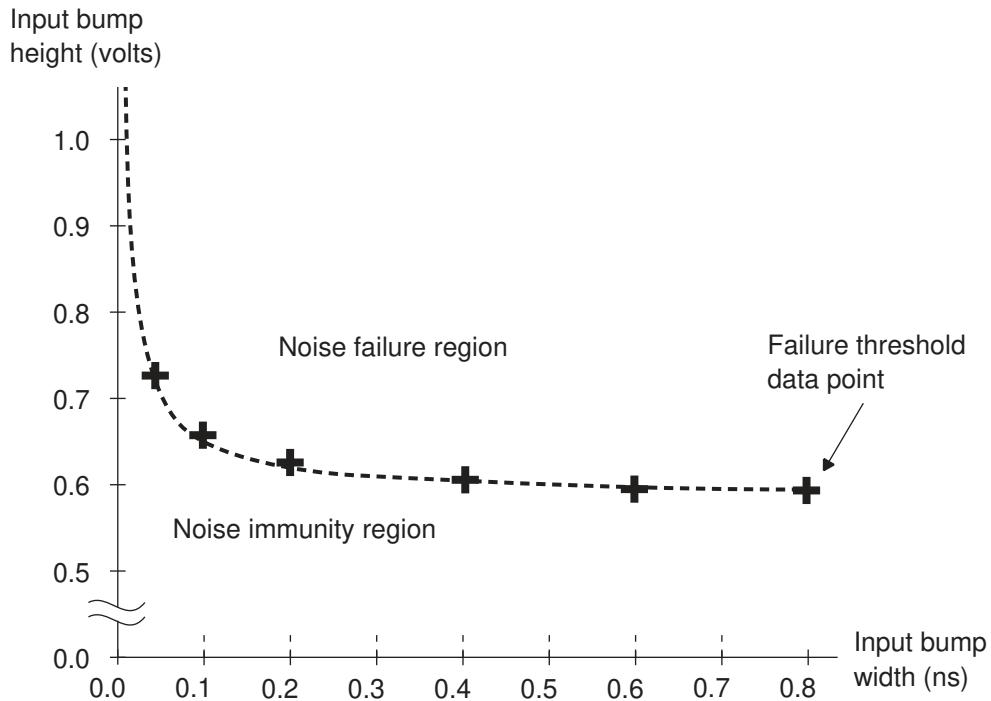
Figure 193 Noise immunity characterization of input



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 194](#). 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 194 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

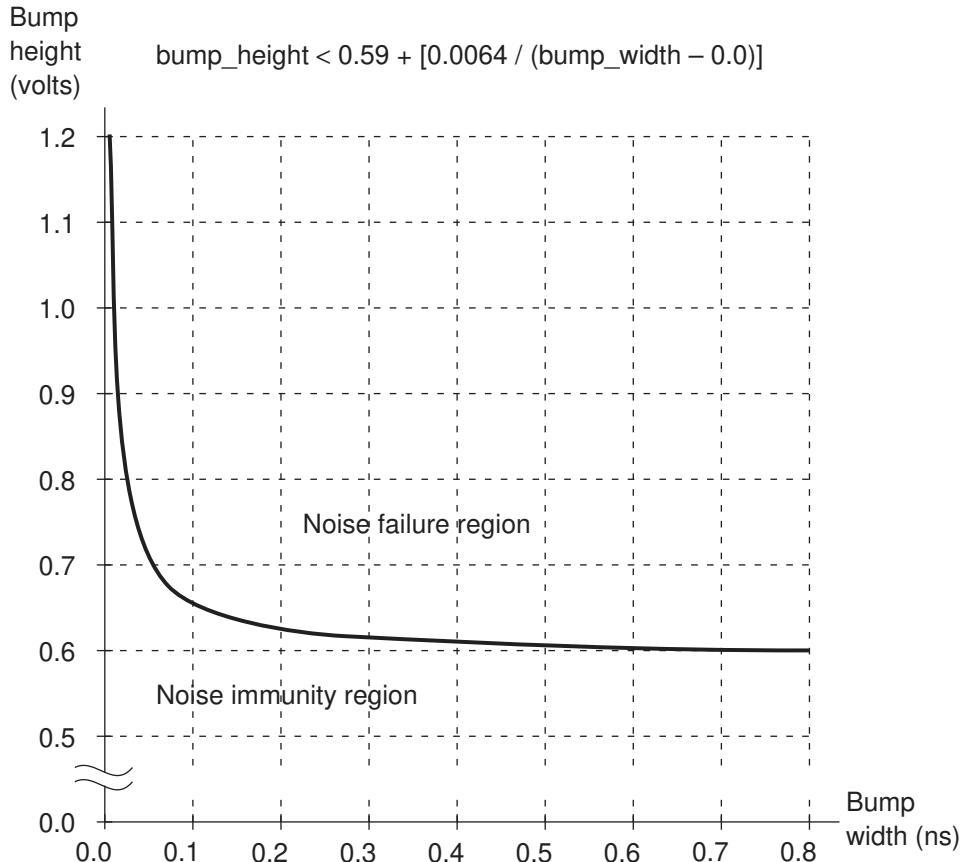
c_2 = size parameter for the hyperbolic curve

c_3 = time-value offset

The three coefficients should be chosen to match the hyperbolic curve to the data points obtained by characterization.

For example, [Figure 195](#) 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 195 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`.

Noise Immunity Polynomials and Tables

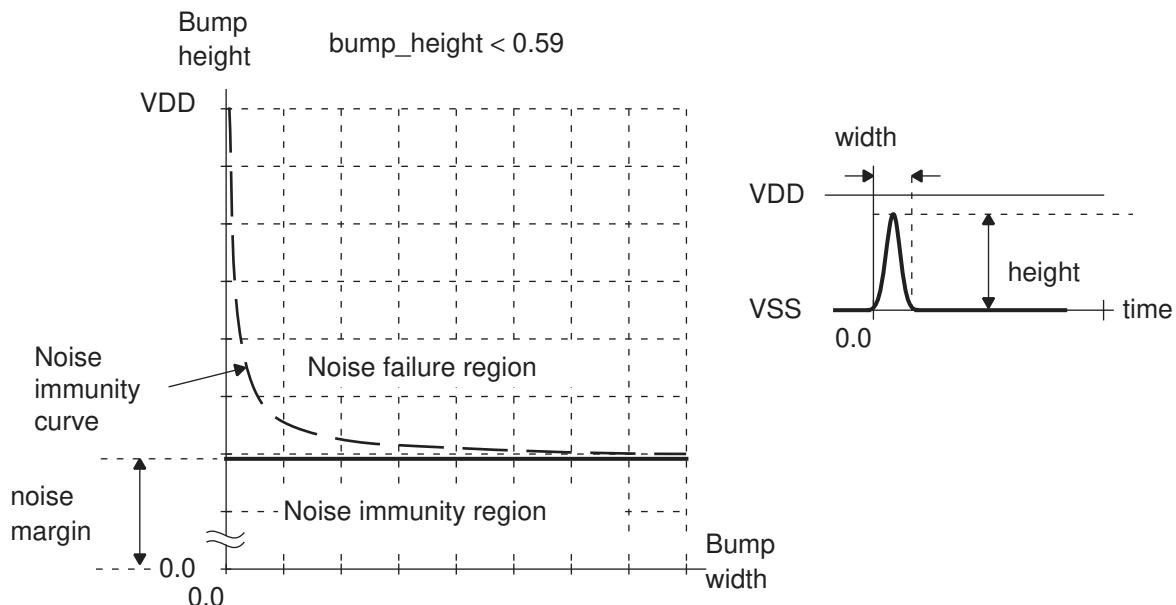
In cases where the noise immunity characteristics vary significantly due to different output loads or different paths through the cell, the library can use polynomials or lookup tables instead of hyperbolic noise immunity curves. Using polynomials or lookup tables, the library specifies the maximum input 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 196](#) compares noise immunity curves and bump height noise margins.

Figure 196 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} .

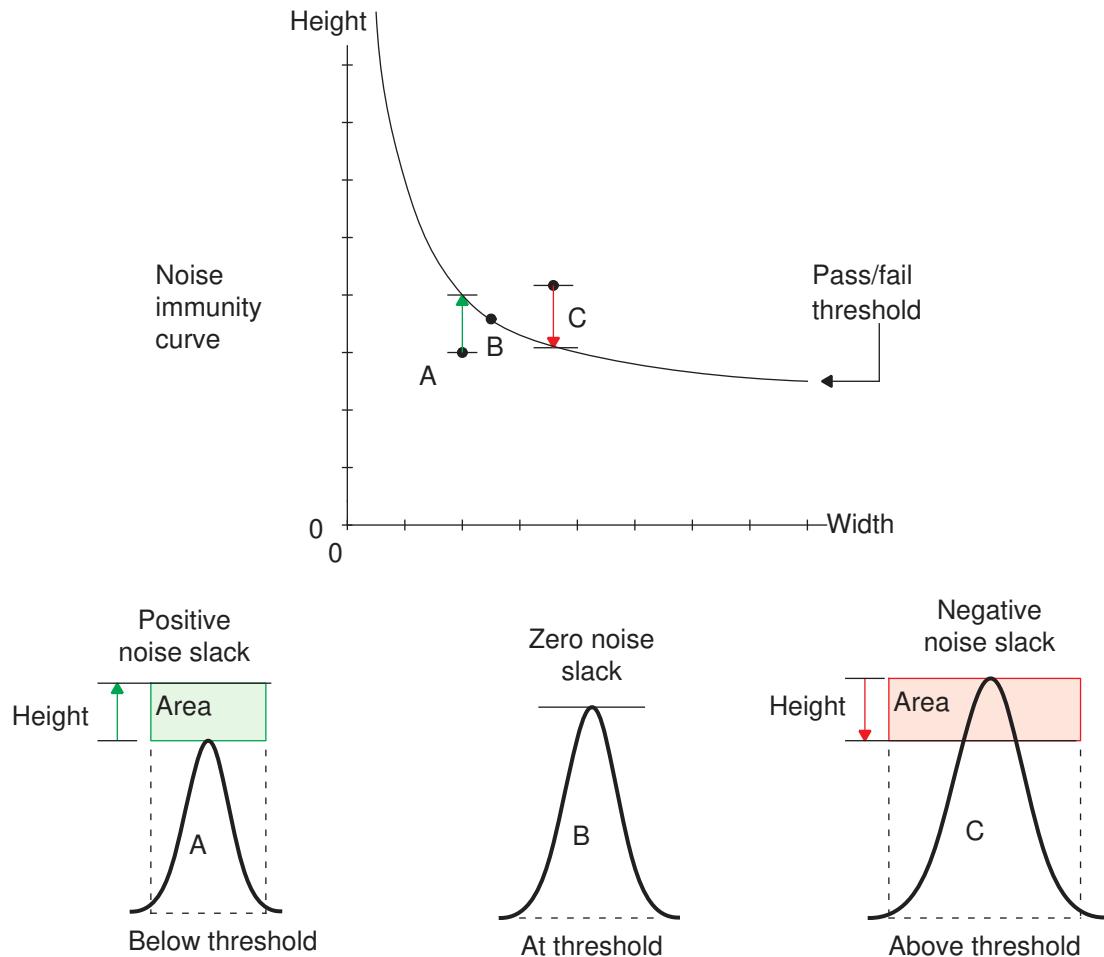
Noise Slack

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 197](#). 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 197 Noise slack calculation



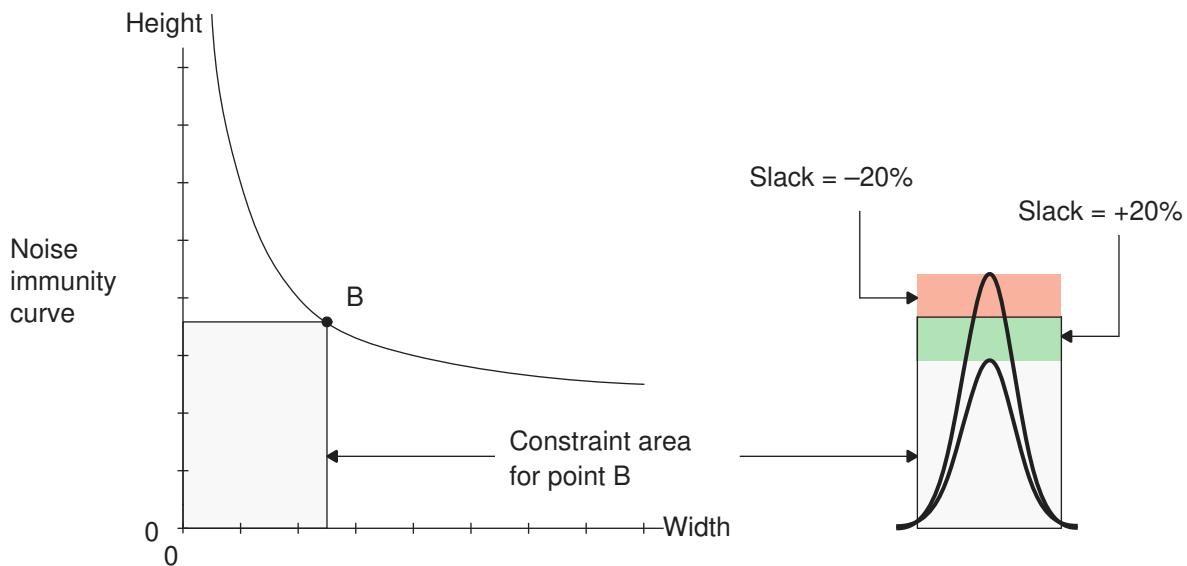
The “area” type noise slack is the voltage margin (height of the curve above the data point) multiplied by the noise bump width, as indicated by the shaded rectangles in Figure 197. For a noise bump below the failure threshold, the slack is positive, or for a noise bump above the failure threshold, the 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 198](#).

Figure 198 Area percent slack calculation

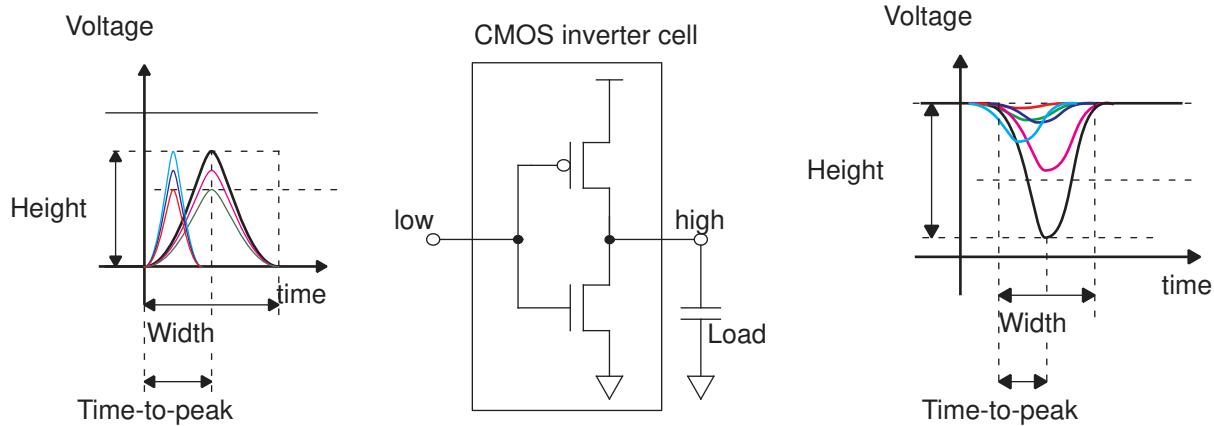


Propagated Noise Characteristics

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 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 199](#). 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 199 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 40 describes the method 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 40 Library specification methods for propagated noise

Specification method	Library type	How specified in library
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.

17

Advanced Analysis Techniques

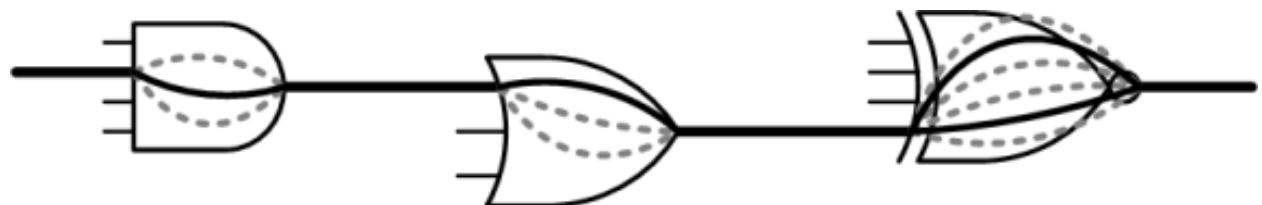
To learn about PrimeTime advanced timing analysis techniques, see

- [Parallel Arc Path Tracing](#)
 - [Support for Retain Arcs](#)
 - [Asynchronous Logic Analysis](#)
 - [Three-State Bus Analysis](#)
 - [Data-to-Data Checking](#)
 - [Time Borrowing in Latch-Based Designs](#)
 - [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 following path tracing.

Figure 200 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

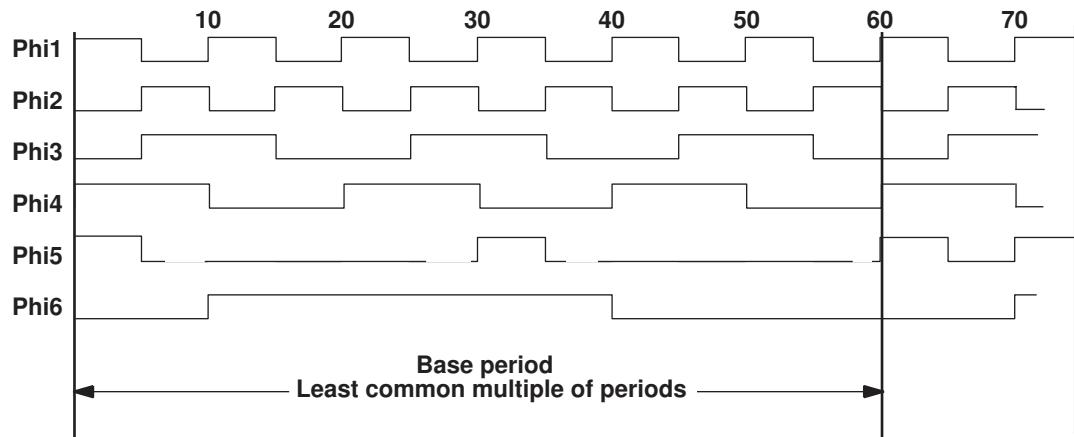
To simplify the timing verification designs with asynchronous logic, isolate the asynchronous logic into separate blocks, then disable the timing of these blocks.

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.)

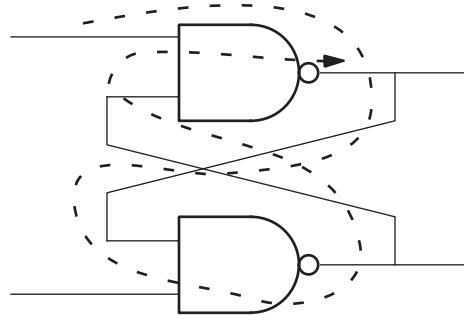
Figure 201 Base period of clocks



Combinational Feedback Loop Breaking

A combinational feedback loop is a path that can be traced through combinational logic back to its starting point.

Figure 202 Combinational feedback loop

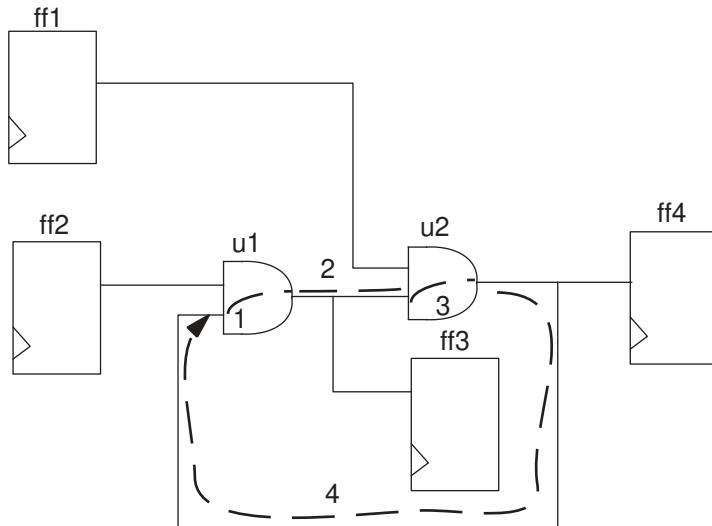


To analyze such a path, PrimeTime must break the loop (stop tracing the path) at some point within the loop. Check a design for the presence of combinational feedback loops with the `check_timing -include loops` command.

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.

In the following example, no timing arc of the combinational loop can be broken without a valid path of the design also being broken.

Figure 203 Loop breaking example



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.

There are multiple places in a feedback loop that could be broken. If you want to break a feedback loop at a timing arc different from the one selected by default, run the `set_disable_timing` command to break the loop at a specific point.

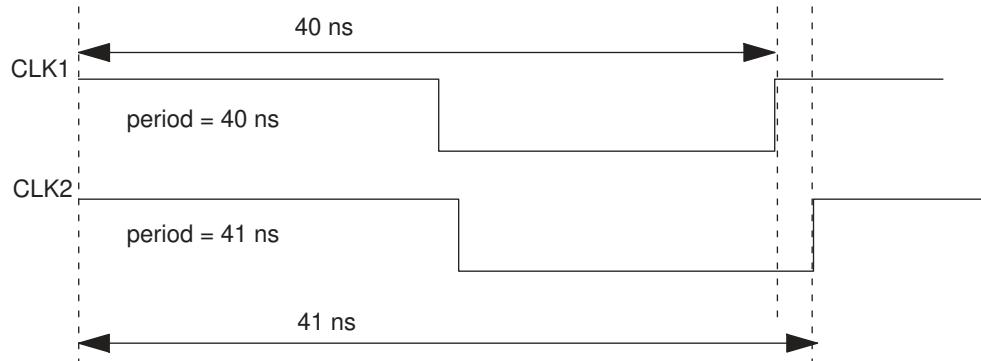
Design Compiler also uses loop breaking to analyze timing. The points at which Design Compiler and PrimeTime break a loop can be different, possibly leading to different timing results. If you want to ensure consistent loop breaking between the two tools, set the `timing_keep_loop_breaking_disabled_arcs` 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.

Figure 204 Unrelated clock waveforms



The clocks 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

The following circuit shows how PrimeTime performs transient bus contention checks and floating bus checks.

Figure 205 Circuit example

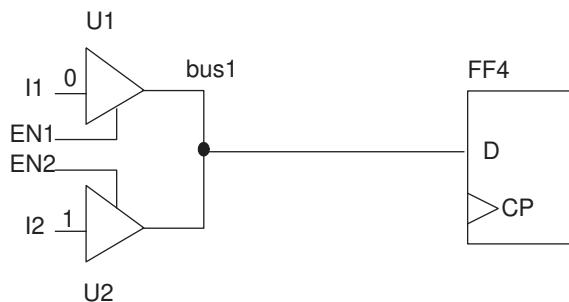
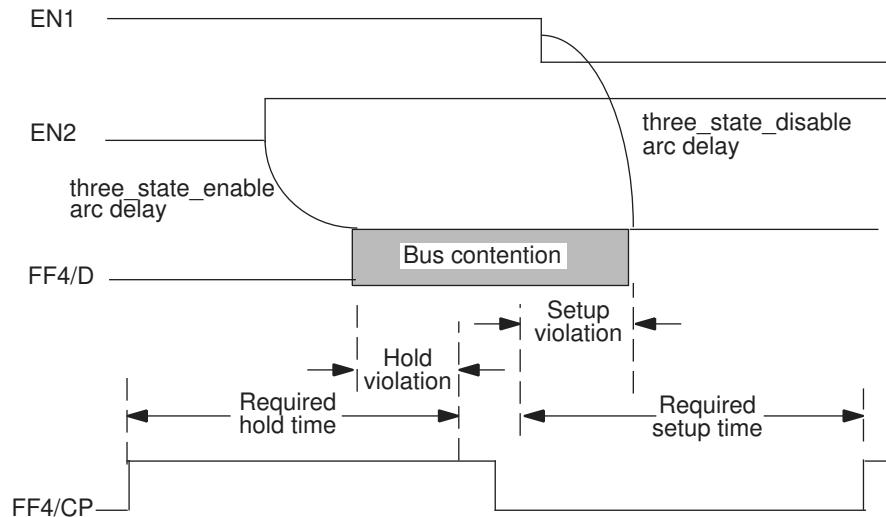


Figure 206 Transient bus contention check



In [Figure 205](#), 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 205](#) are as shown in [Figure 206](#).

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.

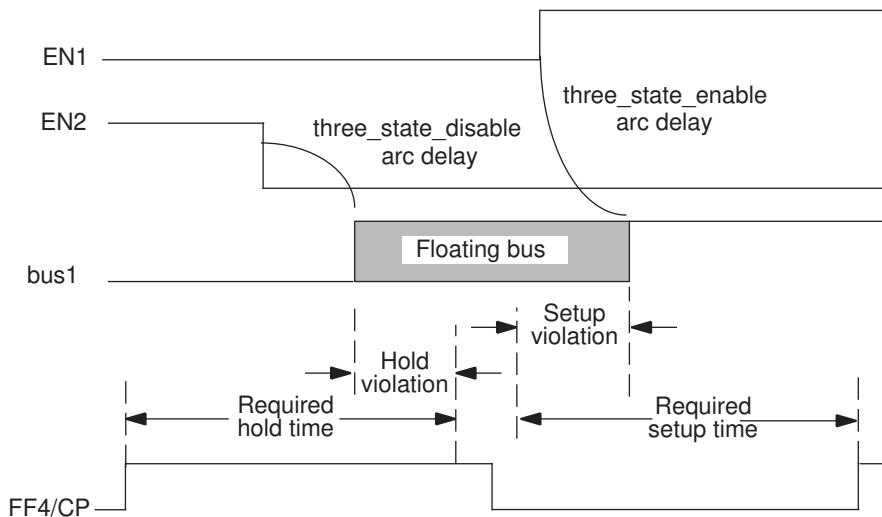
Even when you set this variable to `true`, PrimeTime considers the three_state_enable arc delay for setup violation checking and the three_state_disable arc delay for hold violation checking. This occurs during a floating bus region, which is described in [Performing Floating Bus Checks](#).

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.

The following figure shows another situation that can arise that yields a transient floating bus condition.

Figure 207 Floating bus contention check



In [Figure 207](#), 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:

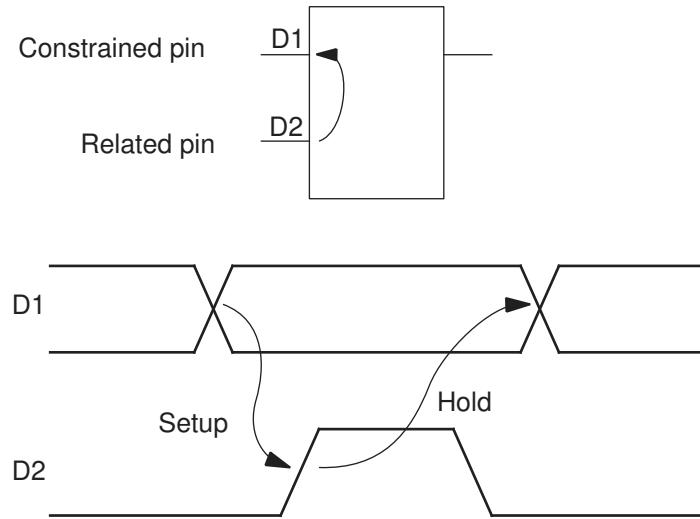
- Constraints on handshaking interface logic
- Constraints on asynchronous or self-timed circuit interfaces
- Constraints on signals with unusual clock waveforms that cannot be easily specified with the `create_clock` command
- Constraints on skew between bus lines
- Recovery and removal constraints 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

The following example shows a cell with a nonsequential constraint.

Figure 208 Simple data check example



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.

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:

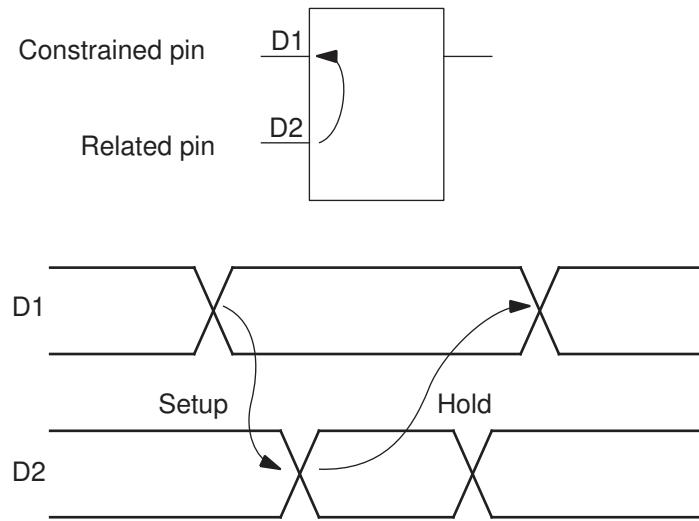
```
pt_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
pt_shell> set_data_check -rise_from D2 -to D1 -hold 6.0
```

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 the following figure.

Figure 209 Data checks on rising and falling edges

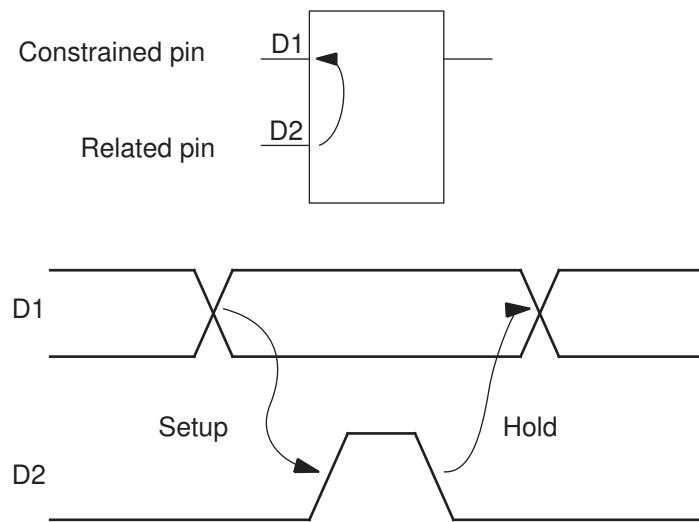


Define a no-change data check by specifying only a setup check from the rising edge and a hold check from the falling edge:

```
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
```

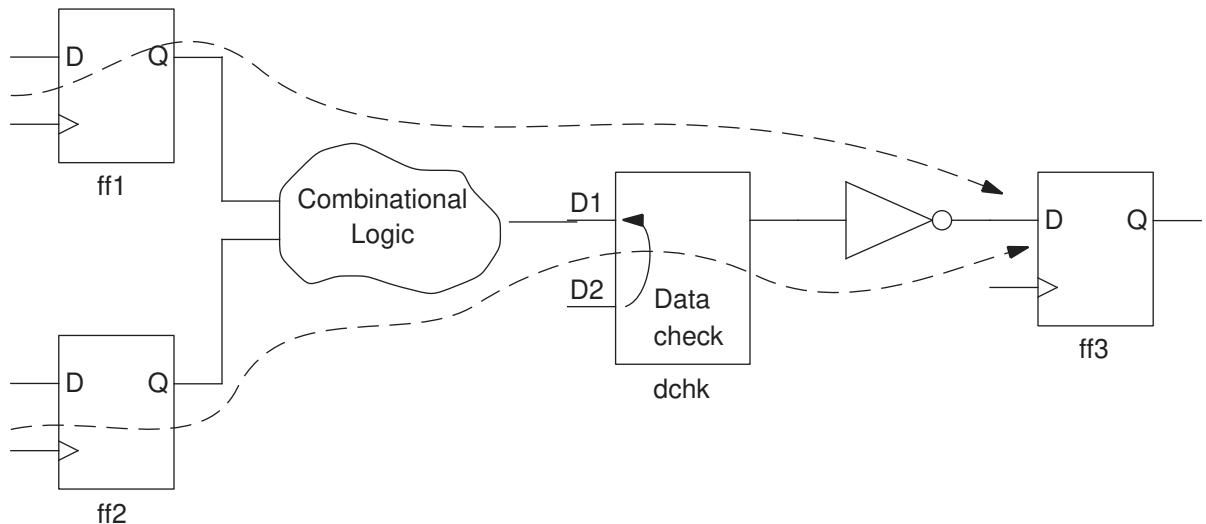
PrimeTime interprets this as a no-change check on a positive-going pulse. The resulting timing check is shown in the following figure.

Figure 210 No-change data check



Data checks are nonsequential, so they do not break timing paths. For example, in [Figure 211](#), 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 211 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 211](#), `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 documentation.

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.

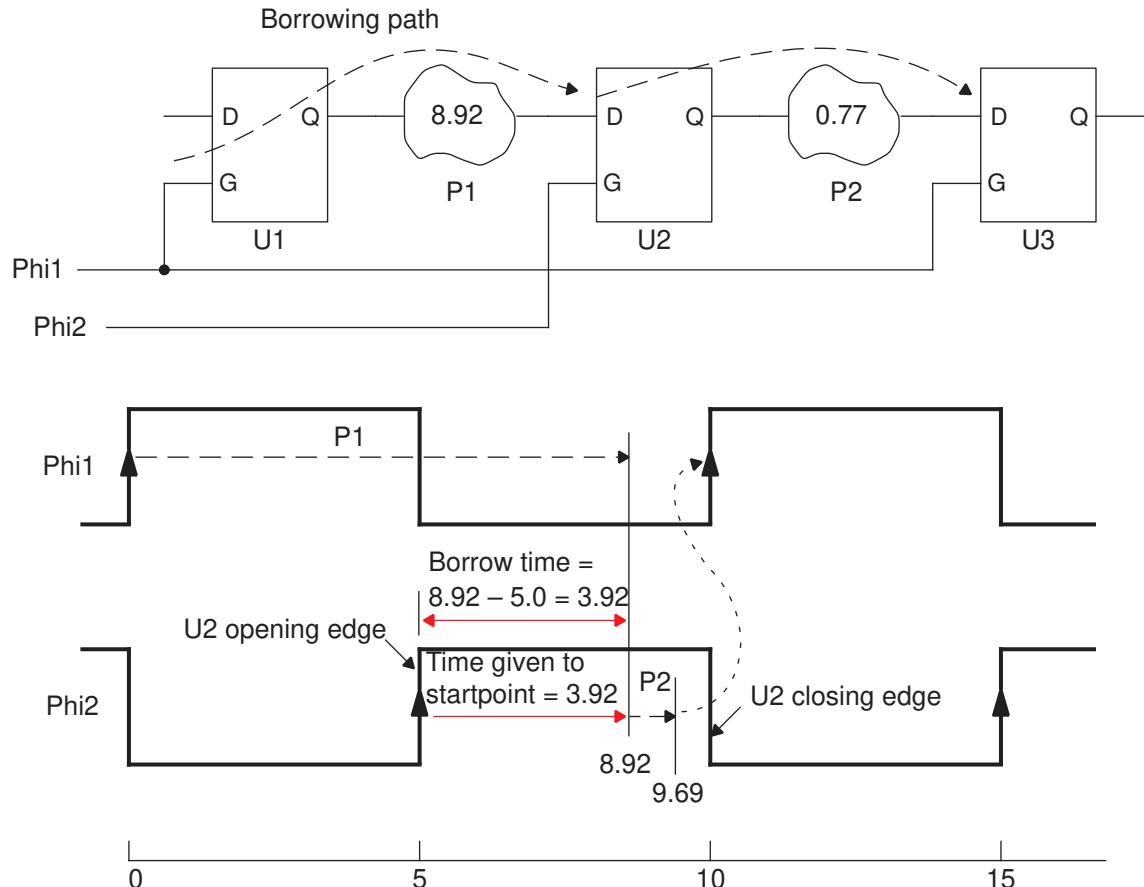
Time Borrowing in Latch-Based Designs

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 212](#) shows latch-based stages using a simple two-phase clocking scheme.

Figure 212 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 212, 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.

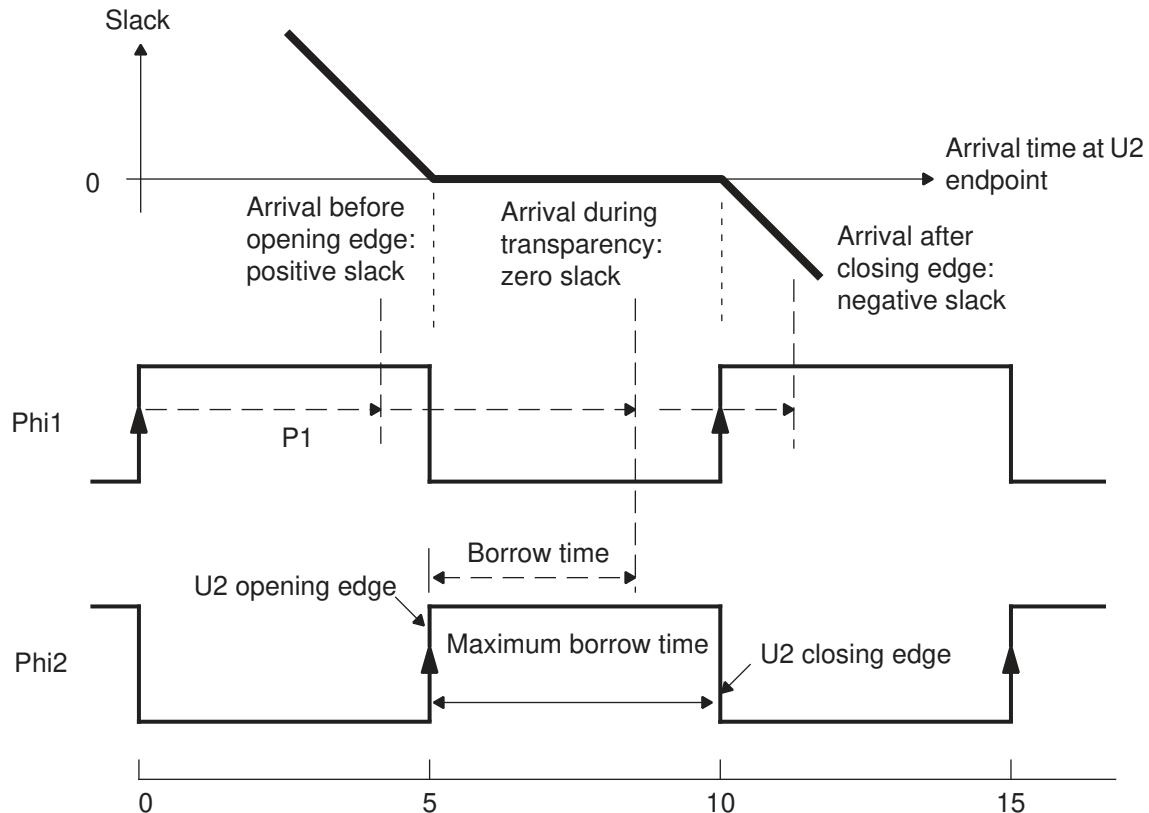
Latch Timing Reports

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 213](#) 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 212](#)). [Figure 213](#) does not consider the effects of latch setup time, latency, uncertainty, and clock reconvergence pessimism removal.

Figure 213 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 212.

```
*****
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
-------	------	------

Chapter 17: Advanced Analysis Techniques

Time Borrowing in Latch-Based Designs

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

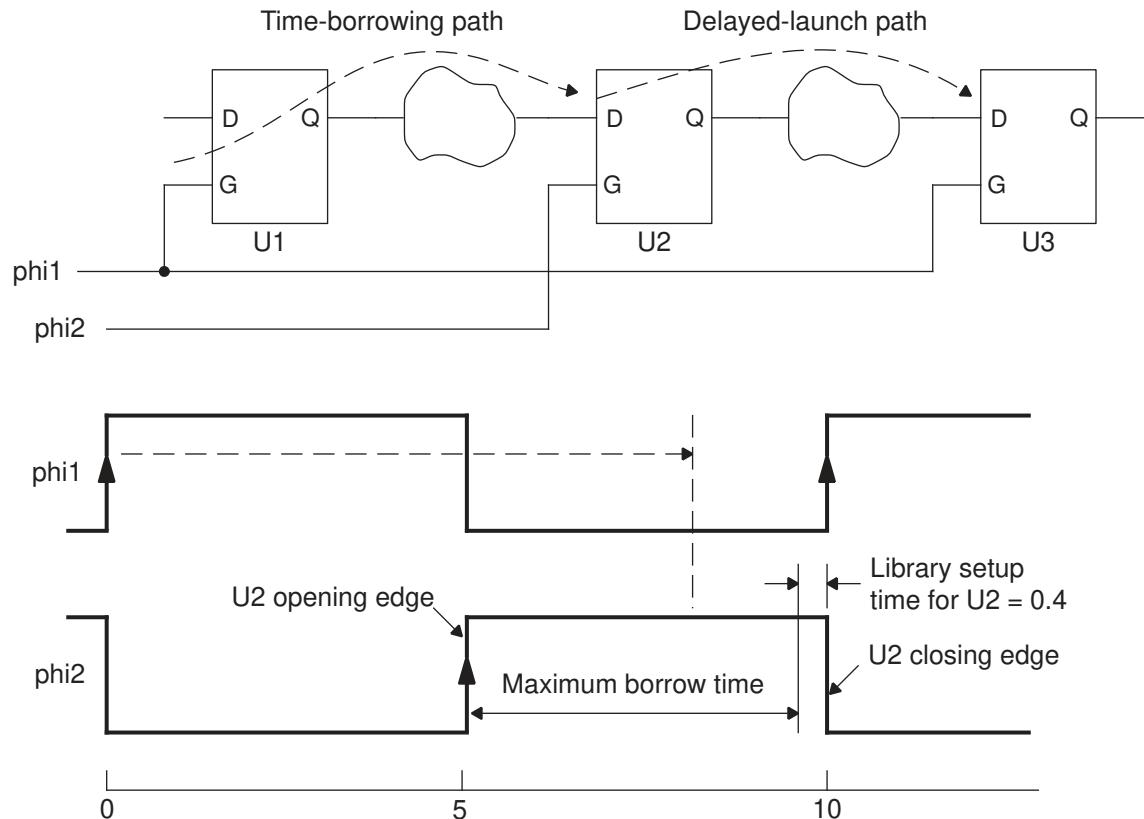
max time borrow	4.51
actual time borrow	0.00

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 214](#).

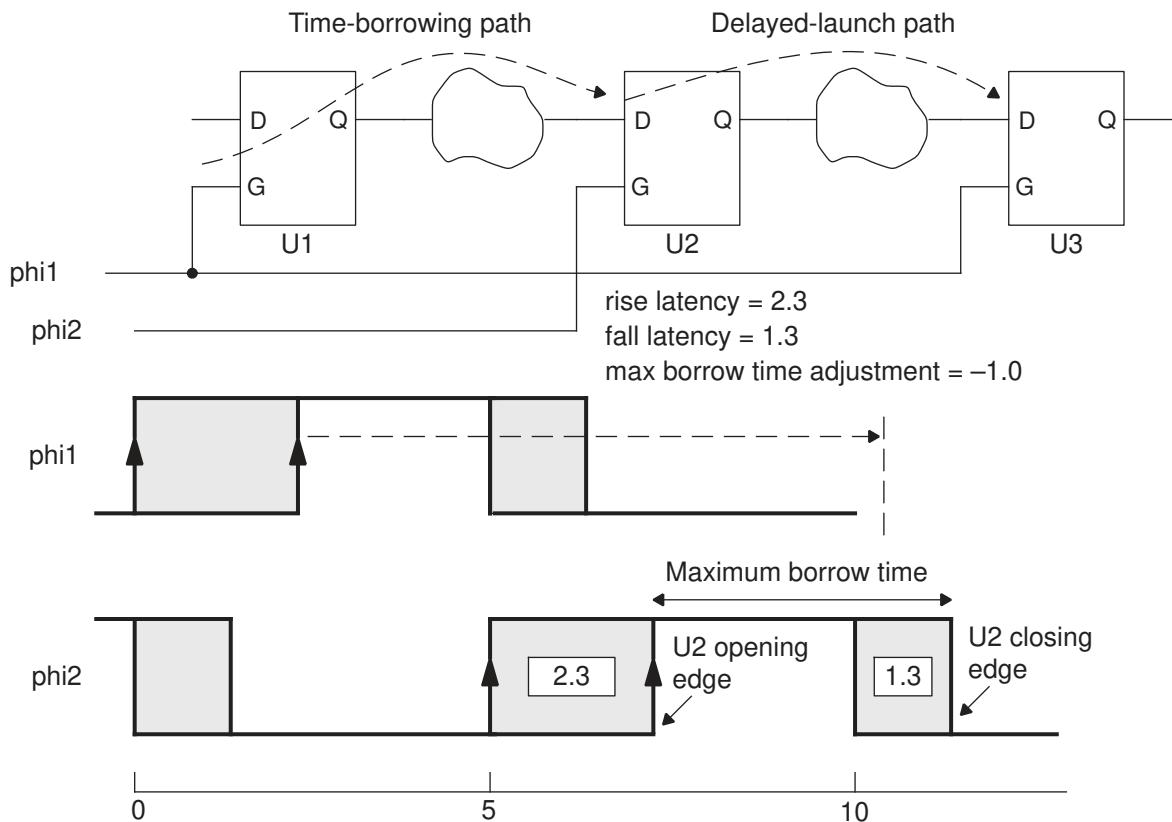
Figure 214 Maximum borrow time reduced by setup requirement



For better accuracy, PrimeTime adjusts this amount further for the effects of clock latency, clock uncertainty, and clock reconvergence pessimism removal.

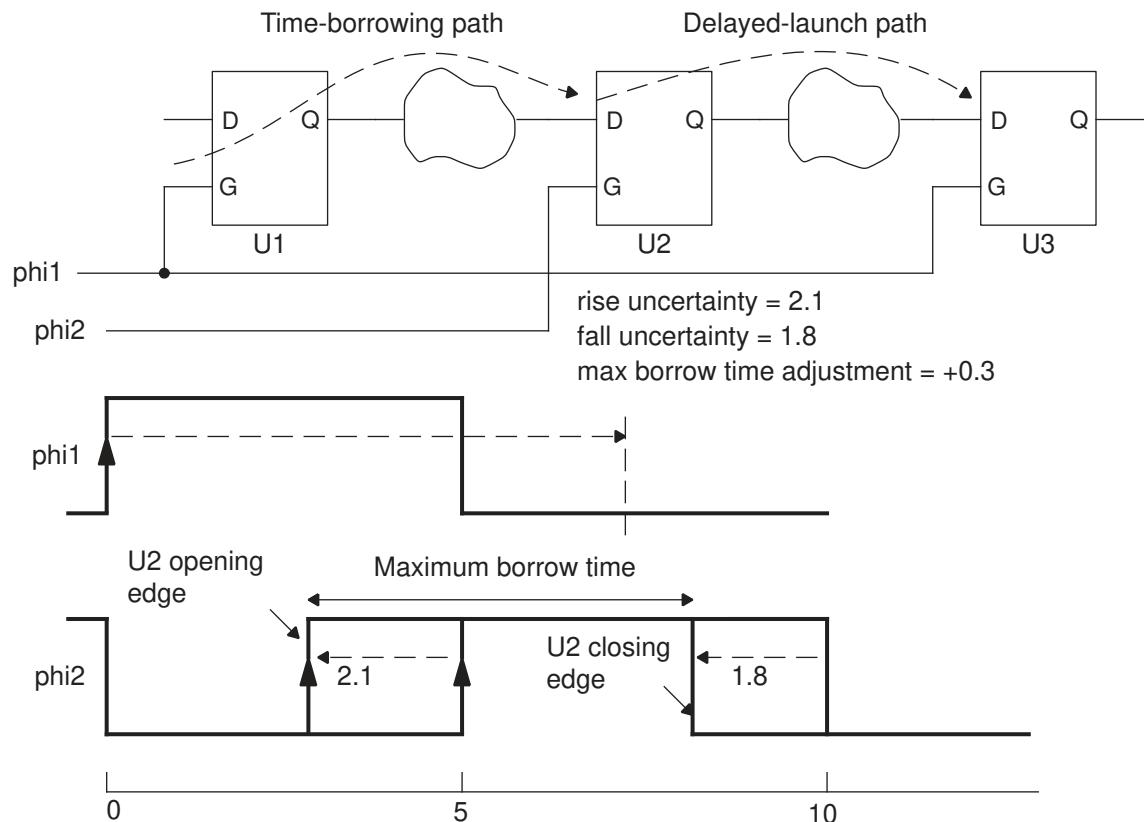
Clock latency can be defined with the `set_clock_latency` command or calculated by PrimeTime from propagated delays in the clock tree (enabled by `set_propagated_clock`). Latency values can be different for the rising and falling edges of the clock pulse, which can affect the pulse width and thus the maximum borrow time. See [Figure 215](#).

Figure 215 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 216](#).

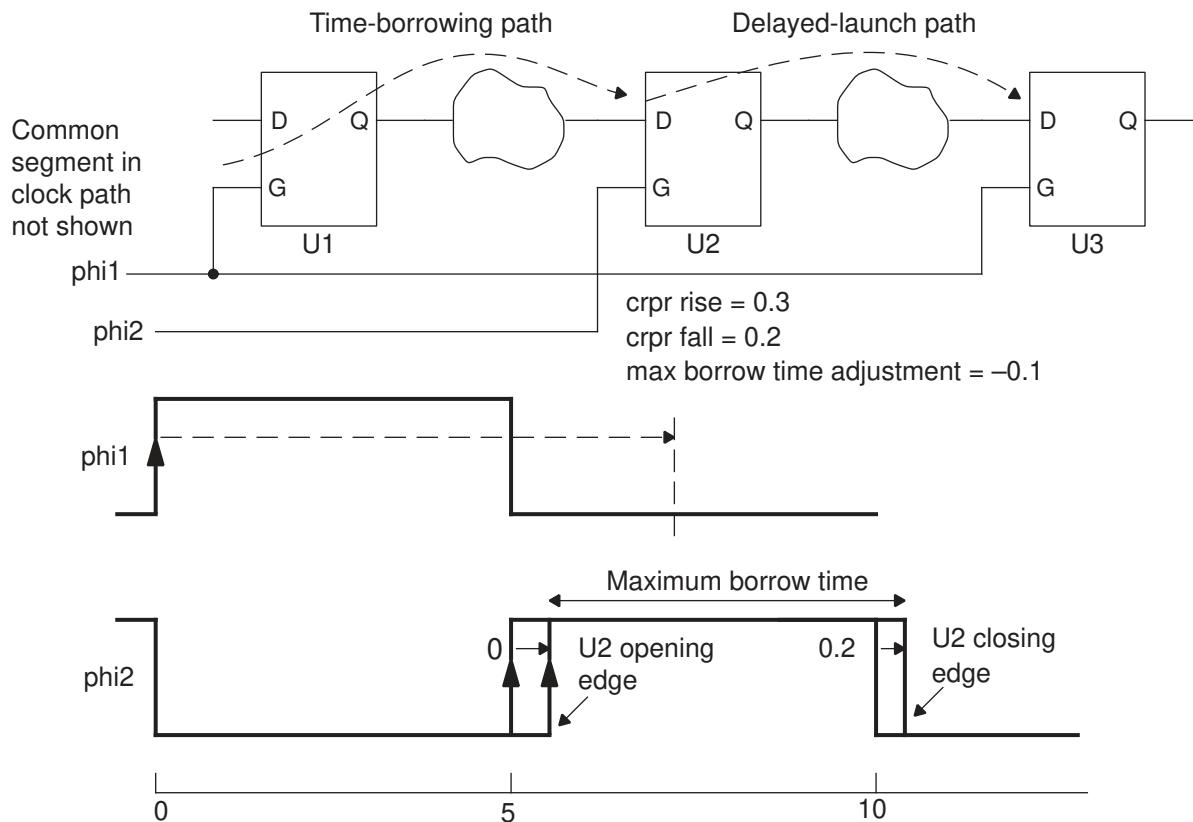
Figure 216 Maximum borrow time adjustment for uncertainty



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 217](#).

Figure 217 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:

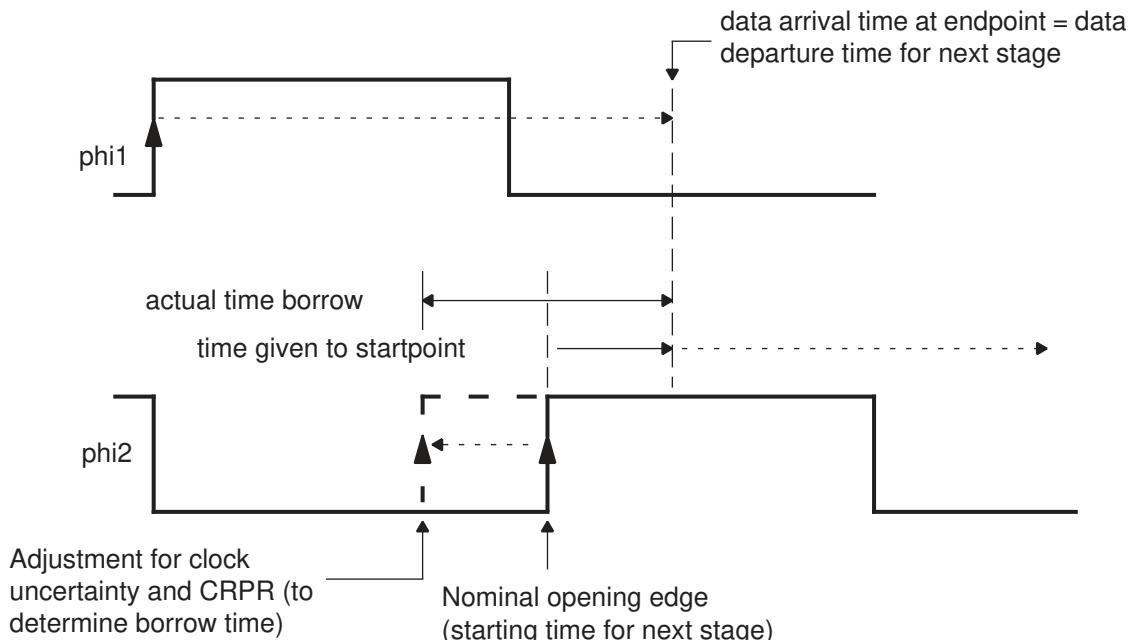
```
Time Borrowing Information
-----
CLK nominal pulse width          5.00
clock latency difference        -1.00
clock uncertainty difference     0.30
CRPR difference                 -0.10
library setup time                -0.40
-----
max time borrow                  3.80
-----
...
```

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 218](#). 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 218 Borrow time and time given to startpoint in transparent mode

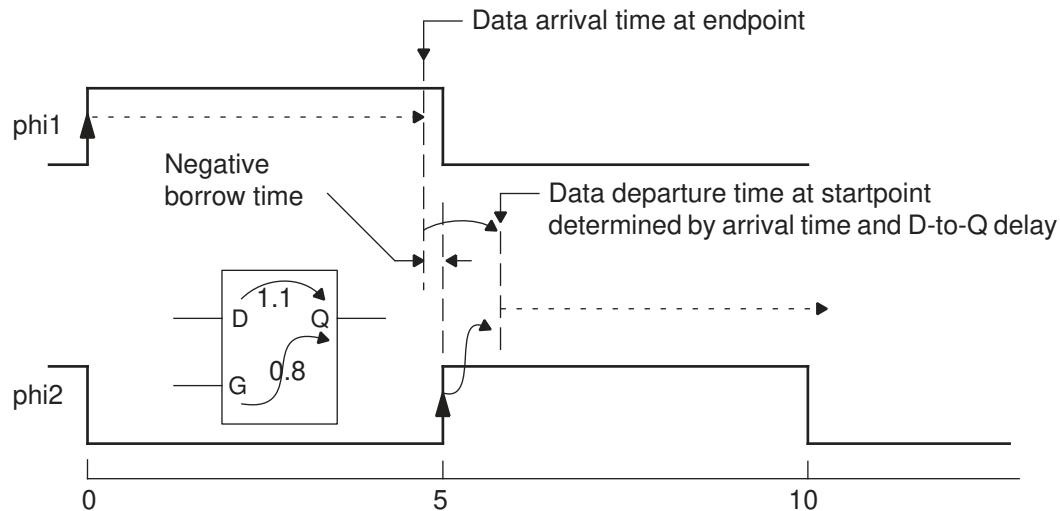


The `report_timing` command reports the amount of time borrowing and uncertainty and CRPR adjustments as follows:

```
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 219](#). 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 219 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 `max_time_borrow` attribute, which is specified by 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.

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 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 41 Commands and variables for advanced latch analysis

Command or variable	Usage
<code>timing_enable_through_paths</code>	Enables advanced latch analysis

Table 41 Commands and variables for advanced latch analysis (Continued)

Command or variable	Usage
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 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](#)
- [Calculation of 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 paths that do not pass through any loop-breaker latch.

Specifying Loop-Breaker Latches

To manually specify loop-breaker latches, use this command:

```
set_latch_loop_breaker -pin pin_list
```

You can 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:

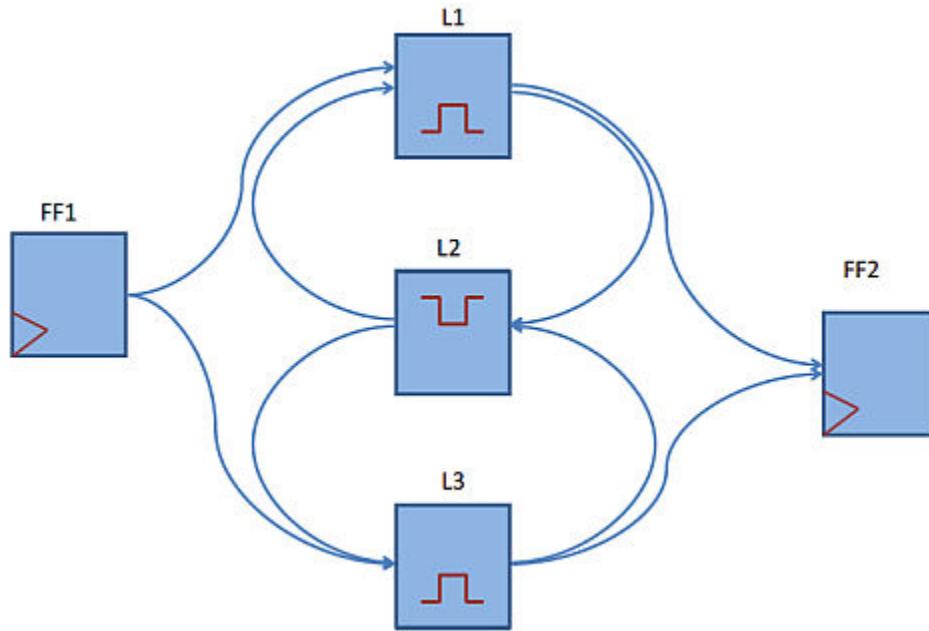
```
get_pins -hierarchical * -filter "@is_latch_loop_breaker"
```

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 the following example, 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 220 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 28](#) 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 28 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 required number. Set a higher number to 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.

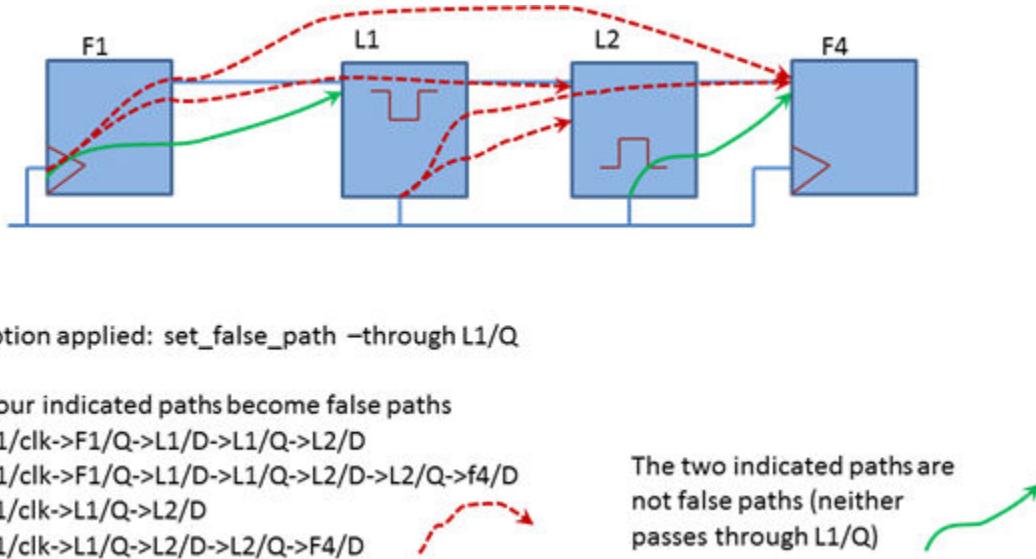
To learn how the tool applies timing exceptions when running advanced latch analysis, see

- 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.

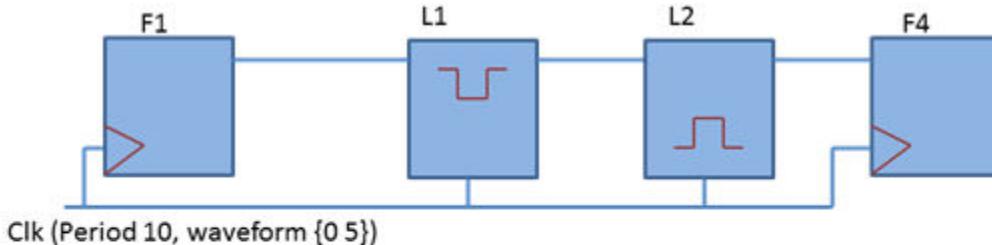
Figure 221 False path exceptions on throughpaths



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 222 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

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:

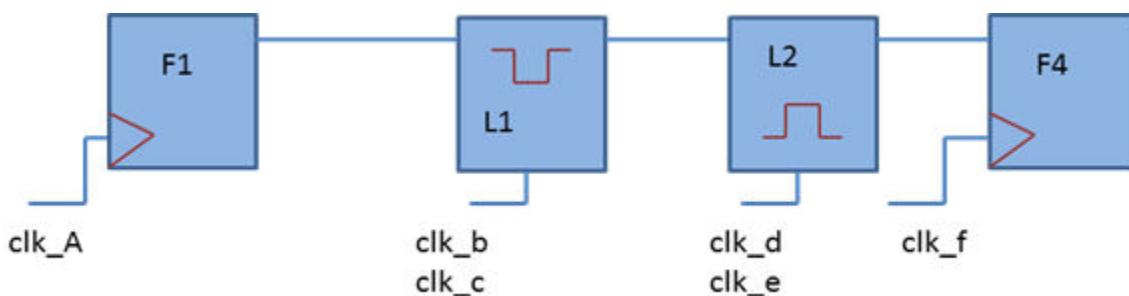
You can use clock-to-clock false paths for hold checks because hold checks are not applied on throughpaths.

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 following circuit example, potentially distinct throughpaths can be created. Exclusive clock groups are honored by preventing some of the potential throughpaths.

Figure 223 Circuit with multiple clocks at latches



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.

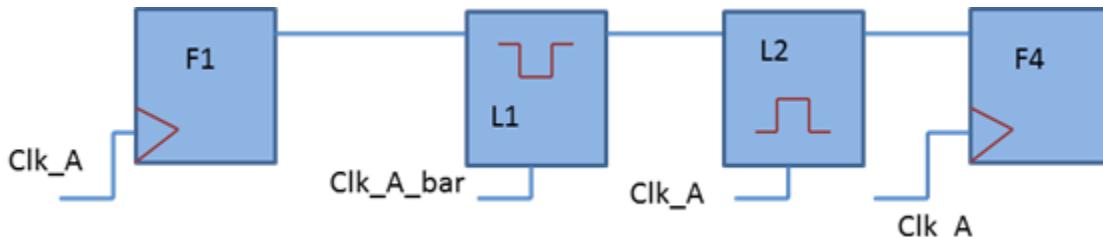
Specification of Exceptions on Throughpaths

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 following circuit, these 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 224 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

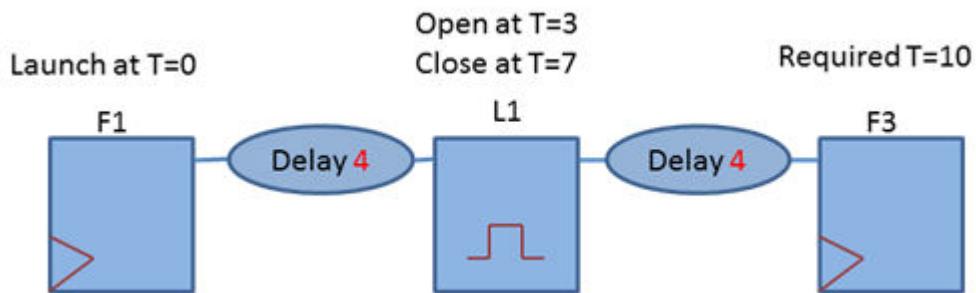
In circuits with latches, the transitive slack provides an estimate of the needed timing improvement for paths through a certain pin. To find the critical path that determines the transitive slack at a specified pin, use this command:

```
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.

Figure 225 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 29 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
<hr/>		
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
<hr/>		
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 r
library setup time	0.00	10.00
data required time		10.00
<hr/>		
slack (MET)		1.00
<hr/>		
normalization delay		10.00
normalized slack		0.10

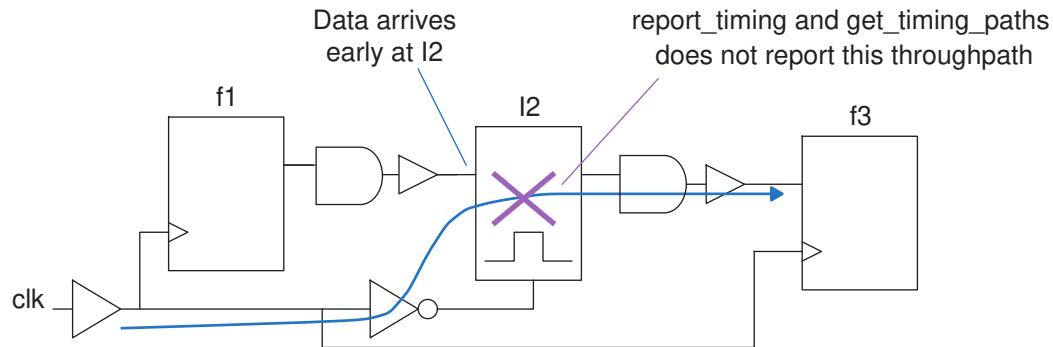
The last section of the report describes the normalized slack data. For more information about normalized slack, see [Normalized Slack Analysis](#).

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`.

This is an example of filtering a path that arrives early.

Figure 226 Filtering throughpaths with the timing_report_skip_early_paths_at_intermediate_latches variable



Reporting Through Loop-Breaker Latches

The 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 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. The following example shows reporting through loop-breaker latches.

Figure 227 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

Calculation of the Worst and Total Negative Slack

The following pin attributes return the local slack in a latch-based path:

- `max_rise_local_slack`
- `max_fall_local_slack`

These attributes are defined on the D pins of latches and other timing endpoints.

The “local” slack (for example, `max_fall_local_slack`) is the timing slack considering the data arrival at the local pin and setup constraints at the local pin, without considering constraints downstream from the pin of a transparent latch. If there are no constraints at the pin, the attribute is undefined. The “normal” slack attributes (for example, `max_fall_slack` and `max_slack`) consider the constraints downstream from the pin of a transparent latch.

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.

The tool calculates the total negative slack (TNS) using the following equation:

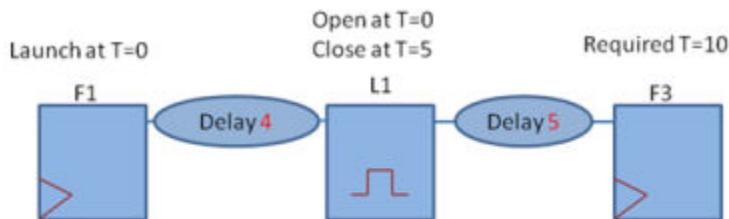
TNS = For all timing endpoints, summation of

`minimum(0, minimum(max_rise_local_slack, 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 unless it is a recovered path (see [Finding Recovered Paths](#)) or a path that contains latches identified as a loop breaker or a path breaker (see [Reporting Latch Loop Groups](#)). The following example shows the normalized slack for three paths in a circuit.

Figure 228 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 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. To find recovered paths,

query the `is_recovered` timing path attribute. The attribute is set to `true` for a path that was recovered at an intermediate latch.

18

Constraint Consistency

The constraint consistency capabilities are designed for the analysis and debug of full-chip or block-level timing constraints. Design analysis, based on rule checking, identifies many types of potential constraint problems. This includes constraints that are missing, invalid, unnecessary, inefficient, or conflicting. Constraint consistency provides a very comprehensive and intuitive graphical user interface (GUI) where further debugging can be performed if needed. You can run analysis with a broad range of built-in rules or define your own consistency checks to further expand the scope of what is to be checked.

To learn more about constraint consistency, see

- [Constraint Consistency Overview](#)
- [Design Consistency Checking](#)
- [Hierarchical Consistency Checking](#)
- [Correlation Consistency Checking](#)
- [Additional Analysis Features](#)
- [Graphical User Interface](#)
- [Tutorial](#)

Constraint Consistency Overview

To learn more about the constraint consistency capabilities, see

- [Constraints in the Design Flow](#)
- [Debugging Features](#)
- [Analysis and Debugging Methodology](#)
- [Starting a Constraint Consistency Session](#)
- [Supported Constraints](#)
- [Reporting Constraint Acceptance](#)
- [Exception Order of Precedence](#)

- [Analyzing a Violation](#)
 - [Suppressing Violations](#)
 - [report_constraint_analysis Command](#)
 - [Using Attributes](#)
 - [Customizing Rules and Reports](#)
 - [Creating and Using User-Defined Rules](#)
 - [Rule-Related Commands](#)
 - [Reading Designs With Incomplete or Mismatched Netlists](#)
-

Constraints in the Design Flow

Correct and complete constraints are essential for timing-driven design implementation steps such as logic synthesis, placement, and routing. Constraints describe the operating environment and other information essential for valid static timing analysis, whether the analysis is stand alone or embedded in an optimization tool. Timing constraints are typically specified as Synopsys Design Constraints (SDC) format files or as more general Tcl scripts. Incorrect or incomplete constraints can lead to wasted design iterations or even silicon failure. Designers can minimize risk and improve productivity by finding and fixing constraint problems as early as possible.

Constraint consistency analyzes constraints at the gate level to determine where and how clock signals, data signals, and constants propagate through the design. Constraints are often changed at various steps. For example, pre-layout constraints can be modified to obtain constraints for placement, clock tree synthesis, routing, and chip assembly. Whenever the constraints are created or modified, they should be analyzed for correctness and consistency.

Constraint consistency analyzes a design and its associated constraints for correctness, completeness, and consistency. You read in and link a top-level or block-level design in Verilog format and apply the constraints using the same SDC commands as you would in the Design Compiler or PrimeTime tools. Constraint consistency applies the SDC constraints and ignores any tool-specific, non-SDC commands in the constraint script. After you read, link, and constrain the design, you can apply one or more of the constraint consistency commands to analyze the design. The first step is to run the `analyze_design` command, which checks the design and its constraints and generates a summary report that you view in a violation browser. The violation browser is a GUI tool that makes it easy to view and examine the causes of potential errors in the design and its constraints. From the summary list, you can click a particular violation to get more detailed information about that violation. From the detailed violation report, you can click links to find the corresponding constraint command in the SDC script, to view the relevant part of the

schematic, or to get suggestions for fixing the violation. The following types of violations are detected:

- Incorrect capacitance values
- Incorrectly specified clocks
- Incorrectly specified generated clocks
- Incorrectly specified clock latency and transition times
- Incorrect or missing driving cells
- Incorrectly specified timing exceptions
- Incorrectly specified input and output delays
- Netlist problems such as conflicting drivers
- Conflicting case analysis values

To get more detailed information about the source of a problem, additional commands are available: `analyze_paths`, `analyze_clock_networks`, `analyze_unclocked_pins`, and `report_case_details`. These commands can help you fix problems such as blocked timing paths and missing clock definitions. The `report_analysis_coverage`, `report_exceptions`, and `compare_block_to_top` commands can help you debug multi-scenario constraints, multiple timing exceptions, and conflicting constraints in hierarchical designs.

Debugging Features

Constraint consistency has numerous unique capabilities. Some of the important features include:

- A set of rules that covers a broad set of potential problems with constraints
- Support for user-defined rules
- Ability to analyze multiple scenarios, such as operating modes, in a single process
- Capacity and performance to efficiently analyze designs containing many millions of cell instances
- Powerful diagnostics and detailed analysis commands that help you find the root cause of reported problems and understand the impact of constraints
- A rich set of collection commands and object attributes that are provided to enable debugging and custom scripting

- Rule checkers that are fully compliant with SDC version 2.0 to identify syntax errors for invalid objects
- An interactive, Tcl-based shell that supports most SDC commands as well as loops, variables, conditional constructs, and collections
- GUI output that is easy to understand and interpret. Most constraint references include detailed information such as the source file name and line number where the constraint was defined
- Excellent consistency with other Synopsys implementation tools. This includes constraint interpretation, propagation of clocks, signals and constants, user interface, and more

Analysis and Debugging Methodology

Constraint consistency accepts the following input data:

- Verilog netlists
- Cell libraries in Synopsys .db format
- Design constraints in SDC or PrimeTime Tcl format
- Tool control scripts in Tcl format

Constraint consistency produces the following output data:

- Text reports
- Query commands and object attributes for interactive debugging, custom reporting, or conditional scripting

The steps in a typical constraint consistency flow are as follows:

1. Start constraint consistency by entering the following command at the Linux prompt:

```
pt_shell -constraints
```

See “[Starting a Session](#)” for more information.

2. Read in your design using the following commands:

```
set_app_var search_path ...  
set_app_var link_path ...  
read_verilog ...
```

3. Link the design using the following commands:

```
current_design design_name  
link_design
```

4. Apply constraints for each scenario.

For Tcl scripts, use the `source constraint_file` command.

For SDC commands, use the `read_sdc constraint_file` command.

See “[Design Consistency Checking](#)” for more information about supported constraints.

5. Analyze your design using the following command:

`analyze_design`

See “[Design Consistency Checking](#)” for more information.

6. Diagnose problems.

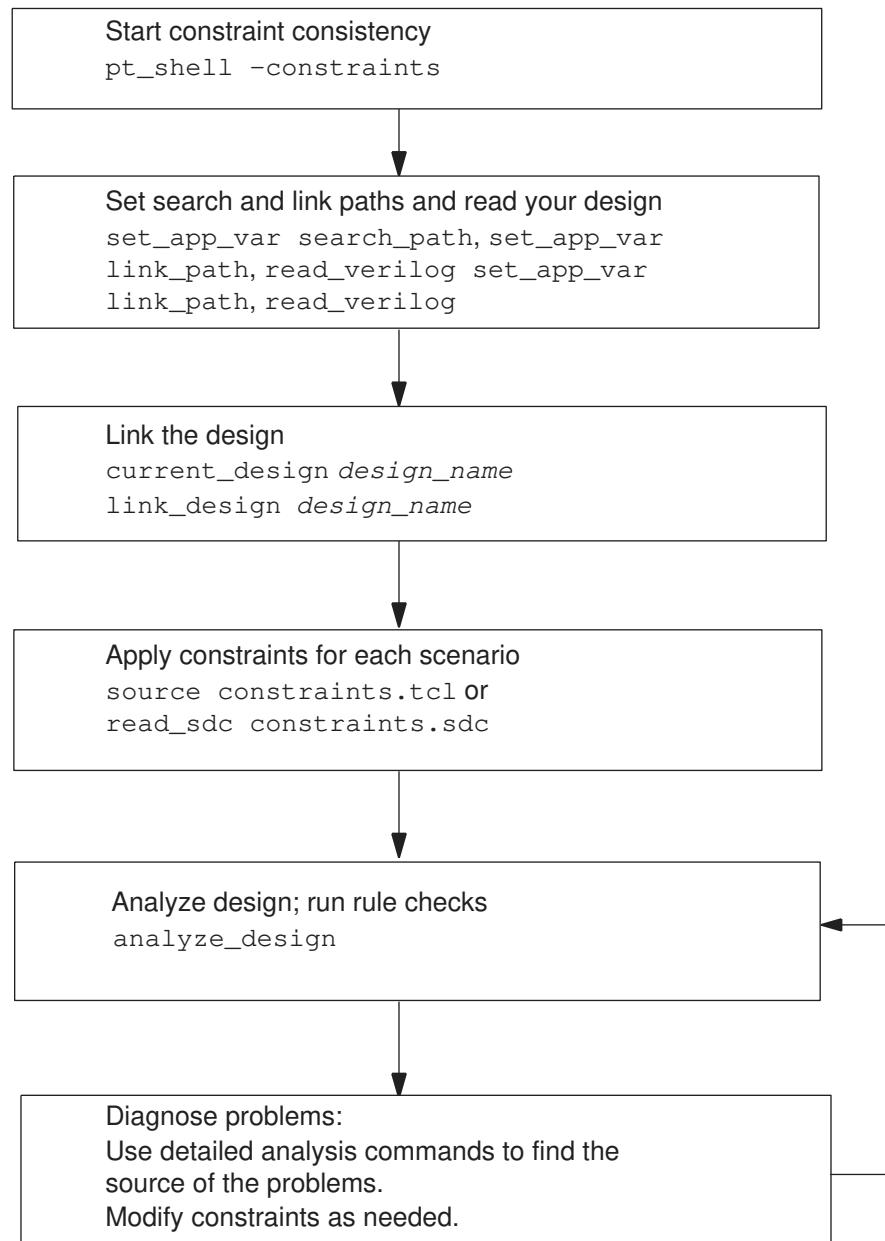
You can diagnose issues with a set of dedicated commands.

See “[Additional Analysis Features](#)” for more information.

7. Repeat the analysis and diagnosis steps as needed.

[Figure 229](#) shows a typical flow for constraint consistency.

Figure 229 Typical Constraint Consistency Flow



Starting a Constraint Consistency Session

Constraint consistency is GUI-driven and runs under the Linux operating system. To learn more about starting a constraint consistency session, see

- [Starting a Session](#)
- [License](#)
- [Setup Files](#)
- [Command Log File](#)

Starting a Session

To start constraint consistency, use the `-constraints` option with `pt_shell`:

```
% pt_shell -constraints
```

A PrimeTime SI license is automatically checked out, the GUI starts, and the `ptc_shell` prompt appears:

```
PrimeTime (R)
Version ... for linux64 -- ...
Copyright (c) 1988-2017 by Synopsys, Inc
...
Running PrimeTime constraint consistency
ptc_shell>
```

If constraint consistency fails to start, verify that

- Constraint consistency is correctly installed
- The PrimeTime installation path is included in your path definition
- The Synopsys license server is running
- Your Synopsys license key file is available and current, and that it includes a PrimeTime SI license

If you need assistance, ask your system administrator or consult the installation and licensing documentation.

To end a constraint consistency session, enter the `quit` or `exit` command at the prompt:

```
ptc_shell> exit
Maximum memory usage for this session: 33.34 MB
CPU usage for this session: 2 seconds
Diagnostics summary: 2 errors
Thank you for using ptc_shell!
%
```

License

You need a PrimeTime SI license to start `ptc_shell`. Constraint consistency automatically checks out a license when you start a session. When you exit the session the license is automatically checked in, allowing others at your site to use it. In addition, constraint

consistency requires that you have a PrimeTime license available on the same server. The PrimeTime license is not checked out, but the license must be present for constraint consistency to start.

Setup Files

Constraint consistency runs the commands contained in a set of setup files each time you start a session. You can put commands into a setup file to set variables, to specify the design environment, and to select your preferred working options. The name of each setup file is `synopsys_gca.setup`. The presence of the file is checked in the following directories:

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

If more than one of these directories contains a `.synopsys_gca.setup` file, the files are executed in the order shown previously: first in the Synopsys setup directory, then in your home directory, and finally in the current working directory. Typically, the file in the Synopsys setup directory contains setup information for all users at your site; the one in your home directory sets your personal preferred working configuration; and the one in your current working directory sets the environment for the current project. To suppress execution of any `.synopsys_gca.setup` files, use the `-no_init` option when you start the session.

Command Log File

Constraint consistency saves the session history in a file called the command log file. This file contains all the commands executed during the session and serves as a record of your work. You can repeat the whole session later by running the file as a script using the `source` command. Constraint consistency creates the log file in the current working directory and names it `ptc_shell_command.log`. Any existing log file with the same name is overwritten. Before you start a new session, be sure to rename any log file that you want to keep. You can specify a different name for the command log file by setting the `sh_command_log_file` variable in your setup file. You cannot change this variable during a working session.

Supported Constraints

Constraint consistency supports all constraints in SDC format and Tcl format.

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

For more detailed information about Tcl, see the *Using Tcl With Synopsys Tools* documentation.

Constraint consistency supports the `source` command to read in constraints in Tcl scripting format. [Example 30](#) shows a Tcl constraint script fragment. You need to use the `source` command to load these constraints, as shown in [Example 31](#).

Example 30 *Tcl Constraints File constraints.tcl*

```
...
foreach_in_collection i [all_clocks] {
    echo [format "Clock : '%s'" [get_attribute $i full_name]]
    set_clock_uncertainty 1.5 $i
}
...
...
```

Example 31 *Using the Source Command to Load Tcl Scripted Constraints*

```
set_app_var sh_continue_on_error true

set design_dir ./

set MyDesign b_top
current_design $MyDesign
link_design

create_scenario system
create_scenario test

current_scenario system
source constraints.tcl
...
```

Constraint consistency supports the `read_sdc` command to read constraints in SDC format.

Constraint consistency supports multiple scenarios. To create constraints for each scenario, first define the scenarios:

```
create_scenario system
create_scenario myscenario_1
create_scenario myscenario_2
```

You can use the script in [Example 32](#) to apply both common and scenario-specific Tcl constraints.

Example 32 *Tcl Script for Loading Common and Unique Constraints*

```
set all_scenarios {system myscenario_1 myscenario_2}
foreach scenario $all_scenarios {
    create_scenario ${scenario}
    source ./scripts/common.tcl
```

```
source ./scripts/${scenario}_constraints.tcl
}
```

Reporting Constraint Acceptance

You can use the `statistics` keyword with the `-include` option of the `report_constraint_analysis` command to report the constraints accepted for analysis. By using the `statistics` keyword, you can quickly see if the constraints are correctly sourced. [Example 33](#) shows the statistics report.

Example 33 Statistics Report

```
ptc_shell> report_constraint_analysis -include statistics
*****
```

```
Report : report constraint analysis
         -include {statistics }
         -style {full}
         ...
Date   : ...
*****
Constraint Processing Statistics

Constraint          Accepted    Rejected    Total
-----
create_clock          7           0           7
set_clock_uncertainty 1           0           1
set_input_delay        6           0           6
set_output_delay       6           0           6
1
```

Exception Order of Precedence

If different timing exception commands are in conflict for a particular path, the exception 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 constraint consistency applies the exception precedence rules independently on each path (not each command). For example, suppose that you use the following commands:

```
ptc_shell> set_max_delay -from A 5.1
ptc_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` command settings
- `set_min_delay` and `set_max_delay` command settings
- `set_multicycle_path -setup` and `-hold` command and option 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,

```
ptc_shell> set_max_delay 12 -from [get_clocks CLK1]
ptc_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` and `-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`

Analyzing a Violation

You can debug constraint problems by using the GUI. In the violation browser window, select the violation you want to analyze, and view the details of the violation that are shown in the information pane. [Figure 230](#) and [Figure 231](#) show the violation tree and the violation details in the information pane.

Chapter 18: Constraint Consistency

Constraint Consistency Overview

Figure 230 Debugging Example - Violation Browser Provides Access to Violation Details

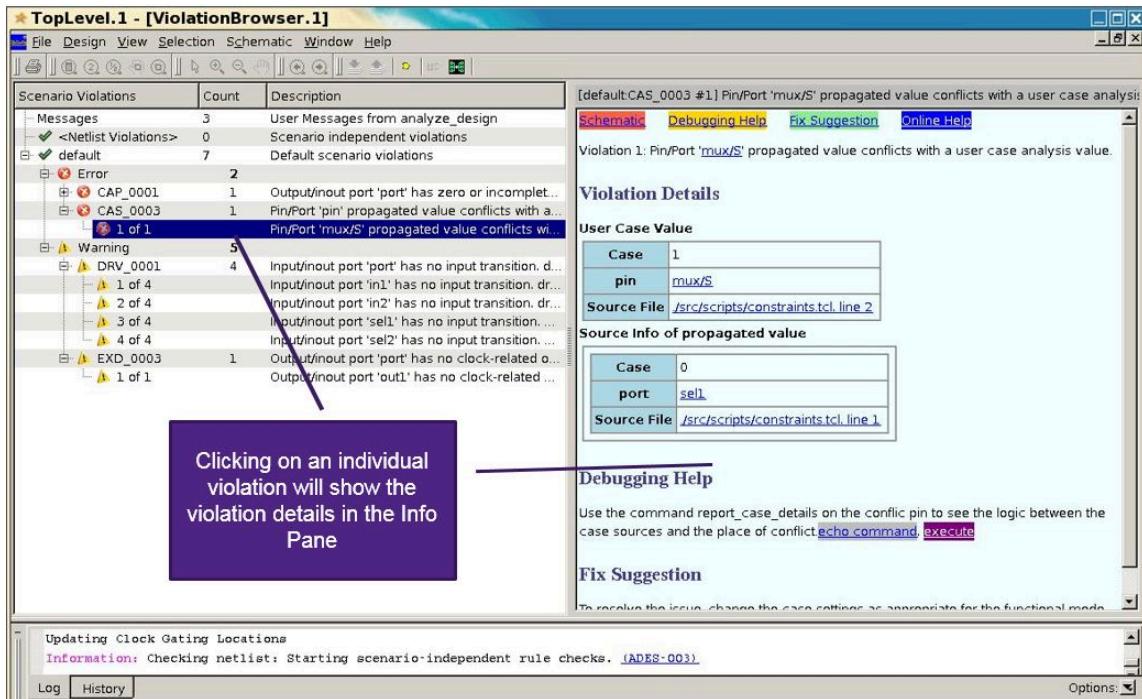


Figure 231 labels the various hyperlinks that you can click to access the SDC file and the relevant exception or command.

Figure 231 Debugging Example - Information Pane Links to Violation Details

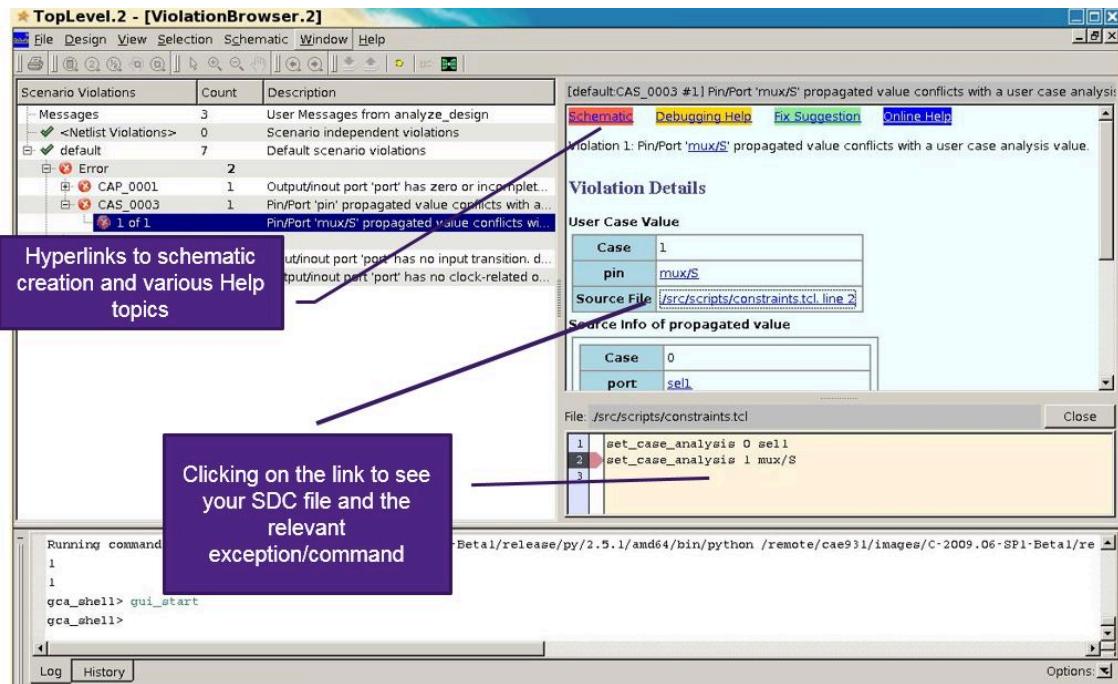
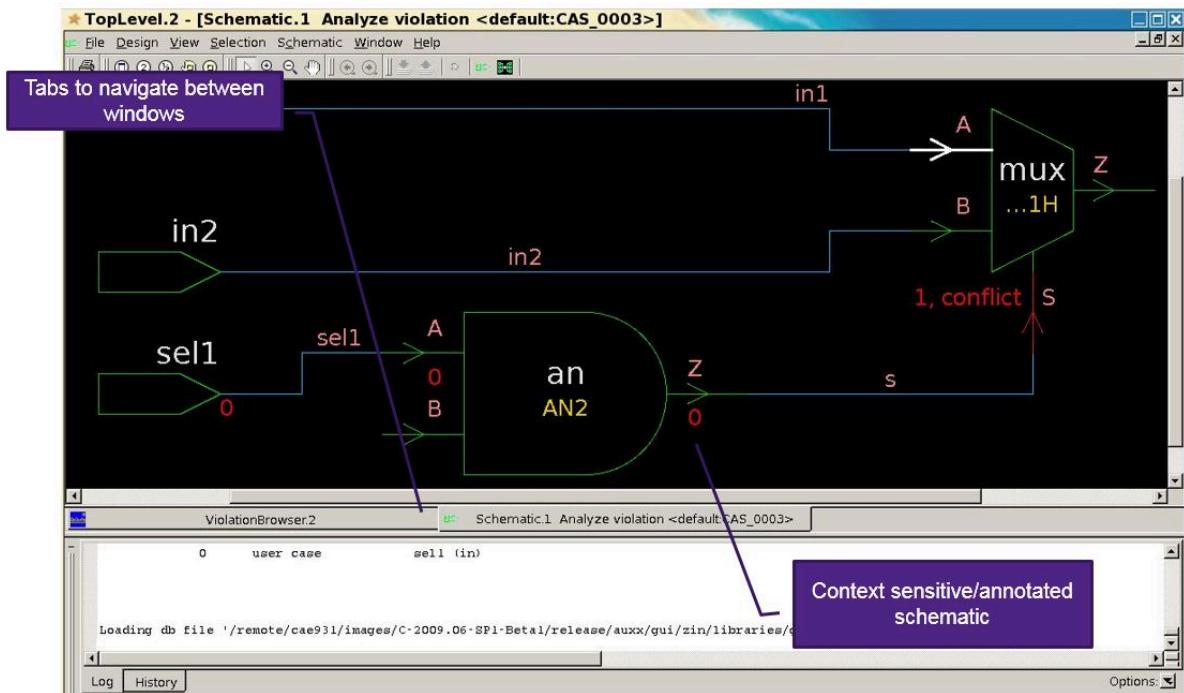


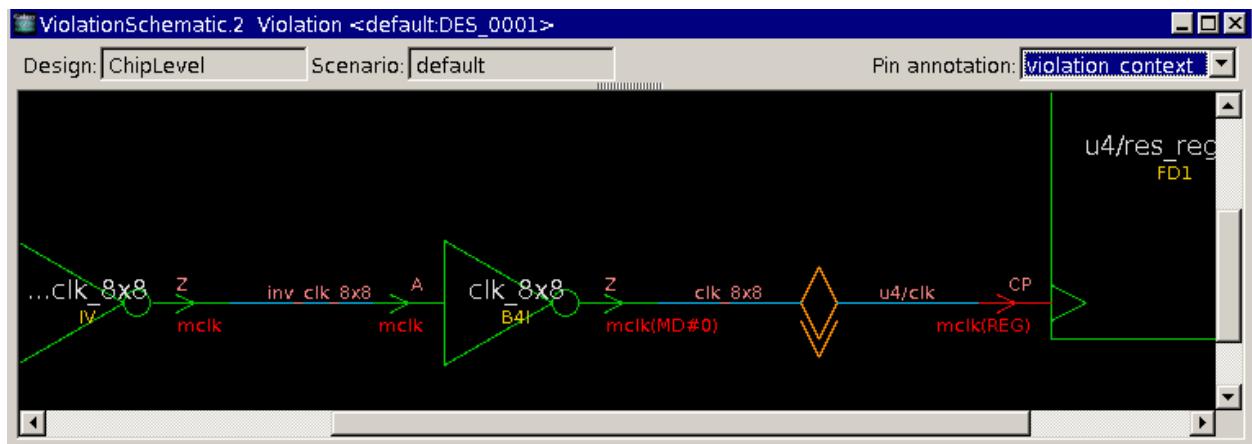
Figure 232 shows the relevant schematic area of the problem constraint. The schematic is accessed from the schematic link in the information pane.

Figure 232 Schematic Viewer Shows Violation Circuit Path



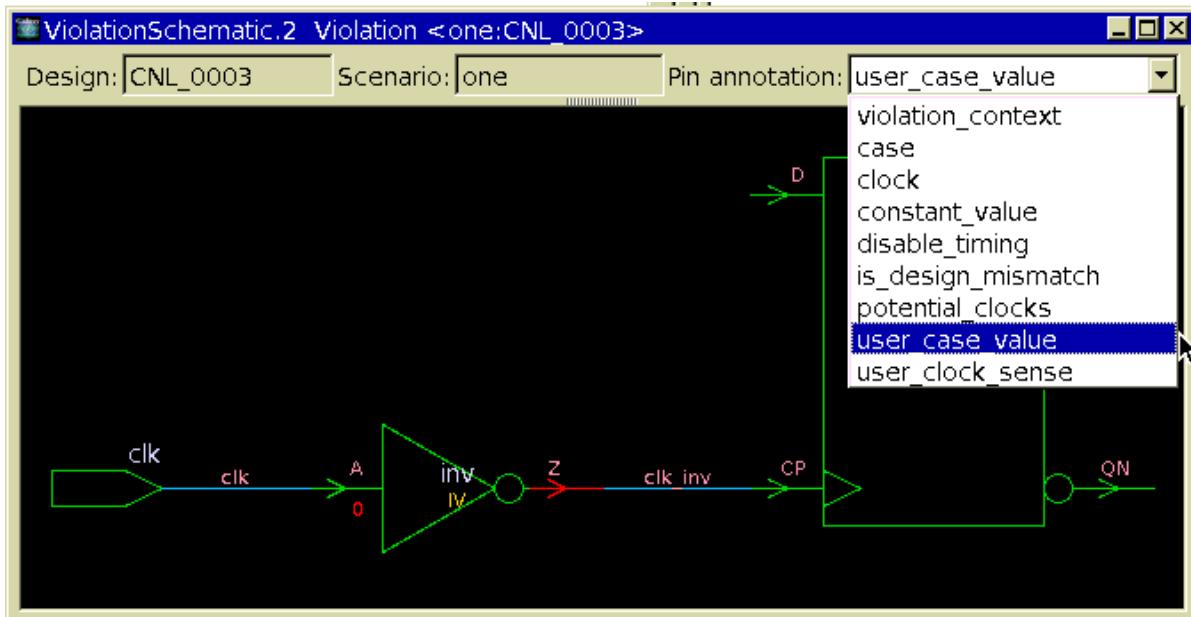
To help debug constraint problems, you can view clock and case annotations in the schematic view using the Pin annotation list, as shown in Figure 233. The information related to a violation is displayed by default.

Figure 233 Schematic Viewer With Full Violation Context



The Pin annotation list is expanded to provide a visualization of any of the annotations in the schematic, as shown in Figure 234.

Figure 234 Schematic Viewer With Complete Pin Annotation List



If further debugging is required, use the Debugging Help link, the Fix Suggestion link, or investigate the constraint in the SDC file by selecting the source file link in the Violation Details section of the information pane. In addition to those sources of help, you can consult the rule reference for this violation in the online Help.

Suppressing Violations

Constraint consistency allows you to suppress rule violations. You can either disable a rule completely or suppress a specific instance of a rule violation, also known as waiving a violation. For example, you might not want to check if your design violates the capacitance-related rules; in this case you would disable the CAP_xxxx rules. As another example, you might want to analyze your design for capacitance rule violations on all but one output port; in this case you would waive violations of the CAP_xxxx rules on the specific output port.

Constraint consistency includes commands for creating, reporting, removing, and writing out waivers. The GUI allows you to easily create and remove waivers.

To learn more about violation suppression, see

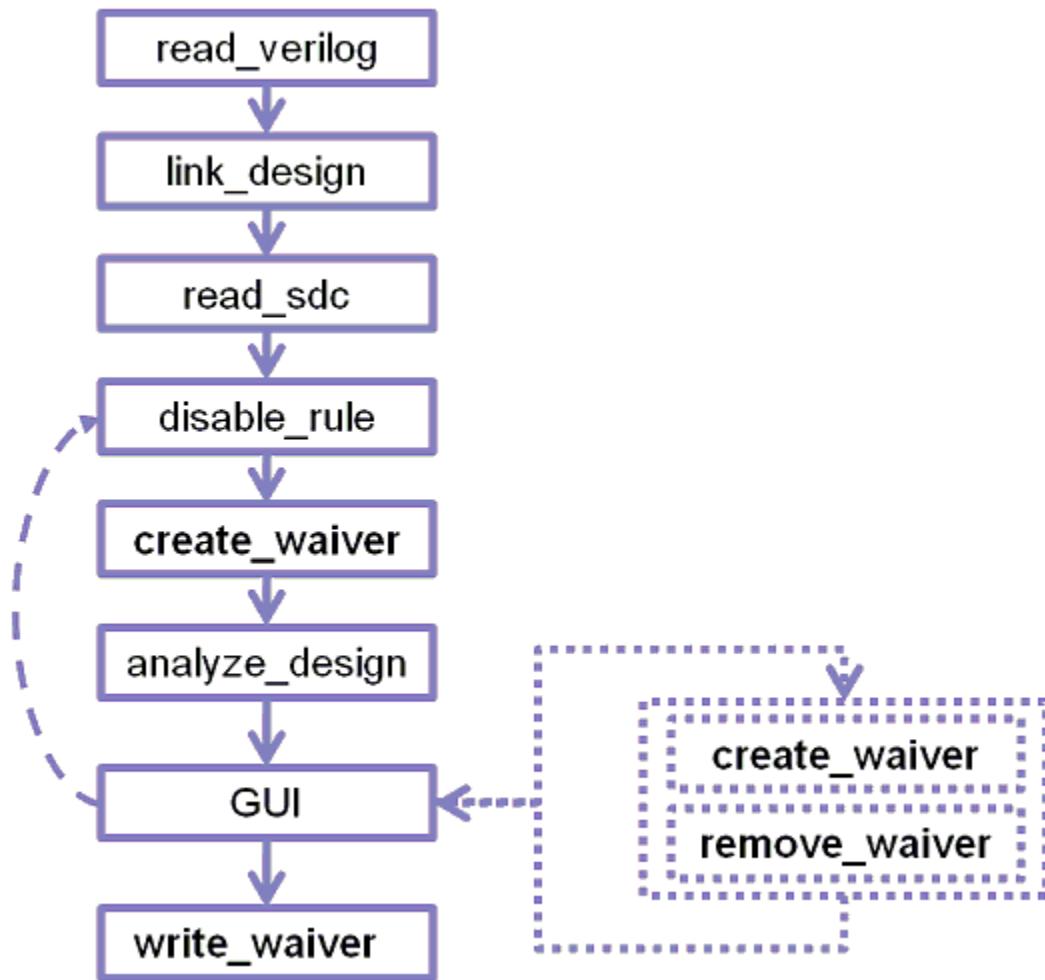
- [Overview of Violation Suppression Flow](#)
- [Disabling Rules](#)
- [Waiving Specific Violations of a Rule](#)

- [create_waiver Command](#)
- [Creating Instance-Level Waivers](#)
- [Usage Guidelines](#)
- [Modifying Waivers](#)
- [Reporting Waivers](#)
- [Removing Waivers](#)
- [Writing Waivers to a File](#)

Overview of Violation Suppression Flow

The violation suppression usage model is shown in [Figure 235](#).

Figure 235 Violation Suppression Usage Model



As shown in [Figure 235](#), you can suppress violations either before or after running the `analyze_design` command. If you suppress a violation after an analysis, the violation browser shows the violation marked it with a special icon to indicate that it has been suppressed. If you suppress a violation before analysis, a rule that has been disabled completely is omitted entirely from the report, whereas a rule violation that has been waived for certain instances is still reported, but shown with the special icon.

If a violation is already listed in the violation browser, its status is immediately updated when you waive it. This allows you to see the impact of the violation suppression on the current set of violations. The next time you run the `analyze_design` command, the suppressed violations are listed with a special icon in the violation browser.

Disabling Rules

If you do not want to check for violations of a rule, you can disable the rule by using the violation browser, the Waiver Configuration dialog box, or the `disable_rule` command.

To disable a rule using the violation browser,

1. Right-click the rule you want to disable.
2. Choose Disable rule.

When you choose Disable rule, constraint consistency automatically generates the `disable_rule` command to disable the selected rule. The disabled rule has a disabled icon next to it in the violation browser.

To disable a rule using the Waiver Configuration dialog box,

1. Choose Design > Waiver Configuration.

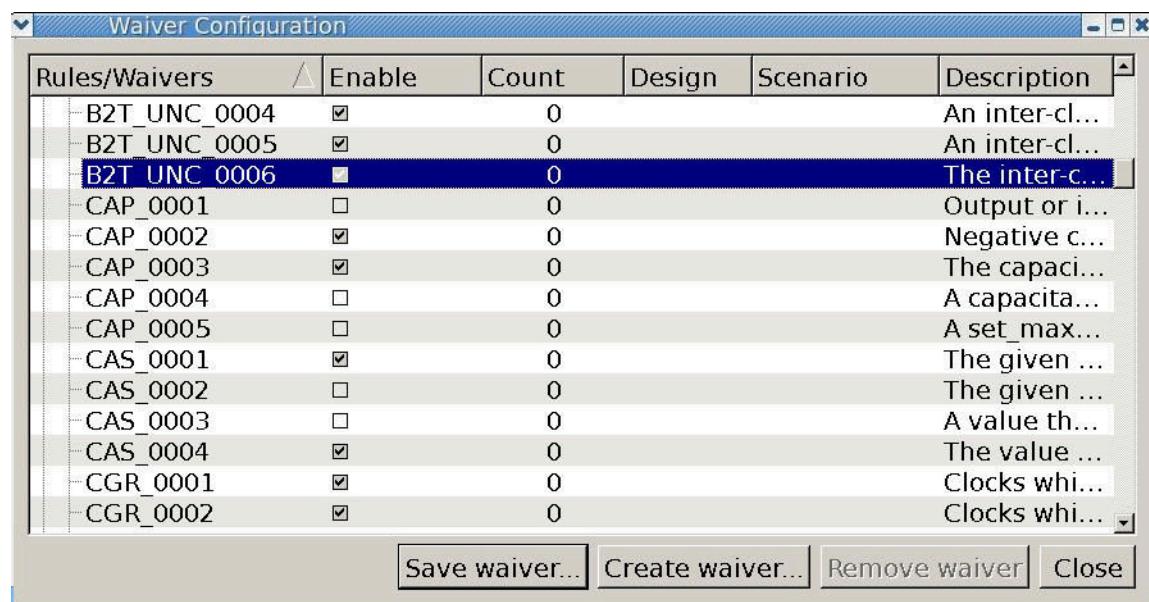
The Waiver Configuration dialog box appears, as shown in [Figure 236](#).

2. Uncheck the box next to the rule you want to disable.

The rule is not checked or reported the next time you run the `analyze_design` command.

For information about disabling rules and reporting disabled rules, see “[Suppressing Violations](#)” and “[Reporting Waivers](#)”.

Figure 236 Waiver Configuration Dialog Box



Waiving Specific Violations of a Rule

You can waive specific violations of a rule by using the violation browser, the Waiver Configuration dialog box, or the `create_waiver` command.

To waive a specific violation of a rule with the violation browser,

1. Right-click the violation.

You can select multiple violations. However, the “Undo last waiver” selection cannot undo multiple waivers; it will undo only the last waiver in the group.

2. Choose Waive violation.

Constraint consistency generates a `create_waiver` command with complete arguments filled in to suppress the selected violation. The waived violation has a “waived” icon next to it in the violation browser, as shown in [Figure 237](#).

The waived violation and the icon disappear from the violation browser after you run the `analyze_design` command.

Figure 237 Waived Violation in the Violation Browser

Scenario Violation	Count	Waived	Description
<Global Violation>	1	0	Scenario independent violations
Error	1	0	
default	13	1	Default scenario violations
Error	5	1	
CAP_0001	2	1	Output/inout port 'port' has zero or incomplete capacitance values.
1 of 2	1	1	Output/inout port 'SUM' has zero or incomplete capacitance values.
2 of 2	1	0	Output/inout port 'CARRY' has zero or incomplete capacitance values.
EXD 0012	3	0	The input delay at 'object_type' 'object' has zero window for min and max values.
1 of 3	1	0	The input delay at 'port' 'A' has zero window for min and max values.
2 of 3	1	0	The input delay at 'port' 'B' has zero window for min and max values.
3 of 3	1	0	The input delay at 'port' 'PREV_CARRY' has zero window for min and max values.
Warning	8	0	
new	14	0	User defined scenario violations
Error	5	0	
Warning	9	0	

To waive a specific violation of a rule with the Waiver Configuration dialog box,

1. Choose Design > Waiver Configuration.

The Waiver Configuration dialog box appears.

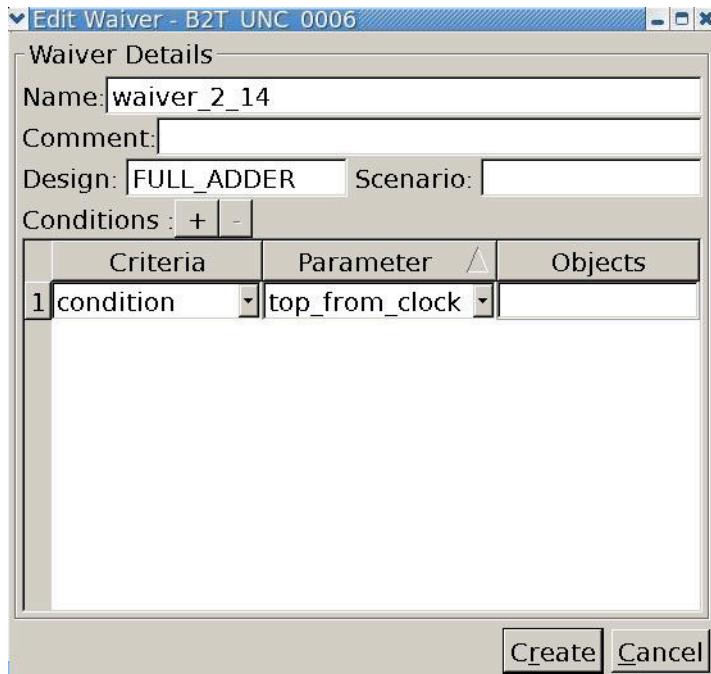
2. Click “Create waiver.”

The Edit Waiver dialog box appears. [Figure 238](#) shows the editor window.

3. Enter the waiver specification.

To disable a specific violation of a rule using the `create_waiver` command, see the “[create_waiver Command](#)”.

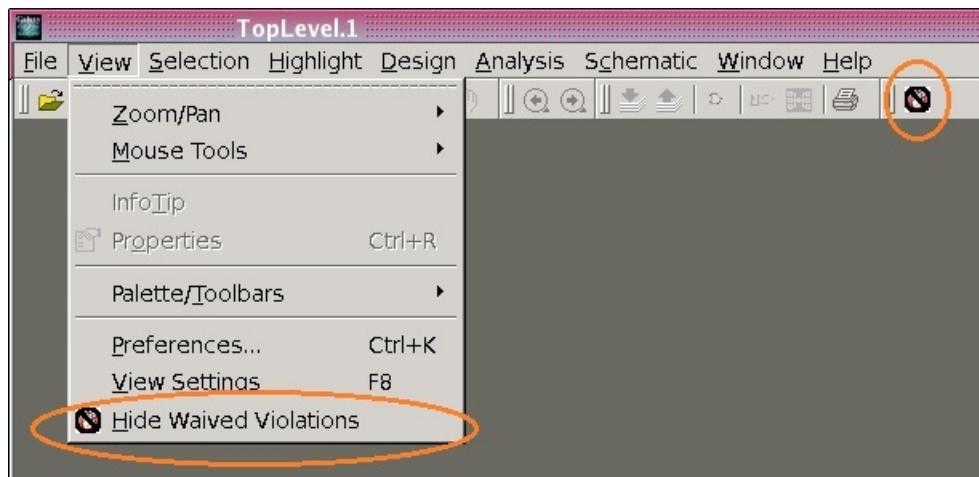
Figure 238 Edit Waiver Dialog Box



For more information about disabling rules, see “[Suppressing Violations](#)”.

When you set the `hide_waived_violations` variable to false, the Hide Waived Violations command is toggled on the View menu, as shown in [Figure 239](#).

Figure 239 Hide Waived Violations



create_waiver Command

You can create waivers by using the `create_waiver` command.

In the `create_waiver` command, you can specify multiple `-condition` and `-not_condition` options. A violation is suppressed only if all of the `-condition` and `-not_condition` options are satisfied. You can specify collections of netlist objects with `-condition` options.

The argument to the `-condition` or `-not_condition` option must be a list consisting of two items: the name of the rule parameter, and a collection of netlist objects or constraints. To create the collection, you need to use a syntax such as `[get_clocks CLK]`.

When working with multiple designs, use the `-design` and `-scenario` options to assign waivers to specific designs. See the waivers in the following examples:

Example 1: Waiver Using the `-condition` Option

To waive violations of the `CLK_0026` rule on clocks that start with `CLK`, use the following command:

```
ptc_shell> create_waiver -rule CLK_0026 -condition [list "clock" \
[get_clocks CLK*]]
```

Example 2: Waiver Using the `-not_condition` Option

To waive violations of the `CLK_0026` rule on all clocks except `CLK3`, use the following command:

```
ptc_shell> create_waiver -rule CLK_0026 \
-not_condition [list "clock" [get_clocks CLK3]]
```

Example 3: Waiver Using Multiple Parameters

The `CLK_0003` rule checks multiple parameters: a clock and a pin.

Figure 240 Multiple Parameters

<code>CLK_0003</code>	error	Generated clock <code>clock</code> is not expanded, it has no clock reaching its master source <code>pin</code> .
-----------------------	-------	---

To create a waiver for this rule, you can specify conditions for both the clock and the pin. For example, to waive `CLK_0003` violations if the clock is `CLK1` or `CLK2` and the source pin is not `U1/A` and not `U1/B`, use the following command:

```
ptc_shell> create_waiver -rule CLK_0003 \
-condition [list "clock" [get_clocks {CLK1 CLK2}]] \
-not_condition [list "pin" [get_pins U1/A] [get_pins U1/B]]
```

Example 4: Multiple Waivers of the Same Rule

To waive violations of the CLK_0003 rule if the clock is CLK1 and the source pin is U1/A or the clock is CLK2 and the source pin is U1/B, you need to create two separate waivers, as shown in the following example:

```
ptc_shell> create_waiver -rule CLK_0003 \
    -condition [list "clock" [get_clocks CLK1]] \
    -condition [list "pin" [get_pins U1/A]] \
ptc_shell> create_waiver -rule CLK_0003 \
    -condition [list "clock" [get_clocks CLK2]] \
    -condition [list "pin" [get_pins U1/B]]
```

Example 5: Waiver Using an Exception Parameter

To waive violations of the EXC_0001 rule with an exception specified from U1/A to U2/Z and the `rise_or_fall` parameter set to `rise`, use the following command:

```
ptc_shell> create_waiver -rule EXC_0001 \
    -condition [list "exception" [get_exceptions -from U1/A -to U2/Z]] \
    -condition [list "rise_or_fall" "rise"]
```

If the rule on which the waiver is applied is scenario-dependent, the waiver is added to current scenario. If the rule is scenario-independent, the waiver applies to scenario-independent checks.

Netlist objects are interpreted with respect to the current instance. For example, the following two sets of commands produce equivalent waivers:

```
current_instance U1
ptc_shell> create_waiver -name "my_waiver4" -rule CAS_0003 \
    -condition [list "pin" [get_pins U2/U3/A]]

current_instance U1/U2
ptc_shell> create_waiver -name "my_waiver4" -rule CAS_0003 \
    -condition [list "pin" [get_pins U3/A]]
```

The waiver suppresses violations based on violation parameters that you specify in the waiver conditions. Parameter names can be retrieved using the `report_rule -parameters rule_name` command.

For example, to report the parameters available for the CLK_0026 rule, run the following command:

```
ptc_shell> report_rule -parameters CLK_0026
*****
Report : rule
Version: ...
Date   : ...
*****
Rule      Severity  Status     Message
```

```
-----
CLK_0026      warning   enabled   Clock 'clock' is used as data. One or
                                more sources of the clock fans out to a
                                register data pin or to a constrained
                                primary output or inout port.

CLK_0026 parameters:
Name          Value  Types
-----
clock         clock
```

Example 6: Creating a Waiver for a Block-to-Top (B2T) Rule

When you create a waiver for a Block-to-Top rule, you work with two contexts: the TOP design and the BLOCK design. In the condition statement, before you can retrieve an object, you have to define from which context to retrieve. Depending on the violation you are waiving, you might also need to define the parameters in your violation such as "top_object_type" and "block_object_type." The parameters are listed in the B2T rules which are described in the online Help. The waiver needs to apply to a particular design and scenario pair and an instance of the block.

```
ptc_shell> create_waiver -rule B2T_CLK_0012 -condition \
    -condition \
        [list "blk_object" \[current_design BLOCK ; get_pins zGate/A\]] \
    -condition [list "blk_object_type" "pin"] \
    -condition \
        [list "top_object" [current_design TOP ; get_pins b1/zGate/A]] \
    -condition [list "top_object_type" "pin"] \
    -design1 TOP -scenario1 default \
    -design2 BLOCK -scenario2 default \
    -inst2 b1
```

Example 7: Creating a Waiver for an SDC-to-SDC Rule

When you create a waiver for an SDC-to-SDC rule, you work with two contexts: the first SDC definition and the second SDC definition. In the condition statement, before you can retrieve an object, you have to define from which context to retrieve. You also need to define the parameters in your violation. The parameters are described in the SDC-to-SDC rules which are available in the online Help. The waiver needs to apply to a particular design and scenario pair.

```
ptc_shell> create_waiver -rule S2S_DIS_0001 \
    -condition [list "object" \
        [current_design S2S_CLK_0001 ; get_pins ck2_i2/Test_pin]] \
    -condition [list "object_type" "pin"] \
    -condition [list "scenario1" "set2"] \
    -condition [list "scenario2" "set1"] \
    -design1 zDesign -scenario1 set1 \
    -design2 zDesign -scenario2 set2
```

You can waive rules for specific cells or design instances by using the GUI or the command line. All related waiver commands, such as `remove_waiver`, `write_waiver`, and `report_waiver` support this feature.

You can create an instance-level waiver from the hierarchy browser or from the Waiver Configuration dialog box.

Creating Instance-Level Waivers

To create an instance-level waiver from the hierarchy browser,

1. Open the hierarchy browser.

Select Design > Hierarchy Browser.

2. Click the cell instance.

3. Right-click the cell instance to open the menu. See [Figure 241](#).

4. Choose Waive Instance.

The instance waiver icon is displayed in the violation browser next to all the cell instances waived for that rule. See [Figure 242](#).

To create an instance-level waiver from the GUI from the Waiver Configuration dialog box,

1. Open the Waiver Configuration dialog box.

Choose Design > Waiver Configuration.

2. Click Instance Waivers. See [Figure 243](#).

3. Click the Create Waiver button.

The Edit Waiver Instance dialog box appears. See [Figure 244](#).

4. Enter your instance waiver details.

5. Click the Create button.

Figure 241 Using the Hierarchy Browser to Create Instance Waivers

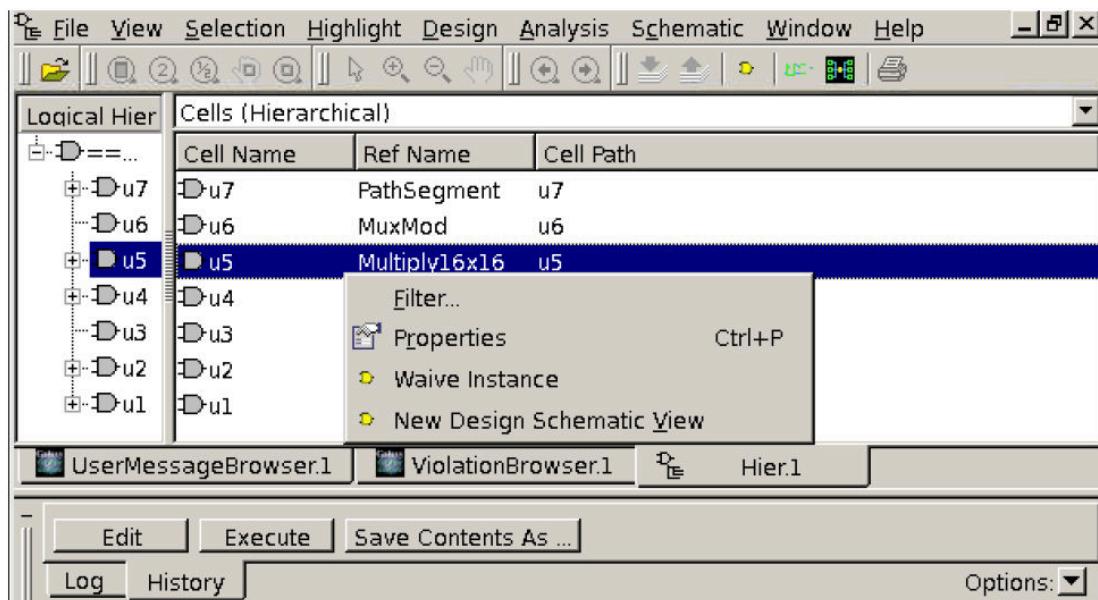


Figure 242 Waiver Icon in Violation Browser

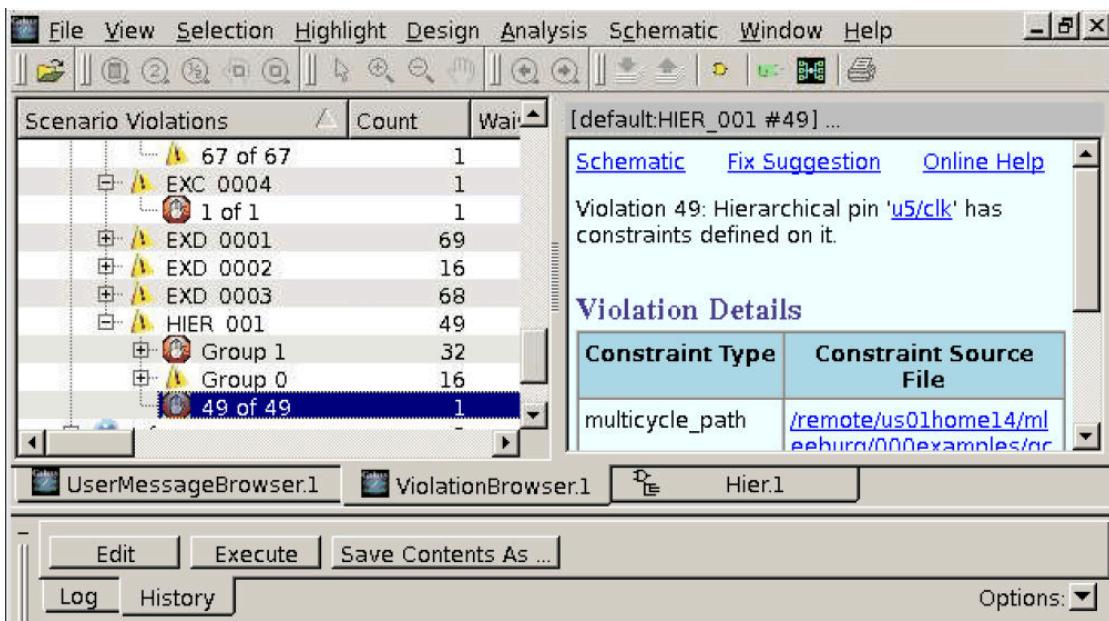


Figure 243 Waiver Configuration Dialog Box Showing Instance Waivers

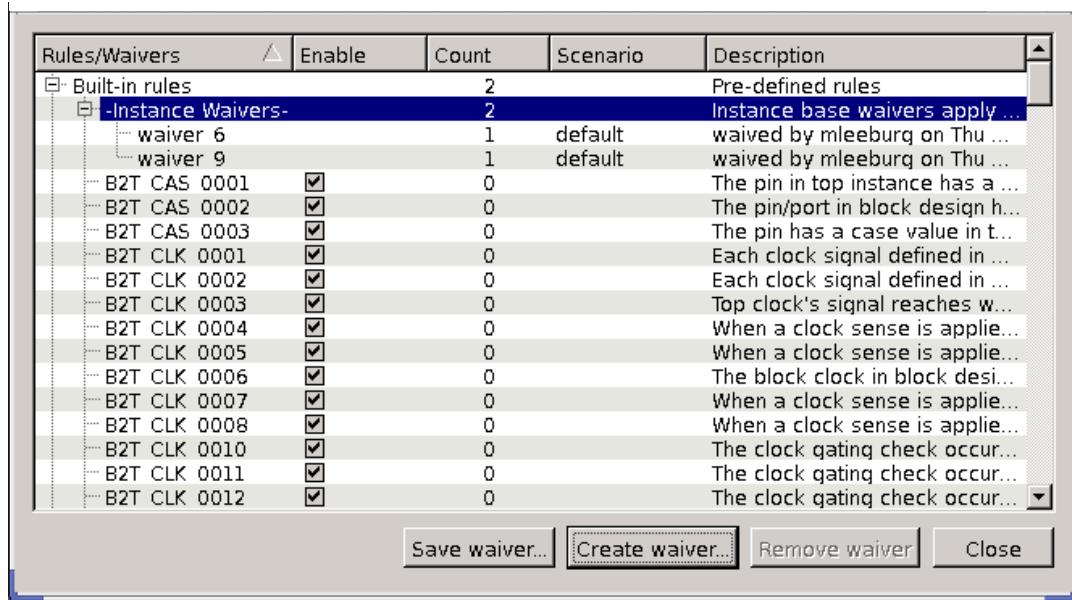
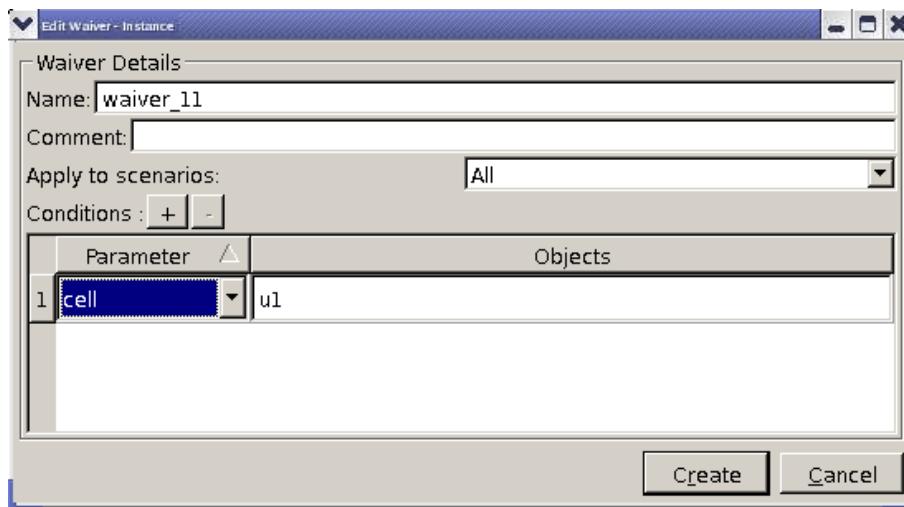


Figure 244 Creating an Instance Waiver From the Edit Waiver Dialog Box



To create an instance-level waiver from the command line, use the `-cells` option with the `create_waiver` command, as shown in [Example 34](#).

In [Example 34](#), `my_waiver_1` is applied to cell U1/U1 and `my_waiver_2` is applied to cells U1/U2 and U1/U3.

Example 34 Waiving Rules for Cells

```
ptc_shell> create_waiver -name my_waiver_1 -cells [get_cell U1/U1]
ptc_shell> create_waiver -name my_waiver_2 -cells [get_cell {U1/U2 U1/U3}]

ptc_shell> report_waiver
*****
Report : report_waiver
Version: ...
Date   : ...
*****
Name      Rule Scenario Condition           Comment
-----
my_waiver_1          -cells {U1/U1}        waived by Jane on ...
my_waiver_2          -cells {U1/U2 U1/U3}    waived by Jane on ...
```

Usage Guidelines

When you create an instance-level waiver, constraint consistency applies it to the current scenario. To apply it across all scenarios of a design, use the `-all_scenarios` option.

Because an instance-level waiver is based on leaf cells or the netlist objects contained in an instance, a violation is waived only if the objects related to the violation are entirely contained within the specified cell or instance. Therefore, some general rules cannot be waived with this type of waiver. Custom rules cannot be waived with instance-level waivers.

Modifying Waivers

Use the following procedure to modify an existing waiver:

1. Open the Waiver Configuration dialog box (choose Design > Waiver Configuration).
2. Double-click the waiver you want to modify.

This opens the Edit Waiver dialog box. The various fields are set to match the waiver you have selected.

3. Make any modifications.
4. Click the Create button.

A message asks you to confirm that you want to replace the existing waiver.

5. Click OK. Your waiver is modified.

Reporting Waivers

To report violation waivers, use the `report_waiver` command. By default, the `report_waiver` command reports all the waivers for each design in the session.

Use the `-design` option to report waivers for specific designs. [Example 35](#) shows the waivers reported for the TOP and HALF_ADDER designs.

Example 35 The report_waiver Output

```
ptc_shell> report_waiver -design TOP HALF_ADDER
*****
Report : report_waiver
Version: ...
Date   : ...
*****
Name      Rule      Scenario Condition           Comment
-----
Design: TOP
waiver_0_0  UNC_0003  scn1      -condition [list clock1 [get_clocks {TOP_VCLK}]] waived by Tom on ...
waiver_0_1  CAP_0001  scn3      -condition [list port [get_ports {PLUS_CARRY}]] waived by Dick on ...
Design: HALF ADDER
waiver_1_0  DRV_0001  default   -condition [list port [get_ports {A}]] waived by John on ...
```

To report only the waivers for a specific rule, use the `-rules` option, for example

```
ptc_shell> report_waiver -rules CAP_0001
```

Removing Waivers

To remove waivers, use the violation browser, the Waiver Configuration dialog box, or the `remove_waiver` command. Select the waiver and click “Remove waiver” as shown in [Figure 245](#).

To remove a waiver using the violation browser,

1. Right-click the waiver you want to remove.
2. Choose Undo created waiver.

To remove a waiver using the Waiver Configuration dialog box,

1. Choose Design > Waiver Configuration.

The Waiver Configuration dialog box appears, as shown in [Figure 245](#).

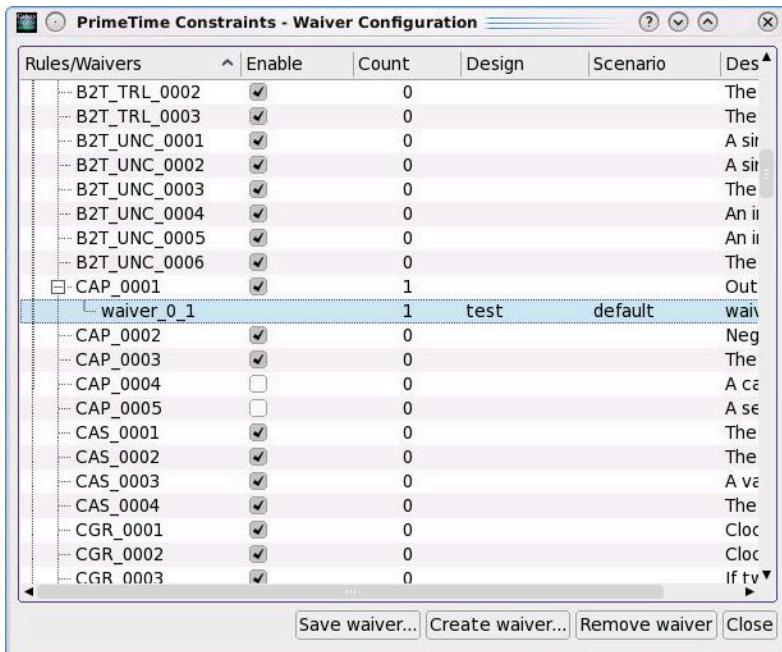
2. Select the waiver you want to remove.

The rule is not checked or reported the next time you run the `analyze_design` command.

3. Choose Remove waiver.

See “[Suppressing Violations](#)” and the `create_waiver`, `report_waiver`, and `remove_waiver` man pages for more information about using waivers.

Figure 245 Removing a Waiver With the Waiver Configuration Dialog Box



Writing Waivers to a File

You can use the `write_waiver` command to write waivers to a Tcl file, which can be sourced later to restore all the waivers. The syntax of the `write_waiver` command is

```
write_waiver -output output_file [-force]
```

Example 36 shows output of the `write_waiver` command:

Example 36 The `write_waiver` Output

```
if {[current_scenario] == "func"} {
    create_waiver -name "my_waiver1" -rule CLK_0026 -condition \
        [list "clock" [get_clocks {clk1 clk2}]] -comment "test"
    create_waiver -name "my_waiver2" -rule CLK_0003 -not_condition \
        [list "clock" [get_clocks clk3]] -comment "test"
}
if {[current_scenario] == "test"} {
    create_waiver -name "my_waiver3" -rule CLK_0003 -condition \
        [list "clock" [get_clocks {clk1 clk2}]] -not_condition \
        [list "pin" [get_pins {U1/A U1/B U1/Z}] [get_pins {U3/A U3/Z}]] \
        -comment "test"
}
```

When waivers are written to the waiver file, full paths of netlist objects are used, even if the original waiver specified an object name relative to the current instance.

Although the output of the `write_waiver` command is an accurate description of the active set of waivers, the waiver file might not be as readable as the original `create_waiver` commands. Also, the waiver output file can be large, particularly if you used wildcards in your original `create_waiver` commands.

report_constraint_analysis Command

To create a text report that summarizes violations in a design, use the `report_constraint_analysis` command. This command reports rule violations, user messages, and information about the defined rules. The results are sorted by analysis type, design, and scenario. Narrow the report by providing the following options:

- The `-include` option specifies the content to be included in the report. Use the `statistics` keyword with the `-include` option to report the constraints accepted for analysis.
- The `-style` option specifies whether a full or a summary report is generated.

You can tailor the output of the `report_constraint_analysis` command by using the `-rules` or `-rule_types` options. Using these options eliminates extraneous information from the report.

The report can be customized by using the `-format` option to generate the report in either a comma-separated values (CSV) file or a plain text file. The CSV format is useful for adding the report data to spreadsheets for data analysis.

[Example 37](#) shows the `report_constraint_analysis` command results based on the script in [Example 38](#).

Example 37 Analysis Report for Multiple Designs

```
ptc_shell> report_constraint_analysis -include violations -style full
*****
Report : report_constraint_analysis
         -include {violations }
         -style {full}
Version: ...
Date   : ...
*****
Scenario Violations Count Waived Description
-----
Design: FULL_ADDER
<Global Violations> 1 0 Scenario independent violations
error                 1 0
          UNT_0002    1 0 Library 'library' has incomplete units
                           defined.
          1 of 1        0 Library '...' has incomplete units defined.
default              13 0 Default scenario violations
error                5 0
          CAP_0001     2 0 Output/inout port 'port' has zero or
                           incomplete ...
```

Chapter 18: Constraint Consistency

Constraint Consistency Overview

1 of 2	0	Output/inout port 'SUM' has zero or incomplete ...
2 of 2	0	Output/inout port 'CARRY' has zero or incomplete ...
EXD_0012	3	The input delay at 'object_type' 'object' has zero ...
1 of 3	0	The input delay at 'port' 'A' has zero window for ...
2 of 3	0	The input delay at 'port' 'B' has zero window ...
3 of 3	0	The input delay at 'port' 'PREV_CARRY' has zero ...
warning	8	
...		
Design: HALF_ADDER		
<Global Violations>	1	Scenario independent violations
error	1	
UNT_0002	1	Library 'library' has incomplete units defined.
1 of 1	0	Library ...
default	12	Default scenario violations
error	4	
CAP_0001	2	Output/inout port 'port' has zero ...
1-of 2	0	Output/inout port 'SUM' has zero ...
2 of 2	0	Output/inout port 'CARRY' has zero ...
...		
Design: TOP		
<Global Violations>	1	Scenario independent violations
error	1	
UNT_0002	1	Library 'library' has incomplete units defined.
1 of 1	0	Library...
scn1	33	User-defined scenario violations
error	18	
CAP_0001	10	Output/inout port 'port' has zero or ...
1-of 10	0	Output/inout port 'PLUS1' has zero ...
2 of 10	0	Output/inout port 'PLUS2' has zero ...
EXD_0012	8	The input delay at 'object_type' 'object' has zero window for min and max values.
CLK_0021	1	Clock 'clock' is not used in this scenario.
1-of 1	0	Clock 'TOP_VCLK2' is not used in this scenario.
...		
scn2	32	User-defined scenario violations
error	18	
CAP_0001	10	Output/inout port 'port' has zero ...
EXD_0012	8	The input delay at 'object_type' 'object' has zero window for min and max values.
1 of 8	0	The input delay at 'port' 'A1' has zero window for min and max values.
2 of 8	0	The input delay at 'port' 'A2' has ...
...		

Top/Block Scenario Count Waived Description

Designs: TOP/HALF_ADDER

Chapter 18: Constraint Consistency

Constraint Consistency Overview

```

scn1/default          2   0
  U1/U1/U2            2   0      Block Instance
    error              2   0
      B2T_CAS_0001     2   0      Pin 'top_pin' in top instance has case value
                                    that is missing in block design

        1 of 2           0      Pin 'U1/U1/U2/U1/B' in top instance has case
                                    value that is missing in block design

        2 of 2           0      Pin 'U1/U1/U2/U2/B' in top instance has case
                                    value that is missing in block design

scn2/new              2   0
  U1/U1/U2            2   0      Block Instance
    error              2   0
      B2T_CAS_0001     2   0      Pin 'top_pin' in top instance has case value
                                    that is missing in block design

        1 of 2           0      Pin 'U1/U1/U2/U1/B' in top instance has case
                                    value that is missing in block design

        2 of 2           0      Pin 'U1/U1/U2/U2/B' in top instance has case
                                    value that is missing in block design...
-----
Total Error Messages  89  0
Total Warning Messages 61  0
Total Info Messages   1   0
Report : report_constraint_analysis

```

Example 38 Script for Running Basic Constraint Consistency and Hierarchical Consistency Checking on Multiple Designs

```

link_design TOP
link_design -add HALF_ADDER
link_design -add FULL_ADDER

current_design TOP
create_scenario scn1
create_clock -period 10 -name TOP_VCLK
create_clock -period 10 -name TOP_VCLK2
set_input_delay 2 -clock TOP_VCLK [all_inputs]
set_output_delay 2 -clock TOP_VCLK [all_outputs]

create_scenario scn2
create_clock -period 20 -name TOP_VCLK
set_input_delay 3 -clock TOP_VCLK [all_inputs]
set_output_delay 3 -clock TOP_VCLK [all_outputs]
analyze_design -scenarios "scn1 scn2"

current_design HALF_ADDER
create_clock -period 10 -name HA_VCLK
set_input_delay 3 -clock HA_VCLK [all_inputs]
set_output_delay 1 -clock HA_VCLK [all_outputs]

create_scenario new
create_clock -period 20 -name HA_VCLK
set_input_delay 2 -clock HA_VCLK [all_inputs]
set_output_delay 2 -clock HA_VCLK [all_outputs]

analyze_design

```

```
report_constraint_analysis -include violations
```

Using Attributes

The following topics describe attributes and how to use them:

- [Overview](#)
- [Setting, Listing, and Reporting Attributes](#)
- [Attribute Groups](#)
- [Using Arcs to Generate Custom Reports](#)
- [Creating a Collection of Library Timing Arcs](#)

Overview

An attribute is a string or value associated with an object in the design that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can write programs in Tcl to get attribute information from the design database.

Setting, Listing, and Reporting Attributes

Constraint consistency provides a set of commands for setting, listing, and reporting attributes as summarized in [Table 42](#).

Table 42 Attribute Commands

Commands	Description
<code>list_attributes</code>	Shows the attributes defined for each object class or a specified object class; optionally shows application attributes.
<code>get_attribute</code>	Retrieves the value of any attribute from a single object.
<code>report_attribute</code>	Displays the value of all attributes on one or more objects; optionally shows application attributes.

The `get_attribute` and `report_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.

Attribute Groups

Constraint consistency supports the attribute groups listed in [Table 43](#). The properties attached to each attribute group are described in the Constraint Consistency Online Help.

To see a list of all attributes available for a class of objects, use the `list_attributes -application` command. For example, to see the attributes associated with the `net` class of attributes, use the following command:

```
ptc_shell> list_attributes -application -class net
```

Attributes are read-only unless otherwise specified.

Table 43 Attribute Groups

Type of attribute group	Class name
Cell Attributes	cell
Clock Attributes	clock
Clock Group Attributes	clock_group
Clock Group Group Attributes	clock_group_group
Design Attributes	design
Exception Attributes	exception
Exception Group Attributes	exception_group
Input Delay Attributes	input_delay
Library Attributes	lib
Library Cell Attributes	lib_cell
Library Pin Attributes	lib_pin
Library Timing Arc Attributes	lib_timing_arc
Net Attributes	net
Output Delay Attributes	output_delay
Pin Attributes	pin
Port Attributes	port
Rule Attributes	rule
Rule Violation Attributes	ruleViolation
Ruleset Attributes	ruleset
Scenario Attributes	scenario
Timing Arc Attributes	timing_arc

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 needed attributes for further processing.

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.

The `from_pin` is an attribute of a timing arc and is the pin or port where the timing arc begins. The `to_pin` attribute is the pin or port where the timing arc ends. To determine the value of an attribute, use the `get_attribute` command.

Use the script in [Example 39](#) to show if any of the positive unate timing arcs of the U1 cell are disabled.

Example 39 Finding Disabled Timing Arcs of Specified Cells

```
ptc_shell> set arcs [get_timing_arcs -of_objects U1 -filter \
                     "sense == positive_unate"]
_se13
ptc_shell> foreach_in_collection arc $arcs { \
              echo [get_attribute $arc is_disabled] \
            }
true
false
```

Creating a Collection of Library Timing Arcs

To create a collection of library timing 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 needed attributes for further processing.

Use the `foreach_in_collection` command to iterate among the library arcs in the collection. Use the `get_attribute` command to obtain information about the arcs. However, you cannot copy, sort, or index a collection of library arcs. Use the `-filter` option of the `get_lib_timing_arcs` command to obtain only the library arcs that satisfy specified conditions. You cannot use the `filter_collection` command to filter a collection of library arcs. For a list of supported library timing arc object class attributes, see the Constraint Consistency Online Help.

The `from_lib_pin` is an attribute of a library timing arc, and is the library pin or port where the timing arc begins. The `to_lib_pin` attribute is the library pin or port where the timing arc ends. To determine the value of an attribute, use the `get_attribute` command.

Use the script in [Example 40](#) to list the senses of timing arcs starting from the clock pin of a flip-flop library cell.

Example 40 Listing the Senses of Timing Arcs

```
ptc_shell> set larc [get_lib_timing_arcs -from class/FD1/CP]
           _sel5
ptc_shell> foreach_in_collection larc $larc \
           { echo [get_attribute $larc sense] }
hold_clk_rise
setup_clk_rise
rising_edge
rising_edge
```

Customizing Rules and Reports

The following topics describe customizing rules and reports:

- [Enabling and Disabling Rules](#)
- [Modifying Rules](#)
- [Using and Creating Rule Sets](#)
- [Using Tcl Scripts to Report Violation Data](#)
- [Opening the SDC Browser From Custom Rules](#)

Enabling and Disabling Rules

Constraint consistency checking supports an extensive set of built-in rules. Not all of the built-in rules are enabled by default. For a list of all built-in rules and their default status, use the `report_rule *` command. When a rule is enabled, constraint consistency checks the design attributes against the rule and reports cases where the design violated the rule.

To enable a rule, use the `enable_rule` command, as shown in [Example 41](#).

Example 41 Enabling the EXD_0012 Rule

```
ptc_shell> enable_rule [list EXD_0012]
```

To disable a rule, use the `disable_rule` command. [Example 42](#) disables the DES_0001 and EXC_0002 rules.

Example 42 Disabling the DES_0001 and EXC_0002 Rules

```
ptc_shell> disable_rule [list DES_0001 EXC_0002]
```

Some of the rules are disabled by default because they inherently conflict with the enabled rules. For example, pre-layout rules are different from post-layout rules and can cause conflicts. Other rules are disabled by default because they are runtime intensive.

Modifying Rules

Constraint consistency enables you to customize any part of a rule. You can customize the message, the description, or the severity by using the `set_rule_message`, `set_rule_description`, and `set_rule_severity` commands.

For example, to change the severity of rule DES_0001 from warning level to error level, use the `set_rule_severity` command, as shown in [Example 43](#).

Example 43 Changing the Rule Severity

```
ptc_shell> set_rule_severity error [get_rules DES_0001]
```

To change the message of rule DES_0001, use the `set_rule_message` command, as shown in [Example 44](#).

Example 44 Changing the Rule Message

```
ptc_shell> set_rule_message [list "The pin of the register" \
    "didn't receive any clock"] [get_rules DES_0001]
```

Using and Creating Rule Sets

Constraint consistency rule sets are groups of rules designed for a specific function, such as signal integrity. Rule sets are designed to simplify constraint analysis and debugging by providing rules for specific areas in the design flow.

Constraint consistency supports the following built-in rule sets:

- compatibility
- postlayout
- prelayout
- primetime_si
- hierarchical

To determine the rules in the rule sets, use the `report_rule` command, as shown in [Example 45](#).

Example 45 Report the Rules in the primetime_si Rule Set

```
ptc_shell> report_rule [get_rulesets primetime_si]
```

To analyze your design using only the built-in `primetime_si` rule set, use the `analyze_design` command with the `-rules primetime_si` option, as shown in [Example 46](#).

Example 46 Analyze the Design Using Only the primetime_si Rule Set

```
ptc_shell> analyze_design -rules primetime_si
```

In addition to built-in rule sets, constraint consistency supports user-created rule sets with the `create_ruleset` command. This feature helps you debug issues specific to your design. For example, if you only need to analyze your design for two rules, such as DES_0001 and DES_0003, you would create a rule set containing only DES_0001 and DES_0003, as shown in [Example 47](#).

Example 47 Create a Rule Set Containing DES_0001 and DES_0003 Rules

```
## Create a ruleset targeting 2 rules only
ptc_shell> create_ruleset -name my_ruleset [list DES_0001 DES_0003]
```

To analyze your design with the user-defined rule set, `my_ruleset`, use the `analyze_design` command with the `-rules my_ruleset` option, as shown in [Example 48](#).

Example 48 Analyze the Design Using a Rule Set

```
ptc_shell> analyze_design -rules my_ruleset ...
```

To create a rule set containing only EXC type rules, use the `create_ruleset` command, as shown in [Example 49](#).

Example 49 Create a Rule Set for EXC Rules

```
ptc_shell> create_ruleset -name myExcRuleset [get_rules EXC*]
```

To create a rule set containing all built-in rules without the EXD rules, use the `create_ruleset` command, as shown in [Example 50](#).

Example 50 Create an Exclusion Rule Set

```
ptc_shell> set allRules [get_rules *]
ptc_shell> set EXDRules [get_rules EXD*]
ptc_shell> set myRules [remove_from_collection $allRules $EXDRules]
ptc_shell> create_ruleset -name myRuleset $myRules
```

Using Tcl Scripts to Report Violation Data

Although violation information is available in the GUI, you can use Tcl scripting to retrieve violation data. The `get_rule_violations` command allows you to obtain violation data in collection object form, so that you can use Tcl collection commands to manipulate the violation data.

Use the `get_rule_violations` command to create a collection of violation objects that meet a specified criteria. For example, if you wanted a collection of all clock rule violations in your design, use the following command:

```
ptc_shell> get_ruleViolations -of_objects [get_rules CLK_*]
```

You can also report the value of the attributes associated with the violation object. In the following example, the value of the `message` attribute for the first rule violation in a collection of rule violations is reported:

```
ptc_shell> get_attribute [index_collection [get_ruleViolations \
                           -of_objects CLK_0020] 0] message
Generated clock 'Gen_CLK' has edge relationships with its master clock
'MCLK' that cannot be satisfied. Only paths with 'negative' sense exist
from the master clock to source pin 'buf1/Z'. A 'positive' sense is
expected.
```

The `get_violation_info` command allows you to query the value of a rule parameter for a violation or the value of a violation details attribute (if applicable) for a rule violation.

To determine if a rule has `violation_details` attributes that can be queried with the `get_violation_info` command, query the `violation_details` attribute for that rule using the `get_attribute` command. For example,

```
ptc_shell> get_attribute [get_rule B2T_CLK_0001] violation_details \
                           top_missing_at_pin
```

The `get_attribute` command returns `top_missing_at_pin`. This indicates that the `B2T_CLK_0001` rule has the `top_missing_at_pin` attribute, which you can query with the `get_violation_info` command.

You can also query the `object_class` of the attributes associated with the violation details of a rule. Each violation details object has all of the attributes for the object class it belongs to. For example,

```
ptc_shell> set top_clk_pins [get_violation_info \
                           -attribute top_missing_at_pins [index_collection \
                           [get_ruleViolations -of_objects [get_rule B2T_CLK_0001]] 0]]
{"u7/ff1/CP", "u7/ff2/CP"}
ptc_shell> foreach_in_collection pin $top_clk_pins { \
               query_object -verbose $pin
}
{"pin:u7/ff1/CP"}
 {"pin:u7/ff2/CP"}
ptc_shell> get_attribute [get_pins u7/ff1/CP] clocks
 {"clk2"}
ptc_shell> get_attribute [get_pins u7/ff2/CP] clocks
 {"clk2"}
```

The first rule violation for the `B2T_CLK_0001` rule has two pins at the top level for which the `clk2` clock is reaching the register at the top level.

Opening the SDC Browser From Custom Rules

You can add hyperlinks that open the SDC browser custom rules.

To create a hyperlink that opens the SDC browser and passes the SDC file name and the line number of the violating constraint, use HTML tags and the `src:filename#line` syntax when you create your violation. The following example creates a hyperlink called File:constraints.sdc, Line 12, within the custom rule Information Pane:

```
<a href="src:constraints.sdc#12"> File:constraints.sdc, Line 12</a>
```

When this hyperlink is selected, the SDC browser opens the constraints.sdc file and points to line 12.

Creating and Using User-Defined Rules

Constraint consistency has many built-in rules which can detect a wide variety of issues. However, your design flow might have specific requirements that are not covered by the built-in rules. For this situation, you can create your own rules. These rules are checked automatically when you run the `analyze_design` command, and they are reported with the built-in rules in the violation browser.

The following topics describe creating and using user-defined rules:

- [Creating a Tcl Rule-Checking Procedure](#)
- [Registering a User-Defined Rule](#)
- [Using User-Defined Rules](#)
- [Viewing Violations of User-Defined Rules](#)
- [User-Defined Rule Example](#)

Creating a Tcl Rule-Checking Procedure

Using Tcl procedures, you can define a rule that tests for specific conditions on objects or attributes in the design. If an unwanted condition is detected, a violation of your rule is reported. The Synopsys Tcl interface provides full access to these objects and attributes.

To determine which attributes exist on any given object class, use the `list_attributes` command. For example, to list all the attributes that exist on the `rule` class, use the `list_attributes -class rule -application` command.

The Tcl procedure defining your custom rule contains

- Tcl code to inspect the objects and attributes in your design
- A `create_ruleViolation` command that generates a violation entry in the violation browser

Your Tcl code tests your design for the presence of an unwanted condition. When the condition occurs, the Tcl code calls the `create_ruleViolation` command, which reports a violation.

A single Tcl procedure can create violations for multiple rules; you do not need a separate procedure for each rule. This allows you to create efficient Tcl code because collections of objects can be shared between rules.

If a large collection is used multiple times in a Tcl procedure, such as `[get_nets -hier *`], it is more efficient to create this collection only one time and store the result in a variable.

The Tcl procedures in [Example 51](#) check that no output delay exists on a hierarchical pin. If any hierarchical pin has an output delay, constraint consistency reports a violation of a rule named UDEF_0001.

Example 51 Creating the Rule-Checking Procedure

```
proc get_details_for_output_delays {output_delays} {
    set text "<b>Relative Clocks</b><ul>"
    foreach_in_collection out_del $output_delays {
        append text "<li>" [get_attribute $out_del clock_name]
    }
    append text "</ul>"
    return $text
}

proc UDEF_0001 {} {
    set hier_cells [get_cells * -filter "is_hierarchical==true"]
    set hier_pins [get_pins -of_objects $hier_cells]
    foreach_in_collection pin $hier_pins {
        set output_delays [get_output_delays -of_objects $pin -quiet]
        if {[sizeof_collection $output_delays] > 0} {
            set pin_name [get_attribute $pin full_name]
            create_ruleViolation -rule UDEF_0001 -parameter_values $pin_name \
                -details [get_details_for_output_delays $output_delays]
        }
    }
}
```

The `create_ruleViolation` command reports a violation for the rule specified with the `-rule` option. You can use the `-parameter_values` option to pass parameters to the message that is displayed when a rule violation occurs. Parameters are passed by position. The order of the parameters needs to match the order in which they are declared with the `create_rule` command.

Extra text can be passed to any violation by using the `-details` option. The text appears in the Violation Details section. The GUI can display HTML formatting; you can use HTML tags to format the text. [Example 54 on page 724](#) shows an example of a user-defined rule.

Registering a User-Defined Rule

After you create a Tcl procedure to check for a design violation, you need to register the rule by using the `create_rule` command, as shown in [Example 52](#).

Example 52 Using the create_rule to Create a New Rule Named UDEF_0001

```
ptc_shell> create_rule -name UDEF_0001 \
    -severity warning \
    -description {"Team XYZ does not allow output delays to be \
specified on hierarchical pins"} \
    -message {"Output delay is specified on hierarchical pin '" \
"'."} \
    -parameters {"pin"} \
    -checker_proc UDEF_0001_checks
```

This command registers the new rule, specifies parameters such as the severity of the violation, and identifies the name of the Tcl procedure that checks for the violation.

Each rule must have a unique name. You must start your rule name with “UDEF_” to avoid conflict with the built-in rule names. The severity can be adjusted with the `-severity` option; legal values are `info`, `warning`, `error`, and `fatal`.

A rule also needs to have a message. The message is displayed whenever a violation occurs. The message field is composed of fields of fixed text and optional parameters that are defined with the `-parameters` option. The parameters are passed to the message from the `create_ruleViolation` command. The parameters are inserted between the message strings in the order they have been defined.

For example, to report the name of the clock and number of clock sources causing a violation, you could use the following parameter definition:

```
ptc_shell> create_rule -name UDEF_0001 -severity warning \
    -message {"The clock " " has " "source(s) on inout ports."} \
    -parameters {"clock" "count"} \
    -description "Clock defined on an inout port" \
    -checker_proc clk_inout_checker
```

Use the `-checker_proc` option to name the Tcl procedure that performs the check. The procedure that you specify with this option needs to exist before you run the `create_rule` command.

Use the following guidelines to define when to run user-defined rules:

- Global rules – This type of rule depends on the structure of the design or libraries. It is not related to the timing constraints. This type of rule is run only one time for the whole design. To define a global rule, use the `-global` switch.
- Scenario-related rules – This type of rule depends on the current set of constraints applied to the design. It is run one time for each scenario.

A Tcl checking procedure can define several custom rules. However, all the rules in a Tcl procedure must be the same type, either global or scenario. The first rule declared in the checker code sets the type, global or scenario. Subsequent rules defined in the same checker code must be the same type.

Using User-Defined Rules

After you create and register your rule, constraint consistency automatically checks the rule when you run the `analyze_design` command. Violations of your rule are displayed in the violation browser along with the built-in rule violations.

Messages written in the log file give you information about the status of your rules, as shown in [Example 53](#).

Example 53 Messages Produced by the `analyze_design` Command for a User-Defined Rule

```
ptc_shell> analyze_design
Information: Checking scenario 'scen1': Starting rule checks (ADES-002)
Information: Checking scenario 'scen1': Starting user-defined rule checks
(ADES-002)
Information: Running user-defined checker : UDEF_Rule_6. (ADES-021)
Information: Running user-defined checker : check_UDEF_CLK_0026.
(ADES-021)
Information: Running user-defined checker : check_UDEF_SID_SOD_virtual.
(ADES-021)
Information: Checking netlist: Starting scenario-independent rule checks.
(ADES-003)
```

Viewing Violations of User-Defined Rules

The violation browser shows violations of all rules. Violations of user-defined rules are displayed with the violations of the built-in rules. See “[Violation Browser](#)” for detailed information about the violation browser.

User-Defined Rule Example

[Example 54](#) shows Tcl code that creates the user-defined UDEF_rule_6 rule.

Example 54 Tcl Code for a User-Defined Rule

```
## Here is a global variable used to store the latency data
global latencyData

proc getHTMLExampleRule_6 {} {
    set HTML_page ""
    append HTML_page "<P>"
    append HTML_page "<tr><th align=\"left\">ptc_shell> report_clocks \\[get_clocks
CLK_virtual\] -skew</td></tr>"
    append HTML_page "<tr><th
align=\"left\">*****</td></tr>"
```

append HTML_page "<tr><th align=\"left\">Report : clock_skew</td></tr>"

Chapter 18: Constraint Consistency

Constraint Consistency Overview

```

append HTML_page "<tr><th align=\"left\">Design : design</td></tr>"
append HTML_page "<tr><th align=\"left\">Scenario: default</td></tr>"
append HTML_page "<tr><th align=\"left\">Version: ...</td></tr>"
append HTML_page "<tr><th align=\"left\">Date : ...</td></tr>"
append HTML_page "<tr><th align=\"left\">*****</td></tr>"                                Min Condition Sou
rce
Latency      Max Condition Source Latency</td></tr>
    append HTML_page "<tr><th align=\"left\">-----"
align=\"left\">-----</td></tr>"                                Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML_page "<tr><th align=\"left\">-----"
align=\"left\">-----</td></tr>"                                --</td></tr>"           .01
3.01          -          - 3.23      3.23
    append HTML_page "</P>"           return ${HTML_page}
}

proc createHTMLPage { clk } {
    global latencyData
    set HTML ""

    append HTML "<style>"
    append HTML " P { font-face : \"Courier New\" }"
    append HTML "</style>"

    append HTML "<table border=\"1\" width=\"500\" cellspacing=\"0\""
cellpadding="3">

## Create the Top 2 rows of the table containing labels
## First Row
    append HTML "<tr><td></td>"
    append HTML "<th colspan=\"2\" align=\"center\""
bgcolor="lightblue">Early</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "<th colspan=\"2\" align=\"center\""
bgcolor="lightblue">Late</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "</tr>"

## Second row
    append HTML "<tr><td><th bgcolor=\"lightblue\" align=\"center\">Min</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "<th align=\"center\" bgcolor=\"lightblue\">Max</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "</tr>"

## Create the next two rows containing latency data
## RISE row
    append HTML "<tr><th bgcolor=\"lightblue\" align=\"center\">Rise</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "<th align=\"center\">$latencyData(clock_source_latency_early_rise_min)</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "<th align=\"center\">$latencyData(clock_source_latency_early_rise_max)</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3
    append HTML "<th align=\"center\">$latencyData(clock_source_latency_late_rise_min)</th>"           Min</th>"           Max</th>"           Min</th>"           Max</th>"           Early_r Early_f  L
ate_r
Late_f      Early_r Early_f Late_r  Late_f      Rel_clk</td></tr>"           2.53      2.53     3

```

Chapter 18: Constraint Consistency

Constraint Consistency Overview

```

append HTML "<th
align=\"center\">>$latencyData(clock_source_latency_late_rise_max)</th>"
append HTML "</tr>""
# FALL row
append HTML "<tr><th bgcolor=\"lightblue\" align=\"center\">>Fall</th>"
append HTML "<th
align=\"center\">>$latencyData(clock_source_latency_early_fall_min)</th>"
append HTML "<th
align=\"center\">>$latencyData(clock_source_latency_early_fall_max)</th>"
append HTML "<th
align=\"center\">>$latencyData(clock_source_latency_late_fall_min)</th>"
append HTML "<th
align=\"center\">>$latencyData(clock_source_latency_late_fall_max)</th>"
append HTML "</tr>""
append HTML "</table>""

# Some spacing to ease reading...
append HTML "<tr><td>""
append HTML "<tr><td>""

## Display where the clock was created
append HTML "<table border=\"1\" width=\"400\" cellspacing=\"0\""
cellpadding="3">""
## File location is a String containing the file name and lines of
operation on
that clock
## The first pair is where the clock has been created
## Other should notably match the eventual set_clock_latency commands
## Other commands can affect this clock and thus be added in this String
as
well...
set fileLocation [get_attribute -quiet [get_clocks ${clk}] file_line_info]
set fileAttributes [split $fileLocation " "]

append HTML "<tr><th align=\"left\" bgcolor=\"lightblue\">>${clk}
definition</th>""
set name [string trim [lindex ${fileAttributes} 0] "{}"]
set line [string trim [lindex ${fileAttributes} 1] "{}"]
append HTML "<th align=\"left\">>File ${name}, line ${line}</th>""
append HTML "</tr>""
append HTML "<tr><td>""

## variable i starts at 2 because we already printed the file/line for
clock
creation
for {set i 2} {$i < [expr [length ${fileAttributes}]]} {set i [expr ${i} +
2]} {
    set name [string trim [lindex ${fileAttributes} ${i}] "{}"]
    set line [string trim [lindex ${fileAttributes} [expr ${i} + 1]] "{}"]
    append HTML "<tr><th align=\"left\" bgcolor=\"lightblue\">> Other
related
exceptions</th>""
    append HTML "<th align=\"left\">>File ${name}, line ${line}</th>""
    append HTML "</tr>""
}
append HTML "</table>""

## Suggest some help.
# Some spacing to ease reading...

```

Chapter 18: Constraint Consistency

Constraint Consistency Overview

```

append HTML "<tr><td>" 
append HTML "<tr><th align=\"left\"> <font size=\"4\"> <font
color=\"DarkSlateBlue\">Debug Suggestions</th></tr>" 
append HTML "<tr><th align=\"left\"> You can get more information about
your
particular issue </th></tr>" 
append HTML "<tr><th align=\"left\">using the following command. Cut and p
aste it
in your console </th></tr>" 
append HTML "<tr><td>" 
append HTML "<tr><th align=\"left\">report_clocks \[get_clocks ${clk}\] -s
kew</
th></tr>" 

## Add an example to complete the page
append HTML "<tr><td>" 
append HTML "<tr><th align=\"left\"> <font size=\"4\"> <font
color=\"DarkSlateBlue\">Example</th></tr>" 
append HTML [getHTMLExampleRule_6]
return ${HTML}
}

proc UDEF_Rule_6 {} {
    global latencyData
    ## I am going to reuse this variable to simplify the scripting below
    set sourceLatAttrList [list \
        clock_source_latency_early_fall_min \
        clock_source_latency_early_fall_max \
        clock_source_latency_early_rise_min \
        clock_source_latency_early_rise_max \
        clock_source_latency_late_fall_min \
        clock_source_latency_late_fall_max \
        clock_source_latency_late_rise_min \
        clock_source_latency_late_rise_max \
    ]
    
    ## Start from all the clocks and only keep the virtual ones.
    set zVirtualClocks [filter_collection [all_clocks] -regexp { undefined(sou
rces) }]
    
    ## Iterate through all the virtual clocks
    ## We have a violation if the latency is incompletely defined
    ## Incompletely defined = at least one attribute missing or
    ## one of the latency value is '0'
    foreach_in_collection itr ${zVirtualClocks} {
        set incompleteLatency false
        ## At the same time we check the attributes, we will also fill a hash
        ## table with the relevant values. This will save time when printing the
        ## values since we won't have to query again.
        foreach att ${sourceLatAttrList} {
            set zAttValue [get_attribute -quiet [get_clocks [get_object_name
${itr}]]]
            ${att}]
            if {(${zAttValue} == "") || (${zAttValue} == 0)} {
                set incompleteLatency true
            }
            if {${zAttValue} == ""} {
                set latencyData(${att}) "--" } else {
                set latencyData(${att}) ${zAttValue} }
        }
        ## If we do have a violation, we need to create the proper display
    }
}

```

```

if ${incompleteLatency} == true} {
    create_ruleViolation -rule UDEF_Rule_6 -parameter_values [get_object_name
${itr}] \
    -details [createHTMLPage [get_object_name ${itr}]]
}
}

create_rule -name UDEF_Rule_6 -severity warning \
    -message {"Clock latency is not fully defined for virtual clock '" "'."} \
    -parameters {"clock"} \
    -description {"The clock source latency is missing or incompletely
defined for a
virtual clock" } \
    -checker_proc UDEF_Rule_6

```

Rule-Related Commands

The rule-related commands include:

- `get_rule_property`

This command returns the property of a single rule.

- `get_rules`

This command creates a collection of rules.

- `get_rulesets`

This command creates a collection of rule sets.

- `remove_ruleset`

This command removes specified user-defined rule sets. Built-in rule sets cannot be removed.

- `report_rule`

This command reports detailed rule information.

- `set_rule_property`

This command enables you to set rule properties.

Reading Designs With Incomplete or Mismatched Netlists

During early design development, design data is updated often. 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. This causes a design mismatch. To link designs with incomplete or mismatched netlists, set the `link_allow_design_mismatch` variable to

true. The default is false. When mismatched pins, ports, cells, or nets are linked, constraint consistency issues DMM warning messages.

If an object is affected by a mismatch, the `is_design_mismatch` attribute of the object is set to true. The default is false. The `is_design_mismatch` attribute is viewable in the Properties dialog box in the GUI.

When the `link_allow_design_mismatch` variable is true, constraint consistency links designs with the following types of netlist problems:

- Pins exist in higher-level hierarchies that do not exist at lower-level hierarchies

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 unconnected higher-level pins are left dangling. Constraints are not applied on these pins.

Constraint consistency also links netlists with anchor cells inserted on the dangling nets. In this case, constraints related to mismatched pins in the in-memory linked design are preserved.
- Constraint consistency ignores extra bits on the bus. 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.
- Library and netlist cells contain different power and ground (PG) information

If there is no PG pin information in the library, but the information is in the netlist, then the mismatch is treated as if pins exist in higher level hierarchies of the design, but not at the lower level.

- Cells missing from the library

By default, the linker automatically creates black boxes for unresolved references, which enables constraint consistency to continue linking. Constraint consistency issues a LNK-005 message for this condition.

When the `link_allow_design_mismatch` variable is set to true, Constraint consistency issues a DMM-905 message for unresolved references and continues linking.

To view the mismatches located while linking a design, use the `report_design_mismatch` command. [Example 55](#) shows a report example.

Example 55 Reporting Mismatches

```
ptc_shell> report_design_mismatch
*****Report : design_mismatch
Design : top
```

Chapter 18: Constraint Consistency

Constraint Consistency Overview

Message	Design	Object	Type	Count	Mismatch Description
DMM-038	design	object	type	1	Pin "%s" of cell "%s" of reference "%s" in design "%s" shows inconsistency between master and instantiation because %s. Ignoring this pin.
top	u_block	reset pin	1 of 1		Pin "reset" of cell "u_block" of reference "top" in design "block" shows inconsistency between master and instantiation because it does not exist in the reference block. Ignoring this pin.
Message	Design	Object	Type	Count	Mismatch Description
DMM-905	design	object	type	48	Cannot find the design '%s' in design libraries.
top	BUFFD1	design	1 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	2 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	3 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	4 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	5 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	6 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	7 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	8 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	9 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	10 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	11 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	12 of 48		Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	13 of 48		Cannot find the design 'BUFFD1' in design libraries.

top	BUFFD1	design	14 of 48	Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	15 of 48	Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	16 of 48	Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	17 of 48	Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	18 of 48	Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	19 of 48	Cannot find the design 'BUFFD1' in design libraries.
top	BUFFD1	design	20 of 48	Cannot find the design 'BUFFD1' in design libraries.

Information: Reaching the message display limit of report_design_mismatch. (UI C-063)
1

Design Consistency Checking

The following topics describe design consistency checking capabilities:

- [Rules and Violations](#)
- [Built-In Rules](#)

Rules and Violations

Constraint consistency checks are performed on a loaded, linked, and constrained design for a variety of design-related and constraint-related conditions. Each condition that is checked is called a rule. An occurrence of something in the design that fails to meet a checked condition is called a rule violation. There can be multiple violations of a given rule in different places within the design. A violation can have a severity level of Information, Warning, or Error, depending on the type of rule. Each rule has a designated code consisting of three or four letters, an underscore, and four digits. For example, the code “CLK_0003” represents a clock rule, “Generated clock clock_name is not expanded because it has no clock reaching its master source pin_name.” If a generated clock is defined in the design, but the `-source` option of the `create_generated_clock` command designates a pin that does not have a clock defined on it, constraint consistency detects and reports this improper constraint specification as a CLK_0003 violation. The `analyze_design` command checks a standard set of rules. The `compare_block_to_top` command checks another, different set of standard rules, and the `compare_constraints` command checks yet another set of standard rules. You can disable any given rule so

that it is not checked. Some rules are disabled by default to save runtime; you can enable these rules when you want them to be checked. You can skip checking of a particular rule for a given set of conditions, such as checking for a particular block in the design or checking of a particular clock. A specified condition that prevents checking of a rule is called a waiver. In addition to the standard rules, you can create your own rules to find and report conditions that are important for your design. These are called user-defined rules.

Built-In Rules

When you start constraint consistency it automatically loads the set of built-in rules shown in [Table 44](#). These rules check your design for potential timing constraint and library-related problems.

Table 44 Built-In Rules

Boundary Conditions	CAP_xxxx – capacitance values DRV_xxxx – drive constraints EXD_xxxx – external delays
Constraints/Exception Analysis	CAS_xxxx – case analysis EXC_xxxx – timing exceptions
Clocks	CGR_xxxx – clock groups CLK_xxxx – clock properties CNL_xxxx – network latencies CTR_xxxx – clock transitions CSL_xxxx – clock source latencies UNC_xxxx – clock uncertainties
General	CMP_xxxx – tools compatibility DES_xxxx – design constraints HIER_xxxx – hierarchical LOOP_xxxx – timing loops PRF_xxxx – performance NTL_xxxx – netlists

Table 44 Built-In Rules (Continued)

UNT_xxxx – library units

See the Constraint Consistency Online Help for detailed rule descriptions.

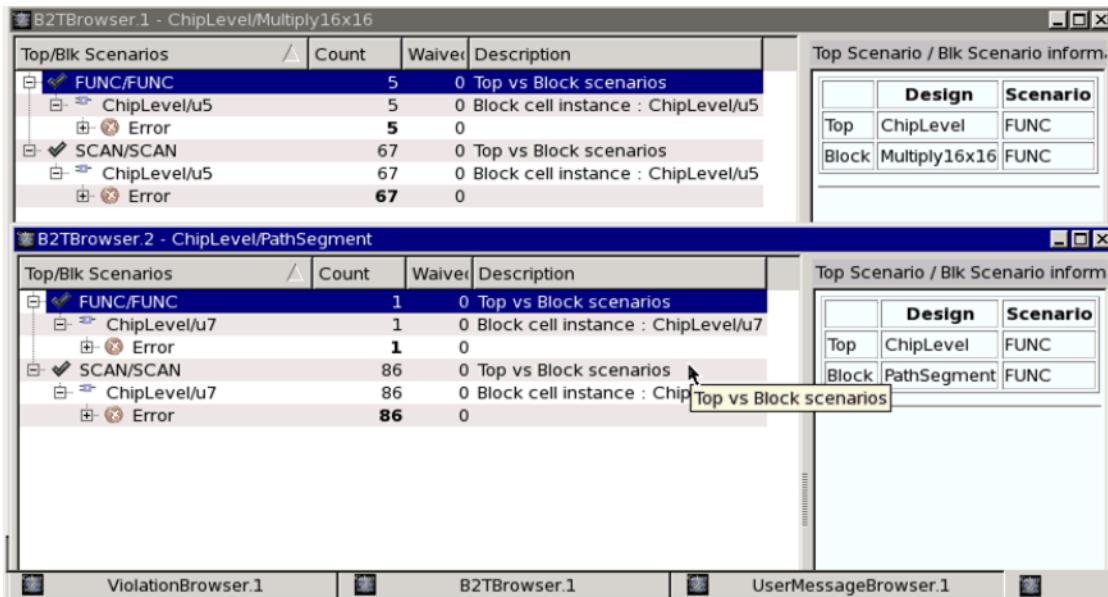
Hierarchical Consistency Checking

Design teams use various strategies to create block-level constraints and top-level constraints. Sometimes block-level constraints are propagated to the top level in a bottom-up design flow. During the design iteration process, new constraints can be manually inserted into both the top-level and block-level designs. To check whether the constraints match after such operations, use the `compare_block_to_top` command.

Each block comes with a set of constraints with which it was synthesized and placed. These constraints include clock definitions, timing exceptions, and boundary conditions. When you use the `compare_block_to_top` command, hierarchical constraint consistency loads the constraints for both the block-level design and the top-level design and performs a comparison between the two sets of constraints. The constraint consistency feature checks a unique netlist against the two sets of constraints. Hierarchical consistency checking generates a set of block-to-top rule violations that can be debugged just like the general design rules.

Hierarchical consistency checking preserves the block-to-top results for multiple blocks in a single session. [Figure 246](#) shows violations for two blocks in two separate violation browser windows.

Figure 246 Violations for Two Blocks in Separate Violation Browser Windows



For more information about analyzing the results of multiple blocks in a single session, see [“report_constraint_analysis Command”](#).

To learn more about hierarchical consistency checking, see

- [Comparing Block- and Top-Level Constraints](#)
- [Built-In Hierarchical Consistency Checking Rules](#)

Comparing Block- and Top-Level Constraints

Incorporating third-party IP blocks or reusing blocks from other designs is a common practice in chip development. Each of these predesigned blocks usually comes with a set of constraints for building the block. To guarantee that the block functions correctly, you must ensure that these block constraints are still valid after you incorporate the block in your bigger design.

To check the consistency between block- and top-level constraints, use the `compare_block_to_top` command to find inconsistencies between hierarchical constraints. To find potential issues within a single set of constraints, use the `analyze_design` command at the top or individual block level.

These features are described in the following topics:

- [Methodology](#)
- [Checking Block and Top Constraints in the GUI](#)

- [Constraint Comparison Example](#)
- [Checking Block Versus Top Consistency](#)
- [Checking Multiple Scenarios](#)
- [Checking Multiple Instances of One Block](#)
- [Creating Waivers](#)
- [Creating Text Reports](#)

Methodology

The following sections describe the hierarchical consistency checking flow:

- [Keeping Multiple Linked Designs for Hierarchical Consistency Checking](#)
- [Flow Script Example](#)

Keeping Multiple Linked Designs for Hierarchical Consistency Checking

When linking a design, the default behavior is to remove previously linked designs. However, the hierarchical consistency checking flow is different from the standard flow because you need to keep two linked designs in memory:

- The block design with block-level constraints applied
- The top design with top-level constraints applied

To keep previously linked designs in memory, use the `link_design` command with the `-add` option. When there are multiple linked designs in memory at the same time, use the `current_design` command to switch between the designs.

Flow Script Example

The script in [Example 56](#) shows the flow for hierarchical consistency checking.

Example 56 Hierarchical Consistency Checking Script

```
set_app_var search_path [list . ${TOP_DIR}/libs ${TOP_DIR}/design]
set_app_var link_path [list * libs.db]

read_verilog ${TOP_DIR}/design/ChipLevel_no2blocks.v
read_verilog ${TOP_DIR}/design/Multiply16x16.v
read_verilog ${TOP_DIR}/design/PathSegment.v

link_design ChipLevel
link_design -add Multiply16x16

current_design ChipLevel
source ${TOP_DIR}/dc_outdir/ChipLevel_propagate.sdc -echo
current_design Multiply16x16
```

```
source ${TOP_DIR}/design/Multiply16x16.sdc -echo
current_design ChipLevel
compare_block_to_top -block_design [get_designs Multiply16x16]
remove_linked_design [get_designs Multiply_16x16]
```

Checking Block and Top Constraints in the GUI

Hierarchical consistency checking provides comprehensive GUI windows for performing block-to-top comparisons and debugging block-to-top rule violations.

Running Hierarchical Consistency Checking from the GUI

Typical steps to perform hierarchical consistency checking are:

1. Load the design.
2. Run the `analyze_design` command at the top level and at the block level and make sure that both the top-level and block-level constraints are clean.
3. Run the `compare_block_to_top` command specifying the block-level and top-level designs to be compared.
4. Open the Block-to-Top mismatch browser window: choose Analysis > B2T Mismatch Browser.
5. Debug the block-to-top rule violations.

Note that it is possible to waive block-to-top rule violations.

The Block-to-Top mismatch browser window shows the design-level rule violations, if any.

Debug and correct the design to ensure that the hierarchical consistency checking results are correctly generated.

Displaying Schematics of Block and Top Rule Violations

You can display schematics of block and top-level constraint mismatches, as shown in [Figure 247](#) and [Figure 248](#). A schematic link is provided in the Block-to-Top (B2T) information pane, if the mismatch can be visualized with a schematic.

[Figure 247](#) shows a B2T_CLK_0006 violation. When you select this violation in the Block-to-Top mismatch browser, the information pane shows violation details that include a Schematic link (circled in red). When you click the Schematic link, the block and top-level schematics are displayed, as shown in [Figure 248](#). The white-colored MUX outputs indicate the first place where the violation occurs.

Chapter 18: Constraint Consistency

Hierarchical Consistency Checking

Figure 247 Block-to-Top Information Pane With Schematic Link

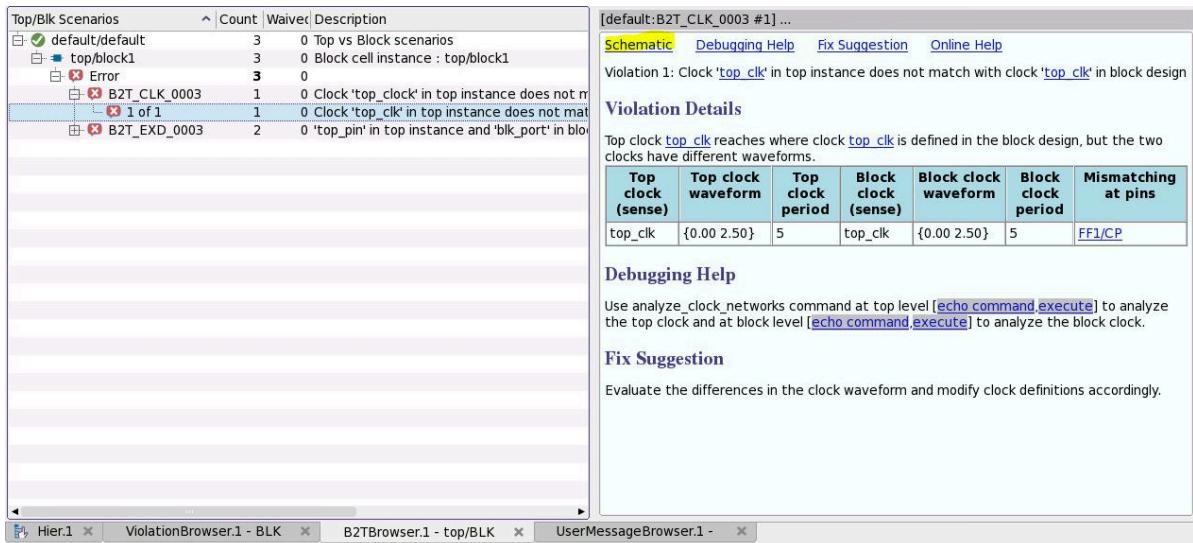
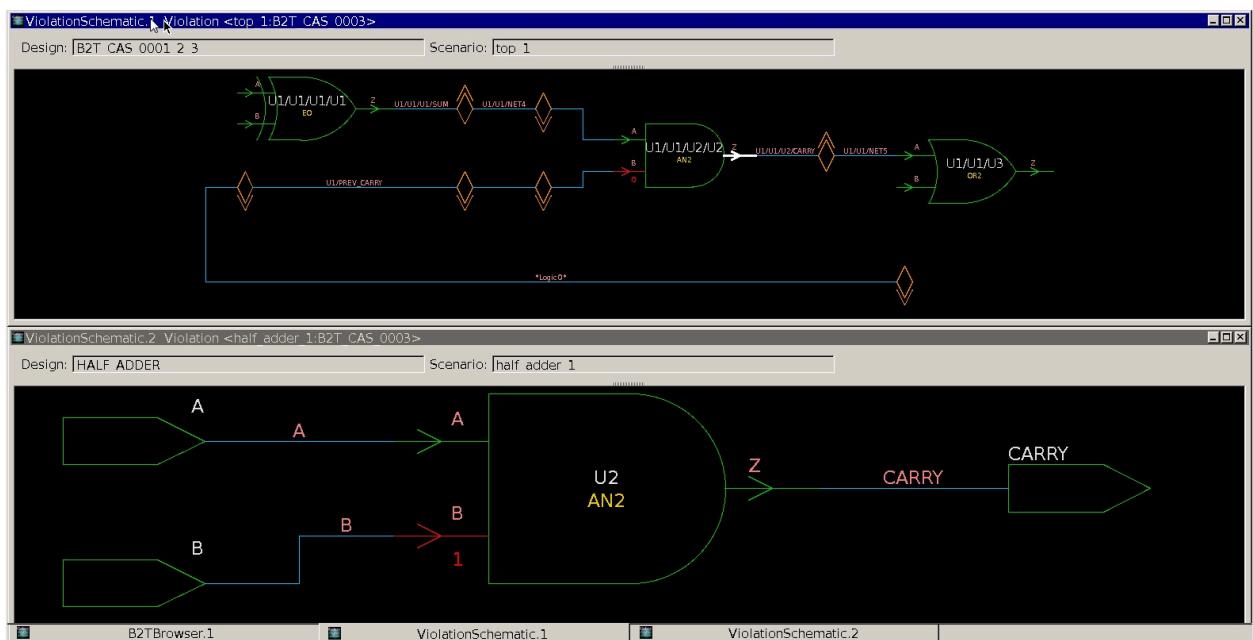


Figure 248 Dual Schematic

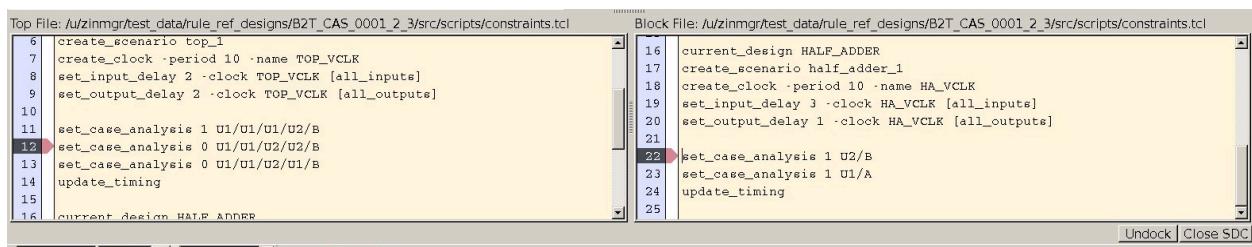


In addition to showing dual schematic views of the violation in Figure 248, the information pane in Figure 249 contains links to the SDC source files associated with both the block and top designs. When you click the source file link for either the block or top SDC file, hierarchical consistency checking displays both top and block SDC source files side-by-side with the violating constraint highlighted in each file, as shown in Figure 250.

Figure 249 Information Pane When a Violation Is Selected



Figure 250 Dual SDC Viewer



Constraint Comparison Example

The script in [Example 57](#) uses the `compare_block_to_top` command to run block versus top consistency checks. The example loads the top-level netlist that includes the netlist for the block, links the top design, applies the top-level constraints, links the block-level design, applies block-level constraints, and sets the output directory. Finally, it runs block versus top consistency checks.

Example 57 Default Report for `report_exceptions`

```

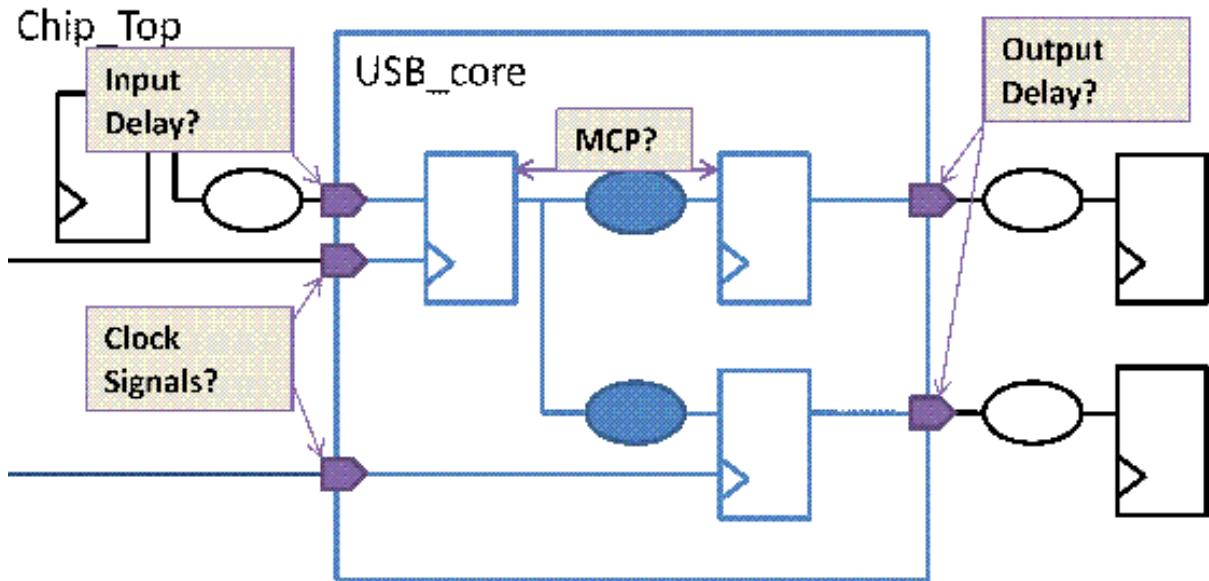
ptc_shell> read_verilog ./chip.v
ptc_shell> link_design chip
ptc_shell> source chip_constraints.tcl
ptc_shell> link_design -add usb_core
ptc_shell> source usb_core_constraints.tcl
ptc_shell> set out_dir /user/abc/public_html/compare_chip
ptc_shell> compare_block_to_top -block_design usb_core

```

Checking Block Versus Top Consistency

The `compare_block_to_top` command performs consistency checks between block-level design constraints and the corresponding top-level constraints. Figure 251 shows the block design `USB_core` inside the design `Chip_Top` and identifies some objects that can be compared.

Figure 251 Block Design `USB_core` Inside Design `Chip_Top`



Hierarchical consistency checking looks for the following categories of constraints:

- Clock constraints
- Boundary condition constraints
- Exception type constraints
- Case value constraints
- Miscellaneous constraints

By default, all five categories are checked by the `compare_block_to_top` command. To restrict the categories, use the `-include` option.

The following sections describe the five categories checked by hierarchical consistency checking:

- [Clock Checks](#)
- [Boundary Condition Checks](#)

- [Exception Checks](#)
- [Case Settings Checks](#)
- [Miscellaneous Settings Checks](#)

Clock Checks

The clocks defined at the block level are compared to the clocks from the top-level reaching this block. Hierarchical consistency checking matches the clocks for period, sense, waveform, and phase.

Tcl-based mapping is available as an alternative to the clock-check matching described previously.

Boundary Condition Checks

The input and output delays of the block are checked against the top level constraints. The actual input and output delay values are not subject to comparison. However, the following items are part of the comparison:

- Presence or absence of input or output delays
- Clock used to establish the input or output delays
- Triggering edge of the clock
- Edge-sensitive versus level-sensitive

Exception Checks

When running hierarchical consistency checking, the following timing conditions are compared:

- `set_min_delay` and `set_max_delay` exceptions
- `set_multicycle_path` exceptions
- `set_clock_groups` clock grouping
- `set_false_path` exceptions

Case Settings Checks

Hierarchical consistency checking compares case settings by looking at

- Case and constant values in the top design at the boundary of the block
- Case settings at the ports of the standalone block

If any case settings in the block design are not covered by the settings in the top design, then those are reported as missing block constraints in the report.

Miscellaneous Settings Checks

Miscellaneous checks compare the top- and block-level settings related to the `set_disable_timing` command, clock gating, clock sense, and operating conditions.

Checking Multiple Scenarios

You can compare multiple scenarios simultaneously. Use the `-block_scenarios` and `-top_scenarios` options to specify the pairs of scenarios to be compared. The first scenario passed in `-top_scenarios` is matched against the first scenario passed in `-block_scenarios`. [Example 58](#) shows such a case.

Example 58 Comparing Multiple Scenarios

```
compare_block_to_top -block_design Multiply16x16 \
    -top_scenarios [list FUNC SCAN] \
    -block_scenarios [list BFUNC BTEST] \
```

In [Example 58](#), the following scenarios are compared against each other:

- Top: FUNC; Block: BFUNC
- Top: SCAN; Block: BTEST

Checking Multiple Instances of One Block

When several instances of one block exist in the design, hierarchical consistency checking compares all of them to the top-level constraints at the same time. This implies, however, that all instances of the block were created with the same set of constraints. If this is not

the case, run the `compare_block_to_top` command several times: one execution of `compare_block_to_top` for each combination of unique constraints per block.

To achieve this behavior, the constraint consistency feature can also work on instances, as opposed to designs. [Example 59](#) demonstrates this approach.

Example 59 Using the compare_block_to_top Command

```
compare_block_to_top -block_design Multiply16x16 \
    -top_scenarios [list FUNC SCAN] \
    -block_scenarios [list BFUNC BTTEST] \
    -cells [get_cells [list TOP/Block1 TOP/Block2]]
```

Creating Waivers

You can create waivers for block-to-top rules through the GUI or the command line.

For more information about creating, editing, removing, and reporting waivers, see “[Suppressing Violations](#)”.

Creating Text Reports

In addition to GUI reports, you can create text reports using the `report_constraint_analysis` command.

For more information, see “[report_constraint_analysis Command](#)”.

Built-In Hierarchical Consistency Checking Rules

Built-in block-to-top rules check top-level constraints against block-level constraints. When you run the `compare_block_to_top` command, the built-in block-to-top rules are automatically loaded.

Table 45 Built-In Block-to-Top Rules

Rule	Description
B2T_EXD_xxxx	Boundary conditions/external delays
B2T_CAS_xxxx	Case analysis Constraints/Exception analysis
B2T_EXC_xxxx	Timing exceptions Constraints/Exception analysis
B2T_CLK_xxxx	Clock properties
B2T_CLT_xxxx	Clock source latency and clock network latency
B2T_UNC_xxxx	Clock uncertainty

Table 45 Built-In Block-to-Top Rules (Continued)

Rule	Description
B2T_DIS_xxxx	Disabled objects and arcs
B2T_OPC_xxxx	Operating conditions

When appropriate, the rules that check these constraints have a tolerance rule property that allows some difference in the values being compared before a violation is issued. The tolerance is specified as a percentage of the smallest value compared. By default, the tolerance is set at 1e-5 percent, which is equivalent to performing exact matches. To change the tolerance, use the `set_rule_property` command.

See the rule reference pages in the online Help for detailed rule descriptions.

Correlation Consistency Checking

During design development, some modifications might be made to the original constraints file. These modifications could cause mismatches between the original constraints file and the modified constraints file. You can use correlation consistency checking to compare these two sets of design constraints using one common netlist, so you can determine the differences between the two sets of constraints. You can also check the constraints consistency between two designs and the two constraints sets.

To learn more about correlation consistency checking, see

- [Comparing Two Sets of Design Constraints](#)
- [Comparing Two Designs With Two Sets of Design Constraints](#)

Comparing Two Sets of Design Constraints

Use the `compare_constraints` command to run the comparison. The following sections describe the `compare_constraints` command:

- [Constraint Set Comparison Overview](#)
- [Methodology](#)
- [Correlation Consistency Violation Browser](#)
- [Rules and Violations When Comparing Constraint Sets](#)
- [Creating Waivers](#)
- [Text Reports](#)

Constraint Set Comparison Overview

To compare constraint sets, define them as scenarios and then use the `compare_constraints` command to compare the scenarios. The scenarios must be created on the current design, which can be specified by the `current_design` command.

The `compare_constraints` command checks the following types of constraints:

- Clock definitions

`create_clock`, `create_generated_clock`

- External delays

`set_input_delay`, `set_output_delay`

- Timing exceptions

`set_false_path`, `set_multicycle_path`, `set_min_delay`, `set_max_delay`,
`set_clock_groups`

- Case analysis

`set_case_analysis`

- Disable timing

`set_disable_timing`

- Uncertainty

`set_clock_uncertainty`

- Transitions

`set_clock_transition`, `set_input_transition`

- Latency

`set_clock_latency`

- Clock gating

`set_clock_gating_check`

- Clock sense

`set_sense`

- Loading

`set_load`

When appropriate, the rules that check these constraints have a tolerance property that allows some difference in the values being compared before a violation is issued. The tolerance is specified as a percentage of the smallest value compared. By default, the tolerance is set at 1e-5% which is equivalent to performing exact matches. To change the tolerance, use the `set_rule_property` command.

Methodology

Use the following methodology to compare constraint sets:

1. Use the `create_scenario` command to model your original constraint set and your modified constraint set.

Your constraints can be in SDC files or in multiple Tcl scripts.

2. Use the `compare_constraints` command to compare the scenarios. For example,

```
ptc_shell> compare_constraints -constraints1 \
               my_original_constraint_scenarios \
               -constraints2 my_modified_constraint_scenarios
```

3. Use the SDC-to-SDC browser to identify violations. Use the `report_constraint_analysis` command to obtain constraint comparison text reports.

4. Fix the violations in the following order:

- a. Clock constraint violations
- b. `set_disable_timing` violations
- c. `set_case_analysis` violations
- d. Exception violations

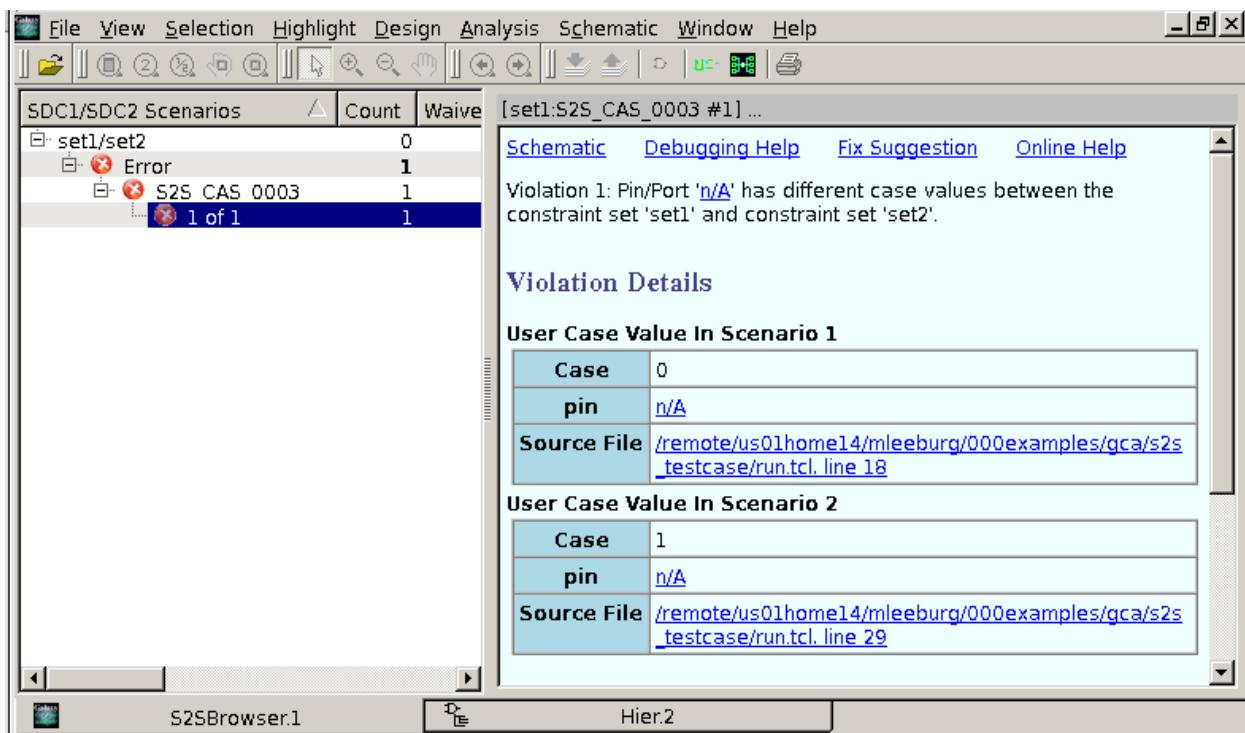
5. Repeat the `compare_constraints` command as needed.

Correlation Consistency Violation Browser

After you compare your original constraints to your modified constraints, use the correlation consistency (S2S) browser, shown in [Figure 252](#), to identify and fix constraint mismatches.

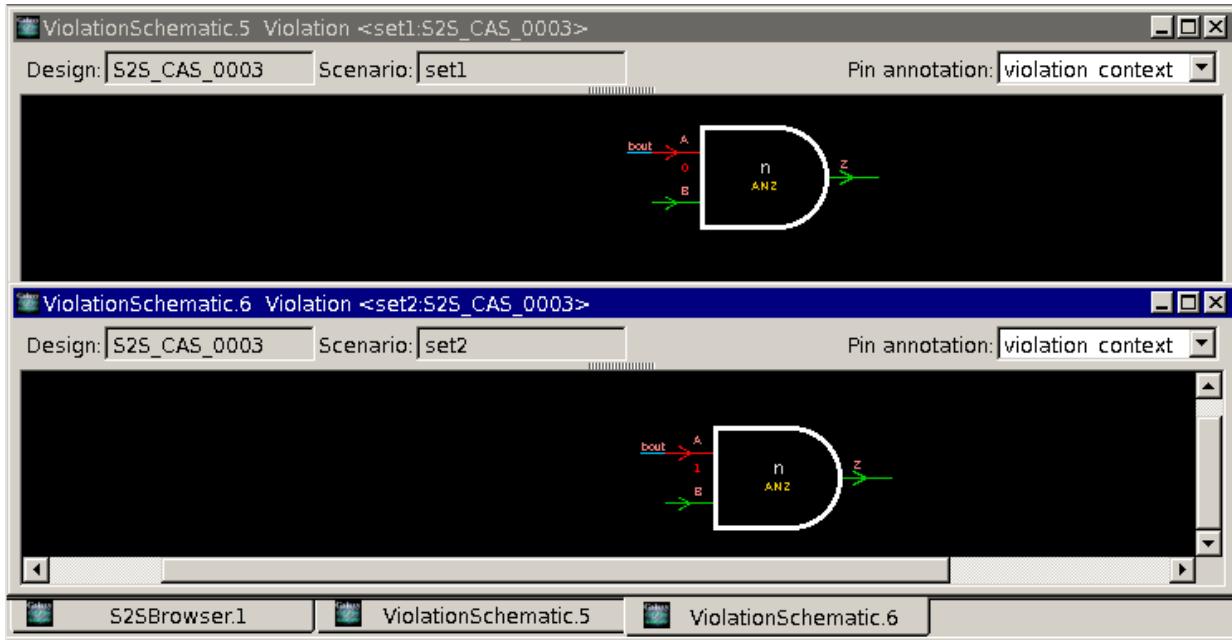
The correlation consistency violation browser is similar to the general violation browser used with the `analyze_design` command. All links in the information view pane function similarly to the links in the Hierarchical consistency violation browser and the general violation browser. If schematics are appropriate for the violation, the information view pane contains a schematic link.

Figure 252 Correlation Consistency (S2S) Violation Browser



When working with correlation consistency constraint violations, schematics for both sets of constraints are provided and can be displayed side by side, as shown in [Figure 253](#). Click the Schematic link to display the schematics. The schematics are annotated with data to help debug the constraint problem.

Figure 253 Correlation Consistency (S2S) Schematic Display

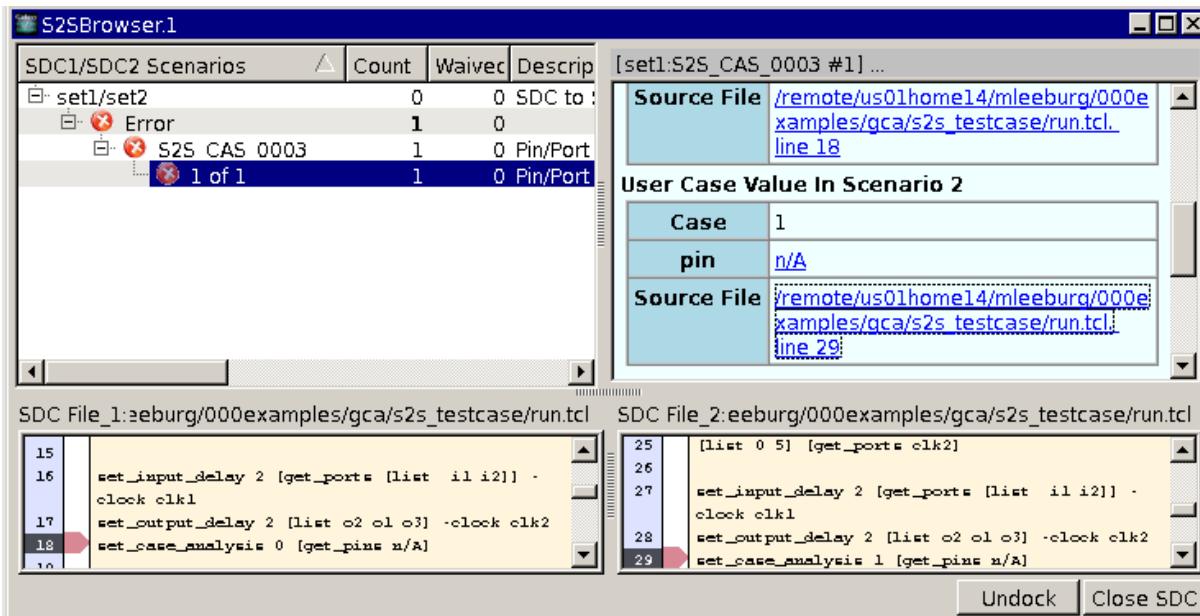


Correlation consistency checking can also display the constraints that caused the violation from each respective SDC file, as shown in [Figure 254](#).

The SDC links are located in the Violations Details section of the information pane. Select the source file link for scenario 1 and scenario 2 to see the relevant constraints. You are not limited to displaying only schematics or SDC information; correlation consistency checking can display both schematics and SDC file contents simultaneously.

To debug correlation consistency violations, analyze the schematics and the constraint definitions in the SDC files. Follow the guidelines in the “Debugging Help” section in the information pane and use the fix suggestion described in the information pane. Additional resources, such as man pages and user guides are available in the online Help.

Figure 254 Correlation Consistency (S2S) Scenarios



Rules and Violations When Comparing Constraint Sets

You can compare constraints that affect datapath propagation but you cannot compare constraints that affect clock network propagation. You cannot compare a subset of constraint types. However, constraint comparison violations can be disabled. When disabled, violations do not appear in the GUI or the text reports.

Creating Waivers

You create waivers for correlation consistency rules through the GUI or command line. For more information about creating, editing, removing, and reporting waivers, see “[Suppressing Violations](#)”.

Text Reports

Use the `report_constraint_analysis` command to obtain constraint comparison text reports, as shown in [Example 60](#). Use the correlation consistency violation browser to view violations interactively. You can view where the mismatch occurs in the schematics and where the mismatch occurs in the SDC constraints file. See “[Suppressing Violations](#)” for more information.

Example 60 Report Generated by the report_constraint_analysis Command for the compare_constraints Constraint

```
ptc_shell> report_constraint_analysis
*****
```

```

        -include {violations user_messages rule_info}
        -style {full}
Version: ...
Date  : ...
*****
Constraint Comparison Violations
Count Waived Description
-----
set1/set2          1   0
    error          1   0
        S2S_CAS_0003  1   0      Pin/Port 'pin_port' has different case
    values
between the constraint set 'scenario1' and constraint set 'scenario2'.
    1 of 1          0      Pin/Port 'n/A' has different case
    values
between the constraint set 'set1' and constraint set 'set2'.
-----
Total Error Messages  1   0
Total Warning Messages 0   0
Total Info Messages    0   0

```

User Messages

MessageID	Count	Description	
Information	5		
ADES-016	1	waiver '%s' removed.	
1 of 1	1	waiver 'waiver_0' removed.	
CMP-006	3	Reading %s configuration file: %s	
1 of 3	1	Reading PrimeTime configuration file: /remote/srm	
489/			
image/path_name/nightly/zinmgmz_in_dev/110423.1/Testing/auxx/path_name/tcl/primeime.pcx			
2 of 3	1	Reading IC Compiler configuration file: /remote/s	
rm489/			
image/path_NAME/nightly/zin_zin_dev/110423.1/Testing/auxx/path_name/tcl/icc.pcx			
3 of 3	1	Reading Design Compiler configuration file: /remo	
te/			
srm489/image/path_name/nightlynmgr_zin_dev/110423.1/Testing/auxx/path_name/t			
cl/syn.pcx			
LNK-040	1	%units loaded from library '%s'	
1 of 1	1	units loaded from library 'lsi_10k'	
Rule Information			
Rule	Severity	Status	Message
...			

Comparing Two Designs With Two Sets of Design Constraints

After you link two functionally equal designs to compare and load the constraints sets, you must provide a mapping of objects in a Name Map file (NMF) before running the `compare_constraints` command. When you run the `compare_constraints` command, use the `-use_clock_map` option to reference your custom clock mapping file.

This feature supports all correlation consistency equivalence checks that are available when using one design and comparing two sets of design constraints.

This feature requires a PrimeTime-ELT license.

The following sections are covered:

- [Methodology](#)
- [Linking the Designs and Loading the Constraints Sets](#)
- [Object Mapping With the Name Map File](#)
- [Creating a Custom Clock Mapping File](#)
- [Example Script](#)

Methodology

Use the following methodology to compare two designs and the constraints set:

1. Link the designs and load the constraints set

See [Linking the Designs and Loading the Constraints Sets](#).

2. Specify the object mapping with the Name Map file (NMF)

See [Object Mapping With the Name Map File](#).

3. Use the `compare_constraints` command to perform constraints equivalence checking between the two designs and the constraints scenarios. For example,

```
compare_constraints -constraints1 scenario_list \
-constraints2 scenario_list [-design2 design_name] \
[-use_clock_map custom_mapping_file]
```

To create a custom mapping file, see [Creating a Custom Clock Mapping File](#).

For information on the `compare_constraints` command, see [Comparing Two Sets of Design Constraints](#).

4. Use the `report_constraints_analysis` command to obtain constraint comparison text reports.

See [Text Reports](#).

5. Fix any violations.

6. Repeat the `compare_constraints` command as needed.

Linking the Designs and Loading the Constraints Sets

You can link two functionally equivalent designs and load the two constraints set in the same session. Before you can compare the two designs, verify that they are formally equivalent. The two designs must have different names.

To link the second design using the latest subdesigns, use the `-latest` option with the `link_design` command. For example,

```
% link_design -add -latest design2
```

To perform constraint equivalence checking between the two designs, use the `-design2` option with the `compare_constraints` command to specify your second design. The `compare_constraints` command compares the current design with your second design.

Object Mapping With the Name Map File

The name map file (NMF) contains the `define_name_maps` command and the name of objects to specify object mapping. The command and options are:

```
% define_name_maps      # define name mapping for the design objects
  -application        # specify the generated application
  -design_name        # the design name
  -columns            # the column names
```

You can specify the following object types: pins, ports, nets, and cells (including hierarchical cells). The object type is mapped as a “class” in the `-columns` option.

The NMF is the only format recognized by the `compare_constraints` command. Any other object mapping format must be converted to NMF before running the `compare_constraints` command.

The following are some examples of defining name maps.

Defining a name map to specify an object due to cloning is shown in [Example 61](#). The REG1 cell is cloned to the REG1_Clone cell and the name mapping is provided in the example.

Example 61 Defining a Name Map to Specify an Object Due to Cloning

```
% define_name_maps -application golden_sdc -design_name design2 \
  -columns {class pattern option names} \
  [list cell REG1 [list] [list] [list REG1]] \
  [list cell REG1 [list] [list] [list REG1_Clone]]
```

Defining a name map to specify an object change due to hierarchical transformations is shown in [Example 62](#). Given the following scenario:

U1 to Wrapper/U1: Leaf Cell of U1 moved within Wrapper hierarchy

SUB01 to Wrapper/SUB01: SUB01 hierarchy moved within Wrapper hierarchy

SUB25/SUB22/FF01 to SUB25/SUB21/FF01: FF01 moved from SUB22 to SUB21

/SUB40 to /SUB40/SUB41: SUB40 hierarchy moved to SUB40/SUB41

Example 62 Defining a Name Map to Specify an Object Changed Due to Hierarchical Transformations

```
% define_name_maps -application golden_sdc -design_name design2 \
    -columns {class pattern options names} \
    [list cell U1 [list] [list Wrapper/U1]] \
    [list cell SUB01 [list] [list Wrapper/ SUB01]] \
    [list cell SUB25/SUB22/FF01 [list] [list SUB25/SUB21/FF01]] \
    [list cell SUB40 [list] [list /SUB40/SUB41]]
```

Defining a name map to specify an object change due to multibit flip-flop banking is shown in [Example 63](#). The FF1 and FF2 flip-flops are merged into a single multibit flip-flop, FF_12. The CP pins and D pins must be mentioned in the name mapping.

Example 63 Defining a Name Map to Specify an Object Change Due to Multibit Flip-Flop Banking

```
% define_name_maps -application golden_sdc -design_name design2 \
    -columns {class pattern options names} \
    [list pin FF1/CP [list] [list FF12_CP]] \
    [list pin FF2/CP [list] [list FF12_CP]] \
    [list pin FF1/D [list] [list FF12_D0]] \
    [list pin FF2/D [list] [list FF12_D1]] \
```

After you define the name map, you can run the `compare_constraints` command to compare the two sets of constraints.

Creating a Custom Clock Mapping File

When you run the `compare_constraints` command, you can create a custom clock mapping file. To create your custom clock mapping between two scenarios, use the `set_clock_map` command. Use the `-design2` option with the `set_clock_map` command to specify your second design. The following table lists the available options for the `set_clock_map` command.

Option	Description
<code>-clocks1 {clock_list}</code>	List of clocks in the first scenario
<code>-clocks2 {clock_list}</code>	List of clocks in the second scenario
<code>-scenario1 first_scenario</code>	Name of the first scenario
<code>-scenario2 second_scenario</code>	Name of the second scenario
<code>-design2 design2</code>	Name of the second design

For example,

```
% set_clock_map -clocks1 gclk1_hier -clocks2 gclk1_buf \
    -design2 design2 sdc1_model -scenario2 sdc2_model
```

Example Script

The following is an example script that shows linking the designs, loading the constraints, defining the name map file, and comparing the constraints sets.

```
set link_library "* tcbn90gwc.db"

read_verilog ./design1.v                                #link design1
current_design design1
link

read_verilog ./design2.v                                #link design2
current_design design2
link_design -add -latest design2
current_design design1                                #load constraints 1
source sdc1.tcl

current_design design2                                #load constraints 2
source sdc2.tcl

define_name_maps \
    -application golden_sdc \
    -design_name design2 \
    -columns {class pattern options names} \
        [list port IN [list] [list IN_2]] \
        [list port OUT [list] [list OUT_2]] \
        [list cell BLK1 [list] [list BLK2]]

current_design design1                                #compare constraints
compare_constraints -constraints1 default \
    -constraints2 default -design2 design2
```

Additional Analysis Features

This topic describes additional features that help you to diagnose constraint problems and to understand the impact of constraints. These features are described in the following sections:

- [analyze_paths Command](#)
- [analyze_unclocked_pins Command](#)
- [analyze_clock_networks Command](#)
- [report_case_details Command](#)

- `report_clock_crossing` Command
- `report_analysis_coverage` Command
- `report_exceptions` Command (Redundant Constraints)

analyze_paths Command

You can debug most constraint problems using the `analyze_paths` command. However, some problems require detailed timing path analysis. When the `report_timing` command in the PrimeTime, Design Compiler, or IC Compiler tools does not show you certain timing paths, it is hard to find out what is causing a problem, and whether it is an expected condition or a constraint problem. In such cases, use the `analyze_paths` command to find out what is blocking propagation of particular timing paths and the cause of the blockage.

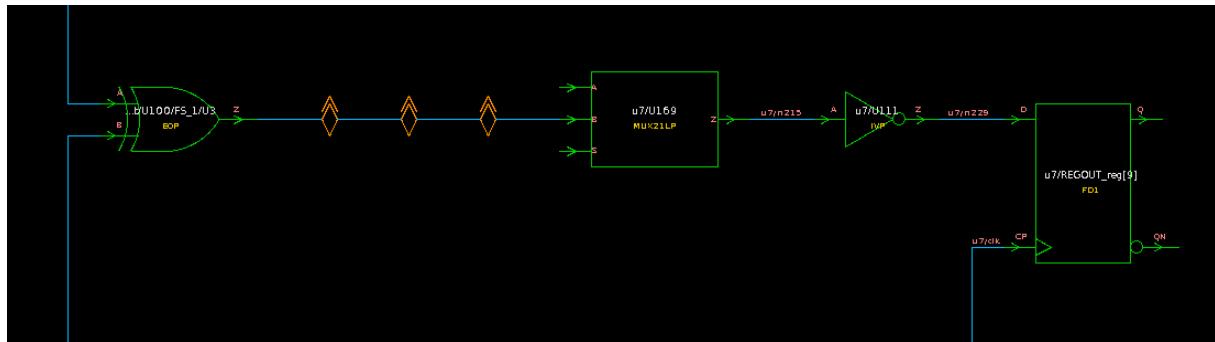
The `analyze_paths` command helps you to

- Identify timing exceptions and constraints on particular timing paths in a design
- Find out how many timing paths are covered, given a particular condition of rise/fall and from/through/to for an object set

The `analyze_paths` command identifies pins involved in specified paths and ranks the information based on the number of timing paths affected by various constraints.

If the GUI is open and you use the `-path_type full` argument, the `analyze_paths` command generates both a text report and a schematic showing the paths that satisfy the specifications you provide as arguments to the command. An example of a schematic is shown in Figure 255.

Figure 255 Paths Identified by the `analyze_paths` Command



You can use the `-max_endpoints` option to specify the maximum number of paths displayed in the schematic.

If the GUI is not open or you use the `-text_only` option, the `analyze_paths` command generates only a text report. Examples of text reports are shown in the remainder of this section.

The analyze_paths Default Output Report

You can use the `analyze_paths` command to see how many paths there are from, through, or to a particular set of objects and the corresponding clock interaction.

[Example 64](#) shows a default output report.

Example 64 Default Report Generated by the analyze_paths Command

```
ptc_shell> analyze_paths -to Tranx/pay_count_reg[1]/D
Updating the exception annotations on the graph
*****
Report : analyze_paths
Design : b_top
Version: . . .
Date   : : : .
*****
Path Endpoints:
Endpoint          Constraints      Path      Clocks
-----          Count      Launch/Capture
Tranx/pay_count_reg[1]/D           10      (set 0)/(set 0)
Tranx/pay_count_reg[1]/D           8       sysclk/(set 0)

Sets of Launching and Capturing Clocks:
set 0 (2 clocks):
  falling sys_clk
  falling sysclk
1
```

The analyze_paths Detailed Output Report

You can use the `analyze_paths -path_type full` command to see how many paths there are from, through, or to a particular set of objects and the corresponding clock interaction with the breakdown of startpoints and endpoints. In addition, the report includes detailed logic levels and through points with possible startpoints and endpoints. See [Example 65](#).

When timing exceptions are set on these paths, as in [Example 66 on page 756](#), the report shows

- The name of the object that the exception is set on
- The name of the timing exception applied
- The script name and line location of the exception

Example 65 Report Generated by the analyze_paths -path_type full Command

```
ptc_shell> analyze_paths -to Tranx/pay_count_reg[1]/D -path_type full
Breaking loops for scenario: system
*****
Report : analyze_paths
Design : b_top
Version: . . .
Date   : : : .
*****
```

Chapter 18: Constraint Consistency
Additional Analysis Features

Summary of Paths:

Path Clocks	Startpoint	Endpoint	Constraints Count	Launch/Capture
	Tranx/pay_count__reg[1]/CPN	Tranx/pay_count__reg[1]/D	6	(2 clocks) / (2 clocks)
enable		Tranx/pay_count__reg[1]/D	4	sysclk/ (2 clocks)
	Tranx/pay_count__reg[0]/CPN	Tranx/pay_count__reg[1]/D	4	(2 clocks) / (2 clocks)
init		Tranx/pay_count__reg[1]/D	4	sysclk/ (2 clocks)

Full report of all pins in the paths

Level	Pin	Attr	Constraints
1	init		Start
1	Tranx/pay_count__reg[0]/CPN		Start
1	enable		Start
1	Tranx/pay_count__reg[1]/CPN		Start
2	Tranx/U146/A		
2	Tranx/pay_count__reg[1]/Q		
2	Tranx/pay_count__reg[0]/Q		
2	Tranx/U925/A		
3	Tranx/U925/Z		
3	Tranx/U146/Z		
3	Tranx/add_67/U1_1_1/A		
3	Tranx/add_67/U1_1_1/B		
4	Tranx/add_67/U1_1_1/S		
2	Tranx/U1108/A		
4	Tranx/U1108/B		
4	Tranx/U384/A		
4	Tranx/U384/B		
5	Tranx/U1108/Z		
5	Tranx/U384/Z		
6	Tranx/U1107/A		
6	Tranx/U383/A		
7	Tranx/U1107/Z		
7	Tranx/U383/Z		
3	Tranx/U1132/A		
8	Tranx/U1132/B		
5	Tranx/U1132/C		
8	Tranx/U1132/D		
9	Tranx/U1132/Z		
10	Tranx/U920/A		
11	Tranx/U920/Z		
12	Tranx/pay_count__reg[1]/D		End
1			

Example 66 Report Generated by the analyze_paths -path_type full Command

```
ptc_shell> analyze_paths -path_type full -to Tranx/CRC_tranx/pay_out_reg/D
*****
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
*****
Summary of Paths:
Path Clocks                                         Constraints   Launch/

```

Startpoint	Endpoint	Count	Capture
Tranx/CRC_tranx/crc_out_reg/CPN	Tranx/CRC_tranx/pay_out_reg/D	F1 2	(2 clocks) / (2 clocks)
Tranx/CRC_tranx/pay_in2_reg/CPN	Tranx/CRC_tranx/pay_out_reg/D	M0 2	(2 clocks) / (2 clocks)
enable	Tranx/CRC_tranx/pay_out_reg/D	2	sysclk/ (2 clocks)

Exception Information:
M0 multicycle_path (run.ztcl, line 27)
F1 false_path (run.ztcl, line 28)

Full report of all pins in the paths

Level	Pin	Attr	Constraints
1	enable	Start	
1	Tranx/CRC_tranx/pay_in2_reg/CPN	Start	
1	Tranx/CRC_tranx/crc_out_reg/CPN	Start	
2	Tranx/CRC_tranx/crc_out_reg/Q		F1
2	Tranx/CRC_tranx/pay_in2_reg/Q		M0
3	Tranx/CRC_tranx/U207A		
3	Tranx/CRC_tranx/U20/C		
2	Tranx/CRC_tranx/U20/D		
4	Tranx/CRC_tranx/U20/Z		
5	Tranx/CRC_tranx/U14/A		
6	Tranx/CRC_tranx/U14/Z		
7	Tranx/CRC_tranx/pay_out_reg/D	End	M0, F1
1			

Summary Output Report for the analyze_paths Command

[Example 67](#) shows the analyze_paths -path_type summary report.

Example 67 Summary Output Report for analyze_paths Command

```
ptc_shell> analyze_paths -to Tranx/CRC_tranx/pay_out_reg/D -path_type summary
*****
Report : analyze_paths
Design : b_top
Version: . . .
Date   : : : .
*****
Summary of Paths:
Path Clocks
Path Clocks
Startpoint          Endpoint      Constraints    Launch/
Startpoint          Endpoint      Count         Capture
-----
Tranx/CRC_tranx/crc_out_reg/CPN Tranx/CRC_tranx/pay_out_reg/D
                                         F1 2      (2 clocks)/
Tranx/CRC_tranx/pay_in2_reg/CPN Tranx/CRC_tranx/pay_out_reg/D
                                         M0 2      (2 clocks)/
```

```

enable           Tranx/CRC_trnx/pay_out_reg/D 2      (2 clocks)
                sysclk/ (2 clocks)

Exception Information:
M0 multicycle_path      (run.ztcl, line 27)
F1 false_path          (run.ztcl, line 28)
1

```

Disabled Paths Output Report for the analyze_paths Command

When using the `analyze_paths` command with the `-traverse_disabled` option (see [Example 68](#)), the output report analysis traces through disabled paths of the given path set and shows

- Possible paths without data propagation blockage
- Reasons for datapaths being blocked or segmented

Example 68 Report for analyze_paths Command With the -traverse_disabled Option

```

ptc_shell> analyze_paths -to Tranx/CRC_trnx/pay_out_reg/D -path_type full \
-traverse_disabled
*****
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
*****
Summary of Paths:
Path Clocks          Endpoint          Constraints Count Launch/ Capture
Startpoint
-----
enable               Tranx/CRC_trnx/pay_out_reg/D 2      sysclk/ (2 clocks)
enable               Tranx/CRC_trnx/pay_out_reg/D D0 2      sysclk/ (2 clocks)
Tranx/CRC_trnx/crc_out_reg/CPN Tranx/CRC_trnx/pay_out_reg/D F1 2      (2 clocks) / (2 clocks)
Tranx/CRC_trnx/pay_in2_reg/CPN Tranx/CRC_trnx/pay_out_reg/D M0 2      (2 clocks) / (2 clocks)
Exception Information:
M0 multicycle_path      (run.ztcl, line 27)
F1 false_path          (run.ztcl, line 28)

Disabled Object Information:
D0:
  set_disable_timing negative_unate Arc
    From: Tranx/CRC_trnx/U567A
    To:  Tranx/CRC_trnx/U56/Z

Full report of all pins in the paths

```

Level	Pin	Attr	Constraints
1	Tranx/CRC_tranx/pay_in2_reg/CPN	Start	
1	Tranx/CRC_tranx/crc_out_reg/CPN	Start	
1	enable	Start	
2	Tranx/CRC_tranx/crc_out_reg/Q		F1
2	Tranx/CRC_tranx/pay_in2_reg/Q		M0
2	Tranx/CRC_tranx/U567A		
3	Tranx/CRC_tranx/U56/Z		
3	Tranx/CRC_tranx/U20/A		
4	Tranx/CRC_tranx/U20/B		
3	Tranx/CRC_tranx/U20/C		
2	Tranx/CRC_tranx/U20/D		
5	Tranx/CRC_tranx/U20/Z		
6	Tranx/CRC_tranx/U14/A		
7	Tranx/CRC_tranx/U14/Z		
8	Tranx/CRC_tranx/pay_out_reg/D	End	M0, F1
1			

analyze_unclocked_pins Command

Use the `analyze_unclocked_pins` command to analyze a collection of pins for missing clock definitions or blocked clock propagation. By default, this command analyzes all pins in the design.

The `analyze_unclocked_pins` command by default reports

- Summary of analysis
- Possible points for missing clock definition
- Locations where clock propagation is blocked

Use the `-verbose` option to print out clock pins that are unclocked, case-disabled, and register-disabled. This option reports detailed lists of unclocked pins. From the output report, you can see register clock pins that do not have clock signals, and clock pins that have sequential checks disabled for various reasons. These are helpful for further analysis.

In addition to using the `analyze_unclocked_pins` command to help debug clock constraint problems, you can turn clock and case annotation on and off in the Schematic View using the pin annotation drop-down menu. See “[Analyzing a Violation](#)” for more information.

The `analyze_unclocked_pins` Verbose Output Report

[Example 69](#) shows the output for the `analyze_unclocked_pins -verbose` command.

The report summarizes register clock pin status as being clocked, unclocked, case-disabled, or register-disabled. Case-disabled is caused by `case_analysis` propagation

disabling timing check/arcs from these pins. Register-disabled is caused by various set_disable_timing constraints applied to all arcs from these pins.

In addition to the summary, the verbose report identifies possible startpoints for debugging missing clock definitions. It also identifies clocks that are blocked from register clock pins at specified locations. Check to see if the clock propagation blockage is intentional.

Example 69 Output Report for the analyze_unclocked_pins -verbose Command

```
ptc shell> analyze unclocked_pins -verbose
*****
Report : analyze_unclocked_pins
          -verbose
Design : ChipLevel
Version: ...
Date   : ...
*****
Analyze Register Clock Pins that do not have clocks assigned

      Summary
-----
Register Clock Pin Status           Number of Pins
-----
Clocked                           148
Unclocked                         67
Case disabled clock pin           0
Register behavior disabled        0
-----
Total register clock pins          215

      Possible Startpoints for Missing Clock Definitions
      Number
      Pin/Port Startpoints       unclocked    Reason a Startpoint
-----
clk_adder_div2/Q                  50          constraint forced startpoint
                                         on interior pin (default)
clk_8x8/Z                          16          constraint forced startpoint
                                         on interior pin (default)
clk_adder/Z                        1           set_disable_timing (default)

      clk_adder/Z
      - fans out to 1 unclocked pins
        In fanin of clk_adder_div2/CP
      clk_adder_div2/Q
      - fans out to 50 unclocked pins
        In fanin of u1/ain_tmp_reg[15]/CP
      ...

Clocks are blocked from register clock pins at the following locations:
      Number
      Clock     Blocking Pin       unclocked    Reason
-----
mclk      clk_8x8/Z             16          constraint forced
                                         startpoint on interior pin
aclk      clk_adder/Z            1           set_disable_timing

Clock mclk Blocking Locations (verbose)
Blocking Pin                    Reason
```

```
-----  
clk_8x8/Z           constraint forced startpoint on interior pin
```

Verbose Output for Blocked Clocks:

Blocking Pin	Clock Pin
clk_8x8/Z	u4/res_reg[15]/CP
clk_8x8/Z	u4/res_reg[12]/CP
...	
Clock aclk Blocking Locations (verbose)	
Blocking Pin	Reason
clk_adder/Z	set_disable_timing

Verbose Output for Blocked Clocks:

Blocking Pin	Clock Pin
clk_adder/Z	clk_adder_div2/CP
1	
ptc_shell>	
.....	

analyze_clock_networks Command

In modern designs, clock networks grow more complex due to features such as generated clocks, clock-gating logic, and clock multiplexers. These structures can be difficult to analyze. In addition, the interaction of the clock network traversal with constraints that have an impact on clock networks can be very hard to understand. To debug problems in complex clock networks, constraint consistency uses the `analyze_clock_networks` command to generate reports that describe the propagation of the clock network to and from specified points. A report can show the impact of constraints on

- The clock network
- The potential clock network (potential senses are detected with the `-traverse_disabled` option)
- A generated clock's source latency network

In addition to using the `analyze_clock_networks` command to help debug clock constraint problems, you can turn clock and case annotation on and off in the Schematic View using the Pin annotation drop-down menu. See “[Analyzing a Violation](#)” for more information.

Output Reports for the `analyze_clock_networks` Command

[Example 70](#) and [Example 71](#) show the `analyze_clock_networks` command usage and output reports.

Example 70 Output Report With Potential Network for the analyze_clock_networks Command

```
ptc_shell> analyze_clock_networks -from [get_clocks sysclk] -to Tranx/U0/ \
o_lfsrl_reg[11]/CP -traverse_disabled -style full_expanded
*****
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full_expanded
        -end_types {register port clock_source }
        -traverse_disabled
Design : b_top
. . .
*****
Key for clock sense abbreviations:
  P = positive
  N = negative
  Potential senses detected with -traverse_disabled:
  [P] = potential positive

Full report for clock: sysclk

Branch 0:
  Branch
Level  Info      Sense Notes      Port/Pin
-----
---  

  0      P      source    clk
  1      P      Tranx/U1775/A
  2      P      Tranx/U1775/Z
  3      P      Tranx/U1772/A
  4      N      Tranx/U1772/Z
  5      N      Tranx/U1771/A
  6      N      Tranx/U1771/Z
  7      N      Tranx/U1417/A
  8      P      Tranx/U1417/Z
  9      P      Tranx/U0/U1154/A
  10     P      Tranx/U0/U1154/Z
  11     P      Tranx/U0/U1139/A
  12     P      Tranx/U0/U1139/Z
  13     P      Tranx/U0/U1121/A
  14     [P]    DT#0   Tranx/U0/U1121/Z
  15     [P]    R      Tranx/U0/o_lfsrl_reg[11]/CP
```

Notes Column Information:

Clock Network End Type Abbreviations:
R = register

Clock Blocking Constraints:
DT#0 at pin: Tranx/U0/U1121/Z
set_disable_timing
run.ztcl, line 25
1

Example 71 Output Report With Generated Clock Path for the analyze_clock_networks Command

```
ptc_shell> analyze_clock_networks -to gclk -traverse_disabled -style full_expanded
```

```
*****
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full_expanded
        -end_types {register port clock_source }
        -traverse_disabled
Design : b_top
Version: . . .
Date   : . . .
*****
Key for clock sense abbreviations:
  P = positive
  R = rise_triggered
  N,R = negative, rise_triggered

Source latency paths for generated clock: gclk
from master clock: sys_clk - 2 partial path branches

Branch 0:
  Branch
  Level Info Sense Notes Port/Pin
-----
  0      P    source   clk
  1      P          Recx/U207/A
  2  S1   P          Recx/U207/Z
  3      P          Recx/CRC2/U69/A
  4      P          Recx/CRC2/U69/Z
  5      P    CG       Recx/CRC2/clk_and/A
  6  E1   N,R   CS       Recx/CRC2/clk_and/Z

Branch 1: from branch 0 reconverges to branch 0
  Branch
  Level Info Sense Notes Port/Pin
-----
  3      P    R       Recx/CRC2/clk_div/CP
  4      P          Recx/CRC2/clk_div/CP
  5      R          Recx/CRC2/clk_div/QN
  6      R    D       Recx/CRC2/clk_and/B

Notes Column Information:

Clock Network End Type Abbreviations:
  R = register
  CS = clock_source
  D = data
  CG = clock_gating
1
```

report_case_details Command

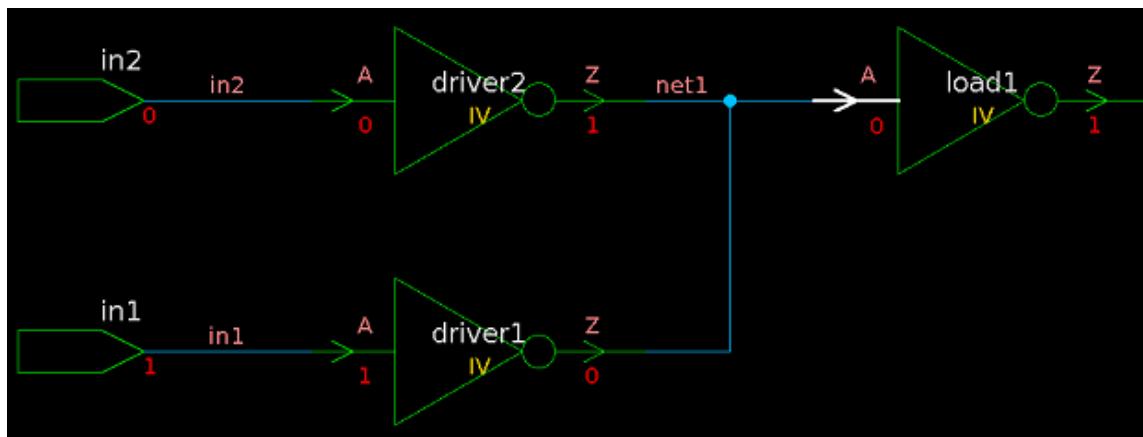
With complex design structures, analyzing case value and logic constant propagation can become very difficult. This analysis can become a very complicated task because case and constant propagation are controlled in a number of different ways, such as user-set values, constraints on the design, interaction between different case and constant values propagated to a gate, and by how the logic gate is modeled functionally.

To see how your case values and logic constant values propagate throughout the netlist, use the `report_case_details` command. This command reports the propagation of case values and logic constant values throughout your design. It can show how case values and logic constants propagate to or from specified pins and ports.

In addition to using the `report_case_details` command to help debug case propagation problems, you can turn clock and case annotation on and off in the Schematic View using the Pin annotation drop-down menu. See “[Analyzing a Violation](#)” for more information.

If the GUI is open, the `report_case_details` command generates both a text report and a schematic showing a cone of logic with propagated values annotated on the cells, as shown in [Figure 256](#).

Figure 256 Schematic Showing Case Propagation



The `-sources` option lists the ports and pins where case values or logic constants have been set and propagated to the list of ports and pins being examined. If you run the command with the `-sources` option and without a `-from` or `-to` option, a schematic is not generated because the cone of logic related to the propagated value is not reported. You can specify the maximum number of gates to be displayed in the schematic with the `-max_objects` option; if the logic cone is larger than this value, no schematic is generated.

If the GUI is not open or you use the `-text_only` option, the `report_case_details` command generates only a text report. An example of a text report is shown in [Example 72](#).

Example 72 Text Report Generated by the report_case_details Command

```
ptc shell> report_case_details -to Tranx/my_fec_encoder/curr_state_reg[1]/D
*****
Report : case_details
Design  : b_top
Version: . . .
Date   : . . .
```

```
*****
Properties      Value   Pin/Port
-----
from user case  0       Tranx/my_fec_encoder/curr_state_reg[1]/D

Case fanin report:
Verbose Source Trace for pin/port Tranx/my_fec_encoder/curr_state_reg[1]/D:
Path number: 1
Path Ref #  Value  Properties          Pin/Port
-----
0
Tranx/my_fec_encoder/curr_state_reg[1]/D
  0      F()=A ! B ! & C ! D ! & +
                                                Tranx/my_fec_encoder/U37/Z
2      1
                                                Tranx/my_fec_encoder/U37/B
3      1
                                                Tranx/my_fec_encoder/U37/D

Path number: 2
Path Ref #  Value  Properties          Pin/Port
-----
1      1      F()=A ! B ! C ! D ! + + +
                                                Tranx/my_fec_encoder/U37/B
  0
                                                Tranx/my_fec_encoder/U36/Z
  0      F()=A B C & &
                                                Tranx/my_fec_encoder/U36/D
  0
                                                Tranx/my_fec_encoder/U50/Z
  0      F()=A !
                                                Tranx/my_fec_encoder/U50/C
  0
                                                Tranx/U1146/Z
  1
                                                Tranx/U1146/A
  1      F()=A
                                                Tranx/U1122/Z
  1
                                                Tranx/U1122/A
  1      F()=A ! B ! C ! + +
                                                Tranx/U1215/Z
  0
                                                Tranx/U1215/B
  0      F()=A !
                                                Tranx/U1213/Z
  1
                                                Tranx/U1213/A
  1      F()=A ! B ! +
                                                Tranx/U1212/Z
  0
                                                Tranx/U1212/B
  0      F()=A ! B ! &
                                                Tranx/U1211/Z
  1
                                                Tranx/U1211/A
  1      F()=A
                                                Tranx/U151/Z
  1
                                                Tranx/U151/A
  1      F()=A
                                                Tranx/U1103/Z
  1
                                                Tranx/U1103/A
  1      user case
                                                rst

Path number: 3
Path Ref #  Value  Properties          Pin/Port
-----
1
  1      F()=A ! B ! +
                                                Tranx/my_fec_encoder/U37/D
  0      user case
                                                Tranx/my_fec_encoder/U49/Z
                                                Tranx/my_fec_encoder/U49/B
```

report_clock_crossing Command

The `report_clock_crossing` command reports the various interactions between clock domains in a text report. By default, it lists all clock combinations in which there is a topological path launched by one clock and captured by another one. Clock combinations that have been declared partially or completely false are noted. The report can be

narrowed by providing a collection of `-from` and `-to` clocks. It can also be narrowed to only reporting the completely false paths or interactions with some valid paths.

report_analysis_coverage Command

Many designs require complex and numerous constraints. The need for multimode and multicorner analysis increases the importance of constraint quality and completeness. The task of manually quantifying the completeness of a set of constraints is difficult and time consuming. To help automate this task, use the `report_analysis_coverage` command, which enables you to gauge the completeness of a given set of constraints.

This section describes constraint consistency capabilities in the following subsections:

- [Analysis Coverage Overview](#)
- [report_analysis_coverage in Single-Scenario Runs](#)
- [Narrowing Down Reports](#)
- [report_analysis_coverage in Multi-Scenario Runs](#)

Analysis Coverage Overview

Use the `report_analysis_coverage` command to analyze constraint coverage. When you run the `report_analysis_coverage` command, constraint consistency looks at the endpoints and timing arcs in the design across all the design scenarios and reports a summary of all tested and untested points for the current constraints. After you review the summary, you can use the `report_analysis_coverage` command options to debug specific problems. These options enable you to obtain detailed information about the status of an endpoint. Detailed report information includes tested and untested endpoints, the reason why an endpoint is untested, what mode the endpoint is tested in, and so forth. These debugging features help speed-up constraint development.

Constraint consistency supports both single-scenario and multi-scenario environments. A scenario is used to represent the constraints for an operating mode along with a particular process, voltage, and temperature (PVT) corner. Different PVT corners only affect delays and transition times in the design; they have no effect on whether the endpoints are constrained or not. It is important to ensure that the constraints of different scenarios cover the design for all expected modes of operation. The valid check types for analysis coverage are:

- `clock_gating_hold` and `clock_gating_setup`
- `hold` and `setup`
- `clock_separation`
- `max_skew`

- min_pulse_width
- min_period
- recovery
- removal

Constraint consistency generates a text report with the percentage and number of checks tested and untested.

report_analysis_coverage in Single-Scenario Runs

If there is only one scenario, constraint consistency automatically generates a report focusing on single-scenario analysis. This report identifies whether an endpoint or timing arc has been tested. If you define several scenarios in the current session, but you only want to generate a report for one of them, use the `-scenario` option. [Example 73](#) shows a standard single-scenario report.

Example 73 Standard Single-Scenario Report

```
report_analysis_coverage
*****
Report : analysis_coverage
Design : zDesign
. . .
*****
Breaking loops for scenario: default
Updating Clock Gating Locations
    Inferring 294 clock-gating checks.

Type of Check      Total      Tested      Untested
-----
clock_gating_hold      294      294 (100%)      0 ( 0%)
clock_gating_setup      294      294 (100%)      0 ( 0%)
hold                  75481     45553 ( 60%)     29928 ( 40%)
min_period             376      196 ( 52%)      180 ( 48%)
min_pulse_width        55778     39171 ( 70%)     16607 ( 30%)
out_hold               359      333 ( 93%)       26 ( 7%)
out_setup               359      333 ( 93%)       26 ( 7%)
recovery              16801     16385 ( 98%)      416 ( 2%)
removal                16801     16385 ( 98%)      416 ( 2%)
setup                 75479     45532 ( 60%)     29947 ( 40%)
-----
All Checks            242022    164476 ( 68%)    77546 ( 32%)
```

[Example 73](#) shows a summary report. After you identify some potential problems in the constraint coverage, constraint consistency can provide more detailed information for specific checks, as shown in the next section.

Narrowing Down Reports

In [Example 73](#), some of the outputs have untested setup and hold checks. To find out more information about why these outputs are untested, run the following command:

```
ptc_shell> report_analysis_coverage -status_details untested \
          -check_type [list out_setup out_hold]
```

[Example 74](#) shows the output, which provides a detailed list of all untested output setup and output hold checks along with a possible reason for this state.

Example 74 Detailed Output for the report_analysis_coverage Command

```
*****
Report : analysis_coverage
         -status_details {untested}
         -check_type {out_hold out_setup}
Design : zDesign
: : :
*****
```

Type of Check	Total	Tested	Untested
out_hold	359	333 (93%)	26 (7%)
out_setup	359	333 (93%)	26 (7%)
All Checks	718	666 (93%)	52 (7%)

Constrained Pin	Related Pin	Check Type	Untested Reason
zOut1		out_hold	no_startpoint_clock
zOut2		out_setup	no_startpoint_clock
zOut3		out_hold	no_startpoint_clock
zOut4		out_setup	no_startpoint_clock
zOut5		out_hold	no_startpoint_clock
zOut6		out_setup	no_startpoint_clock
zOut7		out_hold	no_paths
zOut8		out_setup	no_paths
zOut9		out_hold	constant_disabled
zOut10		out_setup	constant_disabled
zOut11		out_hold	constant_disabled
...			

After constraint consistency points out a potential untested reason, you can use debugging commands to find out more specific information about the source of the untested check. For example, if you want to know why “zOut9” is untested, run the following command:

```
ptc_shell> report_case_details -to [get_ports zOut9]
```

[Example 75](#) shows the output report.

Example 75 Output for the report_case_details Command

```
*****
Report : case_details
Design : zDesign
: : :
```

```
*****
Properties      Value   Pin/Port
-----
constant net    0       zOut9

Case fanin report:
Verbose Source Trace for pin/port zOut9:
Path number: 1
Path Ref #  Value  Properties          Pin/Port
-----
0           F ()=A
0           F ()=A'
0           F ()=A'
1           F ()=A'
0           F ()=A'
1           constant net
1           zOut9
io_buffer_out_231/Y
io_buffer_out_231/A
zDesign_top_i0/Y
zDesign_top_i0/A
zDesign_top_i0/Y
zDesign_top_i0/A
zDesign_top_i0/top_tieoff_i0/U100/Y
zDesign_top_i0/top_tieoff_i0/U100/A
zDesign_top_i0/top_tieoff_i0/U42/HI
```

In [Example 75](#), you can see that the output is constant because of the TIE cell “zDesign_top_i0/top_tieoff_i0/U42”.

report_analysis_coverage in Multi-Scenario Runs

Constraint consistency analyzes several scenarios in one command. This is particularly important with analysis coverage reports. Not all the endpoints are tested in a given mode; however, you should try to cover as many endpoints or timing arcs as possible over all the design modes. For example, if the design has scan chains, the scan-related pins in the flip-flops are not covered in functional mode. This results in a collection of untested checks in the design. However, after constraint consistency analyzes both the functional and test modes, all flip-flop related checks should be covered and reported as tested. To summarize, because not all endpoints are covered in every mode, it is important to distinguish between those endpoints that are not tested in any mode and those endpoints that are tested in at least one of the modes. Constraint consistency takes this into account and generates a report slightly different from the one created in a single-scenario environment. [Example 76](#) shows this variation:

Example 76 report_analysis_coverage for Multi-Scenario Environments

Type of Check	Total	Tested in 1+ scenario	Untested in all scenarios
clock_gating_hold	9400	4165 (44%)	5235 (56%)
clock_gating_setup	9400	4165 (44%)	5235 (56%)
hold	1328927	575149 (43%)	753778 (57%)
min_period	352	351 (100%)	1 (0%)
min_pulse_width	870706	491034 (56%)	379672 (44%)
out_hold	36	36 (100%)	0 (0%)
out_setup	36	36 (100%)	0 (0%)
recovery	315842	295658 (94%)	20184 (6%)
removal	315838	295654 (94%)	20184 (6%)

setup	1328927	575149 (43%)	753778 (57%)
All Checks	4179464	2241397 (54%)	1938067 (46%)

The detailed reports in multi-scenario analysis show the modes where a particular check is tested. To obtain these details, run

```
ptc_shell> report_analysis_coverage -check_type out_hold \
          -style verbose -status_details tested
```

[Example 77](#) shows the detailed output:

Example 77 report_analysis_coverage Detailed Output

Type of Check	Total	Tested in 1+ scenario	Untested in all scenarios	
out_hold	36	36 (100%)	0 (0%)	
All Checks	36	36 (100%)	0 (0%)	
...				
Constrained Pin	Related Pin	Check Type	Scenarios	Untested Reasons
zOut1		out_hold	default FF_FUNC FF_SCAN	x x t
zOut2		out_hold	default FF_FUNC FF_SCAN	x t x
zOut3		out_hold	default FF_FUNC FF_SCAN	x t x
...				

In the “Untested Reasons” column, the “t” stands for tested. As a consequence, you can see that zOut1 is tested in scan mode, whereas zOut2 and zOut3 are tested in functional mode.

report_exceptions Command (Redundant Constraints)

Constraint consistency helps you develop complete and efficient constraints. When developing a design with many constraints, it is common to encounter either incorrect, incomplete, or unnecessary constraints. Incorrect or incomplete constraints have a direct effect on the quality and performance of the design. Unnecessary constraints are difficult to find and lead to increased memory usage and runtime. Therefore, it is important to have

an automated way to detect them. This section describes how to find redundant constraints and determine the dominant constraints in your design in the following subsections:

- [Exception Reporting Overview](#)
- [Using report_exceptions](#)
- [Narrowing Down report_exceptions Reports](#)
- [Reporting Ignored Exceptions](#)
- [Reporting Dominant Exceptions](#)
- [Reporting Dominant Exceptions for Ignored Exceptions](#)

Exception Reporting Overview

To refine your constraint sets and thereby improve your quality of results, use the `report_exceptions` command.

After you load the various constraints for each scenario into your design, use the `report_exceptions` command to analyze and report the status of your constraints. In addition to reporting various types of exceptions, constraint consistency can analyze your design for redundant and dominant exceptions. This capability helps you minimize your constraints.

For example, run the `report_exceptions -ignored` command. Constraint consistency analyzes your design and returns a list of exceptions that have no effect on the design and could be safely removed from the constraint sets. When you run the `report_exceptions -dominant` command, constraint consistency returns a list of exceptions that do affect timing.

With constraint consistency, you can identify incorrect, incomplete, and unnecessary constraints as follows:

- Most incorrect constraints are flagged with one of the many built-in rules checked by the `analyze_design` command.
- Incomplete constraints can be identified through either the built-in rules or the analysis coverage capability provided by the `report_analysis_coverage` command. See “[Creating a Tcl Rule-Checking Procedure](#)” for more information.
- Unnecessary constraints are reported by the `report_exceptions` command.

To find exceptions that do not affect the timing of the circuit, but might have a negative impact on tool performance, use the `report_exceptions -ignored` command. You can also find these occurrences by turning on the EXC_0006 and EXC_0014 rules when you run the `analyze_design` command. These rules are disabled by default because they need extra runtime. See “[Enabling and Disabling Rules](#)” for more information.

Using report_exceptions

The `report_exceptions` command works on the current scenario. If you have several scenarios loaded, you can switch from one scenario to another using the `current_scenario` command.

When used without any options, the `report_exceptions` command reports all timing exceptions as defined in your SDC (or Tcl) files. All of the exceptions and constraint specifications are reported in your constraint file. [Example 78](#) shows an example of a default report.

Example 78 Default Report for the report_exceptions Command

```
*****
Report : exceptions
Design : zDesign
Scenario: SCAN_SCENARIO
. . .
*****
# /remote/designs/zDesign/zDesign_scan.sdc, line 88
set_false_path -through [get_pins {jtag_wrap_inst/LVISION_JTAP_INST/
LVISION_JTAP_DR_CTRL/INT_DR_LATCH_reg_44/q}]
# /remote/designs/zDesign/zDesign_scan.sdc, line 89
set_false_path -through [get_pins {jtag_wrap_inst/LVISION_JTAP_INST/
LVISION_JTAP_DR_CTRL/INT_DR_LATCH_reg_69/q}]
# /remote/designs/zDesign/zDesign_scan.sdc, line 90
set_false_path -through [get_pins {jtag_wrap_inst/LVISION_JTAP_INST/
LVISION_JTAP_DR_CTRL/INT_DR_LATCH_reg_42/q}]
. . .
```

[Example 78](#) provides a list of all of the exceptions but, for this very reason, it can be difficult to find specific data within the report. The next section describes how to narrow down your reports to locate specific problems.

Narrowing Down report_exceptions Reports

The `report_exceptions` command offers all the standard filtering options, such as `-from/through/to` and `-rise_through/fall_through`. These can be used to reduce the number of exceptions reported. When a filter is used, only exceptions that have the same `-from/through/to` clause are reported.

To help understand how the `-from/through/to` filters are used, consider the constraints defined in [Example 79](#).

Example 79 Exceptions Defined in Constraints File

```
set_multicycle_path 3 -from [get_clocks CLK1]
set_multicycle_path 3 -from ff1/CP
```

If ff1/CP is actually clocked by CLK1, when you run

```
ptc_shell> report_exceptions -from [get_clocks CLK1]
```

only the following timing exception is reported:

```
set_multicycle_path 3 -from [get_clocks CLK1]
```

The following multicycle path (MCP) exception is not reported:

```
set_multicycle_path 3 -from ff1/CP
```

When you use the `-from/through/to` options with the `-ignored` or `-dominate` options, the set of reported exceptions is reduced based on whether the filtered set of exceptions are ignored for all timing paths or used for at least one timing path.

Reporting Ignored Exceptions

Ignored exceptions are timing exceptions that do not have any effect in the design. There can be many different reasons why an exception is ignored. When constraint consistency marks an exception as ignored, it also identifies the reason the constraint was ignored as one of the following conditions:

- The exception is fully covered by another exception in the SDC file.
- There is no path in the design related to this exception.
- No valid startpoint is specified in the exception.
- No valid endpoint is specified in the exception.

[Example 80](#) shows an example report. You can also use the `-ignored` option in conjunction with other options, such as `-from`, to narrow the set of exceptions in the report.

Example 80 Output for the report_exceptions -ignored Command

```
*****
Report : exceptions
Design : zDesign
Scenario: FUNCTIONAL
. . .
*****
##### Redundant exceptions (totally overridden by other exceptions):
# /remote/designs/zDesign/sdc/functional.sdc, line 440 \
set_false_path -from [get_clocks {Top_clk_1}] -through [get_pins
{zdesign_core_i0/
zPin1}] -to [get_clocks {Top_clk_2}]
#####
```

```
###  
## Exceptions that do not cover any constrained paths:  
#   /remote/designs/zDesign/sdc/functional.sdc, line 474  
set_false_path -through [get_pins {zDesign_core_i0/i_resetb_pad_din}]  
#####  
## Exceptions with no valid -from objects:  
#   /remote/designs/zDesign/sdc/functional.sdc, line 503  
set_false_path -hold -from [get_ports {JTAG_CE}] -to [get_pins \  
{zDesign_top/  
jtag_control_inst/reg_127_/d}]
```

To reduce runtime and memory usage in future sessions, remove ignored exceptions from the SDC file.

Note:

Constraint consistency does not modify constraints. Edit your constraints and rerun constraint consistency for verification.

Reporting Dominant Exceptions

Constraint consistency reports the dominant exceptions in the design. To report the dominant exceptions, run the following command:

```
ptc_shell> report_exceptions -dominant
```

An exception is considered dominant if it affects the constraints on at least one path and its absence would result in a different set of constraints in the design. [Example 81](#) shows output for the `report_exceptions -dominant` command.

Example 81 Output for the `report_exceptions -dominant` Command

```
*****  
Report : exceptions  
Design : zDesign  
Scenario: FUNCTIONAL  
.  
*****  
#####  
## Exceptions that are dominant for at least one path:  
#   /remote/designs/zDesign/sdc/functional.sdc, line 433  
set_false_path -from [get_clocks {clk_1}] -to [get_clocks {clk_54}]  
#   /remote/designs/zDesign/sdc/functional.sdc, line 434  
set_false_path -from [get_clocks {clk_54r}] -to [get_clocks {clk_1}]
```

You can narrow down the set of reported exceptions by combining the `-dominant` option with other options.

Reporting Dominant Exceptions for Ignored Exceptions

Constraint consistency reports the dominant exceptions for ignored exceptions in the design. To report these exceptions, run the following command:

```
ptc_shell> report_exceptions -ignored -verbose
```

[Example 82](#) shows output for the `report_exceptions -ignored -verbose` command.

Example 82 Output for the report_exceptions -ignored -verbose Command

```
*****
Report : exceptions
Design : test
Scenario: default
...
#####
## Redundant exceptions (totally overridden by other exceptions):
#
# The following exception in test.tcl at line 62 is dominated by other
exceptions:
set_multicycle_path 2 -setup -from [get_clocks {CLK1_1}]
# The following exceptions dominate the above exception :
    # test.tcl, line 62
    set_multicycle_path 3 -setup -from [get_pins {H1/H1/F1/CLK}]
    # test.tcl, line 64
    set_multicycle_path 2 -setup -from [get_pins {H2/H1/F1/CLK}]

# The following exception in test.tcl at line 65 is dominated by other
exceptions:
set_multicycle_path 3 -setup -from [get_clocks {CLK2_1}]
# The following exceptions dominate the above exception:
    # test.tcl, line 62
    set_multicycle_path 3 -setup -from [get_pins {H1/H1/F1/CLK}]
    # test.tcl, line 64
    set_multicycle_path 2 -setup -from [get_pins {H2/H1/F1/CLK}]
```

Graphical User Interface

The constraint consistency graphical user interface (GUI) provides an environment to view and debug constraint problems for PrimeTime and other Synopsys tools, such as Design Compiler and IC Compiler. The GUI environment is the main interface for working with constraint consistency.

The constraint consistency GUI is described in the following sections:

- [Using the GUI](#)
- [The Constraint Consistency Window](#)

- [User Message Browser](#)
- [Violation Browser](#)
- [Block-to-Top Violation Browser](#)
- [Correlation Consistency Violation Browser](#)
- [Waiver Configuration Dialog Box](#)
- [Information Pane](#)
- [Console](#)
- [Online Help](#)
- [SDC View](#)
- [Hierarchy Browser and Schematic View](#)
- [Select By Name Toolbar](#)
- [Properties Dialog Box](#)
- [Schematic Display Options](#)

Using the GUI

The GUI can help you visualize and understand the nature of the violations you might encounter. You can use the visual analysis tools after you read, link, constrain, and analyze the design.

An analysis session might consist of the following tasks:

1. Start the GUI.
2. Read, link, constrain, and analyze the design.
3. Perform in-depth analysis, examining specific aspects of the violation paths in detail.

The default behavior for starting a constraint consistency session that automatically includes the GUI is to use `ptc_shell`. Within the shell you can use the `gui_start` and `gui_stop` commands to start and stop the GUI.

If you want to run the constraint consistency in shell-only mode, enter the `ptc_shell -no_gui` commands.

You can display or hide elements of the GUI. For example, to display or hide a toolbar or panel, choose **View > Palette/Toolbars** and choose the appropriate command to display or to hide the corresponding area of the window. A check mark is shown next to the menu item if that element is currently being displayed in the window.

The lower area of the window contains the console. The `ptc_shell` prompt is on the console. Use the Log and History tabs above the `ptc_shell` prompt to display different types of information about the console.

To stop the GUI while still keeping the original `ptc_shell` session going in the terminal window, use the `gui_stop` command or choose File > Close GUI. To exit, choose File > Exit.

The Constraint Consistency Window

Each window can be minimized or maximized. A window can be moved anywhere within the top-level window. Windows can overlap each other.

A window has a title bar with buttons to minimize, maximize, or close the window, and you can freely move and resize the window.

To lay windows out evenly like tiles, choose the menu command Window > Tile, as shown in [Figure 257](#). To cascade the windows, choose Window > Cascade. You can arrange and resize the windows as shown in [Figure 258](#).

[Figure 257 Tiled Windows](#)

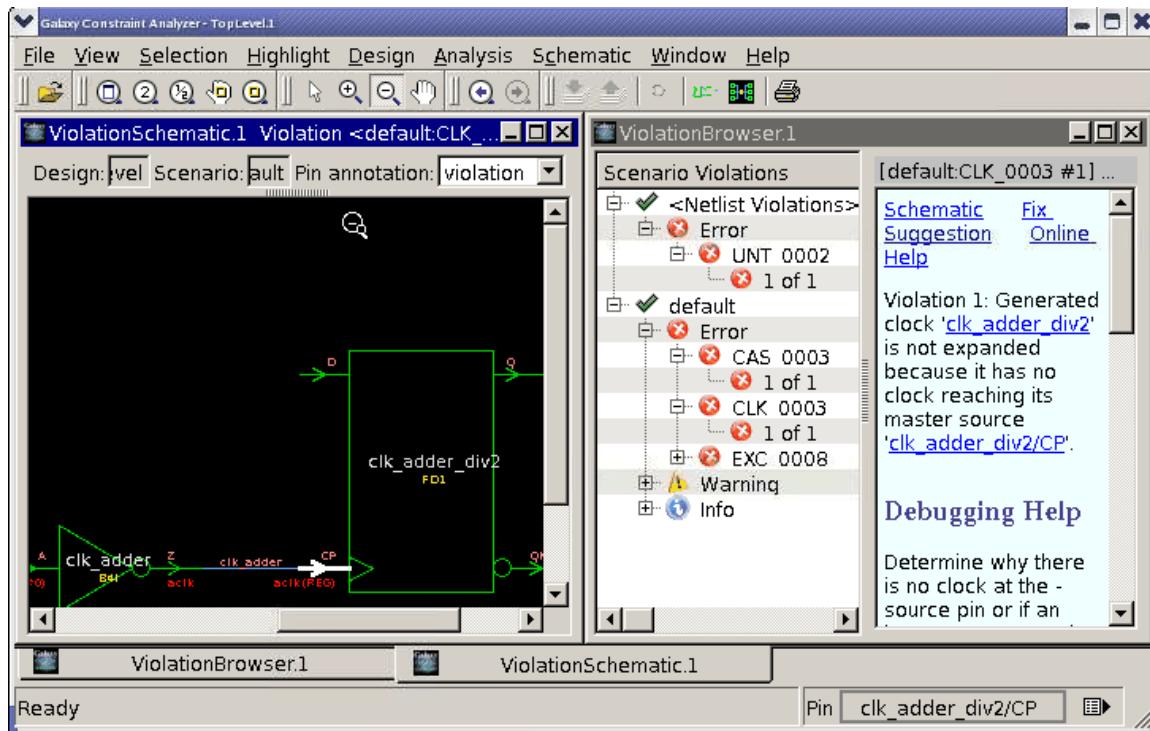
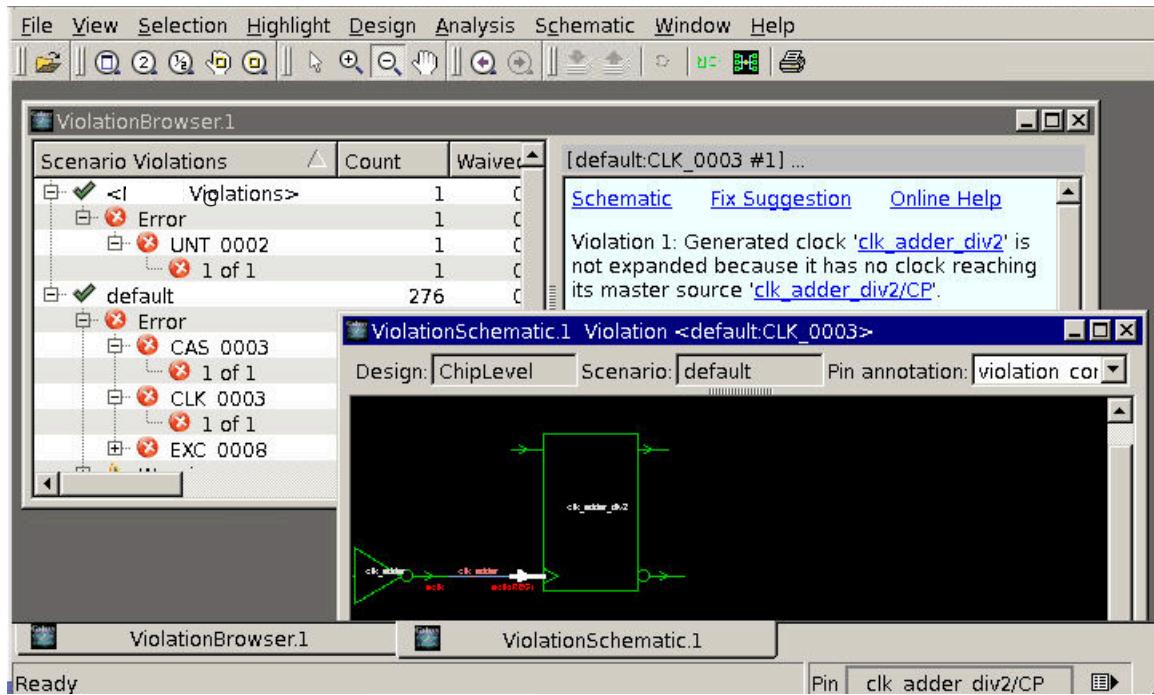


Figure 258 Arranged and Resized Windows



You can move, resize, and minimize windows at any time. To move a window, point to the title bar and drag it to the position you want.

Window Types

[Table 46](#) lists and briefly describes the types of windows that can be displayed within the Constraint Consistency window. You can find additional information about each type of window in the remaining sections of this chapter.

Table 46 Types of Windows Within the Constraint Consistency Window

Window type	Description
Violation Browser	Lists the violations in the current design, classified by scenario and severity.
B2T Mismatch Browser	Lists the violations between the constraints defined for the top level of the current design and the block level, classified by severity.
S2S Mismatch Browser	Lists the differences between two sets of constraints for the current design, classified by severity.
User Message Browser	Lists the user messages in the current design, classified by severity.
Information Pane	Lists detailed information related to the current selection.

Table 46 Types of Windows Within the Constraint Consistency Window (Continued)

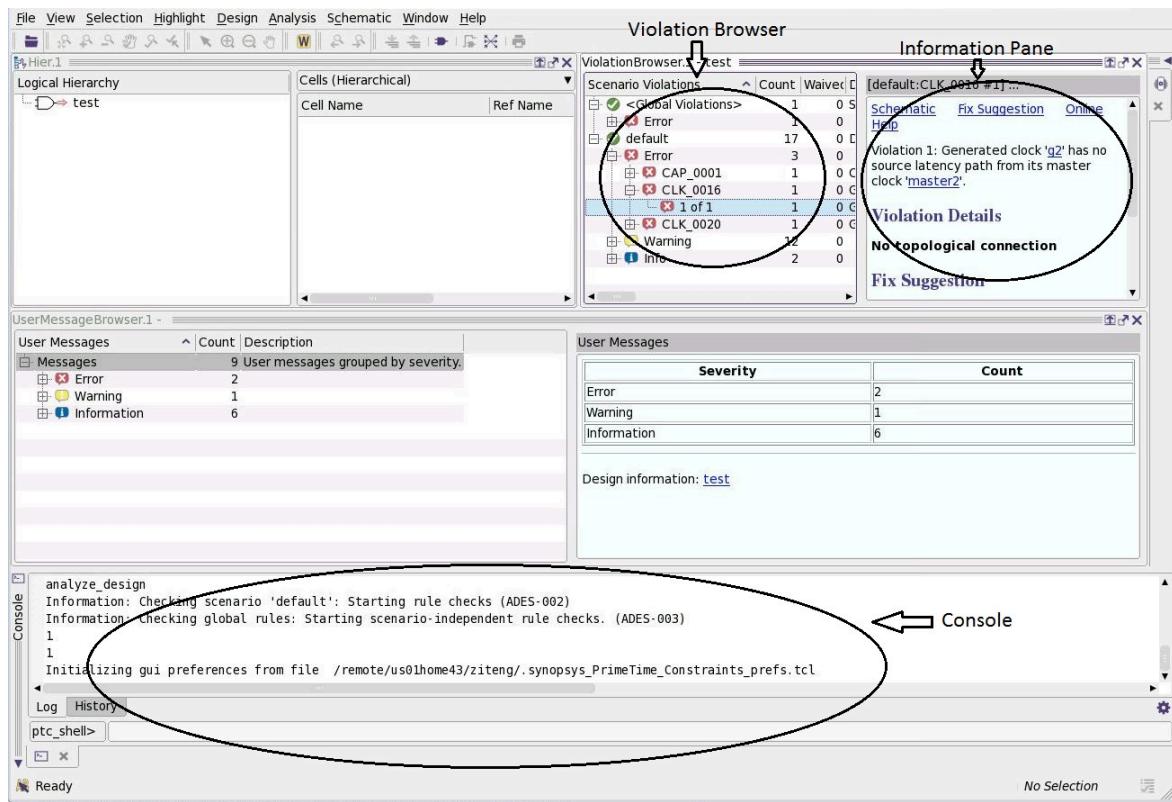
Window type	Description
Console Window	Lets you run <code>ptc_shell</code> command and view the <code>ptc_shell</code> log, command history, or error and warning messages.
SDC View	Displays the SDC constraint file and places a marker on the relevant line.
Hierarchy Browser and Schematic View	Lets you browse through the design hierarchy and select a cell, net, port, or pin in the design.
Path Schematic	Displays a circuit schematic of a hierarchical cell or a path in the design.
Properties Dialog Box	Displays a list of the object's properties.

The Constraint Consistency window starts out with the following windows:

- Violation browser
- Information pane

By default, the violation browser, information pane, and console windows are docked and displayed on the left side, as shown in [Figure 259](#). You can undock or remove these default windows and open additional windows such as Schematics and Query. There is an “on-demand” hierarchy browser window that you can add to the standard appearance of the GUI by modifying your default preferences. If the hierarchy browser window is not present, you can open one by choosing Window > New Hierarchy Browser.

Figure 259 Constraint Consistency Window on Startup



Toolbars

The toolbar buttons provide quick access to often-used menu commands. Figure 260 shows an example of the toolbar buttons.

Figure 260 Toolbar Buttons



Buttons that cannot be used in the active view are dimmed. The current context changes when you make a new selection. For example, after you select a path, the toolbar options that operate on paths become enabled. When no paths are selected, those buttons functions are dimmed.

Menu Commands

Table 47 briefly describes each of the menu bar menu commands.

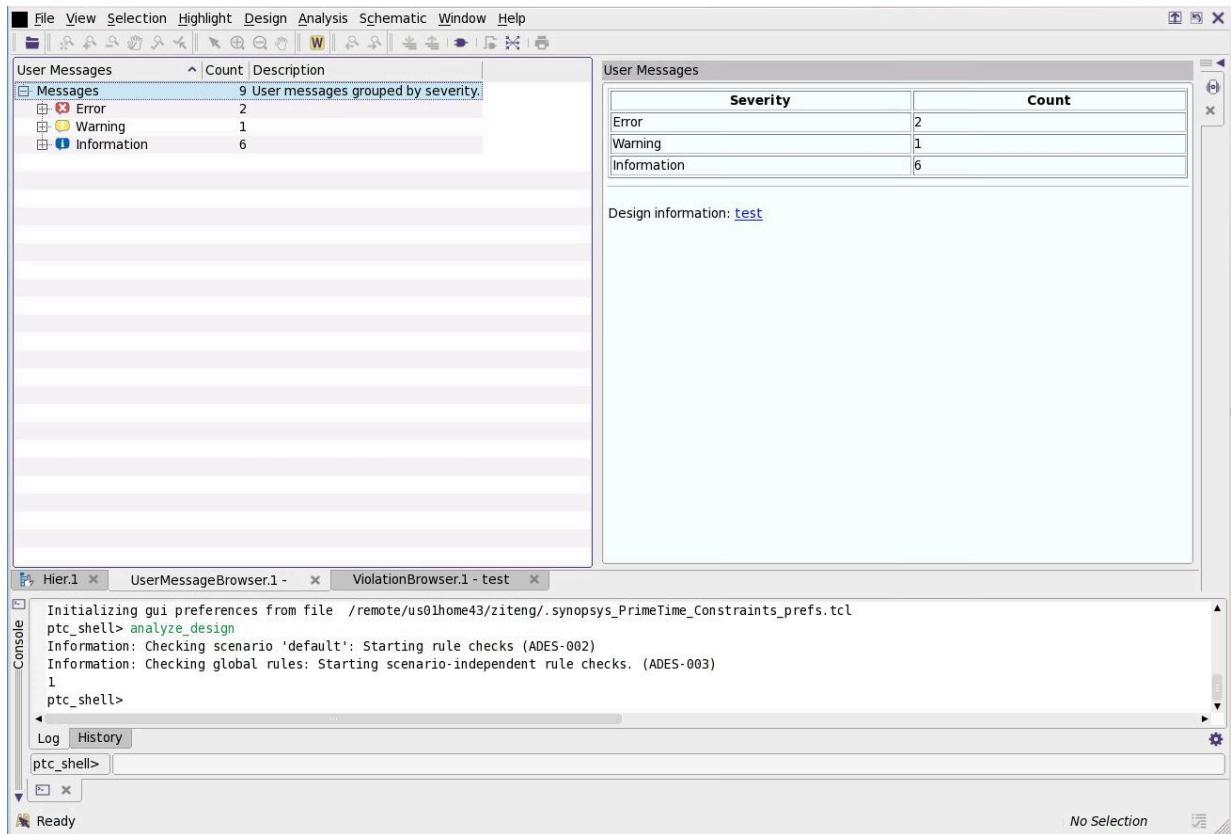
Table 47 Menu Commands

Menu commands	Description
File	Contains commands for executing scripts, printing schematics, and exiting constraint consistency.
View	Contains commands for controlling the design views, mouse function, InfoTip display, toolbars, and other view settings.
Selection	Contains commands for selecting schematic objects and reporting the selected objects.
Highlight	Contains commands for coloring selected objects in schematic views.
Design	Contains commands for analyzing the design and opening the hierarchy browser and the waiver configuration dialog box.
Analysis	Contains commands for opening the violation browser, B2T mismatch browser, S2S mismatch browser, and user message browser.
Schematic	Contains commands for opening and controlling schematic views.
Window	Contains commands for controlling the display of windows within the Constraint Consistency GUI.
Help	Contains commands for getting online Help.

User Message Browser

User messages are displayed in the user message browser shown in [Figure 261](#). To open the user message browser, choose Analysis > UserMessage Browser. In the browser tree, the messages are sorted according to severity. When you select a message in the browser tree, the message details are displayed in the information pane. For the message selected in [Figure 261](#), the information pane identifies the line number of the problem SDC constraint and a link to that line in the SDC file.

Figure 261 User Message Browser



Violation Browser

This section provides an overview and guidelines for navigating the violation browser, in the following sections:

- [Violation Browser Overview](#)
- [Guidelines for Navigating in the Violation Browser](#)

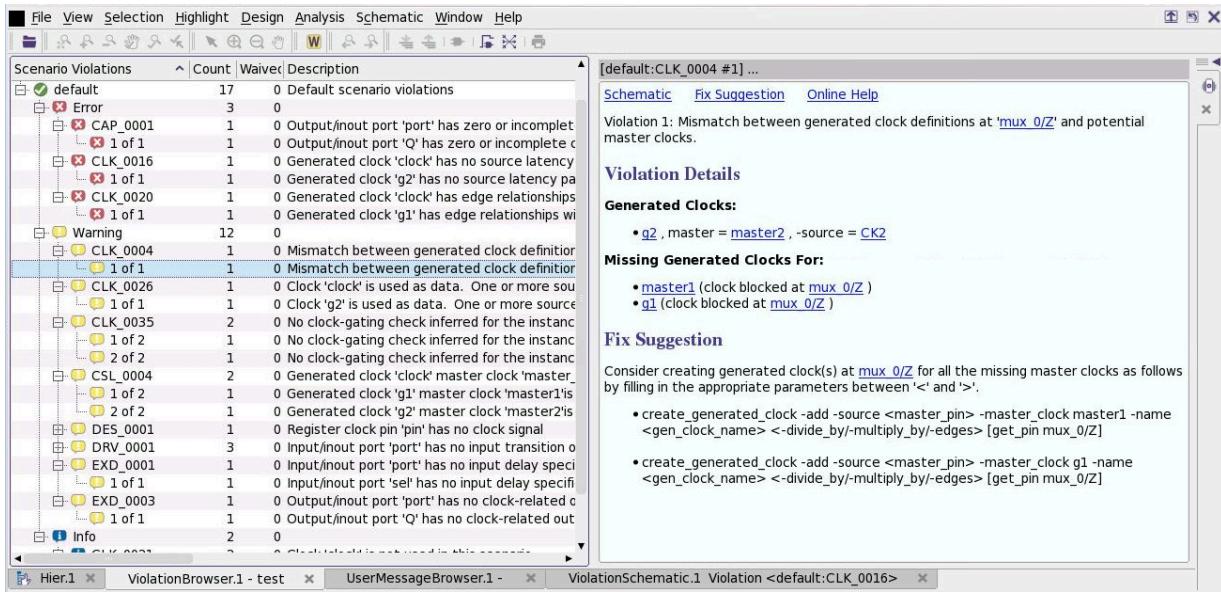
Violation Browser Overview

The violation browser is the main tool for performing debugging tasks. It displays a hierarchical list of the design and constraint violations that have been detected along with suggestions on how to fix the problem. The violation browser supports netlist-related and scenario-related violations (one node per scenario). Violations are displayed in the following order: Error, Warning, and Information.

By default, the violation browser displays up to 1000 violation identifier nodes for each rule violated. Use the `display_violations_per_rule_limit` variable to specify a different limit.

Figure 262 shows a typical violation browser.

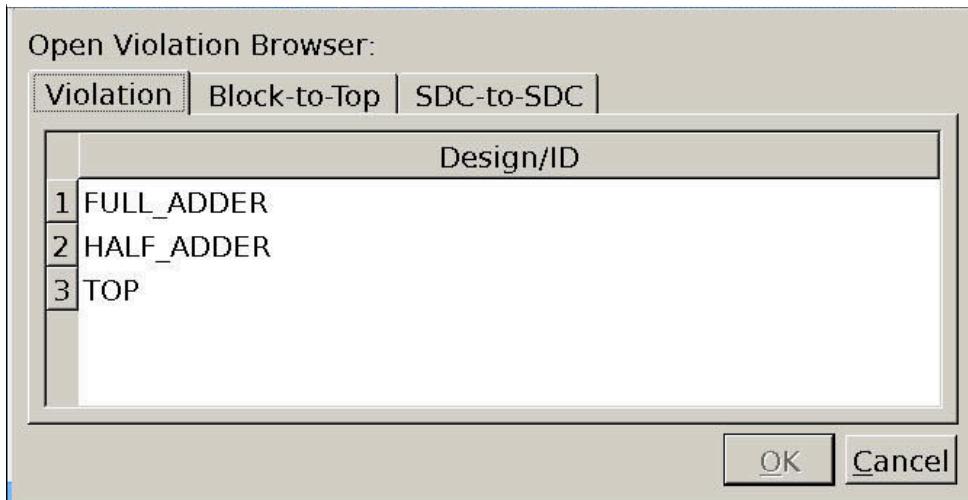
Figure 262 Violation Browser



During a session, constraint consistency stores the results of multiple analysis runs in multiple violation browsers. You use the analysis run chooser window, shown in Figure 263, to choose the specific run results to view.

Access the analysis run chooser window from the Analysis menu on the menu bar.

Figure 263 Analysis Run Chooser



This window displays the available results grouped by violation type. In addition to the standard violation browser, there is the B2T violation mismatch browser, and the S2S violation mismatch browser. For more information about these browsers, see “[Hierarchical Consistency Checking](#)” and “[Correlation Consistency Checking](#)”.

When the GUI is first opened, all of the severity nodes in the violation browser are expanded and all of the violation identifier nodes are collapsed. When expanded, a severity node displays all of the violations present in the design at that severity level.

You can specify the Web browser used to display the messages by setting an environment variable at the operating system prompt.

The browser executable needs to be in your environment variable setting.

```
setenv USER_HELP_BROWSER preferred_browser
```

Guidelines for Navigating in the Violation Browser

In the violation browser, clicking the +/- next to a violation identifier node expands the node and displays every individual violation of that type.

Whenever an individual violation is selected, the information pane shows the specifics of that particular violation. Whenever a violation instance is selected, the information pane shows the relevant data for the violation instance and a general message for the particular set of violations (for example, CLK_0004 message).

The violation browser groups violations of a rule if the violations contain similar object names. The browser displays these multiple violations as a single group, as shown in [Figure 262](#). This feature helps you see the scope of all your design violations. When you select a group name, the information pane displays the Description field shown in the violation browser. Constraint consistency allows you to suppress rule violations.

You can either disable a rule completely or suppress a specific instance of a rule violation, also known as waiving a violation. To disable grouping or enable grouping for a specific number of levels, use the `grouping_violations_hierarchy_separator_limit` variable.

Both waived and unwaived violations are displayed in a group. Rules without netlist objects in their parameters are not grouped because violation grouping is performed on netlist object names, such as pins, ports, and cells.

Block-to-Top Violation Browser

To compare a set of constraints from a block to the constraints from the top-level chip, use the `compare_block_to_top` command. You use the block-to-top violation browser to view the violations.

See “[Hierarchical Consistency Checking](#)” for more information.

Correlation Consistency Violation Browser

To determine the functional differences between the two sets of constraints using one common netlist, use the `compare_constraints` command. You use the correlation consistency violation browser to view the violations.

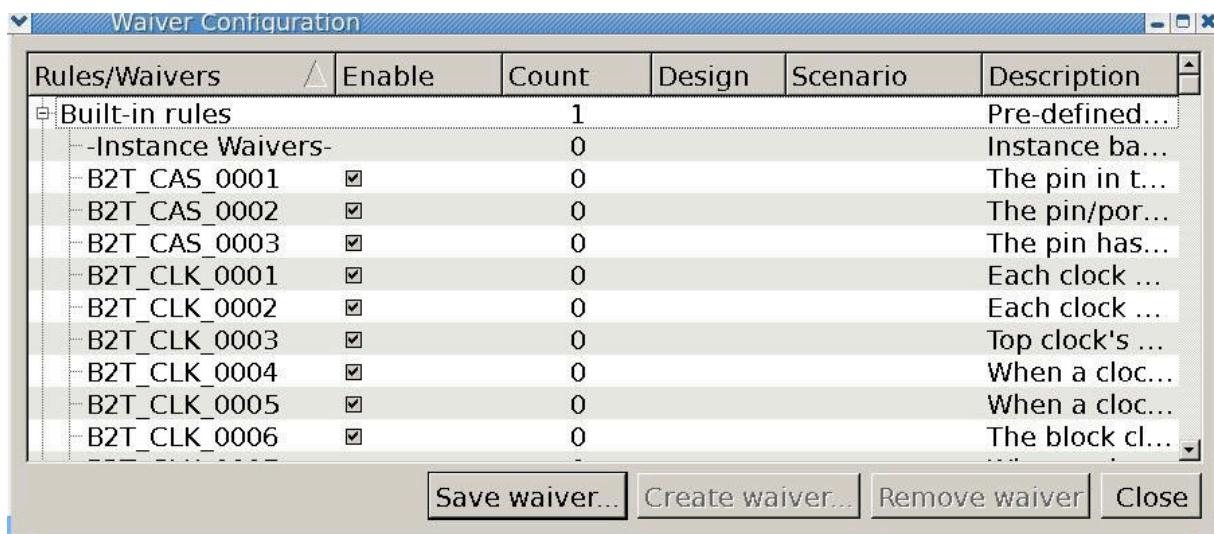
See “[Correlation Consistency Checking](#)” for more information.

Waiver Configuration Dialog Box

The waiver configuration dialog box allows you to waive either an entire rule or specific violations of a rule. Unlike the violation browser, you can use the waiver configuration dialog box both before and after running the `analyze_design` command. In addition, the waiver configuration dialog box gives you complete access to all the rules available, whereas the violation browser shows only those rules with violations.

To open the waiver configuration dialog box, choose Design > Waiver Configuration.

Figure 264 Waiver Configuration Dialog Box



The columns in the waiver configuration dialog box are described in [Table 48](#).

Table 48 Waiver Configuration Dialog Box Columns

Column	Description
Rules/Waivers	Name of the rule. You can access all built-in and user-defined rules with the waiver configuration dialog. A plus sign next to the rule indicates that specific violations of the rule have been waived.
Enable	A check mark indicates that the rule is enabled. To disable the rule, uncheck the box. This method of enabling and disabling a rule applies to all scenarios.
Count	Shows the number of waivers for the rule.
Design	Specifies the design to which a waiver is applied. If no design is specified, the waiver applies to the current design.
Scenario	Lists the scenarios in which the waiver is active. If no scenario is listed, the waiver applies to all scenarios.
Description	Brief explanation of the built-in rule or user-defined comment for a user-defined rule. For detailed information about the built-in rules, see the rule reference documents in the online Help.

Information Pane

The information pane, shown in [Window Types on page 778](#), is a central feature of the GUI environment. Whereas the violation browser allows you to find general information about rule violations, the information pane displays the details of these violations.

Whenever a violation is selected in the violation browser, the information pane automatically displays the relevant information. In this way the information pane is always synchronized with whatever is selected in the violation browser window.

The information pane provides detailed information related to the current violation, including the full message of the violation and hyperlinks to related objects (whenever applicable).

The information pane also provides suggestions on how to debug the violation in more detail and provides the necessary setup details for creating some path schematics.

Console

Constraint consistency provides a console, shown in [Window Types on page 778](#), in which you can enter interactive commands; anything that can be done in a batch constraint consistency session can also be done through the console.

Standard output and error output are displayed in the console log view.

To save the command history list into a file, click the History tab and click the Save Contents As button.

Online Help

The Constraint Consistency online Help system contains man pages and rules related to constraint consistency.

You can access the online Help system from the Help menu in the GUI environment and from the command line when entering a `man topic` command.

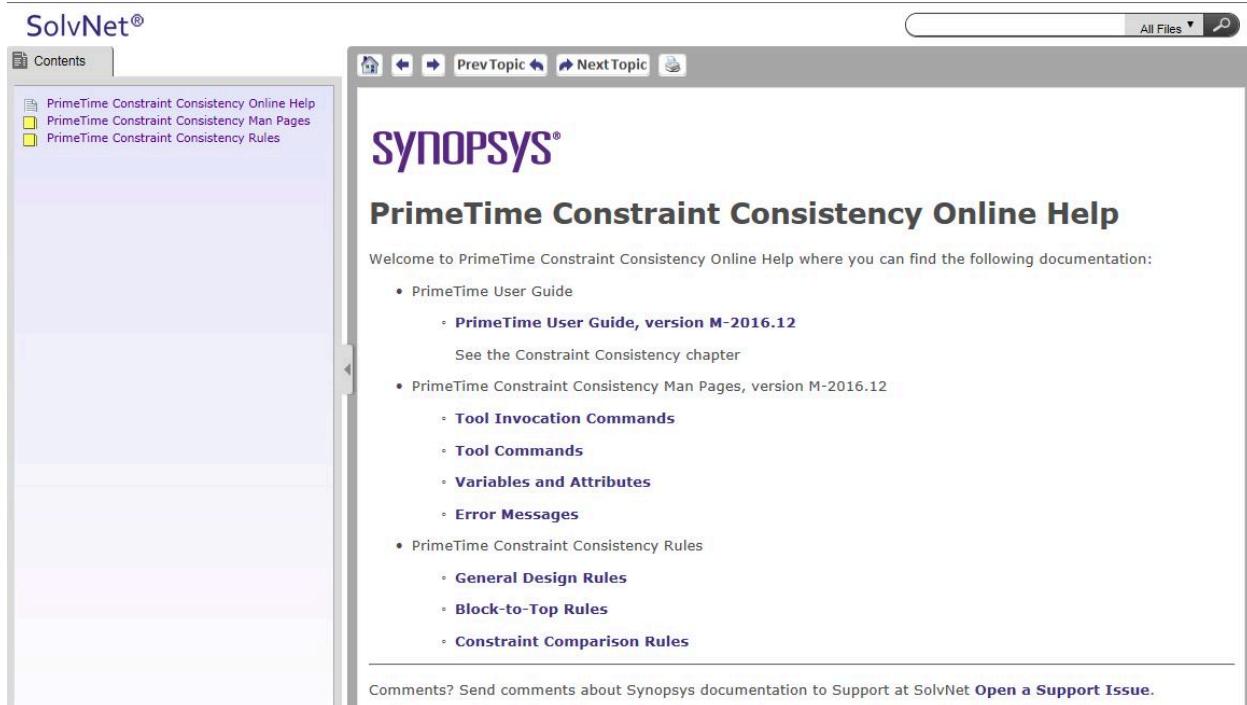
From the Help menu, choose online Help to open the online Help system in a Web browser.

The online Help system is also available through the information pane of any rule by clicking the Help hyperlink.

In the online Help system, which opens in a new Web browser window, the Rule Reference page for any rule contains an explanation of the rule, an example of a design containing a violation of that rule, and a suggestion on how to debug the issue.

[Figure 265](#) shows the startup page of the Constraint Consistency online Help system.

Figure 265 Constraint Consistency Online Help System



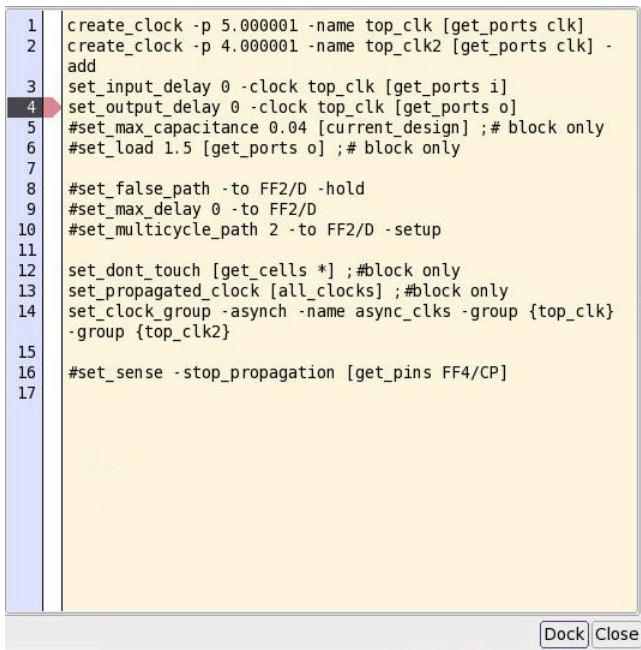
SDC View

Through the use of the SDC viewer, constraint consistency can identify constraints by their definition in a file (file name and line number). This type of information, when provided, is available through a hyperlink in the violation message in the violation browser and in the information pane.

By clicking the hyperlink of a constraint definition, the SDC viewer opens and automatically loads the relevant SDC constraint file and places a marker on the relevant line.

[Figure 266](#) shows a typical SDC view.

Figure 266 SDC View



```

1 create_clock -p 5.000001 -name top_clk [get_ports clk]
2 create_clock -p 4.000001 -name top_clk2 [get_ports clk] -
3 add
4 set_input_delay 0 -clock top_clk [get_ports i]
5 set_output_delay 0 -clock top_clk [get_ports o]
6 #set_max_capacitance 0.04 [current_design] ;# block only
7 #set_load 1.5 [get_ports o] ;# block only
8
9 #set_false_path -to FF2/D -hold
10 #set_max_delay 0 -to FF2/D
11 #set_multicycle_path 2 -to FF2/D -setup
12
13 set_dont_touch [get_cells *] ;#block only
14 set_propagated_clock [all_clocks] ;#block only
15 set_clock_group -asynch -name async_clks -group {top_clk}
16 -group {top_clk2}
17 #set_sense -stop_propagation [get_pins FF4/CP]

```

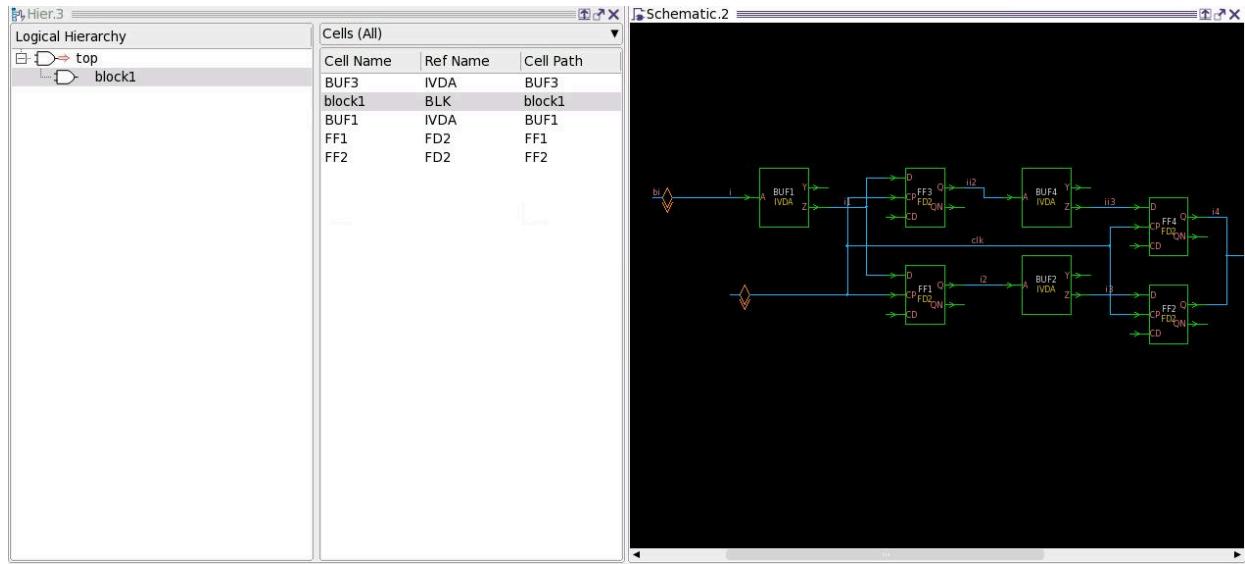
Dock Close

Hierarchy Browser and Schematic View

The hierarchy browser and schematic view let you examine the elements in the design hierarchy and display them in detailed schematic form.

[Figure 267](#) shows a typical design in the hierarchy browser and schematic view.

Figure 267 Hierarchy Browser/Schematic View



The look and feel of the Constraint Checking GUI are designed to be similar to those of other tools, such as the PrimeTime GUI or the Design Vision tool. The primary function of the hierarchy browser is to display the logic hierarchy of the design. Schematics can be created for the whole selection by right-clicking any level of hierarchy or by clicking the toolbar buttons. Several schematics can be opened simultaneously, each in its own window. Each schematic view window opens so that you can see the logic.

The hierarchy browser also provides the following functionality:

- Whenever a level of hierarchy is selected, the cells belonging to that level appear on the cell list.
- The schematic view also allows you to search for a cell, net, or pin and to zoom in or zoom out to view a selection.

The schematic view can be annotated with the:

- Cell name
- Net name
- Library cell name

Selecting objects directly in the schematic also populates the relevant selection box on the status bar at the bottom of the window.

The buttons available on the Schematics toolbar have the following functionality:

- The Create Instance Schematic button draws the level of logic currently selected. If nothing is selected, the current design is drawn.
- The Create Abstract Path Schematic of Selected Objects button draws only the selected objects and their immediate connections. If nothing is selected, the button is disabled.
- The Add Fanin/Fanout to Path Schematic button becomes active whenever a schematic window is opened. Otherwise the button is disabled.
- The zoom, pan, and select buttons on the View Tools and View Zoom/Pan toolbars enable you to navigate through the schematic.

The Schematic View also provides standard printing capabilities similar to other Synopsys GUI tools.

Path Schematic

The path schematic view in the GUI provides visualization of any part of the design rather than the full schematic view. Any path schematic you display is annotated with data, such as constant data and clock identifiers that are relevant to analyzing the issue.

The path schematic is displayed when you request it from the information pane (debugging commands).

There are a number of visibility features built into the path schematic view:

- You can expand or reduce the schematic.
- You can add or remove gates from the view to aid in the granularity of the view.
- You can select one or several objects to obtain more detailed information about them.
- You can display attributes in a separate attribute pane.

There is also a Create Path Schematic button on the toolbar you can use to create a schematic of whatever logic is currently selected.

When creating a path schematic, a warning message displays beforehand if schematic generation is expected to take a long runtime. At this point, you have the option to cancel the current command. If significant runtime is anticipated when creating a path schematic, a Work In Progress indicator displays to show that constraint consistency is not failing.

Creating a Schematic

When a schematic view is applicable to a violation, the Schematic hyperlink appears in the information pane. You can automatically create a schematic for this violation by selecting

the Schematic hyperlink. The schematic is annotated with information relevant to the violation.

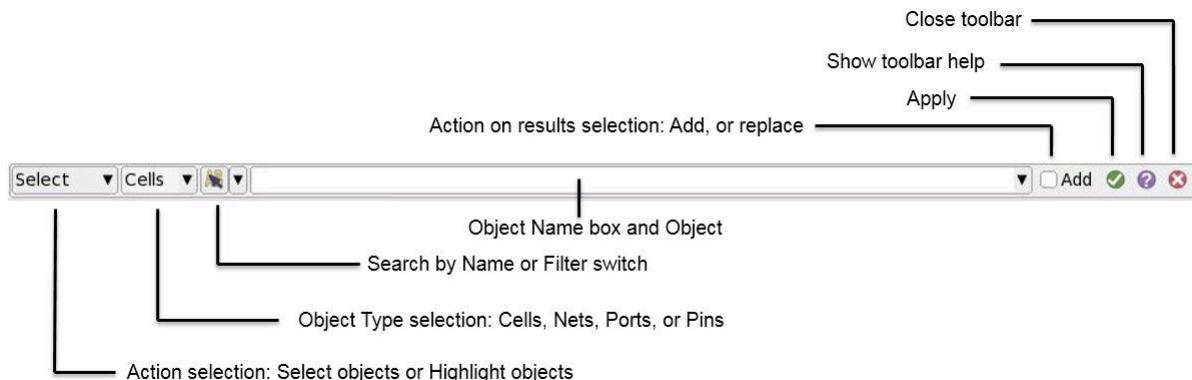
You can also create schematics at any level of logic by using the hierarchy browser or clicking the “Create Instance Schematic” button on the Schematics toolbar. Constraint consistency creates a schematic for all objects in the current selection list.

To open a hierarchy browser window, choose Design > Hierarchy Browser. Select the logic for the schematic and either right-click the selected cells and choose New Design Schematic View or choose Schematic > New Schematic View.

Select By Name Toolbar

Use the Select By Name toolbar, as shown in [Figure 268](#), to quickly select or highlight objects by name in the current design.

Figure 268 Select By Name Toolbar



The typical flow for the Select By Name toolbar involves the following steps:

1. Display the toolbar and set the keyboard focus: choose Selection > By Name Toolbar or press **Ctrl+.**
2. Select the object type: press **Page Up** or **Page Down** to cycle through the object type list.
3. Set the action to be performed: press **F1** to select objects or **F2** to highlight objects.
4. Specify the search string: enter the search string, and press **Tab** to complete the name if necessary.

5. Choose whether to replace or add to the current selection: press Insert to toggle the Add option.
6. Select or highlight the object: press Enter.

When you press Enter, constraint consistency searches the design for an object that matches the specified name or name pattern and automatically selects that object. The selected object is highlighted if it is visible in a schematic.

You can type part of a name and complete it by pressing Tab. The name completes to its longest match. If multiple names match the text, a name completion list appears. You can select a name by pressing Enter or close the list by pressing Esc. Use the Up Arrow and Down Arrow keys to move up and down the list.

You can type multiple object names by separating them with blank spaces.

If an object name contains a space, enclose the name in curly braces ({}) to treat it as a single name. If an object name is invalid because nothing matches the specified name pattern, the name appears on a light red background.

The Select By Name toolbar in [Table 49](#) provides the following interactive shortcut operations:

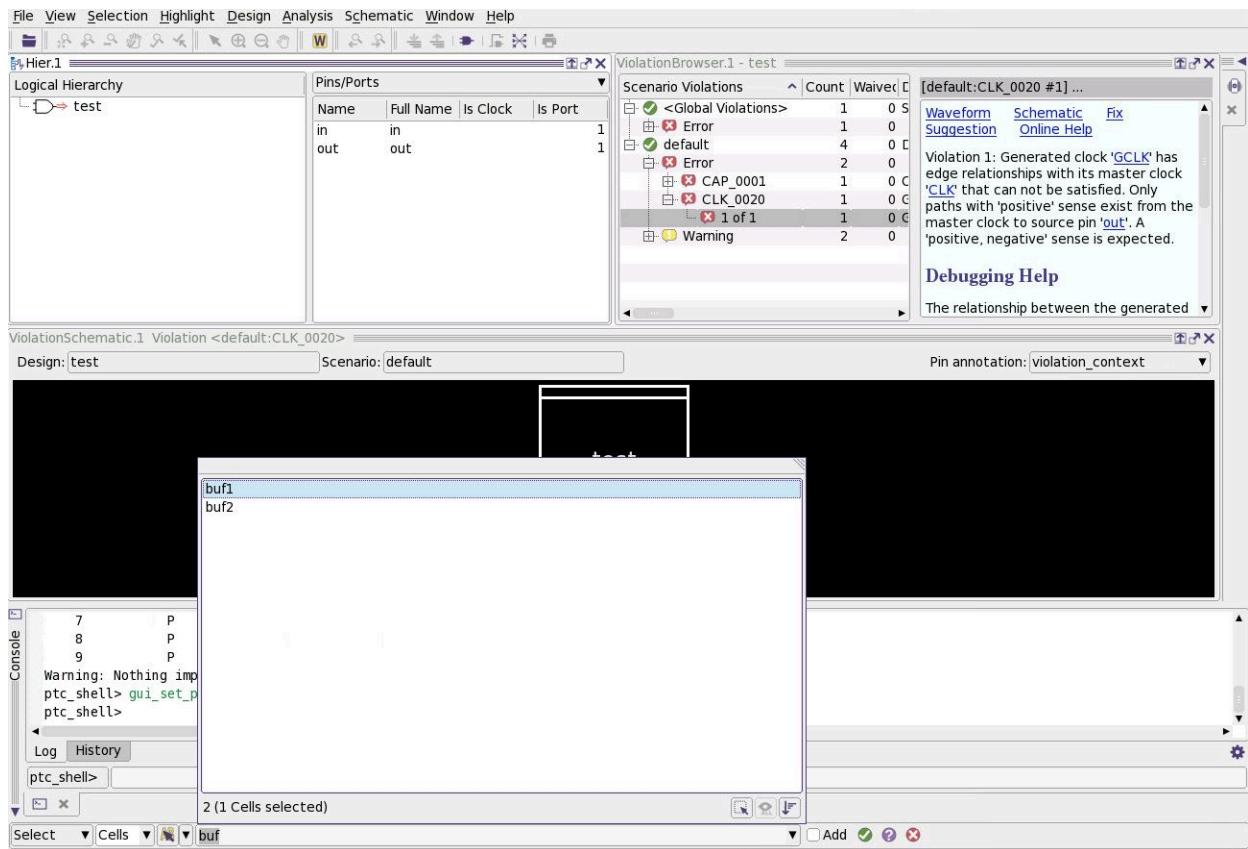
Table 49 Select By Name Toolbar Shortcuts

Key	Action
Enter	Select or highlight the objects
Tab	Complete current name to longest match
Down	Display completion menu
F1	Change to selection mode
F2	Change to highlight mode
Page Up	Move to the previous object type
Page Down	Move to the next object type
Insert	Switch between Add and Replace for selection mode
Escape	Hide the toolbar

Any characters you type while the object list is open automatically appear in the object name box. This allows you to continue typing the object name, which automatically updates the matching names in the name completion list.

Figure 269 shows an example of the Select By Name toolbar with the name completion list open.

Figure 269 Name Completion List



After a successful search, the text is automatically highlighted in the object name box and you can start another search. You can also perform the search by clicking the Apply button.

The selection operation replaces the current selection by default. If you want to add objects to the current selection, select the Add option.

If you press the Tab key and multiple objects are found that match the text you typed, the name completion list appears showing the first fifteen names.

If you enable name sorting, the option remains selected for future searches.

Properties Dialog Box

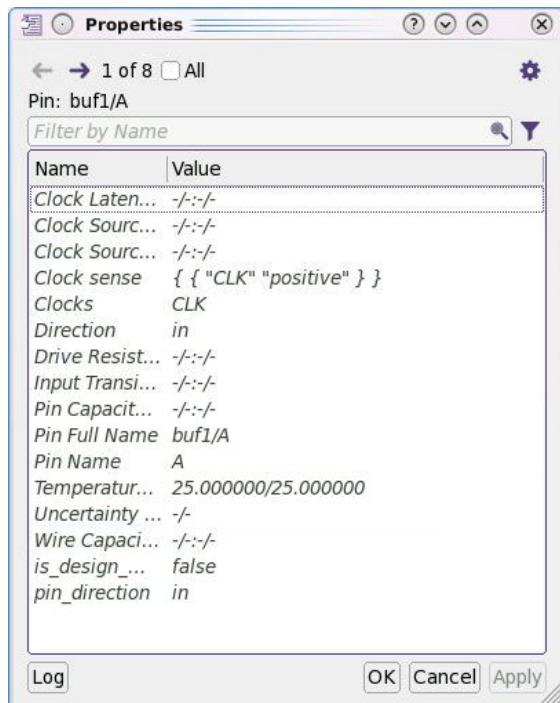
You can query the properties of any object in the design directly in the GUI by using the Properties dialog box. You can display an object's Properties dialog box by selecting the

object in the properties dialog box. Objects that have properties are nets, pins/ports, and cells.

The Properties dialog box can also be accessed by right-clicking an object in a schematic and choosing Properties.

[Figure 270](#) shows a typical attribute list in the Properties dialog box.

Figure 270 Properties Dialog Box



You can click the arrow buttons to scroll through the property lists for multiple selected objects. To show the combined characteristics of all selected objects in a single pane, select the All option.

Schematic Display Options

The following topics describe the schematic display options:

- [Collapse and Expand Objects](#)
- [Display Pin and Port Group Attributes](#)

Collapse and Expand Objects

You can collapse and expand the display of buffer chains, hierarchical objects, and unconnected pins in schematic views. Collapsing these types of objects can greatly simplify the schematic display by hiding the unimportant details.

For example, in [Figure 271](#), the chain of three buffers is selected. By choosing Collapse > Selected Buffers/Inverters on the pop-up menu, the chain is collapsed into a single “meta-buffer” that represents the functionality of the whole buffer chain, as shown in [Figure 272](#).

Figure 271 Buffer Chain Selected

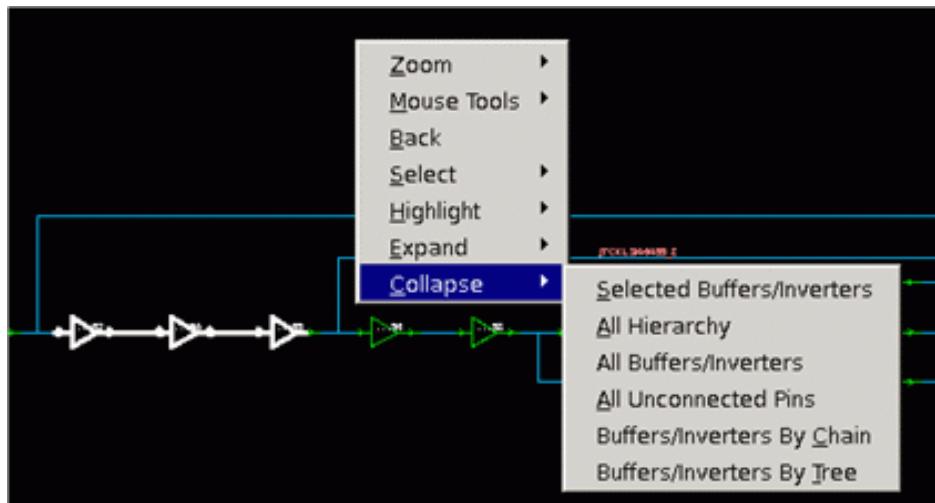
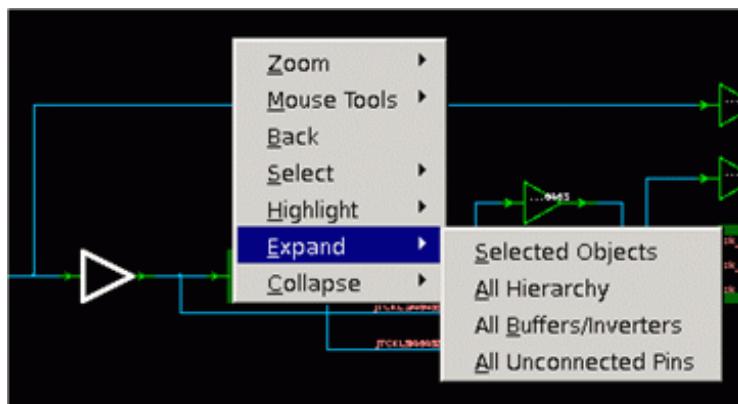


Figure 272 Buffer Chain Collapsed



The “meta-buffer” is drawn larger and with thicker lines to show that it is a compressed representation of the buffer chain, not an actual cell in the design. You can restore the view of the original buffer chain by choosing Expand > Selected Objects on the pop-up menu.

You can similarly collapse and expand the view of buffer trees, cells that belong to hierarchical blocks, and unconnected pins of large macro cells. The design functionality is often easier to understand with a collapsed view. [Figure 273](#) shows an example of an expanded (flat) and a collapsed (hierarchical) view of a path schematic.

Figure 273 Expanded and Collapsed Views of a Design Hierarchy



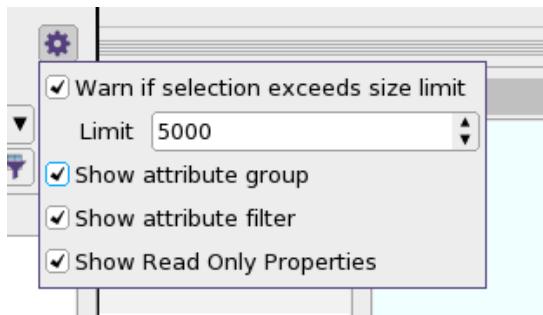
The commands for collapsing and expanding objects are available on the pop-up menu and on the menu bar. On the menu bar, choose Schematic > Collapse or Schematic > Expand and choose the command. You can choose commands to collapse or expand the currently selected objects or all objects of a specified type, such as All Hierarchy, All Buffers/Inverters, Buffers/Inverters By Chain, or Buffers/Inverters By Tree.

Display Pin and Port Group Attributes

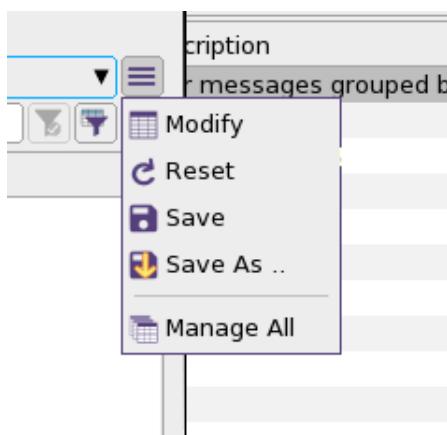
You can display the group of attributes for the pin and port objects in schematic views.

To display the group of attributes, follow these steps:

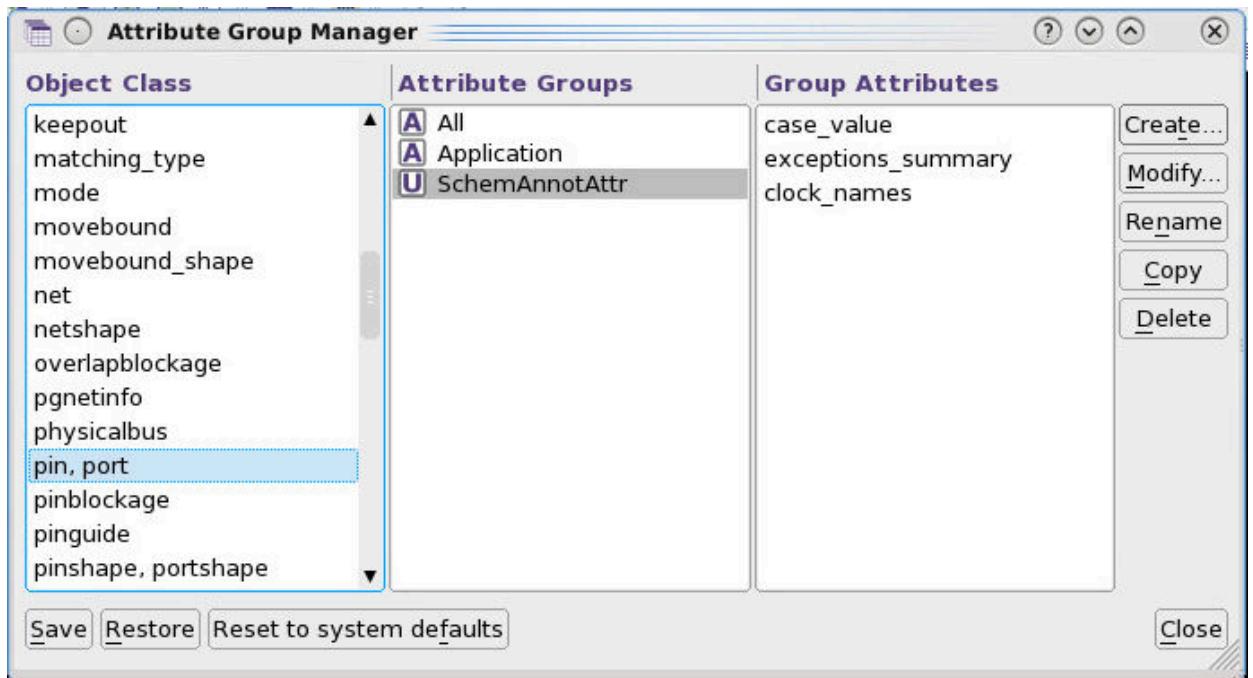
1. Right-click any element on the schematic, and click Properties.
2. Click the Settings button, and select Show attribute group.



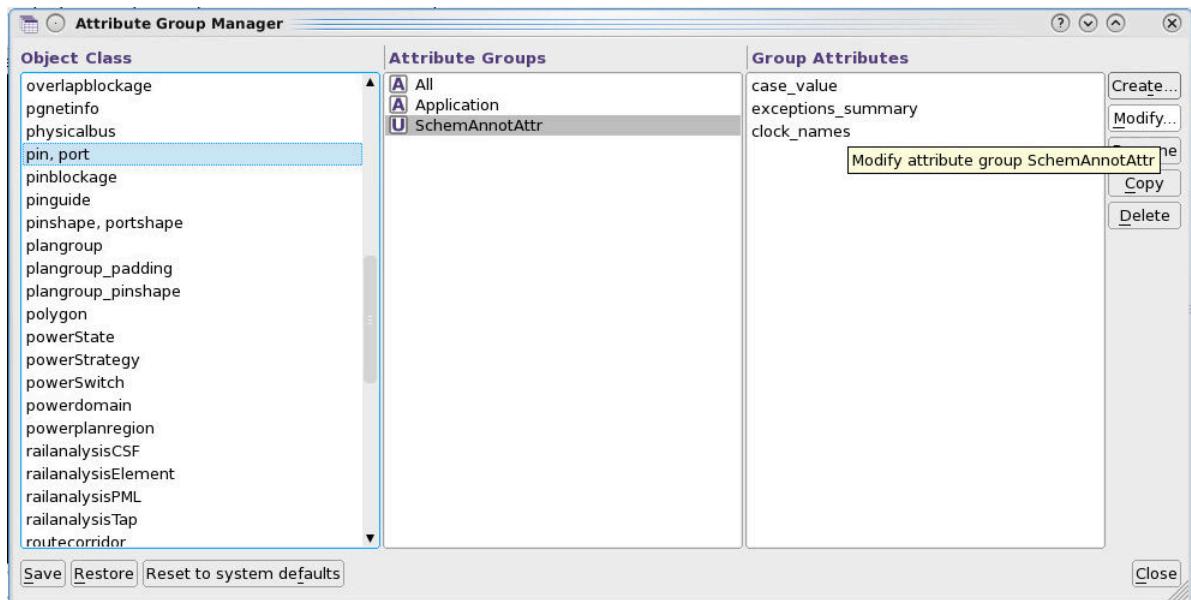
3. Click the Menu  button, and choose Manage All.



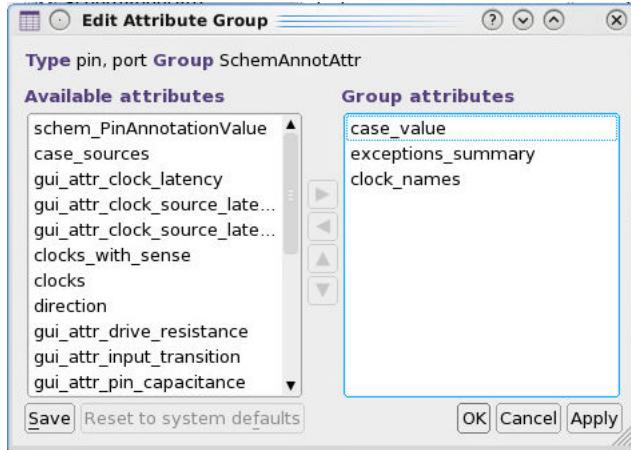
4. Select the pin, port Object Class, and the SchemAnnotAttr Attribute Group in the Attribute Group Manager window.



- Click the Modify button to open the Edit Attribute Group window.

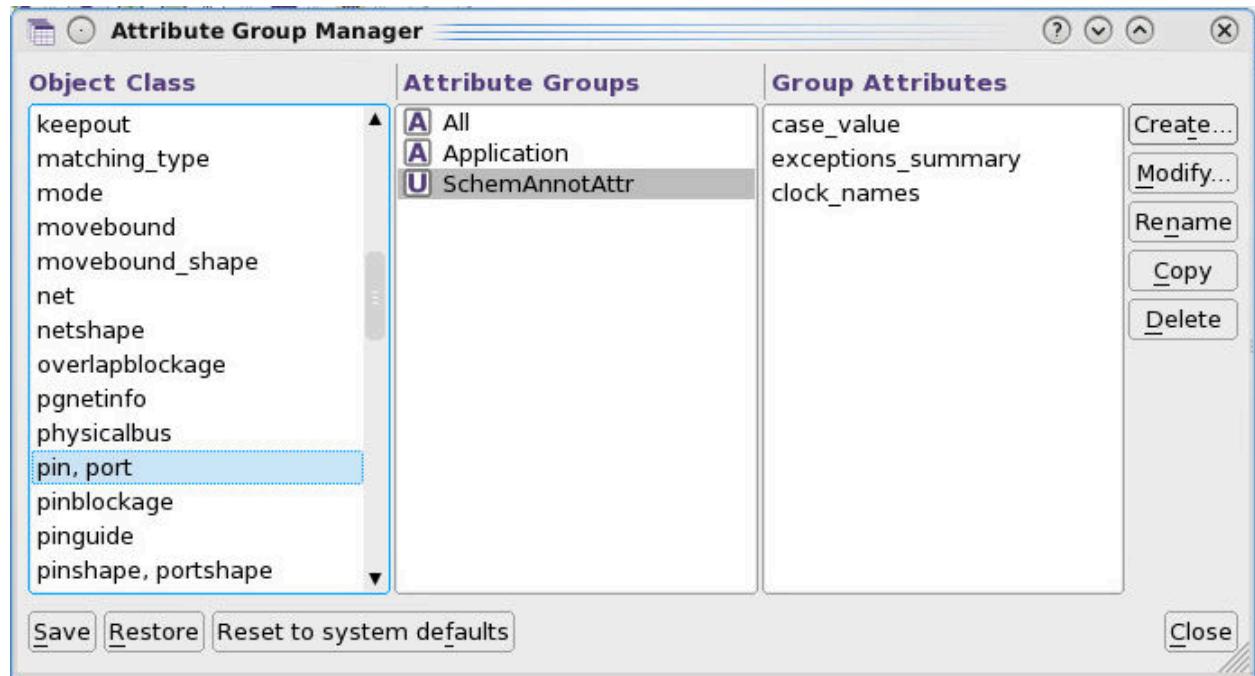


The Edit Attribute Group window shows the available attributes that you can add to the Group attributes. You can remove any attribute from the Group attributes to the Available attributes if you do not want to display it in the schematic.



For example, in [Figure 274](#), the `case_value`, `exceptions_summary`, and `clock_names` attributes will display in the schematic view for the pin and port objects.

Figure 274 Attribute Group Manager Window



Tutorial

The constraint consistency capabilities can help you identify problems with your constraints by checking them against a predefined set of rules. By using the debugging capabilities, you can investigate and resolve constraint problems.

This tutorial assumes that you are familiar with key components of constraint consistency.

This section shows you how to set up and run constraint consistency on the tutorial design and investigate the rule violations in the design.

This section contains the following topics:

- [Overview of Constraint Consistency](#)
 - [About the Tutorial Design](#)
 - [Starting Constraint Consistency](#)
 - [Analyzing the ChipLevel Design](#)
 - [Debugging Constraint Problems](#)
 - [Ending and Restarting the Tutorial Session](#)
-

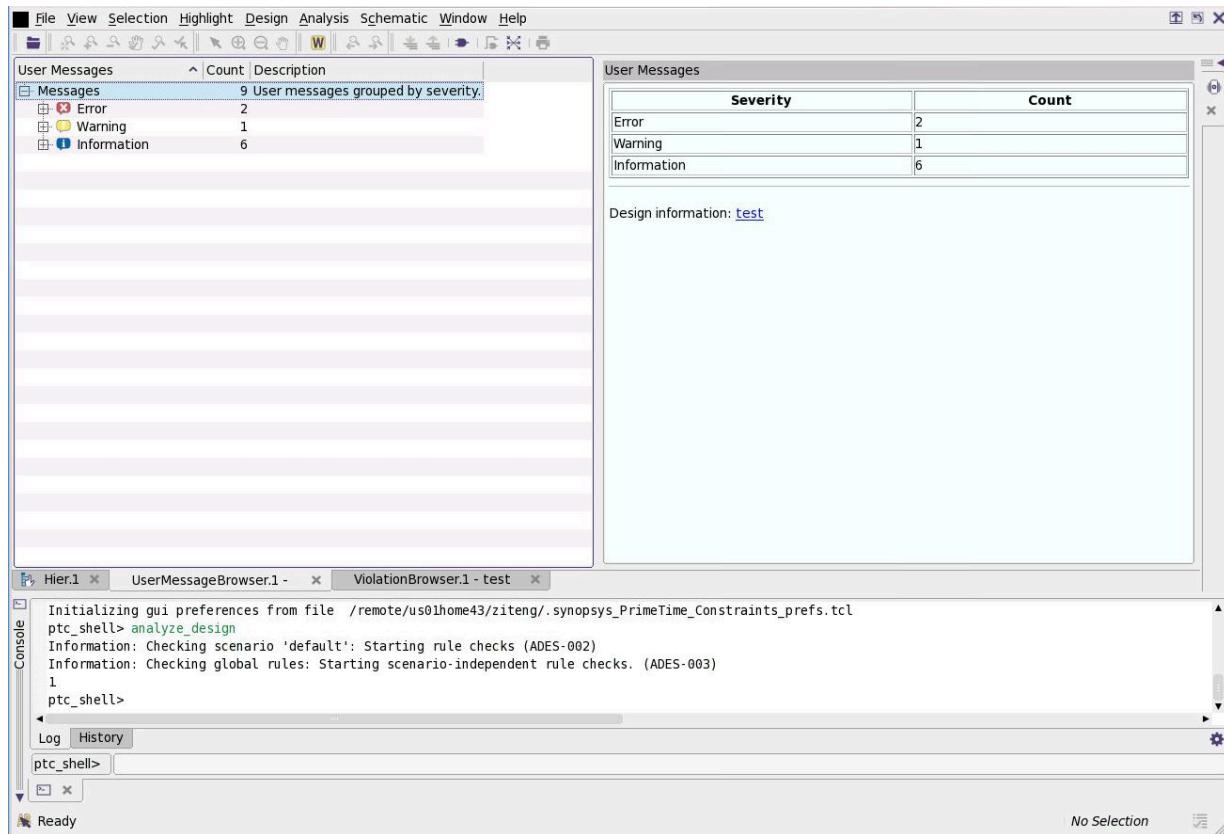
Overview of Constraint Consistency

The violation browser, shown in [Figure 275](#), is a key feature of constraint consistency. The violation browser displays a summary of the rule violations found by the `analyze_design` command. Each scenario has summary of Error, Warning, and Info rule violations. View the specific rule violations in one of the categories by double-clicking it or by clicking the “+” icon for that row. Double-clicking on a specific rule expands the view in the violation Browser to list all violations of that rule for that scenario.

The violation browser contains an area referred to as the information pane. This section displays information specific to the item selected in the violation browser. If a rule is selected in the violation browser, then more information is displayed about that rule in the information pane. If a violation for a rule is selected, the information pane displays specific information about that violation along with links to help you investigate the violation.

The console contains a command line for entering commands interactively during a session. It also displays the log view, which provides a transcript of the session. You can use the scroll bar to the right of the window to view previous commands and their output.

Figure 275 Constraint Consistency GUI Window



Use tabs to switch between different views that are available in some of the GUI components. The console displays the log view by default, but if you click the History tab, you can see a list of the commands you used in the session. If you click a Schematic link in the information pane, a schematic view replaces the violation browser. To revert to the previous view, click the violation browser tab. You can use all of these items in the GUI as you begin debugging constraint problems in the tutorial design. If you need to exit the tutorial, see “[Ending and Restarting the Tutorial Session](#)”.

When you run the `analyze_design` command, the loaded design is checked against a set of predefined rules. These rules check clock definitions, case analysis propagation, timing exceptions, boundary conditions, and other general checks. [Table 50](#) provides an overview of the different types of rule definitions.

Table 50 Rule Types

Design Analysis Intent	Rule	Rule Analysis
Boundary Conditions	CAP_xxxx rules	Capacitance values

Table 50 Rule Types (Continued)

Design Analysis Intent	Rule	Rule Analysis
	DRV_xxxx rules	Drive constraints
	EXD_xxxx rules	External delays
Constraints/Exceptions Analysis	CAS_xxxx rules	Case analysis
	EXC_xxxx rules	Timing exceptions
Clocks	CGR_xxxx rules	Clock groups
	CLK_xxxx rules	Clock properties
	CNL_xxxx rules	Network latencies
	CSL_xxxx rules	Clock source latencies
	CTR_xxxx rules	Clock transition
	PRF_xxxx rules	Clock count
	UNC_xxxx rules	Clock uncertainties
	DES_xxxx rules	Design constraints
	LOOP_xxxx rules	Timing loops
	NTL_xxxx rules	Net properties
	UNT_xxxx rules	Library units
Block-To-Top	B2T_xxx_xxxx rules	Top-level constraints vs. block-level constraints

Each rule defines a specific check that is performed on the design and its constraints. To see a short description of each case analysis rule, enter the following command:

```
ptc_shell> report_rule CAS_*
```

The command returns a report like the one in [Example 83](#).

Example 83 report_rule Output for CAS_* Rules

```
*****
Report : rule
...
*****
Rule      Severity   Status    Message
-----
```

CAS_0001	error	enabled	Net 'net' has conflicting user case analysis values on load pins.
CAS_0002	error	enabled	Pin/Port 'pin' is driven by conflicting values. A value of zero is used.
CAS_0003	error	enabled	Pin/Port 'pin' propagated value conflicts with a user case analysis value.
CAS_0004	error	enabled	Bidirectional pin 'pin' has conflicting values on the load and driver side.

For more details, see the rules in the online Help. To launch the online Help, choose Help > Online Help from the menu in the GUI.

About the Tutorial Design

The tutorial uses a design called “ChipLevel” that implements several binary arithmetic functions such as addition and multiplication.

The ChipLevel design is hierarchical and consists of the following major blocks:

- Adder16
- CascadeMod
- Comparator
- Multiply8x8
- Multiply16x16
- MuxMod
- PathSegment

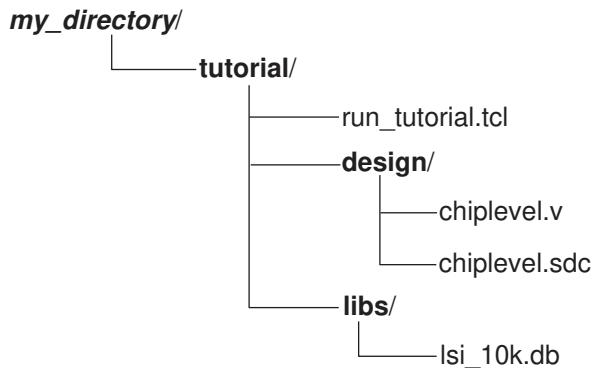
This pre-layout design is in the early development phase, with the clock network and constraints still under construction. You can use constraint consistency to identify constraint problems early in the design cycle.

Constraint consistency must be installed before you run the tutorial. The tutorial directories and files are located under the `doc/gca/tutorial` directory of the PrimeTime installation. Copy this directory to a working directory by entering the following Linux command:

```
% cp -r $SYNOPSYS/doc/gca/tutorial .
```

This command creates a new directory called “tutorial” under the current directory. If you copied the tutorial design files from the PrimeTime installation, you have the directory structure shown in [Figure 276](#) in your working directory.

Figure 276 Tutorial Directory Structure



Run the tutorial from the tutorial directory using the `runTutorial.tcl` script. The subdirectories are:

- `design`

This directory contains the Verilog netlist (`chiplevel.v`) and design constraints (`chiplevel.sdc`).

- `libs`

This directory contains the `lsi_10k.db` library.

Starting Constraint Consistency

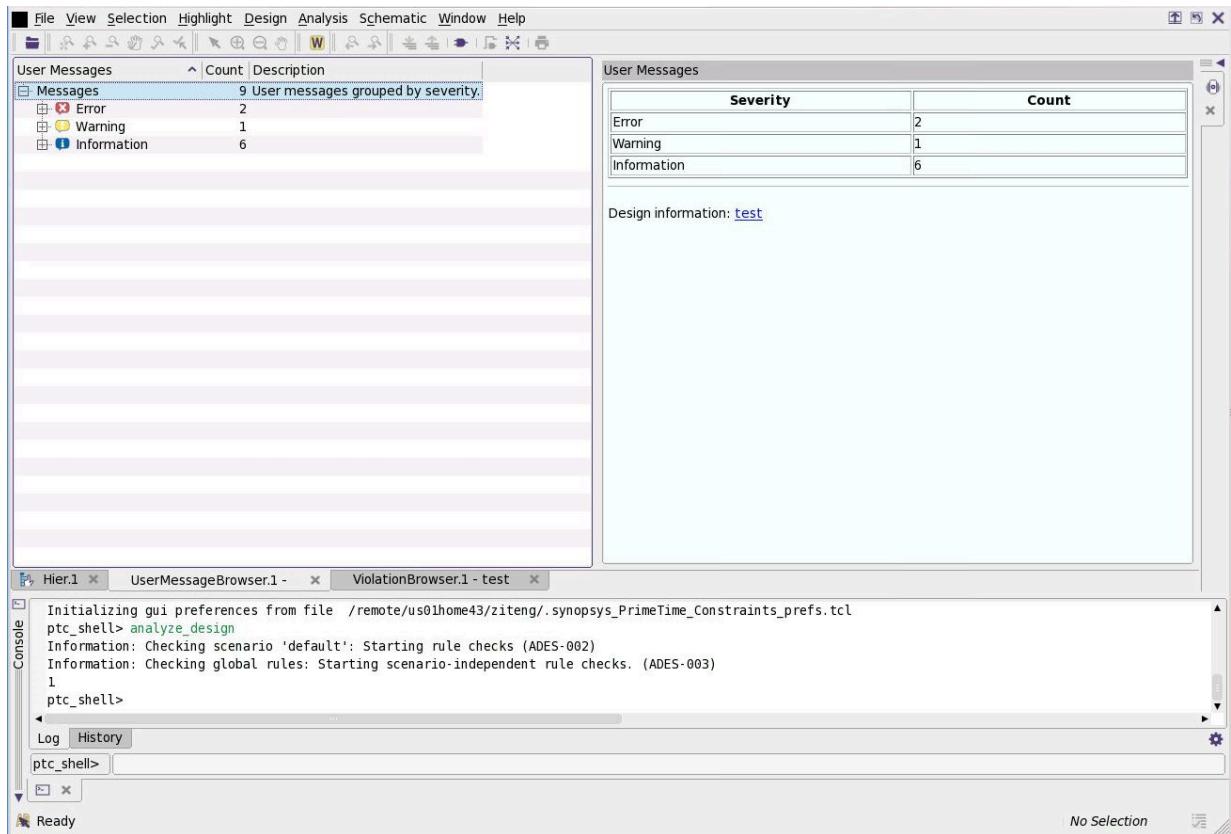
Start constraint consistency from the tutorial directory. To start:

1. Open a Linux shell.
2. Change to the `user_directory/tutorial` directory.
3. At the Linux prompt, enter:

```
% pt_shell -constraints
```

The Constraint Consistency GUI opens as shown in [Figure 277](#). Files can be sourced and commands can be entered at the `ptc_shell` prompt. All commands and their output are saved to the `ptc_shell_command.log` file.

Figure 277 Constraint Consistency GUI



If constraint consistency fails to start, check the following:

- Was constraint consistency correctly installed?
- Is the PrimeTime installation path included in your path definition?
- Is the Synopsys license server running?
- Is your Synopsys license key file available and current, and does it include a PrimeTime and PrimeTime SI license? Constraint consistency requires that you have a PrimeTime license available, although it is not checked out.

If you need assistance, ask your system administrator or consult the installation and licensing documentation.

Analyzing the ChipLevel Design

Constraint consistency requires the design libraries, the gate-level Verilog netlists of the design, and the files containing the constraints for the design. You must read in the design,

link it to the logic libraries, and load the constraints before constraint consistency can analyze the constraints.

Follow the steps below to analyze the constraints in the ChipLevel design.

1. Set the `search_path` variable:

```
ptc_shell> set_app_var search_path [list . ./libs ./design ] \  
    . ./libs ./design
```

The `search_path` variable specifies locations to search for design and library files.

2. Set the `link_path` variable:

```
ptc_shell> set_app_var link_path [list * lsi_10k.db] * lsi_10k.db
```

Instead of the Tcl `set` command, the `set_app_var` command is recommended for assigning values to application shell variables because it verifies that the variable is an application variable and that it is a legal value for that variable in the application.

The `link_path` variable specifies a list of libraries, design files, and library files to use for resolving references during linking. The asterisk (*) indicates that constraint consistency should use the designs and libraries that have already been read into memory for design linking.

3. Read in the Verilog netlist:

```
ptc_shell> read_verilog chiplevel.v  
Loading verilog file '/remote/techp5/grayc/gca/tutorial/design \  
chiplevel.v'  
1
```

The `read_verilog` command reads in the gate-level Verilog netlist for the design.

4. Set the current design to the top-level design:

```
ptc_shell> current_design ChipLevel  
{"ChipLevel"}
```

The `current_design` command specifies the working design for the session.

5. Link the design:

```
ptc_shell> link_design  
Loading db file '/remote/techp5/grayc/gca/tutorial/libs/lsi_10k.db' \  
Information: units loaded from library 'lsi_10k' (LNK-040) \  
Removing previously linked designs ... \  
Design 'ChipLevel' was successfully linked.  
1
```

The `link_design` command resolves the references in the Verilog netlist to the library components and builds an internal representation of the design for analysis.

6. Read the constraints file:

```
ptc_shell> source ./design/chiplevel.sdc
Information: Setting sdc_version outside of an SDC file has no
effect (SDC-1)
Warning: set_max_delay is forcing pin 'clk_8x8/Z' to be a timing
startpoint. (UIC-011) \
Warning: Object 'u5/res_reg[0]/Q' is not a valid endpoint. The
set_multicycle_path command will not match this object. (UIC-009)
Warning: Object 'u5/res_reg[1]/Q' is not a valid endpoint. The
set_multicycle_path command will not match this object. (UIC-009)
...
Warning: Object 'u5/res_reg[31]/Q' is not a valid endpoint. The
set_multicycle_path command will not match this object. (UIC-009)
1
```

The `source` command reads in the constraint file for the design. The `read_sdc` command could also be used because the file only contains SDC commands.

7. Analyze the design:

```
ptc_shell> analyze_design
Warning: Conflicted case values driving and set at pin u7/U168/S.
Using the set logicvalue of '0'. (CASE-003)
Information: Checking scenario 'default': Starting rule
checks (ADES-002)
Information: Checking netlist: Starting scenario-independent rule
checks. (ADES-003)
1
ptc_shell>
```

After you run the `analyze_design` command, the results of the rule checking displays in the violation browser, as shown in [Window Types on page 778](#).

The `analyze_design` command is used to run analysis on your design after the necessary inputs have been loaded. The design and its constraints are checked against a predefined set of rules, and the results of the analysis are summarized in the violation browser in the GUI.

Note:

The commands used in the steps above are provided in a script file located in the tutorial directory. The file name is `run_tutorial.tcl`. The content of the `run_tutorial.tcl` script is shown in [Example 84](#).

Example 84 ChipLevel Script to Read in Design, Link, and Load Constraints

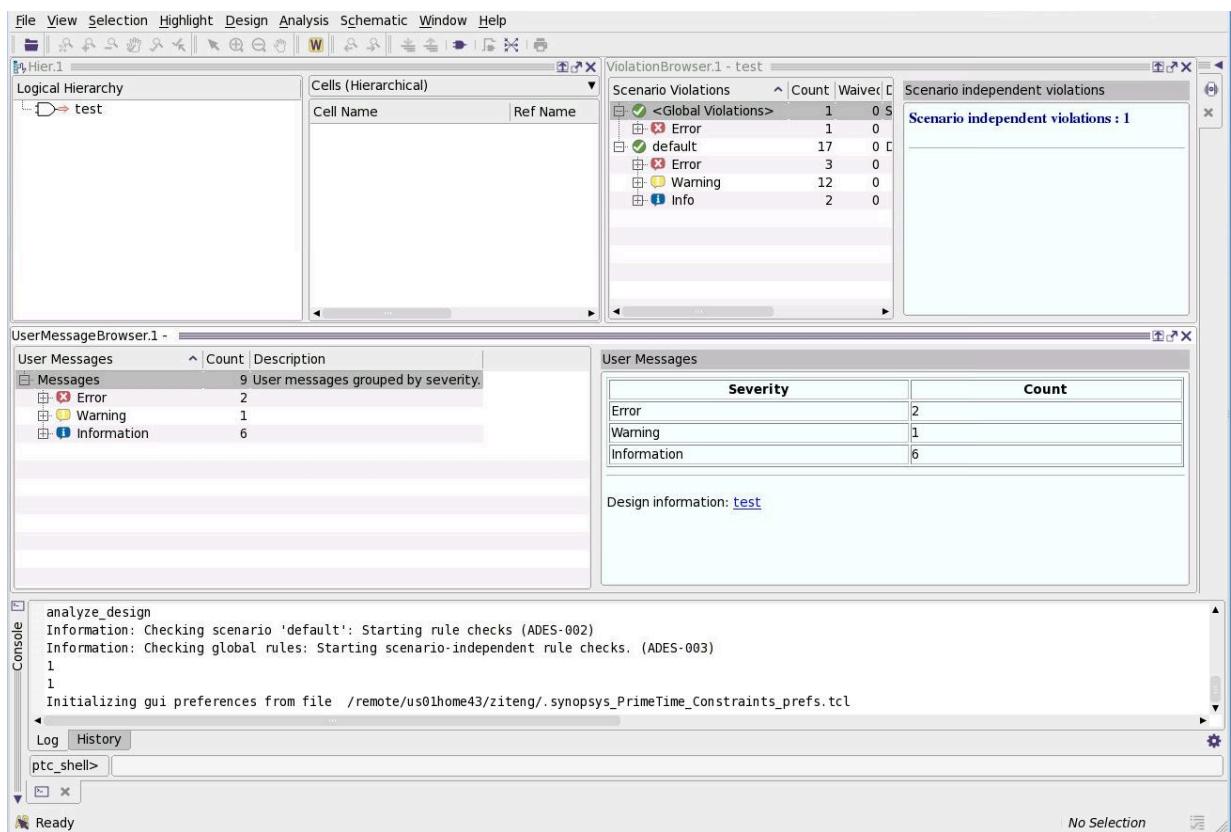
```
#####
#####
#      link design
#####
#####
```

```

set_app_var search_path [list . ./libs ./design ]
set_app_var link_path [list * lsi_10k.db]
read_verilog chiplevel.v
current_design ChipLevel
link_design
#####
#      read SDC
#####
source ./design/chiplevel.sdc
#####
#      run design analysis
#####
analyze_design

```

Figure 278 Constraint Consistency Window on Startup



Constraints are checked for multiple scenarios of the same design. For example, a design can have both a functional mode and test mode, and it can have different boundary

conditions for worst-case and best-case analysis. For more information, see “[Additional Analysis Features](#)”.

Debugging Constraint Problems

The constraint problems found by the `analyze_design` command are displayed in the violation browser, as shown in [Figure 279](#). The violations are organized by scenario in the browser and are grouped by their severity: error, warning, or information. Error and Warning violations can identify constraint inconsistencies and impact design analysis, runtime, and memory usage. Error violations are the most severe violations. Information violations notify you of situations that might be the result of a mistake in the constraints.

Use the GUI to investigate and debug the constraint problems that are reported.

Figure 279 Violation Browser for the ChipLevel Design

Scenario Violations	Count	Waived	Description
☐ <Global Violations>	1	0	Scenario independent violations
☐ Error	1	0	
☐ UNT_0002	1	0	Library 'library' has incomplete units defined.
☐ 1 of 1	1	0	Library '/remote/us01home31/catambay/gca_tutorial/tutorial/libs/lsi_10k.db:lsi_10k' has incomplete units defined.
☐ default	278	0	Default scenario violations
☐ Error	3	0	
☐ CAS_0003	1	0	Pin/Port 'pin' propagated value conflicts with a user case analysis value.
☐ 1 of 1	1	0	Pin/Port 'u7/U168/S' propagated value conflicts with a user case analysis value.
☐ CLK_0003	1	0	Generated clock 'clock' is not expanded because it has no clock reaching its master source 'pin'.
☐ 1 of 1	1	0	Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its master source 'pin'.
☐ EXC_0008	1	0	A 'exception' is forcing start and/or end points, breaking the clock network.
☐ 1 of 1	1	0	A 'max_min_delay' exception at /remote/us01home31/catambay/gca_tutorial/tutorial/design/clock.
☐ Warning	272	0	
☐ Info	3	0	

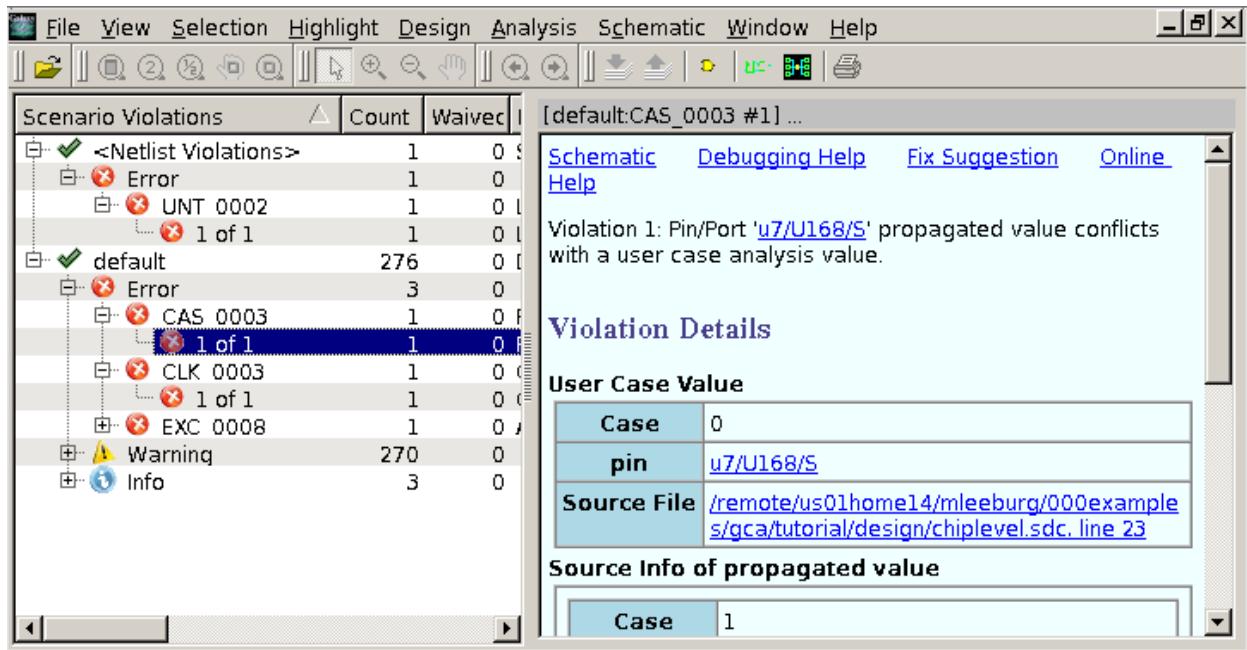
You can display the violations in each category and view the number of violations for each rule by double-clicking on the row of interest in the violation browser or by clicking on the '+' icon at the beginning of each row.

Investigating the CAS_0003 Violation

Begin investigating the CAS_0003 violation by double-clicking the row containing this rule in the violation browser and then clicking on the row listing the violation for this rule, as shown in [Figure 280](#).

The information pane to the right displays details about the CAS_0003 violation for the ChipLevel design.

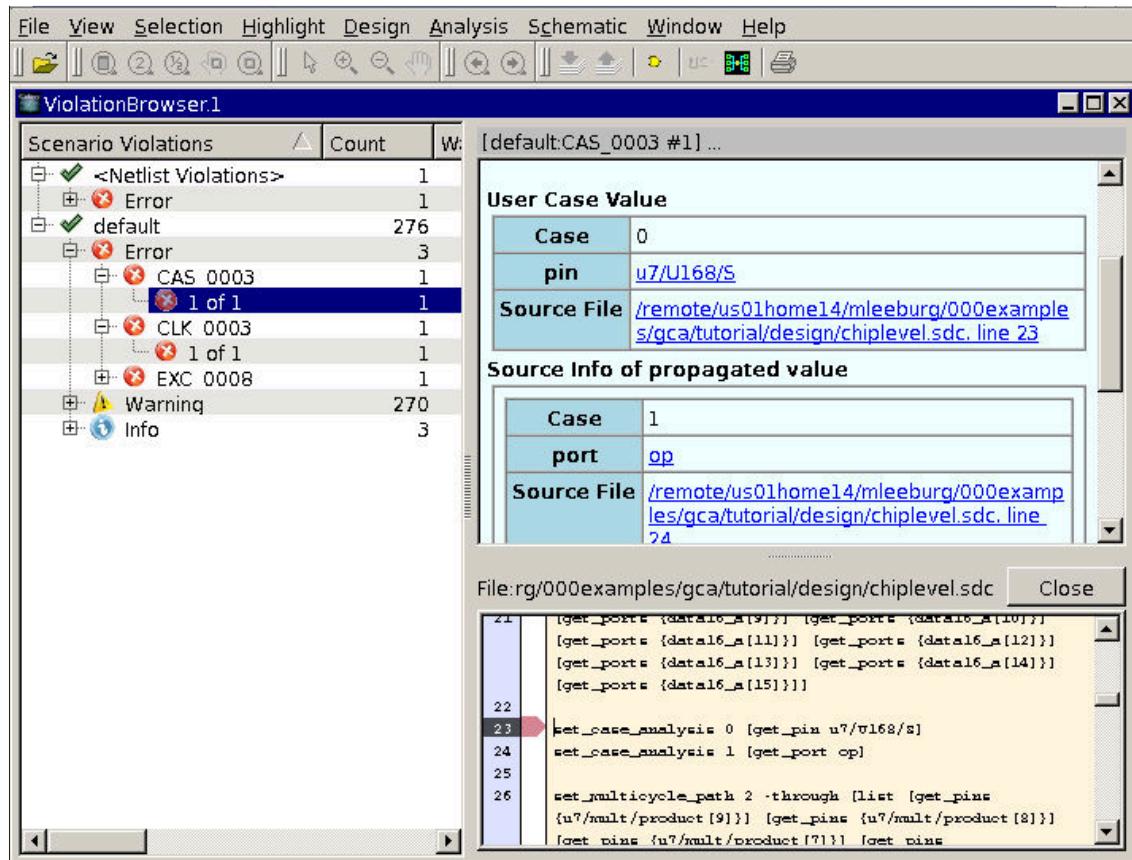
Figure 280 Information Pane Displaying CAS_0003 Rule Violation Details



The Violation Message (Violation 1) in the information pane states that a propagated logic constant or case analysis value conflicts with a user-specified case analysis value on a pin or port. The message includes the pin name, u7/U168/S, with the conflicting values.

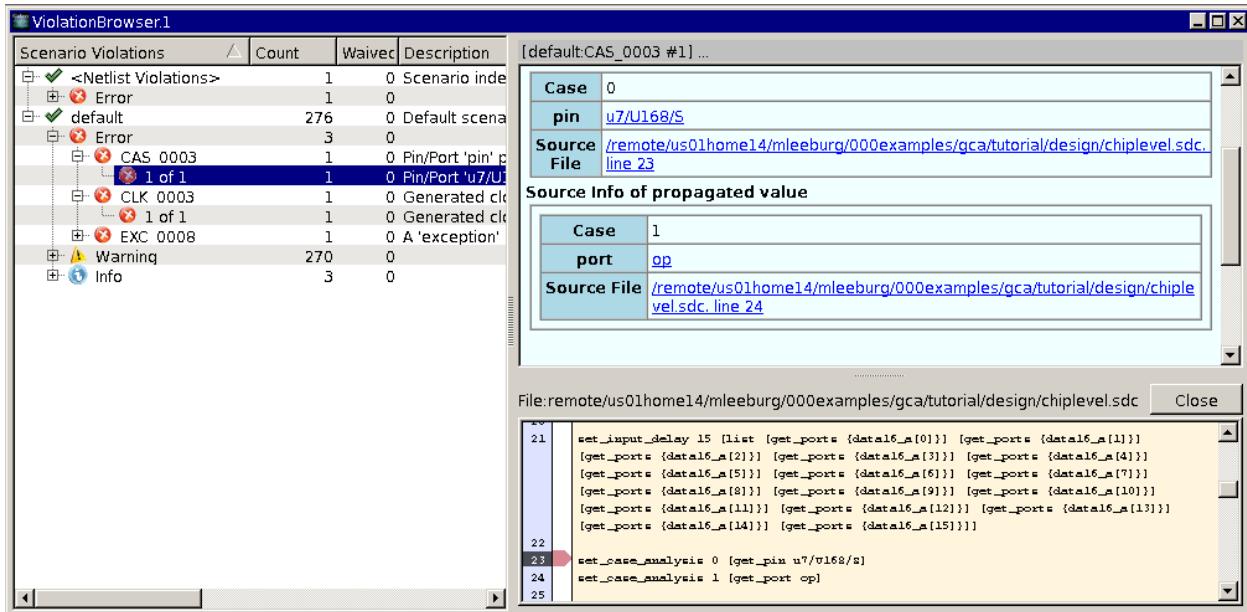
The User Case Value section shows that a user-specified case analysis value of 0 has been set on this pin. The source of the conflicting case analysis value is the port named "op" as shown in the Source Info of propagated value in the Violation Details. To view the constraints related to the case analysis values, click the link in the Source File box. When you click the link, the constraint browser opens in the information pane, and a marker points to the constraint in the SDC file as shown in Figure 281.

Figure 281 Constraint Browser for the User Case Value in the Violation Details



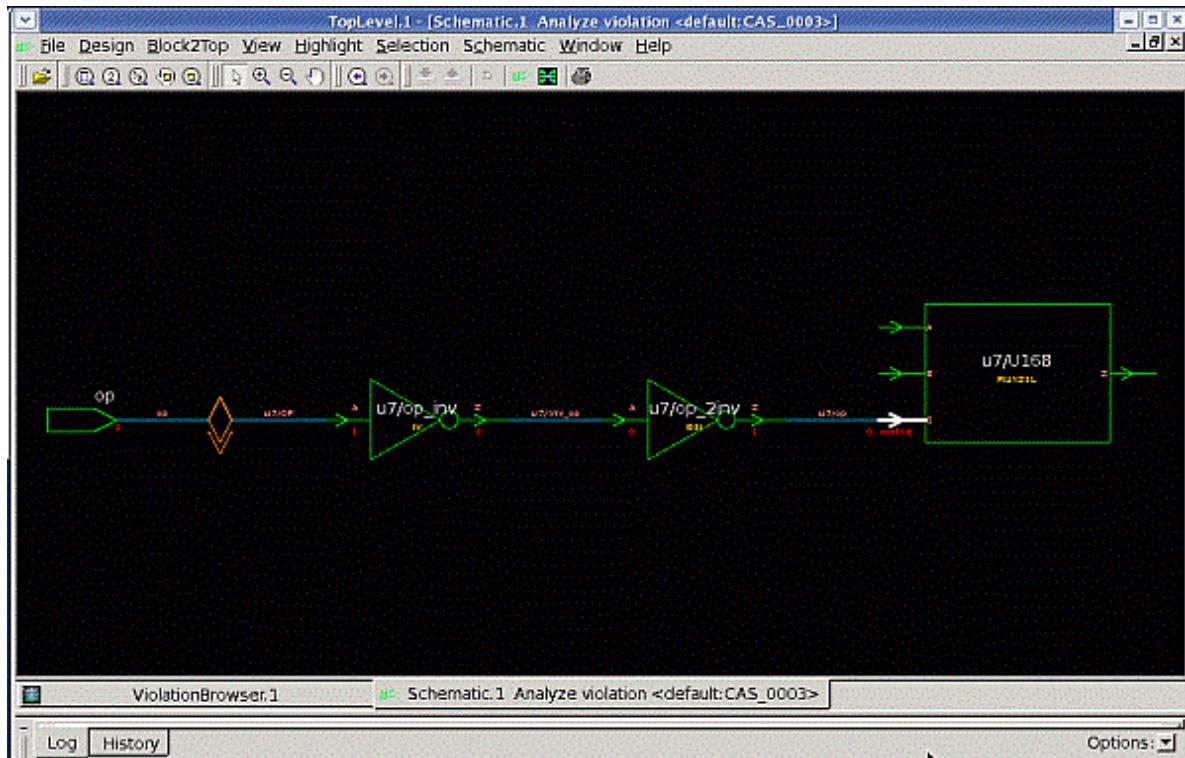
Similarly, by clicking on the Source File link under Source Info of propagated value, the constraint browser moves to the line in the SDC file related to the port named “op” as shown in Figure 282.

Figure 282 Constraint Browser for Source Info of propagated value in Violation Details



To understand how the case analysis value on the port named "op" propagates to the violating pin, scroll up to the top of the information pane and click the Schematic link. The link opens a separate window, shown in Figure 283 that indicates the location of the pin with the conflicting case analysis value.

Figure 283 Schematic View of CAS_0003 Rule Violation



The schematic shows that a case value of 1 is present at the u7/op_2inv/Z pin, and a case value of 0 is present at the u7/U168/S pin. This pin has been highlighted in the schematic and marked as the source of the case analysis conflict.

The schematic shows the case analysis value of 1 on the op port, which propagates through the u7/op_inv inverter to a case value of 0 on its output, and then through the u7/op_2inv inverter, resulting in a case value of 1 on its output. Constraint consistency highlights this situation as a case analysis conflict.

You can also view the same information in report form. First, click the ViolationBrowser.1 tab to go back to the violation browser and information pane. Then click the Debugging Help link in the information pane.

The Debugging Help section related to the violation is displayed in the information pane as shown in Figure 284.

Figure 284 Viewing the Debugging Help for the CAS_0003 Rule Violation



When you click the Execute link under Debugging Help, the `report_case_details` command runs for the related violation. A report similar to the one shown in [Example 85](#) is displayed in the console log view.

Example 85 Case Propagation Details Report

```
ptc_shell> report_case_details -to u7/U168/S
*****
Report : case_details
        -to
        -max_objects = 1000
Design : ChipLevel
...
*****
Properties      Value   Pin/Port
-----
user case          0       u7/U168/S (MUX21L)
Case fanin report:
Verbose Source Trace for pin/port u7/U168/S:
Path number: 1
Path Ref #  Value   Properties           Pin/Port
-----
          0      user case           u7/U168/S (MUX21L)
          1      F()=A'             u7/op_2inv/Z (B4I)
          0      F()=A'             u7/op_2inv/A (B4I)
          0      F()=A'             u7/op_inv/Z (IV)
          1                  u7/op_inv/A (IV)
          1      user case           op (in)
```

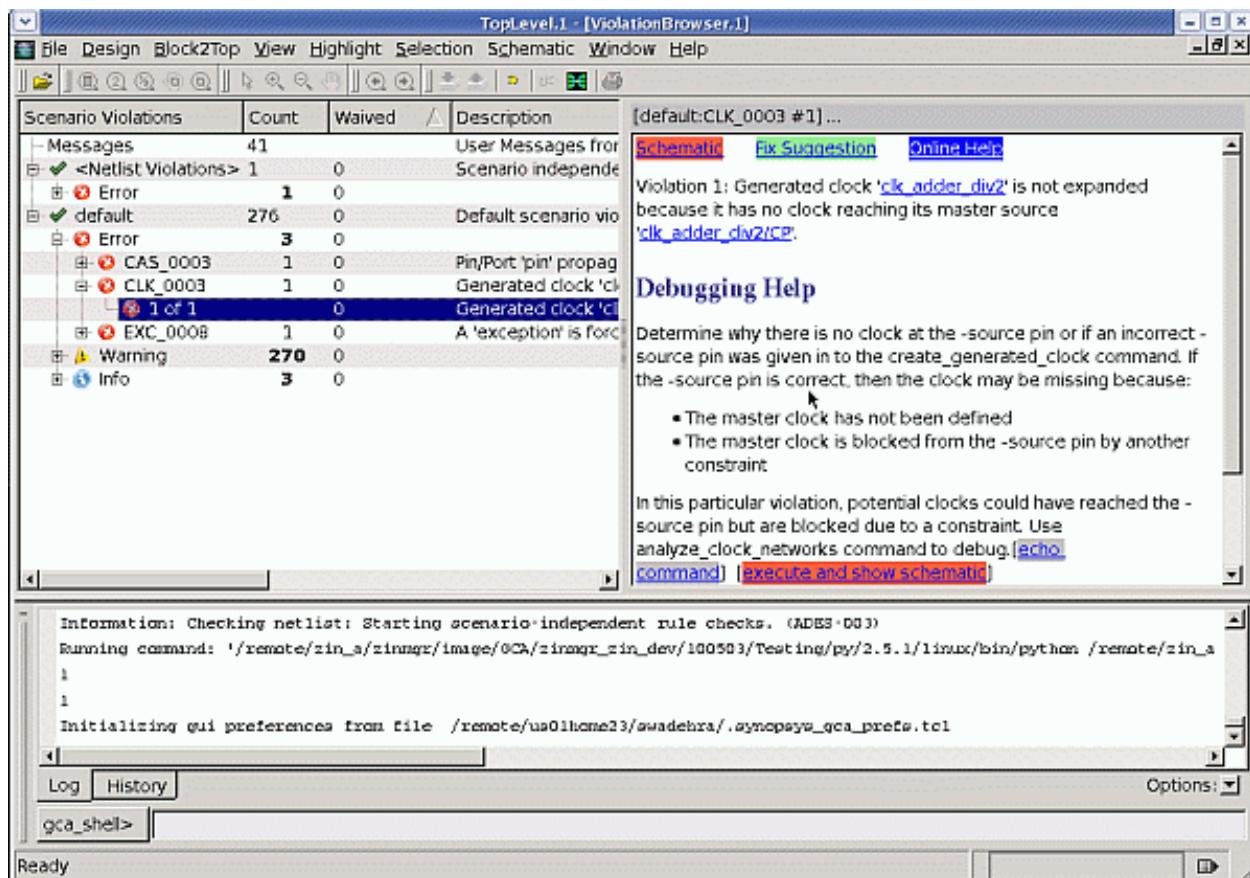
The beginning of the report shows the user-set case analysis value set directly on the u7/U168/S pin. The “Case fanin report” section shows the conflicting value propagated to the same pin, along with the path of the propagated values through multiple gates, originating from a case analysis value set on the “op” port.

The decision on how to resolve the CAS_0003 rule violation is design-dependent and based on knowledge of the design and mode being analyzed. In some cases, removing the case analysis on the u7/U168/S pin might be the solution, and in other cases, removing the case analysis on the port might be the solution. The conflict between propagated and user-specified case values should be resolved to avoid incomplete timing analysis or analysis of false paths for the operational mode of the design.

Investigating the CLK_0003 Violation

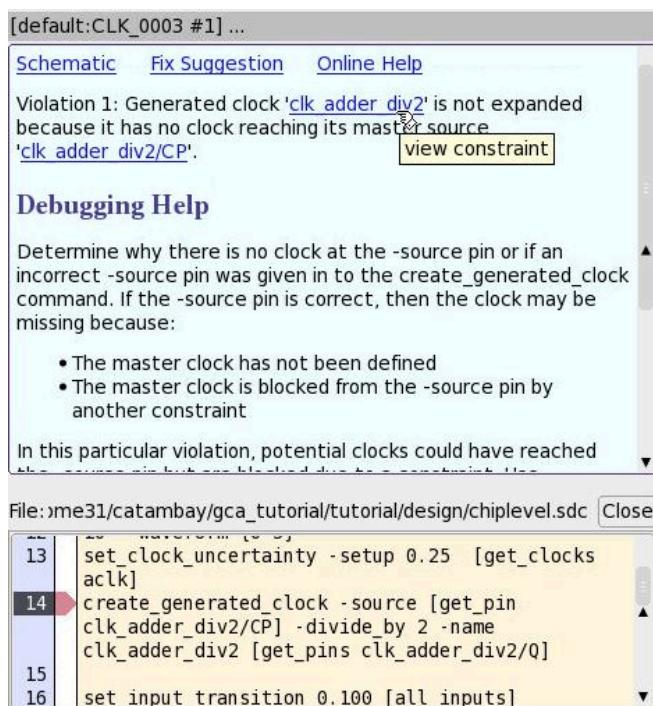
To investigate the CLK_0003 rule violation, go to the violation browser and click the violation for the CLK_0003 rule. The information pane is updated with information about the CLK_0003 rule violation as shown in Figure 285.

Figure 285 Violation Browser and Information Pane Displaying CLK_0003 Rule Violation Details



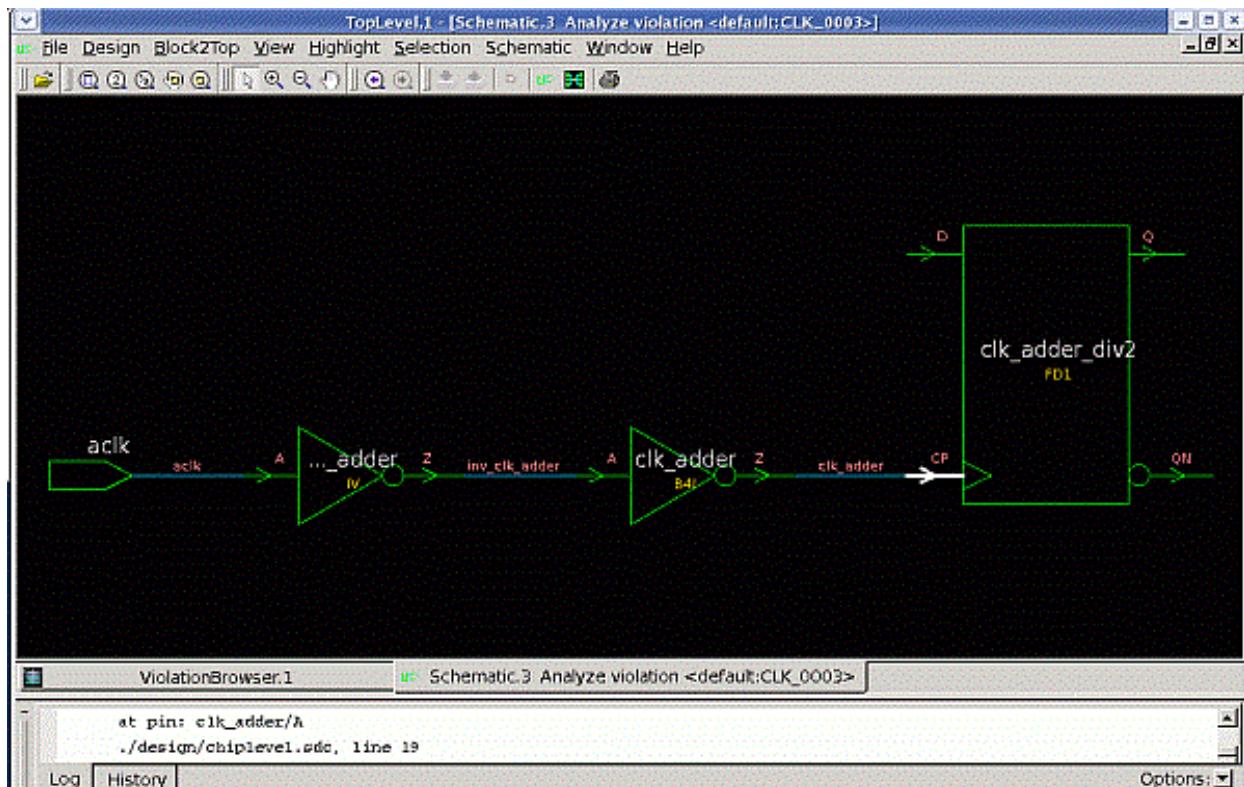
The CLK_0003 rule highlights a problem with generated clock propagation due to the lack of a master source as shown in the Violation 1 message. The pin or port specified as the master source of the generated clock, clk_adder_div2/CP, does not have a clock reaching it. You can view the definition of the generated clock in the constraint file by clicking on the clk_adder_div2 link (the name of the generated clock) in the Violation 1 message. The constraint browser opens and displays the generated clock definition as shown in Figure 286.

Figure 286 Definition of clk_adder_div2 Generated Clock in Constraint File



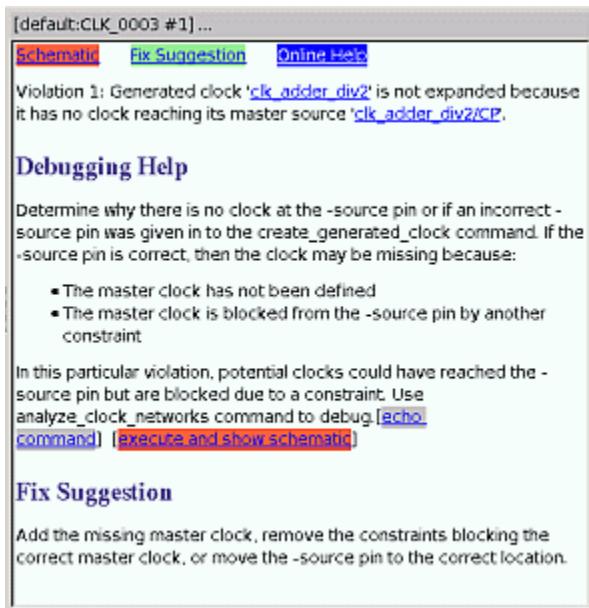
The constraint browser confirms that the pin clk_adder_div2/CP has been defined as the master clock source for the generated clock. To view the related schematic, click the Close button of the constraint browser and then click the Schematic link in the information pane. The schematic opens in a separate window as shown in Figure 287.

Figure 287 Schematic for CLK_0003 Rule Violation



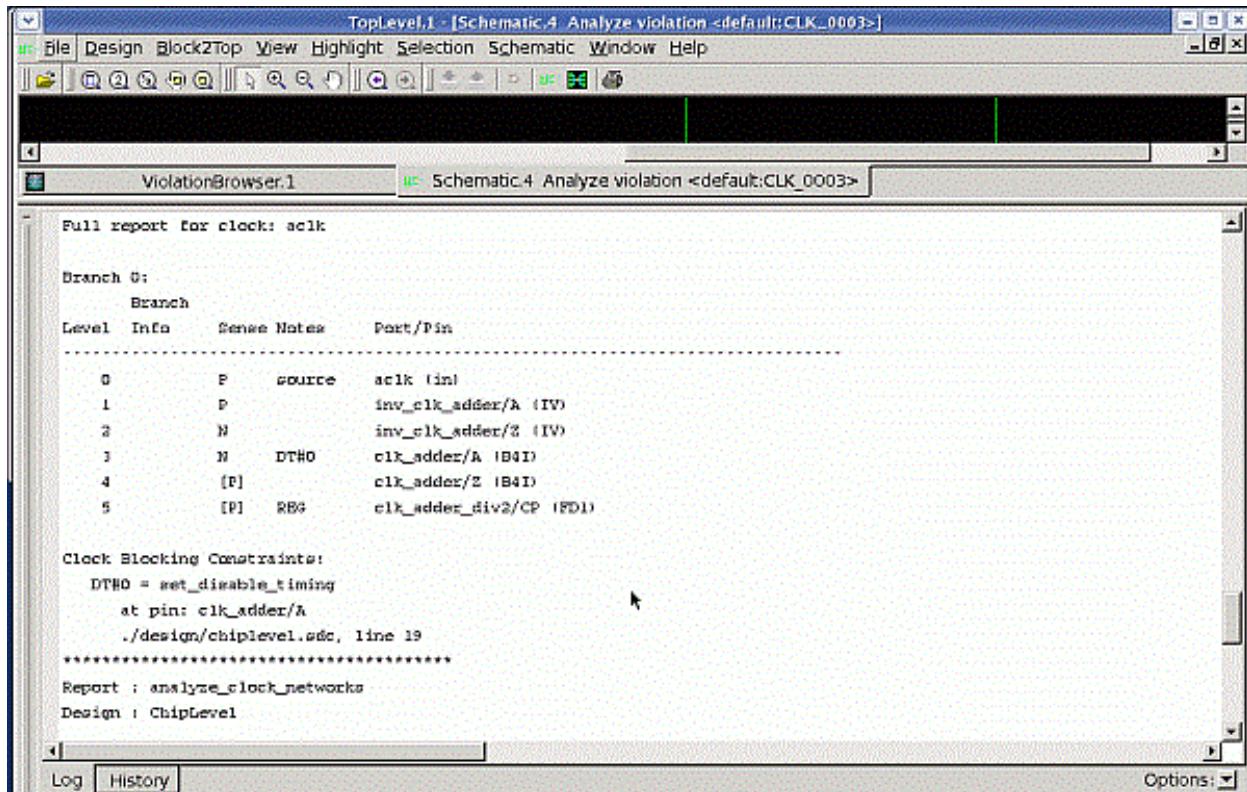
The schematic shows that there is a path from the ack port to the master source for the generated clock, clk_adder_div2/CP. To see what might be causing the problem, go back to the violation browser and look at the Debugging Help section following the violation message, as shown in Figure 288.

Figure 288 Debugging Help for the CLK_0003 Rule Violation



The Debugging Help states that there might be potential clocks that have reached the clk_adder_div2/CP pin. To run the `analyze_clock_networks` command to help debug this violation, click the “execute and show schematic” link under Debugging Help. The schematic related to the violation is displayed again, and the report for the `analyze_clock_networks` command is displayed in the GUI console as shown in Figure 289.

Figure 289 Partial View of the analyze_clock_networks Report in GUI Console



View the report by scrolling up and down in the GUI console window, or use the echo command link under Debugging Help and the redirect command to send the entire report to a text file. For this violation, the following is an example of the redirect command:

```
ptc_shell> redirect -file clk_0003_report.txt \
  { analyze_clock_networks -from [get_clocks {aclk}] \
  -through [get_pins {clk_adder_div2/CP}] -max_endpoints 1 -style full \
  -end_types {register port clock_source} -traverse_disabled -nosplit }
```

The `-traverse_disabled` option of the `analyze_clock_networks` command ensures that the portions of the clock network that are disabled due to a constraint are reported as potential clock networks and that the related constraints that are causing the clock to be blocked are included. The full report is shown in [Example 86](#).

Example 86 Report for analyze_clock_networks for CLK_0003 Violation

```
*****
Report : analyze_clock_networks
Design : ChipLevel
Scenario: default
...
*****
```

```

Clock Sense Abbreviations:
  P - positive
  N - negative
Potential senses detected with -traverse_disabled:
  [P] - potential positive
Clock Network End Type Abbreviations:
  REG - register
Full report for clock: aclk
Branch 0:
  Branch
Level Info      Sense Notes      Port/Pin
-----
  0            P    source      aclk (in)
  1            P
  2            N
  3            N    DT#0      clk_adder/A (B4I)
  4          [P]
  5          [P]    REG      clk_adder/Z (B4I)
                           clk_adder_div2/CP (FD1)

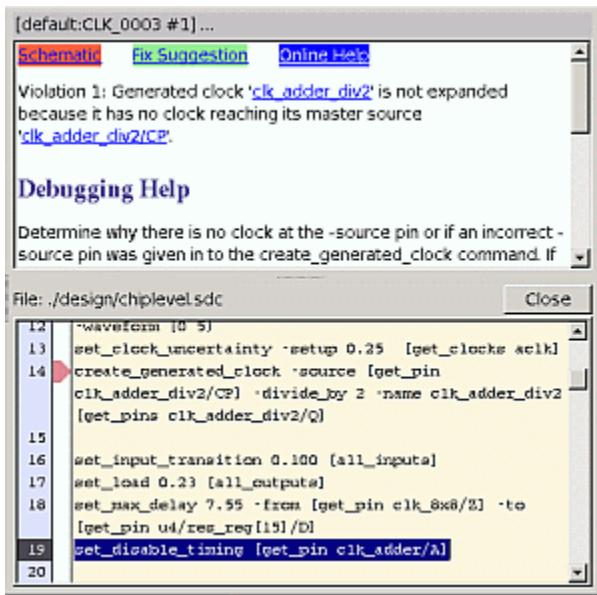
Clock Blocking Constraints:
  DT#0 = set_disable_timing
  at pin: clk_adder/A
  ./design/chiplevel.sdc, line 19

```

The report traces the clock network that begins at port `aclk` and identifies the point in the clock path where the clock stops propagating and the related constraint. A detailed examination of the report shows that the clock `aclk` is a positive-sense clock as shown by the `P` in the `Sense` column of the report. It then becomes a negative-sense clock `N` at the output of the `inv_clk_adder` inverter in the clock network. The output of the next inverter, `clk_adder`, inverts the clock again to a positive-sense clock, but the report shows that the clock propagation has stopped.

A potential clock has been detected on the output of the `clk_adder` inverter with a positive sense, as shown by the `[P]` in the `Sense` column of the report. The `Notes` column shows a reference, `DT#0`, to the Clock Blocking Constraints section of the report. This shows that a `set_disable_timing` constraint has been specified on the pin `clk_adder/A` which is blocking the clock propagation. Similarly, the generated clock master source, `clk_adder_div2/CP`, has a potential positive clock `[P]` because it is downstream of the `clk_adder/A` pin. The Clock Blocking Constraints section also indicates that the relevant constraint is on line 19 of the constraint file. Because this is the same constraint file that contains the generated clock definition, you can open the Constraint Browser again by clicking on the `clk_adder_div2` link and scrolling to line 19, as shown in [Figure 290](#).

Figure 290 Viewing Clock Blocking Constraint Identified in analyze_clock_networks Report



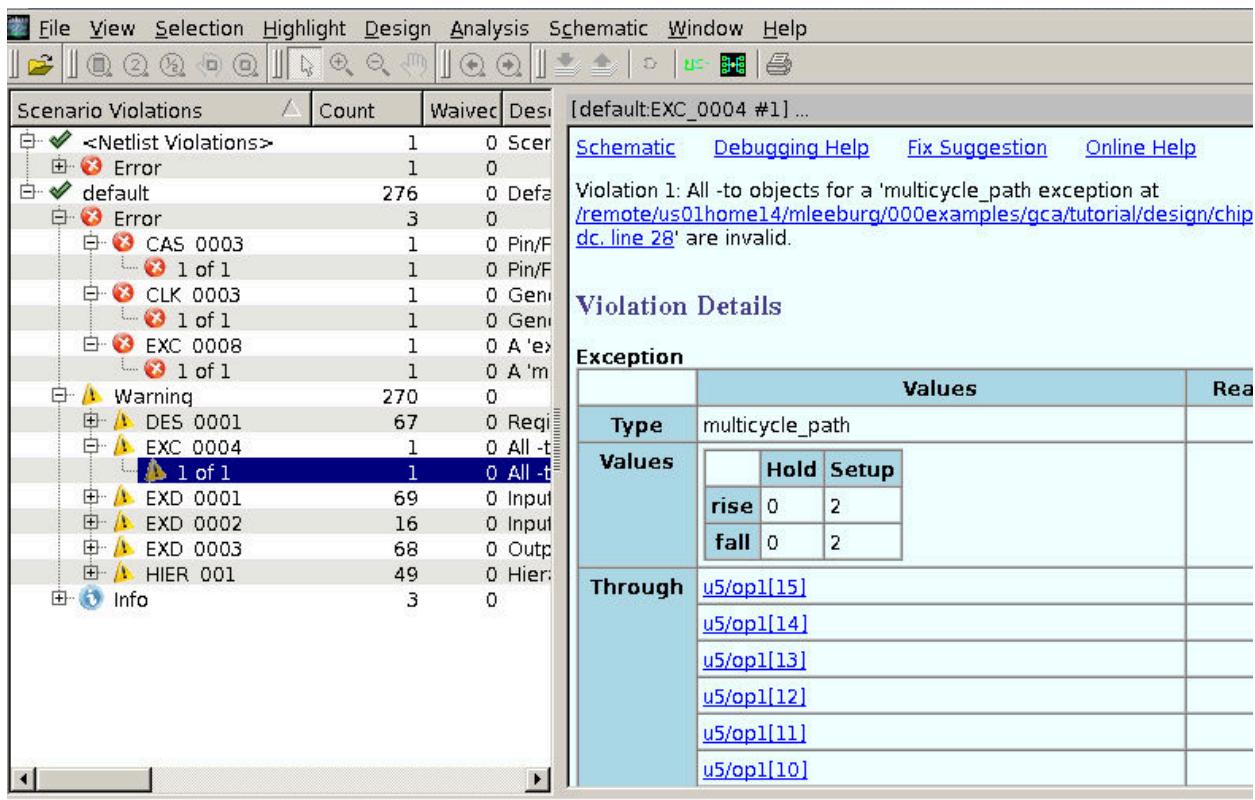
The `set_disable_timing` constraint disables all timing paths through the pin `clk_adder/A` and it is the root cause for the clock not reaching the master source `clk_adder_div2/CP` of the generated clock.

Based on your knowledge of the design, there can be a valid reason for disabling the clock network. If so, the generated clock definition should be reexamined. If left without a clock at the master source, the generated clock does not propagate.

Investigating the EXC_0004 Violation

To examine the EXC_0004 rule violation, expand the Warning category in the violation browser and select the EXC_0004 rule violation. The information pane displays the violation details, as shown in Figure 291.

Figure 291 Information Pane for EXC_0004 Rule Violation



The EXC_0004 rule flags exceptions with invalid path endpoints in the `-to` option of the constraint. The Violation Details show that the Type of exception is a multicycle path for this violation. If you browse further down the Violation Details, you can see that many Through and To points have been listed as part of this violation. The Reason listed for the To points is “nonendpoint”.

To help understand the Reason provided, scroll back to the top of the information pane, and click the Debugging Help link.

The Debugging Help suggests using the `report_disable_timing -all_arcs` command if the Reason code is case disabled, check disabled, sequential disabled, or node disabled. Because the Debugging Help does not have a command suggestion for the nonendpoint Reason, click the online Help link at the top of the information pane for more help. The rule reference for the EXC_0004 rule is displayed in your Web browser.

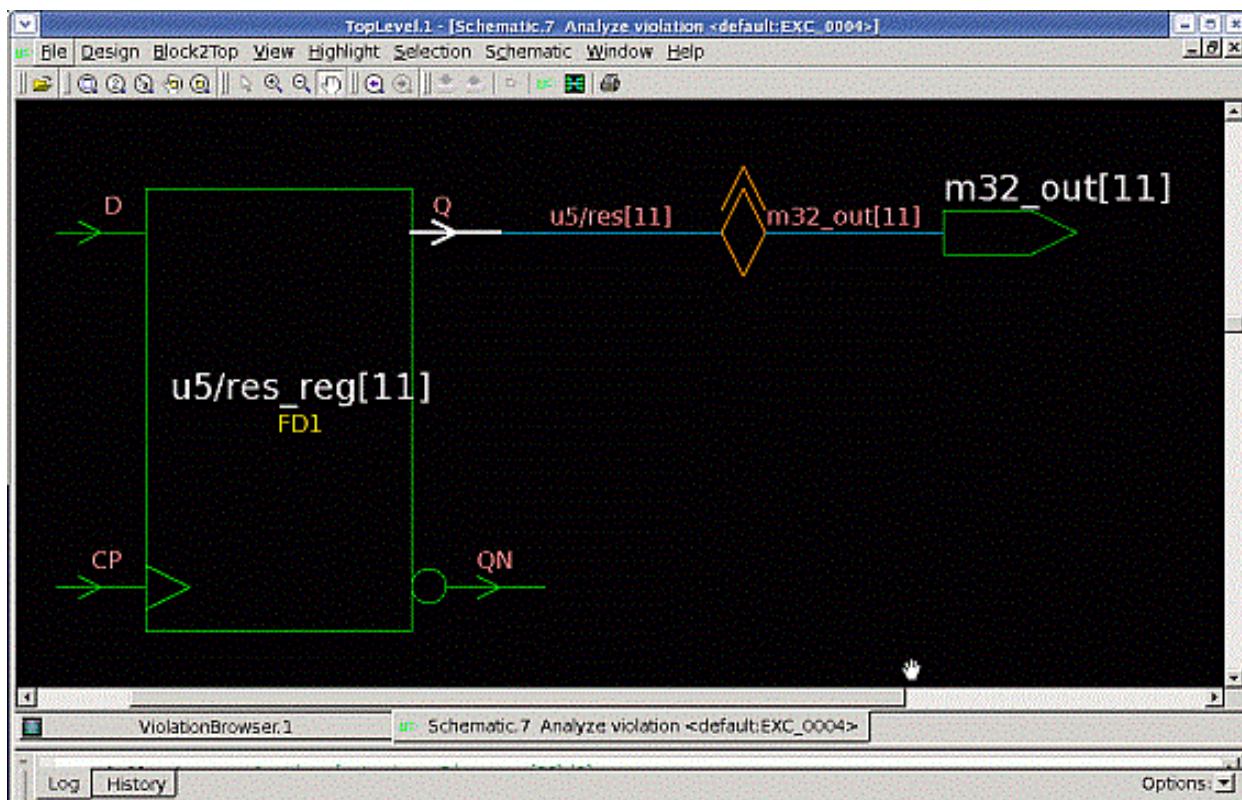
The Description section of the rule reference explains the different Reason codes for the EXC_0004 rule. The nonendpoint reason is given when the `-to` pin is an internal pin instead of a timing path endpoint. The Description section also states that the exception is ignored during timing analysis unless a valid endpoint is provided for the constraint. The

explanation in the rule reference for nonendpoint indicates that the pins specified for the `-to` option of the `set_multicycle_path` command are internal pins and are not timing path endpoints.

The `-to` points listed in the Violation Details are all pins with the name Q. View the schematic related to the violation by clicking the Schematic link at the top of the information pane. The schematic shows all the affected `-to` pins. Use the Zoom In button on the Toolbar to view individual pins.

The Q pin of the register is highlighted in white, indicating it is one of the `-to` points related to the violation. Double-click the pin to show more of the schematic, as shown in [Figure 292](#).

Figure 292 Expanded Schematic View of Register Q Pin



The schematic shows that the Q pin of the register is connected to an output port. The Q pin of the register might have been incorrectly specified instead of the D pin in the constraint. Changing the constraint to the D pin of the register might resolve the violation, but you should verify that this is the intent of the constraint based on your knowledge of the design.

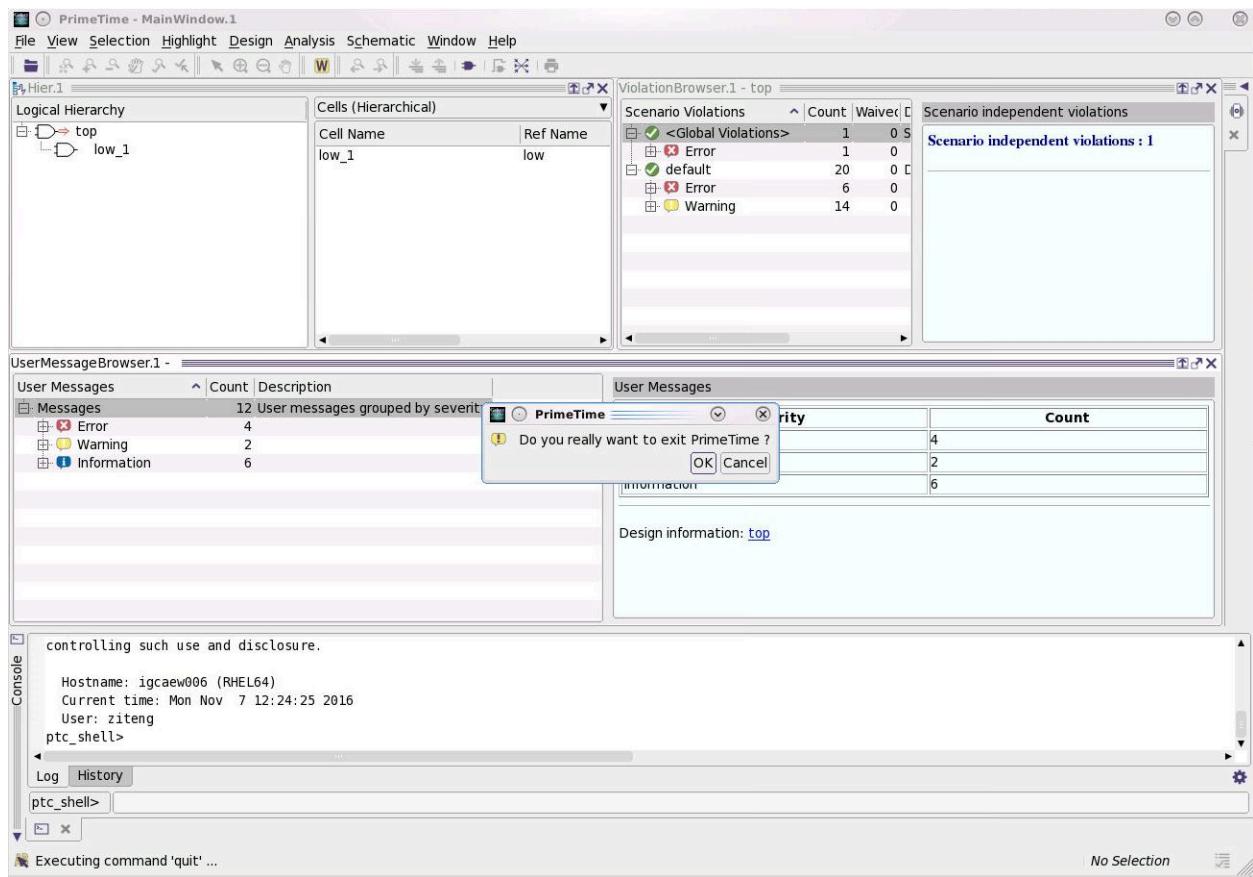
Ending and Restarting the Tutorial Session

You have completed investigation of several constraint problems found by constraint consistency. To quit the shell, enter `exit` at the `ptc_shell>` prompt in the GUI:

```
ptc_shell> exit
```

A message asks if you want to exit, as shown in [Figure 293](#).

Figure 293 GUI Message for Exiting Consistency Constraint Checking



Click OK to exit. The session ends and reports its memory and CPU time usage:

```
ptc_shell> exit
Maximum memory usage for this session: 24.38 MB
CPU usage for this session: 2 seconds
```

If you have ended the tutorial session and want to begin analysis of violations again, you rerun the ChipLevel design. To specify your run script at the command line, use the `-f` option:

```
% ptc_shell -f runTutorial.tcl
```

In this case, constraint consistency loads the libraries, design, and constraints and performs rule checking on the design. The GUI then displays the results of the `analyze_design` command in the violation browser.

If you want to rerun the analysis to investigate one rule, specify the `-rules` option for the `analyze_design` command. This limits the scope of the checking to the rules specified. The command below checks the ChipLevel design for violations of the CAS_0003 rule.

```
analyze_design -rules { CAS_0003 }
```

19

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](#)
- [Quality of Results Report](#)
- [Constraint Reports](#)
- [Bottleneck Report](#)
- [Global Slack Report](#)
- [Analysis Coverage Report](#)
- [Clock Network Timing Report](#)
- [Clock-Gating and Recovery/Removal Checks](#)
- [Timing Update Efficiency](#)
- [Path-Based Timing Analysis](#)

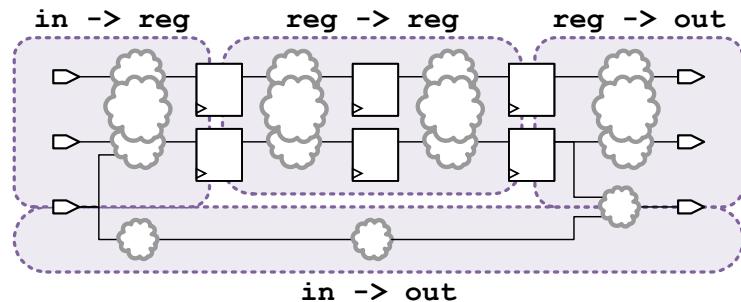
Global Timing Summary Report

The `report_global_timing` command provides a way to report the global timing closure state of the design, including information such as worst negative slack (WNS) and total negative slack (TNS) across all endpoints in the design.

The `report_global_timing` command works with other PrimeTime features, such as path-based analysis (PBA), distributed multi-scenario analysis (DMSA), and simultaneous multivoltage analysis (SMVA/DVFS).

Because the `report_global_timing` command focuses on global timing closure status, it has the following characteristics:

- It considers only violating timing paths; it does not gather or analyze passing paths.
- It organizes timing closure information into four topological path categories:



Internally, the `report_global_timing` command operates in two stages:

- *Path gathering*—obtaining violating timing paths from the design
- *Reporting*—producing formatted output from analysis of the gathered paths

It is important to understand that these two aspects—path gathering ("Which paths do I want to analyze?") and reporting ("What data do I want to see about those paths?")—are independently configured by command options.

Controlling `report_global_timing` Path Gathering

By default, the `report_global_timing` command obtains the *single worst path to each endpoint*, regardless of the path's topology type or path group.

You can modify this default path gathering behavior using the command options shown in [Table 51](#).

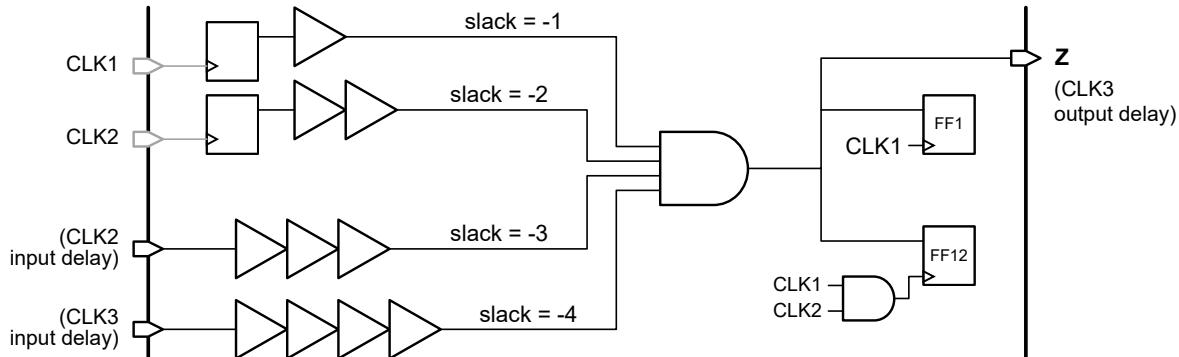
Table 51 Controlling `report_global_timing` Path Gathering

To do this	Use this option
Limit path gathering to only setup (max-delay) or hold (min-delay) paths	<code>-delay_type min max</code>
At each endpoint, gather the worst path	
• For each path group in the specified list	<code>-group group_list</code>
At each endpoint, gather the worst path	
• For each topological path category	<code>-enable_multiple_categories_per_endpoint</code>

Table 51 Controlling report_global_timing Path Gathering (Continued)

To do this	Use this option
At each endpoint, gather the worst path	-include {per_clock_violations}
<ul style="list-style-type: none"> For each topological path category For each path group For each launch/capture clock combination 	
Obtain the paths using path-based analysis	-pba_mode none path exhaustive ml_exhaustive
Limit path gathering to specific DVFS scenarios	-dvfs_scenarios dvfs_scenarios

To better understand how the path gathering options work, consider the following design logic:



The path gathering behavior for this design is as follows (see the inline comments for details):

```

pt_shell> # the default behavior - the single worst path to each endpoint
pt_shell> report_global_timing -delay_type max -path end
...
Slack      Type      Group          From_clock      To_clock      Endpoint
-----+-----+-----+-----+-----+-----+-----+
-4.00    in2out    CLK3           CLK3            CLK3          Z
-4.00    in2reg    CLK1           CLK3            CLK1          FF1/D
-4.00    in2reg    CLK2           CLK3            CLK2          FF12/D
-----+-----+-----+-----+-----+-----+-----+

```



```

pt_shell> # get the worst path to each endpoint,
pt_shell> # for each path group in {CLK1 CLK2}
pt_shell> report_global_timing -delay_type max -path end \
           -group {CLK1 CLK2}
...
Slack      Type      Group          From_clock      To_clock      Endpoint
-----+-----+-----+-----+-----+-----+-----+
-4.00    in2reg    CLK1           CLK3            CLK1         FF1/D
-4.00    in2reg    CLK1           CLK3            CLK1         FF12/D
-4.00    in2reg    CLK2           CLK3            CLK2        FF12/D
-----+-----+-----+-----+-----+-----+-----+

```

```

pt_shell> # get the worst path to each endpoint,
pt_shell> # for each of the four topological path categories
pt_shell> report_global_timing -delay_type max -path end \
    -enable_multiple_categories_per_endpoint
...

```

Slack	Type	Group	From_clock	To_clock	Endpoint
-2.00	reg2out	CLK3	CLK2	CLK3	Z
-4.00	in2out	CLK3	CLK3	CLK3	Z
-2.00	reg2reg	CLK1	CLK2	CLK1	FF1/D
-4.00	in2reg	CLK1	CLK3	CLK1	FF1/D
-2.00	reg2reg	CLK1	CLK2	CLK1	FF12/D
-4.00	in2reg	CLK1	CLK3	CLK1	FF12/D

```

pt_shell> # get the worst path to each endpoint,
pt_shell> # for each of the four topological path types,
pt_shell> # for each path group and launch/capture clock combination
pt_shell> report_global_timing -delay_type max -path end \
    -include {per_clock_violations}
...

```

Slack	Type	Group	From_clock	To_clock	Endpoint
-2.00	reg2out	CLK3	CLK2	CLK3	Z
-1.00	reg2out	CLK3	CLK1	CLK3	Z
-3.00	in2out	CLK3	CLK2	CLK3	Z
-4.00	in2out	CLK3	CLK3	CLK3	Z
-2.00	reg2reg	CLK1	CLK2	CLK1	FF1/D
-1.00	reg2reg	CLK1	CLK1	CLK1	FF1/D
-3.00	in2reg	CLK1	CLK2	CLK1	FF1/D
-4.00	in2reg	CLK1	CLK3	CLK1	FF1/D
-2.00	reg2reg	CLK1	CLK2	CLK1	FF12/D
-1.00	reg2reg	CLK1	CLK1	CLK1	FF12/D
-3.00	in2reg	CLK1	CLK2	CLK1	FF12/D
-4.00	in2reg	CLK1	CLK3	CLK1	FF12/D
-2.00	reg2reg	CLK2	CLK2	CLK2	FF12/D
-1.00	reg2reg	CLK2	CLK1	CLK2	FF12/D
-3.00	in2reg	CLK2	CLK2	CLK2	FF12/D
-4.00	in2reg	CLK2	CLK3	CLK2	FF12/D

Controlling report_global_timing Report Output

By default, the `report_global_timing` command generates a summary table that shows: (1) WNS, TNS, and the number of violating endpoints, (2) in each of the four topological path categories, (3) for setup and hold paths.

For example,

```

pt_shell> report_global_timing -enable_multiple_categories_per_endpoint
...

```

Setup violations

Total	reg->reg	in->reg	reg->out	in->out
-------	----------	---------	----------	---------

WNS	-6.08	-6.08	-0.80	-4.09	0.00
TNS	-51.61	-41.38	-1.60	-8.63	0.00
NUM	13	7	2	4	0

Hold violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-2.90	-0.75	-2.90	-1.01	0.00
TNS	-13.43	-4.44	-5.97	-3.02	0.00
NUM	12	6	3	3	0

You can modify this default path gathering behavior using the command options shown in [Table 52](#).

Table 52 Controlling report_global_timing Reporting

To do this	Use this option
Control whether to generate summary tables or per-endpoint tables	<code>-path_type summary end</code> (default is <code>summary</code>)
Create separate tables for nonstandard ¹ path groups	<code>-separate_non_standard_groups</code>
Create separate tables for all path groups (standard and nonstandard ¹)	<code>-separate_all_groups</code>
Create separate tables for launch/capture clock combinations	<code>-include {per_clock_violations inter_clock}</code> (The <code>per_clock_violations</code> option is also needed to gather paths for each launch/capture clock pair.)
Explicitly include empty path categories (with zero paths) instead of suppressing them	<code>-include {non_violated}</code>
Show scenario-specific details in a DMSA session	<code>-include {scenario_details}</code>
Choose between two summary table formats:	<code>-format narrow wide</code>
<ul style="list-style-type: none"> • Narrow, with group data values in separate rows • Wide, with group data values in adjacent columns 	(default is narrow)
Write the data to an output file, in comma-separated value (CSV) format	<code>-format csv</code> <code>-output filename</code>

1. The nonstandard path groups are `**default**`, `**async_default**`, and `**clock_gating_default**`.

Table 52 Controlling report_global_timing Reporting (Continued)

To do this	Use this option
Specify the significant digit count for reporting	<code>-significant_digits digits</code>

By default, the `report_global_timing` command generates summary tables. To see the per-endpoint contributions to those summary values, use the `-path_type end` option. For example,

```
pt_shell> report_global_timing -delay_type max -group {CLK3}
...
Setup violations
-----
      Total   reg->reg   in->reg   reg->out   in->out
-----
WNS     -6.08       -6.08       0.00      -4.09       0.00
TNS     -26.87      -18.24       0.00      -8.63       0.00
NUM        7            3           0           4           0
-----

pt_shell> report_global_timing -delay_type max -group {CLK3} -path_type end
...
Slack      Type      Group          From_clock      To_clock      Endpoint
-----
-6.08    reg2reg    CLK3          CLK2          CLK3          FF13/D
-6.08    reg2reg    CLK3          CLK2          CLK3          FF3/D
-6.08    reg2reg    CLK3          CLK2          CLK3          FF23/D
-4.09    reg2out    CLK3          CLK2          CLK3          W
-1.51    reg2out    CLK3          CLK1          CLK3          X
-1.51    reg2out    CLK3          CLK2          CLK3          Y
-1.51    reg2out    CLK3          CLK1          CLK3          Z
-----
```

To see details on interclock timing paths, use the `-include` option to specify both the `per_clock_violations` and `inter_clock` keyword values. (This ensures that paths for all launch/capture clock combinations are gathered and reported, respectively.) Intermediate-level summary tables with wildcarded clocks are also shown. For example,

```
pt_shell> report_global_timing -delay_type max \
           -include {per_clockViolations inter_clock}
...
Setup violations
-----
      Total   reg->reg   in->reg   reg->out   in->out
-----
C2C * -> *
WNS     -6.08       -6.08      -5.44      -4.09      -3.96
TNS     -269.83     -142.31     -103.30     -16.88     -7.33
NUM        59          27          22           8           2
-----

C2C * -> CLK1
WNS     -6.08       -6.08      -5.44       0.00       0.00
TNS     -93.14      -52.49     -40.66       0.00       0.00
NUM        19          10          9           0           0
-----
```

Chapter 19: Reporting and Debugging Analysis Results

Global Timing Summary Report

```
C2C CLK1 -> CLK1
WNS      -5.85      -5.85      -0.80      0.00      0.00
TNS     -22.86     -22.06      -0.80      0.00      0.00
NUM          5           4           1           0           0
-----
C2C CLK2 -> CLK1
WNS      -6.08      -6.08      -4.87      0.00      0.00
TNS     -41.92     -23.13     -18.79      0.00      0.00
NUM          8           4           4           0           0
-----
C2C CLK3 -> CLK1
WNS      -5.44      -3.65      -5.44      0.00      0.00
TNS     -28.37     -7.30     -21.07      0.00      0.00
NUM          6           2           4           0           0
-----
C2C * -> CLK2
WNS      -6.08      -6.08      -5.44      0.00      0.00
TNS     -78.46     -46.73     -31.72      0.00      0.00
NUM         16           9           7           0           0
-----
C2C CLK1 -> CLK2
WNS      -5.85      -5.85      0.00      0.00      0.00
TNS     -17.55     -17.55      0.00      0.00      0.00
NUM          3           3           0           0           0
-----
...
-----
C2C CLK3 -> CLK3
WNS      -5.44      -3.65      -5.44      -1.51      -3.96
TNS     -30.60     -7.30     -16.31      -3.03      -3.96
NUM          8           2           3           2           1
-----
```

If you have a long report with many summary tables, you can use the `-format wide` option to show the WNS, TNS and path count values in columns instead of rows. (This option does not affect `-path_type end` per-endpoint tables.) For example,

```
pt_shell> report_global_timing -delay_type max -group {CLK3}
...
Setup violations
-----
      Total   reg->reg   in->reg   reg->out   in->out
-----
WNS      -6.08      -6.08      0.00      -4.09      0.00
TNS     -26.87     -18.24      0.00      -8.63      0.00
NUM          7           3           0           4           0
-----
pt_shell> report_global_timing -delay_type max -group {CLK3} -format wide
...
Setup violations
      Total    |    reg->reg    |    in->reg    |    reg->out    |    in->out
      |    WNS    TNS    NUM    |    WNS    TNS    NUM    |    WNS    TNS    NUM    |    WNS    TNS    NUM
      |
-----  
-6.08 -26.87 7 | -6.08 -18.24 3 | 0.00 0.00 0 | -4.09 -8.63 4 | 0.00 0.00 0  
|
```

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

To specify the number of digits after the decimal point displayed for time values in the report, use the `report_timing` command with the `-significant_digits` option. This option controls only the number of digits displayed, not the precision used internally for analysis.

The following example shows a typical timing report:

```
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	

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
```

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 or C1, and ends at D1.

To report only the paths that have a specific type of startpoint and endpoint:

```
report_timing
-start_end_type reg_to_reg ; Only paths from register to register
-start_end_type reg_to_out ; Only paths from register to output port
-start_end_type in_to_reg ; Only paths from input port to register
-start_end_type in_to_out ; Only paths from input port to output port
```

For example, the command `report_timing -start_end_type in_to_out` reports only paths that start at an input (or bidirectional) port and end at an output (or bidirectional) port.

In this syntax, “`reg`” means any valid startpoint or endpoint that is not an input port, output port, or bidirectional port, even if not a sequential register. For example, a clock-gating check endpoint qualifies as “`reg`” in the `in_to_reg` start-end type.

The four start-end type choices are the same as the categories reported by the `report_global_timing` command. All timing paths fall into one of the four categories.

This report example shows the transition time and capacitance:

```
pt_shell> report_timing -transition_time -capacitance
...
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	9.15 r
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 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 53](#) 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 53 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.

If the paths contain latches, then with advanced latch analysis, you can use the normalized slack to calculate the maximum frequency for a path unless it is a recovered path (see [Finding Recovered Paths](#)) or a path that contains latches identified as a loop breaker or a path breaker (see [Reporting Latch Loop Groups](#)).

Using the report_timing -exclude Option

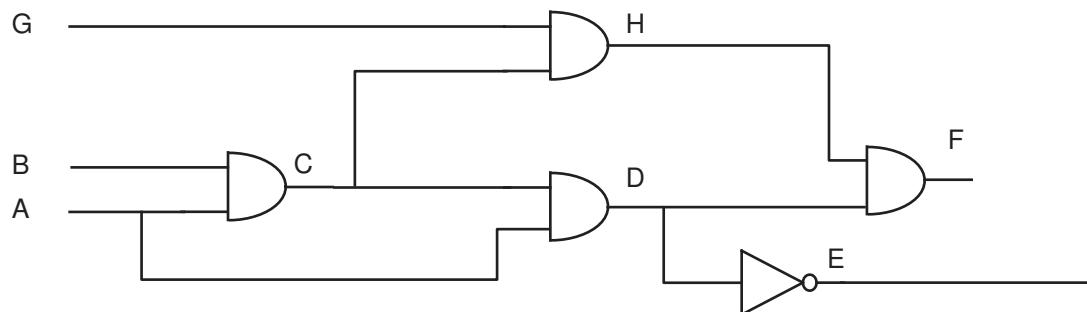
The `report_timing` command has the `-exclude`, `-rise_exclude`, and `-fall_exclude` options that allow you to skip paths that contain any excluded netlist objects. All of these options take a list of objects (pins, ports, nets, and cells). These options behave as follows:

- 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: Using the -exclude Option

For example, assume that you had a design with the paths shown in [Figure 294](#), where C is both an endpoint and a startpoint.

Figure 294 Using the -exclude option



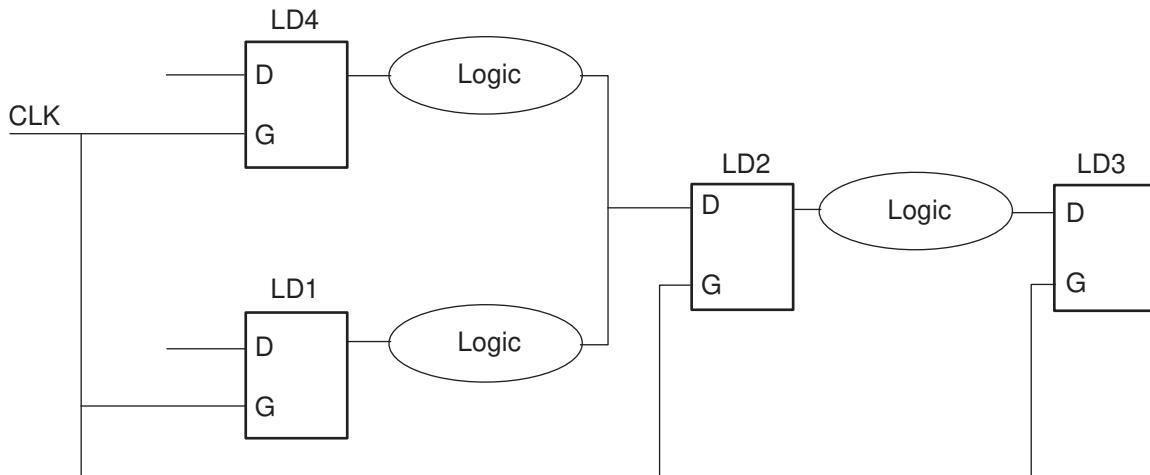
The following command reports three paths: A-->D-->E, A-->D-->F, and G-->H-->F.

```
pt_shell> report_timing -nworst 2 -max_paths 3 -exclude C
```

Example: 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 295](#). In this circuit, LD2 borrows from both LD1 and LD4, but the path from LD1 is the worst path.

Figure 295 Using -exclude with the -trace_latch_borrow option



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 or Cover Through 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 worker processes, but you cannot perform merged reporting at the manager process.

To restrict the report to paths that go through specified objects, use the `-cover_through` *through_list* option. This reports the single worst violating path through each of the objects in a list. For example,

```
pt_shell> report_timing -cover_through {n1 n2 n3}
```

This command reports the worst path through net n1, the worst path through n2, and the worst path through n3, resulting in a collection of up to three paths. Fewer paths are collected if there are no violating paths through the objects, or if the worst path through one object is the same as the worst path through another object in the list.

By default, only paths with negative slack are collected. If the `-slack_lesser_than` option is used, only paths with slack less than the specified value are collected.

Note that the following commands typically do not produce the same results:

```
pt_shell> report_timing -cover_through {n1 n2 3}
```

```
pt_shell> report_timing -through {n1 n2 n3} -nworst 1 -max_paths 3
```

For example, the three worst paths might pass through n1, ending at three different endpoints, without passing through n2 or n3. In that case, the first command would report three different paths through n1, n2, and n3, respectively, whereas the second command would report three different paths through n1 and none through n2 or n3.

Path Tagging

You can mark or “tag” a set of paths and then exclude those paths from consideration by the `get_timing_paths` and `report_timing` commands. This feature lets you work on smaller sets of paths, one set at a time, so that a large analysis can be more manageable and can be completed more efficiently.

For example, the following commands identify and tag the 100 worst-slack paths:

```
pt_shell> set_app_var enable_path_tagging true
true
pt_shell> set paths1 \
[get_timing_paths -nworst 100 -max_paths 100] # creates path collection
...
pt_shell> create_path_tag_set $paths1 -name tag1 # tags path collection
1
```

To exclude these paths from consideration in a subsequent `report_timing` or `get_timing_paths` command:

```
pt_shell> report_timing -exclude tag1
...
pt_shell> set paths2 \
```

```
[get_timing_paths -nworst 100 -max_paths 100 -exclude tag1]  
...
```

The new collection paths2 contains the next 100 worst paths aside from those already tagged.

Enabling Path Tagging

To use path tagging, you must first enable the feature as follows:

```
pt_shell> set_app_var enable_path_tagging true  
true
```

By default, the variable is set to `false` and path tagging is disabled.

Tagging a Path Collection

To tag a set of paths, use the `create_path_tag_set` command. The command takes two arguments, the path collection to be tagged and a name for the tag set. For example,

```
pt_shell> create_path_tag_set \  
[get_timing_paths -nworst 100 -max_paths 100] -name tag1  
1
```

If the path tag set name is already being used, the new set overwrites the older one.

Excluding Tagged Paths

To exclude consideration of tagged paths in the `report_timing` or `get_timing_paths` command, use the `-exclude` option. For example,

```
pt_shell> report_timing -exclude {tag1 tag2}  
...  
  
pt_shell> set paths4 \  
[get_timing_paths -nworst 100 -max_paths 100 -exclude {tag1 tag2}]  
...
```

Reporting and Removing Tag Sets

To report the number of paths in a tag set, use the `report_path_tag_set` command:

```
pt_shell> report_path_tag_set -name tag1  
...  
Tag name | Number of paths  
-----  
tag1 | 100
```

To return the number of paths in a path tag set as an integer, use the `get_path_count_for_tag` command:

```
pt_shell> get_path_count_for_tag -name tag1
100
```

To remove (destroy) a tag set, use the `remove_path_tag_set` command:

```
pt_shell> remove_path_tag_set -name tag1
Removing these path tag sets:
    tag1
1
```

If you do not use the `-name` option, the command operates on all tag sets.

Path-Based Analysis of Successively Worse Paths

You can use path tagging to successively analyze sets of worst paths in path-based analysis mode, without repeating the analysis of previously analyzed paths, as demonstrated in the following example.

```
# Enable path tagging
set_app_var enable_path_tagging true

# Run path-based analysis on the worst 1000 paths and tag the set
set path_set1 [get_timing_paths -slack_lesser_than 0 -max_paths 1000]
report_timing -pba_mode path $path_set1
create_path_tag_set $path_set1 -name done1
report_path_tag_set

# Run path-based analysis again, excluding already-analyzed paths
set path_set2 [get_timing_paths -slack_lesser_than 0 -max_paths 1000 \
    -exclude done1]
report_timing -pba_mode path $path_set2
create_path_tag_set $path_set2 -name done2

# Run path-based analysis again, excluding already-analyzed paths
set path_set3 [get_timing_paths -slack_lesser_than 0 -max_paths 1000 \
    -exclude {done1 done2}]
report_timing -pba_mode path $path_set3
create_path_tag_set $path_set3 -name done3
...
```

Exhaustive Path-Based Analysis

In exhaustive path-based analysis (using `-pba_mode exhaustive` in the `report_timing` or `get_timing_paths` command), the tool first performs graph-based analysis and discards the paths that already meet the timing constraint. These paths need not be analyzed again because graph-based analysis is conservative (pessimistic).

During path-based analysis, the tool might reach the limit on the number of paths analyzed per endpoint (if set the pba_exhaustive_endpoint_path_limit variable to an integer). In that case, you can use path tagging to reanalyze only the path endpoints for which the limit was reached, excluding the paths already discarded during graph-based analysis. Use a script similar to the following.

```
# Enable path tagging
set_app_var enable_path_tagging true

# Exhaustive path-based analysis; tag discarded graph-based paths
set path_set1 [get_timing_paths -slack_lesser_than 0 -max_paths 1000 \
    -pba_mode exhaustive -tag_paths_filtered_by_pba done1]
report_timing -pba_mode path $path_set1
report_path_tag_set

# Create collection of endpoints that reached limit (UIITE-480 warnings)
set exceeded_endpoints [get_attribute [current_design] \
    exhaustive_limit_exceeded_endpoints]

# Exhaustive path-based analysis of UIITE-480 endpoints, skip discarded
set path_set2 [get_timing_paths -slack_lesser_than 0 -max_paths 1000 \
    -to $exceeded_endpoints -pba_mode exhaustive -exclude done1 ]
report_timing -pba_mode path $path_set2
```

Custom Timing Reports With Attributes

The `report_timing` command offers an `-attributes` option that allows you to create custom timing reports that include your attributes of interest.

To do this, specify a list of one or more `point` attribute names with the option. For example,

```
pt_shell> report_timing \
    -nosplit \
    -pba_mode exhaustive \
    -significant_digits 3 \
    -input_pins \
    -attributes {transition object.max_transition}
```

would add `transition` and `object.max_transition` columns to the report:

```
Startpoint: mrxdrv_pad_i
    (input port clocked by mrx_clk_pad_i)
Endpoint: RxEnSync_reg
    (rising edge-triggered flip-flop clocked by mrx_clk_pad_i)
Path Group: inputs
Path Type: max

Point           Incr     Path   transition  object.max_transition
-----
clock mrx_clk_pad_i (rise edge) 0.000  0.000
clock network delay (propagated) 0.000  0.000
```

input external delay	4.000	4.000	f	
mrxdv_pad_i (in)	0.116	&	4.116	f
U3/A (INVX2)	0.000	&	4.117	f
U3/Y (INVX2)	0.181	&	4.298	r
U30/S0 (MX2X1)	0.000	&	4.298	r
U30/Y (MX2X1)	0.307	&	4.605	f
RxEnSync_reg/D (DFFRX1)	0.000	&	4.605	f
data arrival time			4.605	
clock mrx_clk_pad_i (rise edge)	8.000		8.000	
clock network delay (propagated)	1.037		9.037	
clock reconvergence pessimism	0.000		9.037	
RxEnSync_reg/CK (DFFRX1)			9.037	r
library setup time	0.350		8.687	
data required time			8.687	

data required time			8.687	
data arrival time			-4.605	

slack (MET)			4.082	

The attributes must be of the `timing_point` objects along the timing path. To obtain the underlying port, pin, or net objects, use chained attribute access (the “dot” notation) to access the `object` attribute of each `point` that returns the pin or port object. Undefined attribute values in the report are left blank.

Timing report types that print timing reports are supported, including full-clock reports, path-based analysis reports. Merged DMSA reporting is supported for attributes that are available at the master. Use the `-nosplit` option to prevent lines from splitting.

The `timing_report_fixed_width_columns_on_left` variable can be set to `true` to move the point name to the right of the attribute values.

Note the following limitation:

- When the `-nets` option of the `report_timing` command is used, attributes are not reported for the net objects. Only timing point attributes for timing point rows are supported.

Quality of Results Report

To report the quality of results (QoR), use the `report_qor` command. This report provides an overview of the timing of your design with the following information:

- 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 (TNS)
- Minimum delay and hold
- Detailed design rule constraints summary

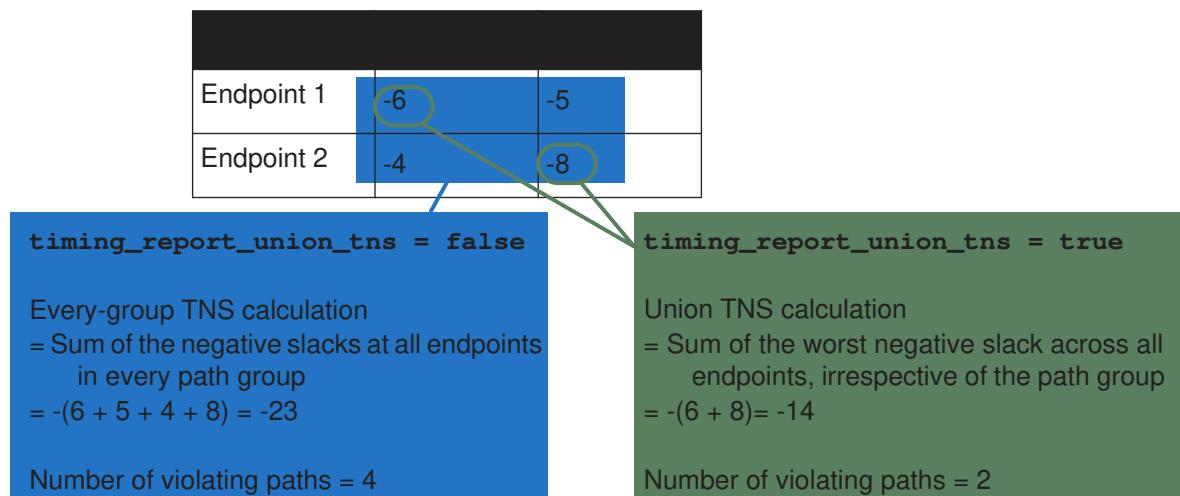
The tool offers two methods of total negative slack (TNS) calculation, the “union” mode and the “every-group” mode:

- Union TNS mode (default) – The tool adds up the worst slack for all endpoints with negative slack, taking the worst-slack values *across all path groups*, across all scenarios. Each endpoint with negative slack contributes only *once* to the TNS, even if that endpoint has negative slack in multiple path groups (for example, when two clocks reach the same endpoint).
- Every-group TNS mode – The tool adds up the worst slack for all endpoints with negative slack *per path group*, across all scenarios. If a given endpoint exists in two path groups and the endpoint has negative slack in both groups, it contributes *twice* to the TNS.

In either mode, for distributed multi-scenario analysis (DMSA), the worst slack for each endpoint *across all scenarios* contributes to the TNS.

You select the mode by setting the `timing_report_union_tns` variable to `true` for the union mode (the default) or `false` for the every-group mode.

The following example shows the negative slack (setup violations) at the endpoints in two path groups and compares the union and conventional methods of calculating the TNS for analysis in a single scenario:



The following example shows the same type of TNS calculation for distributed multi-scenario analysis (DMSA):

	Group1	Group2	Group1	Group2
Endpoint 1	-3	-5	-9	-1
Endpoint 2	-4	-8		-6
<code>timing_report_union_tns = false</code>			<code>timing_report_union_tns = true</code>	
Every-group TNS calculation = Sum of the negative slacks at all endpoints among scenarios in every path group = $-(5 + 9 + 4 + 8) = -26$			Union TNS calculation = Sum of the worst negative slacks at all endpoints, irrespective of the scenario or the path group = $-(9 + 8) = -17$	
Number of violating paths = 4			Number of violating paths = 2	

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 Constraint Report

A default constraint 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
...
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.

```
pt_shell> report_constraint -all_violators
...
    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.

```
pt_shell> report_constraint -all_violators -verbose...
...
    Startpoint: ff3 (rising edge-triggered flip-flop clocked by C1)
    Endpoint: OUT1 (output port clocked by C1)
    Path Group: C1
    Path Type: max

    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     -0.50
    -----
    data required time        -0.50
    data arrival time         -1.00
    -----
    slack (VIOLATED)        -1.50
```

```

Startpoint: ff4 (rising edge-triggered flip-flop clocked by C2)
Endpoint: OUT2 (output port clocked by C1)
Path Group: C1
Path Type: max
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 arrival time         -1.00
-----
slack (VIOLATED)          -1.50

```

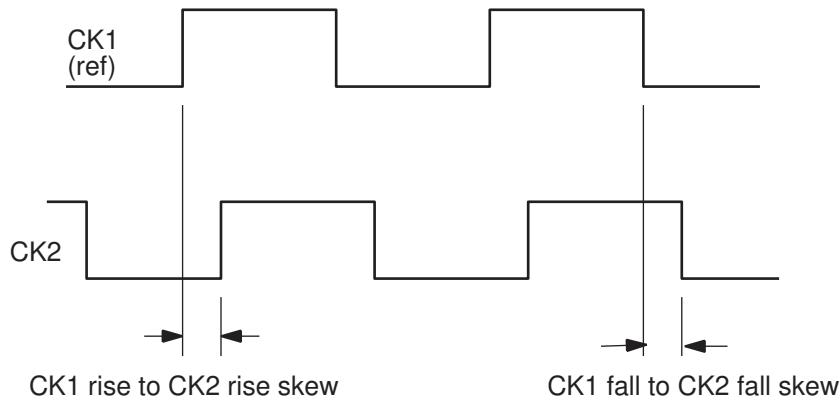
Maximum Skew Checks

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 Liberty (.lib) format. 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 296](#) describes a timing diagram for skew constraint.

Figure 296 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:

```
pt_shell> report_constraint -max_skew -all_violators
          Required           Actual
          Pin      Skew       Skew     Slack
-----+-----+-----+-----+-----+
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 0.05 r
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:

- In the PrimeTime GUI, generate a bottleneck histogram (Reports > Histograms > Timing Bottlenecks in the PrimeTime GUI).
- In the shell, use the `report_bottleneck` command, which lists the leaf cells with the highest bottleneck cost..

Note:

If you use the `report_bottleneck` command in your flow, set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update.

- Write a Tcl program that retrieves, organizes, and reports bottleneck attributes. The Tcl procedure in [Example 88](#) is provided in `install_dir/auxx/pt/examples/tcl/bottleneck_utils.tcl`.

To verify the result of bottleneck analysis, use the `report_timing` or `get_timing_paths` command.

[Example 87](#) 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 87 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 88 *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:

Set the `timing_save_pin_arrival_and_slack` variable to `true` before you perform the first timing update. Otherwise, the `report_global_slack` command sets the variable to `true` and incurs the runtime of another timing update.

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

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, 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 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.)

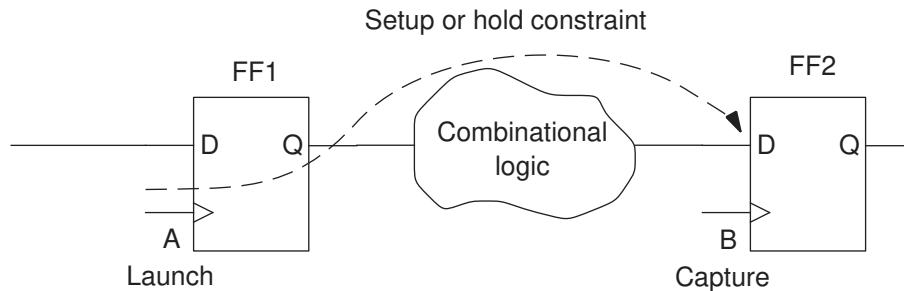
When a pin named in the “to” list is not a clock pin of a sequential element, the default behavior is to replace that pin with the set of clock pins in the transitive fanout of the named pin. (Here, “clock pin” means the clock pin of a flip-flop or the gate pin of a latch.) But if you specify the `-probe` option, then the latency directly at the specified pin is reported, with no transitive fanout tracing.

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 297](#). 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 297 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.

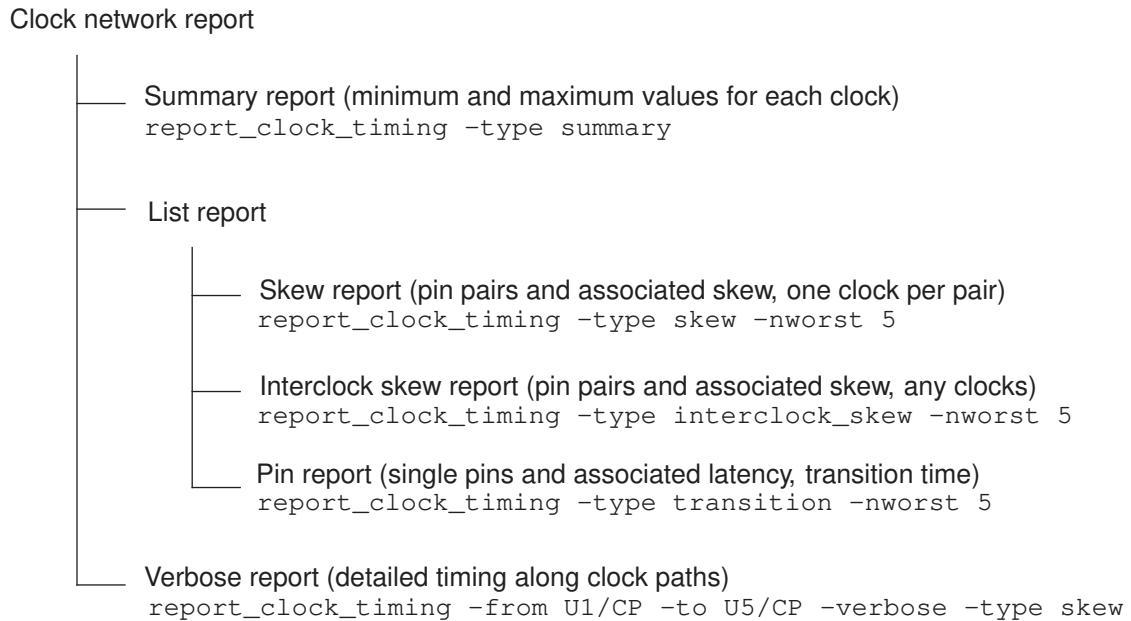
To display the names of the “from” clock and the “to” clock in the interclock skew report, use the `-show_clocks` option of the `report_clock_timing` command.

Clock Timing Reporting Options

The `report_clock_timing` command offers several different types of reports, as summarized in [Figure 298](#). 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 298 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-
  
```

Chapter 19: Reporting and Debugging Analysis Results
 Clock Network Timing Report

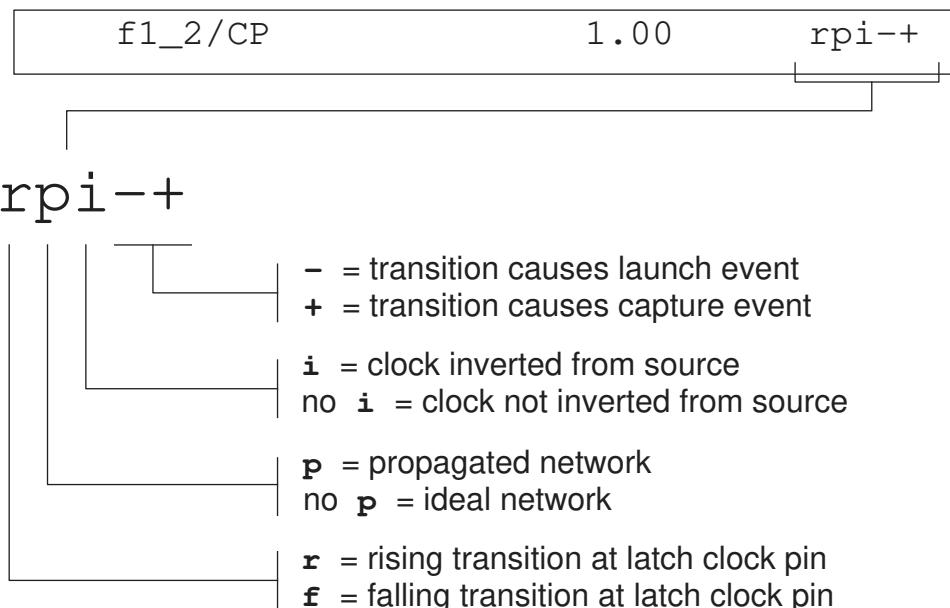
Maximum setup skew:		
f2_2/CP		rp+-
f2_1/CP	4.00	rp+-
Maximum hold skew:		
f3_3/CP		rpi+-
f2_2/CP	3.01	rp+-

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 299](#).

Figure 299 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+-
	12_2/G	4.11		rp-
	f1_2/CP	1.00	-0.10	rpi+-
	f2_2/CP	6.11		rp+-
	13_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 299](#).

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
f2_1/CP           2.01      0.11     4.21      rp-+
...

```

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: CLK1
      --- Latency ---
Clock Pin  Source  Network  Total  Trans
-----
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 299](#).

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)  

...
```

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.

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.

You can perform path-based analysis by using the following commands with the `-pba_mode` option:

- `get_timing_paths`
- `report_constraint`
- `report_global_timing`

- report_qor
 - report_timing
-

Path-Based Analysis Modes

To choose the path-based analysis mode, specify the `-pba_mode` option with one of the following arguments:

- `path` – Does not perform a path search 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.
- `exhaustive` – 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.
- `ml_exhaustive` – Performs an exhaustive recalculated path search, accelerated with machine learning. This mode is faster than the ordinary exhaustive mode, yet is safe for signoff analysis. It trades off runtime for QoR based on real-time prediction of the design timing. It uses more aggressive prediction for better speed in earlier phases of the analysis, and more conservative analysis in later phases for maximum accuracy. See [Exhaustive Path-Based Analysis With Machine Learning](#) for details.

To ensure optimal path search runtime, the following usage is recommended:

- Use the `-pba_mode exhaustive` option with the `-slack_lesser_than` option (if applicable) to keep the search bounded.
- Perform an exhaustive path search only when `-pba_mode path` shows that the slacks are reasonably close to the required threshold.
- Ensure that the reporting is as specific as possible. If reports are needed 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 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 mypaths [get_timing_paths -max_paths 10]
pt_shell> set recalc_paths [get_timing_paths -pba_mode path $mypaths]
```

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. You can limit the number of endpoints considered for exhaustive path searching by setting the `pba_exhaustive_endpoint_path_limit` variable.

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.

Exhaustive Path-Based Analysis With Machine Learning

Exhaustive path-based analysis (PBA) searches all paths in a given scope to search for paths that violate after recalculation.

Machine learning PBA (ML-PBA) performs exhaustive PBA using machine learning techniques to trade off accuracy for runtime. It is faster than the regular exhaustive PBA mode, yet it is safe for signoff analysis and thus can be used throughout the flow. It can be used instead of switching between the `path` and `exhaustive` values of the `-pba_mode` option.

To use ML-PBA, use the `ml_exhaustive` value for any analysis command that has a `-pba_mode` option:

```
pt_shell> report_timing -pba_mode ml_exhaustive
```

The `pba_exhaustive_endpoint_path_limit` variable must be set to `infinity` (the default).

The ML-PBA mode trades off accuracy for runtime when the design contains many violations, gradually converging to the regular exhaustive PBA mode results when the design is free of violations. The ML-PBA results are always a subset of the regular exhaustive PBA results, where the path set is the scope of paths being searched (to an endpoint, in a path group, across the design, and so on).

ML-PBA is safe for signoff—if ML-PBA reports no violations, then the design has no violating recalculated paths.

The ML-PBA behavior can be summarized as follows:

- If regular exhaustive PBA reports no violations, then ML-PBA will also report no violations.
- If regular exhaustive PBA reports a single violation, then ML-PBA will also report a single violation, although possibly not the same violation or not the worst path.
- If regular exhaustive PBA reports violating paths to multiple endpoints, then ML-PBA will also report violating paths to one or more of these endpoints, although possibly not the same violations or not the worst paths.

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 can have a paths-per-endpoint limit that you set with the `pba_exhaustive_endpoint_path_limit` variable.

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.

Exhaustive Path-Based Analysis Settings

By default, when you use the `report_timing` command with the `-pba_mode exhaustive` option, the command performs an exhaustive path-based analysis (PBA) and considers all possible paths leading to an endpoint for finding the worst slack. You can control the operation of this analysis by setting the `pba_exhaustive_endpoint_path_limit` variable to the string `infinity` or an integer.

With the “infinity” setting (the default), the PBA algorithm breaks up the fanin cone of each endpoint into subgraphs and successively analyzes and modifies the subgraphs to find the worst path and slack. This approach is guaranteed to cover all the paths and always returns the worst path-based analysis slack.

Alternatively, you can set the variable to an integer, for example,

```
set_app_var pba_exhaustive_endpoint_path_limit 25000
```

If you set the variable to an integer, the command analyzes each individual path in isolation from other paths. When the number of recalculated paths reaches the specified limit, analysis stops and you get a UITE-480 warning message, “The exhaustive path-based recalculation limit of 25000 is reached ...”.

The “infinity” mode offers better runtime when there is a large gap between graph-based and path-based slacks or the circuit topologies are complex, such as reconvergent datapath logic. However, the integer limit mode can be faster when graph-based and path-based slacks are already similar and relatively few paths need to be considered to reach convergence.

HyperTrace Accelerated Path-Based Analysis

HyperTrace is a technology that computes refined graph-based timing data to accelerate exhaustive path-based analysis. The following topics provide more information:

- [Introduction to HyperTrace Technology](#)
- [Configuring HyperTrace Analysis](#)
- [Using HyperTrace Analysis](#)
- [Unsupported Flows for HyperTrace Analysis](#)

Using this feature requires a PrimeTime-ADV-PLUS license.

Introduction to HyperTrace Technology

Exhaustive path-based analysis (PBA) uses a search algorithm to find the worst recalculated paths in the design. The search obtains graph-based timing paths from the design, then recalculates them.

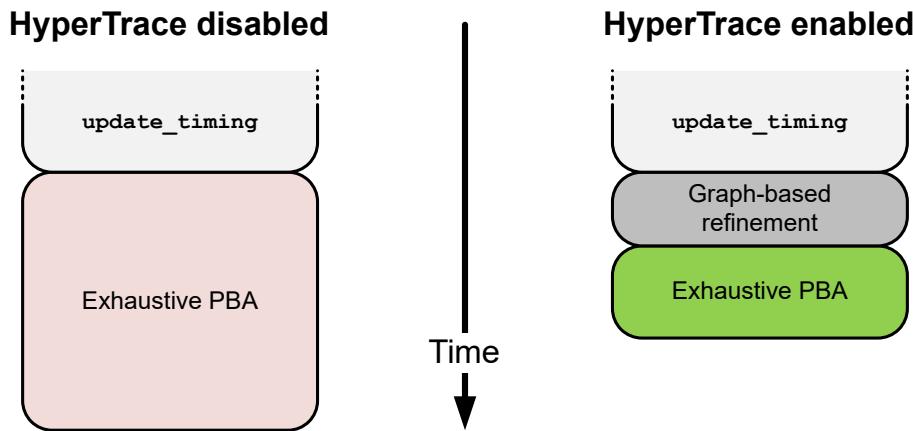
HyperTrace is a technology that accelerates the exhaustive PBA search by computing *refined* graph-based timing data, then using it to drive the exhaustive PBA search.

This graph-based refinement is applied to the *critical region* of the design, which is the set of slack-critical (violating) pins. Refinement performs selective signal merging to improve accuracy while retaining the memory and runtime benefits of a graph-based representation.

The tool automatically performs graph-based refinement before the first exhaustive PBA command. Refinement has two opposing effects on runtime:

1. *Runtime spent* to compute the initial graph-based refinement data
2. *Runtime saved* for all subsequent exhaustive PBA analysis

Figure 300 HyperTrace Exhaustive PBA Runtime Comparison



To achieve an overall runtime benefit, the runtime reduction in exhaustive PBA must be more than the runtime increase from refinement.

HyperTrace is intended for signoff use in cases where exhaustive PBA runtime is a significant part (over 20 percent) of the overall analysis runtime. It is not suitable for designs that have small exhaustive PBA runtimes or have a high percentage of the design in the critical region (as the refinement overhead might exceed the benefit).

HyperTrace graph-based refinement is recommended when:

- The design is near the signoff stage
- Exhaustive path-based analysis takes more than 50 percent of the flow runtime
- Fewer than 10 percent of the pins in the design have timing violations

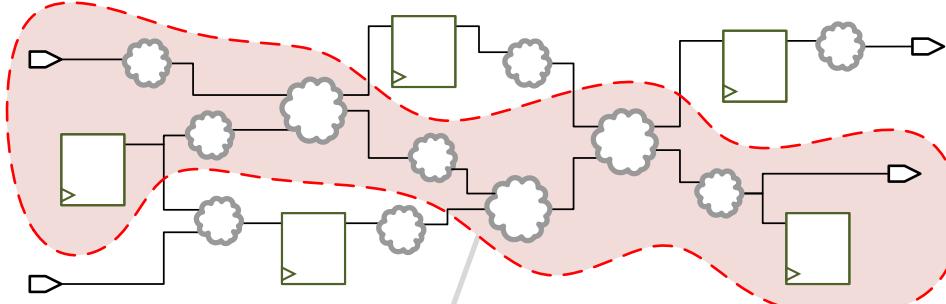
Note that the refined graph-based timing is used only for exhaustive PBA. The original timing graph is still used for graph-based (non-PBA) reporting.

The HyperTrace Critical Region

The *critical region* is the set of slack-critical (violating) pins in the design. HyperTrace works by creating a refined timing graph of the critical region, then using it to drive exhaustive PBA.

To be accelerated, the exhaustive PBA search must take place *entirely within the critical region*. This means that the critical region threshold should be set to the largest `-slack_lesser_than` value (explicit or default) used for the exhaustive PBA search:

Figure 301 The HyperTrace Critical Region (Max-Delay Analysis Shown)



```
max_slack < timing_refinement_max_slack_threshold
(default 0.0)
```

If you perform exhaustive PBA with an explicit `-slack_lesser_than` value that exceeds the critical region threshold, the tool issues a warning and reverts to conventional exhaustive PBA:

```
pt_shell> report_timing -pba_mode exhaustive -slack_lesser_than 1.0
Warning: Graph-based refinement will not be used for this report as
slack_lesser_than exceeds refinement slack threshold. (UITE-588)
```

Because refinement runtime is proportional to critical region size, the critical region should not be larger than needed. During the initial refinement, the tool reports the critical region size as a percentage of the entire design:

Information: **9.160 percent** of pins in the design are being included in the critical region for graph-based refinement. (UITE-579)

In most cases, a good limit for the critical region size is 10 percent of the design. If you exceed this guideline, the tool issues a warning:

Warning: The number of pins considered critical for graph-based refinement is large (**37.405 percent** of pins in the design). Graph-based refinement may incur significant runtime and memory impact. (UITE-589)

However, this just a guideline. Flows dominated by exhaustive PBA runtime might still benefit from enabling graph-based refinement, even if the critical region is larger than 10 percent.

Configuring HyperTrace Analysis

To configure HyperTrace analysis, add the following commands before the first exhaustive PBA command in your script:

1. Set the following variable to enable HyperTrace analysis:

```
# enable HyperTrace analysis
set_app_var timing_enable_graph_based_refinement true
```

2. Set the max-delay critical region threshold, as described in [Setting the Critical Region Slack Threshold Values](#).

Determine the lowest `-slack_lesser_than` value used for max-delay exhaustive PBA in your analysis. If the value is above zero, set the following threshold variable to that value. For example,

```
# set the critical region threshold
set_app_var timing_refinement_max_slack_threshold 0.100
```

If there are no explicit `-slack_lesser_than` options in your script, the default threshold of zero is suitable.

3. (Optional) Normally, HyperTrace analysis is disabled for min-delay paths. If you have complex violating min-delay logic cones (such as I/O or memory interfaces) and the exhaustive PBA runtime for min-delay analysis is significant, repeat [Step 2](#) for the min-delay threshold.

4. If your scripts perform exhaustive PBA within a `parallel_execute` command, set the following variable as described in [Controlling When Graph-Based Refinement Occurs](#):

```
set_app_var timing_update_include_graph_based_refinement always
```

5. (Optional) Configure your script to automatically enable HyperTrace only when the critical region falls within a particular size limit, as described in [Automatically Enabling HyperTrace by Critical Region Size](#):

```
set_app_var timing_refinement_maximum_critical_pin_percentage 20
```

The optimal value for this variable is design-dependent.

Setting the Critical Region Slack Threshold Values

HyperTrace provides separate variables to control the max-delay (setup) and min-delay (hold) critical region thresholds:

- `timing_refinement_max_slack_threshold` (**defaults to 0.0**)
- `timing_refinement_min_slack_threshold` (**defaults to disabled**)

Both variables can be set to:

- A floating-point value
- `disabled`, which disables graph-based refinement for that delay type

Min-delay refinement is disabled by default because min-delay exhaustive PBA is typically fast and thus not a good candidate for refinement.

For example, if `timing_refinement_max_slack_threshold` is set to `-0.10`, then all pins with a setup slack worse than `-0.10` are included in the graph-based refinement for maximum delay analysis. Similarly, if `timing_refinement_min_slack_threshold` is set to `0.05`, then all pins with a hold slack worse than `0.05` are included in the graph-based refinement for minimum delay analysis.

Normally, the critical region threshold should be set to the lowest `-slack_lesser_than` value used for exhaustive PBA analysis. If this threshold results in a large critical region—or more importantly, an overall runtime degradation—then the design needs more timing closure before HyperTrace can be used.

Automatically Enabling HyperTrace by Critical Region Size

To simplify scripting and flow setup, you can set the following variable to the largest critical region size you want to allow:

```
pt_shell> set_app_var \
           timing_refinement_maximum_critical_pin_percentage 20
```

This allows HyperTrace to automatically disable itself when the design is largely violating, then enable itself when the design reaches a suitable point in the flow.

If initial refinement determines that the size of the critical region would exceed this percentage, then the tool issues a message and disables HyperTrace acceleration:

Warning: Graph-based refinement is not being performed because the percentage of pins in critical region exceeds 20.000 percent.
 (UITE-630)

The default for this variable is 100%, which effectively deactivates this feature.

Controlling When Graph-Based Refinement Occurs

By default, HyperTrace defers computation of the graph refinement until the first exhaustive PBA command is run. You can control when refinement is computed by using the `timing_update_include_graph_based_refinement` variable.

[Table 54](#) shows the values for this variable.

Table 54 *timing_enable_graph_based_refinement* Effects on HyperTrace Updates

Variable value	Timing update, full (explicit or implicit)	Exhaustive PBA commands	Timing update, incremental (explicit or implicit)
never		Full HT update ²	
default		Full HT update ²	Incremental HT update
always	Full HT update		Incremental HT update ³

The default of the variable is `default`.

This variable can be used as follows:

- If you are using the `parallel_execute` command to perform exhaustive PBA reporting, set this variable to `always` to precompute the refinement during the initial timing update. (Refinement cannot be computed within a `parallel_execute` command.)

```
# if the timing is already updated, update_timing
# will update *only* the graph refinement data
set_app_var timing_enable_graph_based_refinement always
update_timing

parallel_execute {
    {report_timing -pba_mode exhaustive ...} file1.log
    {report_timing -pba_mode exhaustive ...} file2.log
    ...
}

# set the refinement update mode back to default to avoid
# unnecessary refinement in any subsequent timing updates
set_app_var timing_enable_graph_based_refinement default
```

As noted in the script comments, be sure to set the refinement timing update mode back to `default` to avoid unnecessary refinement in subsequent timing updates.

- If you are running a script that performs manual ECO operations (`size_cell`, `insert_buffer`, and so on) that are not intermixed with exhaustive PBA reporting,

2. No update occurs if the critical region threshold is not met.

3. A full HyperTrace update is performed if no previously computed refinement data exists.

consider temporarily setting the variable to `never` to defer refinement updates until the script is complete.

Note that when the variable is set to `never`, graph refinement updates are always full updates. After the ECO operations are complete, be sure to set the refinement timing update mode back to `default` to allow incremental refinement updates.

Using HyperTrace Analysis

After HyperTrace is enabled and configured, simply update the design timing and perform exhaustive PBA analysis as usual.

The following messages indicate that the graph-based refinement is being computed:

Information: Beginning graph-based refinement. (UITE-585)

Information: 8.459 percent of pins in the design are being included in the critical region for graph-based refinement. (UITE-579)

Information: Graph-based refinement complete. (UITE-586)

These messages are issued by `update_timing` or by the first exhaustive PBA reporting command, depending on how the `timing_update_include_graph_based_refinement` variable is set.

In addition, HyperTrace issues messages for certain conditions that affect or relate to its use, as described in [Table 55](#). If you see any of these messages, refer to the indicated topic.

Table 55 HyperTrace Information and Warning Messages

If you see this message	Refer here for more information
Warning: Graph-based refinement will not be used for this report as slack_lesser_than exceeds refinement slack threshold. (UITE-588)	Setting the Critical Region Slack Threshold Values
Warning: Setting slack_lesser_than to % for this timing report. (UITE-590)	Adjustment of the Default -slack_lesser_than Value
Warning: Graph-based refinement will not be performed as it is not supported with %. (UITE-587)	Unsupported Flows for HyperTrace Analysis
Information: Invalidating the graph-based refinement analysis. (PTE-137)	Invalidation of Graph-Based Refinement
Warning: The number of pins considered critical for graph-based refinement is large (%s). Graph-based refinement may incur significant runtime and memory impact. (UITE-589)	The HyperTrace Critical Region

Adjustment of the Default `-slack_lesser_than` Value

In a non-HyperTrace flow, when the `-slack_lesser_than` option is not specified, the default behavior is as follows:

- If `-max_paths > 1`, then `-slack_lesser_than` defaults to 0.0.
- If `-max_paths == 1`, then `-slack_lesser_than` defaults to INFINITY.

But when HyperTrace is enabled, the default behavior applies a third additional rule:

- If the default `-slack_lesser_than` value would exceed the critical range threshold, then the `-slack_lesser_than` default is lowered to the critical range threshold value.

This avoids the UITE-588 fallback described in [The HyperTrace Critical Region](#). When this adjustment occurs, the tool issues a warning:

```
Warning: Setting slack_lesser_than to 0 for this timing report.  
(UIITE-590)
```

For `-max_paths == 1`, this override always occurs.

For `-max_paths > 1`, this override occurs when the refinement threshold is less than 0.0.

Invalidation of Graph-Based Refinement

The graph-based refinement data persists after the first command that computes it. Subsequent exhaustive PBA commands that satisfy the usage criteria reuse the same graph-based refinement data without repeating the analysis.

The graph-based refinement data is invalidated by any of these events:

- A full timing update is performed.
- An unsupported design or netlist change command.
- Any of the following variables are changed:
 - `timing_enable_graph_based_refinement`
 - `timing_refinement_max_slack_threshold`
 - `timing_refinement_min_slack_threshold`
 - `timing_pocvm_report_sigma`
 - `pba_derate_only_mode`
 - `timing_report_use_worst_parallel_cell_arc`
- The design is removed or reset.

The following information message indicates that the graph-based refinement data has been invalidated:

Information: Invalidating the graph-based refinement analysis.
(PTE-137)

After this invalidation message is issued, any subsequent exhaustive PBA command that satisfies the criteria performs a new graph-based refinement analysis.

Incremental Graph-Based Refinement in HyperTrace

HyperTrace supports the following ECO change commands:

- `size_cell`
- `insert_buffer`
- `remove_buffer`

If these commands are used after a graph-based refinement analysis, subsequent incremental timing updates also update the graph-based refinement data. (This incremental update issues the same status messages as a full update.)

If an unsupported design change command is used, then the existing graph-based refinement data is invalidated (PTE-137) and a full (non-incremental) graph-based refinement is performed if required by a subsequent exhaustive PBA command.

Unsupported Flows for HyperTrace Analysis

The following features and conditions prevent HyperTrace graph-based refinement analysis from being performed:

- The design is not in on-chip variation mode.
- `timing_remove_clock_reconvergence_pessimism` is set to false.
- `timing_enable_cross_voltage_domain_analysis` is set to true.
- `timing_disable_internal_inout_cell_paths` is set to false.
- `timing_disable_internal_inout_net_arcs` is set to false.
- HyperScale analysis with AOCVM analysis is enabled.
- HyperScale distributed analysis is enabled.

In these cases, the tool issues a message indicating the unsupported condition feature or condition:

UITE-587 (Warning) Graph-based refinement will not be performed as it is not supported with %s.

After a UITE-587 warning, all subsequent reports still run, but without the benefit of HyperTrace acceleration.

See Also

- [HyperScale Distributed Analysis](#)

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 (C_{eff}) 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.

20

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](#)
- [Quick Start Guide](#)
- [GUI Windows](#)
- [Analyzing Timing Path Collections](#)
- [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](#)
- [Viewing and Modifying Schematics](#)
- [Inspecting Timing Path Elements](#)
- [Viewing Object Attributes](#)
- [Recalculating Timing Paths](#)
- [Layout View and ECOs in the GUI](#)
- [Interactive Multi-Scenario Analysis of Timing Paths](#)
- [GUI Tcl Commands](#)

Opening and Closing the GUI

The PrimeTime GUI is a window- and menu-driven interface that operates in your X windows environment on 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.

See Also

- [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 Linux 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 Linux 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](#)
-

Quick Start Guide

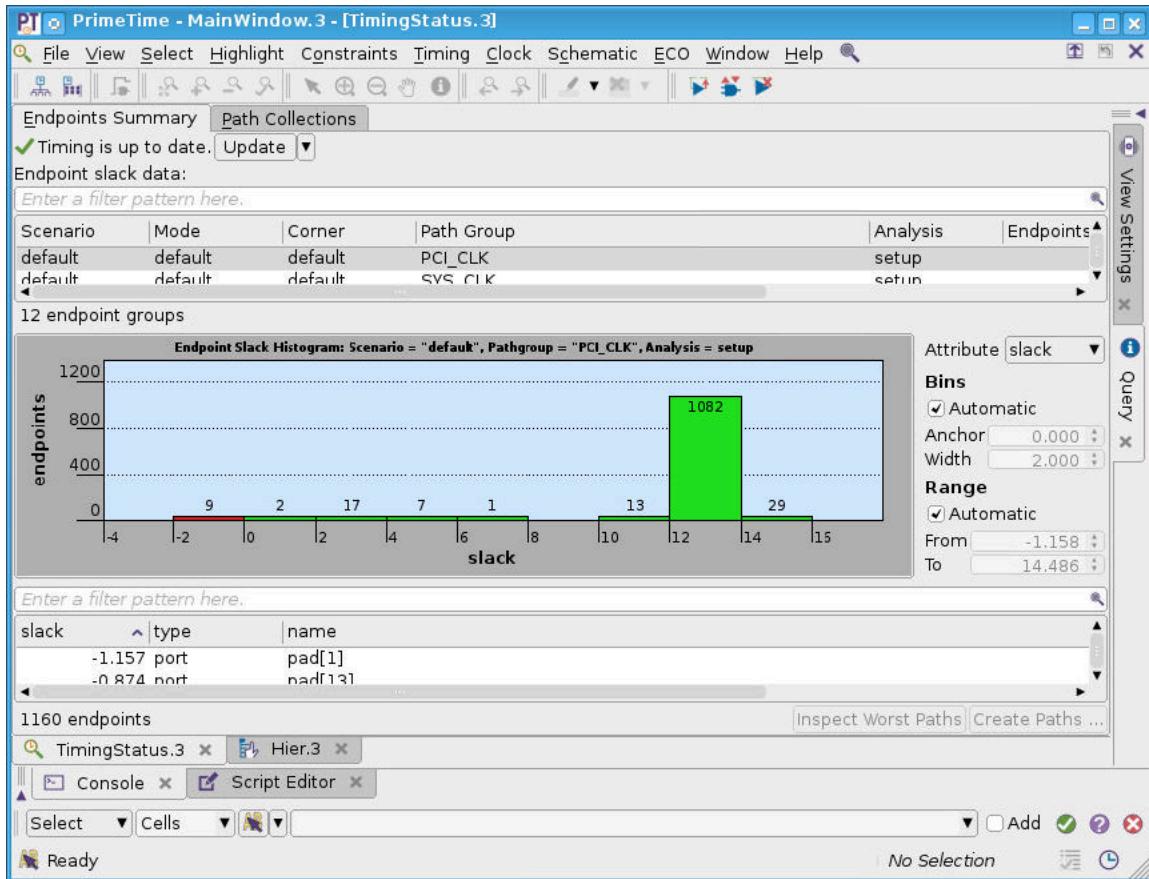
The following procedure introduces some of the features of the GUI. Start with a loaded, constrained, and analyzed design, after a timing update.

1. Create a collection of paths to analyze in the GUI. For example,

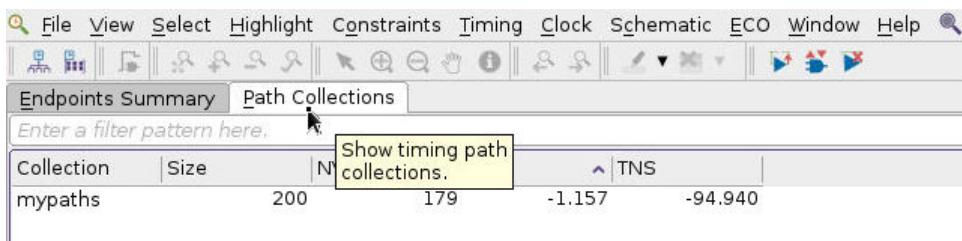
```
pt_shell> set mypaths [get_timing_paths -nworst 20 -max_paths 500 \
    -slack_lesser_than 0.2]
```

2. Open the GUI:

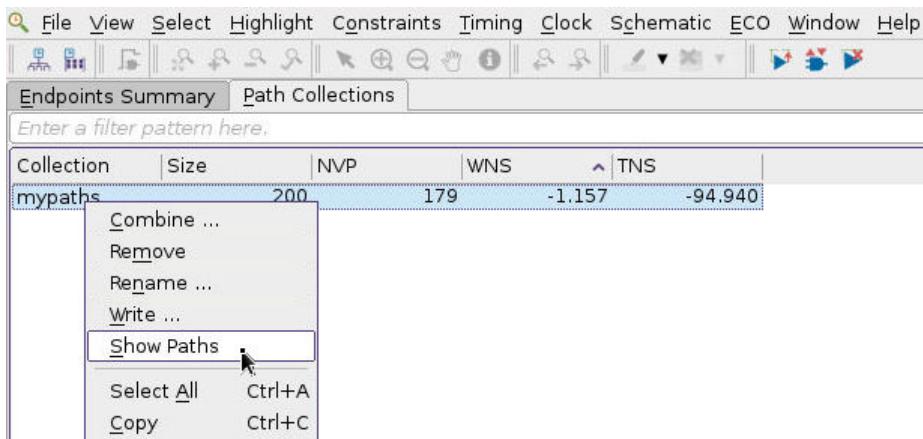
```
pt_shell> gui_start
```



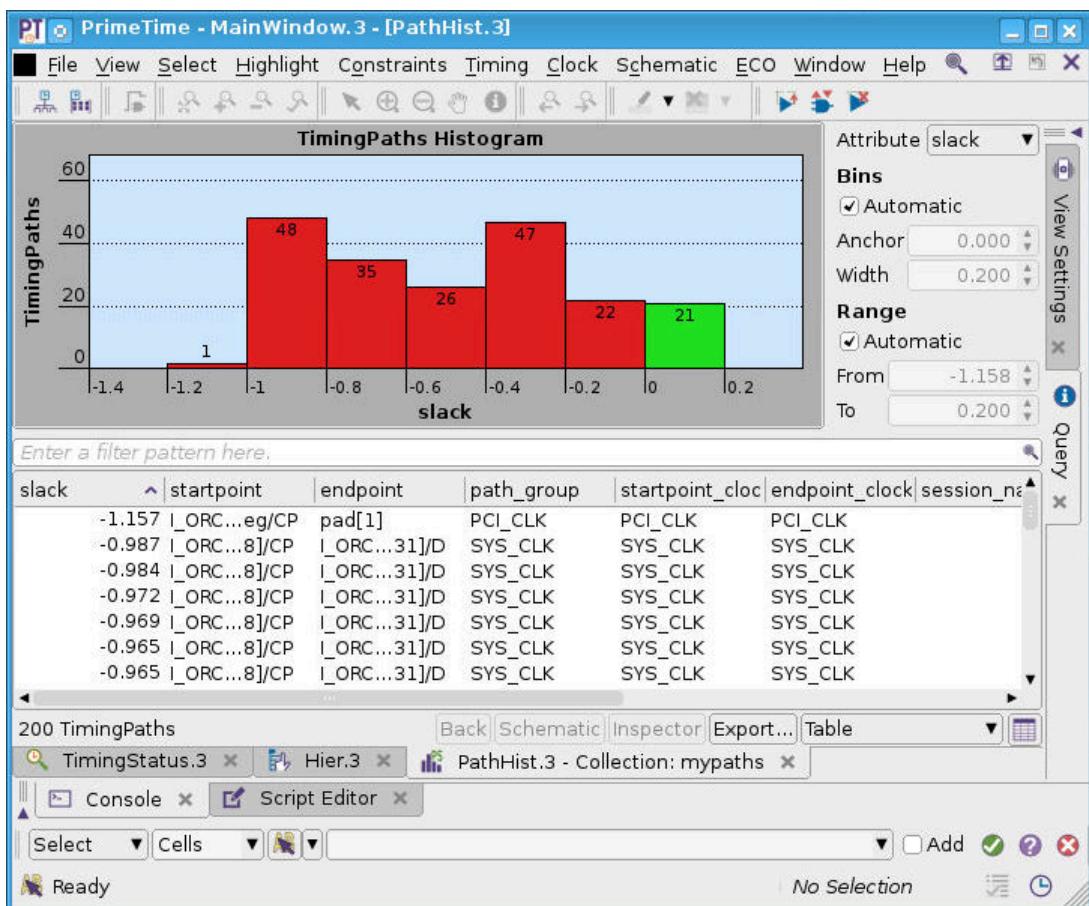
- Click the Path Collections tab.



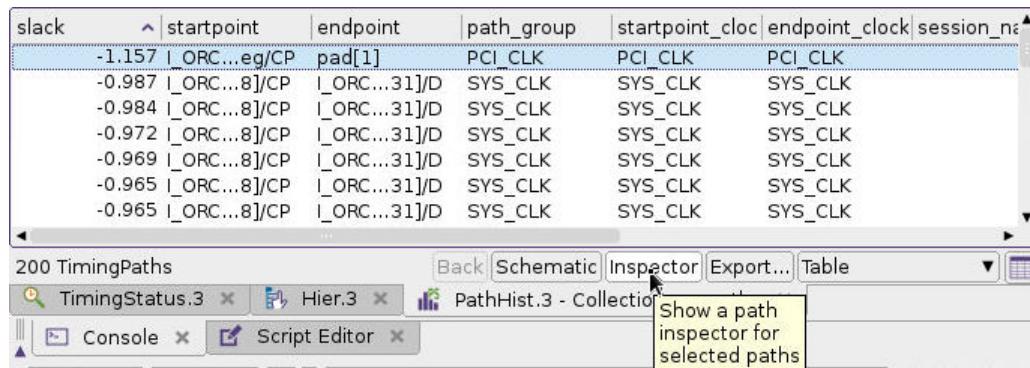
- Right-click the name of your path collection, and choose Show Paths.



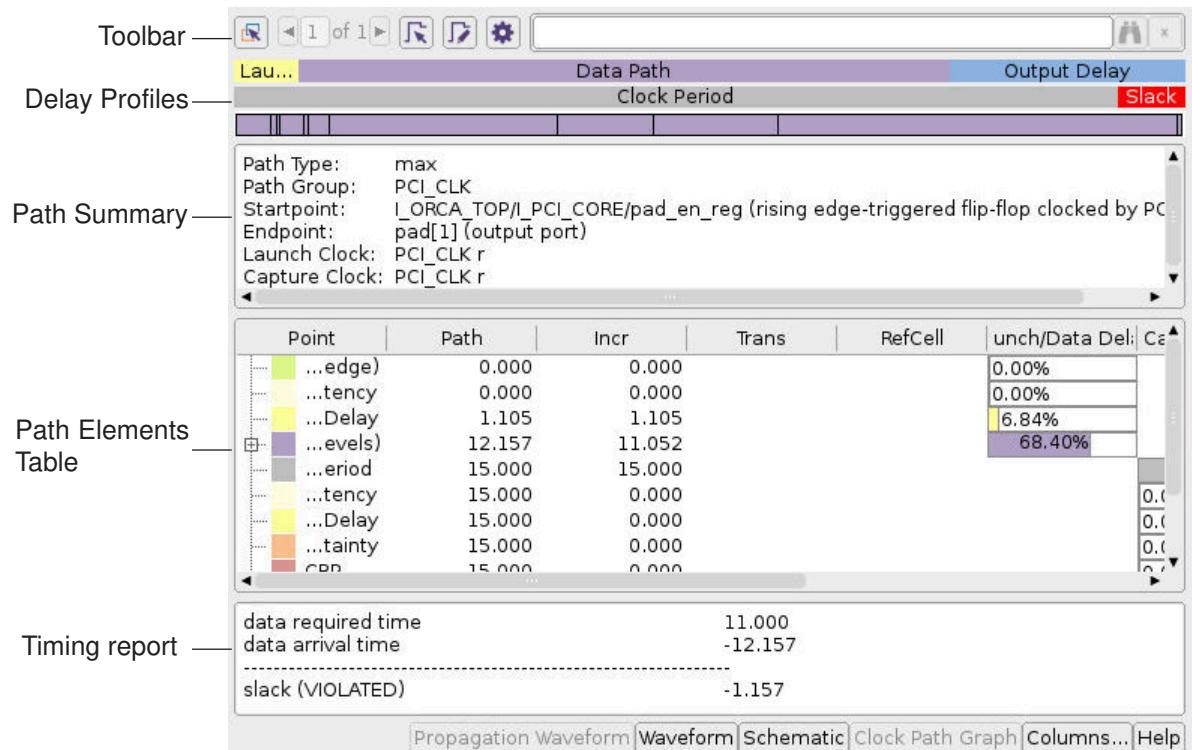
This displays a path slack histogram and a list of the paths in the path collection.



- Select and highlight the path of interest in the list (for example, the one with the worst slack), then click the Inspector button.

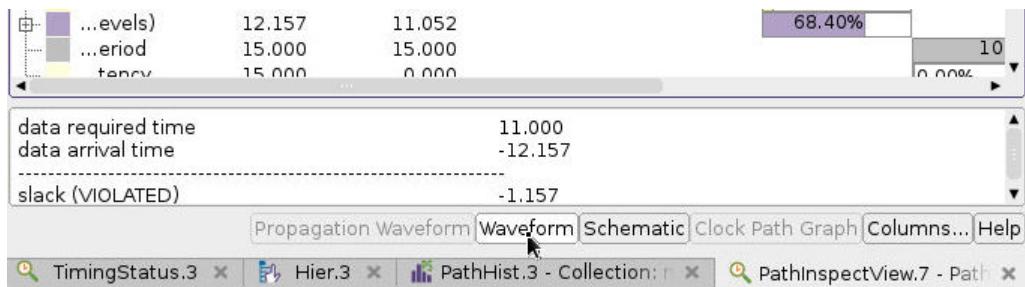


This opens the path inspector, which shows detailed information about the path.



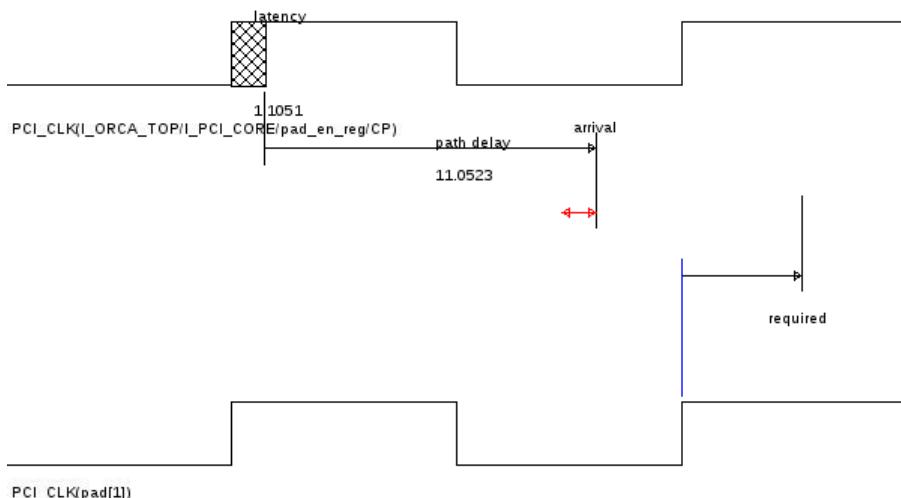
You can learn more about the path inspector by clicking the Help button in the lower-right corner.

6. Click the Waveform button.

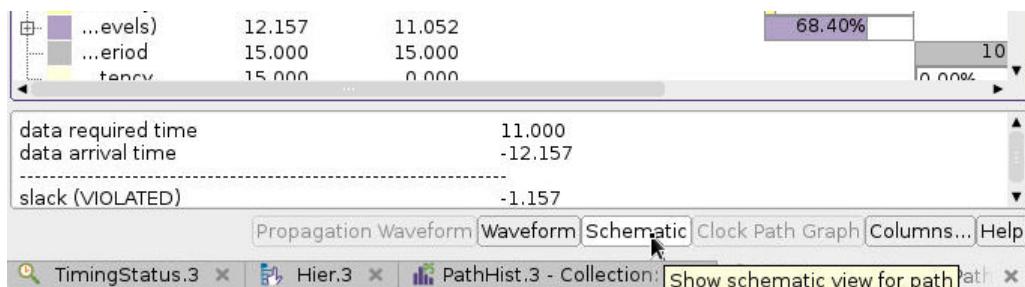


This opens a display of the timing waveforms.

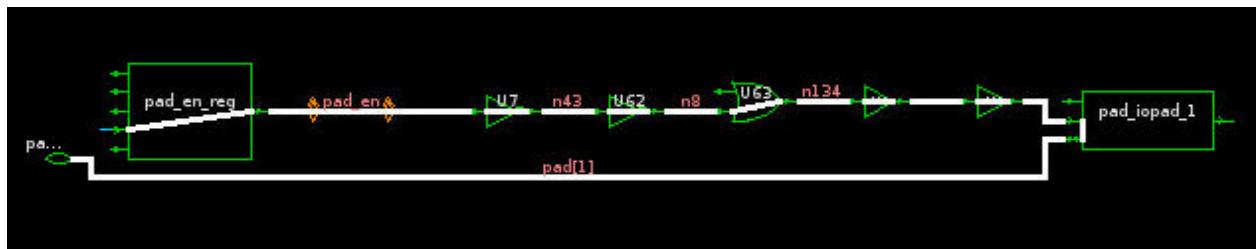
Path from 'I_ORCA_TOP/I_PCI_CORE/pad_en_reg/CP' to 'pad[1]' (slack = -1.15743)



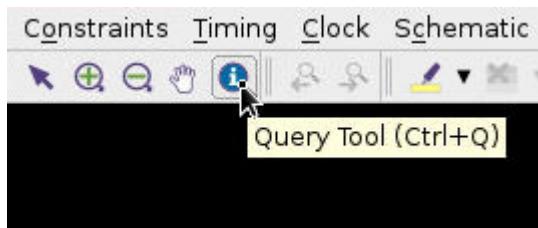
7. Click the Schematic button.



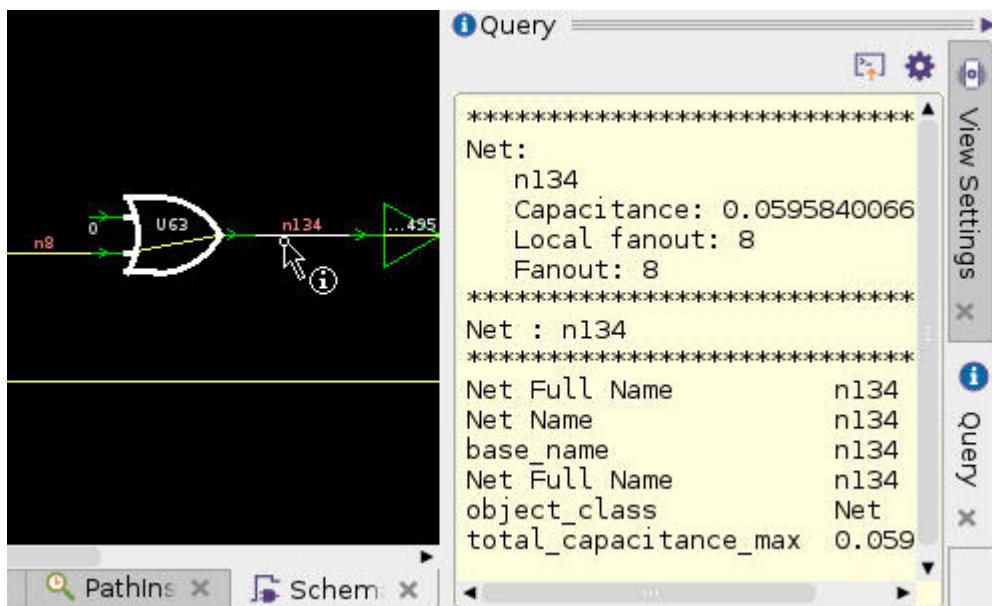
This opens a schematic of the timing path.



8. Click the Query Tool icon.



9. Go to the schematic and click on various objects (cells, nets, pins, pads). Each time you click on an object, information about that object is displayed in the Query window.



This quick start guide introduces only a few of the GUI features. To learn more, explore the GUI commands and dialog boxes, and refer to the remaining sections of this chapter as needed.

GUI Windows

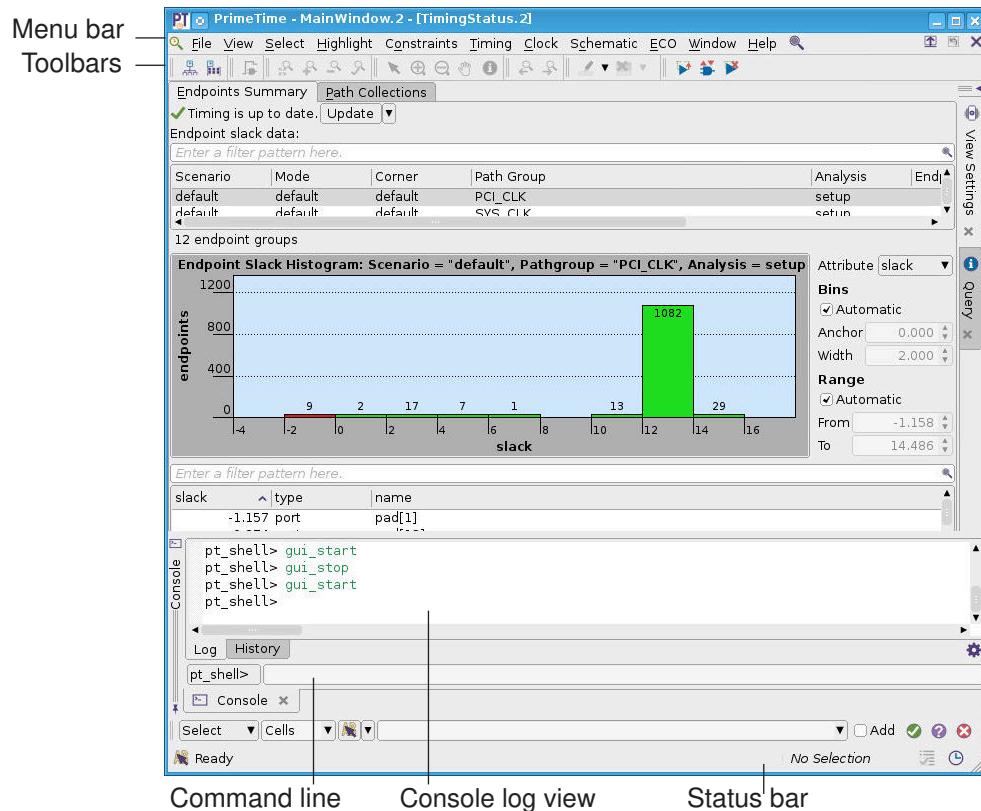
The PrimeTime GUI displays information in windows that provide a flexible working environment for performing different tasks. These windows operate independently in your X windows environment, but they share the same designs in memory, the same current timing information, and the global selection data 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. You can open other views as needed, depending on the types of analysis that you need to perform.

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 Linux desktop.

The following figure shows an example of the main window.

Figure 302 PrimeTime Main Window



The menu bar contains menus with the commands you need to work in the window. 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.

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.

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.

See Also

- [View Windows](#)
 - [Toolbars and Panels](#)
 - [Hierarchy Browser](#)
 - [Console](#)
 - [Object Selection](#)
 - [Setting GUI Preferences](#)
-

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.

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 Window, 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.

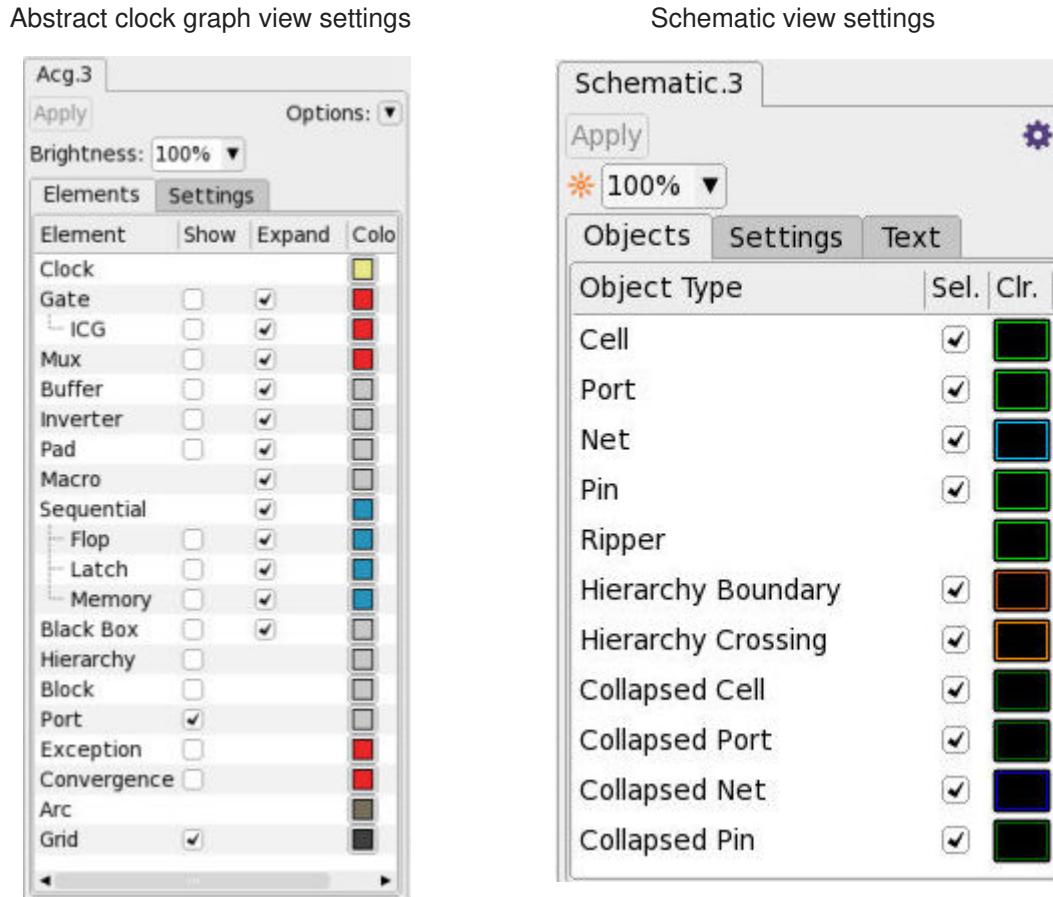
See Also

- [View Settings Panel](#)

View Settings Panel

You can view and modify display settings for the active view by changing options on the View Settings panel. The following figure shows the default appearance of the View Settings panel for abstract clock graph views and for schematic views.

Figure 303 View Settings Panel



To display or hide the View Settings panel, choose Choose View > Toolbars > View Settings, or press the F8 key. The available view settings depend on the window type. By default, when you change an option setting, the option and the Apply button turn blue. Click the Apply button to put the change into effect.

To operate the View Settings panel, choose commands from the Options menu or gear icon. You can save and restore your preferences or write your preferences to a script. An “Auto apply” option lets you bypass the Apply button and apply all changes immediately.

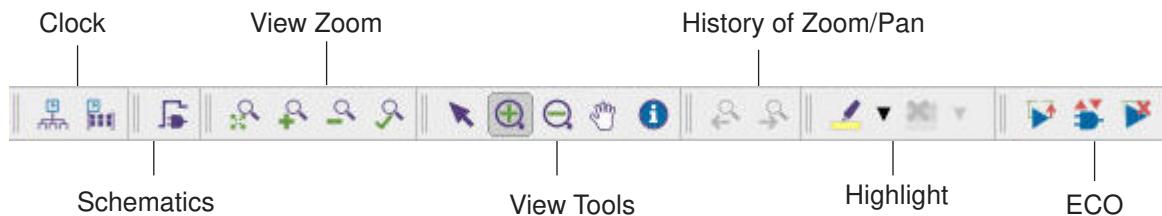
See Also

- [Examining Timing Paths and Design Logic 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 304 Main Window Toolbars

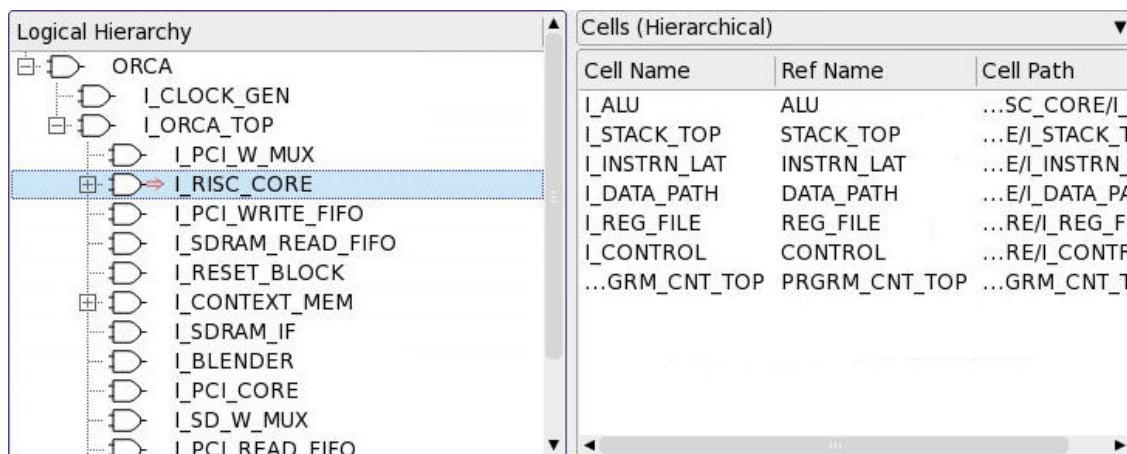


To display or hide a toolbar or panel, choose View > Toolbars > *toolbar_name*.

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 View > Hierarchy Browser.

Figure 305 Hierarchy Browser



The pane on the left lets you browse the cell hierarchy using the [+] and [-] buttons. 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, nets, and so on.

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 clicking on the column heading and adjust the widths of individual columns.

When you select an object in the hierarchy browser, the object is also selected in other views, such as schematic views.

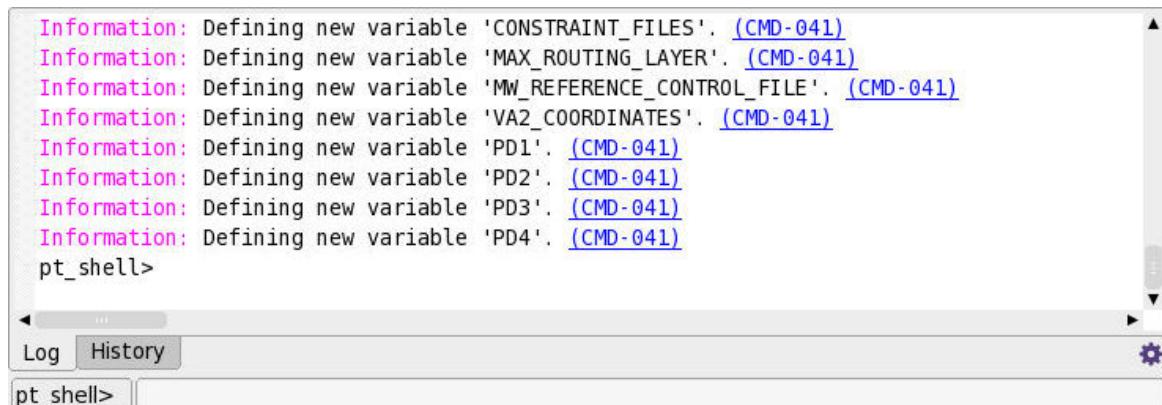
See Also

- [View Windows](#)

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 the following figure. You can enter commands in the long box at the bottom, to the right of the pt_shell button.

Figure 306 Console



Operating the console mirrors what is entered and displayed in the pt_shell terminal window. You can enter commands and view the response in either the pt_shell terminal window or in the console. 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 response from the tool.
- Click the History tab to view a history of recently executed commands.

The gear icon in the lower-right corner provides options to search, select, copy, and reuse the contents of the console.

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 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](#)

See Also

- [Selecting or Highlighting Objects By Name](#)
- [Viewing the Current Selection](#)

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. An easy method is to go to the Path Collections table and click the path of interest.

To create a path collection, use a command similar to the following:

```
set mypath1 [get_timing_paths -from ... -to ...]
```

See Also

- [Object Selection](#)

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.

To set GUI preferences, choose View > Preferences.

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](#)

Analyzing Timing Path Collections

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. 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

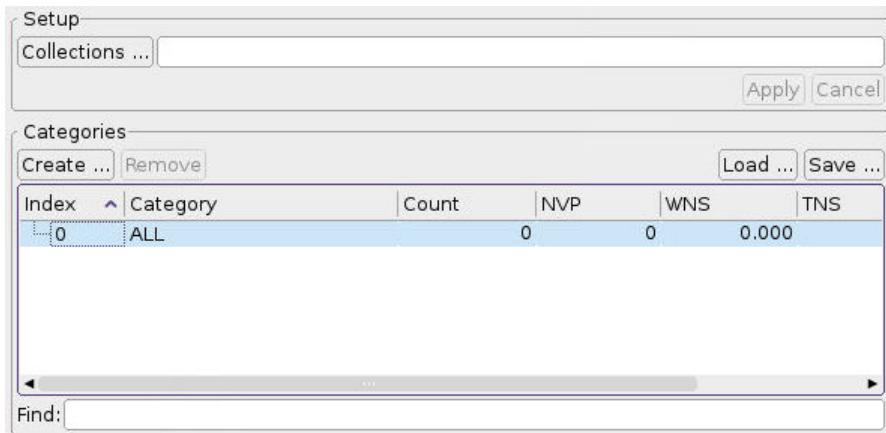
- Categorize paths by path group to see if timing violations are specific to a particular path group
- Create a custom category rule for paths with slack violations of more than one clock cycle to determine whether you need to specify multicycle path constraints on the paths
- Categorize the paths by 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,

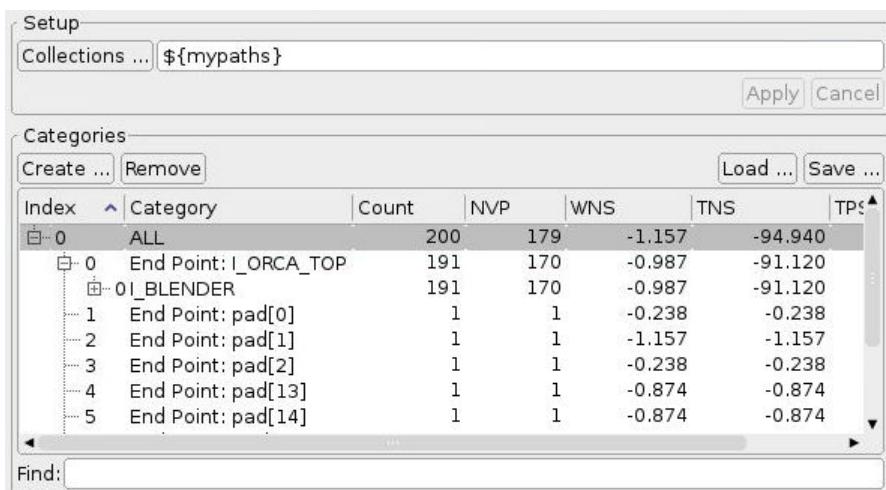
- ▶ Choose Timing > Path Analyzer.

Figure 307 Initial Path Analyzer



For further analysis, you need to load the timing path collections (Collections button) and categorize the timing paths (Create button).

Figure 308 Path Analyzer After Loading and Categorizing Paths



For details, see [Loading Path Collections](#) and [Categorizing the Timing Paths](#).

The path analyzer window displays the path categories in an expandable tree. The All category contains all the paths that you have loaded into the path analyzer. Beneath this category are the paths arranged in the categories you have chosen or defined. In the figure, the paths are organized by timing endpoint.

For each category, the columns display the following information:

- Category name
- Count – number of paths in the category

- NVP – number of violating paths
- WNS – worst negative slack
- TNS – total negative slack
- TPS – total positive slack

You can expand or collapse a category or subcategory by clicking the “+” or “–” button. Right-click the button to choose “Expand All” or “Collapse All.”

You can right-click a category containing paths that you want to analyze further with other analysis tools:

- Select Category Paths
- Create Histogram
- Create Data Table
- Create Path Analyzer

For more information about using the path analyzer, see

- [Loading Path Collections](#)
- [Categorizing the Timing Paths](#)
- [Saving Path Categories to a File](#)
- [Loading Path Categories From a File](#)
- [Marking Blocks and Applying Block Category Rules](#)
- [Creating Custom Category Rules](#)

See Also

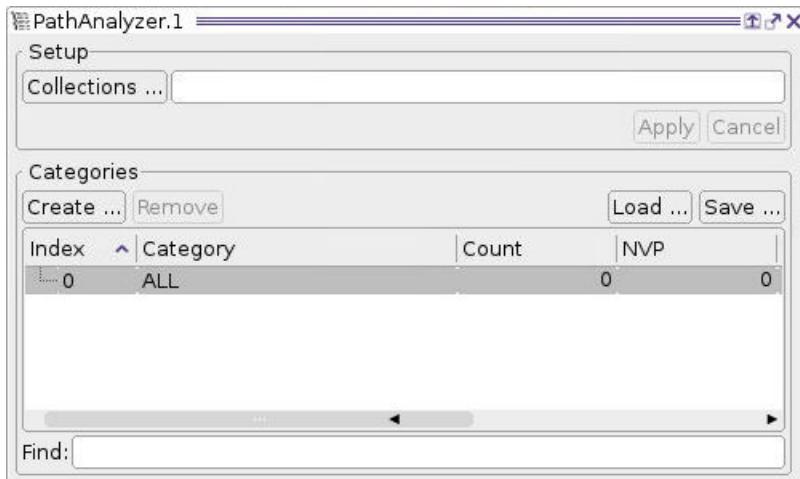
- [Interactive Multi-Scenario Analysis Flow](#)

Loading Path Collections

To load timing path collections into the path analyzer,

1. Choose Timing > Path Analyzer to open the path analyzer window.

Figure 309 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 you want to load, and click OK.
 - Enter the names of one or more collections in the Collections box, for example, \$mypaths1 \$mypaths2
 - Enter one or more `get_timing_paths` commands enclosed braces, for example, `[get_timing_paths -max_paths 20] [get_timing_paths -delay_type min]`
The entries in the Collections field appear in blue.
3. Press the Enter key or click Apply. The ALL category is updated with the number of paths and other information about the paths in the collections.

See Also

- [Categorizing the Timing Paths](#)
- [Analyzing Timing Path Collections](#)

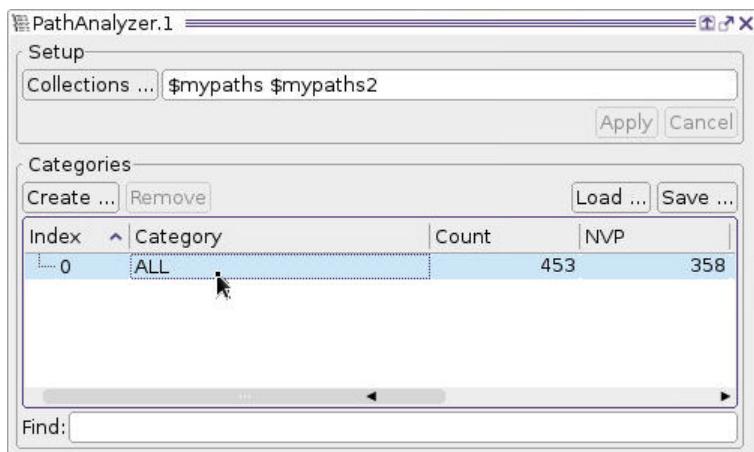
Categorizing the Timing Paths

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 create. 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.

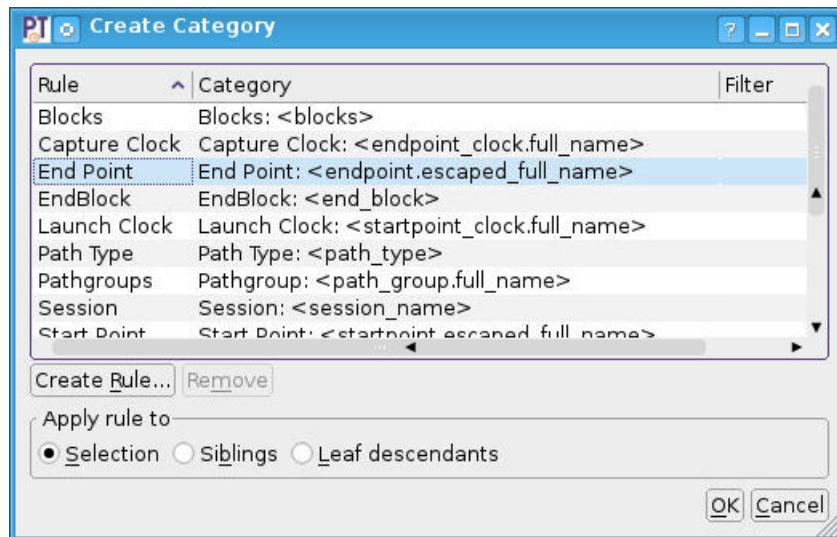
Figure 310 Path Analyzer Categories



2. Click the Create button.

The Create Category dialog box lists the existing category rules.

Figure 311 Create Category Dialog Box



3. Select a rule, for example, "End Point".
4. Apply the rule to the selection, its siblings, or its leaf descendants by selecting the appropriate option.
5. Click OK.

The list is updated with the paths organized by the specified rule. Now you can select paths that end at specific endpoints.

Figure 312 Path List Updated With New Categories

Categories							
		Create ...	Remove	Load ...		Save ...	
Index	Category	Count	NVP	WNS	TNS	TPS	
0	ALL	200	179	-1.157	-94.940	2.193	
0	End Point: I_ORCA_TOP	191	170	-0.987	-91.120	2.193	
1	End Point: pad[0]	1	1	-0.238	-0.238	0.000	
2	End Point: pad[1]	1	1	-1.157	-1.157	0.000	
3	End Point: pad[2]	1	1	-0.238	-0.238	0.000	
4	End Point: pad[13]	1	1	-0.874	-0.874	0.000	

Find:

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.

To remove a category or subcategory,

1. Select the category.
2. Right-click and choose the desired removal option.

See Also

- [Saving Path Categories to a File](#)
 - [Loading Path Categories From a File](#)
 - [Marking Blocks and Applying Block Category Rules](#)
 - [Creating Custom Category Rules](#)
 - [Analyzing Timing Path Collections](#)
-

Saving Path Categories to 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 Timing Path Collections](#)

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 to a File](#)
- [Categorizing the Timing Paths](#)
- [Analyzing Timing Path Collections](#)

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 56](#) shows the available category rules for blocks.

Table 56 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 Timing Path Collections](#)
- [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

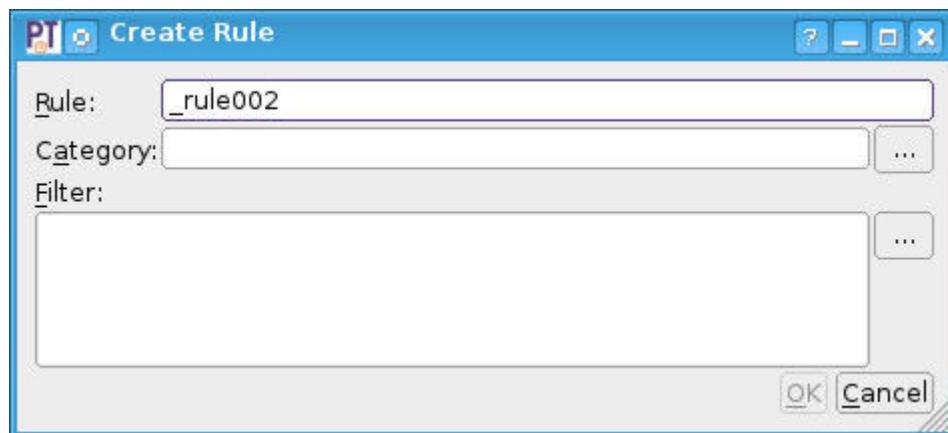
You can create custom category rules based on the timing path attributes. The priority of the custom categories is stored in the Category file. If a path does not satisfy any of the available grouping criteria, it is placed in the Uncategorized group for each level that has at least one subcategory.

To create a custom category rule,

1. In the path analyzer window, click the Create button.
2. In the Create Category dialog box, click the Create Rule button.

The Create Rule dialog box appears.

Figure 313 Create Rule Dialog Box



3. Enter a rule name in the Rule box.

This name is for reference only.

4. Click the Category button and select an attribute in the Category Attribute Chooser dialog box.

5. Click the Filter  button and define a filter expression in the Filter Attribute Chooser dialog box.
6. 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.

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 can 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 Timing Path Collections](#)

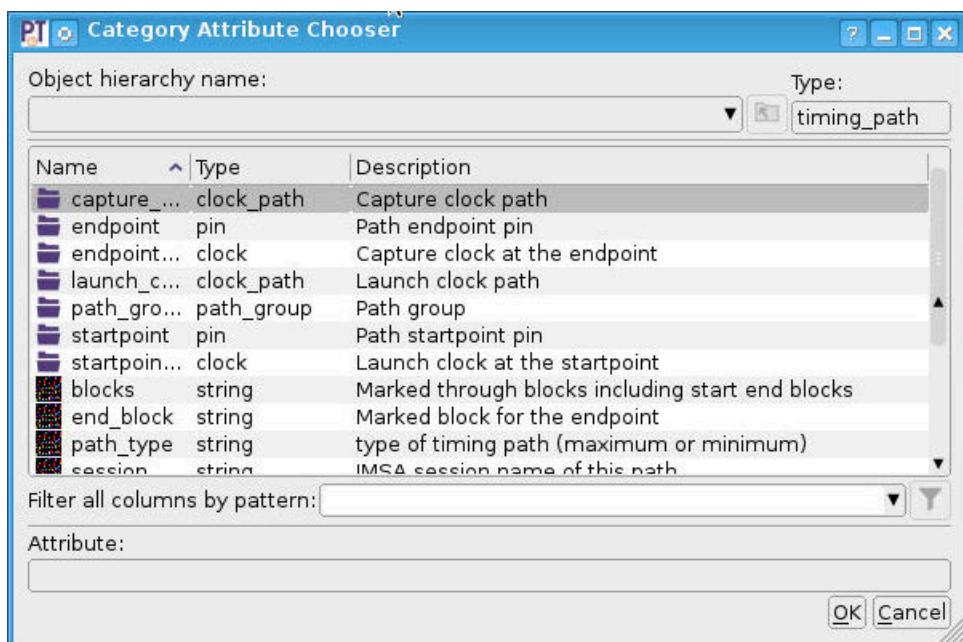
Selecting Category Attributes for a Custom Category Rule

To select an attribute in the Category Attribute Chooser dialog box,

1. Click the Category  button in the Create Rule dialog box.

The Category Attribute Chooser dialog box appears with a list of the available category attributes.

Figure 314 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.
 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.
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.

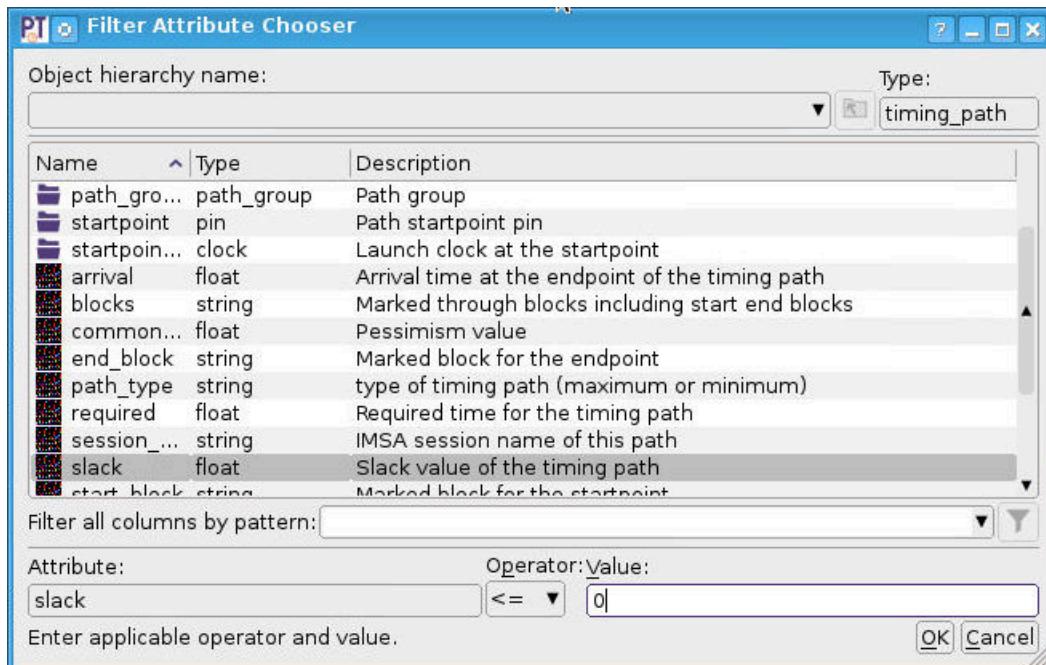
You can concatenate multiple filters just as you would concatenate logical expressions. You can construct full expressions by using `&&` and `||` to concatenate multiple filters. Select `&&` to create an AND relationship between filter expressions, or select `||` to create an OR relationship. Use parenthesis `()` 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.

Figure 315 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.
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.
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.
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](#)
-

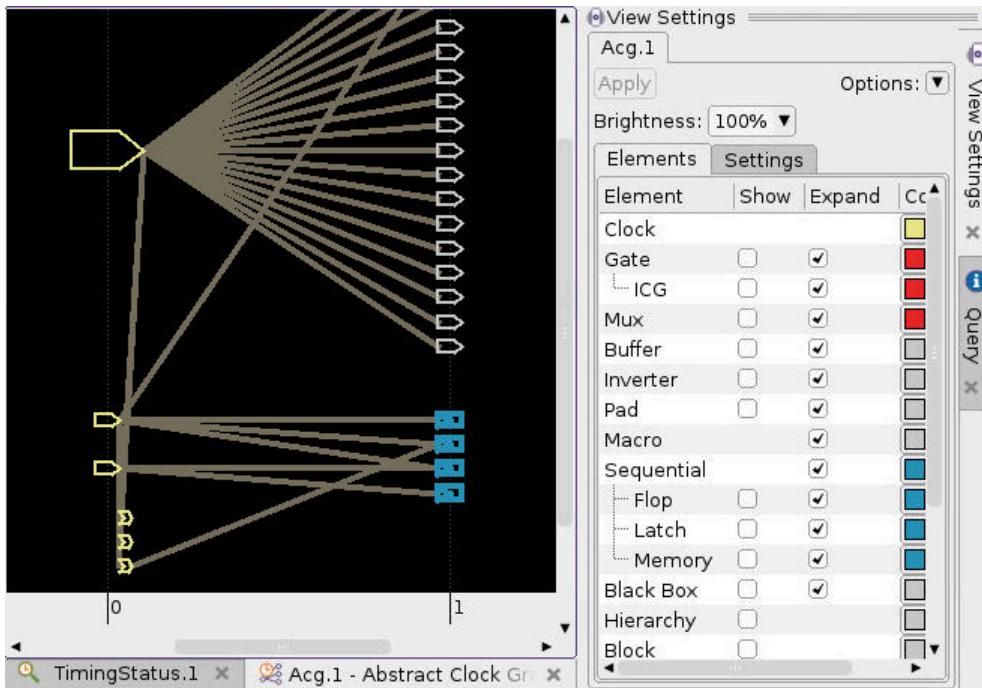
Examining Clock Paths in an Abstract Clock Graph

The abstract clock graph view allows you to visualize the relationships between clocks and between the driving cells and output loads in a clock network. 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.

The following diagram shows an example of an abstract clock graph window.

Figure 316 Abstract Clock Graph Window



The 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.

In the example shown in the figure, the clock graph contains three primary clocks and three generated clocks. 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.

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.

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](#)
- [Viewing Clock Latency Over Time](#)
- [Displaying the Clock Trees Hierarchically](#)

See Also

- [Selecting or Highlighting Objects By Name](#)

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.
3. In the dialog box, specify the clock graph display parameters, and click OK.

The GUI opens a new abstract clock graph window and displays the clocks in an abstract clock graph view.

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.

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.
3. Click Apply.

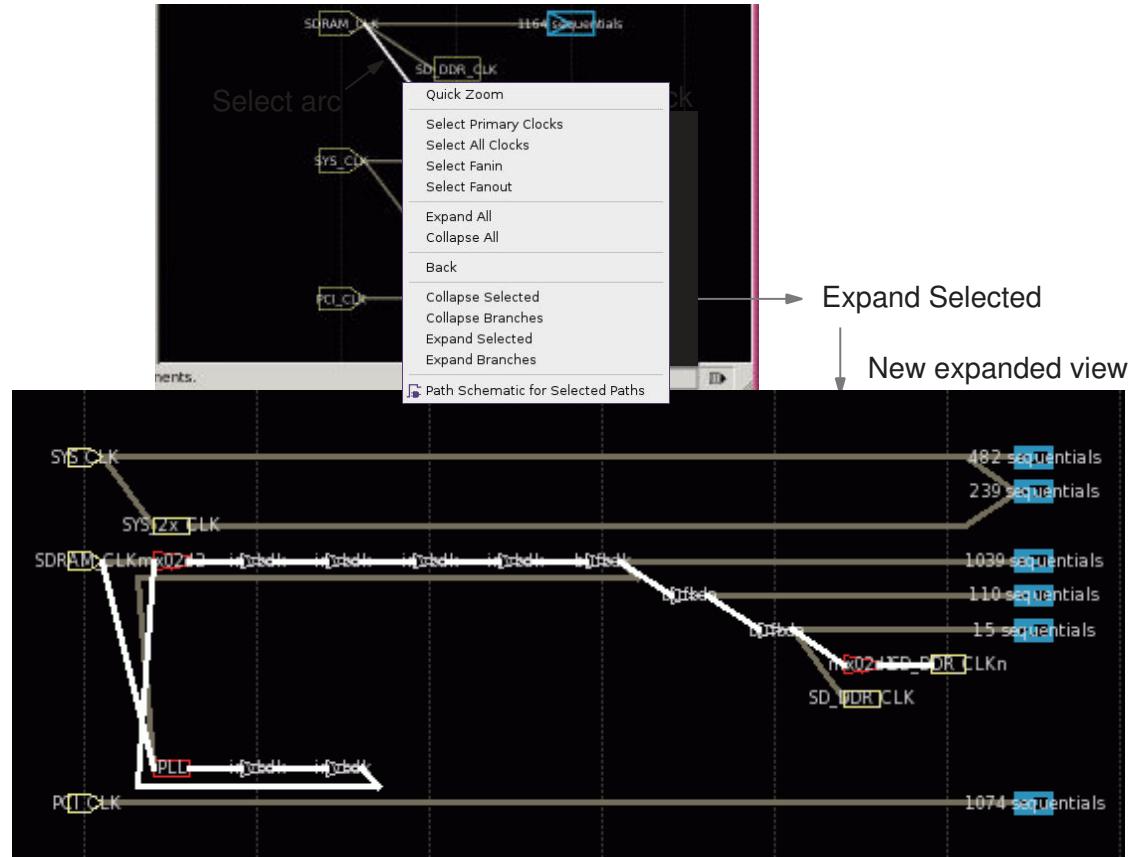
You can expand individual arcs to display their contents or collapse selected objects into arcs. You can also expand metacells to display the sequential cells or collapse selected sequential cells into metacells.

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.

The following figure 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 317 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](#)

- [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.

To reverse an operation in an abstract clock graph view,

- ▶ Right-click and choose Back.

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.

To collapse one or more side branches,

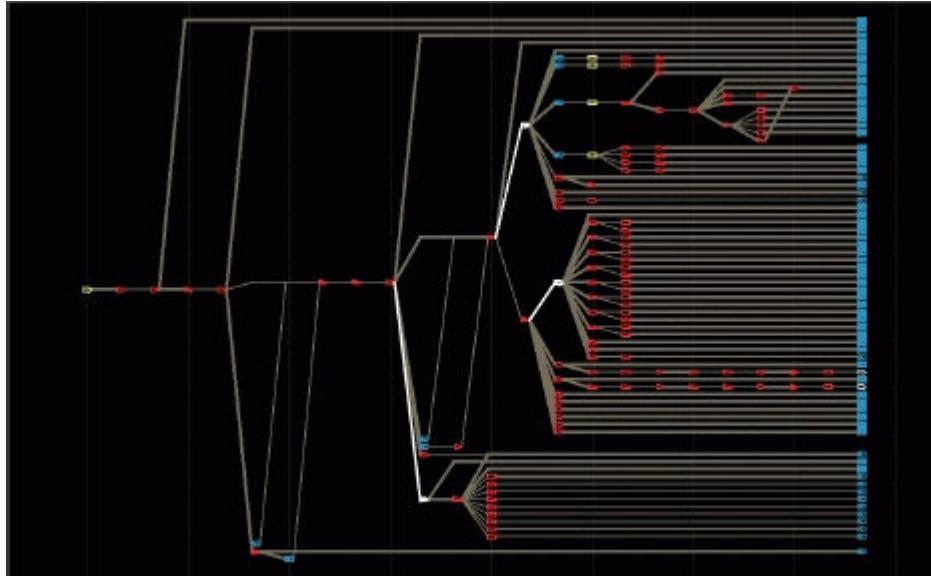
1. Select the branch roots that you need to collapse.
2. Right-click and choose Collapse Branches.

To expand collapsed side branches,

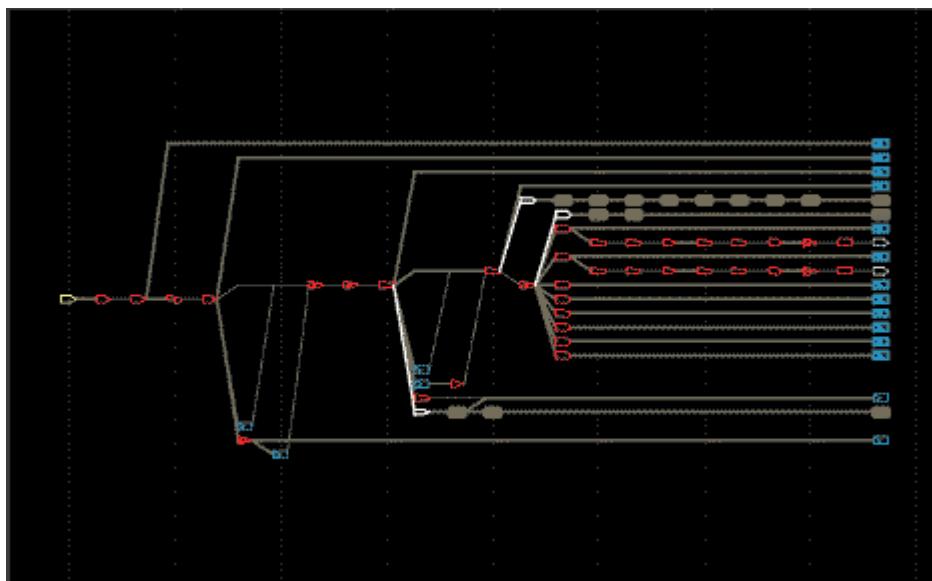
1. Select the branch roots you need to expand.
2. Right-click and choose Expand Branches.

Figure 318 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](#)

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.

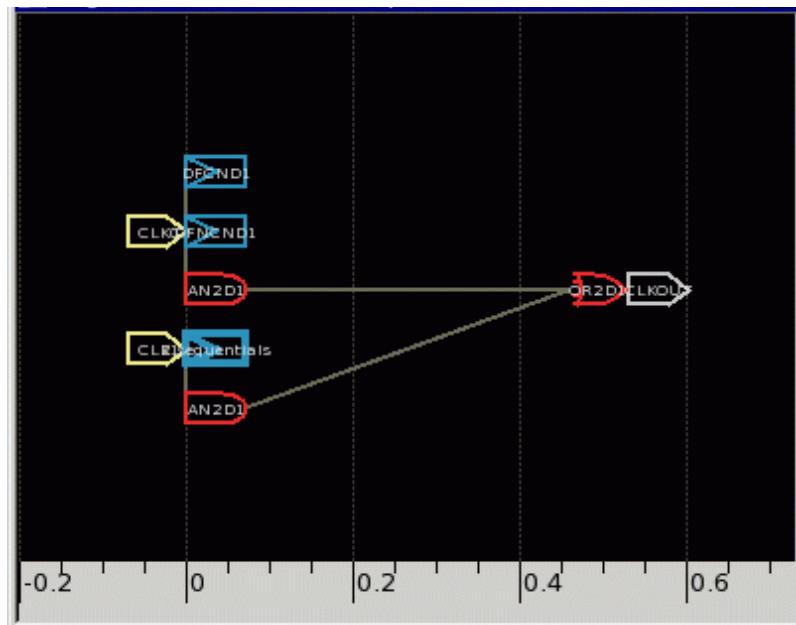
The clock latency layout displays concise and consistent timing information for the clock trees. The x-axis represents clock tree logic relative to time. The graph positions logic gates based on the latency of their input clock pins.

- 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 319 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. In the View Settings panel, under the Elements tab, select the Hierarchy option.
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.

See Also

- [Examining Clock Paths in an Abstract Clock Graph](#)
- [Displaying and Hiding Clock Path Elements](#)

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.

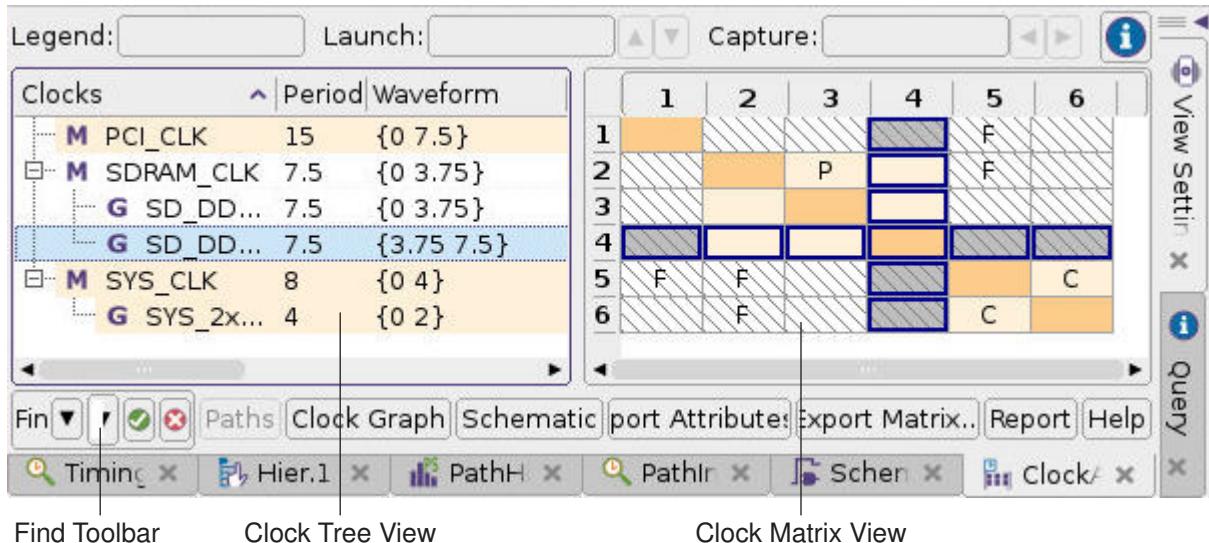
You can select one or more clocks in the clock tree view. By selecting a master clock, you also select the entire clock domain, including the generated clock domains beneath it.

- 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.

The following figure shows an example of the clock analyzer. You can locate specific clocks by using the Find toolbar at the bottom of the window.

Figure 320 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](#)
- [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. Select the clocks, right-click, and choose:

- Clock Graph of Selected Clocks.
- Clock Graph for Clock Domain.
- Schematic of Selected Clocks.
- 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

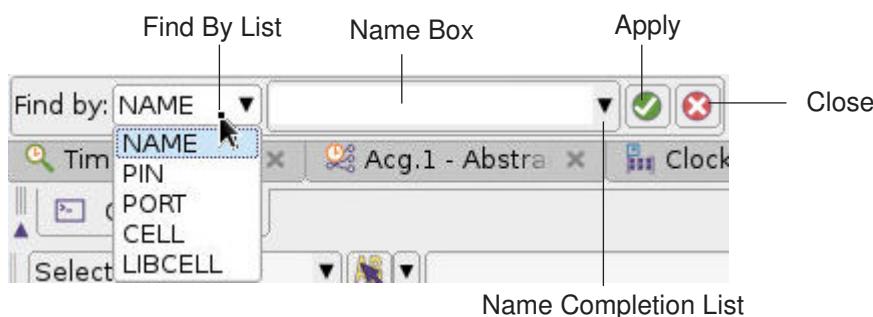
The clock analyzer provides a Find toolbar that can help you to locate a specific clock. You can perform multiple selections and select a range of clocks, and you can locate launch clocks based on the functional definition such as pins along the clock path.

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 appears below the clock tree view at the bottom of the clock analyzer window.

Figure 321 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.
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.

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

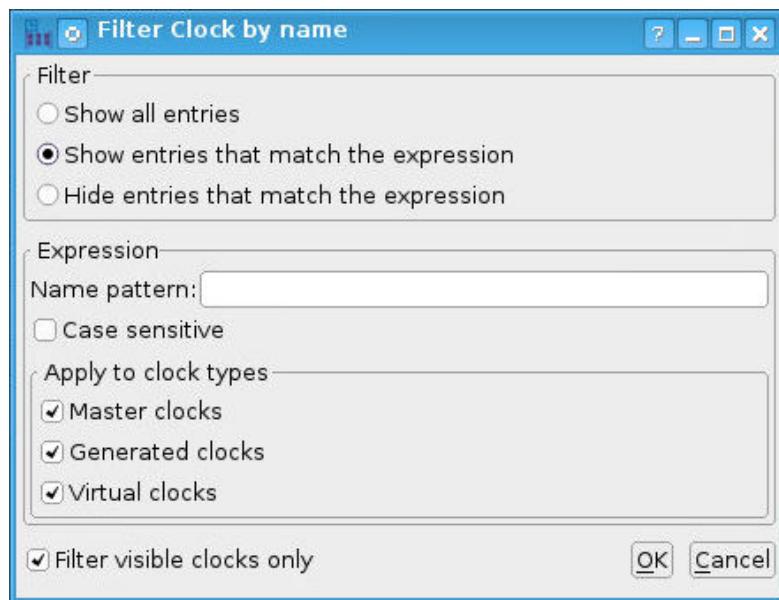
You can filter the clock domains to reduce the number of visible clocks by including or excluding clocks whose names match a certain pattern or type.

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.

Figure 322 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.
- To hide the clocks that match the filter criteria, select the “Hide entries that match the expression” option.

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.

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](#)

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 Save As dialog box appears.

2. (Optional) If you want to save all the clock attributes, select the “All clock attr” 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 to save the file in comma separated value format, which is compatible with most spreadsheet programs.

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.

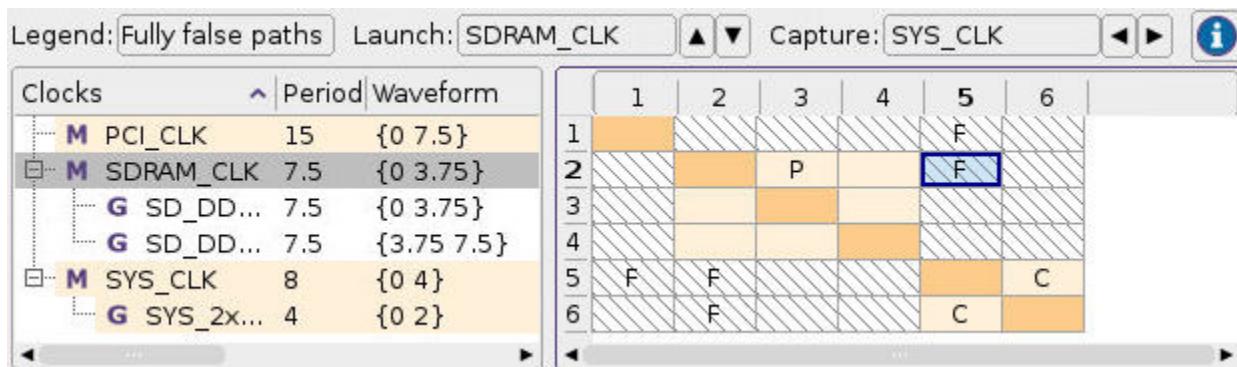
2. Specify the name and location for the file. Specify a file name with the .csv extension to save the file in comma separated value format.

3. Click Save.

Clock Matrix Symbols and Colors

The matrix in the Clock Analyzer shows the usage relationships between the clocks.

Figure 323 Clock Analyzer Matrix



Each cell represents a launch and capture clock pair. The cell is colored to visually show the relationship between the clocks. The following table shows the background cell pattern and color that indicates the synchronous or asynchronous relationships between the launch and capture locks.

Table 57 Background Cell Patterns

Pattern	Description
	The launch and capture clocks are from the same clock domains and asynchronous with respect to each other.
	The launch and capture clocks are from different clock domains and asynchronous with respect to each other.
	The launch and capture clocks are from different clock domains but synchronous with respect to each other.
	The launch and capture clocks are from the same clock domain and synchronous with respect to each other.
	The launch and capture clocks corresponding to a particular cell are the same clock.
	The launch and capture clocks are logically or physically exclusive.

The letter codes inside the matrix indicate the following types of clock crossings for paths with a launch and capture clock pair:

- no letter – 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

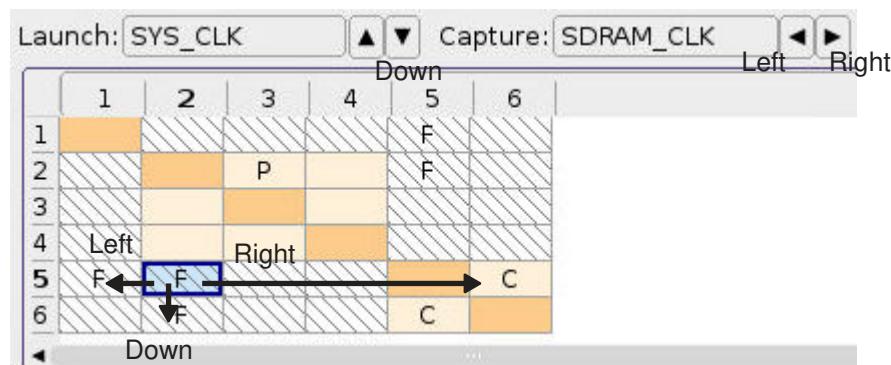
All paths between the selected launch and capture clock pair have been designated false paths

- P – Partially constrained paths

Only some paths between the selected launch and capture clock pair are designated false paths

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 button or arrow key moves the selection to the nearest cell that contains a relationship, skipping cells with no relationship.

Figure 324 Locating Launch and Capture Clocks in the Clock Matrix



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](#)

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.

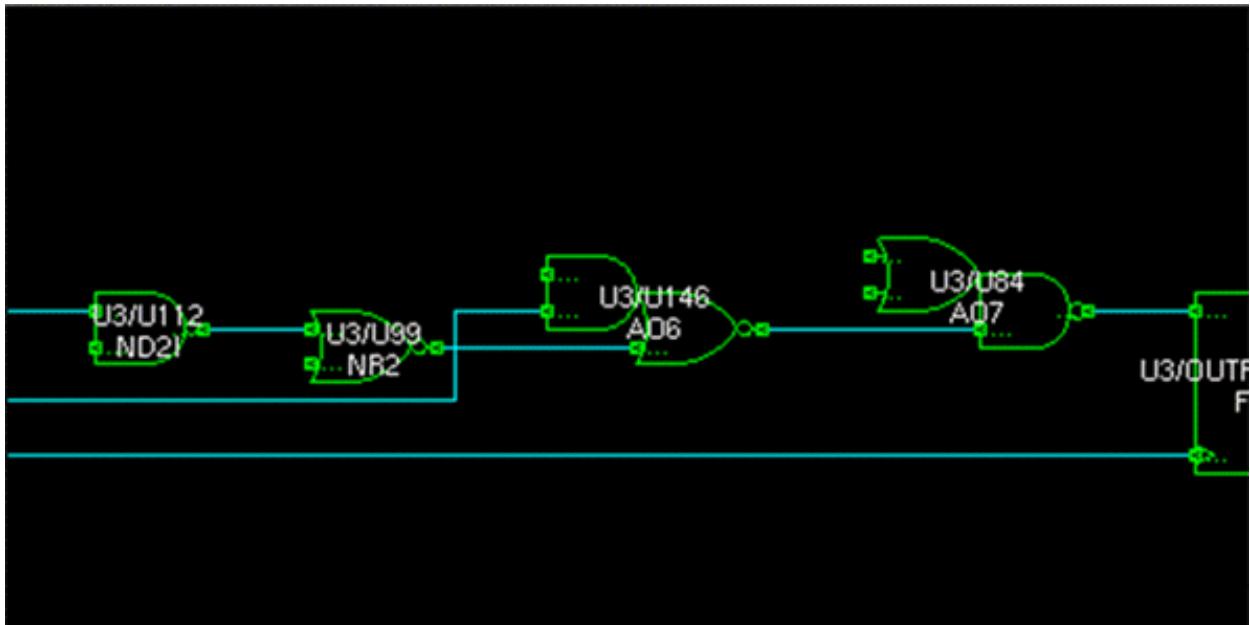
To open an instance schematic,

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. Choose **Schematic > Schematic View**.

Figure 325 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](#)
- [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](#)
 - [Reversing and Reapplying Schematic Changes](#)
-

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. 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. 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 326 Expanded and Collapsed Views of a Design Hierarchy



See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)
- [Moving Down or Up the Design Hierarchy](#)

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. 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,

- ▶ 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](#)

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,

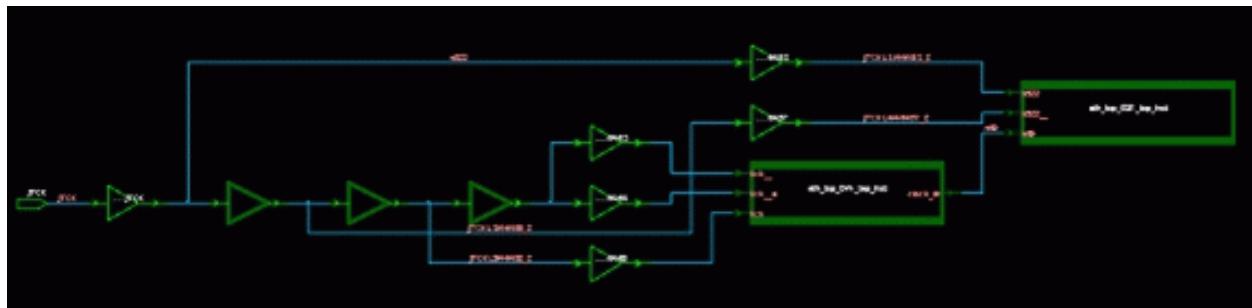
- ▶ Choose Schematic > Collapse > All Buffers/Inverters/Crossings By Tree.

To hide individual buffers, inverters, and hierarchy crossings,

1. Select the objects that you want to hide.
2. 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 327 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 328 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. 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

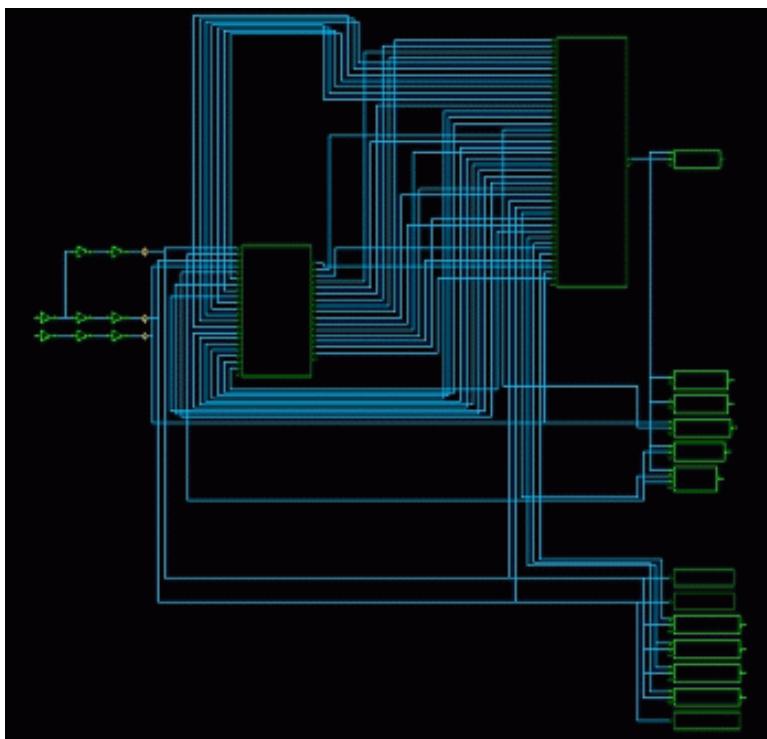
To simplify a schematic and hide unimportant details, you can hide the unconnected pins of macro cells by collapsing them into metapins. The hidden pins are represented by metapins of the same type: in, out, in/out, and bidirectional. A metapin is similar to a design pin but has a thicker line width 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 and a different color.

Figure 329 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

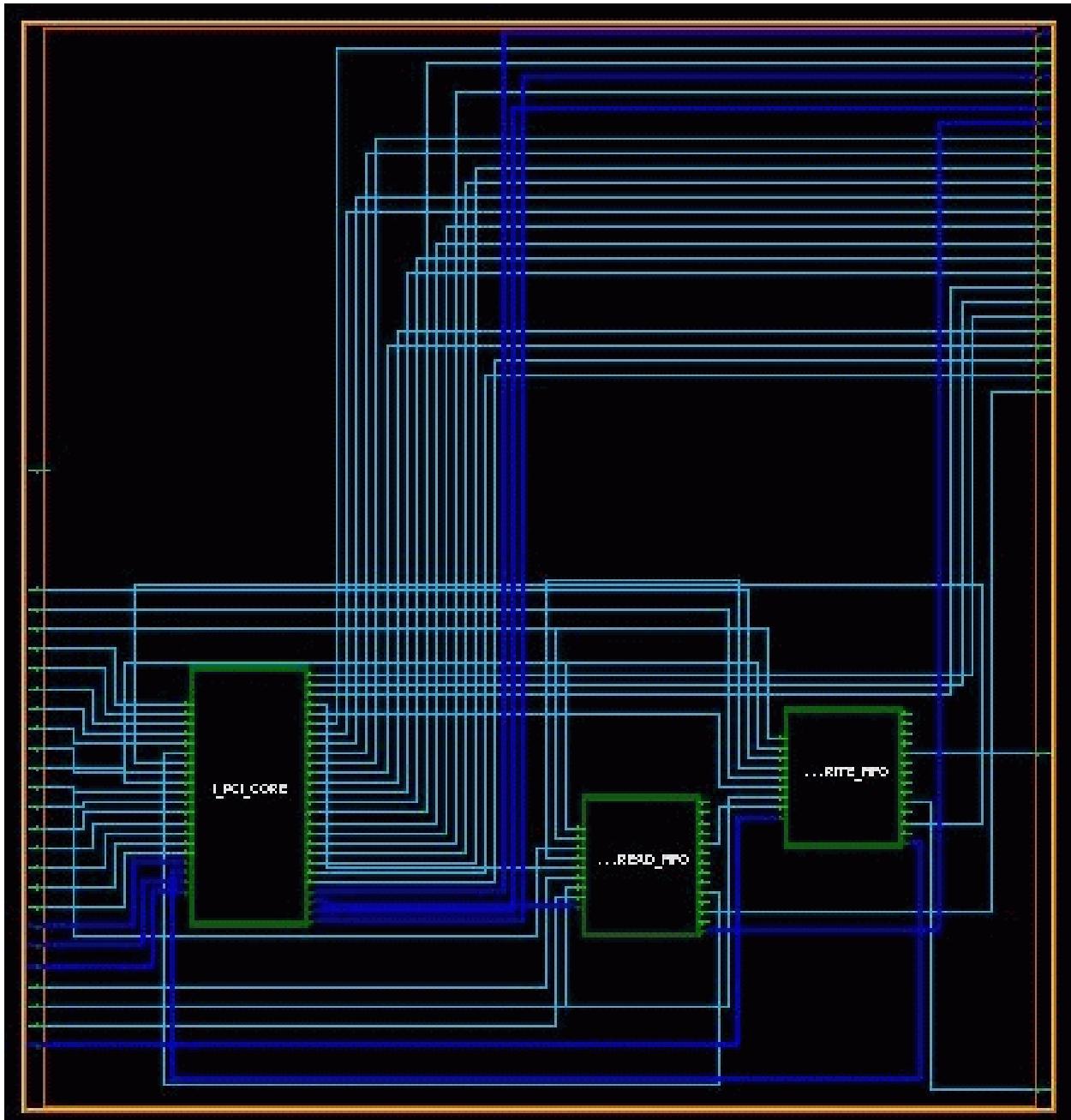
You can save space in a schematic by collapsing nets that are members of a bus and their associated pins. This allows you to focus on the nets that are of interest while keeping the overall context in view as you debug timing problems.

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, and the pins are rendered in a darker green.

Figure 330 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](#)

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](#)
- [Reversing and Reapplying Schematic Changes](#)

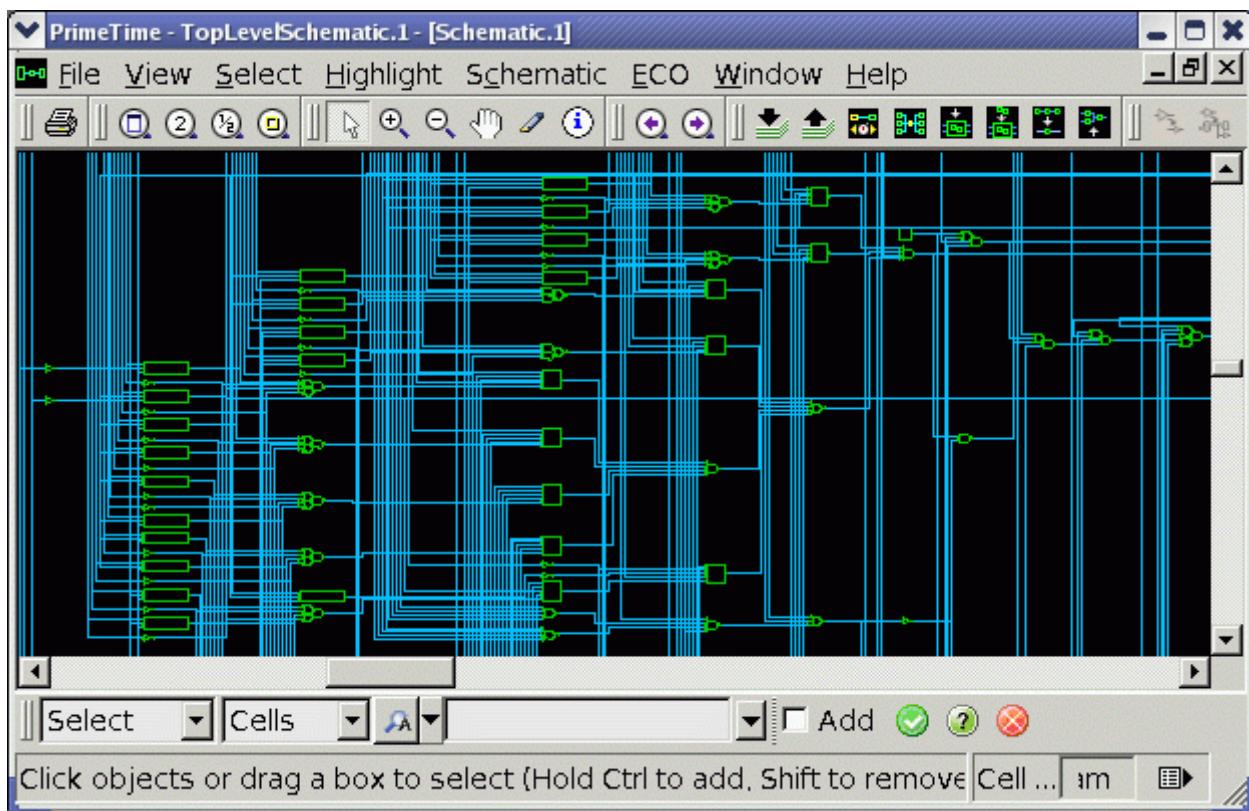
See Also

- [Examining Timing Paths and Design Logic in a Schematic](#)

Schematic Window

The schematic window contains its own menu bar, toolbars, and status bar.

Figure 331 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.

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.

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 Selection tool. To activate this tool, choose View > Mouse Tools > Selection Tool.

You can also highlight objects by selecting them and choosing a command on the Highlight menu. 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, choose View > Mouse Tools > Query Tool.

See Also

- [Viewing and Modifying Schematics](#)
- [View Settings Panel](#)

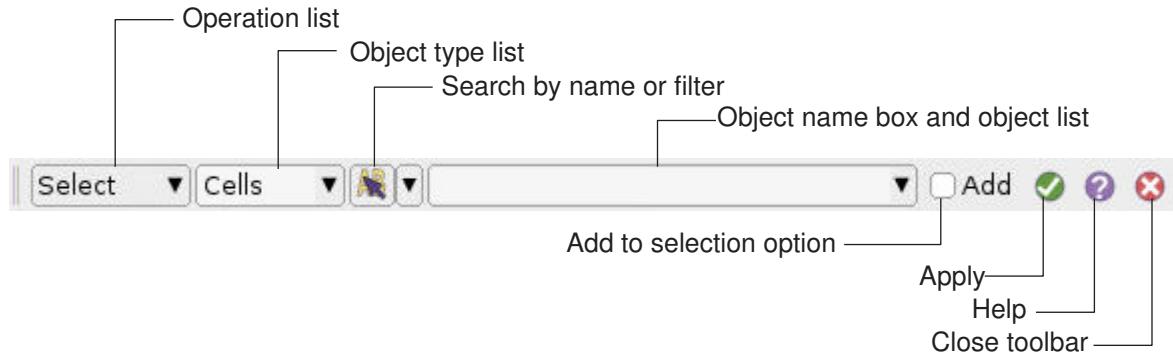
Selecting or Highlighting Objects By Name

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 332 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](#)

Highlighting Schematics By Using Filtering Rules

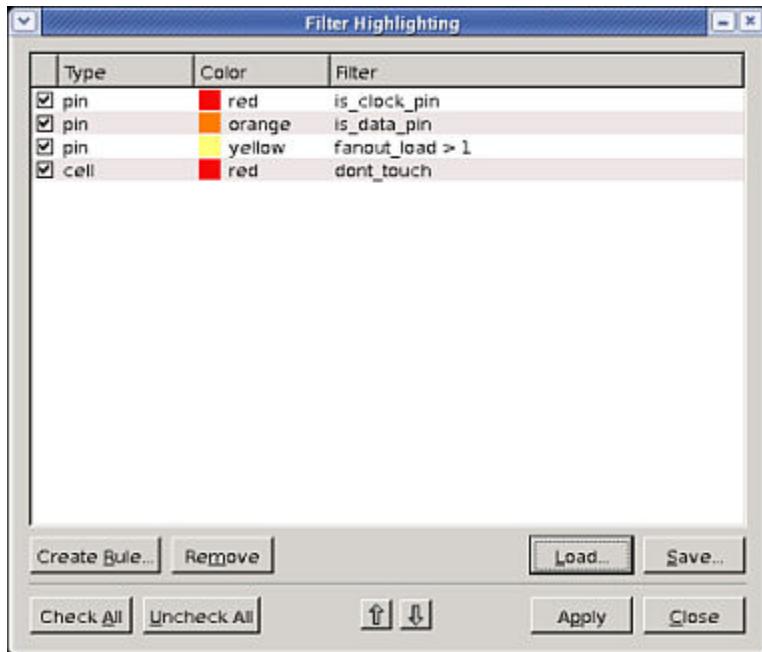
You can highlight elements according to filtering rules that you define.

To open the Filter Highlighting dialog box,

- ▶ Choose Highlight > Filter Highlighting.

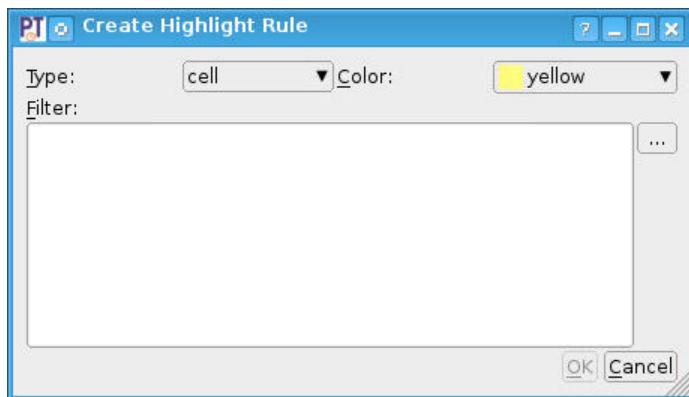
The following figure shows the Filter Highlighting dialog box after several rules have been defined.

Figure 333 Filter Highlighting Dialog Box



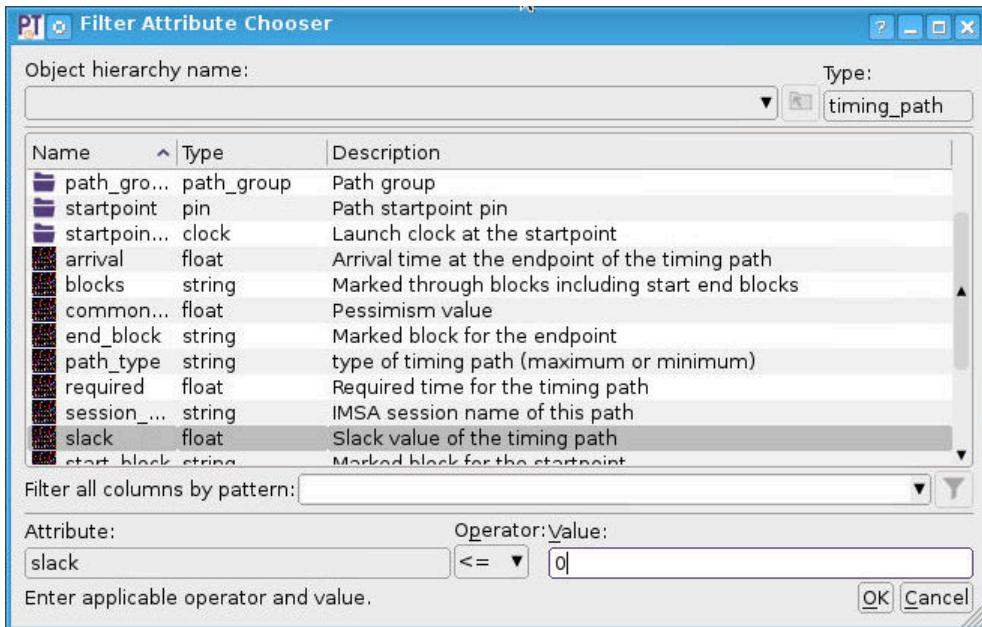
In the Filter Highlighting dialog box, you can specify the active highlighting rules and the order in which they are applied. To create a new filtering rule, click Create Rule to open the Create Highlight Rule dialog box.

Figure 334 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.

Figure 335 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](#)

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](#)
-

Inspecting Timing Path Elements

Use the path inspector to analyze various aspects of one or more timing paths. You can

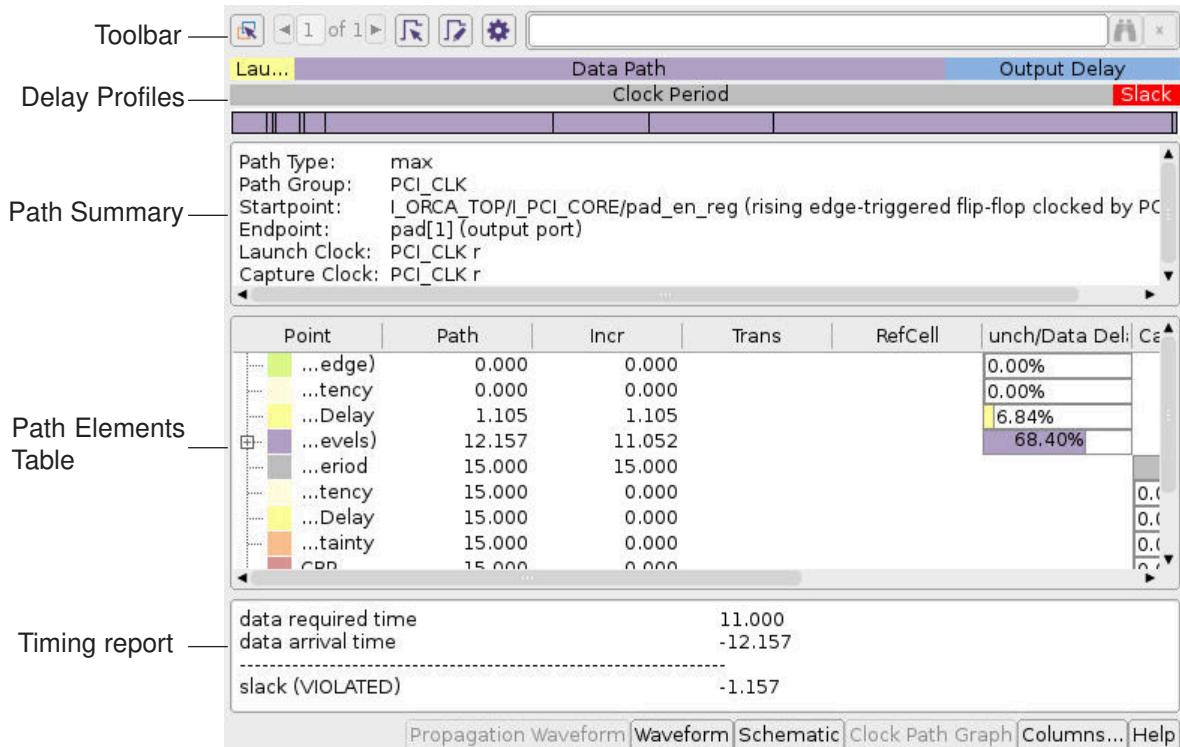
- View path profiles that represent the relative delay and slack contributions of various path components
- Examine timing report information that includes a path summary, path element data, and path slack details
- Perform additional analysis tasks such as selecting the path, highlighting the path, or finding text in the report information

To open the path inspector, right-click the path in a path list and choose Inspector.

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.

Figure 336 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.

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.

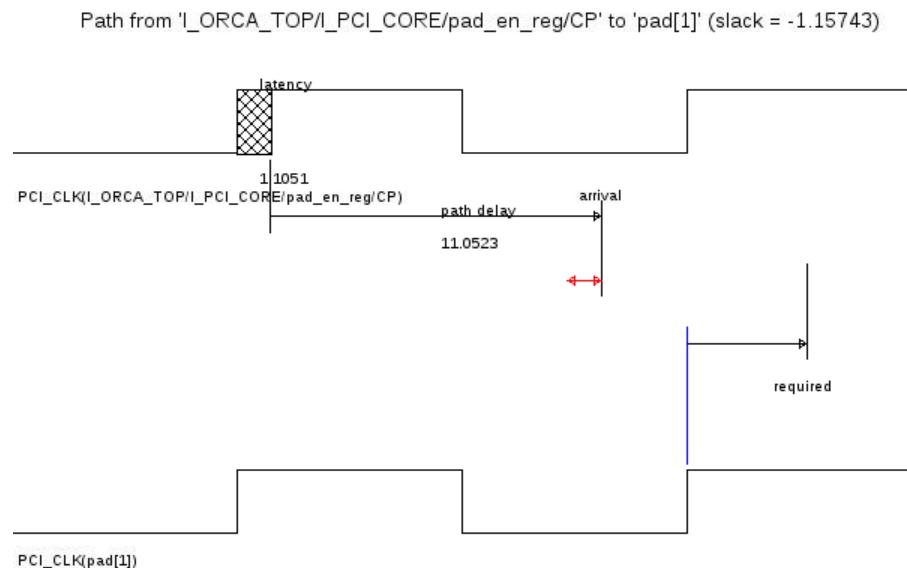
You can configure the path elements table by clicking the gear button and setting options in the Configure the Path Inspector dialog box. For details, see [Configuring the Path Inspector](#).

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 select path button on the path inspector toolbar.
- To highlight the path with the current highlight color, click the highlight path button on the path inspector toolbar.

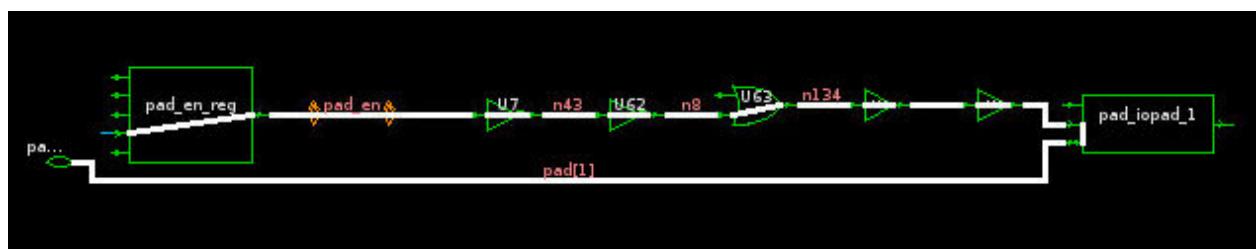
You can also view timing waveforms in the path inspector and display the path in a new schematic window. To view timing waveforms in the waveform view, click the Waveform button at the bottom of the path inspector window.

Figure 337 Waveform Display



To display the path in a new schematic view, click the Schematic button at the bottom of the path inspector window.

Figure 338 Schematic Display



For more information about working with the path inspector, click the Help button at the bottom of the window.

For more information, see

- [Loading Paths Into a Path Inspector Window](#)
- [Configuring the Path Inspector](#)

See Also

- [Recalculating Timing 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. Select one or more timing paths that you want to inspect.
2. Click the “Load selected paths” button in the upper-left corner of the path inspector window if the window is already open. Otherwise, you can right-click the path in a path list and choose Inspector.

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](#)
-

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, right-click in the table, choose Format, and then the formatting option:

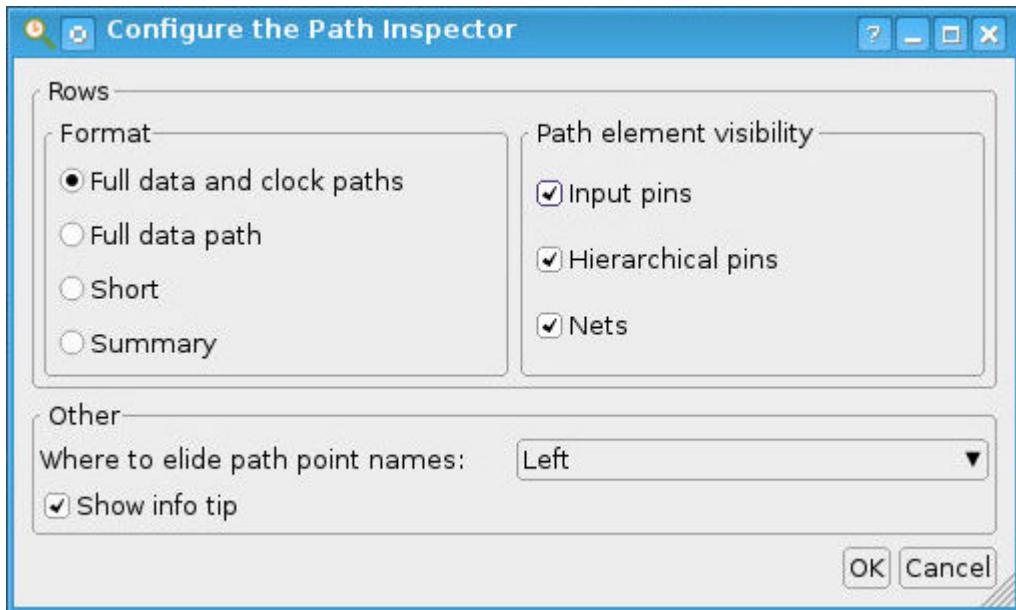
- “Full data and clock paths” to display full data and clock path details, select the option.
- “Full data path” to display full data path details but not clock path details
- “Short” to display only the startpoint and endpoint of the data path.
- “Summary” to hide the path elements and slack panes and display just the report summary.

To configure the path inspector,

1. Click the  button on the path inspector toolbar.

The Configure the Path Inspector dialog box appears.

Figure 339 Configure the Path Inspector Dialog Box



2. Select the desired options.
3. Click OK.

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](#)

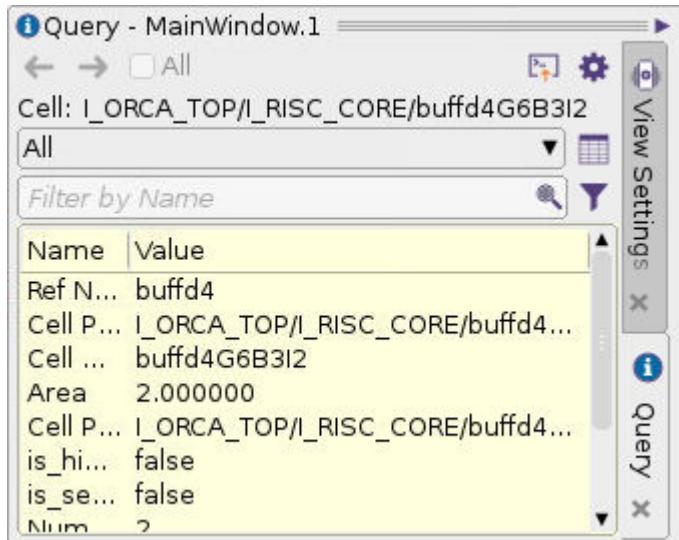
Viewing the Current Selection

To get a summary report on the current selection,

- ▶ Choose Select > Query Selection.

The report appears in a Query window.

Figure 340 Query Window

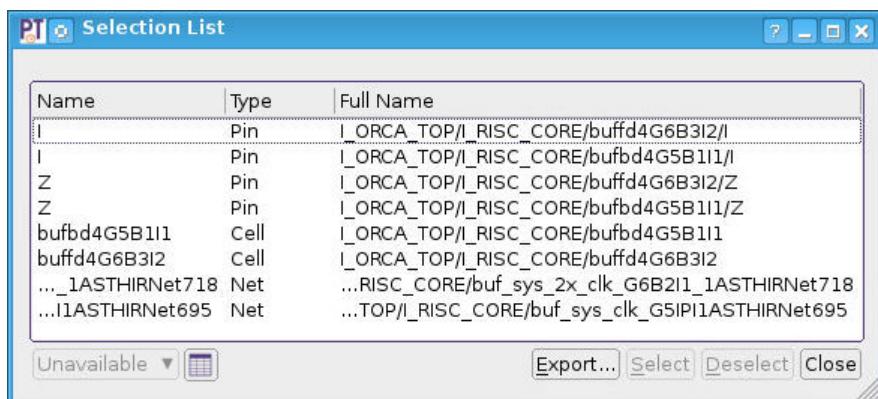


To see a detailed list of the currently selected objects,

- ▶ Choose Select > Selection List.

This opens the Selection List dialog box.

Figure 341 Selection List Dialog Box



You can change the selection by using the Select and Deselect buttons.

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 by using the Export button. The file is formatted with each object or path on a separate line and the column data delimited by commas.

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,

- ▶ 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 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. To set these options, use the gear icon in the upper-right corner.

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](#).

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 View > 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.

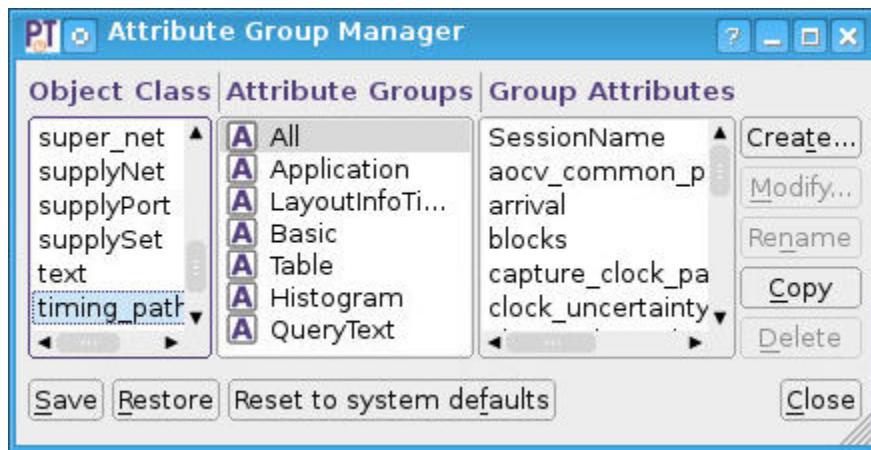
The Properties dialog box lists the properties in a table with two columns: property names and property values. The properties you can view include object names, attribute values, and certain timing and placement values. The list of properties differs depending on the types of objects that you select in the design and the attribute group that you select in the Properties dialog box.

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.

Figure 342 Attribute Group Manager 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.

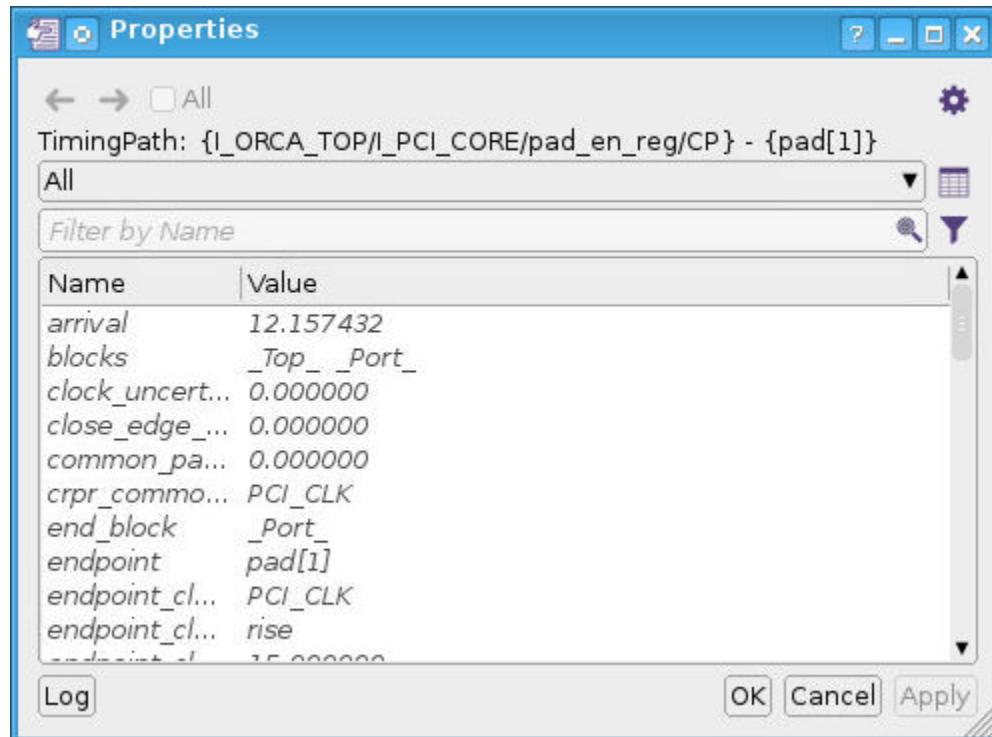
To display the properties for all selected objects,

- ▶ 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 343 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

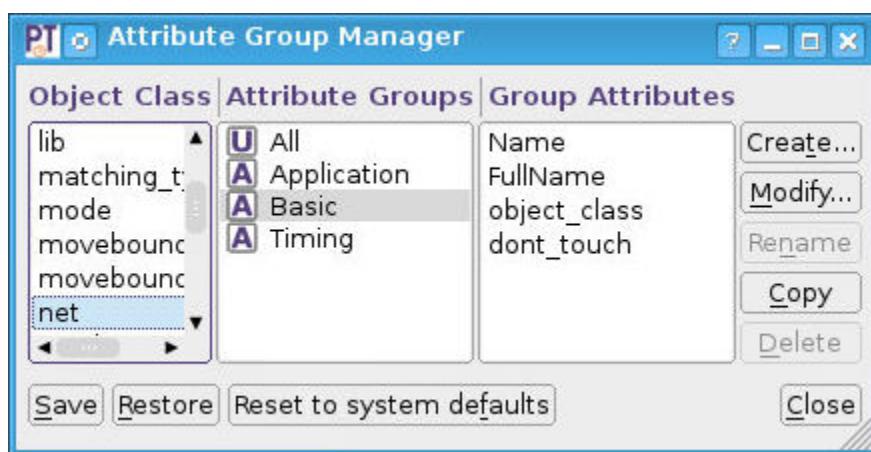
The tool updates the `All` and `User` groups automatically when you create or remove a user-defined attribute.

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.

Figure 344 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](#)

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.

To generate a path collection containing recalculated timing paths, use the `get_timing_paths` command with the `-pba_mode path` option. For example,

```
set my_recalc_path [get_timing_paths -from ... -to ... -pba_mode path]
```

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](#)

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 > Recalculated 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 345 Recalculated Path Table

Endpoint Pin Name	Path Group	Orig. Slack	Recalc. Slack
I ORCA_TOP/I_BLENDER/s4_op1_reg[14]/D	SYS CLK	-0.22268	-0.18439
I ORCA_TOP/I_BLENDER/s4_op1_reg[14]/D	SYS CLK	-0.11878	-0.07776
I ORCA_TOP/I_BLENDER/s4_op1_reg[14]/D	SYS CLK	-0.10284	-0.05438
I ORCA_TOP/I_BLENDER/s4_op1_reg[14]/D	SYS CLK	0.00120	0.04624
I ORCA_TOP/I_BLENDER/s4_op1_reg[30]/D	SYS CLK	-0.66053	-0.65763
I ORCA_TOP/I_BLENDER/s4_op1_reg[30]/D	SYS CLK	-0.65754	-0.64812
I ORCA_TOP/I_BLENDER/s4_op1_reg[30]/D	SYS CLK	-0.64524	-0.64235
I ORCA_TOP/I_BLENDER/s4_op1_reg[30]/D	SYS CLK	-0.64226	-0.63284
pc_be[3]	PCI CLK	-0.20214	-0.20214
I ORCA_TOP/I_BLENDER/s4_op1_reg[13]/D	SYS CLK	0.10760	0.14292
I ORCA_TOP/I_BLENDER/s4_op2_reaf281/D	SYS CLK	-0.32905	-0.32652

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 > 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 346 shows an example of a path pin comparison table.

Figure 346 Path Pin Comparison Table

#	Pin Name	Transition	cal. Transition	Transit. Diff.	Path Delay	cal. Path Delay	Path Delay Diff.	Incremental Del.
1	I_ORCA_T...	1.16982	1.16982	0.00000	3.21649	3.21649	0.00000	0.00000
2	I_ORCA_T...	0.20613	0.20613	0.00000	3.73070	3.73070	-0.00000	0.514...
3	I_ORCA_T...	0.31312	0.31312	0.00000	3.80146	3.80146	-0.00000	0.070...
4	I_ORCA_T...	0.15238	0.15238	0.00000	3.93491	3.93491	-0.00000	0.133...
5	I_ORCA_T...	0.24442	0.24442	0.00000	3.99218	3.99218	-0.00000	0.057...
6	I_ORCA_T...	0.53055	0.53055	0.00000	6.01301	6.01301	0.00000	2.020...
7	I_ORCA_T...	0.66129	0.66129	0.00000	6.13096	6.13096	-0.00000	0.117...
8	I_ORCA_T...	0.28109	0.27727	0.00382	6.50298	6.50298	-0.00000	0.372...
9	I_ORCA_T...	0.31481	0.31145	0.00336	6.54726	6.54726	0.00000	0.044...
10	I_ORCA_T...	0.28840	0.26766	0.02073	6.82325	6.82261	0.00064	0.275...
11	I_ORCA_T...	0.32127	0.30307	0.01820	6.86754	6.86690	0.00064	0.044...
12	I_ORCA_T...	0.28811	0.26716	0.02065	7.11177	7.11061	0.00413	0.277...

The path pin comparison table shows the pin names and compares the original and recalculated pin transition times, path delays, incremental delays, and so on. Positive and negative changes are highlighted in green and red, respectively.

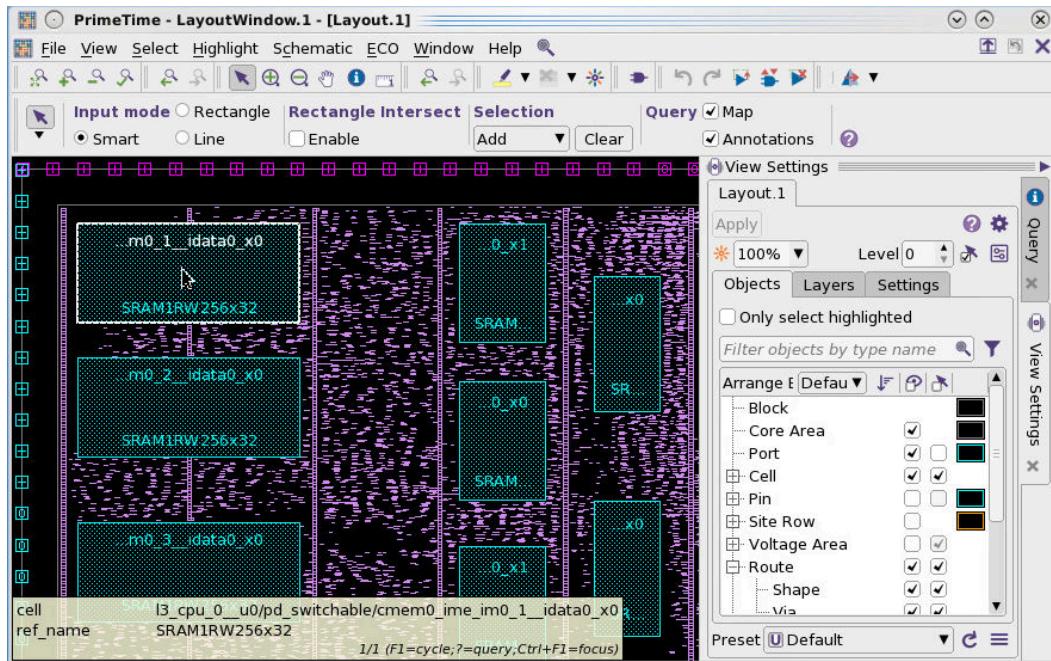
See Also

- [Comparing Normal and Recalculated Paths](#)

Layout View and ECOs in the GUI

To view the chip layout in the GUI in preparation for engineering change orders (ECOs), from the main GUI window, choose Window > Layout window. This opens the layout window, as shown in the following figure.

Figure 347 Layout View in the GUI

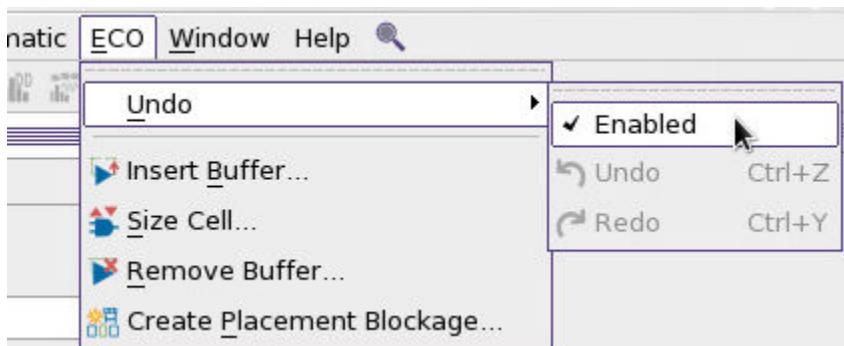


To use the layout view, you need to specify the physical data by using the `set_eco_options` command. The physical data can be in IC Compiler II or LEF/DEF format. To check for the presence of valid physical data, use the `check_eco` command.

You can generate and view ECOs in the GUI, including inserting buffers, removing buffers, and sizing cells. A PrimeTime-ADV-PLUS license is required. After you make a change, you can easily undo it.

To perform an ECO operation in the GUI, use the menu options under ECO, shown in the following figure.

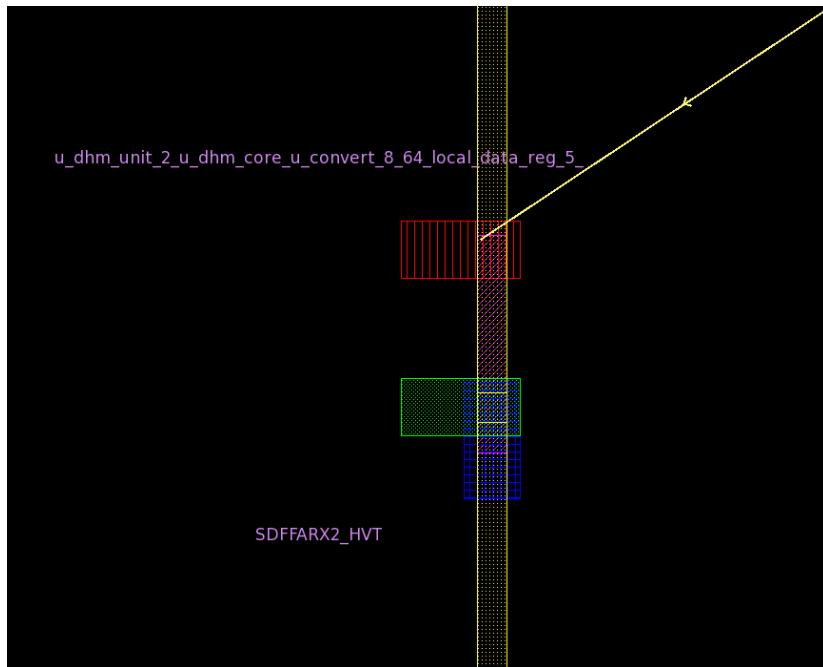
Figure 348 ECO Menu in the GUI



To enable the undo feature, make sure the Enabled option has a check mark. If you do not plan to use the feature, you can disable it to save memory.

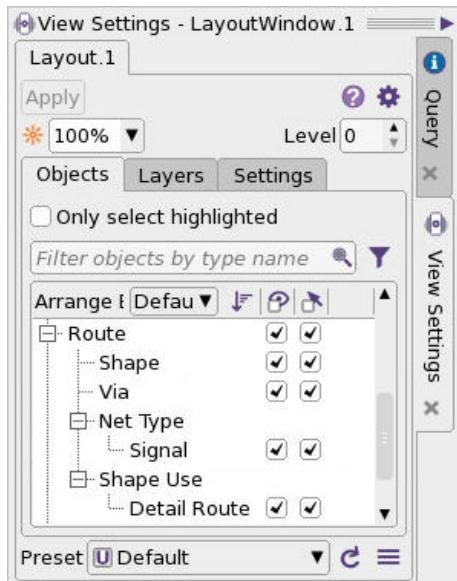
You can highlight a timing path and show only the actual net shapes traversing the path, including relevant vias, without including the unrelated fanout of each driver in the path.

Figure 349 Net Shape Display



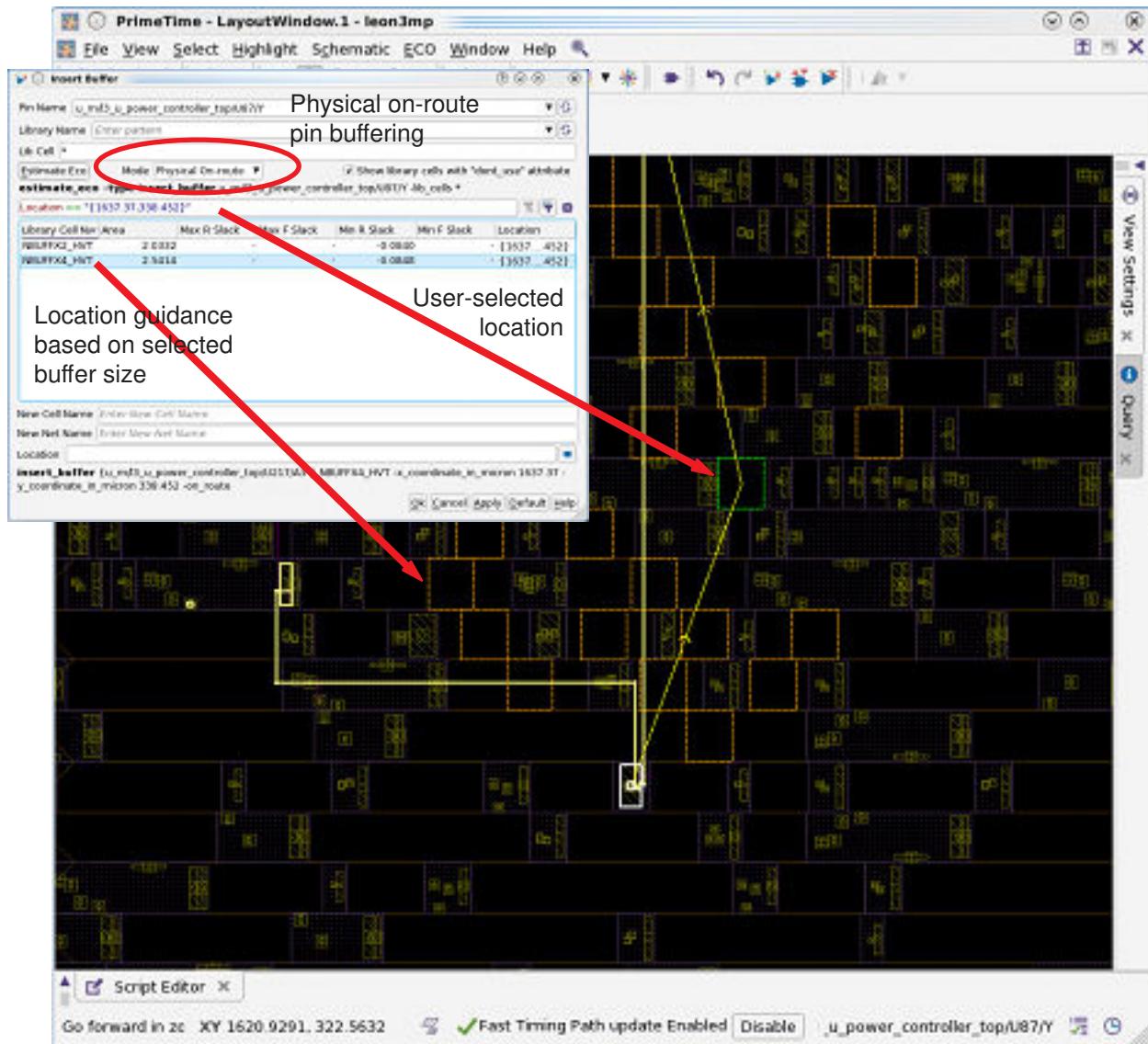
To control the net-shape-only path display mode, use the option settings in the View Settings box under the Objects tab.

Figure 350 Net Shape Display Option Controls



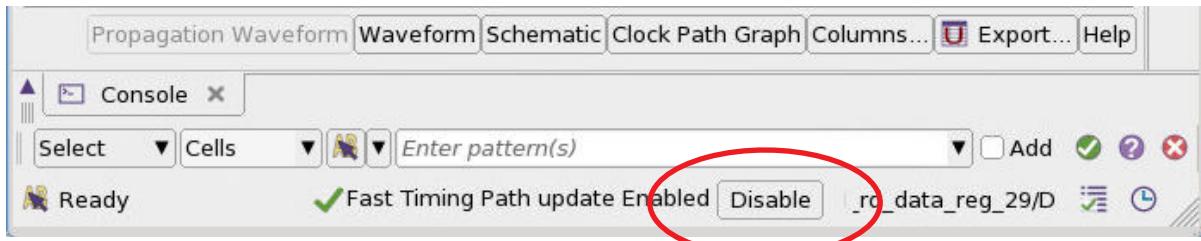
When you perform manual ECO operations using the ECO > Insert Buffer or ECO > Size Cell command (`insert_buffer` or `size_cell`), the tool automatically performs an incremental path-only timing update in memory so that you can immediately determine the timing effects of the change.

Figure 351 Manual ECO Operations in the GUI



Placement blockages are honored during on-route buffer guidance.

To disable the automatic incremental path-only timing update, click the Disable button at the bottom of the GUI window next to the label “Fast Timing Path update Enabled.”



With this feature disabled, you can initiate a regular incremental or full timing update when needed, and using the `estimate_eco` command or querying design attributes triggers a timing update.

To display color-coded cell density maps and pin density maps in the layout window, choose `View > Map > Cell Density` or `View > Map > Pin Density`. A PrimeTime-ADV-PLUS license is required.

Figure 352 View > Map > Cell Density in Layout View

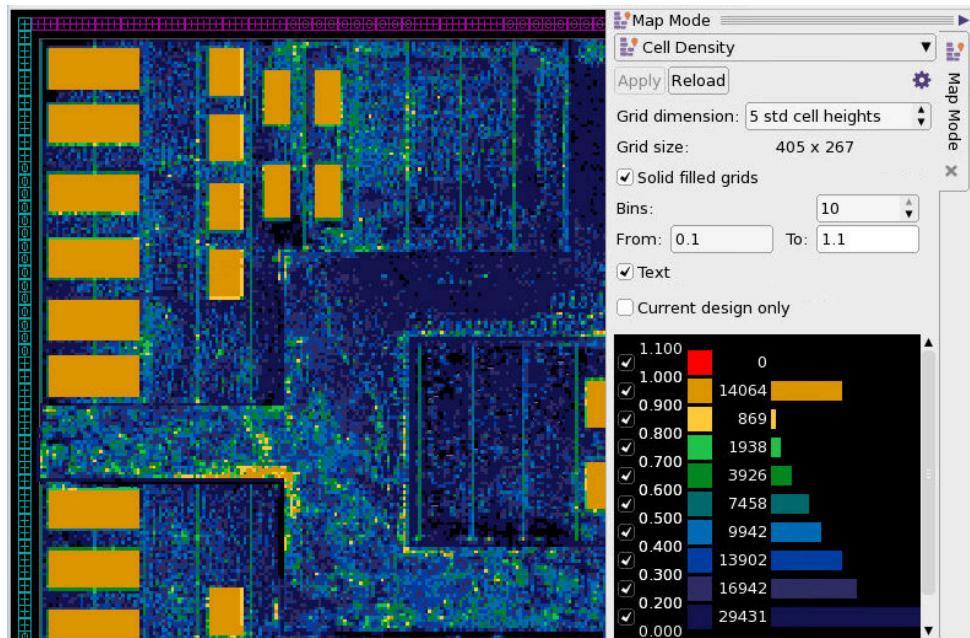
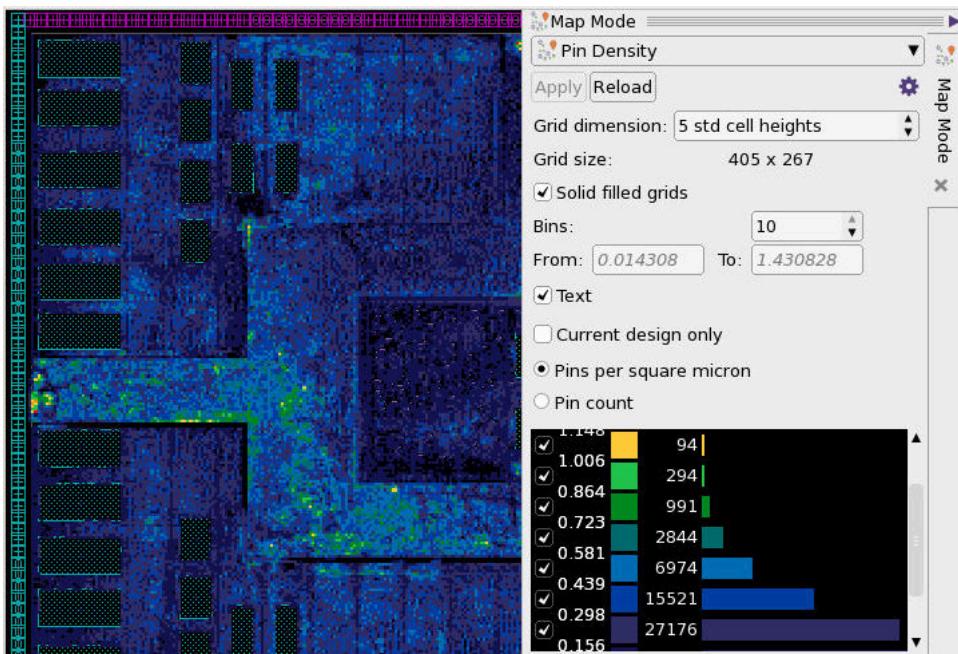
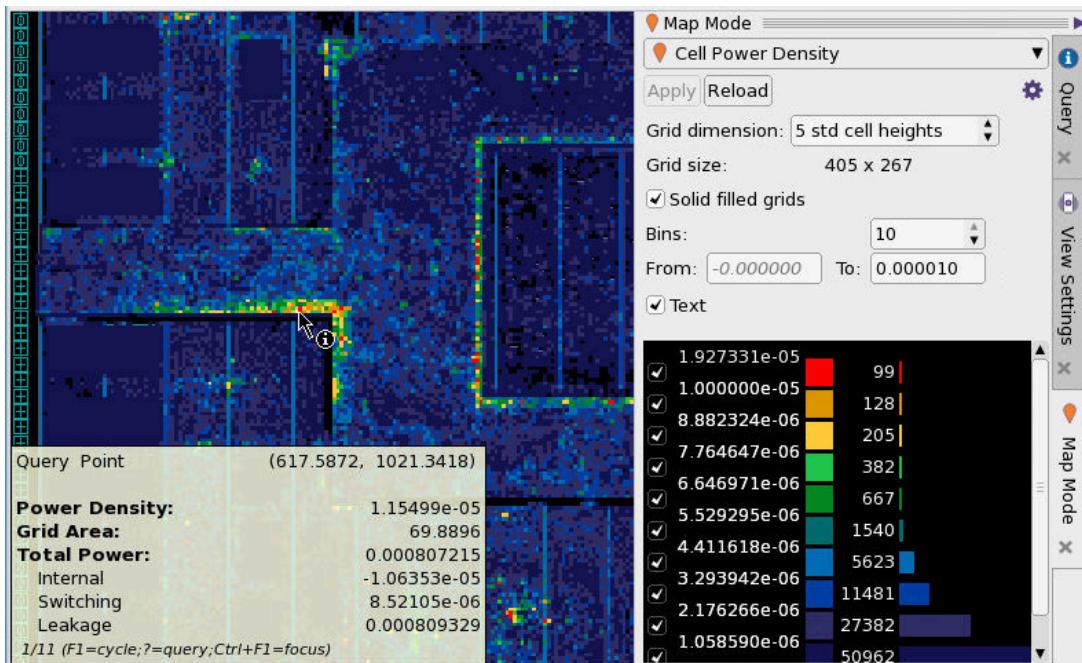


Figure 353 View > Map > Pin Density in Layout View



To display a color-coded power density map in the layout window, choose View > Map > Cell Power Density. Viewing the power map requires PrimePower analysis data generated by the `update_power` or `report_power` command.

Figure 354 View > Map > Cell Power Density in Layout View



Interactive Multi-Scenario Analysis of Timing Paths

The PrimeTime GUI has an interactive multi-scenario analysis mode that lets you restore timing path collections from different runs or scenarios and analyze them together. You can use this feature as a quick analysis tool to help debug and track violations across different analysis runs.

For example, to read in multiple collections of timing paths from different PrimeTime sessions and display and analyze these timing paths simultaneously:

```
# PrimeTime Session 1
...
set my_paths1 [get_timing_paths ...]
save_session -only_timing_paths $my_paths1 /dir/s1

# PrimeTime Session 2
...
set my_paths2 [get_timing_paths ...]
save_session -only_timing_paths $my_paths2 /dir/s2

# PrimeTime multi-scenario analysis session
gui_start
restore_session /dir/s1      # Restore paths saved in session 1
restore_session /dir/s2      # Restore paths saved in session 2
# Display, categorize, and sort paths from multiple sessions in GUI ...
```

The restored path-only sessions must have the same netlist and clocks (but can have different constraints), and the PrimeTime tool version of the “save” and “restore” sessions must match. In the interactive multi-scenario analysis session, the command set is restricted; you can only run commands that analyze the restored path collections.

For more information, see

- [Interactive Multi-Scenario Analysis Flow](#)
- [IMSA Attributes Saved and Restored](#)
- [Using the Path Analyzer Window](#)
- [Shifting the Slack for a Category](#)

Interactive Multi-Scenario Analysis Flow

With the interactive multi-scenario analysis (IMSA) flow, you can

- Load and debug multiple collections of timing paths
- Categorize and sort timing path collections; see [Analyzing Timing Path Collections](#).

- Shift slacks of a category by a specified value for group comparisons; see [Shifting the Slack for a Category](#).
- Add or remove columns of information; see [Inspecting Timing Path Elements](#).

Usage Procedure

The following steps explain the interactive multi-scenario analysis flow in more detail.

- Use the `get_timing_paths` command to create a collection of timing paths objects. For example:

```
pt_shell> set paths1 [get_timing_paths -start_end_pair \
    -slack_lesser_than 0.0]
```

- Save the timing path collection sessions using the `save_session` command with the `-only_timing_paths` option, which saves only the timing paths data:

```
pt_shell> save_session -only_timing_paths $paths1 /dir/s1
```

You can repeat steps 1 and 2 in a session or multiple sessions. Paths from a DMSA session can be saved using the remote execute feature. To save additional paths into an existing saved session, use the `-incremental` option of the `save_session` command.

- In a new PrimeTime session, open the GUI (for example, using `gui_start`).
- Use the `restore_session` command to restore the timing paths from a session or multiple sessions:

```
pt_shell> restore_session /dir/s1
Information: Enter Interactive Multi-Scenario Analysis. (SR-025)
...
pt_shell> restore_session /dir/s2
...
```

Restoring an “only timing paths” session puts the PrimeTime tool in the interactive multi-scenario analysis mode. You can only run commands that analyze the restored path collections, and you cannot change constraints or perform timing updates. The restored sessions must have the same design netlist and clocks, but they can have different constraints.

- Use the path analyzer to categorize and analyze the timing paths. For details, see [Using the Path Analyzer Window](#).

Use schematics, the path inspector, and path data histograms to further analyze the timing paths.

To shift the slack of a category by a user-defined value, see [Shifting the Slack for a Category](#).

To exit the interactive multi-scenario analysis mode and return to a regular PrimeTime session, use the `remove_design -all` command.

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.

IMSA Attributes Saved and Restored

By default, the `save_session -only_timing_paths` command saves only the paths in the specified timing path collection, the attributes of those timing paths and their timing points, and the attributes of the associated clocks; no other attributes are saved.

To save and restore other types of object attributes in sessions, take any one or more of the following actions:

- Set the `imsa_save_reporting_attributes` variable to `true`. This enables saving and restoring a set of attributes to support expanded reporting capabilities in the IMSA session.
- Set the `imsa_save_eco_data` variable to `true`. This enables the saving and restoring of a set of attributes to support physically aware ECO operations in the IMSA session using the PrimeTime GUI, and to restore the session into a linked and timed design.
- Explicitly specify the attributes you want to save and restore by using the `set_imsa_attributes -add` command.

To report the attributes to be saved and restored in IMSA sessions, use the `set_imsa_attributes -report` command. For example,

```
pt_shell> set_app_var imsa_save_eco_data true
true
pt_shell> set_imsa_attributes -class net \
  -add {x_coordinate_min y_coordinate_min}
Information: Added attribute x_coordinate_min of object class net ...
Information: Added attribute y_coordinate_min of object class net ...
```

```
pt_shell> set_imsa_attributes -report
...
Properties:
  A - Application-defined
  U - User-defined
  I - Importable
  R - Values restored per session
```

Class	Attribute Name	Type	Properties
pin	max_rise_slack	float	A
pin	min_rise_slack	float	A
pin	max_fall_slack	float	A

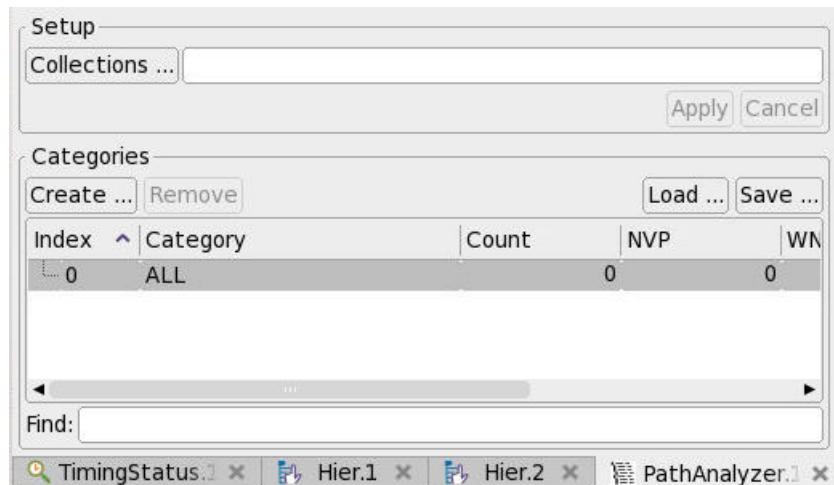
pin	min_fall_slack	float	A
pin	max_capacitance	float	A
pin	max_transition	float	A
pin	max_fanout	float	A
net	x_coordinate_min	float	A
net	y_coordinate_min	float	A

When you save a session with the `save_session -only_timing_paths` command, the command saves the specified paths and also saves the attributes listed in the report. After you restore the IMSA session with the `restore_session` command, you can query the listed attributes.

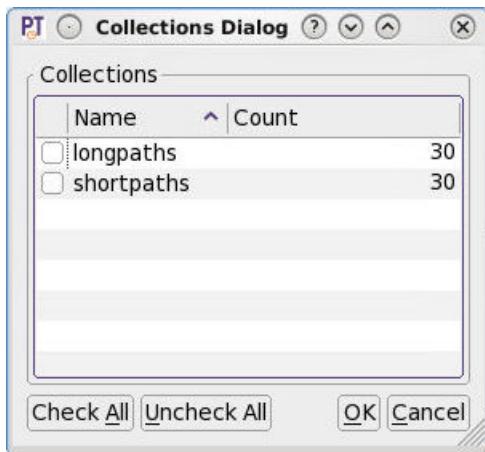
Using the Path Analyzer Window

To view the timing paths in the path analyzer,

1. Choose Timing > Path Analyzer to open a path analyzer window.



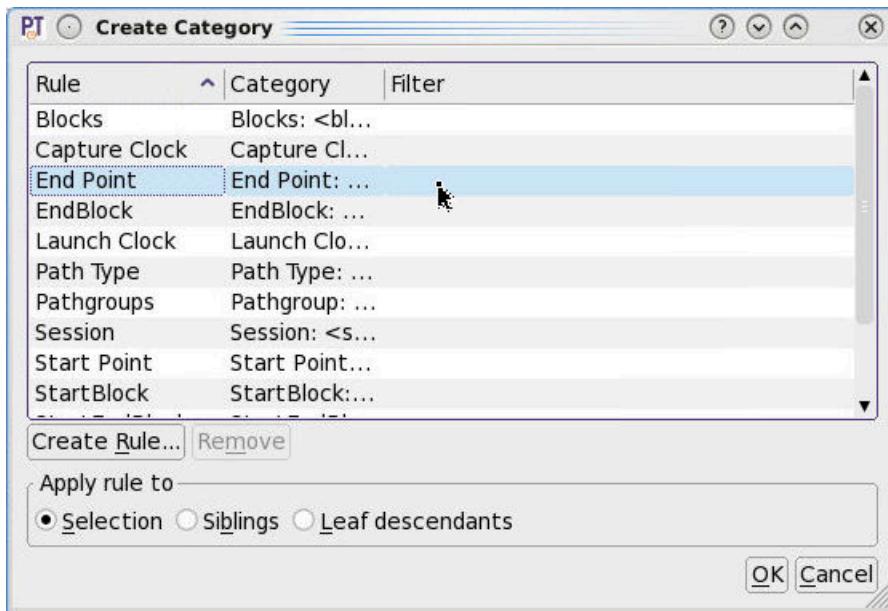
2. Click the Collections button to open the Collections Dialog.



3. Select the collections you want to analyze and click OK. This adds the selected collections to the Setup Collections field.



4. Click Apply.
5. Categorize the timing paths. For example, click the Create button, select a category rule such as "End Point," and click OK.



The Categories box is updated with list of paths sorted according to the category rule.

Setup			
Collections ...	<code> \${longpaths} \${shortpaths}</code>		
	<input type="button" value="Apply"/> <input type="button" value="Cancel"/>		
Categories			
<input type="button" value="Create ..."/> <input type="button" value="Remove"/>	<input type="button" value="Load ..."/> <input type="button" value="Save ..."/>		
Index	Category	Count	NVP
0	ALL	60	60
0	End Point: I_ORCA_TOP	27	27
1	End Point: pad[1]	1	1
2	End Point: pad[13]	1	1
3	End Point: pad[14]	1	1
4	End Point: sd_DQ[0]	3	3
5	End Point: sd_DQ[1]	3	3
6	End Point: sd_DQ[2]	3	3

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.

You can apply existing rules or create and apply custom rules. For more information, see [Shifting the Slack for a Category](#).

For more information about using the path analyzer, see [Analyzing Timing Path Collections](#).

See Also

- [Shifting the Slack for a Category](#)
- [Interactive Multi-Scenario Analysis Flow](#)

Shifting the Slack for a Category

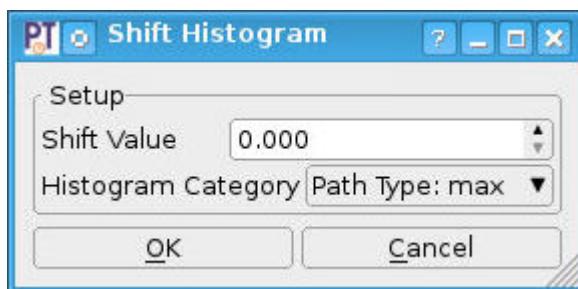
In the interactive multi-scenario analysis flow, you can shift the slack of a category by a user-defined value.

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.

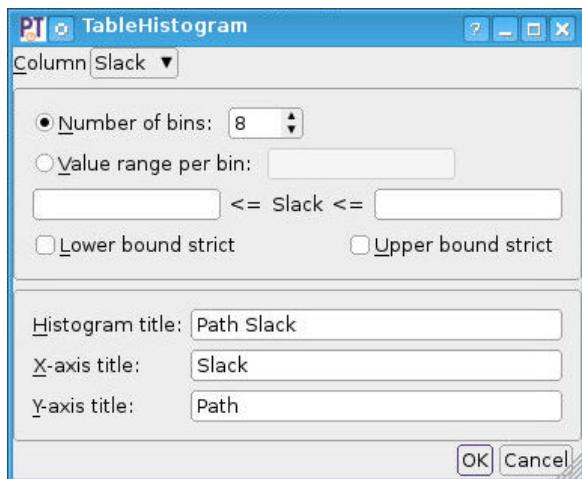
Figure 355 Shift Histogram dialog box



3. Select the slack shift value and parent category of the histogram.
4. Click OK.

The Table Histogram dialog box appears.

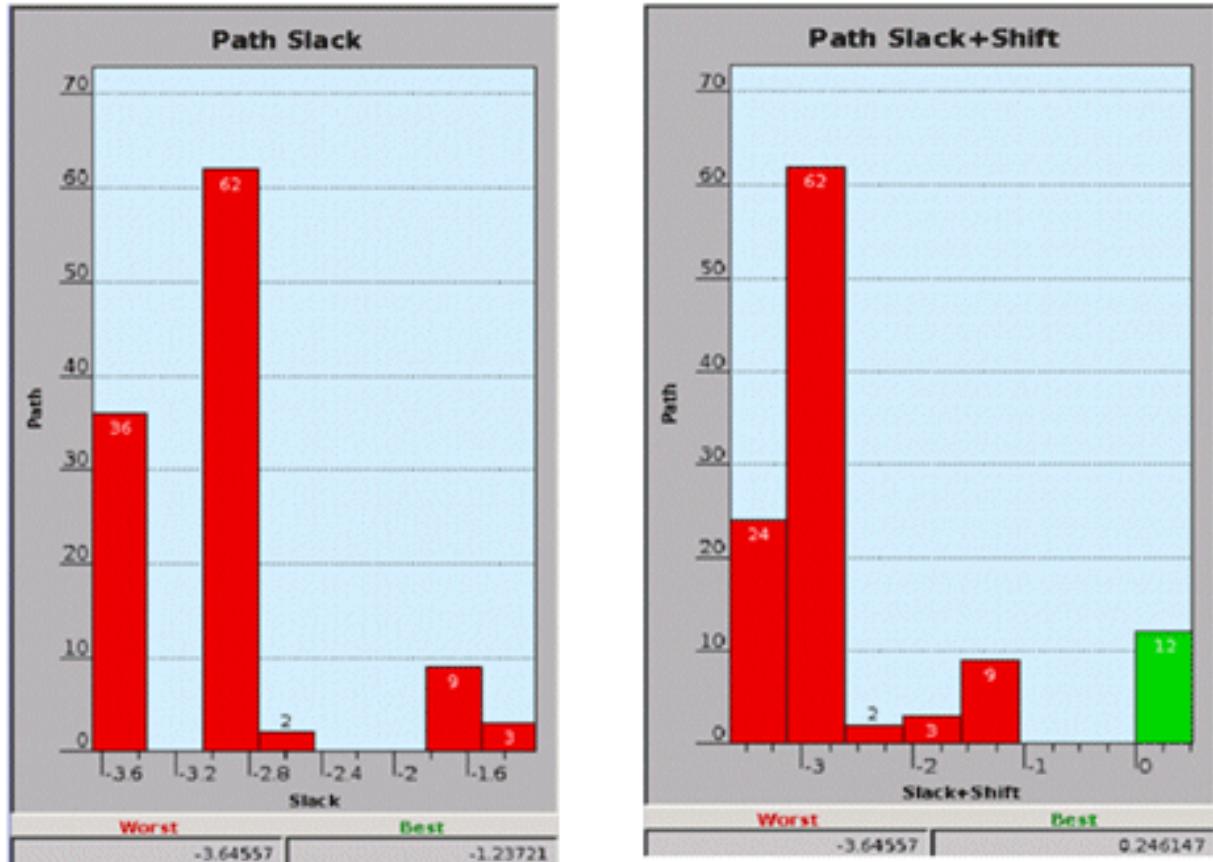
Figure 356 Table Histogram dialog box



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.

Figure 357 Path Slack and Path Slack+Shift



6. Click OK.

See Also

- [Using the Path Analyzer Window](#)
- [Interactive Multi-Scenario Analysis Flow](#)

GUI Tcl Commands

The PrimeTime tool supports a set of Tcl commands that control and query the GUI windows. You can use these commands to write scripts that control the appearance and contents of the GUI windows. For example, you can

- Create menu commands
- Create toolbars
- Add command buttons
- Add buttons to the Favorites palette
- Set object colors
- Add color highlighting and annotations in the layout window

The GUI control capabilities are similar to those in the IC Compiler II tool. For more information, see “Extending the Graphical User Interface” in the *IC Compiler II Graphical User Interface User Guide* on [SolvNetPlus](#).

For a list of the GUI-related commands, get help on them as follows:

```
pt_shell> help gui_*
gui_change_highlight # Change the global highlight state.
gui_create_attrgroup # Creates gui attribute group
gui_create_pref_category # Create a preference category
gui_create_pref_key # Create a preference key-value pair
gui_create_vm # Create new Visual Mode
...
```

To get help on a particular command, view its man page:

```
pt_shell> man gui_change_highlight
...
NAME
    gui_change_highlight
        Manipulate the set of globally highlighted objects.
SYNTAX
    ...
ARGUMENTS
    ...
```

21

ECO Flow

If your design has timing, design rule, or noise violations, or you want to optimize area or power, you can use the engineering change order (ECO) flow to fix the violations, implement the changes, and rerun timing analysis.

- [ECO Fixing Overview](#)
 - [Setting the ECO Options](#)
 - [Physically Aware ECO](#)
 - [ECO Fixing Methods](#)
 - [Reporting Unfixable Violations and Unusable Cells](#)
 - [HyperTrace ECO Fixing](#)
 - [Writing Change Lists](#)
 - [Implementing ECO Changes in Synopsys Layout Tools](#)
 - [Incremental ECO Flow Using Synopsys Tools](#)
 - [ECO Flow Using Reduced Resources](#)
 - [Freeze Silicon ECO Flow](#)
 - [Running ECO Scenarios on Fewer Hosts](#)
 - [Manual Netlist Editing](#)
-

ECO Fixing Overview

In the PrimeTime tool, an engineering change order (ECO) is an incremental change in a chip design to fix timing violations or design rule constraint (DRC) violations, or to reduce power. The PrimeTime tool finds these design issues and corrects them by sizing cells, replacing cells, or inserting buffers, and it writes out the changes in script format so that you can implement the changes in other tools.

The commands to perform ECO fixing are:

```
fix_eco_drc
fix_eco_timing
fix_eco_power
```

After ECO fixing is complete, to implement the changes, use the `write_changes` command to write out the changes, and run the script in the physical implementation tool such as IC Compiler II. After the changes are implemented, you should perform parasitic extraction and run timing analysis again in the PrimeTime tool.

In the PrimeTime tool, you can perform ECO fixing with or without physical placement data:

- Logic-only mode – The tool uses only the design netlist and detailed parasitic data, without considering physical placement data.
- Physically aware mode – The tool uses the design netlist, parasitic data, and physical placement data. It replaces cells and inserts buffers only where there is room to make the changes, and the `write_changes` command writes out the location of each change for fast and accurate physical implementation.

The ECO fixing flow is compatible with single-core analysis, multicore analysis, and distributed multi-scenario analysis (DMSA). To reduce memory, runtime, and turnaround time, use the following ECO flow options:

- Reduced resource ECO flow – In the PrimeTime tool, you perform timing analysis on the whole design but perform ECO on a smaller “reduced” design. The reduced design contains only the design data related to the constraint violations that need to be fixed.
- Synopsys incremental ECO flow – You implement the changes using Synopsys tools (IC Compiler II, StarRC, and PrimeTime), and write out and analyze only the changed part of the design at each step instead of the full design.

Before you attempt ECO fixing, ensure that the design is fully placed and routed, with generated clock trees, and use physically aware ECO to ensure the best quality of results.

See Also

- [Setting the ECO Options](#)
- [DRC, Crosstalk, and Cell Electromigration Violation Fixing](#)
- [Timing Violation Fixing](#)
- [Power Recovery Fixing](#)
- [Order of ECO Fixing Steps](#)

- [ECO Flow Using Reduced Resources](#)
- [Incremental ECO Flow Using Synopsys Tools](#)

DRC, Crosstalk, and Cell Electromigration Violation Fixing

To fix design rule constraint (DRC), noise, crosstalk delay, or cell electromigration violations, use the `fix_eco_drc` command. It fixes the violations reported by the following commands:

```
report_constraint -max_capacitance ...
report_constraint -max_transition ...
report_constraint -max_fanout ...
report_noise ...
report_si_bottleneck ...
report_cell_emViolation ...
```

The `fix_eco_drc` command attempts to fix violations by sizing cells and inserting buffers or inverter pairs, while minimizing the impact on area. By default, it does not consider timing effects.

After the violations are fixed, write out the design changes as a script by using the `write_changes` command and use it to implement the changes in another tool. A list of design changes created in this flow is called an engineering change order (ECO).

The `fix_eco_drc` command has options to specify the following parameters:

- The type of violation to fix and the target amount of slack to achieve
- The scope of the design to fix (a list of pins or ports, or the whole design)
- The fixing methods (cell sizing, buffer insertion, or inverter pair insertion)
- The list of library cells to use for buffer or inverter pair insertion
- Whether to modify cells in data paths or clock networks
- Whether to consider timing constraints during fixing, and if so, the required setup and hold timing margins
- The physical placement and sizing mode (none, open site, occupied site, or freeze silicon)

For example, the following command fixes maximum transition time design rule violations using cell sizing:

```
pt_shell> fix_eco_drc -type max_transition -methods {size_cell}
```

The following command fixes noise violations using both cell sizing and buffer insertion.

```
pt_shell> fix_eco_drc -type noise \
    -methods {size_cell insert_buffer} \
    -buffer_list {BUFX1 BUFX2 BUFX3} \
    -physical_mode open_site
```

The command specifies the list of library cell buffers to use and selects the “open site” physically aware fixing mode, which restricts cell sizing and buffer insertion to occur only where there is already enough room for the change.

The `fix_eco_drc` command performs multiple fixing iterations until all violations are fixed or it determines that further fixing is not worth the runtime cost, based on the current quality of results and fixing option settings.

Upon completion, the `fix_eco_drc` command shows the remaining violations and the number of unfixable violations. To generate a report on the reasons for the unfixable violations, use the `-verbose` option.

Crosstalk Delta Delay Fixing

The `fix_eco_drc` command supports ECO reduction of crosstalk delta delays on victim nets. For example,

```
pt_shell> fix_eco_drc -type delta_delay -verbose \
    -delta_delay_threshold 0.05 -methods {size_cell insert_buffer} \
    -buffer_list {BUFX1 BUFX2 BUFX3} -physical_mode open_site
```

This command performs cell sizing and buffer insertion on victim nets to reduce crosstalk delays. It targets all victim nets that have a delta delay cost exceeding 0.05 time units and that belong to a timing path with negative slack.

Because there is no constraint that limits the allowable delta delay and therefore no “delta delay slack,” you must specify a delta delay threshold for fixing. Specify the threshold as a positive value greater than zero. The same threshold applies to both slowdown and speedup delta delays.

Using this feature requires a PrimeTime-ADV-PLUS license.

Delta Delay Fixing and Path Slack

By default, delta delay fixing considers only the nets that belong to paths with a negative timing slack. You can change this behavior by using the `-slack_lesser_than` option of the `fix_eco_drc` command:

```
pt_shell> fix_eco_drc -type delta_delay -verbose \
    -delta_delay_threshold 0.05 -methods {size_cell insert_buffer} \
    -buffer_list {BUFX1 BUFX2 BUFX3} -physical_mode open_site \
    -slack_lesser_than 0.05
```

The command targets nets for fixing that belong to paths with a slack less than the specified value. Specify a positive value to achieve a positive timing margin, or a negative value to perform less fixing.

SI Bottleneck Optimization and Reporting

The `fix_eco_drc -type delta_delay` command chooses the nets to fix based on SI bottleneck analysis. To preview the nets targeted for fixing, run the `report_si_bottleneck` command first. For example,

```
pt_shell> report_si_bottleneck -cost_type delta_delay \
    -min -max -slack_lesser_than 0.0 -all_nets
...
Bottleneck Cost: delta_delay
net           scenario      cost
-----
DX23[17]       scen1        0.4597
REG_ADDR[2]    scen2        0.3671
REG_DATA[5]    scen1        0.2921
RE_RDY         scen1        0.0934
...

```

This command reports the highest-cost nets first, considering both max and min analysis types together. The “cost” value in the report shows the absolute value of the delta delay on the net (negative delta delays are shown as positive). For a DMSA analysis, the report shows the merged results across all scenarios.

You can use the `report_si_bottleneck` command to preview and check the results of delta delay fixing in the same way that you use `report_constraints` for DRC fixing or `report_timing` for timing fixing.

Cell Electromigration Violation Fixing

The `fix_eco_drc` command can target the cell-internal electromigration DRC violations that are reported by the `report_cell_emViolation` command.

Cell electromigration effects occur inside logic gates when excessive current densities cause a gradual displacement of metal atoms, which can eventually cause a short or open in the metal structure. The PrimePower tool uses a combination of library data, design data, and switching activity data (and physical data, if available) to identify and report high-risk cells as electromigration DRC violations.

Cell electromigration analysis is performed by the PrimePower tool. For details, see the “Cell Electromigration Analysis” chapter of the *PrimePower User Guide*. This feature requires a PrimePower license and a PrimeTime-ADV-PLUS license.

To use this feature, configure the cell electromigration analysis and run the `update_power` command, then target the `cell_em` fixing type of the `fix_eco_drc` command:

```
# perform power/electromigration analysis
set_app_var power_enable_analysis true
set_app_var power_enable_em_analysis true
update_power

# report cell EM violations
report_cell_emViolation

# fix cell EM violations
fix_eco_drc -type cell_em -methods {size_cell insert_buffer}
```

The `cell_em` fixing type can use the sizing and buffering methods to fix cell electromigration violations.

The `cell_em` fixing type supports only a single iteration. To fix additional violations, run the `update_power` command, then rerun the `fix_eco_drc -type cell_em` command again.

Timing Violation Fixing

To fix setup or hold timing violations, use the `fix_eco_timing` command, which reduces timing violations by sizing cells and inserting buffers or inverter pairs. It attempts to fix the specified types of violations while minimizing the impact on area and power.

After the violations are fixed, write out the design changes as a script by using the `write_changes` command and use it to implement the changes in another tool. A list of design changes created in this flow is called an engineering change order (ECO).

The `fix_eco_timing` command has options to specify the following fixing parameters:

- The type of timing violations to fix (setup or hold)
- The scope of the design to fix (from/to startpoints/endpoints, path groups, a path collection, or the whole design)
- The fixing methods (cell sizing, buffer insertion, or inverter pair insertion)
- The list of library cells to use for buffer or inverter pair insertion
- Whether to modify cells in data paths or clock networks
- The target amount of timing margin (slack) to achieve
- The graph-based analysis mode (graph-based, path-based, or path-based exhaustive)
- The physical placement and sizing mode (none, open site, occupied site, or freeze silicon)
- Whether to minimize power while fixing timing

For example, the following command fixes setup violations throughout the design:

```
pt_shell> fix_eco_timing -type setup
```

The following command fixes hold violations throughout the design using both cell sizing and buffer insertion, and using exhaustive path-based analysis for reduced timing pessimism:

```
pt_shell> fix_eco_timing -type hold -pba_mode exhaustive \
           -buffer_list {BUFX2 DLY1X2 DLY2X2}
```

Each time you use the `fix_eco_timing` command, you must specify the fixing type, either setup or hold, because of the different nature of these violations. By default, setup fixing uses cell sizing alone to reduce data path delays, whereas hold fixing uses both cell sizing and buffer insertion to increase data path delays. By default, fixing occurs only in data paths, not in clock networks.

The command performs multiple fixing iterations, starting with the worst violations. It repeats fixing iterations until all violations are fixed or it determines that further fixing is not worth the runtime cost, based on the current quality of results and fixing option settings.

Setup fixing seeks to avoid introducing design rule checking (DRC) violations, but it is allowed to introduce hold violations because setup violations are harder to fix. Hold fixing seeks to avoid introducing both setup and DRC violations.

Upon completion, the `fix_eco_timing` command shows the remaining violations and the number of unfixable violations. To generate a report on the reasons for the unfixable violations, use the `-verbose` or `-estimate_unfixable_reasons` option.

Power Recovery Fixing

The `fix_eco_power` command tries to recover power and area as much as possible by downsizing cells in paths with positive setup slack or by removing buffers from paths with positive hold slack, without introducing or worsening timing and DRC violations.

After power recovery, write out the design changes as a script by using the `write_changes` command and use it to implement the changes in another tool. A list of design changes created in this flow is called an engineering change order (ECO).

By default, the `fix_eco_power` command performs power and area recovery by downsizing cells in all paths with positive setup slack. The command options let you specify any one of the following power recovery methods:

- Replace cells to minimize area (the default)
- Replace cells to minimize a library cell numeric attribute
- Replace cells based on library cell preference, as specified by an explicit string priority list

- Replace cells to minimize switching, leakage, or total power using power analysis data generated by the PrimePower tool (`update_power` command)
- Remove buffers from paths with positive hold slack

Power Recovery Examples

The following command recovers power by downsizing both sequential and combinational cells, without introducing new timing violations:

```
pt_shell> fix_eco_power
```

The following command recovers leakage power by swapping out cells with higher leakage and replacing them with cells that have lower leakage, using the library cell name prefixes to determine which cells are preferred.

```
pt_shell> fix_eco_power -pattern_priority {HVT MVT LVT}
```

The following command recovers area and power by downsizing cells and analyzing the timing effects using path-based analysis.

```
pt_shell> fix_eco_power -pba_mode path
```

The following command recovers leakage power by replacing low-threshold-voltage cells with high-threshold-voltage cells, based on the library cell name prefixes “HVT” and “LVT.”

```
pt_shell> fix_eco_power -pattern_priority {HVT LVT} -pba_mode exhaustive
```

The following command removes buffers in paths with positive hold slack.

```
pt_shell> fix_eco_power -methods {remove_buffer}
```

You can apply multiple fixing methods by running the `fix_eco_power` command repeatedly with different option settings.

Power Recovery Operation

Power recovery is an iterative process. The `fix_eco_power` command starts by making the targeted changes and then analyzes the design for new timing and DRC violations (or worsening of existing violations) caused by the changes. It incrementally backs out of previous changes to meet the timing and DRC constraints, and also explores further changes for improved power recovery. It analyzes the design again and repeats this process until it determines that the cost of further progress exceeds the potential benefit.

For all of the cell replacement methods, replacement occurs only when the following conditions are met:

- The change does not introduce or worsen any timing violations or DRC violations
- The replacement cell has the same logical function as the original cell and meets any usage restrictions defined by the `eco_alternative_cell_attribute_restrictions` and/or `eco_alternative_cell_instance_based_restrictions` variables
- The replacement cell is preferred over the original cell in one of the following ways, depending on the option settings: less area, smaller power attribute value, higher-priority name or attribute string, or less power based on PrimePower analysis data

For more information, see [Power Recovery](#).

Order of ECO Fixing Steps

The PrimeTime ECO commands and command options let you perform different types of ECOs. When deciding on the order in which to fix design rule constraint (DRC), setup, and hold violations, consider the status of your design and the ECO fixing goals.

Setup or hold fixing does not degrade DRC (`max_capacitance` and `max_transition`) violations, but DRC fixing can degrade setup or hold violations because DRC fixing has the highest priority. Also, hold fixing preserves setup slack, but setup fixing is allowed to introduce some hold violations, because setup is a harder problem to fix. Therefore, it is a good idea to fix DRC violations first, then setup violations, and finally hold violations.

If the number of changes is high and the changes disturb ECO route and legalization in IC Compiler or IC Compiler II, setup, hold, and DRC fixing can be done one at a time to reduce the disturbances that might occur in one pass. Therefore, you should first fix violations that cause more changes during ECO fixing.

To prioritize the fixing of timing violations over design rule violations, run the `fix_eco_timing` command with the `-ignore_drc` option. If you use this option, the command ignores and can potentially degrade the `max_transition`, `max_capacitance`, and `max_fanout` design rule violations. By default, the `fix_eco_timing` command does not fix timing violations on paths with design rule violations.

The following table summarizes the recommended flow for PrimeTime ECO fixing.

Table 58 Recommended Order of PrimeTime ECO Fixing

Steps	Commands	Fixing mechanisms	Precedence rules
Step1: Power recovery	<code>fix_eco_power</code>	Cell sizing, buffer removal	Does not introduce new timing or DRC violations

Table 58 Recommended Order of PrimeTime ECO Fixing (Continued)

Steps	Commands	Fixing mechanisms	Precedence rules
Step 2: Fix DRC and noise violations	<code>fix_eco_drc</code>	Buffer insertion, cell sizing	Alters setup and hold slacks as needed
Step 3: Fix timing violations	<code>fix_eco_timing</code>	Buffer insertion, cell sizing	Setup fixing honors DRC and alters hold slack if needed; hold fixing honors setup slack and DRC
Step 4: Final leakage recovery	<code>fix_eco_power</code>	Threshold voltage swapping	Does not introduce new timing or DRC violations; does not change layout

See Also

- [DRC, Crosstalk, and Cell Electromigration Violation Fixing](#)
- [Timing Violation Fixing](#)
- [Power Recovery Fixing](#)
- [Setting the ECO Options](#)

Setting the ECO Options

Before you run any ECO fixing commands, follow these steps to set the ECO options:

1. Import the design into PrimeTime by reading the Verilog, SDC, and parasitic data files.
2. Set the `timing_save_pin_arrival_and_slack` variable to `true`.
3. Perform a full timing update (`update_timing` or `report_timing`).
4. Set the ECO variables that are relevant for your design or analysis conditions.

Table 59 Commonly Used ECO Variables

Condition	Relevant variables
physically aware ECO	<code>eco_allow_filler_cells_as_open_sites</code> <code>eco_insert_buffer_search_distance_in_site_rows</code>
Design contains multiply instantiated modules (MIMs)	<code>eco_enable_mim</code>
Multi-scenario ECO with more scenarios than hosts	<code>eco_enable_more_scenarios_than_hosts</code>

For a complete list of all variables related to ECO, including the current and default settings, run this command:

```
pt_shell> report_app_var eco*
```

5. Run the `set_eco_options` command with the relevant options for your design.

Table 60 Commonly Used Options for the set_eco_options Command

To specify this information...	Use these <code>set_eco_options</code> command options
Data for physically aware ECO	<ul style="list-style-type: none"> -physical_tech_lib_path -physical_lib_path -physical_design_path -physical_icc2_lib -physical_icc2_blocks -physical_constraint_file -physical_lib_constraint_file
Handling of physical data	<ul style="list-style-type: none"> -physical_enable_clock_data -physical_enable_all_vias -keep_site_names -power_ground_net_layer -enable_pin_color_alignment_check -allow_missing_lef -convert_sites -enable_via0_alignment_check
Multiply instantiated module (MIM) groups	-mim_group
Filler cells	-filler_cell_names
Spare cells (freeze silicon flow)	-programmable_spare_cell_names
Additional setup or hold timing margin	<ul style="list-style-type: none"> -drc_setup_margin -drc_hold_margin -timing_setup_margin -timing_hold_margin -power_setup_margin -power_hold_margin
Output log file	-log_file

To verify the ECO options, run the `report_eco_options` command. To reset the ECO options, run 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.

Globally Excluding Specific Library Cells (`dont_use`)

To globally exclude specific library cells from being used for ECO, set the `dont_use` attribute on those library cells by running the `set_dont_use` command:

```
set_dont_use lib_cells [true | false]
```

Excluding Library Cells on a Per-Cell-Instance Basis (`dont_use`)

To exclude library cells from ECO use on specific parts of the design, use the `set_target_library_subset` command:

```
set_target_library_subset
  -objects objects | -top
  -dont_use lib_cells
```

These commands are similar to the same-named commands in the synthesis tools, except that the subset specifications support only the `-dont_use` option.

Subset specifications can be applied to hierarchical cells and/or the top-level design. Specifications propagate downward implicitly. A specification explicitly applied at a lower level completely overrides any specifications applied at a higher level. An empty library cell list (`-dont_use {}`) blocks restrictions applied from a higher level.

For example, the following commands disallow HVT cells throughout the design, except for the SLOWBLK hierarchical cell where LVT cells are disallowed instead:

```
pt_shell> set_target_library_subset -top -dont_use */*HVT*
pt_shell> set_target_library_subset -objects {SLOWBLK} -dont_use */*LVT*
```

Subset specifications restrict only the `fix_eco_*` commands. They take precedence over any `-buffer_list` specifications. In addition, they apply only to the logic being optimized; existing unoptimized logic is unaffected.

Preserving Parts of the Design (`dont_touch`)

To preserve parts of the design that were previously optimized, apply the `dont_touch` attribute to those parts. The PrimeTime ECO fixing commands do not make changes on objects that have the `dont_touch` attribute set to `true`.

Apply the `dont_touch` attribute by using the following commands.

Command	Description
set_dont_touch	Applies the <code>dont_touch</code> attribute to specific cells, nets, designs, or library cells, preventing those objects from being changed.
set_size_only	Applies the <code>size_only</code> attribute to specific cells, preventing them from being deleted while allowing them to be sized.
set_dont_touch_network	Applies the <code>dont_touch</code> attribute on the transitive fanout of specific pins, ports, or clocks; recursively applies the attribute on all nets and combinational cells but stops at cells with setup or hold constraints, such as registers and latches.

To include reporting of the `dont_touch` attribute and similar attributes in timing reports, set the `timing_report_include_eco_attributes` variable:

```
pt_shell> set_app_var timing_report_include_eco_attributes true
true
pt_shell> report_timing ...
...
Attributes:
  d - dont_touch
  u - dont_use
  i - ideal_net
...
```

Point	Incr	Path	Attributes
clock clk (rise edge)	0.00	0.00	
clock network delay (propagated)	3.30	3.30	
A1/B1/reg_out_reg_11_/CP (dfn)	0.00	3.30 r	
A1/B1/reg_out_reg_11/_Q (dfn)	0.87	4.18 f i	
icc_route_opt6/ZN (inv7)	0.00	4.18 r i d u	
icc_route_opt8/ZN (inv7)	0.07	4.25 f d u	
reg_out[11] (out)	0.01	4.25 f	
data arrival time		4.25	

Resizing Power Management Cells

By default, the ECO commands `fix_eco_timing`, `fix_eco_power`, and `fix_eco_drc` do not consider power management cells for possible resizing: level shifters, isolation cells, retention registers, and always-on cells.

To allow the tool to consider these power management cells for resizing during ECO generation, and to specify which types of cells to consider for resizing, set the `eco_allow_sizing_with_lib_cell_attributes` variable. For example,

```
pt_shell> set_app_var \
  eco_allow_sizing_with_lib_cell_attributes "is_level_shifter"
```

You can set the variable to a list of one or more of the following strings:

- `is_level_shifter` – Consider level shifters for resizing during ECO
- `is_isolation` – Consider isolation cells for resizing during ECO
- `is_retention` – Consider retention registers for resizing during ECO
- `always_on` – Consider always-on cells for resizing during ECO

When a library cell's attribute of the specified name (such as `is_isolation`) is set to `true`, that cell is considered for resizing.

Physically Aware ECO

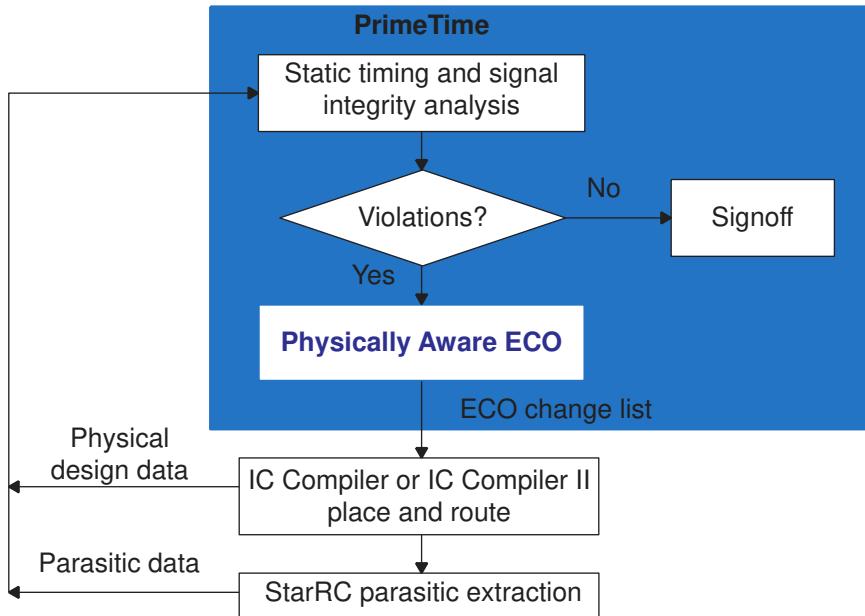
The basic logic-only PrimeTime ECO flow fixes design rule and timing violations without considering physical design information, whereas the physically aware ECO guidance flow considers physical information during fixing, providing the following benefits:

- Improves ECO fixing rates and predictability
- Writes out placement locations for 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

Using the physically aware ECO flow requires a PrimeTime-ADV license.

The following flow diagram shows the overall ECO flow.

Figure 358 Synopsys Physically Aware ECO Flow



For physically aware ECO fixing in PrimeTime, you need these files:

- [Design Data Files](#) – Design netlist, constraints, and timing information.
- [Block-Level LEF Library Data](#) – Library technology and physical cell information
- [DEF Files or IC Compiler II Database Files](#) – Physical layout information of the design
- [Physical Constraint File](#) – Voltage areas and placement blockages in multivoltage designs (optional)
- [Advanced Spacing Labels and Rules](#) – Spacing labels and rules (optional).
- [Parasitic Data Files From StarRC](#) – Parasitic data with location information.

After you prepare these files and set the ECO options ([Setting the ECO Options](#)), you can perform physically aware ECO fixing ([Performing Physically Aware ECO](#)).

Performing Physically Aware ECO

To perform physically aware ECO fixing:

1. Enable the reading of parasitics with location data by setting the `read_parasitics_load_locations` variable:

```
pt_shell> set_app_var read_parasitics_load_locations true
```

2. Set the physically aware ECO options:

- To enable the consideration of filler cells as open sites:

```
set_app_var eco_allow_filler_cells_as_open_sites true
```

- To specify the area in which the tool searches for an open site for buffer insertion:

```
set_app_var eco_insert_buffer_search_distance_in_site_rows n
```

where *n* is the number of site rows away in which to search (default 8)

- To consider the presence of vertical bottom-metal PG straps and avoid overlaps with signal pins in the same layer:

```
set_eco_options -power_ground_net_layer layer_name
```

3. Specify ECO options and read the physical data by using the `set_eco_options` command:

```
set_eco_options
-physical_tech_lib_path tech_LEF_files
-physical_lib_path LEF_files
-physical_design_path DEF_files
-physical_constraint_file physical_constraint_file
-physical_lib_constraint_file spacing_rule_file_list
-filler_cell_names cell_names
-log_file my_log
```

To get the physical data directly from the IC Compiler II database:

```
set_eco_options
-physical_icc2_lib icc2_lib_path
-physical_icc2_blocks icc2_blocks
...
```

4. (Optional) Preserve specific cells, nets, designs, and library cells by using the `set_dont_touch` command, which applies the `dont_touch` attribute to the specified objects. To allow cells to be sized but not removed, use the `set_size_only` command.
5. Check your LEF/DEF files for missing, incomplete, or problematic data by running the `check_eco` command. If the `check_eco` command reports issues with the LEF/DEF files, resolve the issues, and rerun `check_eco` on the updated LEF/DEF files.

6. Run ECO fixing with these commands:

- fix_eco_drc -physical_mode ...
- fix_eco_timing -physical_mode ...
- fix_eco_power ...

To choose the `-physical_mode` setting, consider the design characteristics and ECO objectives:

- `occupied_site` – Places or resizes cells even if no open sites are available; appropriate for an early-stage ECO or a high utilization design; results in higher fix rates due to more aggressive placement.
- `open_site` – Places cells only in open sites; appropriate for a late-stage ECO or a low utilization design; results in smaller cell displacement due to more conservative placement.
- `freeze_silicon` – Places new cells only on spare cells to preserve the silicon layers.

7. Generate the block-level and top-level ECO change list files by using the `write_changes -format icctcl` command. The resulting change list file includes location information. For 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
...
```

8. Use the ECO change list files in the IC Compiler or IC Compiler II tool to implement the changes at the block and top levels.

For scripts that you can use as a starting point to implement the PrimeTime ECO flow, see the Synopsys Reference Methodology at <https://solvnet.synopsys.com/rmgen>.

Design Data Files

Physical ECO fixing requires the design data files shown in [Table 61](#).

Table 61 Required Design Data Files for Physical ECO

Type of data	Description
Verilog netlist	Netlist written from IC Compiler II or Fusion Compiler, or from a third-party layout tool
Script and setup files	Files to configure SDC, UPF, DMSA, analysis setup

Table 61 Required Design Data Files for Physical ECO (Continued)

Type of data	Description
SPEF/GPD parasitics	Parasitics must have location information Use the following settings in the StarRC command file: NETLIST_NODE_SECTION: YES REDUCTION: NO_EXTRA_LOOPS
StarRC extraction script	Must be the same extraction script used to generate the initial parasitics in the preceding item
Physical design information (NDM or LEF/DEF)	NDM design database from IC Compiler II or Fusion Compiler, or LEF/DEF from a third-party layout tool (LEF/DEF must be consistent with netlist and parasitics files; reextract if necessary)
Additional physical constraints	Supplemental voltage area definitions and placement blockage constraints outside of NDM, if applicable
Advanced spacing rules	Use the <code>export_advanced_spacing_rule</code> command in IC Compiler II to generate this file

The design must be free of legality and routing violations. Otherwise, when the layout tool fixes these violations, it invalidates the initial state given to physical ECO fixing. In the IC Compiler II or Fusion Compiler tool, use the following commands to check the design.

<code>check_legality -verbose</code>	All cells should be legal or fixed.
<code>check_route</code>	There should be no shorts or opens (or if so, all should be understood). All other DRC violations should be understood and expected to have no effect on ECO fixing.

To ensure that the design files are consistent with each other, generate them by following these steps *in this order*:

1. Open the block NDM in the IC Compiler II or Fusion Compiler tool.
2. Ensure that there are no legality or routing violations.
3. Write the Verilog netlist.

This can modify netlist object names to meet Verilog name requirements. By performing this step first, the NDM block and the parasitics reflect any name changes.

4. Save the NDM block.
5. Extract the parasitics from the NDM saved in [Step 4](#).

Block-Level LEF Library Data

From the IC Compiler II tool or Milkyway environment tool, write out the block-level Library Exchange Format (LEF) data for all standard cells as well as all macros. Only one LEF file needs to include the technology information for the design. For example,

From the IC Compiler II Database

```
open_lib lib1
write_lef -include {cell tech} lib1.lef
close_lib

open_lib lib2
write_lef -include {cell} lib2.lef
close_lib

open_lib lib3
write_lef -include {cell} lib3.lef
close_lib
```

From the Milkyway Database

If your physical implementation tool is IC Compiler, you can use the Milkyway Environment tool to write out the LEF files from the design stored in the Milkyway database.

To invoke the Milkyway Environment tool and run a script:

```
$ milkyway -nullDisplay -nogui -tcl -f write_lef.tcl
```

Here is an example of the Milkyway Environment script called write_lef.tcl:

```
set mw_reference_library "lib1 lib2 lib3"
set out 1
foreach lib $mw_reference_library {
    open_mw_lib -readonly $lib
    write_lef -lib_name $lib l$out.lef
    close_mw_lib $lib; set out [expr $out +1]
}
exit
```

If you use this script, all the output LEF files contain the technology section. However, the technology section is only required in one LEF file. If you write out the LEF files one by one, after you generate the first LEF file, you can omit the technology section by using the `write_lef -ignore_tech_info` command.

For more information on using the Milkyway Environment tool, see the *Library Data Preparation for IC Compiler User Guide*.

DEF Files or IC Compiler II Database Files

A Design Exchange Format (DEF) file contains the placement and routing information of the design. PrimeTime physically aware ECO fixing needs the hierarchical top-level and block-level DEF files (version 5.7 or higher) or IC Compiler II database files to find available sites and place buffers near the original routes.

Here is an example of an IC Compiler script called write_def.tcl:

```
open_mw_lib design.mw
open_mw_cel post_route_design
current_design design
link
write_def -all_vias -output file_com.def
```

To include via definitions that might not be part of the via definitions in the technology LEF, use the `write_def -all_vias` command.

For more information, see the “Writing the Floorplan to a DEF File” section in the *IC Compiler Design Planning User Guide*.

DEF to LEF Site Name Conversion

The physically aware ECO flow supports the mapping of DEF site names to LEF site names. For example,

```
pt_shell> set_eco_options -convert_sites {{unit CORE2T}} ...
```

This maps the site name “unit” in the DEF file:

```
ROW STD_ROW_324 unit 560 85680 FS ...;
ROW STD_ROW_325 unit 560 88200 N ...;
```

to the site name “CORE2T” in the LEF file:

```
SITE CORE2T
  CLASS CORE ;
  SYMMETRY X Y ;
  SIZE 0.14 BY 1.2 ;
END CORE2T
```

Missing LEF Files for Hierarchical Blocks

You can read in physical data in LEF/DEF format even if the DEF component for a hierarchical block lacks a LEF macro model, due to unavailable or incomplete LEF data. In these cases, the tool gets the size, shape, port/pin location, and routing obstruction data from the DEF description of the component.

Furthermore, you can specify a list of macro models in the design for which there is no LEF or DEF description available, and allow ECO operations to proceed without considering any instances of those models. Use the `set_eco_options` command as follows:

```
pt_shell> set_eco_options -physical_tech_lib_path $tech_lef_files \
    -physical_lib_path $lef_files \
    -physical_design_path design.def.gz \
    -allow_missing_lef {X2 Y2}
```

In this example, physically aware ECO operations ignore macro models X2 and Y2. The ignored components are omitted from the GUI layout display.

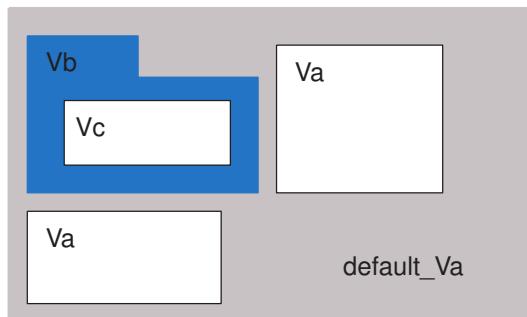
Because the tool ignores these components entirely, the physical spaces that they occupy are treated as empty. You can add placement blockages to these areas to prevent ECO commands from inserting or moving anything there. To report ignored macro models, use the `report_eco_options` command.

If you specify the names of macro models that have LEF or DEF data, the tool overrides your directive to ignore them, proceeds with using the data, and issues a warning message.

Physical Constraint File

When providing ECO guidance, the tool can consider physical information in multivoltage designs, such as disjoint and nested voltage areas.

Figure 359 Multivoltage Physical Constraint Example



- `default_Va` is the default voltage area
- `Va` is a disjoint 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, you need

a physical constraint file that defines voltage areas and placement blockages with these commands:

- `create_placement_blockage` – Creates a placement blockage.
- `create_voltage_area` – Creates a voltage area.

In a flat single-DEF flow, you can specify the physical constraint file in the global `set_eco_options` configuration command:

```
pt_shell> set_eco_options \
           -physical_tech_lib_path ... \
           -physical_lib_path ... \
           -physical_design_path TOP.def \
           -physical_constraint_file TOP_phys_cons.tcl
```

If your design has multiple DEF design files, each must be configured with its own physical constraint file in a separate `set_eco_options` command:

```
pt_shell> set_eco_options \
           -physical_design_path TOP.def \
           -physical_constraint_file TOP_phys_cons.tcl
pt_shell> set_eco_options \
           -physical_design_path BLOCK.def \
           -physical_constraint_file BLOCK_phys_cons.tcl
```

Advanced Spacing Labels and Rules

The PrimeTime physically aware ECO flow can consider advanced spacing rules to minimize displacements and timing degradation during ECO implementation. The tool can consider the following rules:

- Rules exported by the `export_advanced_technology_rules` command in the IC Compiler or IC Compiler II tool:
 - Minimum Vt rules
 - Jog (OD) rules
 - Trim poly (TPO) rules
 - Legal index rules
 - Vertical abutment rules
 - Advanced pairwise ECO physical-only cell spacing rules
- User-specified pairwise interlibrary cell spacing labels and rules

```
set_lib_cell_spacing_label
  -name label_name
```

```

[-left_lib_cells {lib_cell_list}]
[-right_lib_cells {lib_cell_list}]

set_spacing_label_rule
  -labels {label_name_list}
  { minimum maximum }

```

Parasitic Data Files From StarRC

In the StarRC tool, perform parasitic extraction and write out the parasitics in GPD or SPEF format. If multiple scenarios are used, you might need to perform multiple extractions.

The physically aware ECO flow uses the parasitic node locations to accurately determine the effects of placing an ECO cell on a net. For accurate computation, no loops should be present in the parasitic file. Therefore, set these options in the StarRC command file:

```

*****
*StarRC settings for physically-aware ECO
*****
REDUCTION: NO_EXTRA_LOOPS
NETLIST_NODE_SECTION: YES

```

Site-Aware Physical ECO Fixing

The physically aware ECO flow supports downsizing, upsizing, swapping, and insertion of single-height buffer cells, as well as and upsizing and insertion of multiple-height buffer cells for designs containing mixed single-height and double-height cells in the same placement region.

You can restrict usage of specific multiple-height ECO cells to specific site rows, for example, usage of cell BUFFLVT only in site rows of type 2Tunit, as shown in the following figures.

Figure 360 Regular Single-Height and Double-Height Site Rows

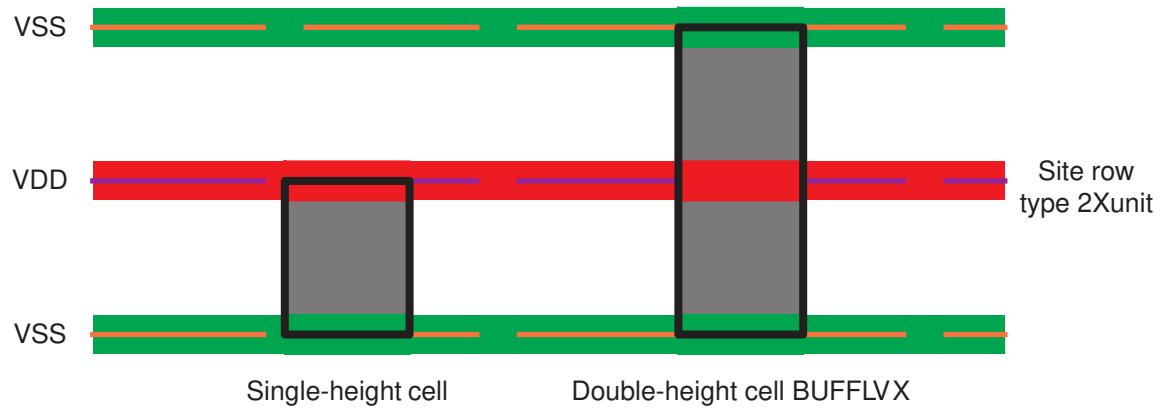
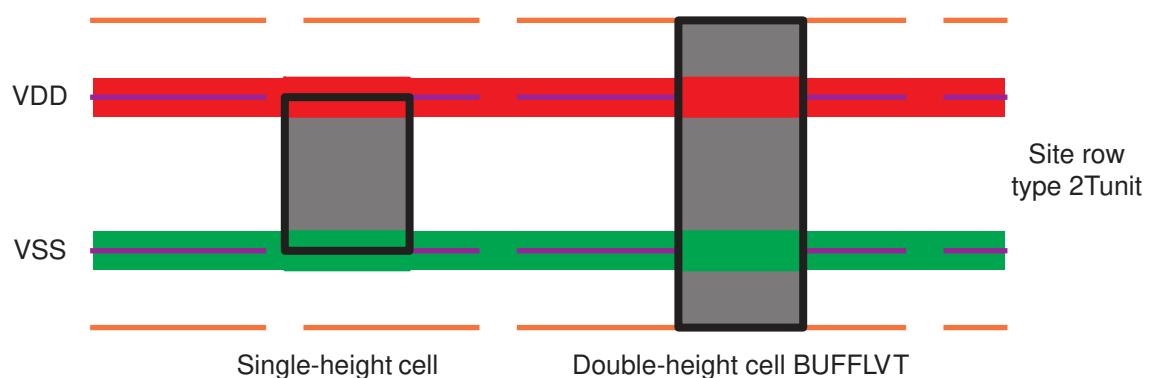


Figure 361 Offset Single-Height and Double-Height Site Rows



Note that physically aware ECO sizing only changes cells widths, not heights.

The multiple-height site row feature requires an exact match between the library cell LEF site name and the corresponding design DEF site row name. For example, here is a library cell definition in a LEF file that specifies the site name `2Tunit` for the cell:

```
MACRO BUFLVT1
  CLASS CORE ;
  ORIGIN 0 0 ;
  SIZE 1.632 BY 0.768 ;
  SYMMETRY X Y ;
  SITE 2Tunit ;
  ...

```

Here are site row definitions in a design DEF file that specify the same `T2unit` name:

```
ROW _row_1 unit 1500 1488 FS DO 4080 BY 1 STEP 96 0 ;
ROW _row_2 unit 1500 1872 N DO 4080 BY 1 STEP 96 0 ;
```

```
...
ROW 2Xrow_1 2Tunit 1500 1680 N DO 4080 BY 1 STEP 96 0 ;
ROW 2Xrow_2 2Tunit 1500 2448 N DO 4080 BY 1 STEP 96 0 ;
ROW 2Xrow_3 2Tunit 1500 3216 N DO 4080 BY 1 STEP 96 0 ;
...
```

To invoke site-aware ECO, set the `eco_physical_match_site_row_names` variable:

```
pt_shell> set_app_var eco_physical_match_site_row_names true
```

When the variable is set to `true`, ECO cell sizing and buffer insertion place the `BUFLVT1` macro cell only in `2Tunit` type rows.

You can change the `eco_physical_match_site_row_names` variable setting at any time to enable or disable the feature for successive open site ECO fixing operations. In the following example, ECO fixing is performed first in default mode using single-height cells, followed by site-aware ECO fixing performed on double-height cells in `2Tunit` rows.

```
set_eco_options \
    -physical_tech_lib_path technology.lef.gz \
    -physical_lib_path library.lef.gz \
    -physical_design_path design.def.gz \
    -physical_lib_constraint_file icc2_adv_tech_rule.gz \
    -log_file lef_def.log

# Keep 2Tunit site name data when reading LEF/DEF physical data
set_eco_options -keep_site_names {2Tunit}

# Check and read in LEF/DEF physical data
report_eco_options
check_eco

# Run default ECO with single-height cells, $SINGLE_BUF buffer list
set_app_var eco_physical_match_site_row_names false
fix_eco_drc -type max_transition -physical_mode open_site \
    -buffer_list $SINGLE_BUF -verbose
fix_eco_timing -type setup -physical_mode open_site -verbose
fix_eco_timing -type hold -physical_mode open_site \
    -buffer_list $SINGLE_BUF -verbose

# Run site-aware ECO with double-height cells, $DOUBLE_BUF buffer list
set_app_var eco_physical_match_site_row_names true
fix_eco_drc -type max_transition -physical_mode open_site \
    -buffer_list $DOUBLE_BUF -verbose
fix_eco_timing -type setup -physical_mode open_site -verbose
fix_eco_timing -type hold -physical_mode open_site \
    -buffer_list $DOUBLE_BUF -verbose
```

The site-aware feature can be used only when the ECO fixing command is invoked with the `-physical_mode open_site` option.

ECO Fixing Methods

You control ECO changes by using the appropriate ECO command, `fix_eco_timing`, `fix_eco_drc`, or `fix_eco_power`; and command options, such as `-type` and `-methods`. These are some of the fixing methods:

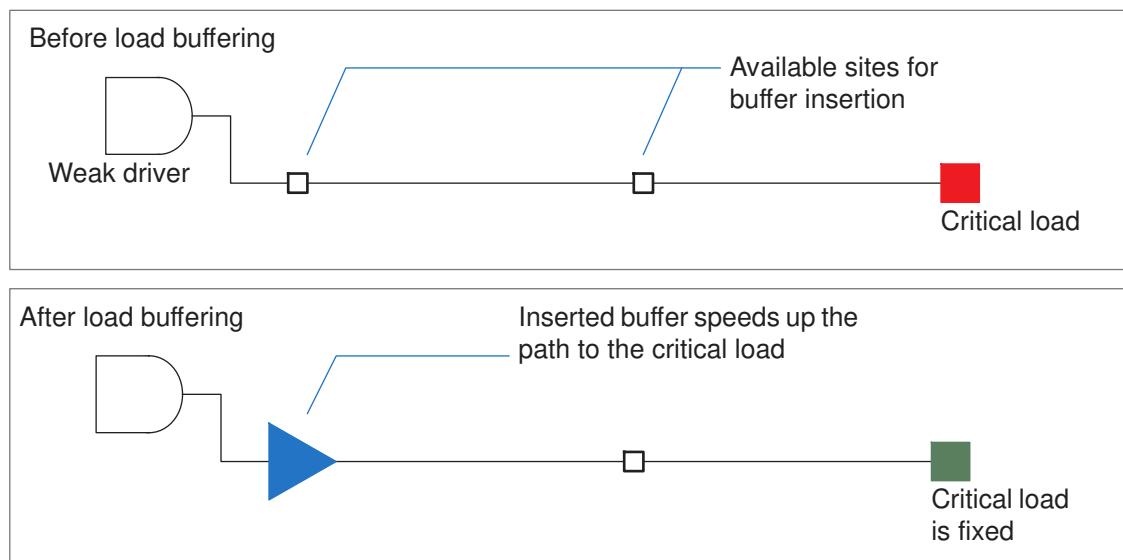
- Load Buffering and Load Shielding
- Side-Load Cell Sizing
- Clock Network ECO Fixing
- ECO Hold Fixing Using Load Capacitance Cells
- Power Recovery
- ECOs With Multiply Instantiated Modules (MIMs)

Load Buffering and Load Shielding

The PrimeTime tool can fix setup violations by performing physically aware load buffering and load shielding, resulting in more fixed violations.

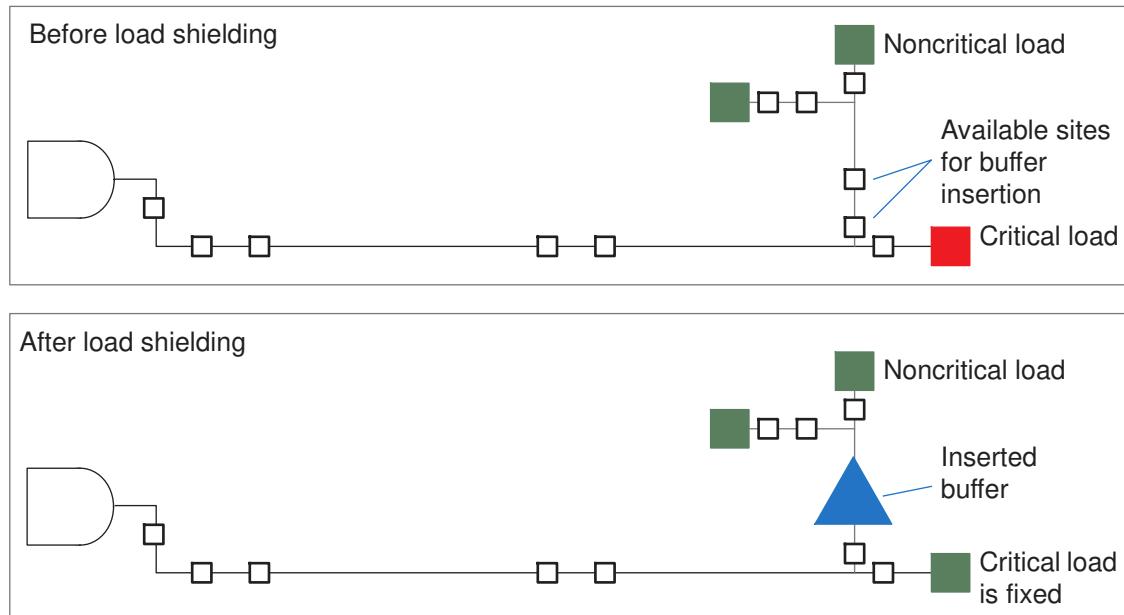
In load buffering, the tool inserts buffers to strengthen weak drivers. The tool automatically chooses the best available location for buffer insertion while considering placement blockages, as shown in the following example.

Figure 362 Load Buffering Example



In load shielding, the tool inserts buffers to shield critical loads from noncritical loads. The tool automatically chooses the best available location for buffer insertion while considering placement blockages, as shown in the following example.

Figure 363 Load Shielding Example



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
  -methods insert_buffer
  -buffer_list {list_of_buffers}
  ...
```

You can use the `insert_buffer` and `size_cell` methods at the same time. If you use them separately for setup fixing, it is recommended to use `size_cell` first, and then `insert_buffer` and other methods to gain incremental improvements.

Side-Load Cell Sizing

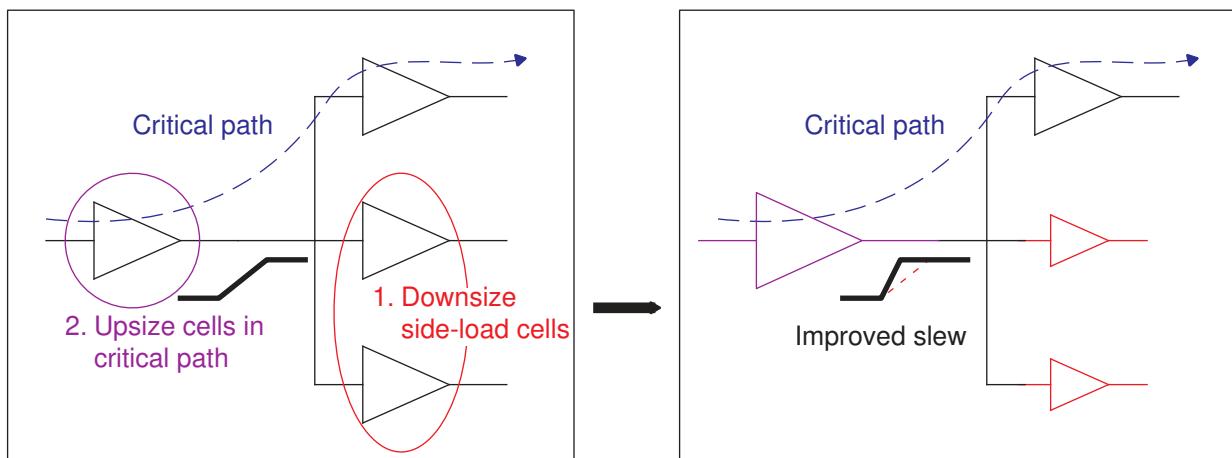
The `-methods` option of the `fix_eco_timing` command specifies the types of ECO changes performed to fix timing violations. The supported methods are cell sizing, general buffer insertion, buffer insertion at load pins, and side-load cell sizing.

The `size_cell_side_load` method first downsizes side-load cells in noncritical paths to reduce parasitic capacitance and then upsizes the cells in the critical path. This method works only with the `-type setup` and `-cell_type combinational` options. For example,

```
pt_shell> fix_eco_timing -type setup -methods {size_cell_side_load} ...
```

This command performs setup timing fixing as shown in the following figure.

Figure 364 Side-Load Cell Sizing Example



Clock Network ECO Fixing

By default, ECO fixing modifies only cells in data paths. However, you have the option to perform fixing in clock networks. Using this feature, you can fix setup and hold violations by changing clock arrival delays in the clock network, resolving hard-to-fix timing issues more efficiently; and you can fix DRC violations in the clock network.

Clock network ECO fixing is supported only in the physically aware ECO flow. Before you read in the LEF/DEF or IC Compiler II data, set the following ECO option to enable reading of physical data for the clock network:

```
set_eco_options -physical_enable_clock_data ...
```

Use the following ECO command options to perform clock network ECO fixing:

```
fix_eco_timing
  -type setup | hold
  -cell_type clock_network
  [-clock_fixes_per_change integer]
  [-clock_max_level_from_reg integer]
  [-methods {size_cell | insert_buffer | insert_inverter_pair}]
  ...
  ...
```

```
fix_eco_drc
  -type max_transition | max_capacitance | max_fanout
  -cell_type clock_network
  [-methods {size_cell | insert_buffer | insert_inverter_pair}]
  ...
  
```

Clock network ECO fixing can affect clock arrival skew in clock trees. To minimize changes to the arrival skew, perform data path ECO fixing first, then apply clock network ECO fixing afterward to fix the remaining violations. For example,

```
pt_shell> fix_eco_timing -type setup -cell_type combinational ...
...
pt_shell> fix_eco_timing -type setup -cell_type clock_network ...
...
```

Fixing Timing Violations by Modifying Clock Networks

When you choose clock network ECO fixing, the `fix_eco_timing` command fixes setup or hold violations by modifying the clock paths to change the clock arrival times. For example,

```
pt_shell> set_eco_options -physical_enable_clock_data ...
...
pt_shell> fix_eco_timing -type setup -methods {size_cell insert_buffer} \
  -buffer_list {buf2 buf4 buf6 buf8} \
  -cell_type clock_network      # Setup fixing in clock paths
...
```

With the `-cell_type clock_network` option, the command fixes setup violations by resizing and inserting buffers throughout the clock network. For example, it can upsize buffers in the launch clock path and insert or downsize buffers in the capture clock path.

Restricting Clock Network Fixing

You can restrict clock network ECO fixing by using additional options:

```
pt_shell> fix_eco_timing -type setup -methods {size_cell insert_buffer} \
  -buffer_list {buf2 buf4 buf6 buf8} \
  -cell_type clock_network \
  -clock_fixes_per_change 4 \
  -clock_max_level_from_reg 6      # Up to 6 levels from register
```

- `-clock_fixes_per_change` – This option specifies a minimum number of violations to be fixed per change, such as 4 in this example. That means each change must fix at least four violations. For example, resizing a cell that fans out through buffers to four register clock pins can fix four timing violations. The default is 1.

Setting a larger limit results in fewer changes, and the changes occur at higher levels in the clock tree, closer to the source and farther from the sequential cells.

- `-clock_max_level_from_reg` – This option specifies the maximum number of buffer cells away from the sequential cell where fixing can occur, such as 6 in this example. That means each buffer resizing or buffer insertion must occur no more than six buffers away from the register's clock pin. The default is no limit.
- Setting a smaller number limits the scope of the clock tree affected by each ECO change. For example, setting the limit to 1 limits each change to the immediate driver of the register clock pin. To restore the default (no limit), set the number to 0.

Fixing DRC Violations in Clock Networks

When you choose clock network ECO fixing of DRC violations, the `fix_eco_drc` command fixes maximum transition, maximum capacitance, or maximum fanout violations in the clock network. For example,

```
pt_shell> set_eco_options -physical_enable_clock_data ...
...
pt_shell> fix_eco_drc -type max_transition \
    -methods {size_cell insert_buffer} \
    -buffer_list {buf2 buf4 buf6 buf8} \
    -cell_type clock_network      # DRC fixing in clock paths
...
```

With the `-cell_type clock_network` option, the command fixes maximum transition violations in the clock network by upsizing buffers and inserting buffers. The default behavior is to fix violations only in data paths.

To enable hold fixing for paths that use clock signals as data, set the following variable:

```
pt_shell> set_app_var eco_enable_fixing_clock_used_as_data true
```

TNS-Driven Clock Network Timing ECO

The `fix_eco_timing` command has options to make clock network changes that fix some violations while allowing others to become worse, while targeting total negative slack (TNS):

```
fix_eco_timing
[-targetViolationType endpoint | tns]
[-wns_limit float]
```

The target violation type settings operate as follows:

- endpoint – Specifies the default targeting method, the same as omitting the `-targetViolationType` option. Changes to the clock network are not allowed to worsen any endpoint violation.
- tns – Specifies TNS-driven fixing. Changes to the clock network are allowed to worsen some violations while fixing others, while targeting total negative slack. The worst negative slack (WNS) is limited to either the existing WNS or a new value specified by the `-wns_limit` option.

A PrimeTime-ADV license is required for TNS violation targeting.

Use the following command for TNS-driven clock network timing fixing:

```
pt_shell> fix_eco_timing -type ... \
    -cell_type clock_network \
    -targetViolationType tns
```

This type of fixing has the following results:

- Total negative slack (TNS) is reduced.
- The slack of some paths may become worse.
- The worst negative slack (WNS) is not worsened, or if the `-wns_limit` option is used, the WNS is no worse than the specified limit.

The following examples demonstrate TNS targeting options:

- [Figure 365](#): Negative hold violations cannot be fixed by default ECO command
- [Figure 366](#): Negative hold fixing targeting TNS, new WNS limited to existing WNS
- [Figure 367](#): Negative hold fixing targeting TNS, new WNS limited to a specific value

Figure 365 Negative Hold Slack Example

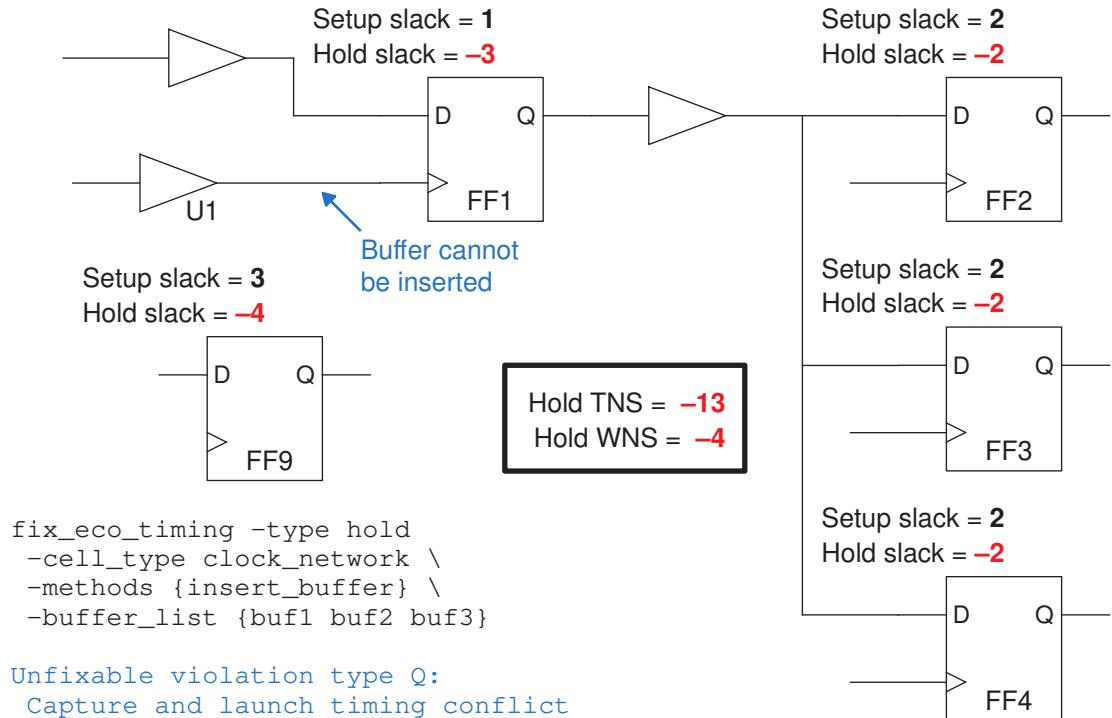


Figure 366 TNS-Driven Hold Fixing With Default Slack Degradation Limit

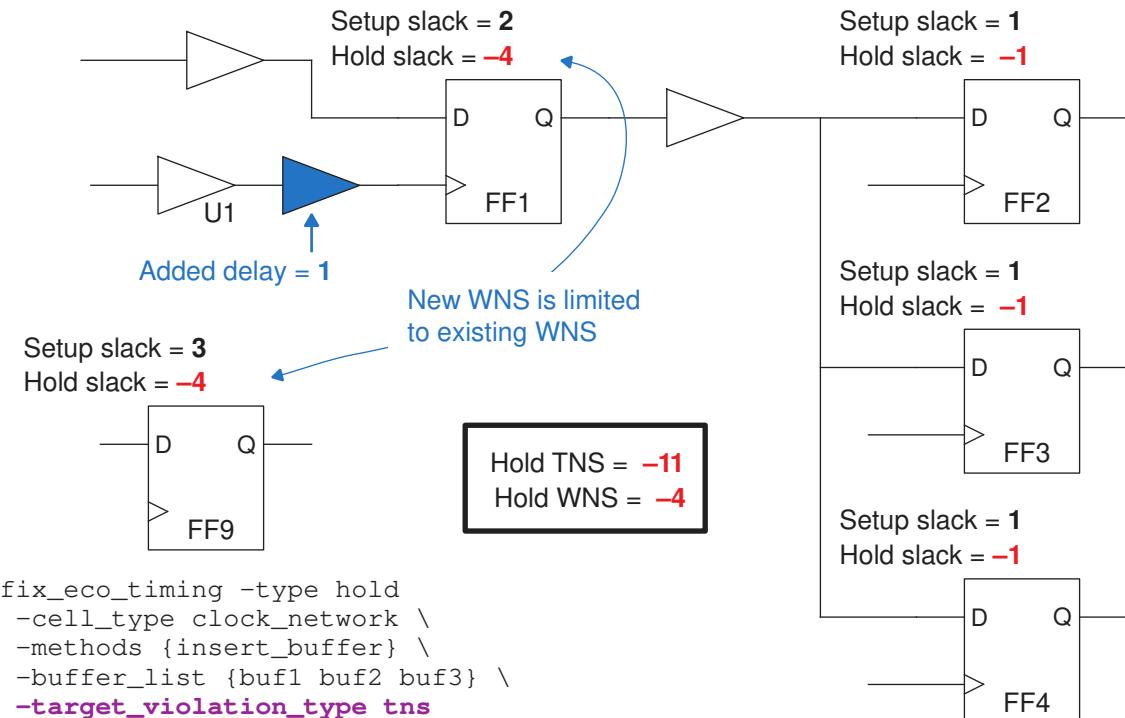
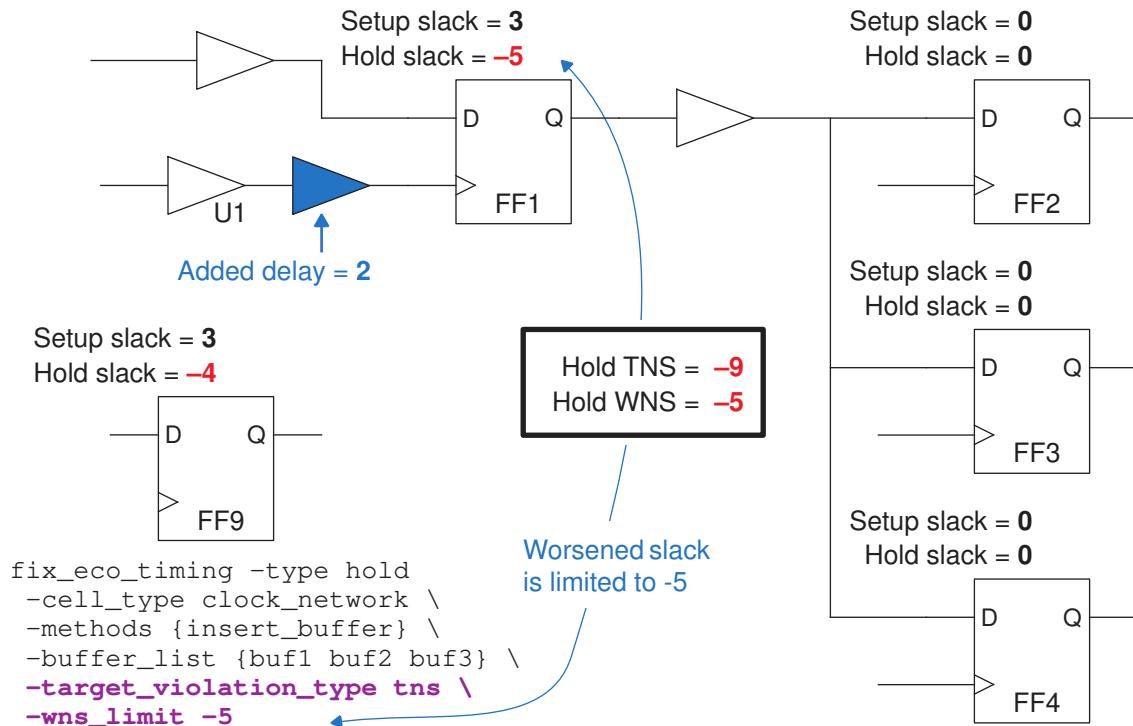


Figure 367 TNS-Driven Hold Fixing With Explicit Slack Degradation Limit



In Figure 365, the three hold violations at FF2, FF3, and FF4 could be fixed by inserting a single buffer in the clock network at inverter U1. However, this would worsen the hold violation at FF1, which is not allowed by default.

In Figure 366, ECO fixing inserts a buffer with a delay of 1, which improves the TNS from -13 to -11 while allowing the negative slack at FF1 to worsen from -3 to -4. The WNS is not allowed to become any worse than the existing WNS, -4.

In Figure 367, the WNS limit is explicitly set to -5, so ECO fixing inserts a buffer with a delay of 2. This improves the TNS to -9 while allowing the negative slack at FF1 to worsen to -5.

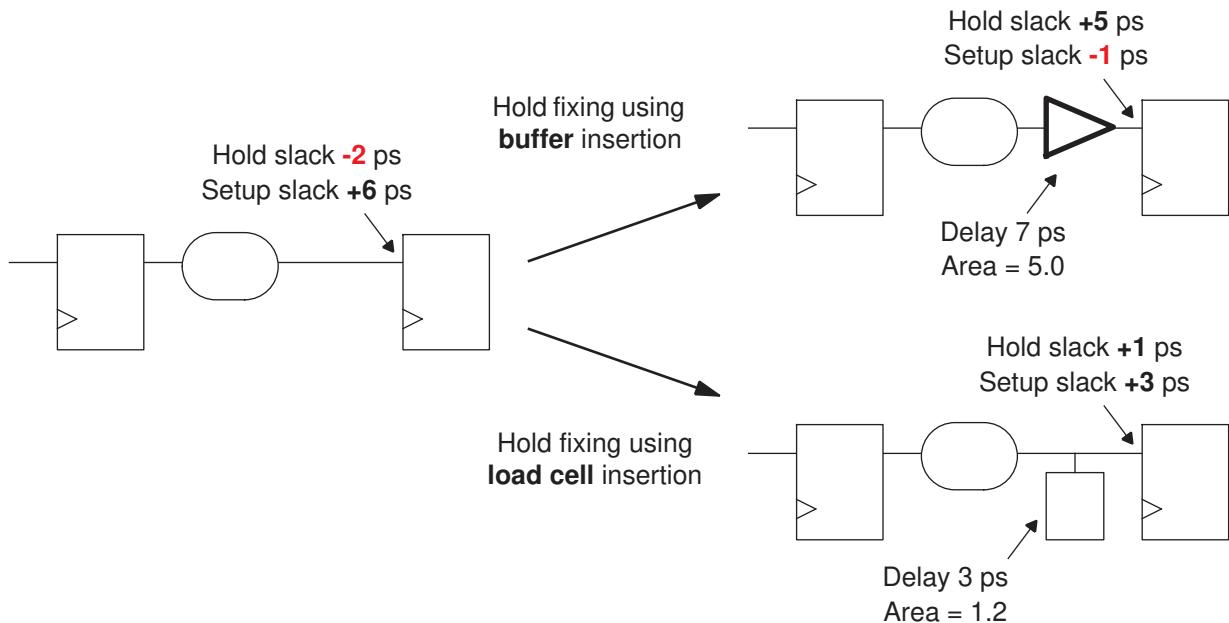
You can set the `-wns_limit` option to a value larger or smaller than the existing WNS. Clock network timing fixing respects the specified limit in either case.

ECO Hold Fixing Using Load Capacitance Cells

Hold fixing using the `fix_eco_timing` command has an option to insert load capacitance cells in the data path. Inserting a load cell has the same delay effect as inserting a buffer but offers a better way to fix small hold violations of about 5 ps or less.

In the following figure, the small hold violation of 2 ps can be fixed by inserting the smallest available buffer, which increases the data path delay by 7 ps. However, this over-fixes the hold violation and introduces a setup violation of 1 ps.

Figure 368 Hold Fixing Using Buffer Insertion and Load Cell Insertion



Instead, inserting a load cell that increases the path delay by 3 ps, resulting in a positive hold slack of 1 ps while maintaining a positive setup slack. An additional benefit is the smaller area of the load cell compared to the buffer.

You can use buffer cells, inverter cells, and dedicated load cells for load cell insertion. Buffer and inverter cells inserted as load cells have unconnected outputs. Dedicated load cells are smaller single-pin cells that are easier to place. They have the following attributes:

Attribute Name	Type	Value
<hr/>		
function_id	string	unknown
is_black_box	boolean	true
is_combinatorial	boolean	true
number_of_pins	int	1

To insert load cells, use the `fix_eco_timing -type hold` command with the `-load_cell_list` option. You can fix small hold violations using load cells and then fix the remaining hold violations using buffers, as shown in the following script:

```
# Fix small hold violations by inserting load cells
fix_eco_timing -type hold -methods insert_buffer -load_cell_list $clist \
-slack_lesser_than 0.000 -slack_greater_than -0.003

# Fix remaining hold violations by inserting buffers
fix_eco_timing -type hold -methods insert_buffer -buffer_list $bflist
```

Alternatively, you can allow the tool to insert both load cells and buffers in a single operation:

```
# Fix hold violations by inserting load cells and buffers
fix_eco_timing -type hold -methods insert_buffer \
-load_cell_list $clist -buffer_list $bflist
```

In this case, the tool automatically fixes small hold violations by inserting load cells, and larger hold violations by inserting buffers.

For physically aware ECO fixing, write out the changes as IC Compiler or IC Compiler II commands as follows:

```
pt_shell> write_changes -format icctcl -output wc_eco.tcl
```

The file written by the `write_changes` command contains commands like the following to set the location and orientation of each new load cell:

```
create_cell U_LOAD_CAP_CELL_1 CLOAD1
connect_net n231 [get_pins {U_LOAD_CAP_CELL_1/A}]
set_cell_location -coordinates {150.00 30.60} -orientation N
```

In the physically aware flow, the `fix_eco_timing` command limits the search area for an available site by considering the `eco_insert_buffer_search_distance_in_site_rows` variable setting, for both buffer insertion and load cell insertion.

Power Recovery

The `fix_eco_power` command tries to recover power and area by downsizing cells in paths with positive setup slack or by removing buffers from paths with positive hold slack, without introducing or worsening timing violations or DRC violations. The command options let you specify any one of the following power recovery methods:

- Replace cells to minimize area (the default)
- Replace cells to minimize a library cell numeric attribute (use the `-power_attribute` option)

- Replace cells based on library cell preference, as specified by an explicit string priority list (use the `-pattern_priority` option)
- Replace cells to minimize switching, leakage, or total power using power analysis data generated by the `PrimePower update_power` command (use the `-power_mode` option)
- Remove buffers from paths with positive hold slack (use the `-methods remove_buffer` option)

To perform power recovery, use the following procedure.

1. (Optional) Guide power recovery with these controls:

- Restrict alternative library cells by attribute values by setting the `eco_alternative_cell_attribute_restrictions` and/or `eco_alternative_cell_instance_based_restrictions` variables
- Exclude unconstrained cells as candidates for swapping by setting the `eco_power_exclude_unconstrained_cells` variable to `true`

2. (Optional) 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
```

3. (Optional) Report the cells used in the design with 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%)
```

4. Perform power recovery by using the `fix_eco_power` command:

```
fix_eco_power
[-methods { ... }]
[-power_attribute ...]
[-pattern_priority { ... }]
[-power_mode ...]
...
```

5. Report the power recovery results by using the PrimePower `report_power` command.
For example:

```
pt_shell> set_app_var power_enable_analysis true
pt_shell> report_power -groups {register combinational sequential}
```

Power Group	Internal Power	Switching Power	Leakage Power	Total 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.

The following topics describe additional power recovery features:

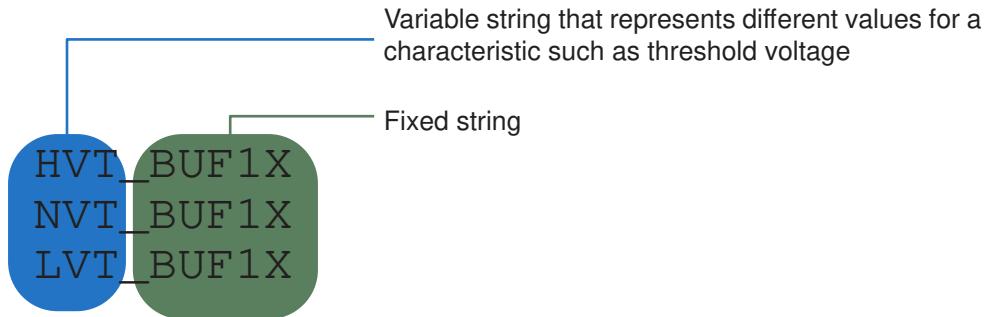
- [Power Recovery Based on Library Cell Names](#)
- [Power Recovery Based on a Library Cell String Attribute](#)
- [Power Recovery Based on a Library Cell Leakage Attribute](#)
- [Power Recovery Based on PrimePower Data](#)

- Accelerated Power Recovery Using Machine Learning
- Excluding I/O Paths From Power Recovery

Power Recovery Based on Library Cell Names

For replacing higher-power cells based on the library cell names, the cells must follow a naming convention starting with a variable string and ending with a fixed string, as shown in the following example.

Figure 369 Library Cell Naming Example for Power Recovery Cell Swapping



For example, these 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 replacement 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}
```

Be sure to list the variable name prefixes in order of priority, lowest-power cells first.

Power Recovery Based on a Library Cell String Attribute

If the library cell names do not follow a naming convention, you can still perform cell swapping based on a user-defined library cell string 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 replacement based on a priority list for a user-defined string attribute, use commands like the following:

```
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
```

Be sure to list the attribute strings in order of priority, lowest-power cells first.

Power Recovery Based on a Library Cell Leakage Attribute

Leakage power may increase when cells are upsized to improve setup timing. By default, the `fix_eco_timing` command performs optimization to meet timing requirements while minimizing the increase in area, without considering leakage power.

If leakage power is more important than area for setup timing optimization, use the `-power_attribute` option of the `fix_eco_timing` command. Specify the name of a user-defined `lib_cell` attribute that defines the leakage power for the cell. For example,

```
# Create a user-defined attribute "leak_attr" for lib_cell object
define_user_attribute leak_attr -type float -class lib_cell

# Assign values that reflect leakage of each lib_cell at worst corner
set_user_attribute -class lib_cell \
    [get_lib_cells LIB_H/INV1X_HVT] leak_attr 1.0
set_user_attribute -class lib_cell \
    [get_lib_cells LIB_H/INV2X_HVT] leak_attr 2.0
set_user_attribute -class lib_cell \
    [get_lib_cells LIB_L/INV1X_LVT] leak_attr 10.0
```

```
set_user_attribute -class lib_cell \
    [get_lib_cells LIB_L/INV2X_LVT] leak_attr 15.0
...
# Perform setup timing fixing with leakage power consideration
fix_eco_timing -power_attribute leak_attr
```

In this example, the `fix_eco_timing` command chooses library cells that minimize leakage power. It only replaces existing cell instances that have the `leak_attr` attribute with other library cells that also have the `leak_attr` attribute; it ignores cells that do not have the attribute.

Instead of assigning the leakage values explicitly, you can import the attribute values from the cell library database. For details, see [SolvNetPlus article 000019424, “Extracting Leakage Power for Library Cells”](#).

In distributed multi-scenario analysis, the respective library cells in different libraries must be assigned matching leakage power values.

Power Recovery Based on PrimePower Data

The PrimePower tool performs comprehensive power analysis using actual switching activity and library-defined power data such as dynamic and leakage power of each cell. The tool performs a power analysis when you use the `update_power` or `report_power` command at the `pt_shell` prompt.

To use PrimePower analysis data in power recovery, use the `fix_eco_power` command with the `-power_mode` option set to `dynamic`, `leakage`, or `total`. Then the power recovery process modifies the design to minimize the dynamic (switching) power, leakage power, or total (dynamic plus leakage) power as measured by the `update_power` command.

With the `-power_mode total` option, the tool chooses actions to reduce the total power the most, which can vary with local conditions in the design. For example, where not much timing slack is available, it can choose to recover power by either downsizing the cell or by increasing the cell threshold voltage, the choice depending on the local switching activity. High switching activity favors downsizing (to reduce switching power), whereas low switching activity favors increasing the threshold voltage (to reduce leakage). Where plenty of timing slack is available, it can do both.

Power Recovery ECO Options

The `fix_eco_power` command has the following options to support optimized power recovery:

```
fix_eco_power
...
[-power_mode total | dynamic | leakage]  (Specifies power mode)
[-dynamic_scenario scenario_name]  (Specifies dynamic power scenario)
[-leakage_scenario scenario_name]  (Specifies leakage power scenario)
```

The `-power_mode` option specifies the types of power recovery to consider: dynamic power only, leakage power only, or both (total). The `-power_mode` option cannot be used with the `-power_attribute` or `-pattern_priority` option.

In distributed multi-scenario analysis (DMSA) flow, the tool gets its dynamic power data from exactly one scenario, which you specify with the `-dynamic_scenario` option. Similarly, it gets its leakage power data from exactly one scenario, which you specify with the `-leakage_scenario` option. These options are used only in a DMSA flow.

In general, you should specify the scenario showing the worst dynamic power and worst leakage power, respectively, for these two options. They could be the same scenario or two different scenarios. You should continue to analyze all scenarios for their timing and DRC constraints, which are honored by the power recovery command.

Single Scenario Power Recovery ECO Example

The following script example demonstrates the ECO total power recovery process.

```
restore_session Session1          # Restore analysis session
set_app_var power_enable_analysis true    # Enable PrimePower
set_app_var power_clock_network_include_register_clock_pin_power false
read_saif or read_vcd activity_file      # Read switching activity data
...
update_power                         # Set up power analysis
report_power                         # Power analysis
fix_eco_power -power_mode total       # Dynamic/static power recovery
report_power                         # Report power improvement
```

If you set the `power_clock_network_include_register_clock_pin_power` variable to `false`, the PrimePower tool accounts for the internal power of register cells at the clock input pin as part of the register cell and not as a member of the clock network, so the power recovery is better reflected as part of the data path. The power recovery process can potentially replace register cells to reduce clock pin power (it does not modify the clock network).

Multi-Scenario Power Recovery ECO Example

In the following DMSA example uses six scenarios. Scenario S1 has the most leakage power and scenario S3 has the most dynamic power. To perform total power recovery, use a script similar to the following:

```
current_session {S1 S2 S3 S4 S5 S6}      # Apply timing/DRC constraints
                                             # from all scenarios, S1-S6
current_scenario {S1 S3}                   # update_power is only needed for
                                             # power scenarios, S1 and S3
remote_execute {
  set_app_var power_enable_analysis true
  set_app_var power_clock_network_include_register_clock_pin_power false
  read_saif or read_vcd activity_file      # Read switching activity data
  ...
  update_power
```

```

}
current_scenario -all
fix_eco_power -power_mode total \
    -leakage_scenario S1 \
    -dynamic_scenario S3

```

Accelerated Power Recovery Using Machine Learning

The PrimeTime tool offers machine learning to speed up power recovery performed by the `fix_eco_power` command. Power recovery can be a computation-intensive process, especially when using path-based analysis, because the tool must analyze the timing, topology, and power in detail to find the optimum cell replacements.

If the machine learning feature is enabled, when the `fix_eco_power` command finishes power recovery, it saves its cell replacement decisions in a file called the training data file. The decision data includes the cell instance replacements, as well as decisions not to replace, and the associated timing and circuit topology conditions that led to the decisions.

The next time you perform power recovery, the `fix_eco_power` command reads the previous training data files, trains the machine learning model, and looks for cells and subcircuits that match the conditions of the earlier sessions that can be provided to the trained model.

Where matching conditions are found, the tool makes the same cell replacement decisions based on the machine learning model, avoiding the time-consuming detailed analysis already performed.

Using this feature requires a PrimeTime-ADV-PLUS license.

Machine Learning Runtime Benefit

The amount of runtime reduction depends on the similarity between the current design and the designs used during training, with respect to timing environment and circuit topology.

For example, you are likely to see a large runtime benefit under the following conditions:

- Same design or similar design type
- Same or similar clock characteristics
- Same or similar layout and parasitic data

On the other hand, you are likely to see little or no runtime benefit under the following conditions:

- Different design type (CPU versus modem; latches versus flip-flops)
- Different clock characteristics (period, uncertainty)
- Different operating conditions (voltage, temperature)
- Different technologies or process nodes

PrimeTime ECO Machine Learning

To enable machine learning for faster power recovery by the `fix_eco_power` command, set the `training_data_directory` variable to the name of a directory. No other user action is needed. For example,

```
pt_shell> set_app_var training_data_directory ./train_dir
```

The `fix_eco_power` command uses the specified directory to retrieve the training data from previous power recovery sessions and to store the learning data from the current session. The `fix_eco_power` command performs these actions automatically when the `training_data_directory` variable is set to a directory name.

The first time you do power recovery with the machine learning enabled, there is no training data available, so power recovery takes the same amount of time as usual. However, upon completion, the `fix_eco_power` command writes out what it has learned to a file in the training directory.

```
pt_shell> read_verilog my_design1.v
...
pt_shell> set_app_var training_data_directory ./train_dir
...
pt_shell> fix_eco_power -power_mode total ...
Information: Machine learning enabled ...
...
Training coverage 0.0%
...
Information: Completed
Information: Writing training data train_dir/tr1.td
Information: Elapsed time [ 4280 sec]
```

The next time and subsequent times that you do power recovery, the tool reads in all training data from the directory and applies what it can.

```
pt_shell> read_verilog my_design2.v
...
pt_shell> set_app_var training_data_directory ./train_dir
...
pt_shell> fix_eco_power -power_mode total ...
Information: Machine learning enabled ...
Information: Reading training data train_dir/tr1.td
...
Machine Learning Summary:
-----
Total number of cells          10272492
Cells eligible for power recovery 4788129
Trained cells                  4106599
Adjusted cells                 2109
Untrained cells                679421
Training coverage              85.0%
Training adjustment            0.0%
```

```

Retraining scope           15.0%
...
Information: Completed
Information: Writing training data train_dir/tr2.td
Information: Elapsed time [ 998 sec]

```

In the “Machine Learning Summary” report:

- Training coverage = (Trained cells) / (Cells eligible for power recovery)
- Training adjustment = (Adjusted cells) / (Cells eligible for power recovery)
- Retraining scope = (Untrained cells) / (Cells eligible for power recovery)

Each usage of the `fix_eco_power` command reads all of the machine learning files from the specified directory and writes out any new learned data to a new file in the directory.

Training Coverage

When you use training data, the `fix_eco_power` command reports the usability of the available data as the “training coverage.” This is the percentage of cells eligible for power recovery that use the learned optimization actions. For example,

```

pt_shell> fix_eco_power -power_mode total ...
Information: Machine learning enabled ...
Information: Reading training data train_dir/tr1.td
...
Training coverage          85.0%
...

```

In this example, the training coverage is 85%, which means that 85% of the cells eligible for power recovery matched the timing and topology conditions available from the training data generated by previous runs, and the tool applied the learned optimization actions to those cells. The remaining 15% of these cells did not match previous conditions and were analyzed in detail for power recovery, the same as using no training data.

If you perform power recovery on a design and save the training data, performing another power recovery on the same design under the same timing conditions results in 100% coverage and greatly increased performance, for example, 5X faster. On the other hand, performing power recovery on a very different type of design or a very different clock period may result in 0% coverage and no runtime benefit.

The best runtime benefit comes from training coverage in the 90% to 100% range. You might see very little runtime benefit for training coverage below 50%.

Training Data Files

The `fix_eco_power` command reads training data files from the directory specified by the `training_data_directory` variable and writes out new training data to the same directory.

The file format is binary and you cannot read or edit the contents. However, you can copy and delete these files to modify the training data set. The files are named according to the design from which the data originated and the date and time at which they are generated.

Generally, you should allow the `fix_eco_power` command to read all of the files in the training directory, so that it can find the most suitable data to match the current design. The runtime and memory overhead for reading many files is minor. However, if you wish to read only a subset of the files in the directory, you can do so by using the `-training_data` option of the `fix_eco_power` command.

When to Enable Machine Learning

There is never any harm in enabling machine learning because it can only help the runtime; it cannot hurt. If the `fix_eco_power` command cannot use the available data, it simply ignores that data and performs power recovery using normal detailed analysis, and it still stores what it learns in the current session for possible use in future sessions.

The runtime overhead for reading and saving training data is considered negligible. Therefore, it is recommended that you leave the feature enabled every time you use the `fix_eco_power` command, so you can gather data from the widest possible range of design styles, timing conditions, and topologies. The more data you have, the more likely that the data will be useful in future power recovery sessions.

Using the machine learning feature reduces runtime, with little effect on quality of results. You can expect to see similar power recovery results whether or not you use the feature.

Comprehensive Training Data Coverage

If you generate training data to cover all stages of your design cycle, you can gain the benefit of machine learning all the time in future sessions. For example, you can run one or more setup, hold, and DRC fixing operations in parallel to reach different states of the design, and run the `fix_eco_power` command to create training data from each of these stages:

```
# Initial run with DRC, setup, and hold fixing
...
update_timing
save_session after_update      # save session after timing update
fix_eco_drc
save_session after_drc_fix     # save session after DRC fixing
fix_eco_timing -type setup
save_session after_setup_fix   # save session after setup fixing
fix_eco_timing -type hold
save_session after_hold_fix    # save session after hold fixing

# Training Run 1 on host 1
set_app_var training_data_directory ./all_training_data
restore_session after_update
fix_eco_power
```

```
# Training Run 2 on host 2
set_app_var training_data_directory ./all_training_data
restore_session after_drc_fix
fix_eco_power

# Training Run 3 on host 3
set_app_var training_data_directory ./all_training_data
restore_session after_setup_fix
fix_eco_power

# Training Run 4 on host 4
set_app_var training_data_directory ./all_training_data
restore_session after_hold_fix
fix_eco_power
```

After all four runs finish, the training data directory contains four different sets of training data created from different stages of the design cycle. When you start working on a new revision of the design, you are likely to get high training data coverage due to matching of current timing and topology conditions with one or more of the previous training sessions, resulting in greatly reduced runtime during power optimization.

User-Scripted ECO Machine Learning

If you have created custom scripts to perform power optimization, you can direct the tool to learn from these strategies and apply them in future execution of the `fix_eco_power` command. To do this, use the `record_training_data` and `write_training_data` commands as shown in the following example.

```
# Machine learning session
...
record_training_data                      # Start recording training data
...
source my_eco_changes.tcl                 # User's power optimization script
...
write_training_data -output my_training.td # Generate training data
...
```

In a later session, the `fix_eco_power` command uses the training data when you specify the training data directory with the `-training_data` option.

```
# Machine learning usage
...
fix_eco_power -training_data my_training.td # Use training data
...
```

In this machine learning flow, the PrimeTime tool does not check or evaluate the quality of the power optimization script. It simply accepts the changes made by the user script as beneficial and saves the timing and topological conditions surrounding the changes performed by the script. It writes the training data to the specified file.

When you later use the `fix_eco_power` command with the `-training_data` option, the command reads in the training data and applies the changes where it finds cells with similar timing and topological conditions, without performing a detailed analysis. Where there is no match with the learned timing and topological conditions, the command performs normal power optimization using detailed analysis, the same as using no machine learning.

Excluding I/O Paths From Power Recovery

The `fix_eco_power` command can perform power recovery on internal (register-to-register) paths only, while leaving I/O paths untouched.

To use this feature,

- Set the `-start_end_type` option to `reg_to_reg`.
- Set the `-pba_mode` option to `path` or `exhaustive`. This is required.

For example,

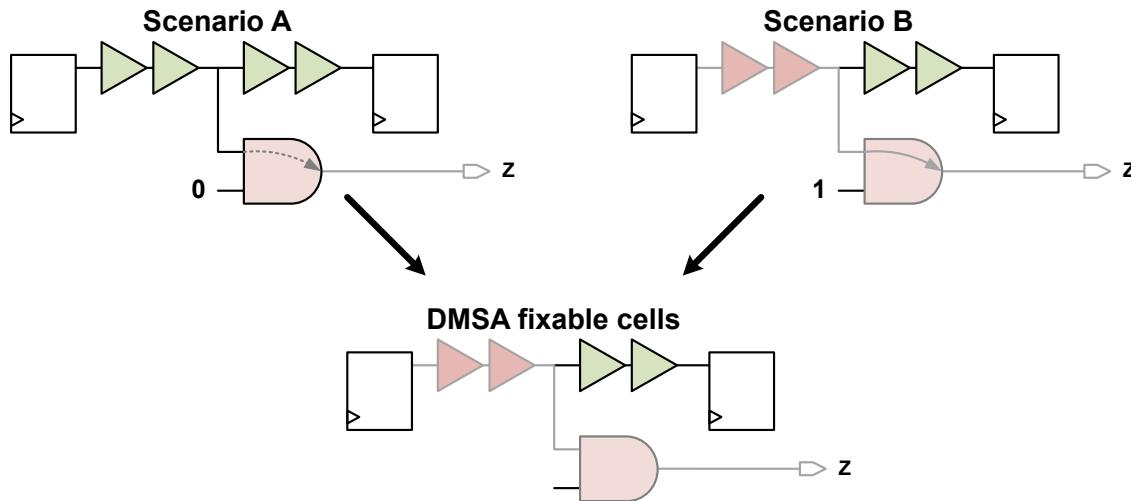
```
pt_shell> fix_eco_power \
           -start_end_type reg_to_reg \
           -pba_mode exhaustive
```

The `-start_end_type` option also exists on the `report_timing` and `get_timing_paths` commands. However, the only valid value for the `fix_eco_power` command is `reg_to_reg`.

For power reduction, a cell is considered “internal” if there are no paths, constrained or unconstrained, that reach the cell from an I/O port. Case analysis and disabled timing arcs are considered during the check.

In a DMSA analysis, the cell is excluded if it reaches an I/O port in any scenario, as shown in [Figure 370](#).

Figure 370 Merging the Set of Unfixable I/O Cells in DMSA fix_eco_power



Cells that reach I/O ports are left untouched, even if their slack is less than the setup or hold margin used for fixing. They are reported with reason code “V” in the unfixable reasons report.

ECOs With Multiply Instantiated Modules (MIMs)

When you set the `eco_enable_mim` variable to `true`, the tool performs the same ECO changes across each set of MIMs and writes out a single change list file for each set of MIMs.

In physically aware fixing, the tool recognizes a set of MIM instances when they share the same DEF file; in logic-only fixing, it recognizes them when they share the same parasitics file according to the `-path` option in the `read_parasitics` command.

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 the 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 because it separately analyzes each MIM group.

To report the defined MIM groups, use the `report_eco_mim_instances` command.

3. Perform ECO fixing by using the `fix_eco_drc`, `fix_eco_timing`, and `fix_eco_power` commands.
4. 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:

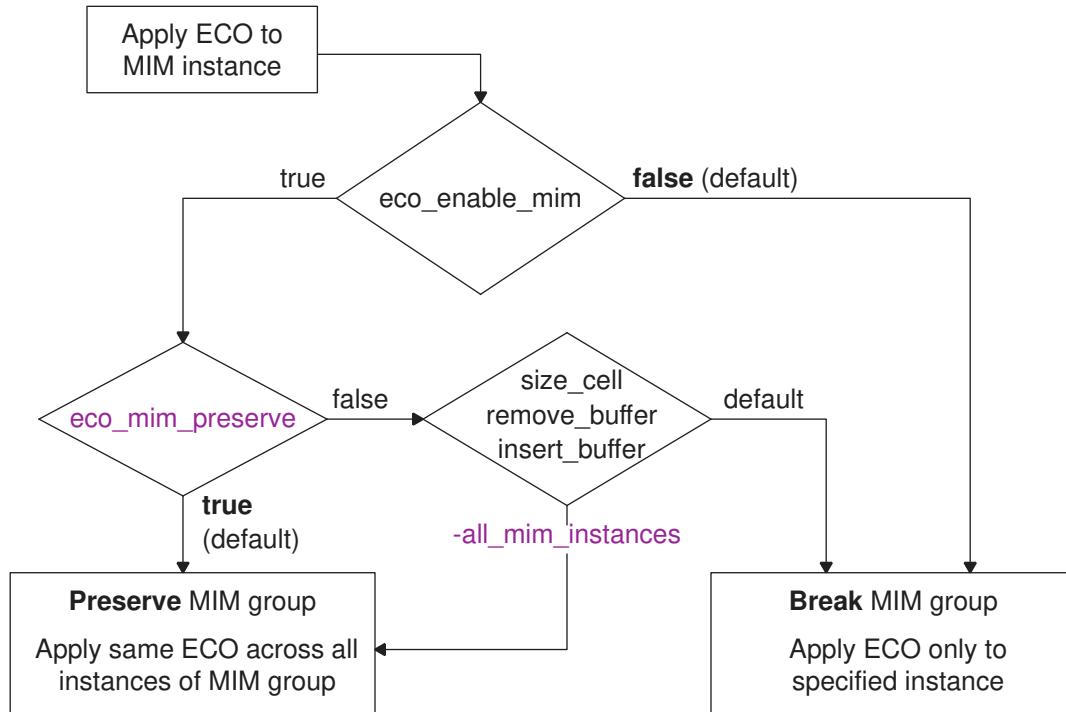
This is a custom configuration example. By default, all changes in the CPU module are specified in one single ECO file.

The following options let you selectively control and report the application of ECO changes to MIM instances:

- The `eco_mim_preserve` variable specifies whether to preserve MIM groups when the `eco_enable_mim` variable is set to `true`.
- The `-all_mim_instances` option of the `size_cell`, `insert_buffer`, and `remove_buffer` commands overrides the `eco_mim_preserve` variable set to `false`.

The following diagram shows the options for applying ECO changes to MIM instances.

Figure 371 ECOs Applied to MIM Instances



Reporting Unfixable Violations and Unusable Cells

When you perform ECO timing or DRC fixing, you can choose to report information about why violations could not be fixed. Similarly, for ECO power fixing, you can choose to report information about why cells could not be resized for power reduction.

These features are described in the following topics:

- [Reporting Unfixable Timing and DRC Violations](#)
- [Reporting Unusable Cells for Power Reduction](#)

Reporting Unfixable Timing and DRC Violations

The `fix_eco_timing` and `fix_eco_drc` commands offer an “unfixable violations” reason feature that reports why a violation could not be fixed.

To use this feature, first set the `eco_report_unfixed_reason_max_endpoints` variable to the number of unfixable violations to report:

```
pt_shell> set_app_var eco_report_unfixed_reason_max_endpoints 50
```

You can then include the reasons in the command output, write them to a file, or estimate the reasons without fixing (for timing ECO fixing only):

Table 62 Reporting Unfixable Timing and DRC Violations

To do this	Use this command
Estimate the unfixable violations without fixing (<code>fix_eco_timing</code> only)	<code>fix_eco_timing -estimate_unfixable_reasons</code>
Run fixing, and include the unfixable violations in the command output	<code>fix_eco_timing fix_eco_drc -verbose</code>
Run fixing, and write the unfixable violations to a log file	<code>fix_eco_timing fix_eco_drc -unfixable_reasons_prefix file_name_prefix [-unfixable_reasons_format text csv]</code>
Run fixing, include the unfixable reasons in the output, and write them to a log file	<code>fix_eco_timing fix_eco_drc -verbose -unfixable_reasons_prefix file_name_prefix [-unfixable_reasons_format text csv]</code>

For example,

```
pt_shell> set_app_var eco_report_unfixed_reason_max_endpoints 50
50
pt_shell> fix_eco_timing -type setup -estimate_unfixable_reasons
Information: Using option -estimate_unfixable_reasons. ...
Information: The design will not be changed.
Information: 22 violating endpoints located... (PTECO-022)
Information: 22 endpoints are being considered for fixing... (PTECO-027)
...
Unfixable violations:
A - There are available library cells outside area limit
B - Delay improvement is too small to fix the violation
C - The violation is in clock network
I - Buffer insertion with given library cells cannot fix the violation
S - Cell sizing with alternative library cells cannot fix the violation
T - Timing margin is too tight to fix the violation
U - UPF restricts fixing the violation
V - Net or cell is invalid or has dont_touch attribute
W - Fixing the violation might degrade DRC violations

Violation                                         Reasons   Prio/slk
-----
S:I_ORCA_TOP/I_PCI_CORE/pad_en_reg/Q
```

```

U7/ZN          A B W      P8
U62/ZN          A B W      P8
U63/Z          A B W      P8
U63ASTipoInst495/Z  A B W      P5
U63ASTipoInst494/Z  A B W      P3
pad_iopad_1/PAD      B W      P9
E:pad[1]           -1.157
...
C:U54/Z          B W      P9
    pc_be_iopad_1/PAD
E:pc_be[1]           -0.017
1
Final ECO Summary:
-----
Number of size_cell commands      276
Total number of commands          276
Area increased by cell sizing     128.50
Total area increased             128.50

Information: Elapsed time ...
Information: Completed at ...

```

For detailed information about unfixable violation reports, see the man page for the `fix_eco_timing` or `fix_eco_drc` command.

For timing ECO fixing, the estimation report can guide you in fine-tuning the command options without actually performing ECO changes. This report provides guidance on how to address the unfixable violations. For example, the report can help you decide between modifying the buffer list, relaxing area constraints, or relaxing margins before you perform actual fixing. Generating the report is faster than actual fixing.

Estimation of unfixable violation reasons is compatible with all the other `fix_eco_timing` command options. For example, to restrict the report to specific paths, specify those paths using `-from` or `-to`, as in the following example.

```
pt_shell> fix_eco_timing -type setup -estimate_unfixable_reasons \
           -from I_ORCA_TOP/I_PCI_CORE/pad_en_reg/Q -to pad[1]
```

Reporting Unusable Cells for Power Reduction

The `fix_eco_power` command offers an “unusable cells” reason feature that reports why a cell could not be downsized for power reduction.

The `fix_eco_timing` and `fix_eco_drc` commands have a similar “unfixable reasons” feature. However, there is no concept of a violation with power reduction, so the terminology is somewhat different for the `fix_eco_power` command.

To use this feature, first set the `eco_power_report_max_unusable_cells` variable to the number of unusable violations to report:

```
pt_shell> set_app_var eco_power_report_max_unusable_cells 100
```

You can then include the reasons in the command output, write them to a file, or both:

Table 63 Reporting Unusable Cells for Power Reduction

To do this	Use this command
Run recovery, and include the unusable reasons in the command output	<code>fix_eco_power -verbose</code>
Run recovery, and write the unusable violations to a log file	<code>fix_eco_power -unusable_reasons_prefix file_name_prefix</code>
Run recovery, include the unusable reasons in the output, and write them to a log file	<code>fix_eco_power -verbose -unusable_reasons_prefix file_name_prefix</code>

For example,

```
pt_shell> set_app_var eco_power_report_max_unusable_cells 8
8
pt_shell> fix_eco_power -verbose
...
```

Unusable Cells:

- B - Power benefit is too small
- L - Physical constraints restrict sizing
- S - Cell has no alternate library cell with better power
- T - Sizing cell might degrade timing
- U - UPF restrictions prevent sizing
- V - Cell is invalid or has dont_touch attribute
- W - Sizing cell might degrade DRC
- X - Cell is unusable for ECO
- Z - Cell is sized

Cell name	Lib_cell name	Reasons
ff1	SDFFARX1_HVT	W
ff2	SDFFARX2_HVT	T
ff3	SDFFARX1_HVT	U
ff4	SDFFARX2_HVT	L
U1676	AO22X1_HVT	X V
U1674	OA21X1_HVT	S
U1655	AO22X1_HVT	W
U1654	OA21X1_HVT	T W Z

Unusable cells summary:

Total number of cells checked:	8
Total number of cells with reason code S :	1
Total number of cells with reason code L :	1
Total number of cells with reason code T :	2
Total number of cells with reason code U :	1
Total number of cells with reason code V :	1
Total number of cells with reason code W :	3
Total number of cells with reason code X :	1
Total number of cells with reason code Z :	1

The reported cells are ordered by (and thus truncated to the limit by) the type of power fixing being performed:

Power fixing type	Ordering and truncation method
Area-based downsizing	In order of [get_cells -hier *]
Leakage swapping	In order of pattern priority (lowest to highest), with the name as a tiebreaker
Attribute-based downsizing	By decreasing order of library cell attribute value, with the name as the tiebreaker
Power-mode downsizing	By decreasing order of instance cell power In total power mode for DMSA, ordering is determined by the -dynamic_scenario scenario.

HyperTrace ECO Fixing

HyperTrace is a technology that accelerates path-based analysis (PBA). The PrimeTime tool can use this technology to speed up ECO fixing when the `-pba_mode` option of the `fix_eco_timing` or `fix_eco_power` command is used. Graph refinement data is automatically updated during the ECO fixing process.

This feature requires a PrimeECO license.

The following topics provide more information:

- [HyperTrace ECO Fixing With fix_eco_timing](#)
- [HyperTrace ECO Fixing With fix_eco_power](#)

HyperTrace ECO Fixing With fix_eco_timing

The `fix_eco_timing` command supports HyperTrace ECO fixing for the following path-based analysis modes:

- `-pba_mode exhaustive`
- `-pba_mode ml_exhaustive`

Internally, the `fix_eco_timing` command calls HyperTrace reporting to accelerate the exhaustive PBA search. As a result, the graph refinement variables used by HyperTrace reporting are also used during ECO fixing:

```
timing_refinement_max_slack_threshold
timing_refinement_min_slack_threshold
timing_refinement_maximum_critical_pin_percentage
```

In particular, you must ensure that the refinement slack thresholds bound the implicit or explicit `-slack_lesser_than` value of the `fix_eco_timing` command.

To use HyperTrace ECO fixing with the `fix_eco_timing` command,

1. Enable HyperTrace ECO fixing:

```
pt_shell> set_app_var eco_enable_graph_based_refinement true
```

Note that this variable setting (for HyperTrace ECO fixing) is independent from the `timing_enable_graph_based_refinement` variable (for HyperTrace reporting).

2. If you are fixing setup violations to a slack value other than zero (positive or negative), set the HyperTrace max-delay slack threshold to your target slack value:

```
pt_shell> # the default max-delay threshold is 0
pt_shell> set_app_var timing_refinement_max_slack_threshold 0.05
...
pt_shell> fix_eco_timing -type setup -slack_lesser_than 0.05
```

3. (Optional) If you are fixing hold violations, note that HyperTrace is disabled for min-delay paths by default because hold violations are typically short paths that do not benefit from HyperTrace acceleration.

If your design has complex violating min-delay logic cones (such as I/O or memory interfaces) that benefit from HyperTrace acceleration, set the min-delay slack threshold to your target slack value:

```
pt_shell> # the default min-delay threshold is 'disabled'
pt_shell> set_app_var timing_refinement_min_slack_threshold 0
...
pt_shell> fix_eco_timing -type hold
```

4. (Optional) Look for the following message to confirm that HyperTrace ECO fixing is being used:

Information: Using HyperTrace to accelerate PBA-based ECO fixing.
(PTECO-116)

HyperTrace ECO Fixing With fix_eco_power

The `fix_eco_power` command supports HyperTrace ECO fixing for the following path-based analysis modes:

- `-pba_mode path`
- `-pba_mode exhaustive`
- `-pba_mode ml_exhaustive`

The `fix_eco_power` command incorporates HyperTrace algorithms and data directly into its ECO fixing algorithms. As a result, the graph refinement variables used by HyperTrace reporting are not used or needed:

```
timing_refinement_max_slack_threshold
timing_refinement_min_slack_threshold
timing_refinement_maximum_critical_pin_percentage
```

To use HyperTrace ECO fixing with the `fix_eco_power` command,

1. Enable HyperTrace ECO fixing:

```
pt_shell> set_app_var eco_enable_graph_based_refinement true
```

Note that this variable setting (for HyperTrace ECO fixing) is independent from the `timing_enable_graph_based_refinement` variable (for HyperTrace reporting).

2. Run one of the following ECO fixing commands:

```
fix_eco_power -pba_mode path ...
fix_eco_power -pba_mode exhaustive ...
fix_eco_power -pba_mode ml_exhaustive ...
```

3. (Optional) Look for the following message to confirm that HyperTrace ECO fixing is being used:

Information: Running HyperTrace-based ECO fixing (PTECO-115)

The following options of the `fix_eco_power` command are unsupported for HyperTrace ECO fixing:

```
fix_eco_power -start_end_type reg_to_reg
fix_eco_power -pba_path_selection_options
```

Writing Change Lists

A change list describes all the changes made to the design during all ECO performed in the PrimeTime session and tells the physical implementation 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 these arguments:

- `ptsh` – Tcl script for PrimeTime shell (the default)
- `text` – List of changes in descriptive text format
- `dctcl` – Tcl script for Design Compiler
- `icctcl` – Tcl script for IC Compiler or IC Compiler II
- `eco` – Binary script for the PrimeTime `read_eco_changes` command
- `aprtcl` – Tcl script for third-party place-and-route tools

After completing an ECO run, you can replay the scripts in tools such as IC Compiler II to implement the changes as part of the Synopsys ECO flow. You can also replay the changes in different PrimeTime sessions, which can be useful for assessing the ECO results in different scenarios.

Replaying an ECO Change List in PrimeTime

To write and replay ECO changes in the PrimeTime tool, follow this recommended flow:

1. In the original PrimeTime session, generate a change list file in the `eco` binary format:

```
write_changes -format eco -output change_list_file
```

2. In a new or restored PrimeTime session, check the LEF/DEF files with the `check_eco` command, and replay the ECO changes with the `read_eco_changes` command:

```
restore_session ...
set_eco_options ...
check_eco
read_eco_changes change_list_file
```

3. Assess the ECO results, and run addition ECO commands if needed.

4. Generate a change list file for implementation:

```
write_changes -format icctcl new_change_list_file
```

Replaying Block-Level ECO Changes in a Top-Level Run

In hierarchical design methodologies, ECO operations can be performed at both the block level and top level. You can read ECOs from previous block-level runs at the top level, allowing blocks to be ECO-fixed by different teams, possibly using different PrimeTime versions, and then combined at the top level.

At the block level, perform the ECO and save the changes:

```
pt_shell> write_changes -format eco -output changesBlkA
```

At the top level, read in the changes:

```
pt_shell> read_eco_changes -design_name BlkA changesBlkA
```

The module name argument BlkA is the Verilog module name of the block, or it can be the reference name for a multiply instantiated module (MIM). For a MIM, the changes are applied to all instances in the MIM group.

Writing ECO Change Lists for Third-Party Place-and-Route Tools

In a physically aware ECO flow, you can write change lists for third-party place-and-route tools by using the `-format aprtcl` option of the `write_changes` command.

The `aprtcl` format (short for Automatic Place-and-Route) writes the changes as a set of pseudo-Tcl commands that are not specific to a particular tool. These commands can be parsed and incorporated into third-party place-and-route tools.

[Table 64](#) lists the pseudo-Tcl commands specific to this format. Commands not shown in the table are equivalent to the `-format icc2tcl` format.

Table 64 Pseudo-Tcl Commands Used by the write_changes -format aprtcl Command

ECO operation	Pseudo-Tcl command syntax
Insert buffer (pin-based)	<pre>insert_buffer new_lib_cell -new_net_names net_names -new_cell_names buff_names [-location {x_y}] # buffers [-location {x1_y1_x2_y2}] # inverter pairs [-orientation dir] [-inverter_pair] {pin_or_port_list}</pre>

Table 64 Pseudo-Tcl Commands Used by the write_changes -format aprtcl Command (Continued)

ECO operation	Pseudo-Tcl command syntax
Insert buffer (on-route)	<pre>insert buffer new_lib_cell -on_route -new_net_names net_names -new_cell_names buff_names [-location {x y}] # buffers [-location {x1 y1 x2 y2}] # inverter pairs [-orientation dir] [-inverter_pair] [-route_cut_location {x y}] # buffers [-route_cut_location {x1 y1 x2 y2}] # inverter pairs {pin_or_port_list}</pre>
Resize cell	<pre>size_cell leaf_cell new_lib_cell [-overlap] [-location {x y}] [-orientation dir]</pre>
Create cell (for load cells)	<pre>create_cell load_cell_name new_lib_cell [-location {x y}] [-orientation dir]</pre>
Connect net (for load cells)	<pre>connect_net net_name pin_or_port_list</pre>
Remove buffer	<pre>remove_buffer cell_name</pre>

For details, see the `write_changes` man page.

[Example 89](#) shows an example change list that creates a buffer tree to drive four load pins, and adds a load cell to fix a hold violation. The buffer insertion order and load pin lists re-create the required tree structure.

Example 89 Example Change List From write_changes -format aprtcl

```
#####
# Change list, formatted for Third Party Compiler
#
#
#
# aprtcl_file_syntax_version : 1.0
```

```

#
#####
##### current_instance
##### current_instance {blk/subblk3}
##### insert_buffer -on_route BUFFD4 \
#####   -new_net_names {net1} -new_cell_names {BUF1} \
#####   -location {x1 y1} -route_cut_location {x1' y1'} \
#####   {sink1/I sink2/I sink3/I sink4/I}
##### insert_buffer -on_route BUFFD2 \
#####   -new_net_names {net2} -new_cell_names {BUF2} \
#####   -location {x2 y2} -route_cut_location {x2' y2'} \
#####   {sink1/I sink2/I sink3/I}
##### insert_buffer -on_route BUFFD4 \
#####   -new_net_names {net4} -new_cell_names {BUF4} \
#####   -location {x4 y4} -route_cut_location {x4' y4'} \
#####   {sink4/I}
##### insert_buffer -on_route BUFFD4 \
#####   -new_net_names {net3} -new_cell_names {BUF3} \
#####   -location {x3 y3} -route_cut_location {x3' y3'} \
#####   {sink1/I sink2/I}
##### insert_buffer -on_route BUFFD2 \
#####   -new_net_names {net5} -new_cell_names {BUF5} \
#####   -location {x5 y5} -route_cut_location {x5' y5'} \
#####   {sink3/I}
create_cell {U_LOAD_CAP_CELL_1} {CLOAD1_LVT} \
#####   -location {150.6320 61.8640} -orientation N
connect net {design ack signal} {U LOAD CAP CELL 1}

```

The header of the change list file includes a version string so that external parsers can adapt to changes in the format over time. Currently, the only possible version value is 1.0. It is recommended that parser scripts check this version for an expected value, so that future syntax upgrades do not cause unexpected behavior.

Implementing ECO Changes in Synopsys Layout Tools

Note:

If you have a PrimeECO license, you can use the PrimeECO tool to perform ECO physical implementation, parasitic extraction, and signoff timing analysis in a single-shell environment. For details, see the *PrimeECO User Guide*.

To implement the ECO changes using discrete Synopsys tools, follow these steps:

1. In PrimeTime, generate the change list file by executing this command:

```
write changes -format icctcl new change list file
```

2. In the IC Compiler II tool, source the change list file, then place and legalize the ECO cells with the `place eco cells` command, insert filler cells with the

`create_stdcell_fillers` command, and update the routing with the `route_eco` command.

In the IC Compiler tool, read the change list file using the `eco_netlist` command, then execute the `place_eco_cells`, `legalize_placement`, and `route_zrt_eco` commands.

3. Perform parasitic data extraction using the StarRC tool.
4. Read the updated full netlist and parasitics files into the PrimeTime tool.
5. Rerun timing analysis and validate the QoR.
6. Repeat additional ECO fixing iterations as needed.

Incremental ECO Flow Using Synopsys Tools

Note:

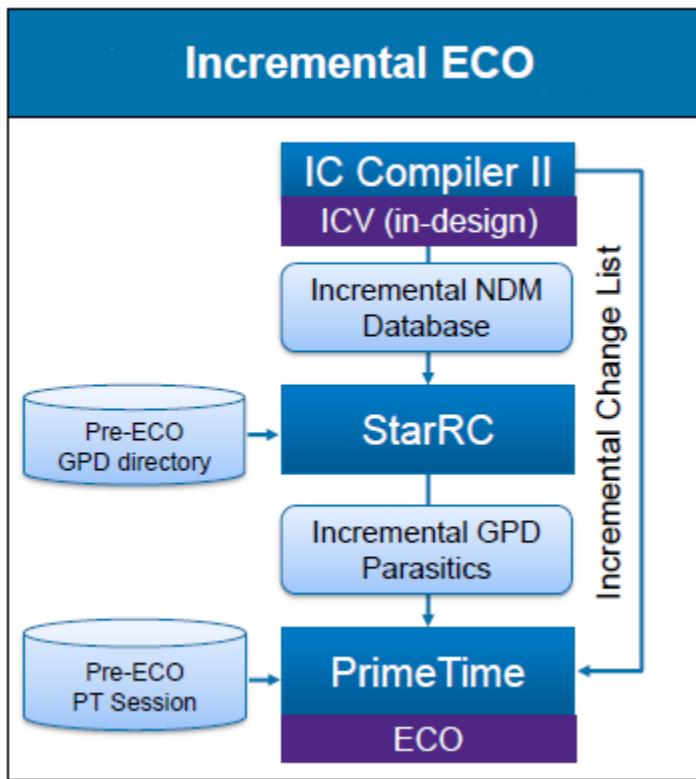
If you have a PrimeECO license, you can use the PrimeECO tool to perform ECO physical implementation, parasitic extraction, and signoff timing analysis in a single-shell environment. For details, see the *PrimeECO User Guide*.

An alternative to the full-netlist, full-parasitics ECO flow is the Synopsys incremental ECO flow. In this flow,

- The IC Compiler II tool implements the ECO changes and writes out only the incremental changes instead of the full design.
- The StarRC tool extracts only the incremental physical changes and writes out an incremental-change parasitic data file.
- The PrimeTime tool applies the incremental parasitic data changes to a restored PrimeTime session and verifies the timing of the modified design.

The incremental ECO flow is summarized in the following diagram.

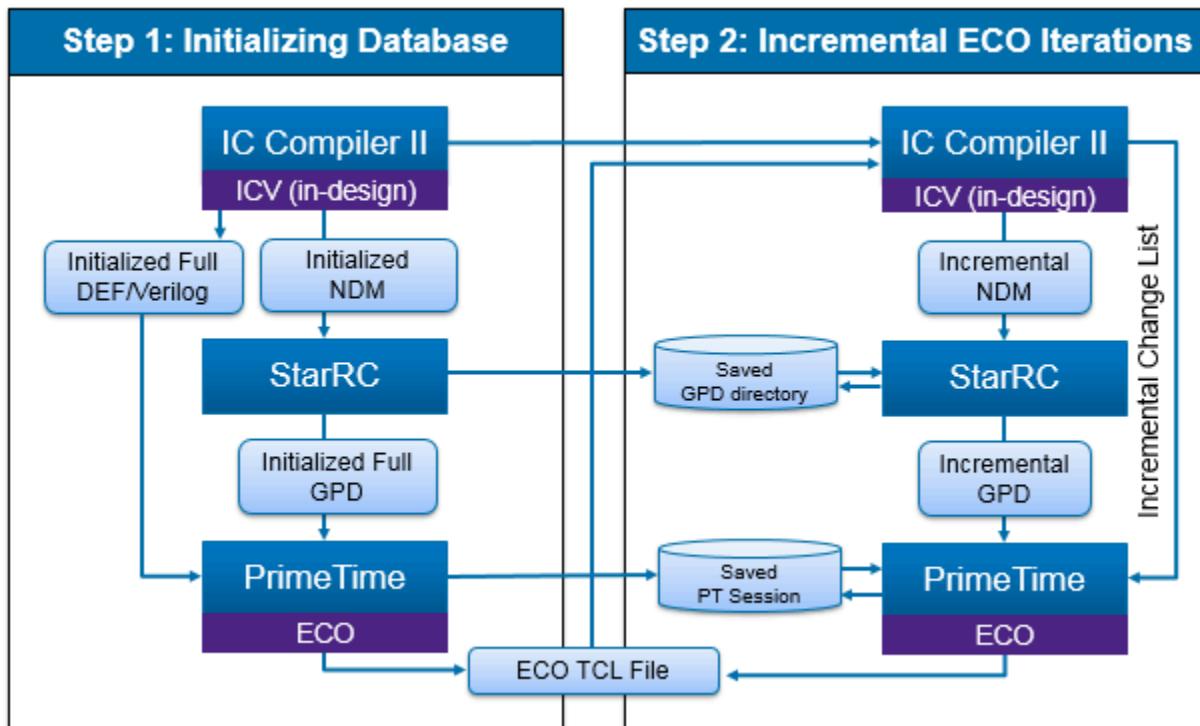
Figure 372 Synopsys Incremental ECO Flow



By extracting and analyzing only the incremental changes instead of the full design, the runtime and turnaround time are reduced for each ECO iteration. Note that final signoff requires a full extraction and full timing analysis.

The Synopsys incremental ECO flow operates on the incremental changes in each ECO loop, using a two-step process summarized in the following figure.

Figure 373 Synopsys Incremental ECO Flow Steps



These are the actions taken in the two steps:

- Establish the initial reference design data using all three tools: IC Compiler II, StarRC, and PrimeTime; and enable tracking of incremental changes in the IC Compiler II tool:

```
icc2_shell> record_signoff_eco_changes -init -def
```

- Perform ECOs in PrimeTime and record the ECO changes in the IC Compiler II tool:

```
icc2_shell> record_signoff_eco_changes -start
icc2_shell> ... Perform ECO implementation ...
icc2_shell> record_signoff_eco_changes -stop
```

At the end of the ECO implementation step, IC Compiler II generates:

- An incremental IC Compiler II NDM database for StarRC extraction
- An incremental change list for PrimeTime to use in the next iteration

See Also

- [Initialize the Incremental ECO Flow](#)
- [Incremental ECO Iteration](#)
- [Hierarchical Incremental ECO Flow](#)

Initialize the Incremental ECO Flow

The incremental ECO flow begins with initializing the database on which ECO operations will be performed.

1. In the IC Compiler II tool, open the library and copy the block on which to perform ECO changes, and initialize the ECO database:

```
icc2_shell> open_lib Design.nlib
icc2_shell> copy_block -from_block Block -to_block Block_pre_eco
icc2_shell> open_block Block_pre_eco
icc2_shell> record_signoff_eco_changes -init -def
```

The `record_signoff_eco_changes` command saves the design, writes the Verilog netlist, and writes the DEF design information. In step 3, the PrimeTime tool uses this information in the incremental ECO flow.

2. In the StarRC tool, perform parasitic extraction in ECO mode and incremental netlist mode by including commands similar to the following in the StarRC command file:

```
NDM_DATABASE: Design.nlib
BLOCK: Block_pre_eco
ECO_MODE: YES
GPD: Block_pre_eco.gpd
```

3. In the PrimeTime tool, execute a baseline full timing update using the full design netlist and full parasitics, save the session, and then perform the initial ECO:

```
pt_shell> read_verilog ndm_path/design.full.v.gz
pt_shell> link_design
pt_shell> read_parasitics Block_pre_eco.gpd
pt_shell> read_sdc Block.sdc
pt_shell> update_timing -full
pt_shell> save_session eco_session1

pt_shell> set_eco_options -physical_design_path \
  ndm_path/design.full.def.gz ...
pt_shell> fix_eco_timing ...
pt_shell> fix_eco_drc ...
pt_shell> write_changes pt-eco_incl.tcl

(ndm_path = Design.nlib/Block_pre_eco/attach/design.signoff.eco.data)
```

Incremental ECO Iteration

After the flow has been initialized, you can perform the incremental ECO iterations:

1. In the IC Compiler II tool, implement the ECO changes written out by the PrimeTime tool and track the incremental changes:

```
icc2_shell> open_lib Design.nlib
icc2_shell> copy_block -from_block Block_pre_eco -to_block Block_eco1
icc2_shell> open_block Block_eco1
icc2_shell> record_signoff_eco_changes -start -input pt-eco_incl.tcl

... Perform ECO implementation using minimal physical impact flow ...

icc2_shell> record_signoff_eco_changes -stop -def
```

The `record_signoff_eco_changes` command saves the design, writes the Verilog netlist, and writes the DEF design information. In step 3, the PrimeTime tool uses this information in the incremental ECO flow.

2. In the StarRC tool, perform incremental parasitic extraction of the modified design by using commands similar to the following in the StarRC command file:

```
NDM_DATABASE: Design.nlib
BLOCK: Block_eco1
ECO_MODE: YES
GPD: Block_pre_eco.gpd
```

3. In the PrimeTime tool, read in the incremental netlist changes and incremental parasitics for the modified design and check to see if it now meets all timing constraints.

```
pt_shell> restore_session eco_session1
pt_shell> read_eco_changes ndm_path/design.incr.pt
pt_shell> read_parasitics -eco Block_eco1_inc.gpd
pt_shell> update_timing      # incremental update performed by default
pt_shell> save_session eco_session2
pt_shell> report timing ...

... Assess timing to determine whether another ECO is needed ...
... If so, perform the next ECO iteration, then go back to step 1 ...

pt_shell> fix_eco_timing ...
pt_shell> fix_eco_drc ...
pt_shell> write_changes pt-eco_inc2.tcl
```

Repeat steps 1 through 3 using ECO data from the previous iteration, until you achieve timing closure. For final chip signoff, perform a final full-chip extraction and timing analysis.

For more information, see “[Incremental ECO Flow](#)” in the *StarRC User Guide and Command Reference* and “[Incremental Signoff ECO Flow](#)” in the *IC Compiler II Implementation User Guide*.

Hierarchical Incremental ECO Flow

The incremental ECO flow for a hierarchical design is similar to the flow for a flat design. Typically, in the IC Compiler II and StarRC tools, you use a separate run for each block and for the top level, whereas in the PrimeTime tool, you assemble all the blocks and top together and analyze them as a unit. Therefore, you need to gather the separate incremental ECO data from the other tools for timing analysis in the PrimeTime tool.

Start by initializing the top-level and block-level designs as described previously in “[Initialize the Incremental ECO Flow](#).“ To perform an ECO iteration hierarchically, use scripts similar to the following for the top-level design and each block-level design.

IC Compiler II

```
# Top Level

open_lib Top_design.nlib
copy_block -from_block Top_pre_eco -to_block Top_eco1
open_block Top_eco1
record_signoff_eco_changes -start -input pt-eco_top_incl.tcl

# Minimum physical impact flow for ECO (place_eco_cells + route_eco)
# Standard filler cell re-insertion
# Incremental Metal Fill

set_app_options -list { signoff.create_metal_fill.flat true \
    signoff.create_metal_fill.user_defined_options \
    { -D NUM_SEGMENTS=4 -turbo ... }
    signoff.create_metal_fill.auto_eco_threshold_value 99 }
signoff_create_metal_fill -auto_eco true
record_signoff_eco_changes -stop -def

# Block Level, Block 1

open_lib Block1_design.nlib
copy_block -from_block Block1_pre_eco -to_block Block1_eco1
open_block Block1_eco1
record_signoff_eco_changes -start -input pt-eco_Block1_incl.tcl

# MPI flow for ECO implementation (place_eco + route_eco )
# Standard filler cell re-insertion
# Incremental Metal Fill

set_app_options -list {
    signoff.create_metal_fill.flat true \
```

```

signoff.create_metal_fill.user_defined_options \
{ -D NUM SEGMENTS=4 -turbo ...}
signoff.create_metal_fill.auto_eco_threshold_value 99 }
signoff_create_metal_fill -auto_eco true
record_signoff_eco_changes -stop -def
...

```

StarRC

Top Level

```

NDM_DATABASE: Top_design.nlib
BLOCK: Top_eco1
ECO_MODE: YES
NETLIST_INCREMENTAL: YES
GPD: Top_eco1_inc.gpd

```

Block Level, Block 1

```

NDM_DATABASE: Block1_design.nlib
BLOCK: Block1_eco1
ECO_MODE: YES

```

...

PrimeTime

Top and Block Levels

```

restore_session eco_session1      # restore initialization session
read_eco_changes $ndm_path_Top/design.incr.pt
read_eco_changes $ndm_path_Blk1/design.incr.pt
read_eco_changes $ndm_path_BlkN/design.incr.pt
read_parasitics -eco Top_eco1_inc.gpd
read_parasitics -eco Blk1_eco1_inc.gpd
read_parasitics -eco BlkN_eco1_inc.gpd

update_timing      # incremental update performed by default
save_session eco_session2        # save session for next iteration
report_timing ...

... Assess timing to determine whether another ECO is needed ...
... If so, perform the next ECO iteration, then go back to step 1 ...

fix_eco_timing ...
fix_eco_drc ...
write_changes pt-eco_inc2.tcl

($ndm_path_Top =
  Top_design.nlib/Top_eco1/attach/design.signoff.eco.data)
($ndm_path_Blk1 =
  Block1_design.nlib/Block1_eco1/attach/design.signoff.eco.data)

```

```
(ndm_path_BlkN =
  BlockN_design.nlib/BlockN_ecol/attach/design.signoff.eco.data)
```

The `read_eco_changes` and `read_parasitics` commands with the `-eco` option automatically apply the hierarchical information contained in the incremental change files. You do not need to use the `current_instance` command to change the hierarchical levels or the `-path` option of the `read_parasitics` option to read the parasitic data files.

PrimeTime With DMSA

For analysis using DMSA, start with an initial run similar to “[Initialize the Incremental ECO Flow](#)” using a standard DMSA script. To perform an ECO iteration, use the `create_scenario` command with `-image` option to read in the incremental information hierarchically at the worker level, and perform an incremental timing update.

```
# Top and Block Levels
# restore initialization sessions
create_scenario -name $scenario \
  -image pre_session_dmsa/${scenario}_pre_eco_session
remote_execute {
  read_eco_changes ndm_path_Top/design.incr.pt
  read_eco_changes ndm_path_Blk1/design.incr.pt
  read_eco_changes ndm_path_BlkN/design.incr.pt

  read_parasitics -eco Top_eco1_inc.gpd
  read_parasitics -eco Blk1_eco1_inc.gpd
  read_parasitics -eco BlkN_eco1_inc.gpd

  update_timing    # incremental update performed by default
}
save_session eco_session2_dmsa
report_timing ...

... Assess timing to determine whether another ECO is needed ...
... If so, perform the next ECO iteration, then go back to step 1 ...

fix_eco_timing ...
fix_eco_drc ...
write_changes pt-eco_inc2_dmsa.tcl

(ndm_path_Top =
  Top_design.nlib/Top_eco1/attach/design.signoff.eco.data)
(ndm_path_Blk1 =
  Block1_design.nlib/Block1_eco1/attach/design.signoff.eco.data)
(ndm_path_BlkN =
  BlockN_design.nlib/BlockN_eco1/attach/design.signoff.eco.data)
```

ECO Flow Using Reduced Resources

Note:

The reduced resource ECO feature is deprecated, and will no longer work in a future release. If you attempt to use it, the tool issues a warning:

Warning: The Reduced Resource ECO feature is deprecated and will be obsolete in a future release. This feature is superseded by the Hybrid view feature of PrimeECO. (PTECO-042)

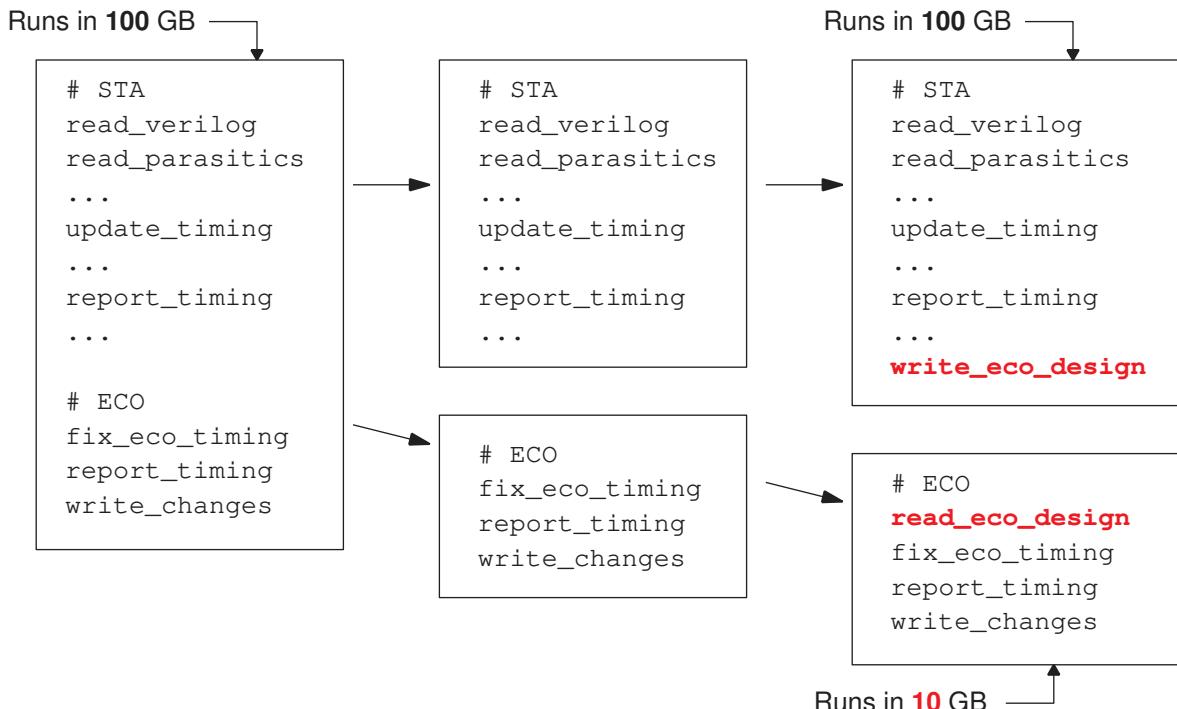
To reduce resource requirements during ECO fixing, use the Hybrid timing view feature in the PrimeECO tool instead.

The PrimeTime tool offers an optional method to significantly reduce the memory and runtime requirements for performing ECOs. This method is especially helpful for reducing the memory footprint when running ECOs under multiple scenarios using distributed multi-scenario analysis (DMSA).

This method is based on generating and using a smaller representative design for ECO operations. The new design written, called an *ECO design*, contains only the design data related to the constraint violations that need to be fixed, for example, setup and hold constraints. The ECO command (for example, `fix_eco_timing`) uses the reduced design to fix violations relative to the specified constraints.

The following diagram shows the reduced-resource ECO flow.

Figure 374 Single-Scenario Reduced-Resource ECO Flow



The first box shows the original full-memory ECO flow in which static timing analysis (STA) is followed by ECO timing fixing. To use the reduced-resource ECO flow, you divide the original script into two parts, one for STA and the other for ECO. Then you modify the STA script to end with the `write_eco_design` command, which creates the reduced ECO design; and modify the ECO script to start with the `read_eco_design` command, which reads in the reduced ECO design and uses it for the ECO operation.

If the STA script requires 100 GB of memory to run, the ECO script might require only 10 GB to run. This memory reduction can be very valuable if you need to run many ECOs for different types of fixing or for multiple scenarios.

When you use the `write_eco_design` command, you can specify the types of timing or design rule constraints that need to be optimized. The more types of constraints considered, the larger the generated ECO design. Also, you can optionally restrict the list of path endpoints considered for generating the ECO design.

The reduced-resource ECO flow is intended only for ECOs, not for final signoff of the whole design, because the reduced ECO designs might not fully reflect all possible constraint violations. You should run a final full-design analysis for signoff.

Running Multiple Scenarios in the Same Session

To run multiple ECOs for different scenarios in the same session after static timing analysis, use the `write_eco_design` command to write out the ECO designs and the `stop_hosts` command to specify the new host options for the ECO runs. For example,

```
# Create DMSA scenarios
create_scenario ...
# Configure DMSA hosts
set_host_options -num_processes 20 \
    -submit_command "bsub [mem=100G] ..."
start_hosts
current_session -all
# STA
report_global_timing

# Create ECO design
write_eco_design

# Stop current hosts and launch new smaller hosts
stop_hosts
remove_host_options
set_host_options -submit_command "bsub [mem=10G] ..." #10 GB for ECOS
start_hosts
# Read ECO design
read_eco_design

# ECO
fix_eco_timing
```

The commands shown in red are inserted into the script to take advantage of the reduced-resource ECO feature. The memory footprint of each ECO run is reduced from 100 GB to 10 GB.

Running Multiple Saved Sessions

It is a common practice to run and save multiple PrimeTime sessions to handle different PVT corners, and then later restore the sessions and perform ECOs on them from a manager session. For example,

```
# -----
# STA for corner 1
read_verilog ...
read_parasitics ...
...
report_timing
...
# Save session
save_session ./corner01
# -----
```

Chapter 21: ECO Flow

ECO Flow Using Reduced Resources

```

# STA for corner 2
read_verilog ...
read_parasitics ...
...
report_timing
...
# Save session
save_session ./corner02
# ----

# ... STA for corners 3-16 ...

# ----

# Create scenarios from saved sessions 1-16
create_scenario -image ./corner01
create_scenario -image ./corner02
...
create_scenario -image ./corner16

# Configure DMSA hosts
set_host_options -num_processes 16 \
    -submit_command "bsub [mem=100G] ..."      # sixteen 100-GB
start_hosts
current_session -all

# ECO
fix_eco_timing
report_timing

```

To take advantage of the reduced-resource ECO feature in this flow, in each STA scenario session, use the `eco_save_session_data_type` variable to save additional endpoint information inside the session. The `write_eco_design` uses this information to create the DMSA ECO design.

The following example creates ECO designs using fewer resources for timing-based ECO:

```

# -----
# STA for corner 1
read_verilog ...
read_parasitics ...
...
report_timing
...
# Save ECO data
set eco_save_session_data_type timing
# Save session
save_session ./corner01
# ----
# STA for corner 2
read_verilog ...
read_parasitics ...

```

```

...
report_timing
...
# Save ECO data
set eco_save_session_data_type timing
# Save session
save_session ./corner02
# -----
# ... STA for corners 3-16 ...
# -----

# Create scenarios from saved sessions 1-16
create_scenario -image ./corner01
...
create_scenario -image ./corner16

# Configure DMSA hosts
set_host_options -num_processes 2          # two 100-GB
    -submit_command "bsub [mem=100G] ..."   # subordinate processes
start_hosts
current_session -all

# Create ECO designs from saved sessions 1-16
write_eco_design
stop_hosts
remove_host_options

# Load ECO design
set_host_options -num_processes 16 \        # sixteen 10-GB
    -submit_command "bsub [mem=10G] ..."   # subordinate processes
start_hosts
read_eco_design

# ECO
fix_eco_timing
report_timing

```

The DMSA run uses the saved sessions from each scenario analysis. After that, the `write_eco_design` command finds the ECO data from the saved sessions, restores those sessions, and extracts the ECO designs. The `read_eco_design` command launches 16 smaller processes on 10-GB machines to perform the ECOs.

This example uses two 100-GB machines to create ECO designs serially, then later 16 smaller 10-GB machines to run distributed ECOs using reduced ECO designs. Compare this to the full-design flow, which uses 16 large 100-GB machines to run the same ECOs.

For more information, see the man pages for the `write_eco_design` and `read_eco_design` commands, and for the `eco_save_session_data_type` variable.

Using Negative Timing Margins to Restrict Testing

You can specify a large negative margin for hold testing under worst-case conditions, and a large negative margin for setup testing under best-case conditions, to test only the critical constraint in each scenario and save runtime. For example,

```
set scenario ${mode}_${corner}

# Check setup timing, ignore hold timing in mode_worst scenario
if { $scenario == "mode_worst" } {
    read_parasitics -format SPEF ./worst.spf
    set_eco_options \
        -timing_hold_margin -100.0 \
        -timing_setup_margin 0.050
}
# Check hold timing, ignore setup timing in mode_best scenario
if { $scenario == "mode_best" } {
    read_parasitics -format SPEF ./best.spf
    set_eco_options \
        -timing_hold_margin 0.000 \
        -timing_setup_margin -100.0
}
```

Reduced Resource Non-DMSA ECO Flow

You can run the `write_eco_design` command in standalone (non-DMSA) sessions and then combine the failing endpoints from these sessions into a single ECO design.

To use this alternative flow, set the `eco_save_session_data_type` variable in each session to specify the types of ECO data saved in the session directory by the `save_session` command. Then use the `write_eco_design` command with the `-merge_sessions` option to specify the list of session directories containing the failing endpoint data.

The following example demonstrates the non-DMSA `write_eco_design` merging flow.

```
# Standalone STA script for corner FF running on Host1
read_verilog ...
update_timing ...
...
set_app_var eco_save_session_data_type timing
save_session $my_sess/STA_SESS_FF

# Standalone STA script for corner SS running on Host2
read_verilog ...
update_timing ...
...
set_app_var eco_save_session_data_type timing
save_session $my_sess/STA_SESS_SS
```

```
# Script to write ECO design for FF running in Host1
restore_session $my_sessions/STA_SESSIONS_FF
set my_sess_list "$my_sess/STA_SESS_FF $my_sess/STA_SESS_SS"
write_eco_design $rreco_db/FF -merge_sessions $my_sess_list

# Script to write ECO design for SS running in Host2
restore_session $my_sessions/STA_SESSIONS_SS
set my_sess_list "$my_sess/STA_SESS_FF $my_sess/STA_SESS_SS"
write_eco_design $rreco_db/SS -merge_sessions $my_sess_list
```

The rest of the flow is the same as the full-DMSA flow. The `read_eco_design` command in the DMSA manager detects two ECO designs in the shared directory, launches two worker processes to read the ECO designs, and runs the `fix_eco_timing` or `fix_eco_drc` command:

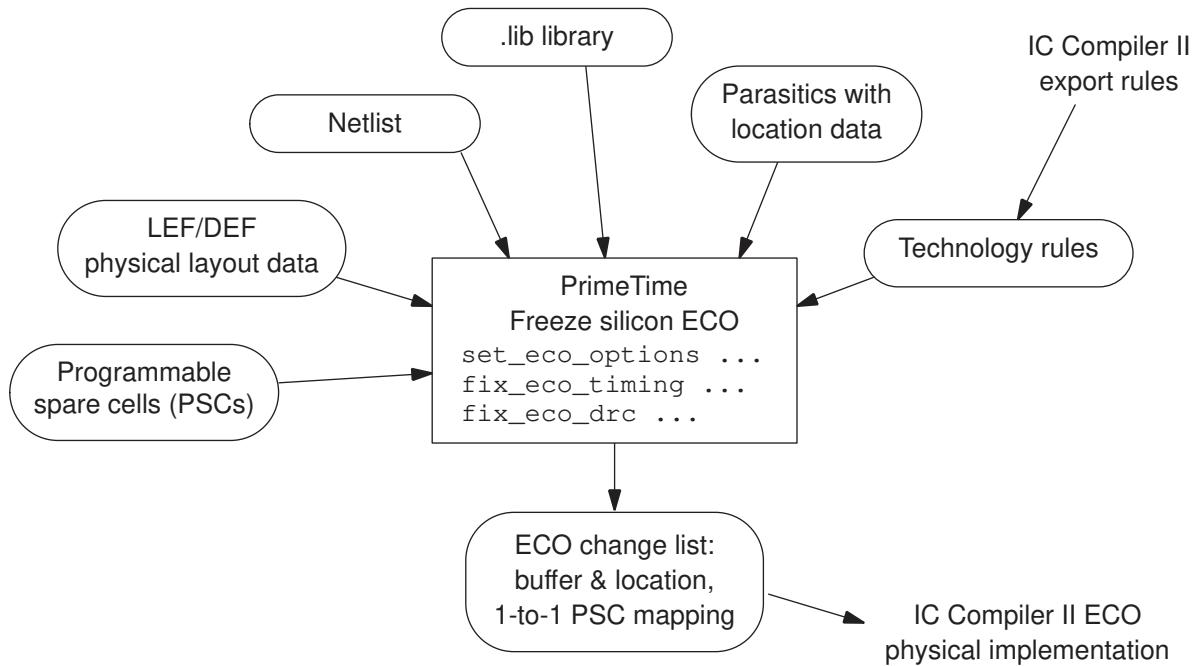
```
# DMSA script to read the ECO design for SS and FF scenarios
read_eco_design $rreco_db
...
```

Freeze Silicon ECO Flow

In the freeze silicon flow, the IC Compiler II tool implements ECOs by modifying only the metal and via layers, leaving the silicon-level layers unchanged, thereby saving the time and expense of generating new masks for the implant, diffusion, and poly layers. This flow requires the placement of spare cells throughout the chip layout, allowing the IC Compiler II tool to implement ECOs by connecting the spare cells into the design using metal routes.

The following diagram shows the types of data typically used in the freeze silicon flow.

Figure 375 Freeze Silicon Data Flow



The following figure shows the commands typically used in the IC Compiler II and PrimeTime tools in the freeze silicon flow.

Figure 376 IC Compiler II and PrimeTime Commands in the Freeze Silicon Flow

IC Compiler II ECO preparation

```
add_spare_cells -cell_name ... -lib_cell ... -num_cells ...
create_stdcell_fillers -lib_cells ...
```

PrimeTime freeze silicon ECO

```
set_eco_options ...
fix_eco_drc -type ... -physical_mode freeze_silicon
fix_eco_timing -type ... -physical_mode freeze_silicon
write_changes -format icctcl -output pt_eco.tcl
```

IC Compiler II freeze silicon ECO

```
set_app_options -name design.eco_freeze_silicon_mode -value true
source pt_eco.tcl
set_app_options -name design.eco_freeze_silicon_mode -value false
connect_pg_net -automatic
# Set up ECO-to-PSC mapping rules here if not already done
set_app_options -category route.global -list {timing_driven false}
set_app_options -category route.track -list {timing_driven false}
set_app_options -category route.detail -list {timing_driven false}
check_routes
route_eco -reroute modified_nets_only -open_net_driven true
```

Preparing for the Freeze Silicon ECO Flow

The following requirements apply to the freeze silicon ECO flow:

- Spare cells must be available in the chip layout. For details, see “ECO Flow” and “Freeze Silicon ECO Flow” in the *IC Compiler II Implementation User Guide*.
- The spare cells must be defined in the .lib library.
- The library-based and instance-specific layout properties of the spare cells must be available in a set of LEF/DEF files, which you can generate from the IC Compiler II tool.

- The applicable spacing rules used for cell placement must be available in a file. You can provide this file in any of the following ways:
 - In the IC Compiler II tool, use the `export_advanced_technology_rules` command, which writes out the technology rules in a binary encrypted format.
 - Write a script file containing a set of `set_lib_cell_spacing_label` and `set_spacing_label_rule` commands that define the rules.
 - Provide the rules in Synopsys technology file format, as described in the *Synopsys Technology File and Routing Rules Reference Manual*.

In the PrimeTime tool, use the `set_eco_options` command to choose the freeze silicon ECO flow, specify the spare cell names, specify the spacing-rule constraint file, and specify the physical design (LEF/DEF) files. For example,

```
pt_shell> set_eco_options \
  -programmable_spare_cell_names {PSC1 PSC2 PSC3 PSC4 PSC5 PSC6} \
  -physical_lib_constraint_file ../tech_data/my_spacing_rules \
  -physical_tech_lib_path {..phy_data/my_tech.lef} \
  -physical_lib_path {..phy_data/my_lib.lef ..phy_data/my_design.def} \
  -log_file my_lef_def.log \
  ...
```

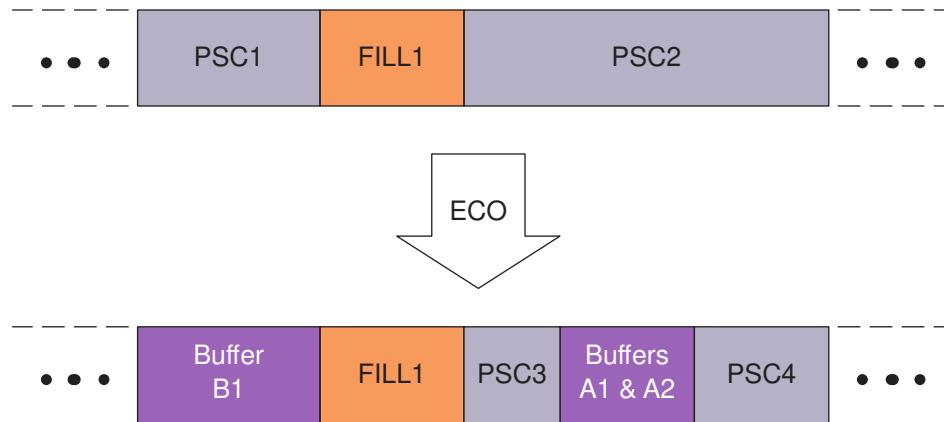
The `-log_file` option causes the tool to write out LEF/DEF processing messages (errors and warnings) into the specified log file.

Programmable Spare Cells

The freeze silicon ECO flow supports the use of *programmable spare cells* (PSCs). Instead of performing only a single fixed logic function, a PSC can be programmed to perform any of various logic functions. The cell is made up of transistor-level parts that can be “wired up” in a variety of ways, allowing great flexibility in implementing ECO changes. If a particular ECO uses only part of a PSC, the remaining part of the cell is still available for later ECOs.

The following figure shows how an ECO replaces spare cells with functional cells. At the top, you can see part of a standard-cell row in the chip layout after cell placement, as displayed in the IC Compiler II tool, before applying an ECO. The boxes labeled PSC1 and PSC2 are unconnected PSC-type spare cells. Because this is a freeze silicon flow, no cells can be moved, but the PSC cells can be wired to perform different logical functions.

Figure 377 Freeze Silicon Site Row Cell Replacement



As a result of applying an ECO, the PSC1 spare cell is replaced entirely by a large buffer, B1. The space originally occupied by the PSC2 spare cell is used to implement two buffers, A1 and A2. Only part of this space is used, so the tool backfills the remaining available parts of the original PSC2 spare cell with two new spare cells, PSC3 and PSC4, which are available to be used in later ECOs.

LEF/DEF Descriptions of PSCs

The library-based and instance-specific layout properties of the spare cells must be available in a set of LEF/DEF files, which you can generate from the IC Compiler II tool by using the `write_lef` and `write_def` commands.

The description of a PSC in a LEF file is similar to the description for an ordinary fill cell. However, for ordinary fill cells, the `MACRO` section must contain `CLASS CORE FEEDTHRU` or `CLASS CORE SPACER`; this is optional for PSCs. For example,

```
# LEF Programmable Spare Cell construct, cell name = e0hd_cap12x
MACRO e0hd_cap12x
  CLASS CORE ;
  ORIGIN 0 0 ;
  SIZE 0.816 BY 0.42 ;
  ...
END e0hd_cap12x
```

In the description of a PSC instance in a DEF file, the `COMPONENTS` section must contain `SOURCE DIST` (because the instance is not defined in the netlist) and `PLACED` (because it has a physical location); and it cannot contain `FIXED` or `COVER`. For example,

```
# DEF instantiation of Programmable Spare Cell
-xofiller!ECO_DECAP_FILLER_!e0hd_cap12x!179638
e0hd_cap12x + SOURCE DIST + PLACED (264300 624100) N ;
```

You can check the validity of LEF/DEF descriptions of PSCs by using the `-log_file` option of the `set_eco_options` command. For example,

```
set_eco_options \
    -programmable_spare_cell_names {PSC1 PSC2 PSC3 PSC4 PSC5 PSC6} \
    -physical_lib_constraint_file my_spacing_rules \
    -physical_tech_lib_path {../phy_data/my_tech.lef} \
    -physical_lib_path {my_lib.lef my_design.def} \
    -log_file lef_def_info.log
fix_eco_timing -type hold -physical_mode freeze_silicon \
    -buffer_list $hold_buffer_list
```

After you run the `fix_eco_timing` command, the `lef_def_info.log` file shows PSC identification messages:

```
...
Information: Reading LEF file my_lib.lef
Information: Information: Identified programmable spare cell PSC1 at
lineNumber 1238.
Information: Information: Identified programmable spare cell PSC2 at
lineNumber 1279.
...
Information: Reading DEF file my_design.def
Physical Information Summary:
Identified programmable spare cells PSC1 ...
...
```

Exporting DEF and Advanced Spacing Rules From IC Compiler II

To generate the required DEF files and technology spacing rules from the IC Compiler II tool, you can use the `write_def` and `export_advanced_technology_rules` commands. At the `icc2_shell` prompt, execute a script similar to the following:

```
set myblocks "block1 block2 block3"
set mylib "design_A"
open_lib $mylib                                # include all required libs
foreach blockx $myblocks {
    open_block $blockx                          # open the design
    check_legality
    link_force
    write_def $blockx.def                      # write block DEF file
    export_advanced_technology_rules $blockx_rules.tcl   # and tech rules
    close_blocks $blockx
}
```

Freeze Silicon ECO in PrimeTime

To perform a freeze silicon ECO in the PrimeTime tool, use the following commands:

- `set_eco_options` to specify the ECO parameters
- `fix_eco_timing` or `fix_eco_drc` command to generate the ECO changes
- `write_changes` to write out the ECO change script

The following script performs freeze silicon ECO hold fixing in multiple scenarios.

```
# Create scenarios
...
set_app_var read_parasitics_load_locations true
# Update timing with pin slack and arrival info
remote_execute {
    set_app_var timing_save_pin_arrival_and_slack true;
    update_timing -full }

# Configure Physical Information needed for ECO
remote_execute { set lef_files [glob ./*.lef]
set_eco_options \
    -physical_tech_lib_path $tech_lef_files \
    -physical_lib_path $lef_files \
    -physical_design_path ./design.def.gz \
    -physical_lib_constraint_file $rule_files \
    -programmable_spare_cell_names $pscs \
    -log_file ./lef_def.log }

# Alternative to LEF/DEF: direct access to IC Compiler II data
# set_eco_options \
#     -physical_icc2_lib $ic平2_lib_path \
#     -physical_icc2_blocks $ic平2_blocks \
# ...
# Fix hold violations
set buffer_list { BUF_1 BUF_2 ... BUF_N}
set_app_var eco_report_unfixed_reason_max_endpoints 50
fix_eco_timing -type hold -physical_mode freeze_silicon \
    -buffer_list $buffer_list -verbose

# Write Changes
remote_execute { write_changes -format icctl -output pt_eco.tcl }
```

In the ECO change file written by the `write_changes` command, the `insert_buffer` commands cite the new buffer location and the name of the PSC being replaced. For example,

```
insert_buffer [get_pins {U1/z}] e0hd_cap12x \
    -new_net_names {eco_hold_net1} \
    -new_cell_names {eco_hold_buf1} \
    -location {665.4320 193.6320} -orientation FS
```

```
map_freeze_silicon \
-eco_cell [get_cells -hier " eco_hold_buf1 "] \
-spares_cell xofiller!ECO_DECAP_FILLER_!e0hd_cap12x!179638 \
-filler_map_strategy closest
```

Running ECO Scenarios on Fewer Hosts

You can run the PrimeTime ECO fixing commands in distributed multi-scenario analysis (DMSA) mode even if the number of scenarios exceeds the number of available hosts. To enable this capability, set the `eco_enable_more_scenarios_than_hosts` variable to true.

Minimum Host Requirement

To run DMSA ECO with more scenarios than hosts, 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.

As the number of available hosts decreases, the runtime 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.

Disk Space Requirement

When running DMSA ECO with more scenarios than hosts, 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.

Optimizing DMSA ECO Scripts for Faster Turnaround

When PrimeTime ECO runs with more scenarios than available hosts, the hosts run in scenario swapping mode, where they rotate and swap scenarios one after the other. This results in significant performance degradation.

To minimize scenario swapping and achieve faster runtimes, use these methods:

- Minimize the number of merged reporting commands, such as `report_timing` and `report_analysis_coverage`.
- Reduce the number of `remote_execute` commands and merge them into one `remote_execute` block. Each `remote_execute` command incurs one round of scenario swapping.

Here is a script example for PrimeTime ECO in DMSA:

```
# Requires scenario swapping just one time
remote_execute {
```

```

set_dont_touch $dont_touch_list
set_dont_use $dont_use_list
set eco_alternative_area_ratio_threshold 2
update_timing
}

fix_eco_timing -type setup

# Requires scenario swapping just one time
remote_execute {
    report_analysis_coverage
    report_timing -delay_type max ...
    report_timing -delay_type min ...
    report_constraint -all_violators ...
}

# Minimum reports at manager to minimize scenario swapping
report_constraint -all_violators ...

# Write changes to only one scenario to avoid scenario swapping
current_scenario scen1
remote_execute { write_changes -output my_eco.tcl }

```

See Also

- [SolvNetPlus article 000007372, “How Can I Restore Saved Sessions in Distributed Multiscenario Analysis?”](#)

Manual Netlist Editing

You can manually edit the netlist and analyze how the changes affect the timing. This can be useful late in a design cycle when only small edits can be made. However, if you need to fix many violations or make substantial changes, you should use PrimeTime ECO fixing commands.

To manually edit the netlist, follow these steps:

1. If you need to replace or resize cells, find alternative library cells by running the `get_alternative_lib_cells -base_names` command:

```
pt_shell> get_alternative_lib_cells -base_names U135
AN2 AN2i
```

2. Edit the netlist by using one or more of these commands.

Table 65 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

3. Generate a change list file by using the `write_changes` command.

“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 fix violations, you can increase the sizes of the driving cells with the `size_cell` command or insert buffers with the `insert_buffer` command. To fix crosstalk violations, you can 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 the physical implementation tool.

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 complete GPD or SPEF parasitic data and netlist data from the physical implementation tool.

Automatic Uniquifying of Blocks

If you edit a hierarchical block that is one of multiple instances of the same block, 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 from 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 when you relink the design.

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 multiple 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
```

As soon as you edit a block, its parent and ancestor blocks up to the top level are edited. This information is available from a Boolean attribute, `is_edited`, which is available on a design and on hierarchical cells. In the foregoing example, after you use the `size_cell` command, the `is_edited` attribute is true on i1/low block:

```
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.

To get library cell objects, use the `get_lib_cells` command, for example,

```
pt_shell> get_lib_cells /*
```

The name of a library cell can be either the following:

- The full name of the library cell, such as `mylib1/AND2`, in which case there is no ambiguity about what to link the cell to
- 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](#).

You can resolve library cell base names from the cell's current library or from a specific library by 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 library cell base name using the currently linked library.
- 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.

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 or `set_link_lib_map` command
- Libraries specified by the `link_path` variable

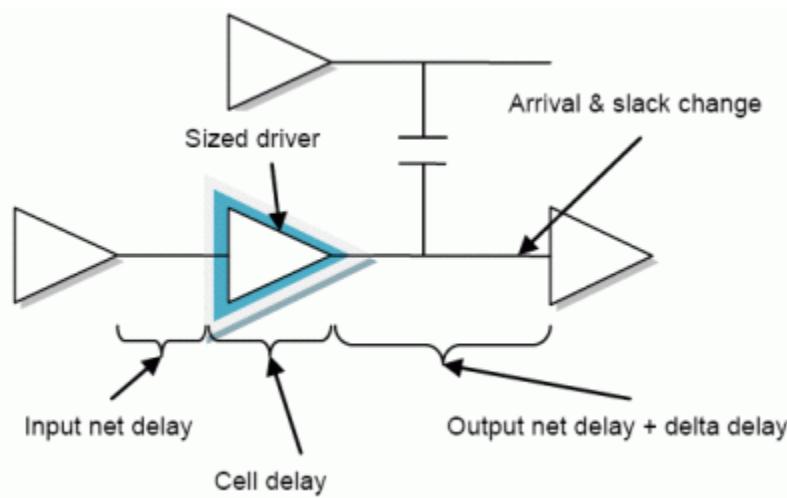
Use the `-base_names` option with the `get_alternative_lib_cells` command to return the alternative library cells as a list of library cell base names (strings) rather than a collection.

Estimating Delay Changes

When performing ECO design modifications, it is important to understand the timing effects. Reporting the effects of a change with the `report_timing` command incurs the cost of an incremental timing update.

For faster analysis of proposed `size_cell` and `insert_buffer` changes, 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 considers the incoming transition times, output loading, fanout loading, detailed parasitics if present, and driver characteristics of the candidate cells, as shown in the following diagram.

Figure 378 Parameters Considered by ECO Timing Estimation



The `estimate_eco` command is fast but the results are not guaranteed to be exact, so you should use this command only to explore different ECO options. For full accuracy, analyze the final results using the `report_timing` command.

When using the `estimate_eco` command, follow these steps:

1. Show the available options.

In this stage, you evaluate possible ECO options. For example,

```
pt_shell> estimate_eco -type size_cell -max -rise CPU2/U93

delay type : max_rise
lib cell           area stage_delay    arrival      slack
-----
mylib90/INVD24    21.17   0.0265     0.0265    0.1671
mylib90/INVD20    18.35   0.0291     0.0291    0.1645
mylib90/CKNXD8    14.11   0.0610     0.0610    0.1326
mylib90/INVD3     3.53    0.1295     0.1295    0.0212
*mylib90/INVD2    2.82    0.1936     0.1936    -0.0013
mylib90/CKNXD1    2.82    0.4271     0.4271    -0.3214
```

This report shows alternative library cells and the timing that would result from using each library cell. The current library cell is marked with an asterisk (*).

2. Estimate the stage delay improvements.

After you decide on the change, analyze the delay effects at the stage where the netlist change will occur. For example, to insert a buffer at a specified pin and estimate the stage delay change, use this command:

```
pt_shell> estimate_eco -type insert_buffer -inverter_pair \
          -max -rise -verbose -lib_cells {INVD2} CPU2/U25/Y

cell name : CPU/U25
current lib cell: mylib90/INVD1
buffer lib cell : mylib90/INVD2

max_rise           current      estimate
-----
area              0.925       1.270
input net delay   0.102       0.052
stage delay       0.000       0.000
  cell delay      0.371       0.175
  output net delay 0.206       0.110
  buffer delay     0.023       0.013
  delta delay      0.140       0.051
arrival time       0.371       0.175
slack              0.304       0.499
```

3. Commit changes and verify.

If you are sure about the fix, run the `size_cell` or `insert_buffer` command, and then run the `update_timing` command to see the actual timing changes with full accuracy.

Note:

The timing update does not take into account the changes in parasitics that could result from new placement and routing.

To customize the types of information reported by the `estimate_eco` command, set the `eco_estimation_output_columns` variable. In addition to the default information, you can also report the transition time, stage min/max transition, stage min/max capacitance, and stage max fanout.

Sizing Cells

To replace an existing cell with a new cell of a different size, use the `size_cell` command.

A library might contain several alternative library cells. For example:

```
pt_shell> get_alternative_lib_cells CPU2/reg318
{"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 CPU2/reg318
...
Report : alternative_lib_cells
        CPU2/reg318
        -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 CPU2/reg318 class/FD2P3
Information: Sized 'CPU/reg318' with 'class/FD2P3'
1
```

See Also

- [Library Cell Functional Equivalence](#)

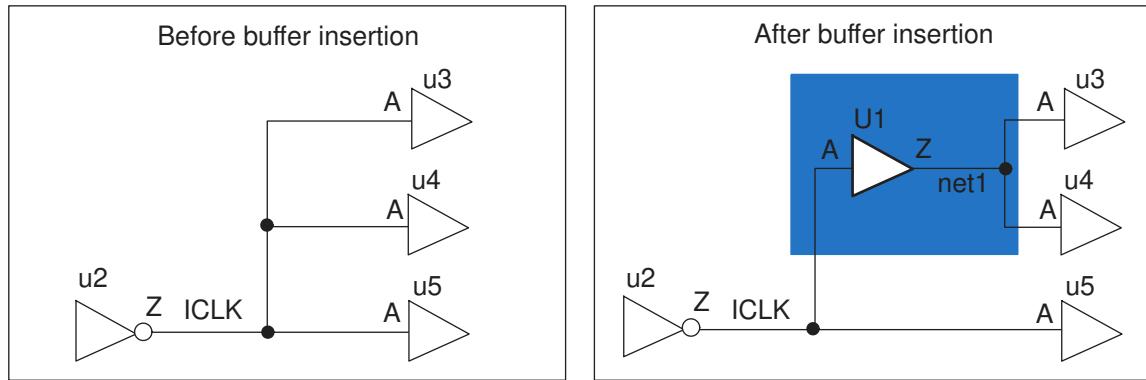
Inserting and Removing Buffers

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.

For example, to insert buffers at pins u3/A and u4/A using library cell class/B1I:

```
pt_shell> insert_buffer {u3/A u4/A} class/B1I
```

Figure 379 Buffer Insertion Example



The `insert_buffer` command creates a new buffer “U1” and a new net “net1.” The input of the new buffer is the existing net “ICLK” and the output is the new net “net1.”

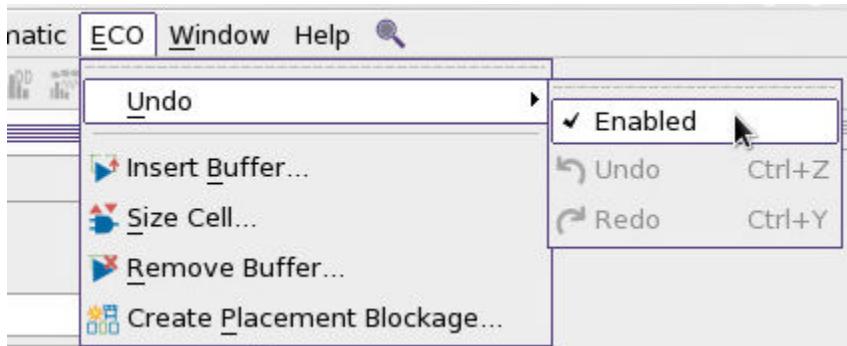
To change the prefix name of newly inserted buffers, set the `eco_instance_name_prefix` variable. By default, this variable is set to U.

Sizing Cells and Inserting Buffers in the GUI

The PrimeTime GUI provides menu commands for inserting buffers, removing buffers, and sizing cells. A PrimeTime-ADV-PLUS license is required to perform ECO actions in the GUI. After you make a change, you can easily undo it.

To perform an ECO operation in the GUI, use the menu options under ECO, shown in the following figure.

Figure 380 ECO Menu in the GUI



To enable the undo feature, make sure the Enabled option has a check mark. If you do not plan to use the feature, you can disable it to save memory.

For more information on using the GUI for manual ECOs, see [Layout View and ECOs in the GUI](#).

Manual Netlist Editing and dont_use Restrictions

Library cells can be excluded from use by the `fix_eco_*` commands by applying `dont_use` restrictions using the `set_dont_use` and `set_target_library_subset` commands.

These `dont_use` specifications do not restrict manual ECO commands or the `get_alternative_lib_cells` or `report_alternative_lib_cells` commands. However, the tool does issue a warning if you perform a manual ECO operation that implements a restricted library cell:

```
pt_shell> insert_buffer U9/Y slow/HVT_BUFX1
Warning: Library cell 'slow/HVT_BUFX1' is part of a -dont_use subset
specification that applies to cell 'U9'. (NED-076)
Information: Inserted 'U1' at 'U9/Y' with 'slow/HVT_BUFX1'. (NED-046)

pt_shell> size_cell U3 slow/HVT_OR2X2
Warning: Library cell 'slow/HVT_OR2X2' is part of a -dont_use subset
specification that applies to cell 'U3'. (NED-076)
Information: Sized 'U3' with 'slow/HVT_OR2X2'. (NED-045)
```

Swapping Cells

To replace the cells in the specified cell list with a different design or library cell, use the `swap_cell` command.

Note:

The `swap_cell` command is intended for swapping complex cells or hierarchical cells. When cells are swapped, a full timing update is incurred. For simple cell resizing, use the `size_cell` command instead.

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_verilog A.v
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]
```

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, ensure that the old and new names are at the same level of hierarchy. Reference any cell or net below the current instance by its full hierarchical name.

This example renames cellA to cellB in the current instance, middle.

```
pt_shell> rename_cell cellA cellB
Information: Renamed cell 'cellA' to 'cellB' in design 'blkA'. (NED-008)
1
```

This 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, so they are uniquified as part of the editing process.

```
pt_shell> rename_net [get_nets 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 'blkA/u1/low'. (NED-009)
1
```

The `write_changes` command writes out the name changes using the `rename_cell` and `rename_net` commands.

Library Cell Functional Equivalence

PrimeTime ECO operations can replace existing cells with functionally equivalent cells that have different area, drive strength, and power.

A functionally equivalent cell must have the same logical function, same number and direction of I/O pins, and same timing arcs.

If the `eco_strict_pin_name_equivalence` variable is set to `false` (the default), the tool can size cells even when the pin names do not match between the original and replacement cells. The tool uses other library cell information to find the matching pin functionality between the original and replacement cells.

If the `eco_strict_lib_arc_equivalence` variable is set to `false` (the default is `true`), the tool can size cells even when the timing arcs do not match between the original and replacement cells. The tool uses other library cell information to find the matching pin functionality between the original and replacement cells.

For some complex library cells, you might need to create a library cell attribute named `user_function_class` to describe the functional behavior of the cell. For details, see [SolvNetPlus article 000010601, "What are the ECO Cell Equivalency Rules for PrimeTime?"](#) The foundry can provide information about function classes to ensure that the proper cells are chosen for upsizing.

Netlist Editing in Multiple Scenarios

When editing the netlist to fix violations, you should 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 and get 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 manager. However, complex nested command structures are not supported in this mode of operation.

In DMSA, you can use the netlist editing commands in [Table 65](#). You cannot use the `swap_cell` command in the DMSA manager, but you can execute it through the `remote_execute` command.

By using the `remote_execute` command, you can execute the following commands in a worker 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.

22

Hierarchical Analysis

For a large chip design, performing a flat timing analysis from the top level of hierarchy can consume memory resources or use excessive runtime. To accommodate a large design, you can analyze it hierarchically so that each individual analysis run applies to only a portion of the whole chip. To learn about hierarchical analysis, see

- [Overview of Hierarchical Analysis](#)
 - [HyperScale Analysis](#)
 - [Context Characterization](#)
 - [Context Budgeting](#)
 - [Extracted Timing Model \(ETM\)](#)
 - [Quick Timing Model \(QTM\)](#)
-

Overview of Hierarchical Analysis

Analysis of a large design can consume memory resources or use excessive runtime. To accommodate a large design, you can analyze it hierarchically so that each individual analysis run applies to only a portion of the whole chip.

The PrimeTime tool offers the following features to support hierarchical analysis:

- [HyperScale Analysis](#)

HyperScale technology is an advanced method that analyzes the block-level and top-level portions using separate runs and accurately handles the timing interfaces between the top level and the lower-level blocks. This technology gives the accuracy of full flat analysis while using the reduced runtime and memory footprint of hierarchical block-level analysis.

- [Context Characterization](#)

Context characterization is a feature that supports block-level analysis by writing out the timing context of a block within the top level. Then you can analyze the timing of the block by itself, in isolation from the top level, by reading the context information into the

block analysis session. This produces accurate timing results for the block operating in the context of the top level.

- [Extracted Timing Model \(ETM\)](#)

An ETM is a block timing models extracted directly from the lower-level block netlist. The model consists of a set of timing arcs between the clock, input, and output pins of the block. This model can be used in place of the full block netlist for analysis at the next higher level of hierarchy.

ETMs are portable and easily exported to other tools, so they are often used to represent the timing of intellectual property (IP) blocks. Because the model consists only of timing arcs, it keeps the internal logic of the block confidential.

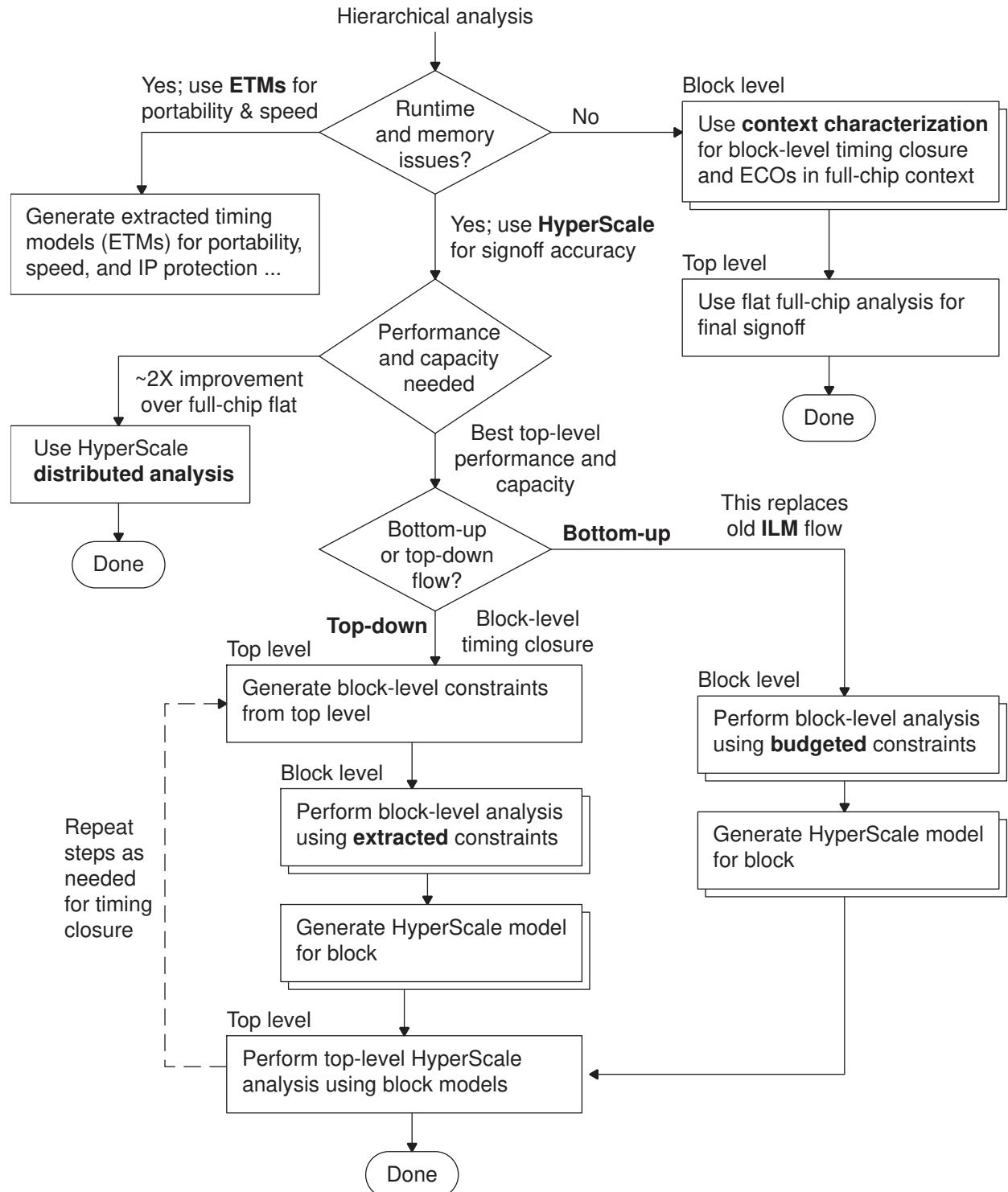
- [Quick Timing Model \(QTM\)](#)

You create a QTM by using a set of PrimeTime commands to specify the inputs, outputs, and timing arcs of the model. This method can be used in early stages of the design cycle, when no netlist is available for the block. The model typically contains rough estimates of the expected block timing.

When the block netlist becomes available, you can replace the QTM with an ETM or analyze the block using HyperScale technology.

The following flowchart can help you select the best PrimeTime hierarchical analysis flow to use for your design.

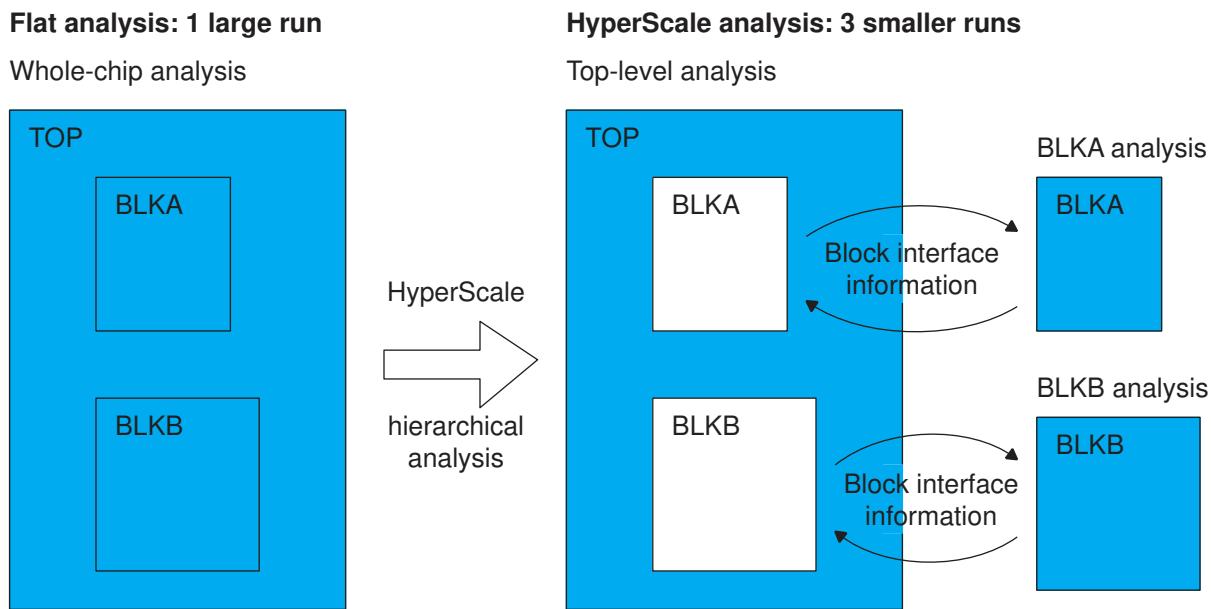
Figure 381 Hierarchical Analysis Flows



HyperScale Analysis

The PrimeTime HyperScale hierarchical analysis method analyzes the block-level and top-level portions of the design using separate runs and accurately handles the timing interfaces across hierarchical boundaries. The following figure compares a full flat analysis and a hierarchical analysis using HyperScale technology.

Figure 382 Flat Analysis Versus HyperScale Hierarchical Analysis



In the HyperScale flow, the tool creates a context-accurate model for each lower-level block. This model allows each block to be analyzed separately, taking into account the constraints and timing characteristics of the higher-level design. It also allows the higher-level design to be analyzed using an efficient representation for each lower-level block that excludes the internal logic of the block.

The tool handles the transfer of interface data between the top-level and block-level analysis runs, including input delays, output delays, and cross-boundary effects such as timing exceptions, crosstalk, and CRPR. The resulting analysis has the same high accuracy of a full flat analysis, but offers the reduced turnaround time and lower memory footprint of hierarchical analysis.

PrimeTime HyperScale technology offers the following benefits:

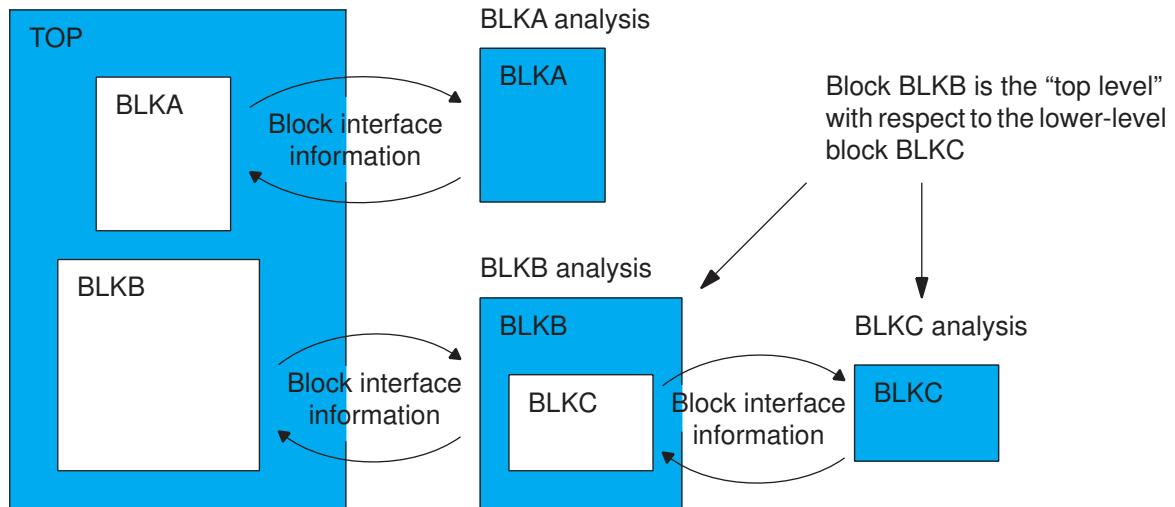
- **Runtime and capacity** – Compared to flat, full-chip analysis, HyperScale analysis provides significantly faster runtimes and uses less peak memory. This is achieved by intelligently reusing block-level and top-level analysis data and reducing the repeated computations.
- **Accuracy** – The HyperScale analysis flow performs the same delay computation and timing analysis as flat analysis, with the accuracy of the flat PrimeTime flow for both uncoupled and coupled signal integrity analysis. HyperScale maintains full accuracy, including accounting for signal integrity effects across block boundaries.
- **Efficient top-level analysis** – The HyperScale flow automatically records and reduces block timing data to enable more efficient top-level analysis. This relieves designers from the tasks of creating, validating, and managing timing models. In addition, the tool captures the timing context of the block at the higher level, allowing context-accurate block-level analysis as a separate process.
- **Block scope checking** – When the tool loads previously saved block data to run top-level timing analysis, it automatically verifies that each block is correctly instantiated within its scope for valid timing analysis.
- **Automatic context updates** – The HyperScale flow automatically updates and consumes the block timing context so that no rebudgeting is required to drive timing closure ECOs.
- **ECO guidance for place-and-route** – IC Compiler or IC Compiler II can use information from PrimeTime HyperScale technology to guide the timing ECO process, which assists in achieving block-level and top-level timing closure. By providing focused fixing information, the physical implementation tool needs fewer iterations to close timing.
- **Constraint management** – With the HyperScale flow, you close timing at the block level using the latest context information from the top-level analysis along with the block-level constraints. During top-level analysis, the latest constraints from the top-level timing overwrite the block-level constraints. Top-level constraints can be used in either the flat or HyperScale flow. The tool also performs constraint extraction to get block-level constraints consistent with the flat top-level constraints.

A large chip design can span multiple hierarchical levels. Using HyperScale technology, you can analyze such a design hierarchically across multiple levels as shown in the following figure.

Figure 383 Hierarchical Analysis Across Multiple Levels

HyperScale analysis: 4 smaller runs

Top-level analysis



The term “top level” can mean any analysis level containing one or more lower-level blocks analyzed separately from that level. In this example, block BLKB can be considered the “top level” with respect to the lower-level block BLKC, even though BLKB is itself a block within the higher level called TOP.

To learn how to use HyperScale technology, see the following topics:

- [Key HyperScale Technology Features](#)
- [HyperScale Usage Flows](#)
- [Choosing a Top-Down or Bottom-Up Flow](#)
- [HyperScale Usage Details](#)
- [HyperScale ECO Closure Methods](#)
- [Timing Analysis in HyperScale](#)
- [Block Models](#)
- [Block Boundary Modeling](#)
- [Model-Context Mismatch Handling](#)

- Controlling the Block Boundary Adjustment
- HyperScale Attributes

Key HyperScale Technology Features

These are the key features of HyperScale technology:

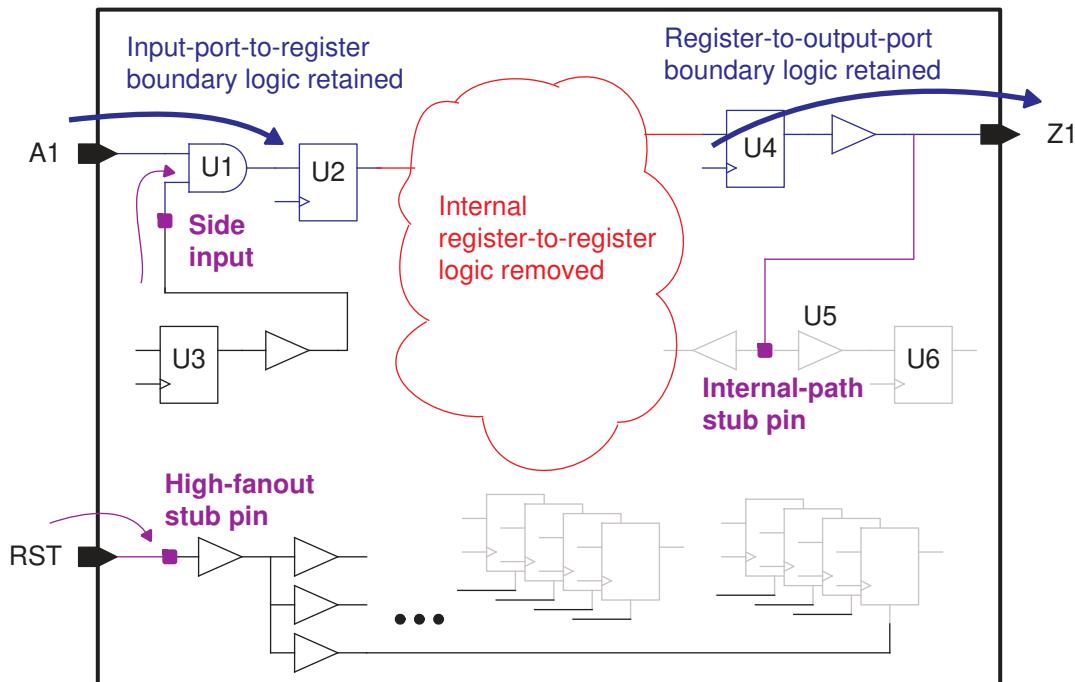
- HyperScale Modeling
- HyperScale Block Context
- Multiply Instantiated Model (MIM) Analysis

HyperScale Modeling

In hierarchical analysis, the tool analyzes the timing of lower-level blocks separately from the top level. In a bottom-up hierarchical flow, the top-level analysis uses a HyperScale model to represent each lower-level HyperScale block. Each model includes the interface logic of the block boundary but excludes the internal register-to-register logic of the block.

The following diagram shows the logic of a HyperScale block model.

Figure 384 HyperScale Block Model



A HyperScale block model includes the following types of logic:

- Input-to-register boundary logic, plus related side-input pins for accurate modeling of possible worst paths
- Register-to-output boundary logic, plus related internal-path stub pins to model the effects of network load
- High-fanout pins to represent the timing of removed high-fanout logic

The HyperScale block model includes all the logic necessary for accurate timing analysis at the top level while excluding the internal register-to-register logic unrelated to top-level timing analysis. As a result, the model is as fast and compact as possible for analysis with flat-like accuracy.

To create a HyperScale model for a block, use commands similar to the following:

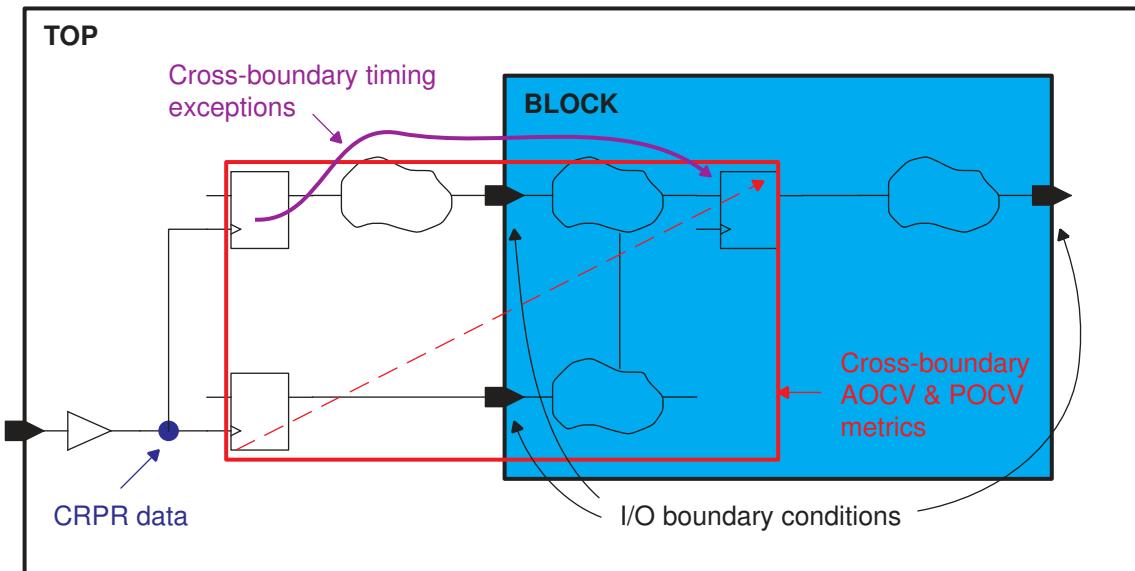
```
set_app_var hier_enable_analysis true    # Enable hierarchical analysis
read_verilog block_A.v                  # Read in the block
link_design BLK_A
...
update_timing                           # Run timing analysis
update_noise                            # Run noise analysis if applicable
...
write_hier_data $blkModel              # Write out hierarchical data
```

HyperScale Block Context

The HyperScale block context is the information about the timing environment surrounding a block in the context of the top level where the block resides. This information lets you analyze the block separately, in isolation from the top-level logic, while still considering the constraints on the block imposed at the top level.

The HyperScale context includes not only the input delays and output delays along the I/O boundary, but also conditions such as cross-boundary timing conditions, cross-boundary on-chip variation data (AOCV and POCV metrics), and clock reconvergence pessimism removal (CRPR) data, as shown in the following diagram.

Figure 385 HyperScale Block Context



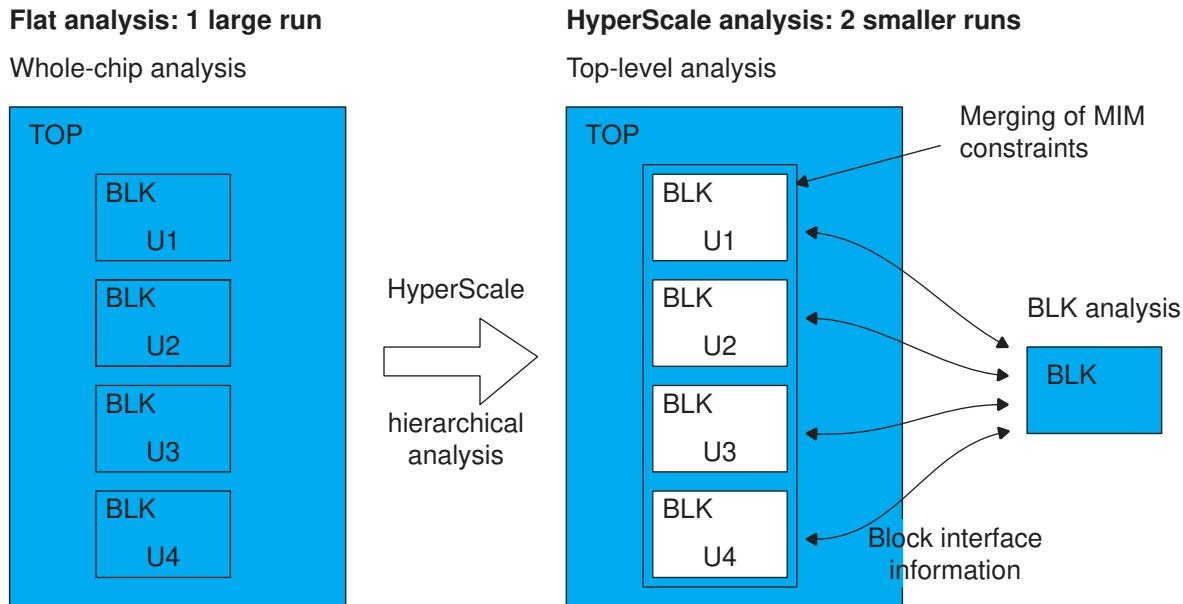
To generate the context for a block, use commands similar to the following:

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
set_hier_config -block A -path $model_A    # Specify block model
...
# read_verilog blk_A.v                      # Block Verilog not needed
read_verilog top.v                         # Read top Verilog
link_design TOP
...
update_timing                                # Run timing analysis
update_noise                                  # Run noise analysis if applicable
...
write_hier_data $topContextDir               # Write out block context
```

Multiply Instantiated Model (MIM) Analysis

If a design contains multiply instantiated modules (MIMs), you can optionally run just one analysis of the module and use the same results for multiple instances of that block at the top level, as shown in the following figure.

Figure 386 Context Merging for Multiply Instantiated Modules



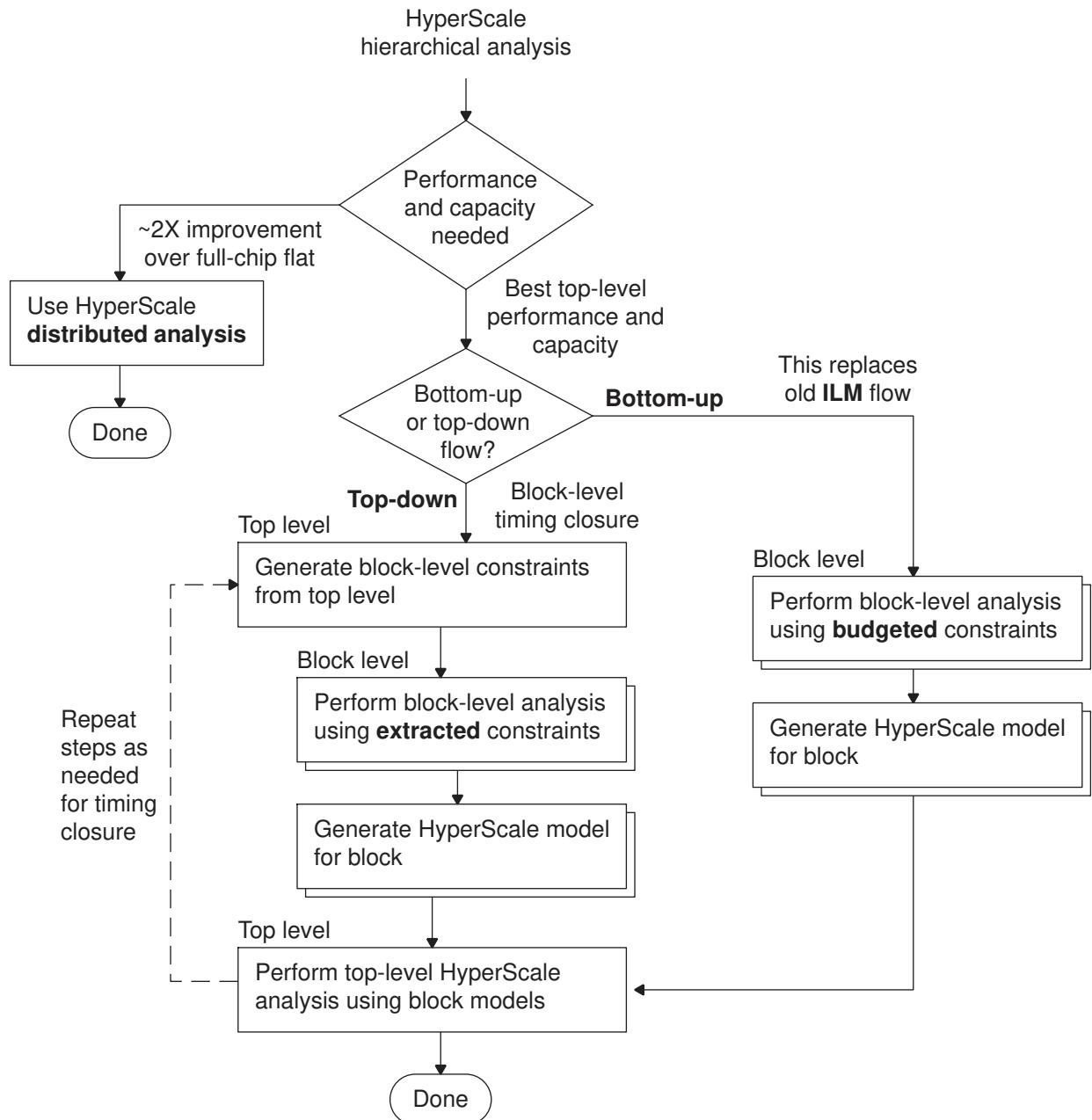
Where different constraints apply to different instances of the module, the tool “merges” the constraints, choosing the most restrictive constraints among all instances, so that the single block-level analysis results can be used for all instances of the module. An instance in an unusual and distinct environment can be handled separately, if needed.

For ECO fixing, the merged context can be applied to all MIM instances so fixing is performed only once for all the instances.

HyperScale Usage Flows

There are several different ways to use HyperScale technology to improve runtime and capacity for analysis of a large hierarchical design. The following flowchart can help you select the best HyperScale flow for your design.

Figure 387 Hierarchical Analysis Flows



These are some of the more common HyperScale usage flows:

- **Bottom-Up Analysis Using Block Models Without Context** – You analyze the block timing using budgeted timing constraints. Use this method if you do not know the exact timing environment in which the block will be used. You can use this method as a replacement for ETM flows. This method provides the best top-level performance and capacity compared to flat analysis.
- **Top-Down Analysis Using Block Models and Context** – You separately analyze the top level and each lower-level block. Each block-level analysis reads in the context from the last top-level analysis, and each top-level analysis uses HyperScale models generated by block-level analysis. This method provides the best top-level performance and capacity compared to flat analysis, and also achieves block-level timing closure.

Bottom-Up Analysis Using Block Models Without Context

In bottom-up modeling without context, you analyze the block timing using budgeted timing constraints, without considering the context of the block in the top level. Use this method if you do not know the exact timing environment in which the block will be used.

This method is more accurate than using extracted timing models (ETMs) because the model includes more block interface information. However, the model can be used only with the PrimeTime tool, not with other tools.

Block Analysis

To create each block model, use commands similar to the following:

```
set_app_var hier_enable_analysis true    # Enable hierarchical analysis
read_verilog block_A.v
link_design BLK_A
...
update_timing                                # Update timing
update_noise                                 # Update noise if applicable
...
write_hier_data $blkModel                   # Write out hierarchical data
```

Top-Level Analysis

For the top-level analysis in a later session, use commands similar to the following:

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
set_app_var timing_save_hier_context_data false # No hier context needed
set_hier_config -block A -path $blkModel # Specify block model

### read_verilog blk_A.v                  # Block Verilog not needed
read_verilog top.v
link_design TOP
...
update_timing
```

```
update_noise
...
```

Moving From an ETM Hierarchical Flow

If you have an existing hierarchical analysis flow using extracted timing models (ETMs), you can easily convert your scripts to perform HyperScale analysis instead for higher accuracy. Only minor modifications of your scripts are needed to generate HyperScale block models, which replace ETM blocks.

To create each block model, use commands similar to the following:

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
# Run existing block-level setup script
    read_verilog block_A.v      # Read block Verilog
    link_design BLK_A
    read_parasitics BLOCK_A.specf
    source BLOCK_A.sdc
    read_context $topDir          # Read HyperScale context if available
    update_timing -full          # Also run update_noise if applicable
...
write_hier_data $blkModel        # Write out hierarchical block model
```

For the top-level analysis in a later session, use commands similar to the following:

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
set_app_var timing_save_hier_context_data false  # No hier context needed
set_hier_config -block A -path $blkModel    # Specify block model

# Run existing top-level setup script
    read_verilog blk_A.v          # Read block Verilog (optional)
    read_verilog top.v           # Read top-level Verilog
    link_design TOP
    read_parasitics TOP.specf
    ...
    update_timing -full          # Also run update_noise if applicable
    ...
write_hier_data $topContext_dir      # Write out block context if needed
save_session $topSession            # Save session if needed
```

In moving from the ETM flow to the HyperScale flow, the HyperScale block model replaces the .lib or .db extracted timing model. You can change the `link_path` variable to remove the path to the ETM .lib or .db file.

After top-level analysis, if further block-level analysis is needed, write out the block context by using the `write_hier_data` command. Then you can analyze each lower-level block in isolation from the top level, while including the full context of the block in the top level.

Top-Down Analysis Using Block Models and Context

In the top-down HyperScale analysis flow, you separately analyze the top level and each block. The top-level analysis writes out the context for each lower-level block. Each block-level analysis reads in the context from the last top-level analysis and writes out a HyperScale block model with timing interface data. A subsequent top-level analysis uses the generated HyperScale block models.

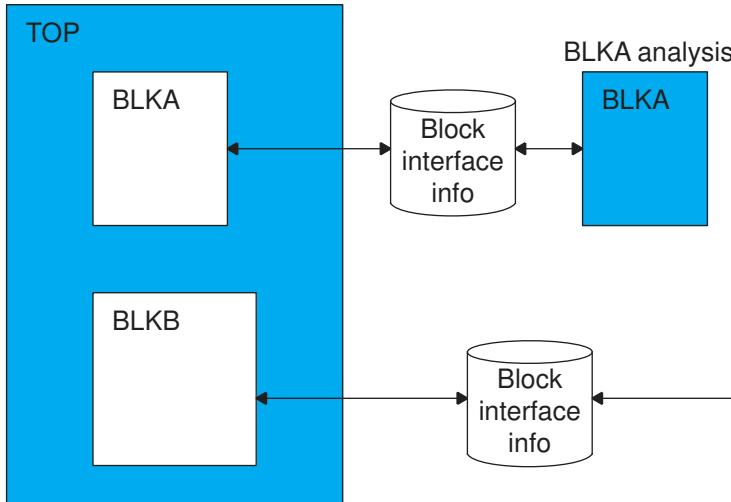
You can run top-level and block-level analyses multiple times to achieve timing closure. The top-level and block-level runs can be controlled by different teams at different locations.

For example, the following diagram shows an analysis that specifies two blocks at the top level for separate analysis.

Figure 388 HyperScale Analysis Using Top-Level and Block-Level Analysis Runs

HyperScale analysis: 3 runs (TOP, BLKA, BLKB)

TOP analysis



```

set hier_enable_analysis true
...
read_verilog BLKA.v
link_design BLKA
...
read_context $stopContextDir
...
update_timing
...
write_hier_data $model_A

```

BLKB analysis

```

set hier_enable_analysis true
...
read_verilog BLKB.v
link_design BLKB
...
read_context $stopContextDir
...
update_timing
...
write_hier_data $model_B

```

Top-Down Analysis With Multiply Instantiated Modules (MIMs)

The following example is a design containing a single instance of a block at the top level, or multiple instances of the same block to be represented as a single model using merged context for the multiple instances.

For the top-level analysis session, use commands similar to the following:

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
set_hier_config -block A -path $blkModel # Specify block model
set_hier_config -enable_mim               # Optional for fast MIM analysis

# read_verilog blk_A.v                   # Block Verilog not needed
read_verilog top.v
link_design TOP
...
update_timing
...
write_hier_data $topContext_dir          # Write out block context
```

For each block-level analysis, use commands similar to the following:

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
read_verilog block_A.v
link_design BLK_A
...
read_context $topContext_dir             # Read context for block
update_timing
...
write_hier_data $blkModel                # Write out hierarchical data
```

Instead of enabling MIM analysis for the whole design, you can put multiple instances of a block into groups for analysis, using a single model of the block for each group.

```
set_app_var hier_enable_analysis true      # Enable hierarchical analysis
set_hier_config -block A -path $blkModel # Specify block model
set_hier_config -block A -path $blkModel_g1 \
    -instances {a1 a2} -name g1           # Instances a1 a2 in group g1
set_hier_config -block A -path $blkModel_g2 \
    -instances {a3 a4} -name g2           # Instances a3 a4 in group g2
...
```

Automatic Generation of HyperScale Scripts and Block Context

If you have an existing script that performs full-chip flat analysis, you can have the tool automatically create template scripts and block constraints to perform top-down HyperScale analysis.

Use this feature as follows:

1. Before linking the design, set the following variables to enable HyperScale template script creation:

```
set_app_var hier_enable_analysis true
set_app_var hier_create_template_scripts true
```

This feature affects the behavior of the `link_design`, `update_timing`, `save_session`, and `write_hier_data` commands. Correspondingly, it must be enabled before linking the design.

2. Also before linking the design, specify the blocks to create block-level scripts for:

```
set_hier_config -enable_mim \
-partitions \
{SMALLBLK1 SMALLBLK0 \
 MEDIUMBLK1 MEDIUMBLK0 \
 LARGE_BLK}
```

The `-enable_mim` option is required. All block-level instances for script creation must be included in the list, whether for the same design or different designs.

3. After the timing update, create the top-level and all block-level scripts in an output directory:

```
write_hier_data HS_SCRIPTS
```

In template-creation mode, this command does the following:

- Creates a partition0/ directory with a RUN.ptsh analysis script for the HyperScale top-level run
- Creates partition1/ through partitionN/ directories, each with a RUN.ptsh analysis script, for the block-level runs of the specified blocks
- Creates initial constraint files for the block-level runs using context characterization

The script directories are named partition0/ through partitionN/. To see the design that corresponds to each directory, grep the `link_design` commands from the run scripts:

```
% grep link_design HS_SCRIPTS/partition*/RUN.ptsh
HS_SCRIPTS/partition0/RUN.ptsh:link_design TOP
HS_SCRIPTS/partition1/RUN.ptsh:link_design SMALLBLK
HS_SCRIPTS/partition2/RUN.ptsh:link_design MEDIUMBLK
HS_SCRIPTS/partition3/RUN.ptsh:link_design LARGEBLK
```

A full example script for this feature is as follows:

```
# enable HyperScale
set_app_var hier_enable_analysis true
```

```
# enable HyperScale script creation
set_app_var hier_create_template_scripts true

# specify blocks for script creation
set_hier_config -enable_mim \
    -partitions \
    {SMALLBLK3 SMALLBLK2 SMALLBLK1 SMALLBLK0 \
    MEDIUMBLK1 MEDIUMBLK0 \
    LARGEBLK}

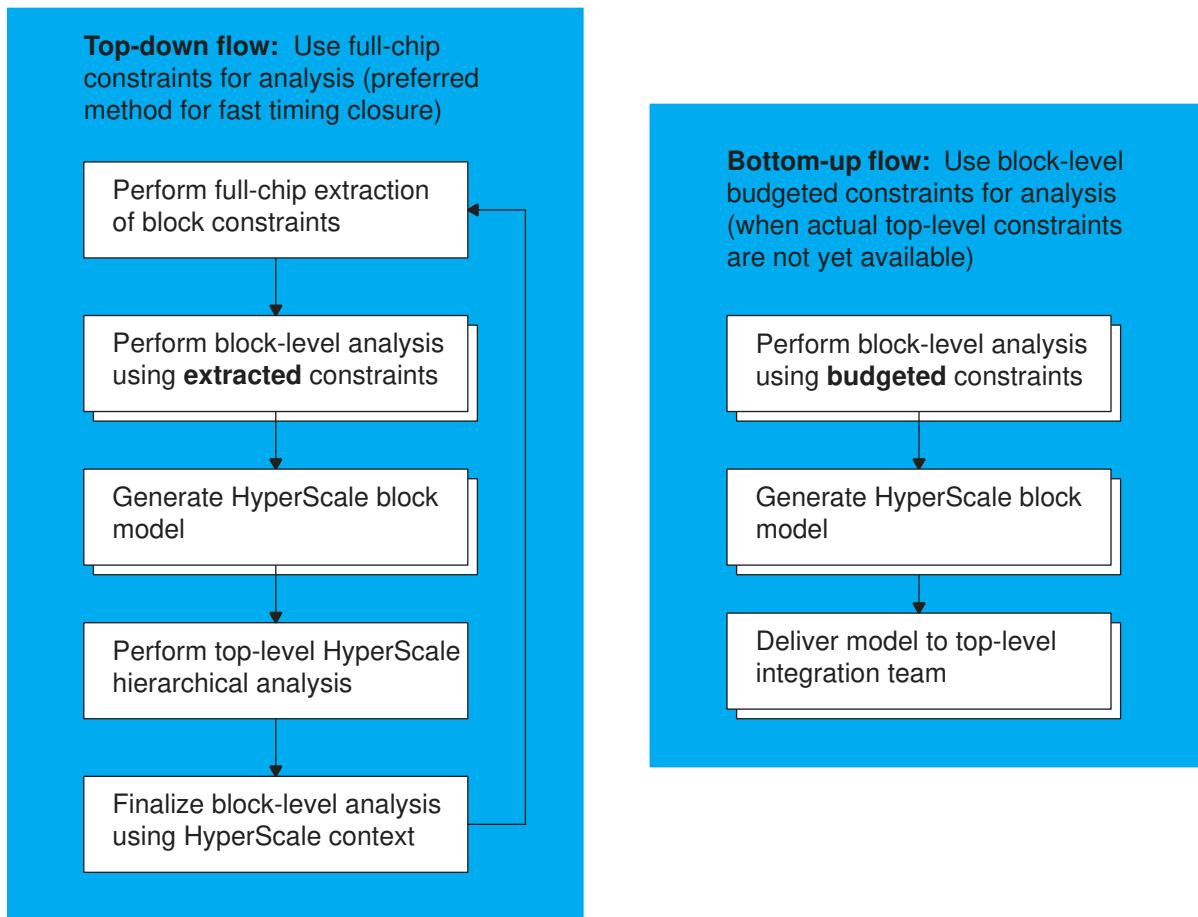
# Read full-chip design data
ad_verilog TOP_full.v
link_design TOP
read_parasitics TOP_full.spef
read_sdc TOP_full.sdc
...
update_timing

# Generate HyperScale scripts and constraints
write_hier_data HS_SCRIPTS
```

Choosing a Top-Down or Bottom-Up Flow

You can choose either a top-down or bottom-up hierarchical analysis flow. The following figure shows the steps in these two flows.

Figure 389 HyperScale Top-Down and Bottom-Up Hierarchical Analysis Flows



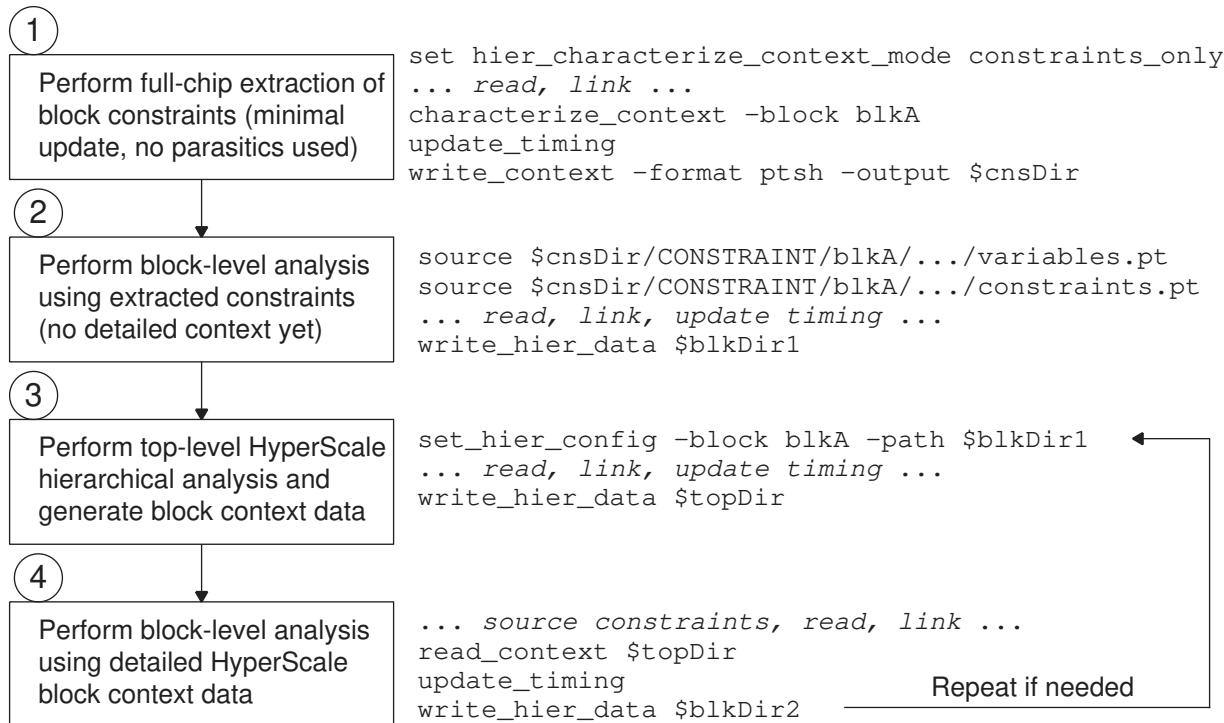
The top-down flow is the preferred method because timing closure is easier and faster to achieve, and the block constraints are guaranteed to be consistent with the top-level constraints from the start.

However, if a block is ready for analysis before the top-level constraints become available, you can use the bottom-up flow, starting with budgeted instead of extracted constraints for block analysis. In this case, when top-level analysis is performed later, it is important to check for consistency between the top-level and block-level constraints. The tool performs several types of consistency checking and reports any problems.

Top-Down HyperScale Flow

The following flow diagram summarizes the HyperScale top-down analysis flow.

Figure 390 HyperScale Top-Down Hierarchical Analysis Flow



The top-down flow consists of four basic steps, each run in a separate PrimeTime session:

1. Perform full-chip extraction of block constraints, using a minimal update and no parasitics, as a separate constraints-only PrimeTime session:

```

set_app_var hier_characterize_context_mode constraints_only
source full_chip_analysis_script.tcl          # Read, link, update
characterize_context -block blkA              # Generate context
update_timing
write_context -format ptsh -output $cnsDir   # Write constraints
  
```

Setting the `hier_characterize_context_mode` variable to `constraints_only` invokes the `constraints-only` extraction mode. In this mode, the tool ignores all `read_parasitics` commands and limits the operation of the `update_timing` command to only what is necessary to generate the block-level constraints; no actual timing analysis is performed. Perform this step in its own separate PrimeTime session.

2. Perform block-level analysis using the extracted constraints to get initial block-level timing results:

```

set_app_var hier_enable_analysis true
source $cnsDir/CONSTRAINT/blkA/hier_default/variables.pt
read_verilog BlkA.v
  
```

```

link_design
read_parasitics BlkA.spf
source $cnsDir/CONSTRAINT/blkA/hier_default/constraints.pt
update_timing -full          # Also run update_noise if applicable
...
write_hier_data $blkDir1      # Write initial block-level timing data

```

3. To generate accurate block context data, perform top-level HyperScale hierarchical analysis using the block-level timing results:

```

set_app_var hier_enable_analysis true
set_hier_config -block blkA -path $blkDir1    # Set block configuration
source full_chip_analysis_script.tcl    # Read, link, parasitics, update
write_hier_data $topDir                  # Write block context

```

4. Perform block-level analysis using accurate block context data:

```

set_app_var hier_enable_analysis true
source $cnsDir/CONSTRAINT/blkA/hier_default/variables.pt
read_verilog blkA.v
link_design
read_parasitics blkA.spf
source $cnsDir/CONSTRAINT/blkA/hier_default/constraints.pt
read_context $topDir          # Read block context
update_timing -full          # Also run update_noise if applicable
write_hier_data $blkDir2      # Write block-level timing data

```

Steps 3 and 4 can be repeated if needed to converge on timing closure.

Top-Level Timing Reports in the Top-Down Flow

You can report the timing as if a whole-chip analysis was done flat from the top level. For example, you can report a path that starts at the top level and ends inside a lower-level block:

```
pt_shell> report_timing -path_type full_clock_expanded \
    -from IN1 -to BLKA/reg2 ...
```

Block-Level Timing Reports in the Top-Down Flow

The `report_timing` command run at the block level shows timing paths that cross a hierarchical boundary with the top level. Constraints imposed from the top level are marked with the “at” character (@), as shown in the following example.

```
pt_shell> report_timing -from data[1] -to ff11 ...
...
Startpoint: data[1] (input port clocked by clock)
Endpoint: ff11 (rising edge-triggered flip-flop clocked by clock)
Path Group: clock
Path Type: max
```

Point	Incr	Path

clock clock (rise edge)	0.00	0.00
clock network delay (propagated)	0.00	0.00
input external delay	0.34 @	0.34 r
data[1] (in)	0.00 @	0.34 r
and1/A (AND2)	0.00 @	0.34 r
and1/Z (AND2)	0.06 &	0.41 r
ff11/D (FD3)	0.00 &	0.41 r
data arrival time		0.41
clock clock (rise edge)	5.00	5.00
clock source latency	0.25 @	5.25
clock (in)	0.00 @	5.25 r
cbuf1/A (BUF)	0.00 @	5.25 r
cbuf1/Z (BUF)	0.13 &	5.38 r
ff11/CP (FD3)	0.02 &	5.40 r
clock reconvergence pessimism	0.00	5.40
clock uncertainty	-0.05	5.35
library setup time	-0.15	5.20
data required time		5.20

data required time	5.20	
data arrival time	-0.41	

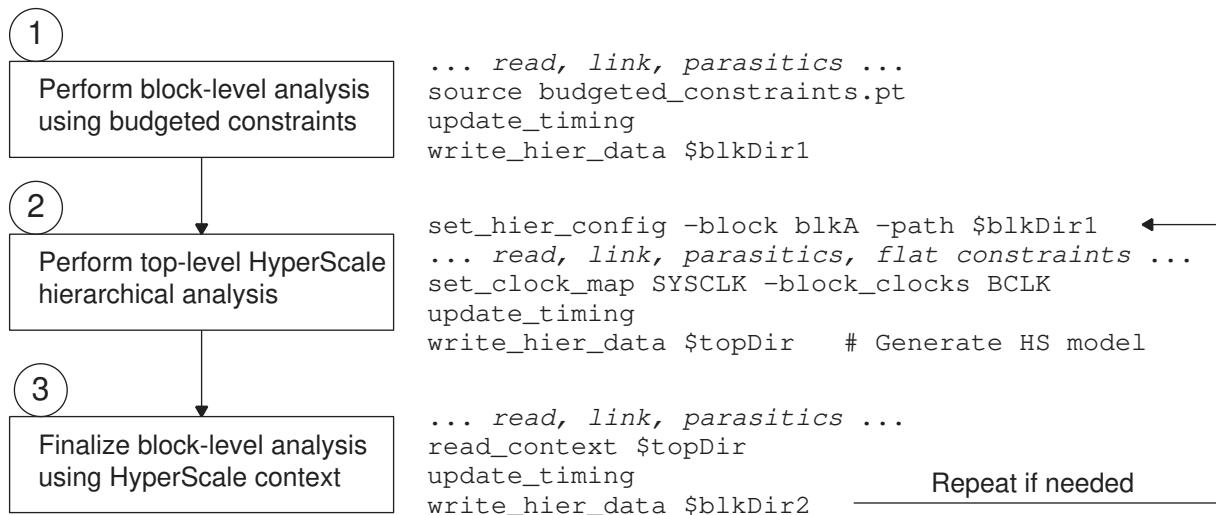
slack (MET)		4.79

The third line of the timing point list shows an “input external delay” of 0.34 followed by the “@” character. This means that the delay from the clock edge to arrival at the block-level input was calculated externally, in the top-level analysis session.

Bottom-Up HyperScale Flow

Use the bottom-up flow to analyze block-level timing when the top-level timing information is not yet available. The following flow diagram summarizes the HyperScale bottom-up flow.

Figure 391 HyperScale Bottom-Up Hierarchical Analysis Flow



The bottom-up flow consists of the following steps, each run in a separate PrimeTime session:

1. Perform block-level analysis using budgeted constraints to get initial block-level timing results and generate a HyperScale block model:

```
set_app_var hier_enable_analysis true
read_verilog BlkA.v
link_design
read_parasitics BlkA.spf
source budgeted_constraints.pt # Source budgeted constraints
update_timing -full           # Also run update_noise if applicable
...
write hier data $blkDir1       # Write initial HyperScale block model
```

- When the top-level design and constraints are available, to generate accurate block context data, perform top-level HyperScale hierarchical analysis using the HyperScale block model:

```

set_app_var hier_enable_analysis true
set_hier_config -block blkA -path $blkDir1      # Set block configuration
source full_chip_setup_script.tcl              # Read, link, load parasitics
set_clock_map SYSCLK -block_clocks BCLK        # Use clock mapping if needed
update_timing -full                         # Also run update_noise if applicable
...
report_clock -cells                      # Check clock mapping
report_clock -map                        # Check clock mapping
write_hier data $topDir                  # Write block context

```

The clock-related commands check for consistency between the top-level and block-level clock definitions.

3. Perform block-level analysis using accurate block context data:

```
set_app_var hier_enable_analysis true
read_verilog blkA.v
link_design
read_parasitics blkA.spf
source budgeted_constraints.pt      # Source budgeted constraints
read_context $topDir                # Read block context
update_timing -full                 # Also run update_noise if applicable
...
report_clock -map                  # Check clock mapping
write_hier_data $blkDir2           # Write block-level timing data
```

Steps 2 and 3 can be repeated as needed to converge on timing closure.

Checking for Constraint Consistency Between Top and Block

In the bottom-up HyperScale flow, you can use the constraint consistency checker to compare the budgeted block-level constraints against the top-level flat constraints.

Start by invoking the PrimeTime tool in constraint consistency checking mode:

```
% pt_shell -constraints
```

To check the constraint consistency, use a script similar to the following:

```
read_verilog TOP.v          # Read top design
read_verilog blkA.v         # Read block design
link_design top -verbose    # Link top design
link_design blkA -add -verbose # Link block design, keep top

current_design TOP
source flat_constraints.tcl # Source top-level flat constraints
current_design blkA
source budgeted_constraints.tcl # Source block-level budgeted constraints
current_design TOP

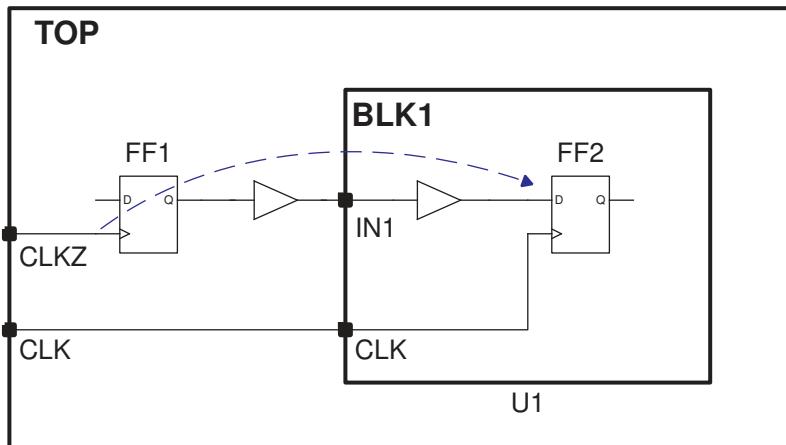
disable_rule *               # Disable all rules except
enable_rule hierarchical     # Hierarchical rule set
compare_block_to_top -block_design blkA # Perform comparison
report_constraint_analysis -include violations # Report violations
```

The final `report_constraint_analysis` command reports any inconsistency between the top-level and budgeted block-level constraints.

Clock Mapping Between Top and Block

In a bottom-up analysis flow, issues can arise when there is a mismatch between the block-level and top-level clocks. For example, in the following diagram, the block has one clock named CLK and the top-level design has two clocks, CLK and CLKZ.

Figure 392 Clock Matching Issue in a Bottom-Up Flow



For initial block-level analysis, there is no way for the tool to know that the signal arriving at input IN1 is clocked by a different clock, so the block model does not take this possibility into account. When the block model is used in top-level analysis, this situation can result in incorrect clock mapping, an unconstrained path, and an incorrect timing report.

To debug this type of issue, you can use the `report_clock` command at the top level:

```
pt_shell> report_clock -map
...
-----  

Instance      Top level clock      Block level clock      References  

-----  

U1           CLKZ                  (N/A)                (<virtual>)  

            CLK                   CLK                  U1/CLK
```

The report shows that the top-level clock CLKZ has no corresponding clock in the block.

One way to resolve this issue is to create a virtual clock in the block-level session:

```
pt_shell> create_clock -period 10.0 -name CLKZ
Warning: Creating virtual clock named 'CLKZ' with no sources. (UIITE-121)
```

Then the new model generated in the block-level session considers the possibility of arrivals clocked by the virtual clock.

When using the updated model, the top-level session reports the correct clock mapping:

Instance	Top level clock	Block level clock	References
U1	CLKZ	CLKZ	(<virtual>)
	CLK	CLK	U1/CLK

Subsequent timing reports from the top level correctly report the cross-boundary timing path.

If the top-level and block-level designs use the same clocks, but they happen to use different naming conventions, it is not necessary to rename the clocks. Instead, you can create a *clock map* that defines the associations between the top-level and block-level names. Use the `set_clock_map` command to create the map and the `report_clock -map` command to report the mapping. For more information, see [Clock Mapping Check](#).

HyperScale Usage Details

Using HyperScale technology consists of the following steps:

- [HyperScale Configuration](#)
- [Reading and Linking the Design Data](#)
- [Block-Level Analysis](#)
- [Top-Level Analysis](#)
- [Multiply Instantiated Modules \(MIMs\)](#)
- [Hierarchical Constraint Extraction](#)
- [Performing Manual Clock Mapping](#)
- [Mismatches Between Top and Block Levels](#)
- [Block Context Margin](#)
- [Block Boundary Checking](#)
- [Clock Mapping Check](#)
- [HyperScale Session Data](#)

HyperScale Configuration

Specify the HyperScale configuration for the current design by using the `set_hier_config` command. Use the `-block` option to specify each block to be analyzed

separately from the top level. For example, the following HyperScale configuration command configures an analysis of the TOP top-level design, which instantiates the blkA HyperScale block:

```
set_hier_config -block blkA -path $STA_DIR/blkA/HS_BLK
```

When you use the `set_hier_config` command with the `-block` option, you must also specify the path to that design's HyperScale session data directory with the `-path` option. The tool uses that directory to store information about that design. Although you can specify any directory name, consider using a common naming convention such as the `HS_` prefix used in the examples.

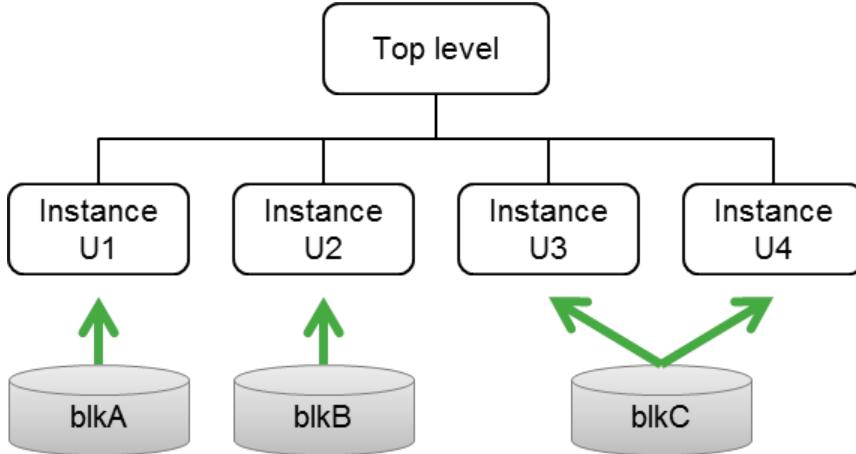
In the preceding example, the top-level analysis is performed inside the directory location defined by the `$STA_DIR` variable, and that the block-level analysis is performed inside the `blkA` subdirectory. You can specify a relative or absolute path. If the analysis is performed using multiple machines, the session data directories should be located on shared network-mounted file systems that are visible to all machines.

For each `set_hier_config` command, you can also specify the `-name` option to associate the HyperScale data with a particular label name. This option allows you to store analysis data from multiple analyses into the same session data directory. For example, the analyses for each mode and corner combination could store their own information in the same session data directory using a unique mode and corner name. If the `-name` option is not specified, a default label name is used.

Example of Top-Level HyperScale Configuration

[Figure 393](#) shows a design example with two levels of hierarchy. The top-level block is TOP, and the subblocks are blkA, blkB, and blkC.

Figure 393 Design example with two levels of hierarchy



To configure HyperScale for the design in [Figure 393](#), use these commands:

```
# Block design setup, where blkC is a multiply instantiated module (MIM)
set_hier_config -block blkA -path blkA/HS_DATA -name fast_func
set_hier_config -block blkB -path blkB/HS_DATA -name fast_func
set_hier_config -block blkC -path blkC/HS_DATA -name fast_func
```

Reporting the HyperScale Configuration

After you configure the design with the `set_hier_config` command, confirm the design configuration with the `report_hier_analysis` command. You can use this command in block-level or top-level analysis any time after `link_design`.

By default, the `report_hier_analysis` command reports the basic top-level and block-level configuration data applied by the `set_hier_config` command. For example,

```
pt_shell> report_hier_analysis
*****
Report : hyperscale
Design : TOP
...
*****
Cell           Name      Timestamp          HyperScale path
-----
blkA           default   2012-11-02 10:42:31 /disk1/blkA/HS_DATA/
<design>       default   2012-11-02 10:44:38 /disk1/TOP/HS_DATA/
```

This report shows the directories where the hierarchical block models and top-level session and context data are stored, the date and time that the HyperScale data was last updated, and the associated name label.

If the HyperScale path or design data is incorrect, then exit the current PrimeTime session, update the path values with the `set_hier_config` command, and start again.

The `report_hier_analysis` command can also report additional types of information, such as parasitics file information and block-level port abstraction data. For details, see the man page.

Reading and Linking the Design Data

After specifying the HyperScale configuration commands, you read and link the design. When HyperScale subblocks are instantiated in the design being analyzed, you do not read the netlists for these blocks; the HyperScale analysis automatically satisfies the link requirements for these blocks using the HyperScale block session data configured with the `set_hier_config` command.

After the design has been read and linked, apply any detailed parasitics with the `read_parasitics` command. Hierarchical extraction is required for a HyperScale flow, with separate detailed parasitics files generated for the top-level and for each of the

physically-unique HyperScale blocks. If an analysis contains HyperScale subblocks, you do not read the detailed parasitics for those blocks; the detailed parasitics are already a part of the HyperScale block session data, and they are automatically loaded after the design is linked. Instead, read in the top-level parasitics file that contains the detailed parasitic information up to the HyperScale block boundaries. PrimeTime stitches the top-level parasitics and HyperScale block parasitics together.

Behavior of link_design With HyperScale Configuration

With the HyperScale configuration setup, the design link step in top-level analysis expects to read the HyperScale block session data from the paths defined `set_hier_config -block -path`. The flow does not require reading Verilog netlist and parasitic files for the corresponding block to link the child design in the top-level design. For example, here is the setup for blkA:

```
set_hier_config -block blkA -path /disk1/blkA/HS_DATA \
    -name fast_func
```

The `link_design` command looks for the block-level session data under the path `/disk1/blkA/HS_DATA/fast_func_all_inst` and uses the session for reading block data for linking. If it does not find the data under the location, the tool issues a link error.

If you specify the `-instances` option in the block configuration, the tool looks for the specific instance of the same block:

```
set_hier_config -block blkA -instances U1 \
    -path /disk1/blkA/HS_DATA -name fast_func
```

The `link_design` command looks for the block data under the specified path with the assigned configuration name and instance name appended:

```
/disk1/blkA/HS_DATA/fast_func_U1
```

Block-Level Analysis

To perform block-level HyperScale analysis, use a script similar to the following:

```
set link_path {* slow.db}
# Enable HyperScale analysis
set hier_enable_analysis true

# Read and link design; read constraints
read_verilog BLK.v
current_design BLK
link_design
source $SDC/BLK.sdc

# Infer HyperScale configuration and load top-level context data
read_context topDir

# Update and report timing and noise
```

```
update_timing
report_timing -max_paths 100 -slack_lesser_than 0
report_constraint -all_violators

update_noise
report_noise

# Publish block data for use in top-level analysis
write_hier_data blkDir ...
```

See Also

- [HyperScale Configuration](#)
- [Reading and Linking the Design Data](#)
- [Timing Analysis in HyperScale](#)
- [Writing and Reading HyperScale Data](#)

Block-Level Analysis With Adjusted Context

After the top-level HyperScale analysis is done, true block context data is available to the block-level designers. When the timing update begins, the analysis checks to see if a block-level context from a previous top-level run is available in the top-level session data. If so, this updated block-level context for each port is overlaid on top of the designer's budgeted constraints.

If block-level constraints are missing or unnecessary (such as the missing case analysis constant violation), they are applied or removed. If constraints were provided but with incorrect values (such as the nonbounding input delay violation), the values are corrected. This explains the mechanism through which fixable boundary violations are actually fixed.

The following user-specified boundary constraints are automatically adjusted:

- Data input delays and output delays
- Clock latencies (static and dynamic)
- Input transitions
- Output loadings
- Case analysis values

This context adjustment process does not modify your budgeted block-level constraint script files. They remain unmodified on disk, and the context adjustment process happens in memory during the beginning stages of the `update_timing` command.

Some boundary violations are unresolvable; these violations are the most serious differences in constraints, such as missing or incorrect clock definitions. To be absolutely

safe when it comes to the design intent, the HyperScale flow requires user action rather than making such significant changes to the block-level constraints.

The context adjustment process is not limited to boundary violations. Whenever possible, the HyperScale flow uses the actual block context information in place of the budgeted constraints for all ports of the block. The goal is to provide a block-level analysis that provides the runtime and capacity advantages of a block-level analysis, yet accurately reproduces the block timing as it exists within the top-level design.

Top-Level Analysis

When you perform HyperScale top-level analysis, the analysis works a little bit differently from a standard analysis. The netlists for any HyperScale blocks are automatically obtained from their respective HyperScale session data directories to satisfy the link requirements for those blocks. In addition, the detailed parasitics for the HyperScale blocks are automatically applied. Because the block-level design teams are actively running their own HyperScale block-level analyses, the top-level run automatically uses the latest block session data available from those teams.

To perform top-level HyperScale analysis, use a script similar to the following:

```
set link_path {* slow.db}

# Enable HyperScale analysis
set_app_var hier_enable_analysis true

# Specify HyperScale configuration information for all subblocks
set_hier_config -block BLK -path $STA_DIR/BLK/HS_BLK

# Read and link the design
read_verilog TOP.v # BLK.v is optional if HyperScale block data exists
current_design TOP
link_design

# Optionally allow design mismatches
# set_app_var link_allow_design_mismatch true
#
# Check hierarchy configuration, settings, and session attributes
report_hier_analysis

# Read the constraints
source $SDC/Flat.sdc

# Update and report timing
update_timing
report_timing -max_paths 100 -slack_lesser_than 0

update_noise
report_noise

# Check the clock mapping
```

```
report_clock -map

# Verify that block-level constraints bound top-level context
report_constraint -all_violators
report_constraint -boundary_check -all_violators -verbose

# Publish revised block contexts for subsequent block-level analyses
write_hier_data topDir
```

In bottom-up flows, you do not need the context in block-level runs. To disable context capture and scope checking, set the `timing_save_hier_context_data` variable to `false`.

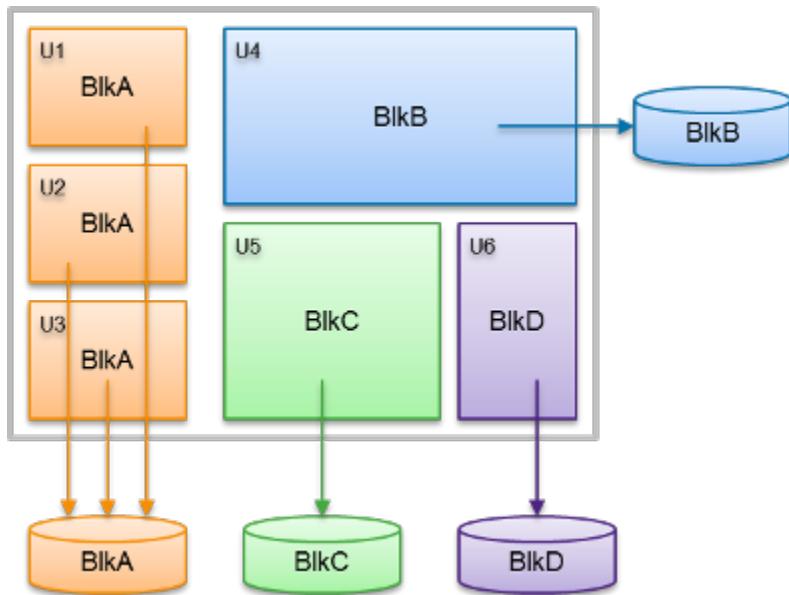
See Also

- [HyperScale Configuration](#)
- [Reading and Linking the Design Data](#)
- [Timing Analysis in HyperScale](#)
- [Block Boundary Checking](#)
- [Writing and Reading HyperScale Data](#)

Multiply Instantiated Modules (MIMs)

A *multiply instantiated module* (MIM) is a block that is instantiated multiple times in the design. For example, [Figure 394](#) shows a top-level design with three instances of subblock BlkA, a multiply instantiated module. The other subblocks, BlkB, BlkC, and BlkD, are used only one time at the top level; they are single-instance modules.

Figure 394 Design with multiply instantiated modules



When you first implement a block-level module, you can treat it as a single entity with budgeted boundary constraints. Next, the block proceeds through the usual synthesis, placement and routing, and extraction steps. After the initial implementation is complete, the block is integrated into the top-level design, where these differences between multiply instantiated modules appear:

- Placement differences – The location and orientation of the multiply instantiated module are different.
- Routing differences – The routing from the top-level nets to the block ports are different. This causes
 - Variation in arrival times at the block ports of each instance
 - Variation in clock latencies at block boundaries
 - Environmental differences such as deratings and cross-couplings

As a result, the subblock design needs to accommodate a different context for each instance. A design with multiply instantiated modules presents challenges in managing the boundary of the block.

In the HyperScale flow, PrimeTime manages the environment around each multiply instantiated module and automatically generates the bounded context for the multiply instantiated module. The generation of the bounded context is called context merging. The bounded context allows block-level analysis with conservatism to ensure that all instances of the block operate at the top level after integration.

When you enable HyperScale in the top-level session, the tool automatically performs context merging during a timing update and writes out the merged context to the session data with the `write_hier_data` command.

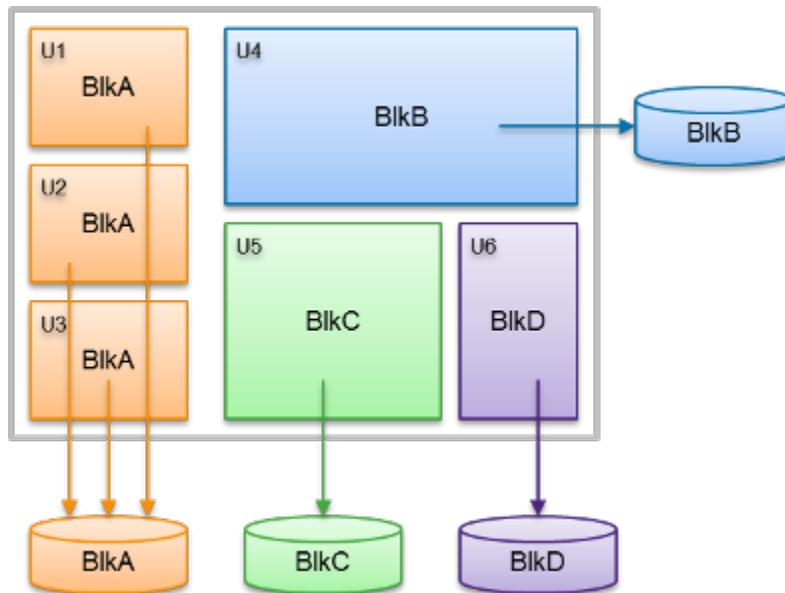
Configuring the Design For MIMs

By default, when you define a subblock with only a block name and no instance name, HyperScale assumes that the block is a multiply instantiated module. If the tool detects more than two instances of the same block at the top level, it automatically enables context merging.

The following example shows the HyperScale design configuration at the top level for the design in [Figure 395](#):

```
set hier_enable_analysis true
set_hier_config -block BlkA -path blkA/HS_DATA -name fast_func
set_hier_config -block BlkB -path blkB/HS_DATA -name fast_func
set_hier_config -block BlkC -path blkC/HS_DATA -name fast_func
set_hier_config -block BlkD -path blkD/HS_DATA -name fast_func
```

Figure 395 Design with multiply instantiated modules



You can use the `-instances` option to force PrimeTime to treat the instances as unique instantiations of the subblock and therefore prevents context merging.

To report and check the design configuration, use the `report_hier_analysis` command.

Linking the Design

After you define the HyperScale configuration for the blocks and read the top-level netlist, PrimeTime uses the block-level HyperScale session data to link the block-level netlist and

parasitic data in the top-level design. PrimeTime automatically performs design linking and uses the HyperScale block data for multiply instantiated subblocks. After design linking completes without errors, you can proceed to update timing and analysis.

Context Data of HyperScale Blocks

The context data includes any necessary data for block-level timing, crosstalk delay, and noise analysis. This data includes not only the SDC constraints but also advanced analysis data, such as common clock pessimism removal (CRPR) credit and AOCV metrics at the boundary.

The context data consists of the data types listed in [Table 66](#). During context merging, the tool automatically chooses the most conservative values for each type of context data.

Table 66 Context data types and values chosen by context merging

Context data type	Value chosen by context merging
Arrival time and slew at the block boundary for both inputs and outputs	Minimum value of all early values and maximum value of all late values (launch paths are not merged if they are from different registers or have any timing exceptions applied)
Clock latency and slew at the boundary clock for the block	
Crosstalk arrival windows per clocks	
Timing derating	
Noise	
Input driving cells	If a different driver or cell library is used on the block port of each instance, the tool uses the driver with the fastest input slew
Case values	If different case values are defined on the equivalent block port of each instance, the tool uses logic X
Timing exceptions	If each instance has difference in exception definition, tool chooses the most constraining case as the setup for single merged context
CRPR credit	Minimum value
AOCV metrics	Minimum values

Context Merging During a Timing Update

During execution of the `update_timing` command, the tool analyzes the multiply instantiated modules and merges the context data, considering multiple scopes from each

instance of the same block. After this process, PrimeTime reports the results of context merging for each multiply instantiated module.

HyperScale generates conservative context data for multiply instantiated modules in the top-level design. Since this makes the block-level design flow more manageable, you can easily refine the block-level design based on the timing, crosstalk, and noise context from the top level, while maintaining single-block implementation.

By default, MIM context merging automatically chooses a reference instance from the largest mergeable instance cluster. To explicitly specify the reference instance, use the `-mim_reference` option of the `set_hier_config` command.

In the top-level analysis, you use the single-block implementation that is refined with merged context. Compared to the standard flat timing analysis, the HyperScale flow might produce more conservative results for the timing paths to each multiply instantiated module.

Successful Context Merging

For the successful context merging of multiply instantiated modules, the instances must exist in same constraint environment. The boundaries of all instances of the same block must have exactly matching clocks and clock topology.

If the context merging completes successfully, PrimeTime issues the following message:

```
Information: HyperScale timing context around instance 'U1' is
successfully merged with instance 'U2'. (HS-100)
```

Failed Context Merging

If the context merging fails, PrimeTime issues the following message:

```
Information: Failed to merge HyperScale timing context around
instance 'U1' with reference instance 'U2'. (HS-103)
```

Usually, context merging fails because multiply instantiated modules have inconsistent clock definitions or clock topology. When this happens, PrimeTime

- Issues the following error message:

```
Error: Clock 'LSCLK' referred in the context for instance 'U1' at
pin 'SCK' has no equivalent clock found in the context for
instance 'U2' of the same block. (HS-017)
```

- Discards the context of the failing instance and keeps only the context of the reference instance

When context merging fails during the `update_timing` command at the top level, you can resolve the problem with either of these steps:

- If you intend to use the multiply instantiated module in same context at the top level, examine the cause of the failure and fix the clock inconsistency issue between the instances.
- Make each multiply instantiated module into a unique block design. To do this, change the block-level run to be specific to each instance by defining the `-instances` option in the HyperScale configuration and creating unique HyperScale block data for each instance. Then you must run block-level timing analysis multiple times to account for the number of unique contexts that exist in the top-level design.

MIM Pessimism Reporting

MIM pessimism occurs when different instances of the same module are subject to different clock latencies. For synthesis and physical implementation, the tools use the worst clock latency value among all instances in the MIM set. Timing pessimism occurs for the instances where less-than-worst clock latencies apply.

The `report_timing` command takes into account the varying clock latency values and corrects this pessimism by adding back the difference between the actual clock arrival for the specific instance and that of the worst-arrival instance, as shown in the following example.

```
pt_shell> report_timing -path_type full_clock -delay_type max \
    -input_pins -max_paths 1 -sort_by slack
...
Startpoint: ffL (rising edge-triggered flip-flop clocked by clk)
Endpoint: Po (output port clocked by clk)
Path Group: clk
Path Type: max

      Point          Incr      Path
-----+
clock clk (rise edge)      0.00      0.00
clock source latency      10.57 @    10.57
clk (in)                  0.00 @    10.57 r
u1/A (B1I)                0.00 @    10.57 r
u1/Z (B1I)                0.57      11.15 r
u2/A (B1I)                0.00      11.15 r
u2/Z (B1I)                0.58      11.73 r
ffL/CP (FD2)              0.00      11.73 r
ffL/Q (FD2) <-            1.77      13.50 r
u4/A (B1I)                0.00      13.50 r
u4/Z (B1I)                0.56 @    14.06 r
Po (out)                  0.00 @    14.06 r
data arrival time          0.00      14.06

clock clk (rise edge)      10.00     10.00
clock network delay (propagated) 0.00     10.00
```

clock reconvergence pessimism	4.28	14.28
MIM pessimism	0.28	14.56
output external delay	5.16 @	19.72
data required time		19.72
<hr/>		
data required time		19.72
data arrival time		-14.06
<hr/>		
slack (MET)		5.67

You can also query the `mim_pessimism` attribute on timing path objects.

Hierarchical Constraint Extraction

You can extract consistent block-level clocks, case values, and exceptions from full-chip flat constraints by using the HyperScale constraint extractor. You can use these extracted constraints to perform context-accurate analysis at the block level. The tool reports any issues in the full-chip flat constraints, so you can resolve the issues before using the extracted constraints in block-level analysis runs.

To learn how to extract and use constraints in HyperScale, see these topics:

- [Configuring and Running Constraint Extraction](#)
- [Constraint Extractor Input and Output Files](#)
- [Block-Level Constraint Assumptions by the Constraint Extractor](#)
- [Unsupported Constraints](#)
- [Block-Level Constraint Errors](#)
- [Inverted Clock Extraction](#)
- [Generating Block Constraint and Block Boundary Context Data](#)

Configuring and Running Constraint Extraction

To configure and run constraint extraction, use a script similar to the following:

```
# Enable constraints_only mode in a separate PrimeTime session
set_app_var hier_characterize_context_mode constraints_only

# Run the standard top-level script; the tool automatically
# skips reading parasitics and SDF
source full_chip_analysis_script.tcl          # Read, link, update

characterize_context -block BlockA
update_timing

# Write constraint files
write_context -format ptsh -output $outDir
```

The tool writes out the following constraint files under the output directory:

```
clock_map.pt  
constraints.pt  
variables.pt
```

Constraint Extractor Input and Output Files

The constraint extractor reads a flat design with full-chip constraints. The design can contain abstracted instances, such as an extracted timing model (ETM), quick timing model (QTM), HyperScale, or black box.

The constraint extractor generates these output files:

- Block-level constraints as Synopsys Design Constraints (SDC) or `write_script` format
 - Clocks
 - Exceptions, excluding crossing-boundary exceptions
 - Case, disable timing
 - Deratings
 - I/O delays, represented as placeholder values that are overwritten by accurate top context during HyperScale block runs
- Nondefault settings of environment and PrimeTime application variables

Block-Level Constraint Assumptions by the Constraint Extractor

The constraint extractor assumes these settings:

- `set_app_var timing_input_port_default_clock false`
The PrimeTime default is `false`; you cannot set this variable to `true` in the HyperScale flow
- The `group_path` command is not written in block-level constraints. If needed, manually add the command.
- `set auto_wire_load_selection false; set_wire_load_mode top`

These are automatically set by HyperScale top; set these variables in the top-level constraint file.

Unsupported Constraints

The constraint extractor does not support these constraints:

- UPF commands
- `set_ocvm_table_group -type pocvm`
- `set_si_delay_disable_statistical`
- `set_clock_sense` is not supported in the SDC format; in the .pt format, `set_clock_sense` is written as `set_sense -type clock`

If such commands are encountered, the constraint extractor issues the HCEXT-011 warning, and you need to manually add those constraints to the block-level constraint files.

Block-Level Constraint Errors

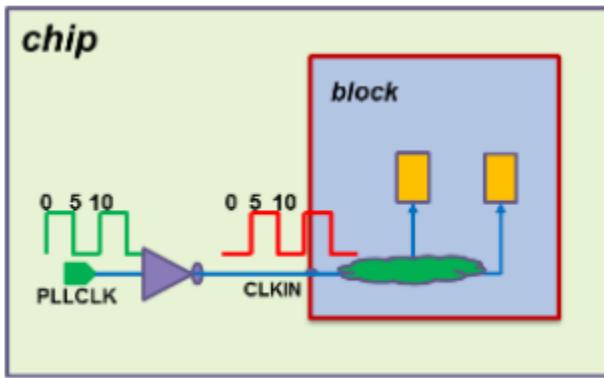
The following errors and warnings report design issues that prevent the extraction of correct block-level constraints; you need to resolve these issues in the flat chip-level constraints:

- HCEXT-004 error: Generated clock definition inside HyperScale block has a problem.
- UITE-461 error: Generated clock is not satisfiable.
- PTE-004 error: The generated clock pin is in a loop or is in the fanout of two clock sources.
- PTE-024 error: A loop of generated clocks is formed in that there is a circular dependency of generated clock to master clock.
- PTE-025 error: The master of the generated clock is not connected to any clock source.
- PTE-023 warning: The generated clock has not been expanded.
- PTE-075 error: Generated clock has no path to its master clock.

Inverted Clock Extraction

Constraint extraction writes consistent constraints that refer to block local waveforms. At the top level, HyperScale automatically performs sense-aware clock mapping and data capturing. In the following example, constraint extraction generates the subsequent constraints.

Figure 396 Sense-aware clock mapping and data capturing



Original constraints:

```
create_clock -name PLLCLK -period 10 -waveform {0 5} [get_pins PLL/OUT]
set_input_delay -clock PLLCLK ...
set_clock_latency -source -rise -clock PLLCLK ...
set_false_path -rise_from PLLCLK ...
...
```

Constraints generated by constraint extraction:

```
create_clock -name PLLCLK -period 10 -waveform {5 10} [get_ports CLKIN]
set_input_delay -clock PLLCLK ...
set_clock_latency -source -fall -clock PLLCLK ...
set_false_path -fall_from PLLCLK ...
...
```

Generating Block Constraint and Block Boundary Context Data

You can generate accurate block-level constraints with allocated I/O delay values or block boundary context data in the following flows:

- Early in the design cycle to automatically allocate rough timing budgets across the blocks.
- Later in the design cycle when most of the blocks are in place. This flow is suitable if you use HyperScale and want to export the top-level context data for block design to other flows, such as implementation and ECO.
- In a hierarchical design flow but with a flat signoff flow. When you use context characterization, the tool imports the entire design netlist, including parasitics, and runs a full timing update. This flow generates block constraints that are consistent with input chip constraints and contain accurate block I/O delays from delay calculation. It is suggested for smaller designs.

Generating Block Constraints

To generate and use block-level constraints:

1. Enable the constraints-only context characterization mode:

```
set_app_var hier_characterize_context_mode constraints_only
```

In this mode, the tool ignores all `read_parasitics` commands and limits the operation of the `update_timing` command to only what is necessary to generate the block-level constraints; no actual timing analysis is performed.

2. Perform context characterization:

```
source full_chip_analysis_script.tcl      # Read, link, update
characterize_context -block blkA          # Generate context for blkA
update_timing
```

3. Write out the constraints using the `write_context` command:

```
write_context -nosplit -format ptsh -output $outDir ...
write_context -nosplit -format sdc -output $outDir ...
write_context -nosplit -format dctcl -output $outDir ...
```

The tool writes out the constraint files under the output directory.

4. Use the generated data during a HyperScale block-level run:

- For the PrimeTime format, use

```
source constraints.pt
```

- For the SDC format, use

```
source constraints.sdc
```

- For the dctcl format, use

```
source constraints.dctcl
```

Performing Manual Clock Mapping

By default, the HyperScale flow performs automatic mapping of clocks between hierarchical levels. You can make the following changes to the default clock mapping:

- Override the automatic clock mapping by using the `set_clock_map` command
- Augment the automatic clock mapping with user-defined clock mapping by using the `set_clock_map -add` command
- Specify N-to-one, one-to-N, and one-to-one clock mappings

To automatically map compatible clocks in one-to-one configurations, but disable automatic mapping of compatible clocks in multi-clock configurations (N-to-one or one-to-N), set the `hier_disable_auto_multi_clock_mapping` variable to `true`. This setting does not disable automatic mapping of different-period clocks that coexist in a one-to-one configuration.

To disable automatic clock mapping entirely, set the `hier_disable_auto_clock_mapping` variable to `true`. If you use this setting, you must manually specify all clock mappings with the `set_clock_map` command.

To allow the automatic mapping of clocks with small differences in frequency or duty cycle, specify a tolerance by setting the `hier_clock_mapping_period_percent_tolerance` variable to a positive fraction.

For signoff accuracy, use a one-to-one clock mapping between top-level and block-level clocks.

Mismatches Between Top and Block Levels

HyperScale technology uses the established static timing analysis signoff flow and enables many people to work simultaneously on different parts of a large chip. As the blocks mature, and each subsequent block-level analysis is performed, the final `write_hier_data` command publishes the updated block for use by the top-level analysis. As the top-level layout matures and each subsequent top-level analysis is performed, the final `write_hier_data` command derives new block contexts using all of the updated block data and publishes the new block contexts for use by the block-level analyses.

Each block's design team decides when to publish the design data with the `write_hier_data` command. When the block is undergoing many design changes, it can be desirable to publish the block's design data only after reaching certain stability milestones. After the block has stabilized and block changes are more controlled, the team can choose to always publish the latest block data.

In most design flows, daily or near-daily block-level and top-level analysis runs are common. With the runtime and capacity advantages provided by the HyperScale flow, this daily run methodology can be applied to very large designs. Furthermore, block-level designers have the benefit of timing their blocks using the most up-to-date context information available from the top level, with top-level effects such as boundary timing, CRPR, split exceptions, and AOCV reproduced accurately in their block-level analyses.

As the design evolves, there can be cases where the port interfaces of the reference blocks are modified by the block-level design teams, but these interfaces changes are not yet reflected in the top-level netlist instance. These port mismatches might result in errors that prevent the linking process to be completed.

To allow these design mismatches during the link process, set the following variable:

```
set_app_var link_allow_design_mismatch true
```

When this variable is set to `true`, the following design mismatches are permitted during linking:

- Instance ports that have been removed from reference
- Bus width mismatch between the reference and instance
- PG pin information inconsistent between the reference and instance
- Pin direction mismatch between the reference and instance

The following variables consider the matching of clock characteristics between the top and block levels:

- `hier_override_clock_sense_from_context` – When set to `true` (the default), HyperScale analysis applies the clock source sense from context when the `read_context` command is applied for block-level analysis. Setting the variable to `false` overrides the default behavior and uses the block-level constraints.
- `hier_context_merge_strict_clock_equivalence` – When set to `true` (the default), HyperScale analysis strictly enforces clock equivalence when merging context across different instances of multiply instantiated modules (MIMs). Setting this variable to `false` relaxes this check by allowing the tool to attempt merging by either finding and using a compatible clock from the reference instance set or by ignoring context differences that are not clock-related.

Block Context Margin

By default, the HyperScale flow pushes the exact top-level conditions down into a block's context. However, subsequent changes in top-level layout and the resulting block contexts might result in small block violations that did not previously exist.

To help stabilize block analysis results during this top-level integration process, you can add timing margin to the block contexts with the `set_context_margin` command. After the top-level layout is stable, you can remove the `set_context_margin` commands to ease timing closure.

The following topics provide more information on block context margin:

- [How Block Context Margin Works](#)
- [Controlling How Margins Are Applied](#)
- [Applying Context Margin From a Top-Level Run](#)
- [Applying Context Margin in a Block-Level Run](#)

How Block Context Margin Works

The `set_context_margin` command adds margin to the clock/data delay values in the block context data written or read by the following commands:

- `write_hier_data` (in a HyperScale top-level run)
- `read_context` (in a HyperScale block-level run)
- `write_context`

Specifically, this margin modifies the delay values of the following commands in the block context data:

```
set_input_delay
set_output_delay
set_clock_latency -source
```

For example, the following command applies 0.50 time units of margin:

```
pt_shell> set_context_margin 0.50 [get_cells BLK*]
```

This adds 0.50 to the `-max` delay values and subtracts 0.50 from the `-min` delay values in the `set_input_delay` and `set_output_delay` commands, resulting in more conservative setup and hold analysis at the block boundary.

Additional `set_context_margin` options (`-min`, `-max`, `-rise`, `-fall`, `-type clock`, `-type data`) can restrict the specification as needed.

Using the `set_context_margin` command triggers a full timing update. To reduce runtime, use the command early in the analysis flow, before the `update_timing` command.

To report margin settings, use the `report_context -margin` command. To cancel a margin setting made by the `set_context_margin` command, use the `remove_context_margin` command.

Controlling How Margins Are Applied

The default behavior of the `set_context_margin` command is to apply the margin as a relative adjustment in time units.

[Table 67](#) lists command options that allow you to change how the margin is applied.

Table 67 Options to Control How Block Context Margin Is Applied

<code>set_context_margin</code> options	Description of how margin is applied
<code>-percent value</code>	Adds margin as a percentage of the existing input or output delays.

Table 67 Options to Control How Block Context Margin Is Applied (Continued)

set_context_margin options	Description of how margin is applied
-percent value -allocation pin_slack	Adds margin as a percentage of slack (positive or negative). Positive slacks tighten the timing, while negative slacks relax the timing.
-percent value -allocation clock_period	Adds margin as a percentage of clock period.
-allocation user_override	<i>Completely replaces</i> the existing input or output delays with the specified value in time units. Use this option with caution, as relaxed delay values cause optimism.

For more details, see the man page.

Applying Context Margin From a Top-Level Run

In a top-level run, you can apply margin to block instance cells or instance pins, or you can omit the object list to apply margin to all block instances:

```
set_context_margin 0.2      ;# apply to all blocks
set_context_margin 0.4 [get_cells {UBLK*}]
set_context_margin 0.5 [get_pins {UBLK2/RSTN}]
...
write_hier_data
```

The margins are included in the block context data written out by the `write_hier_data` command. Block context margins specified in a top-level analysis *do not* affect the top-level analysis results.

The `set_context_margin` command offers the following options:

- The amount of the margin adjustment, and whether to apply the margin as an absolute amount, a percentage of the existing delay, or a percentage of the clock period
- The allocation of available slack between the block and top levels
- Whether to apply the margin to maximum or minimum constraints only, to rise or fall transitions only, and to data or clock paths only, or both
- Whether to apply the margin to all blocks for which the context is being written or only specific blocks

Applying Context Margin in a Block-Level Run

In a block-level run, you can apply margin to specific ports:

```
read_context HS_TOP -name BLK
set_context_margin 0.4 [get_ports *]
set_context_margin 0.7 [get_ports RX*]
update_timing
```

The margins are applied to the context data read in by the `read_context` command. If no context data is applied, the margins do not apply. If the context data from the top-level run already contains margins, they are combined additively.

Note the following restrictions for margins applied in a block-level run:

- Only port-specific margin specifications are supported. Global margin specifications are not supported.
- The `-allocation pin_slack`, `-allocation clock_period`, and `-percent` options of the `set_context_margin` command are not supported.

Block Boundary Checking

In a HyperScale flow, a top-level analysis uses block models written out by block-level analysis. A block model contains only the boundary logic of the block; the internal logic is abstracted away because it was already analyzed in the block-level run. To ensure correct analysis of the abstracted block logic, a top-level analysis should not analyze a block in a context *outside the context it was analyzed at* in its block-level run.

The `report_constraint` command provides a boundary checking feature for top-level runs that reports when a block is used outside its context. This includes not only numerical checks (such as arrival, transition, and DRC characteristics), but also existence and correctness checks (such as clocks, case analysis, and tool configuration).

For example, if a clock arrives at the input pin of a HyperScale block, but that clock was not defined (or was defined improperly) in the block-level constraints, the mismatch is reported.

Although these checks are performed in a top-level run, the following variable must be set to `true` in both the block-level and top-level analysis runs:

```
pt_shell> set_app_var timing_enable_hier_boundary_checks true
```

The `report_constraint` command does not report boundary checks by default. To report them, use the `-boundary_check` option:

```
pt_shell> report_constraint -boundary_check
```

There are two types of block boundary violations:

- **Automatically Fixable Block Boundary Violations** – These violations indicate a boundary violation; however, HyperScale automatically adjusts the block-level boundary context during the next block-level run to resolve the violation.
- **Manually Fixable Block Boundary Violations** – These violations indicate serious problems with the budgeted block-level constraints. HyperScale cannot automatically fix these violations; you must manually adjust the constraints.

By default, the `report_constraint -boundary_check` command reports only manually fixable checks. You can use the `-boundary_check_include` option to specify what check types to report:

- To see only the manually fixable violations (the default), use the `-boundary_check_include non_auto_fixable` option.
- To see only the automatically fixable violations, use the `-boundary_check_include auto_fixable` option.
- To see all boundary check violations, use the `-boundary_check_include all` option.
- To report specific violation categories, use the `-boundary_check_include` option with the values in the following table.

Table 68 Block Boundary Checks

Violation type	Name of boundary check
Automatically fixable	<code>boundary_logic_value</code> <code>clock_latency</code> <code>clock_skew_with_uncertainty</code> <code>data_arrival</code> <code>input_slews</code>
Manually fixable	<code>boundary_ideal_network</code> <code>clock_attributes</code> <code>clock_mapping</code> <code>clock_relations</code> <code>clock_uncertainty</code> <code>env_variables</code> <code>global_timing_derate</code> <code>library_mapping</code> <code>operating_conditions</code>

By default, a summary of the total number of check violations of each type is reported. You can use the `-all_violators` and `-verbose` options to see more details:

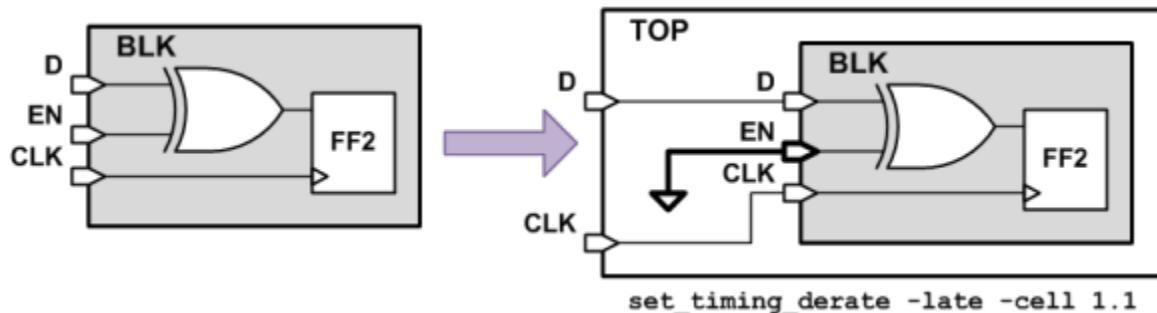
- `-all_violators` – reports the total number of check violations for each block
- `-verbose` – reports verbose details of the worst check violation of each type across all blocks
- `-verbose -all_violators` – reports verbose details of all check violations

Automatically Fixable Block Boundary Violations

HyperScale automatically fixes certain block boundary violations with a context adjustment. Examples of automatically fixable violations include arrival time, clock latency, case analysis, and timing exceptions.

The following example shows some fixable block boundary violations. In [Figure 397](#), no case analysis constant was specified on the EN port in the block-level analysis. However, a constant does actually arrive at block pin BLK/EN in the top-level analysis. In addition, a late timing derating is specified at the top level but not at the block level.

Figure 397 A fixable block boundary violation



In the top-level analysis, the `report_constraint -boundary_check` command reports the following boundary violations:

HyperScale constraints report

Constraint	Num_Violations
<hr/>	
* data_arrival	1
* global_timing_derate	1
* boundary_logic_value	1

In addition to the case analysis and derating differences, the derating differences have also caused differences in arrival times at the D pin of BLK. The `report_constraint` command marks these violations with an asterisk (*) in the summary report. The asterisks indicate that the HyperScale analysis can adjust the constraints in the next block-level run to resolve these violations.

For details about the violations, use the `report_constraint -boundary_check_include {all} -verbose` command:

Constraint: `global_timing_derate`

Instance	Derate type	Window type	Block	Top	Slack
BLK	data_cell	max_rise	1.00	1.10	-0.10
BLK	data_cell	max_fall	1.00	1.10	-0.10
BLK	clock_cell	max_rise	1.00	1.10	-0.10
BLK	clock_cell	max_fall	1.00	1.10	-0.10

Constraint: `boundary_logic_value`

Instance	Pin	Block	Top
BLK	BLK/EN	0	1

Constraint: `data_arrival`

Instance	Pin (Port name)	Window type	CLK	Block	Top	Slack
BLK	BLK/U1/A(D)	max_rise	CLK(r)	0.00	0.25	-0.25
BLK	BLK/U1/A(D)	max_fall	CLK(r)	0.00	0.20	-0.20

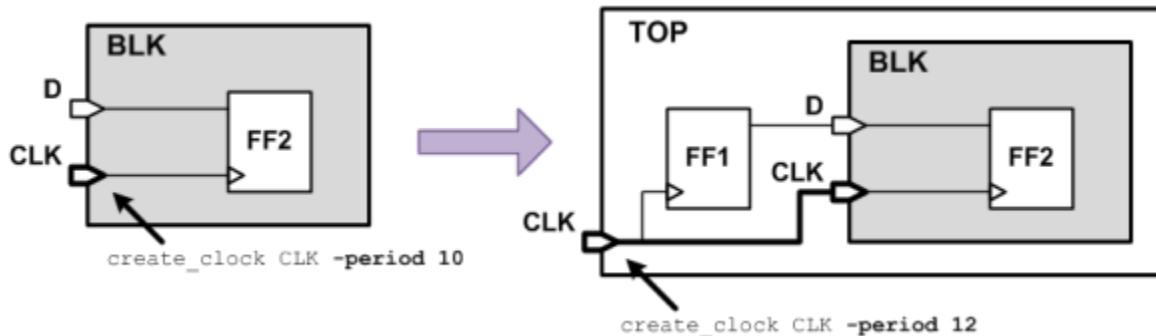
Automatically fixable violations do not require you to modify the original constraints. The HyperScale analysis applies or adjusts the block context, as needed, in the next block-level analysis run. However, it might be desirable to update the budgeted block-level constraints so that future non-HyperScale block-level runs benefit from the improved block constraints.

Manually Fixable Block Boundary Violations

You need to manually fix certain types of violations, such as clock mapping, clock attribute violations, or mismatching clock characteristics.

[Figure 398](#) shows an example in which the clock period was incorrectly defined as 10 in the block-level analysis, when the actual clock period arriving from the top level is 12.

Figure 398 A Manually Fixable Block Boundary Violation



To report block boundary violations, use the `report_constraint -boundary_check` command. This example shows the report output:

HyperScale constraints report

Constraint	Num_Violations
clock_mapping	1

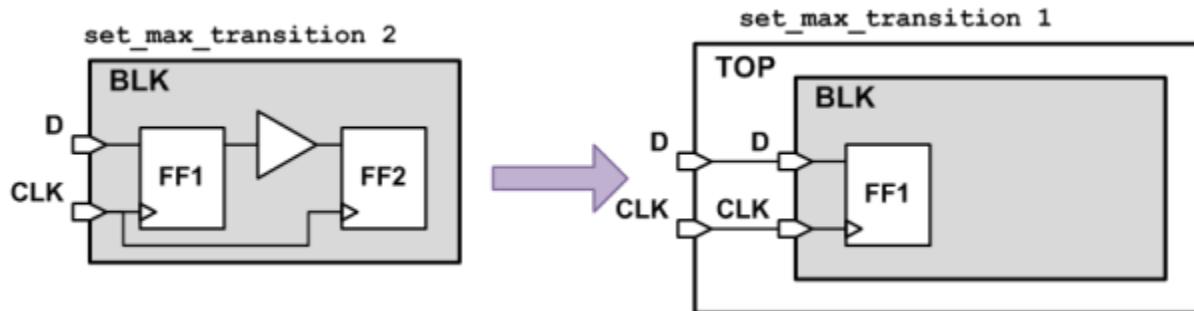
To see more information about the error, use the `-verbose` option:

Constraint: `clock_mapping`

Instance	Block level clock	Top level clock	Attributes	Block clock ref-pin
BLK	CLK	---	<code>p=12.00,e={0.00 6.00},a={p A} {BLK/CLK}</code>	

Another example of an unresolvable block boundary violation is the existence of a more restrictive design rule requirement in the top-level analysis versus the block-level analysis. [Figure 399](#) shows an example where a `max_transition` requirement of 2 is applied by the block-level constraints; however, a more restrictive `max_transition` requirement of 1 is applied by the top-level constraints. As a result, the tighter top-level `max_transition` requirement cannot be applied to the internal logic of the block, resulting in an incomplete `max_transition` analysis.

Figure 399 Another Manually Fixable Block Boundary Violation



The `report_constraint -boundary_check` command reports the following error:

HyperScale constraints report

Constraint	Num_Violations
global_drc	1

To see more information about the error, use the `-verbose` option:

Constraint: `global_drc`

Instance	DRC type	Clock	Block	Top	Slack
BLK	max_transition	*	2.00	1.00	-1.00

A third example of a manually fixable boundary violation is a difference in application variable settings between the block- and top-level analyses. Consider a case where clock reconvergence pessimism removal (CRPR) is disabled at the block level but enabled at the top level. The `report_constraint -boundary_check` command reports the following error:

HyperScale constraints report

Constraint	Num_Violations
env_variables	1

To see more information about the error, use the `-verbose` option:

Constraint: `env_variables`

Instance	Attribute	Block	Top
BLK	timing_enable_clock_reconvergence_pessimism	0	1

To write out all nondefault application variable settings in the current analysis, use the `write_app_var -only_changed_vars` command. By using this command, you can write

the variable settings from one analysis, such as the top-level analysis, and apply them to another analysis, such as a block-level analysis.

Unresolvable boundary violations should be treated like any other timing or design rule violation reported by the `report_constraint` command. An unresolvable boundary violation must be fixed before the analysis is considered clean. Sometimes, the violation can be fixed by modifying the top-level logic or constraints to improve the top-level context to meet the block boundary requirement. At other times, the violation can be fixed by updating the block-level constraints to properly bound or match the top-level context.

The following example shows a clock mapping violation that is detected in `report_constraint -boundary_check` in top-level analysis:

```
HyperScale constraints report
```

Constraint	Num_Violations
clock_mapping	2
clock_attributes	1
clock_uncertainty	4
* data_arrival	8741
* clock_latency	8
* input_slews	4452

This example is the default report in top-level analysis, which shows the summary of all boundary constraint violations in the current design. It indicates two `clock_mapping` violations and others, which is a manually fixable violation. Next, check the origin of violation by using the `report_constraint -boundary_check -all_violators` command, which shows the block constraint that mismatches the top-level constraint. The `-all_violators` report shows the number of violations per instance per type of violation.

In the following example, the `u0_1/pd_switchable` instance encounters two `clock_mapping` violations from different causes: one due to an extra clock specified, and one due to a missing clock in the block.

```
HyperScale constraints report
```

```
Constraint: clock_mapping
```

Instance	Num_vio	Reason
u0_1/pd_switchable	1	extra_clock
u0_1/pd_switchable	1	missing_clock
Constraint: clock_attributes		
Instance	Num_vio	Reason
u0_2/pd_switchable	1	non-propagated block clock

```
Constraint: clock_uncertainty
```

Instance	Num_vio	Reason

```
-----  
u0_3/pd_switchable           4          top/block mismatch
```

To further debug the clock violation for a possible fix, generate a verbose report of the instance:

```
pt_shell> report_constraint -boundary_check -all_violators \
           -verbose [get_cell u0_1/pd_switchable]
...
HyperScale constraints report
Constraint: clock_mapping
Instance      Block   Top   Attributes      Block
           level   level
           clock   clock
           ref_pin
-----
u0_1/pd_switchable  pci_clk  ---  p=34.0000,
                           e={0.0000 17.0000},
                           a={p A}           {u0_1/pd_switchable/clk}
u0_1/pd_switchable  ---      pci_clk  p=36.0000,
                           e={0 18},
                           a={p A}           {u0_1/pd_switchable/clk}
```

The verbose report of the instance provides details on each clock mapping violation. The first violation happens on the block-level clock, pci_clk, and the tool did not find a matching clock that arrives to the block boundary source in the top-level constraint. The second violation seems related to the first violation because it is the same clock but missing a block-level clock. In this case, both clocks arrive at the same source reference pin, but the clock period does not match. This clock attribute difference caused two `clock_mapping` violations, one is missing block-level clock and the other missing top-level clock. To fix this problem, correct the `create_clock` constraint in the block-level run. Confirm that this problem is resolved during top-level analysis after block analysis with the new constraint.

Consider another manually fixable violation, `clock_attributes`:

```
pt_shell> report_constraint -boundary_check -all_violators -verbose \
           [get_cell u0_2/pd_switchable]
...
HyperScale constraints report
Constraint: clock_attributes
Instance      Attribute name  Block level (clock)  Top level (clock)
-----  
u0_2/pd_switchable  is_propagated  false        (pci_clk)  true       (pci_clk)
```

The second instance seems to have a mismatch in clock attribute, `is_propagated`. To resolve this issue, set `set_propagated_clock` for given clock in the block analysis.

The following is verbose report on remaining instance, u0_3/pd_switchable that has a `clock_uncertainty` violation:

```
pt_shell> report_constraint -boundary_check -all_violators -verbose \
           [get_cell u0_3/pd_switchable]
...
HyperScale constraints report
Constraint: clock_uncertainty
Instance          Type      From Clock        To Clock        Block    Top     Slack
-----
u0_3/pd_switchable setup   pci_clk (rise)  pci_clk (rise)  0.0000  0.2000 -0.2000
u0_3/pd_switchable setup   pci_clk (rise)  pci_clk (fall)   0.0000  0.2000 -0.2000
u0_3/pd_switchable setup   pci_clk (fall)   pci_clk (rise)   0.0000  0.2000 -0.2000
u0_3/pd_switchable setup   pci_clk (fall)   pci_clk (fall)   0.0000  0.2000 -0.2000
```

The report indicates differences in the clock uncertainty value. The block level has zero uncertainty specified for `pci_clk` clock. However, in top-level constraint, `clock uncertainty` was defined and the value is 0.2 time unit. By adding clock uncertainty in block-level constraint would address this boundary constraint violation.

Other boundary constraint violations that are fixable by a HyperScale context update are reduced by continuing the refinement of the flow. Running block-level analysis with the context update from the previous top-level analysis and continuing on the next round of the top-level analysis results in reduction of the violation count for automatically fixable violations.

Clock Mapping Check

In hierarchical analysis, you need consistent clock definitions between block-level and top-level analysis. HyperScale establishes the clock relationships between the block level and top level automatically. Complete clock mapping is required for HyperScale to ensure good accuracy in timing analysis. The tool maps the clocks for each HyperScale instances in current design, including physical, virtual, and generated clocks, as well as inverted and noninverted clocks across the block boundary.

The tool uses the following mapping rules in priority order:

- Sharing the same physical clock source network
- Exact match in clock period and waveform
- Matching clock names as prefix

If the clock mapping is not successful, the tool issues the HS-004 error message but completes `update_timing`.

Error: There are unresolved clocks in block BLK. (HS-004)

To report the clock mapping results, use the `report_clock -map` command after `update_timing`. The report lists the clocks at the top level and the clocks in the block-

level session side that are mapped to top-level clock in same row per each instance in the design. If any column value shows “N/A”, it means the corresponding clock was not found in either block level or top level. This directly corresponds to the HS-004 error message.

This example shows a clock mapping report:

Instance	Top level clock	Block level clock	References
U1	CLK	CLK	U1/CLK
	SCLK	SCLK	U1/SCLK
U2	CLK	CLK	U2/CLK
	N/A	TCLK	U2/TCLK

The “References” column indicates the pin or port where the clock mapping occurs:

- In a top-level run (indicated by block instance names in the “Instance” column), the block instance pin where the mapping occurs is shown.
- In a block-level run (indicated by <design> in the “Instance” column), the input port where the clock mapping occurs is shown.
- In either type of run, for virtual block-level clocks that do not physically enter the block, a value of (<virtual>) is shown.

If “N/A” shows in the “Top level clock” column, it means that the tool did not find a corresponding top-level clock to map to the clock that is found in block session data or there is no top-level clock arriving at the clock source at the block boundary. If “N/A” shows in the “Block level clock” column, it means that the tool did not find a block-level clock to map to the top-level clock arriving at the block boundary. These types of clock mapping issues typically appear as clock mapping boundary constraint violations in the `report_constraint -boundary_check` output.

When a top-level clock arrives at a block boundary, and there is a clock source from the block session data, it is possible to have a clock mapping issue if clock attributes do not match. Those attributes include clock period and waveform. This typically results in a `clock_attributes` boundary constraint violation in the `report_constraint -boundary_check` output.

The HS-004 clock mapping error and “N/A” in the `report_clock -map` report indicates major inconsistency in timing constraints between block- and top-level analyses. You should address them and resolve to zero clock violations. One way to ensure fully consistent clock constraints is to use the HyperScale constraint extractor. For more information, see [Hierarchical Constraint Extraction](#).

Clock Mapping in SMVA/DVFS Multivoltage Designs

The simultaneous multivoltage analysis (SMVA) feature allows multiple voltage conditions to be analyzed simultaneously. In addition, dynamic voltage and frequency scaling (DVFS) scenarios allow the operating voltage configurations to be explicitly defined.

Clocks can be defined by DVFS scenario by using the `-dvfs_scenarios` option. During clock mapping, if multiple top-level clocks match a block-level clock, the top-level clock with the matching period is selected:

Top-level clocks

```
create_clock -period 1.2 tclk \
-dvfs_scenario $top_low_blk_low
create_clock -period 1.4 clk \
-dvfs_scenario $top_hi_blk_low
create_clock -period 1.6 tclk \
-dvfs_scenario $top_low_blk_hi
create_clock -period 1.8 clk \
-dvfs_scenario $top_hi_blk_hi
```

Block-level clocks

```
create_clock -period 1.2 bclk \
-dvfs_scenario $blk_low
create_clock -period 1.6 bclk \
-dvfs_scenario $blk_hi
```

Infinite-window aggressors
used in clock/data fanout

If unmapped top-level clocks remain for a clock port, then the block context also propagates “clockless” arrival windows from the clock port and its associated data ports. This causes infinite-window aggressors in the fanout, ensuring that the block-level analysis bounds the top-level analysis.

Note the following limitation:

- Block contexts must be written in binary format (`write_context -format gbc`). ASCII block contexts do not include DVFS-specific information, although they can be augmented manually.

See Also

- [SMVA Graph-Based Simultaneous Multivoltage Analysis](#)

Querying Clock Mapping Using Collections

You can use the `get_hier_clocks` command to query hierarchical clock definitions in HyperScale block-level and top-level runs.

The `get_hier_clocks` command returns `hier_clock` collection objects instead of `clock` objects. `hier_clock` objects cannot be used in place of `clock` objects in reporting commands, but their attributes can be reported with the `report_attributes` command.

Consider the following block-to-top clock map:

Instance	Top level clock	Block level clock	References
MY_BLK1	TOP_CLK1	BCLK1	MY_BLK1/BCLK1 (<virtual>)
	TOP_CLK2	(N/A)	
MY_BLK2	TOP_CLK3	BCLK2	MY_BLK2/BCLK2 (<virtual>)
	TOP_CLK4	(N/A)	

These clocks can be queried using the `hier_clock` attribute as follows.

Querying Top-Level Clocks in a Block-Level Run

In a block-level analysis, the `get_hier_clocks` command returns the top-level clocks defined in the block's context, whether selected by the clock mapping or not:

```
pt_shell> get_clocks
{"BCLK1"}
pt_shell> get_hier_clocks
{"TOP_CLK1", "TOP_CLK2"}
```

The `sources` attribute of each `hier_clock` object indicates the port where the top-level clock reaches the block:

Design	Object	Type	Attribute Name	Value
BLK1	TOP_CLK1	string	edges	0.0000 5.0000
BLK1	TOP_CLK1	string	full_name	TOP_CLK1
BLK1	TOP_CLK1	boolean	is_active	true
BLK1	TOP_CLK1	boolean	is_generated	false
BLK1	TOP_CLK1	boolean	is_propagated	false
BLK1	TOP_CLK1	float	period	10.000000
BLK1	TOP_CLK1	collection	sources	BCLK1
BLK1	TOP_CLK2	string	edges	0.0000 5.5000
BLK1	TOP_CLK2	string	full_name	TOP_CLK2
BLK1	TOP_CLK2	boolean	is_active	true
BLK1	TOP_CLK2	boolean	is_generated	false
BLK1	TOP_CLK2	boolean	is_propagated	false
BLK1	TOP_CLK2	collection	mapped_clock	BCLK1
BLK1	TOP_CLK2	float	period	11.000000
BLK1	TOP_CLK2	collection	sources	BCLK1

Querying Block-Level Clocks in a Top-Level Run

In a top-level analysis, the `get_hier_clocks` command returns all block-level clock definitions by default. However, you can also return the block-level clocks for a given HyperScale block instance with the `-of_objects` option:

```
pt_shell> # this is the top-level analysis
pt_shell> get_clocks
{"TOP_CLK1", "TOP_CLK2", "TOP_CLK3", "TOP_CLK4"}
pt_shell> get_hier_clocks
{"BCLK1", "BCLK2"}
pt_shell> get_hier_clocks -of_objects [get_cells MY_BLK1]
{"BCLK1"}
pt_shell> get_hier_clocks -of_objects [get_cells MY_BLK2]
{"BCLK2"}
```

The `sources` attribute of each `hier_clock` object indicates where a clock is defined in a block context, and the `mapped_clock` attribute indicates which top-level clock is mapped to it:

```
pt_shell> report_attributes -application [get_hier_clocks]
...
Design    Object      Type          Attribute Name   Value
-----
top       BCLK1      string        edges           0.0000 6.5000
top       BCLK1      string        full_name      BCLK1
top       BCLK1      boolean       is_active      true
top       BCLK1      boolean       is_generated  false
top       BCLK1      boolean       is_propagated false
top     BCLK1    collection  mapped_clock  TOP_CLK1
top       BCLK1      float         period         11.000000
top     BCLK1    collection  sources       MY_BLK1/BCLK

top       BCLK2      string        edges           0.0000 6.0000
top       BCLK2      string        full_name      BCLK2
top       BCLK2      boolean       is_active      true
top       BCLK2      boolean       is_generated  false
top       BCLK2      boolean       is_propagated false
top     BCLK2    collection  mapped_clock  TOP_CLK3
top       BCLK2      float         period         12.000000
top     BCLK2    collection  sources       MY_BLK2/BCLK
```

HyperScale Session Data

The HyperScale flow requires the saving and restoring of session data using the `write_hier_data` and `read_context` commands.

Writing and Reading HyperScale Data

To write the data generated by the current HyperScale analysis, run the `write_hier_data` command:

```
write_hier_data
[-netlist]
[-parasitics spef_format | gpd_format]
[-config]
[-exclude parasitics_data]
directory_name
```

To read the HyperScale data, run the `read_context` command before running `update_timing`:

```
read_context
[-name mim_group]
directory
```

In addition, you can use the `write_sdc` and `write_script` commands in HyperScale under the following situations:

- At the leaf block level
- When no context override has been done

HyperScale Data Directory Structure

The `write_hier_data` command writes the HyperScale data in the directory structure shown here.

Figure 400 Top-Level Directory for HyperScale Data

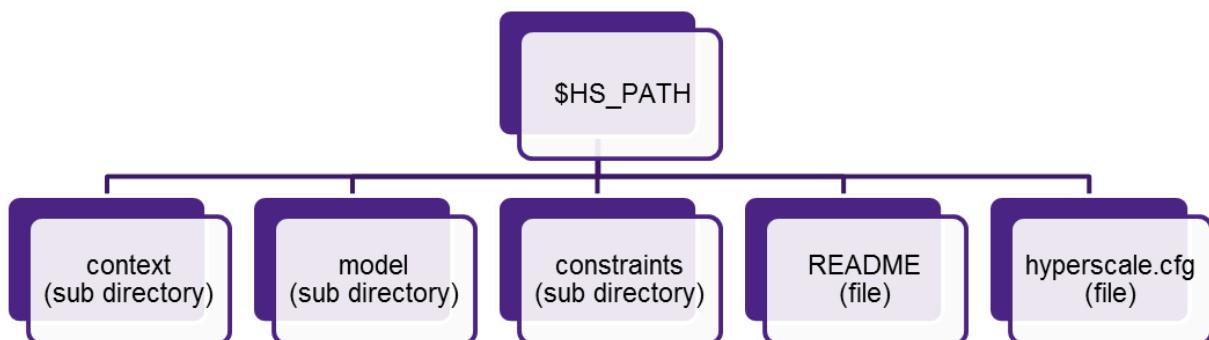


Figure 401 The Context Subdirectory

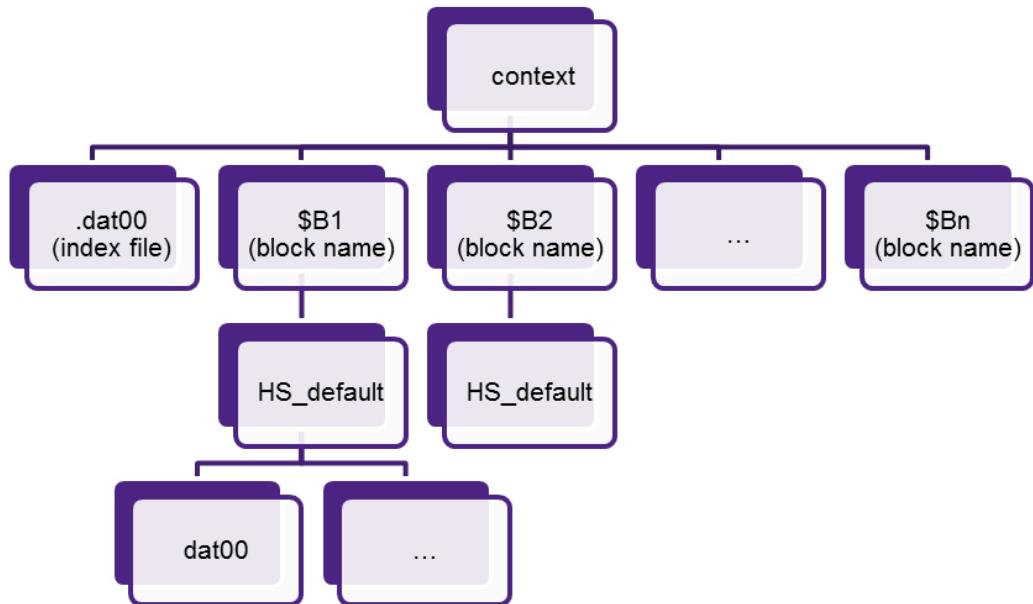


Figure 402 The Model Subdirectory

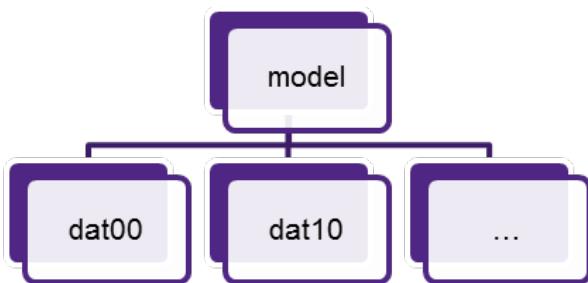
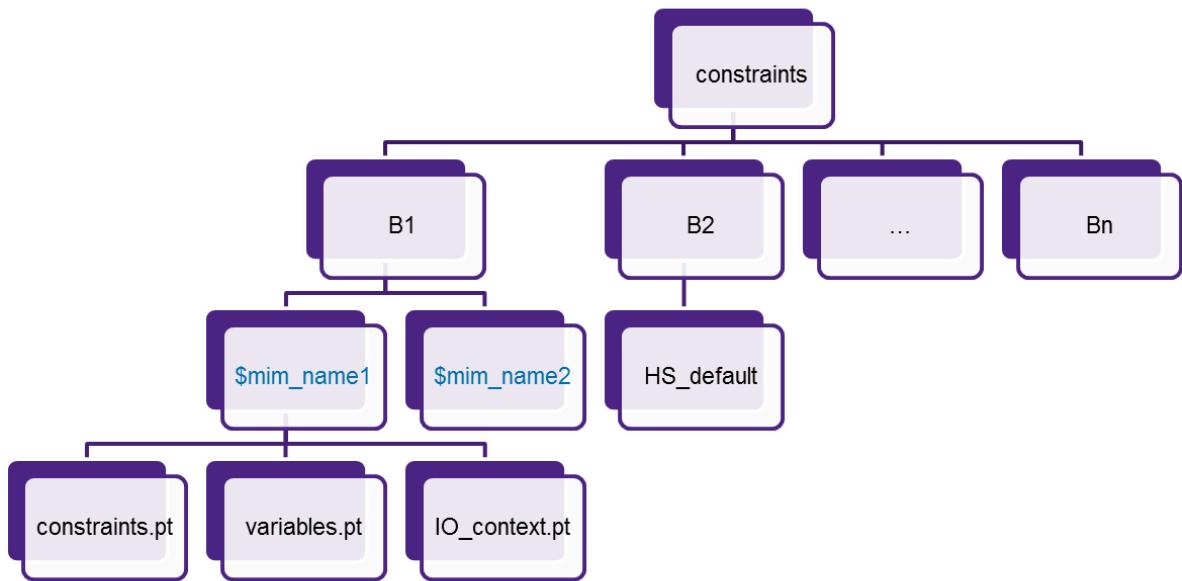


Figure 403 The Constraints Subdirectory



Exporting HyperScale Top Data

You can use the `write_sdc` and `write_script` commands at the HyperScale top level with the following limitations:

- I/O constraints, such as `set_input_delay` and `set_output_delay`, are filtered from the resulting SDC or script file.
- Some data in the HyperScale database cannot be expressed by `write_sdc` or `write_script`.
 - The script or SDC cannot be used to match HyperScale results in PrimeTime using these scripts.
 - The results are not guaranteed to be pessimistic.
- Top- and mid-level runs have “holes” in the logic, so constraints referring to these objects are missing.
 - Timing exceptions, such as multicycle paths, false paths, `set_min_delay`, and `set_max_delay`, with objects outside the block are removed.
 - Some constraints need to be added manually.

Reloading a HyperScale Block Model

For debugging purposes, you can load a block-level model into the tool without having to run the entire top-level analysis.

To do this, include the following option when writing your block model:

```
pt_shell> write_hier_data ${HS_DIR}/BLK -include {model_reload_data}
```

This creates an additional RELOAD directory (alongside the existing MODEL directory) in the block model directory. You can then start a new PrimeTime session and read in the block model using the `read_hier_data` command:

```
pt_shell> read_hier_data ${HS_DIR}/BLK/RELOAD
```

This is similar to a `restore_session` operation, but for a HyperScale block model. You can then explore the preserved logic and other design data in the block model.

Writing Netlists and Constraints for a HyperScale Block

For debugging purposes, you can write out ASCII design data files for a HyperScale block model, which include only the boundary-logic design data.

To do this, use the `-netlist` option of the `write_hier_data` command. [Table 69](#) summarizes the output files that can be written.

Table 69 `write_hier_data` Commands to Write ASCII Design Data Files

<code>write_hier_data</code> options	Block model output files
<code>write_hier_data -netlist</code>	<code>BLK_Hyperscale.v.gz</code> <code>BLK_Hyperscale.spf.gz</code>
<code>write_hier_data -netlist \ -exclude parasitics</code>	<code>BLK_Hyperscale.v.gz</code>
<code>write_hier_data -netlist \ -include model_constraints_data</code>	<code>BLK_Hyperscale.pt.gz</code>
<code>write_hier_data -netlist \ -include model_constraints_data \ -script_format sdc</code>	<code>BLK_Hyperscale.sdc.gz</code>

When you use the `-netlist` option, the usual binary block model data *is not* written to the specified output directory.

Database and Configuration Reports

The `write_hier_data` command generates the following files in the HyperScale directory:

- [The README file](#)
- [The hyperscale.cfg file](#)

The README file

The README file contains Information about HyperScale session data, including

- PrimeTime version
- Time stamp
- Paths of generated and loaded data for current model and context

Figure 404 README File Example

```
PrimeTime Version:
    I-2013.12 for amd64 -- Dec 09, 2013
Save Session Date:
    Mon Dec 10 18:03:25 2013
Current Design: top

Type: HyperScale Top Context

Block Name      Instances   Generated context path          Loaded block model path
-----
-----  

block  hs_default  <all_inst>  mid_path/block/hs_default_all_inst  block_path/
hs_default_all_inst

Type: HyperScale Block Model
```

The hyperscale.cfg file

The `hyperscale.cfg` file facilitates database portability. It is similar to the `lib_map` text file in the standard session image.

To update the configuration file, specify the paths to the new locations:

```
set hier_enable_analysis true
set_hier_config -name hs_default -block block -path block_path
```

HyperScale ECO Closure Methods

HyperScale hierarchical analysis supports the implementation and evaluation of design changes using the engineering change order (ECO) process with the PrimeTime, StarRC, and IC Compiler or IC Compiler II tools, including setup and hold fixing, DRC fixing, and power recovery.

You can use commands like `insert_buffer`, `remove_buffer`, and `size_cell` to make changes. Netlist editing commands such as `create_cell`, `create_net`, and `connect_net` are generally supported, except in certain cases where a change affects a large part of the design beyond the logical interface between hierarchical levels.

To learn about using the HyperScale ECO flow, see these topics:

- [The PrimeTime ECO Flow With HyperScale](#)
- [Using Updated Block Contexts for Block-Level ECOs](#)
- [Context Update Supports Accurate Fixing of Internal Paths](#)
- [Fixing Block Boundary Paths](#)
- [ECOs for Multiply Instantiated Modules \(MIMs\)](#)
- [Resolving Block Violations in the ECO Flow](#)
- [Performing ECOs at the Top Level](#)
- [Running ECOs in Parallel for Runtime and Capacity](#)
- [HyperScale ECO Interface to IC Compiler and StarRC](#)

The PrimeTime ECO Flow With HyperScale

HyperScale supports the PrimeTime ECO flow with the following capabilities:

- Top-level ECO power recovery and fixing of setup, hold, and design rule constraint (DRC) violations extends into the block interface logic. This capability applies to logical and physically aware ECO flows and is compatible with distributed multi-scenario analysis (DMSA) and multiply instantiated modules (MIMs).
- ECO changes can be made on port nets at the block level with context.

Top-Level ECO Fixing

To perform top-level ECO, use the following flow:

1. Start top-level analysis with the initial block model.
2. Perform ECO fixing for both the top logic and block interface logic by using the `fix_eco_timing`, `fix_eco_drc`, and `fix_eco_power` commands.
3. Generate the change list by using the `write_changes` command, which generates these files:
 - A separate change list for each DEF file in physically aware ECO
 - A single change list for logical ECO

4. Implement the top-level ECO change list in the IC Compiler place-and-route tool and the StarRC parasitic extraction tool.
5. Validate the top-level ECO changes in HyperScale. Repeat steps 2 to 4 as ECO iterations until convergence. No further ECO fixing can be done on top-level logic.
6. Use the `save_session` command to generate post ECO context.

Block-Level ECO Fixing

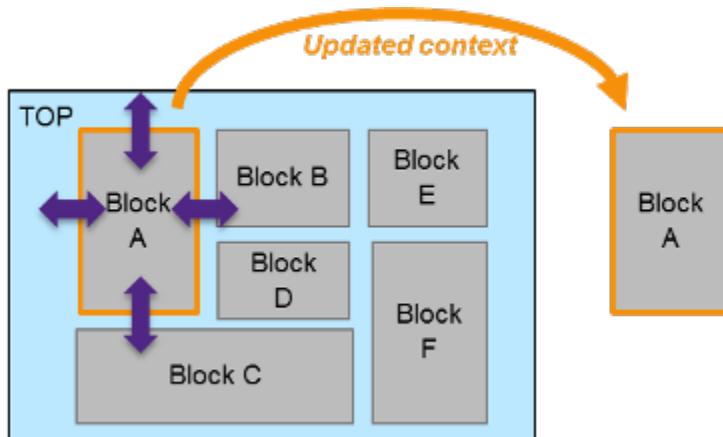
To perform block-level ECO, use the following flow:

1. Implement the latest block-level ECO change list from top in IC Compiler and StarRC. Obtain the new netlist and parasitic data.
2. Use the new netlist and parasitic data and load the latest post-ECO context from top.
3. Run block-level analysis and ECO iterations with the top context to perform fixing on block internal and interface paths until convergence.
4. Implement the final ECO changes in IC Compiler and StarRC.
5. Use the `save_session` command to create a block model that represents the implemented netlist and timing.

Using Updated Block Contexts for Block-Level ECOs

With an updated and accurate block context, the timing for internal block paths and boundary block paths is more accurate than the original block timing when using budgeted constraints. In [Figure 405](#), Block A is isolated to show an example of the improvements provided in the HyperScale flow for ECO fixing.

Figure 405 HyperScale Top Analysis Provides Updated Block Context for ECO Fixing



The updated and accurate context supplied to Block A from the top level provides specific updates to the clock data signal constraints and other conditions that represent the

environment seen in a standard flat chip run. The block-level HyperScale run includes all logic of the block, and the fixing includes both block internal (register-to-register) paths as well boundary paths (input-to-register, output-to-register).

Context Update Supports Accurate Fixing of Internal Paths

For block internal (register-to-register) paths, HyperScale updates the following block boundary conditions, which have a direct effect on setup, hold, and DRC violations:

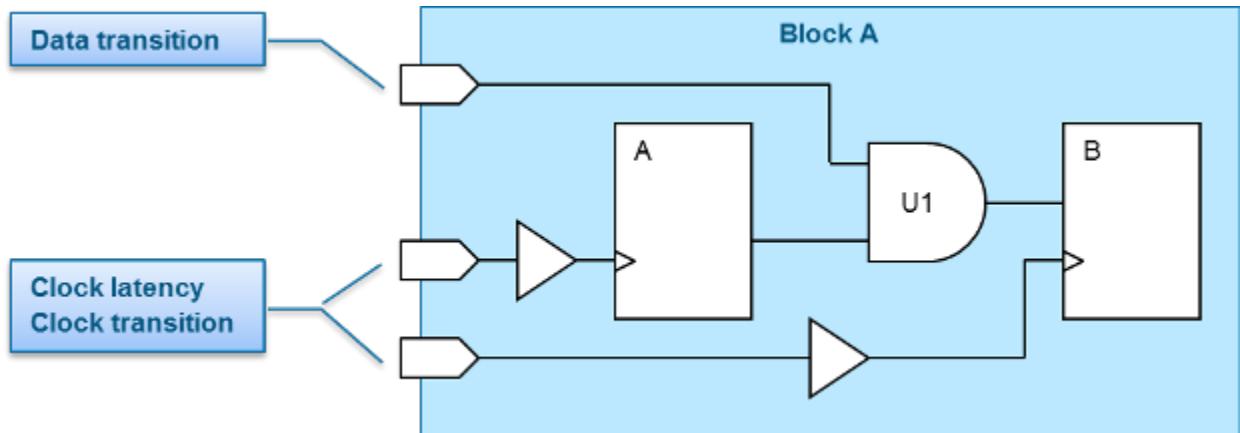
- Clock latency
- Clock transition
- Data transition

For the clock latency condition, the original block budgeted clock latency information can be pessimistic, optimistic, or just estimated. Clock latency information has a large impact on the magnitude of setup and hold violations as well as the number of violations for block internal paths. For setup and hold analysis, a pessimistic latency for early and late arrivals can cause significant numbers of fixing false violations and over-fixing real violations. For optimistic values, violation counts and magnitudes can be smaller and under fixing could occur; therefore, the violation is not truly corrected by ECO.

For the clock and data transition conditions, inaccurate input slew affects the block internal by affecting propagated slew and the setup and hold requirements relative to the clock slew. Although transition time can be more of a secondary effect on timing, it still contributor to the integrity of and accuracy of the block-level timing analysis. Additionally, these conditions are an important factor for DRC violations in the block internals. Inaccurate and pessimistic input transitions can lead to pessimistic propagated transition times for clock and data lines that result in `max_transition` violations.

[Figure 406](#) shows the top context for register-to-register path ECO fixing.

Figure 406 Top Context for Accurate Register-to-Register Path ECO Fixing



In HyperScale, context updates for the block level apply the actual clock and data transition times and clock arrivals seen by the top-level analysis. This addresses inaccuracies in the block-level budgeted constraints for internal block register-to-register paths. Because of this capability, block-level ECO fixing with the `fix_eco_timing`, `fix_eco_drc`, and `fix_eco_power` commands see the accurate timing and therefore can perform prudent fixing.

Fixing Block Boundary Paths

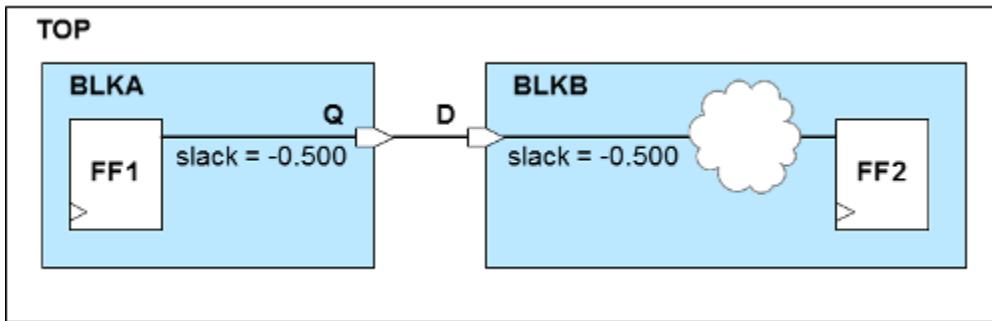
In addition to the clock latency, clock transition, and data transition conditions, HyperScale also provides context updates for many other conditions to accurately fix violations relative to the block boundary paths. In some of these conditions, the level of accuracy in block boundary modeling cannot be achieved through standard Tcl or Synopsys Design Constraint (SDC) — this being clock reconvergence pessimism removal (CRPR) and advanced on-chip variation (AOCV). Some examples of the dominant conditions that contribute to accurate block-level analysis include:

- Input and output delays
- Output loading
- Clock reconvergence pessimism removal (CRPR)
- Advanced on-chip variation (AOCV)

The clock latency, clock transition, and data transition conditions have the same importance in block boundary accuracy, and they are required to achieve ECO closure for timing and DRC. In addition, for the input and output delay conditions, the impact the slack related to input paths ending in a boundary endpoint as well as paths starting in the block and ending at an output. Budgeted constraints typically do not have the level of accuracy; therefore, over-fixing or under-fixing occurs. The output loading condition is also applicable for correct DRC fixing relative to outputs. There is no way to describe the CRPR and AOCV dependencies that exist outside the block. However, the HyperScale context adjustment fully reproduces the effects of these external relationships at the block boundary.

Additionally, when fixing paths at the block boundary top-level timing context is pushed down to each block, each block analysis sees the full timing violation. If desired, you can select one of the blocks based on design knowledge, and attempt to fix the entire violation within that block. After the block has been fixed, subsequent analyses use the updated block timing. [Figure 407](#) shows ECO fixing across multiple blocks.

Figure 407 ECO fixing across multiple blocks



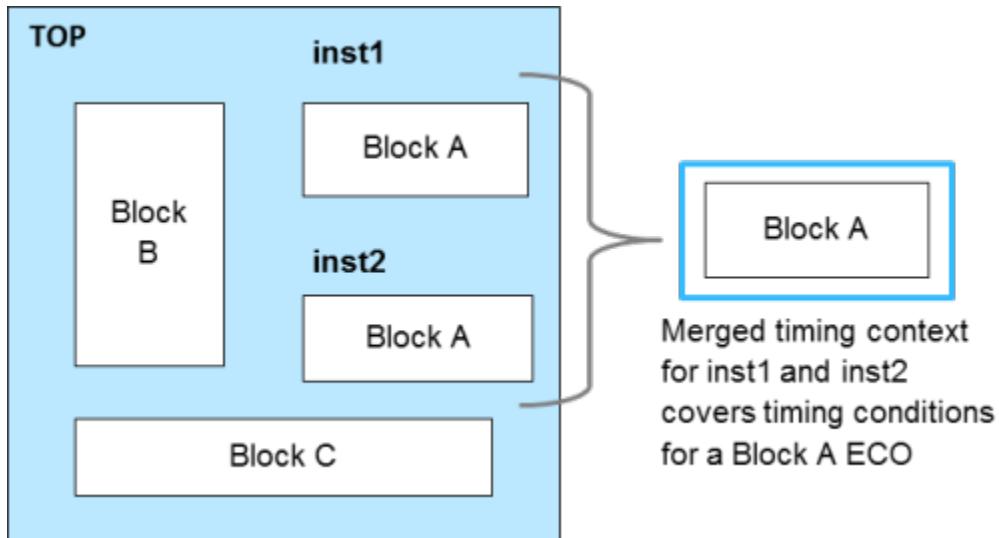
Alternatively, you can fix the full timing violation in both blocks in parallel. In some cases, this can lead to over-fixing of the path. However, this over-fixing can be desirable due to the less predictable nature of cross-block paths, which communicate through top-level routing. Typically, the outputs of the driving block are registered, and there is little opportunity to improve the timing. The full magnitude of the violation needs to be fixed in the receiving block. Additionally, the fixing approach also depends on the design methodology and whether inputs or outputs are registered for respective blocks. Depending on the methodology, the strategy might change on a block-per-block basis.

ECOs for Multiply Instantiated Modules (MIMs)

A block that is used multiple times in a design is called a multiply instantiated module (MIM). Implementation tools such as IC Compiler and IC Compiler II typically generate a single cell for a MIM and place multiple instances of that cell in the chip layout. An important and powerful feature in HyperScale is the ability to create a merged context for MIMs so that a single ECO operation applies to all instances of the block.

[Figure 408](#) shows a circuit example where inst1 and inst2 both use Block A. Within the HyperScale flow and hierarchical configuration, a merged context can be determined that covers the worst case conditions around both instances of Block A. Using this merged context allows a single ECO to be performed, which covers Block A and mirrors implementation. This capability provides productivity benefits over a standard flat ECO flow for which the ECO fixing is performed across multiple instances and changes are then merged for implementation. By being able to perform the ECO with a merged context across all instances, only a single ECO for Block A is needed to be done and implemented in IC Compiler.

Figure 408 ECO fixing across blocks used multiple times



Resolving Block Violations in the ECO Flow

In a HyperScale ECO flow, most of the violation fixing occurs in the block-level ECO. Because of this, it is recommended to take the block-level ECO changes through IC Compiler or IC Compiler II and StarRC and realize the changes before performing the top-level ECO. The block-level ECO can change the timing view seen at the top-level. Therefore, to obtain the most up-to-date and accurate view at the top level, it is a preferred practice to implement the block-level ECOs in IC Compiler and StarRC.

[Figure 409](#) shows script examples from PrimeTime, IC Compiler, and StarRC. It demonstrates performing ECOs at the block-level, implementing them in IC Compiler and StarRC, and then using the updated outputs in HyperScale runs for the block-level and top-level. Additionally, after the top level is rerun with the updated blocks, you can then use report_constraint and determine if the current context provided by the top is still providing a bounding environment for the block. If there are violations, you can choose to rerun the block level with a new updated context to determine any effects for block-level analysis.

Figure 409 Implementation for Block-Level ECO Changes Before Top-Level ECO

PrimeTime ECO Script Example

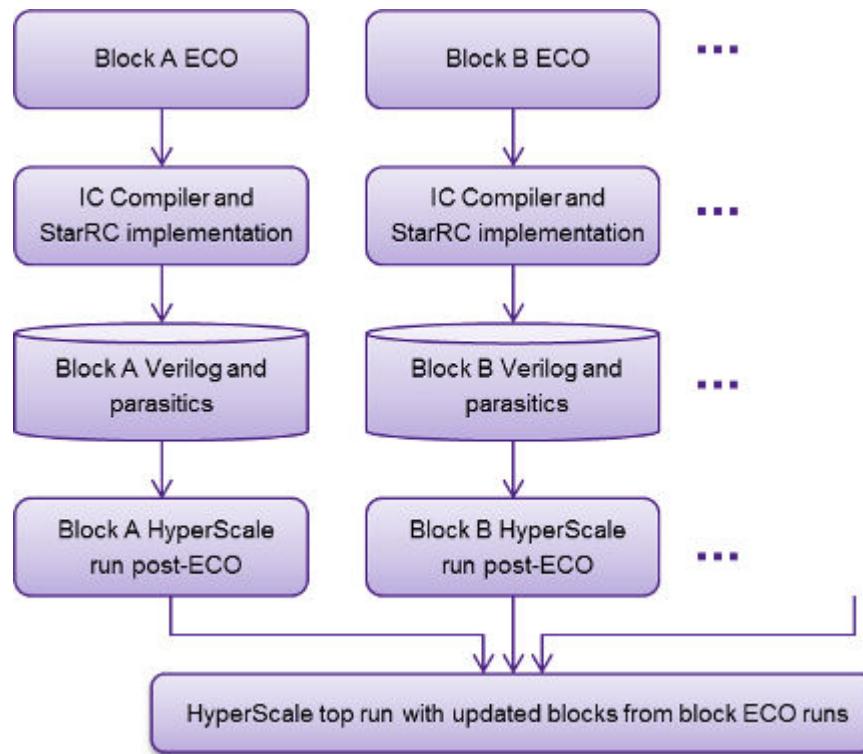
```
...
set_app_var hier_enable_analysis true
set_hier_config -path ...
read_verilog BlockA.v
link
read_parasitics -keep_cap ...
update_timing
fix_eco_timing -type setup
write_changes -format icctcl ...
```

IC Compiler Script Example

```
open_mw_lib Design.mw
open_mw_cel BlockA
...
eco_netlist -by_tcl_file BlockA.tcl
place_eco_cells
route_zrt_eco -reroute \
modified_nets_first_then_others
save_mw_cel -as BlockA_eco
write_verilog BlockA_eco.v
```

StarRC Command File Example

```
BLOCK: BlockA_eco
MILKYWAY_DATABASE: Design.mw
NETLIST_FORMAT: GPD
```



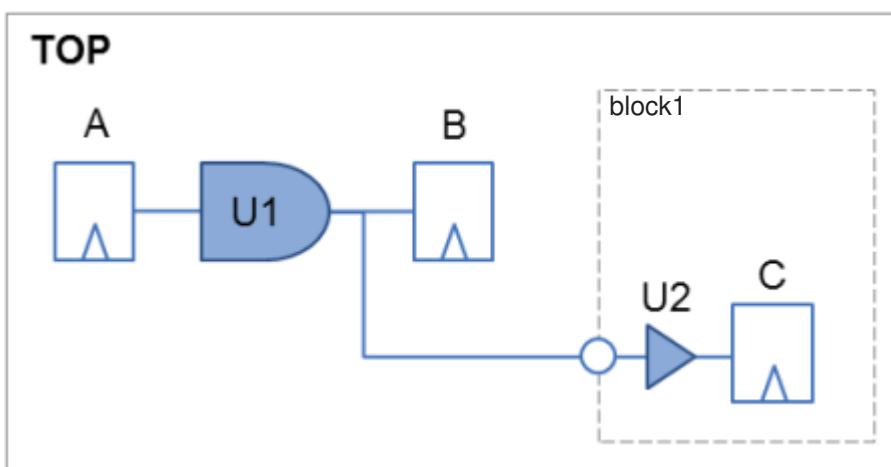
See Also

- [HyperScale ECO Interface to IC Compiler and StarRC](#)

Performing ECOs at the Top Level

Top-level ECOs can fix top-level-only paths or paths that cross block boundaries that are not corrected during the block-level ECOs. When correcting block boundary paths from the top level, both the top-level logic and the block interface path change. Block-level logic beyond the interface paths are considered to have the `dont_touch` attribute. A top-level ECO is restricted to sizing and buffering top-level and block-level interface path logic. Since most of the fixing for block boundary paths occurs at the block level, you need to implement the block-level fixes in IC Compiler and StarRC and provide updated block data to HyperScale so that it reflects the implemented block.

Figure 410 Top-level path ECO fixing



For the design example in [Figure 410](#), HyperScale can perform the following top-level ECO changes:

- Resize the U1 AND gate to correct timing paths between register A and register B and paths between register A and register C.
- Resize the U2 buffer to correct timing paths between register A and register C.

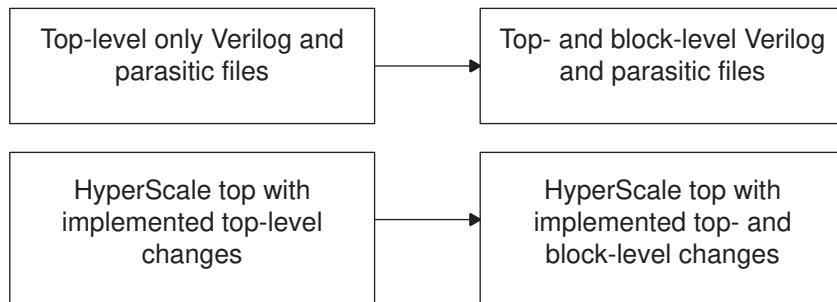
Note:

If the U1 gate is resized at the top-level, the characteristics of that sizing are reflected in the context for block1. Therefore, during the next block-level run of block1, the driving cell of the port for register C is updated to reflect the driving characteristics and input delay of ECO operations performed at the top level. This ensures that the block-level context is up-to-date with the interface of the top level.

In the top-level HyperScale flow, the top- and block-level Verilog and parasitics are required to be updated for the next top-level HyperScale run. The block-level ECOs are

performed and implemented in IC Compiler and StarRC first, and then these updates are reflected before a top-level ECO. [Figure 411](#) outlines the top-level ECO and the required outputs from IC Compiler and StarRC for the next top-level HyperScale run.

Figure 411 Top-level ECO fixing flow



Running ECOs in Parallel for Runtime and Capacity

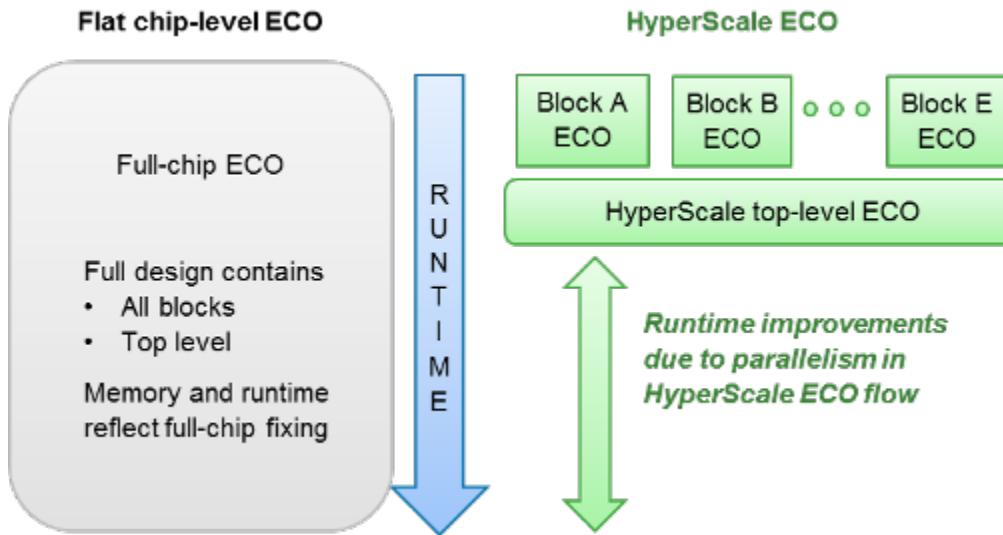
HyperScale enables ECOs to be run at the block-level and top-level accurately with the runtime and capacity benefits of the HyperScale flow. This enables faster turnaround time for the complete the ECO process.

With HyperScale, the block-level ECOs can be done with a “divide and conquer” technique allowing them to be run in parallel. Although running the blocks serially is supported, to take advantage of the parallelism native to HyperScale, it is recommended that the block ECOs be performed concurrently. Additionally, as discussed in the previous section, ECOs for MIMs can be done all at once. A merged context can be used for these types of blocks to perform a single ECO and implemented directly by IC Compiler.

Also, since HyperScale provides accurate contexts, the violations are the same as what would be seen in a flat chip-level PrimeTime run.

[Figure 412](#) shows that the HyperScale block-level ECOs can be performed in parallel with one another, providing improved overall turnaround time. Essentially, the total runtime for the HyperScale ECO process is the sum of the runtime of the longest block run and the runtime of the top-level ECO. For most hierarchical designs, the top-level ECO is relatively short since most of the ECO fixing occurs at the block level. Regarding capacity, the HyperScale top-level run consumes much less memory than the flat ECO, which is a key benefit of using the HyperScale infrastructure.

Figure 412 Parallel block ECO fixing provides runtime benefits

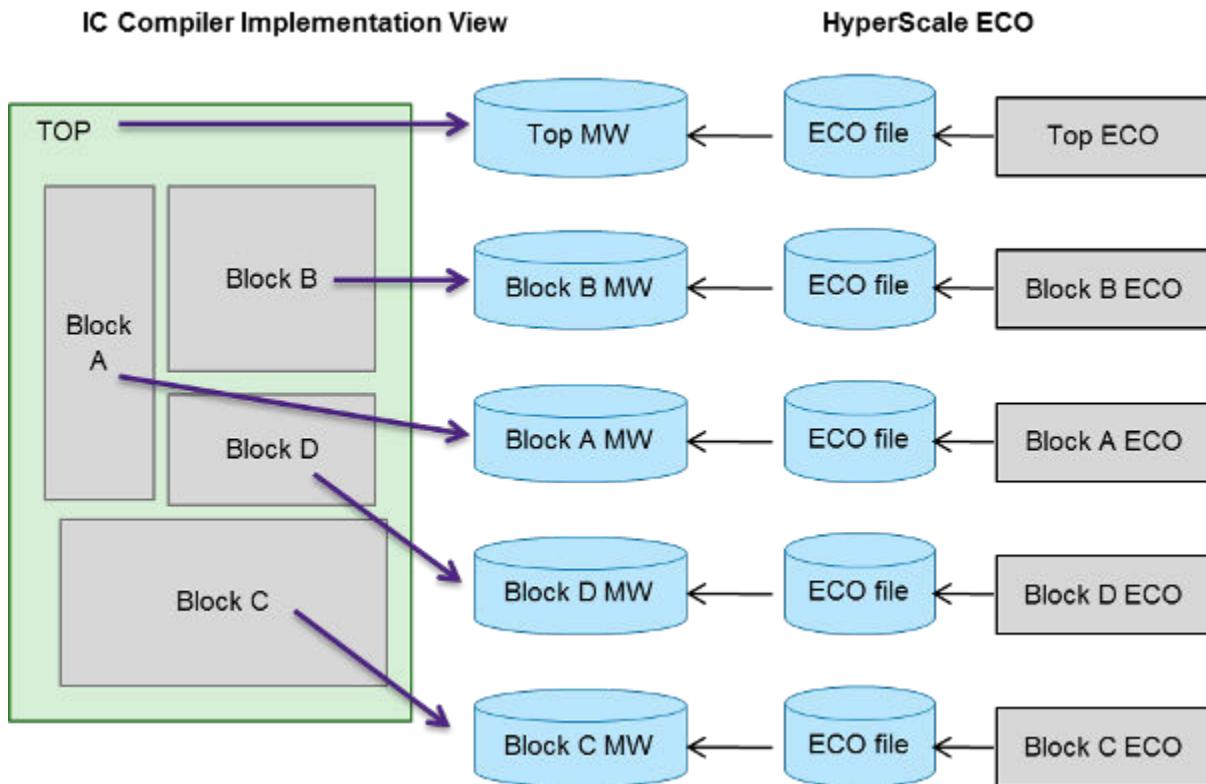


HyperScale ECO Interface to IC Compiler and StarRC

In a standard flat ECO flow, the ECO is performed on the flat PrimeTime design and then the suggested changes from the `write_changes` command are manipulated to map to the hierarchical IC Compiler or IC Compiler II database. Another disadvantage of performing a flat ECO is that the ECO changes need to be restructured to apply them to block-level and top-level physical implementations in IC Compiler or IC Compiler II. For a HyperScale ECO, the ECOs performed on the block and top levels mirror partitioning in the physical implementation tool; therefore, they can be sourced directly by the IC Compiler or IC Compiler II implementation.

Figure 413 shows the one-to-one mapping between IC Compiler implementation and HyperScale. Therefore, within the HyperScale ECO, the same design architecture implemented in IC Compiler can be reused.

Figure 413 Exact Mapping From HyperScale to Physical Implementation



In Figure 413, the IC Compiler implementation is done hierarchically, and the blocks and top are implemented separately, each having its own Milkyway cell view. HyperScale uses the same block and top configuration. The ECO performed in HyperScale can be directly implemented by IC Compiler and StarRC.

The HyperScale block ECO and implementation should be performed before the top-level ECO and implementation. This sequence flow ensures that the top-level ECO step has the most up-to-date block-level timing details based on the block-level ECO data. As shown in Figure 414, for each block-level and top-level, the ECO guidance file is passed by HyperScale to IC Compiler for ECO placement and routing. This result is then used by StarRC to extract the parasitics again. Figure 414 shows the links that are used to implement the suggest ECO changes provided by HyperScale to fix violations.

Figure 414 Flow links from HyperScale ECO to IC Compiler and StarRC

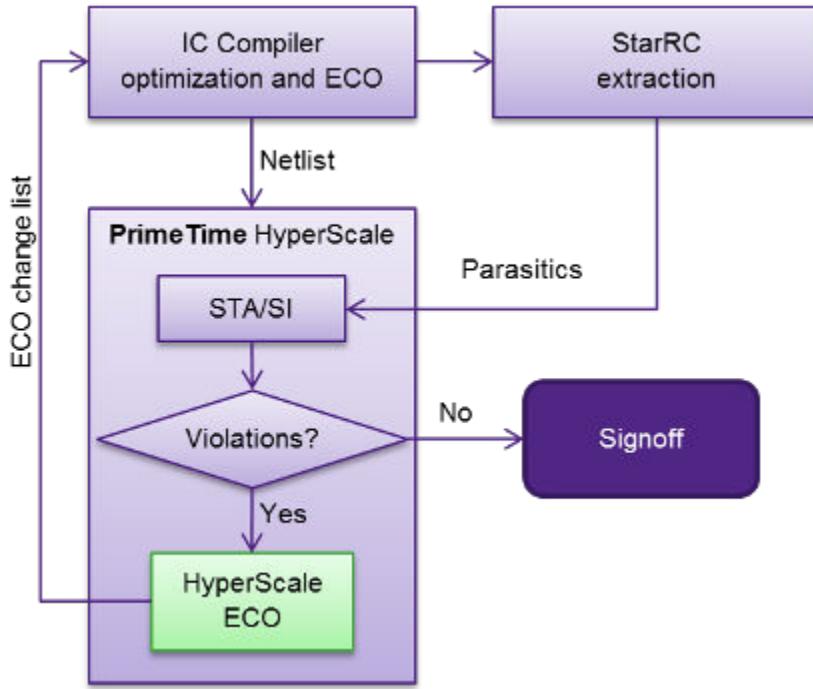
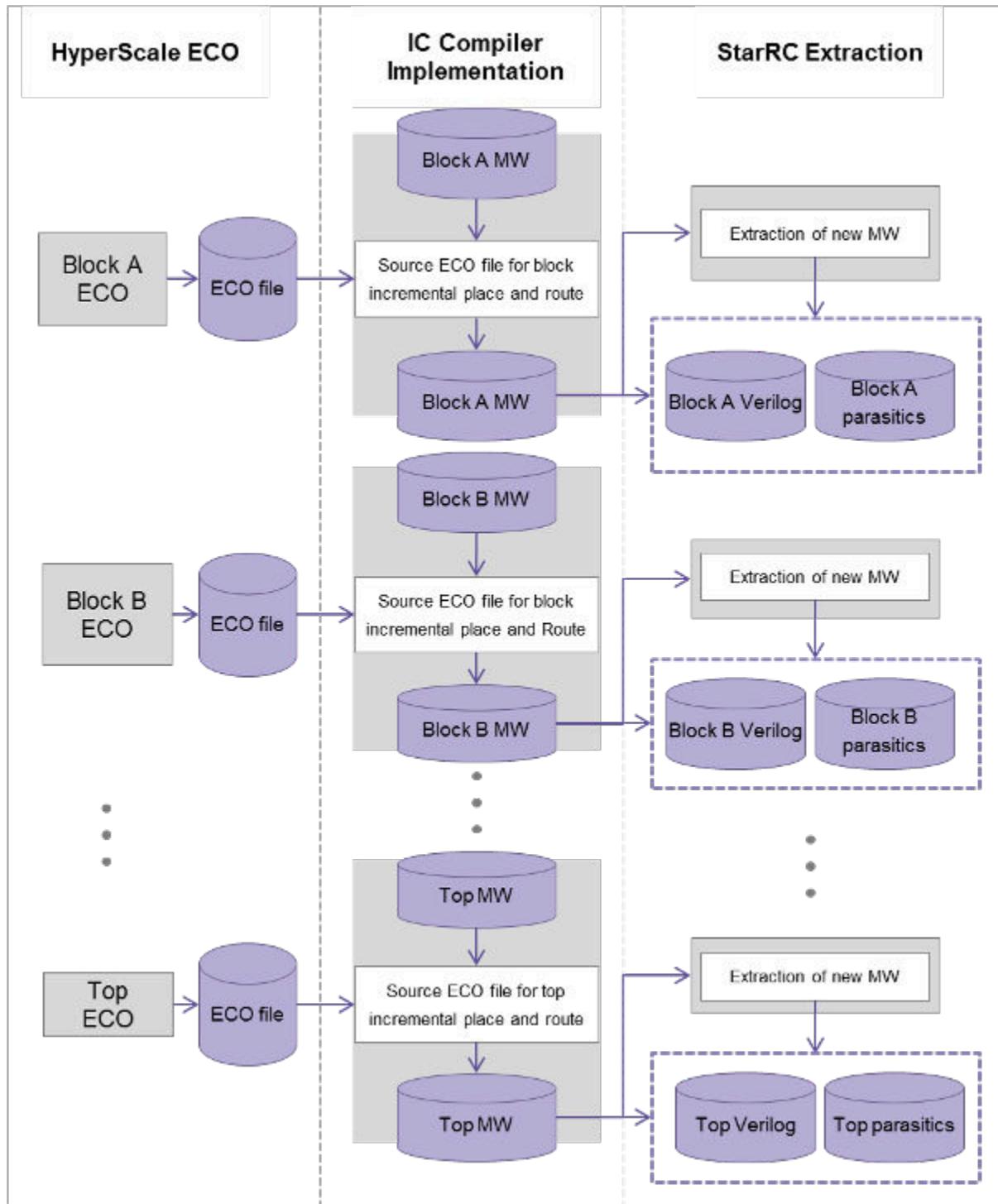


Figure 415 shows the data flow in detail. For each block-level and top-level, the ECO file is provided to IC Compiler. IC Compiler reads the respective Milkyway cell view, sources the ECO file generated by HyperScale, performs an incremental placement and routing, and saves the new Milkyway cell. The output provided by IC Compiler for the next HyperScale run is a new Verilog netlist. StarRC uses the new Milkyway cell saved after the incremental placement and routing by IC Compiler to generate a new parasitics file. This file is provided to HyperScale. The new Verilog and parasitics files are used as inputs into HyperScale for the next block-level and top-level runs.

Figure 415 Data flow from HyperScale to IC Compiler and StarRC then back to HyperScale



Timing Analysis in HyperScale

You typically provide a post-route design for HyperScale analysis. The nets in the design have parasitic data from the StarRC extraction tool. For the boundary timing constraints, instead of using virtual clocks, HyperScale requires the reference clock to be defined in input delay and output delay. This ensures that the timing analysis considers them during crosstalk analysis as victim and aggressor, respectively. By default, HyperScale uses the following design setup variables:

```
set auto_wire_load_selection false
set_wire_load_mode top
set_timing_input_port_default_clock false
```

These settings happen automatically during the `link_design` command, if the `hier_enable_analysis` variable is set to `true`. The tool indicates this change in the flow with the HS-012 information message:

```
Information: value 'false' is preset for variable
  'auto_wire_load_selection' for HyperScale flow enabled with
  'hier_enable_analysis'. (HS-012)
Information: value 'top' is preset for command 'set_wire_load_mode'
  for HyperScale flow enabled with 'hier_enable_analysis'. (HS-012)
false
Information: value 'false' is preset for variable
  'timing_input_port_default_clock' for HyperScale flow enabled
  with 'hier_enable_analysis'. (HS-012)
```

Top-level timing analysis is done respective session for HyperScale subblocks. Reporting works same way as reporting in standard PrimeTime and PrimeTime SI. There are a few additional feature and options added to the reporting commands.

HyperScale Annotations in Reporting

In HyperScale block-level analysis, some commands indicate the use of context data through annotations.

In the output of the `report_timing` command, the at symbol (@) indicates that a stage has timing information that is annotated by HyperScale context update. The timing information includes the external input and output delays that arrive on port, annotated transition time on leaf pins. The following example shows HyperScale annotated delay on the timing path from input port to the register in the block:

```
Startpoint: crami[391] (input port clocked by pci_clk)
Endpoint: cmem0_dtags1_1_x0_memarr_reg_4_10_
(rising edge-triggered flip-flop clocked by pci_clk)
Path Group: pci_clk
Path Type: max
```

Point	Incr	Path
-------	------	------

clock pci_clk (rise edge)	0.0000	0.0000
clock network delay (propagated)	0.0000	0.0000
input external delay	34.2691 @	34.2691 f
crami[391] (in)	0.0000 @	34.2691 f
icc_place1039/INP (INVX0_HVT)	0.0000 @	34.2691 f
icc_place1039/ZN (INVX0_HVT)	3.0902 &	37.3593 r
icc_place1038/INP (INVX4_HVT)	0.1158 &	37.4751 r
icc_place1038/ZN (INVX4_HVT)	1.8121 &	39.2872 f
cmem0_dtags1_1_x0_memarr_reg_4__10_/D (SDFFX1_HVT)	0.0239 &	39.3111 f
data arrival time		39.3111
clock pci_clk (rise edge)	36.0000	36.0000
clock network delay (propagated)	4.9610	40.9610
clock uncertainty	-0.2000	40.7610
cmem0_dtags1_1_x0_memarr_reg_4__10_/CLK (SDFFX1_HVT)		40.7610 r
library setup time	-1.3354	39.4257
data required time		39.4257

data required time		39.4257
data arrival time		-39.3111

slack (MET)		0.1145

In the output of the `report_port -verbose` command, the “H” annotation indicates that the value comes from the HyperScale context data:

```
pt_shell> report_port IN -verbose
Attributes:
  I - ideal network
  H - HyperScale context override

          Pin Cap           Wire Cap
Port      Dir      min/max      min/max   Attributes
-----
IN        in       0.0000/0.0000  0.0000/0.0000  H
...
          Input Delay
          Min            Max           Related Related
          Rise   Fall    Rise   Fall   Clock   Pin   Attrs
Input Port
-----
```

Input Port	Rise	Fall	Rise	Fall	Clock	Pin	Attrs
IN	20.00	20.00	20.00	20.00	BCLK2	--	
	30.00	30.00	30.00	30.00	BCLK3	--	
	3.57	4.12	3.57	4.12	--	--	H
	1.57	2.12	1.57	2.12	BCLK1	--	H
	2.57	3.12	2.57	3.12	--	--	H

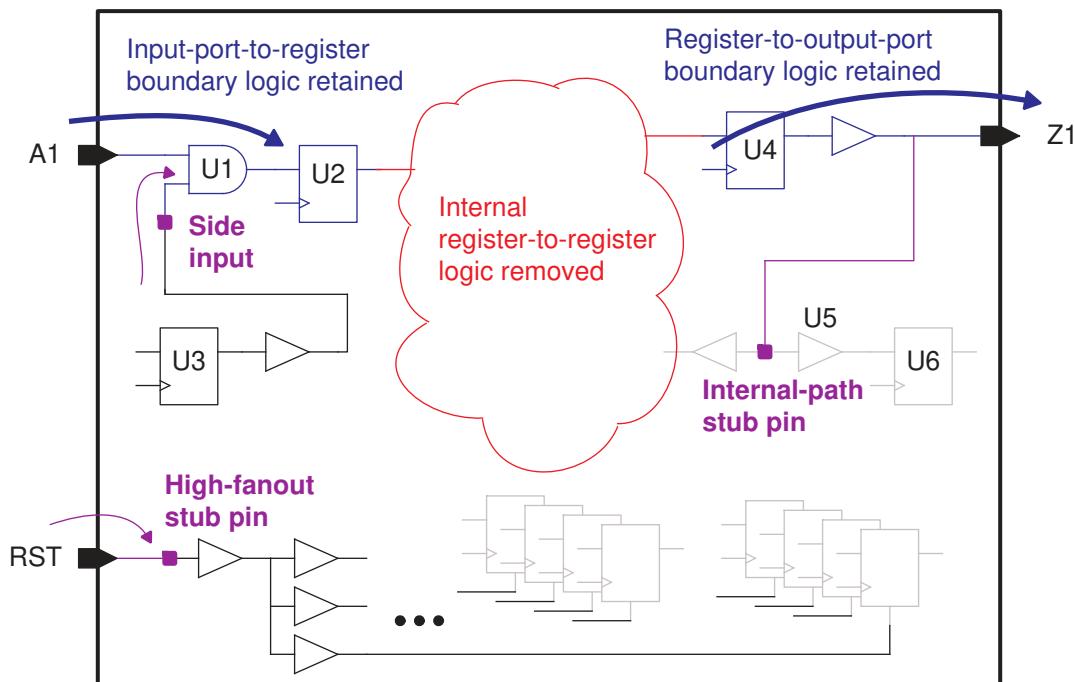
Block Models

After you perform block-level HyperScale analysis, the `write_hier_data` command writes out a block model that can be used in top-level HyperScale timing analysis. The model

contains the interface logic relevant to top-level timing analysis and excludes the rest of the logic.

The following diagram shows the logic contained in a HyperScale block model. The logic shown in blue is the input-to-register and register-to-output logic, which is retained because it is needed to analyze the timing of paths that enter the block inputs and leave the block outputs. The red cloud represents the internal register-to-register logic, which is removed because it does not affect top-level analysis.

Figure 416 HyperScale Block Model



The portions of logic shown in gray are removed and replaced by a pin that represents the removed logic. There are three types of such pins:

- **Side input** – This is an input pin of a cell (U1/B) in a path from a block input port (A1) to a block interface register (U2), but not in the path itself; it is in a path that starts from an internal register. The worst-timing path to the interface register (U2) could originate either from the block input or the side input.
- **Internal-path stub pin** – A cell input pin (U5/A) in the fanout of a path from an interface register (U2) to a block output port (Z1), but not in that path itself; the pin is in

a path that ends at an internal register. The stub pin represents the load of the removed cells on the preserved net.

- **High-fanout stub pin** – A pin that represents the worst-case timing constraints at a port or pin with a large transitive fanout. The inserted pin allows the removal of all the logic in the large fanout.

Some of the pins inserted into the model have their own timing constraints to represent the removed logic. Top-level timing reports show the same results as block-level analysis for paths that pass through these pins, even though the related logic has been removed.

HyperScale block-level analysis makes decisions about how much interface logic to retain based on the block interface topology and timing characteristics. The logic needed to calculate worst-case timing is retained so that it is completely visible for top-level analysis, whereas the unimportant logic is discarded and made invisible from the top level. The decision to retain or discard logic is based on maintaining full timing accuracy while reducing runtime and memory.

Usually it is best to accept the logic removal decisions made by the tool. However, you can modify the default behavior by using the following commands and variables:

- `set_port_abstraction` command – Explicitly specifies a list of ports (along with their respective fanout) to be either retained or removed in the HyperScale block model, overriding the default abstraction.
- `set_pin_abstraction` command – Explicitly specifies a list of pins to be retained in the HyperScale block model, overriding the default abstraction that would otherwise remove them.

For [Side Inputs](#), use the following variables to control abstraction and reporting:

- `hier_enable_detailed_side_input_path_timing` – When set to `true`, captures the full timing details of the removed logic leading up to side input pins, instead of capturing only the effects on interface paths such as worst slew and crosstalk arrival windows.
- `timing_separate_hier_side_inputs` variable – When set to `true`, puts the paths that start from side inputs into their own path group, `**HyperScale_side_input**`, instead of reporting them with all the other paths clocked by the same clock.

For [Stub Pins](#), use the following variables to control abstraction and reporting:

- `hier_keep_required_time_mode` variable – Specifies the types of stub pins that retain their required-time path constraints in the block model: none, internal-path stub pins only, high-fanout stub pins only (the default), or all stub pins.
- `hier_enable_detailed_stub_path_timing` – When set to `true`, captures the full timing details of the logic removed from paths extending beyond internal-path stub pins, instead of using more compact timing data.
- `timing_report_hier_stub_pin_paths` variable – When set to `false`, suppresses reporting of paths that end at stub pins, instead of reporting them in their own path group, `**HyperScale_stub_default**`.

See Also

- [Port Abstraction](#)
- [Side Inputs](#)
- [Stub Pins](#)

Port Abstraction

In block-level HyperScale analysis, the `write_hier_data` command writes out a block model containing only the information needed for top-level analysis. A block port is abstracted (removed) when the entire fanout or fanin logic of the port is not required for accurate top-level analysis. This is the case for clock ports, other high-fanout input ports (for example, a reset input), ports with a fixed logic value enforced by case analysis, and ports with no timing constraints applied.

When a block port is partially or completely abstracted, some or all of the fanout or fanin logic cone becomes unavailable in top-level analysis. The tool uses advanced heuristics to determine the amount of logic to remove for accuracy, runtime, and memory usage.

To report port abstraction done by the tool, use the following command:

```
pt_shell> report_hier_analysis -port_abstraction
```

The report shows the ports that have been abstracted and the reason for the abstraction.

You can modify the default port abstraction by using the `set_port_abstraction` command. The command explicitly specifies a list of ports to be either retained or removed (along with their respective fanout or fanin) in the block model. For example,

```
pt_shell> set_port_abstraction -keep [get_ports RST]
```

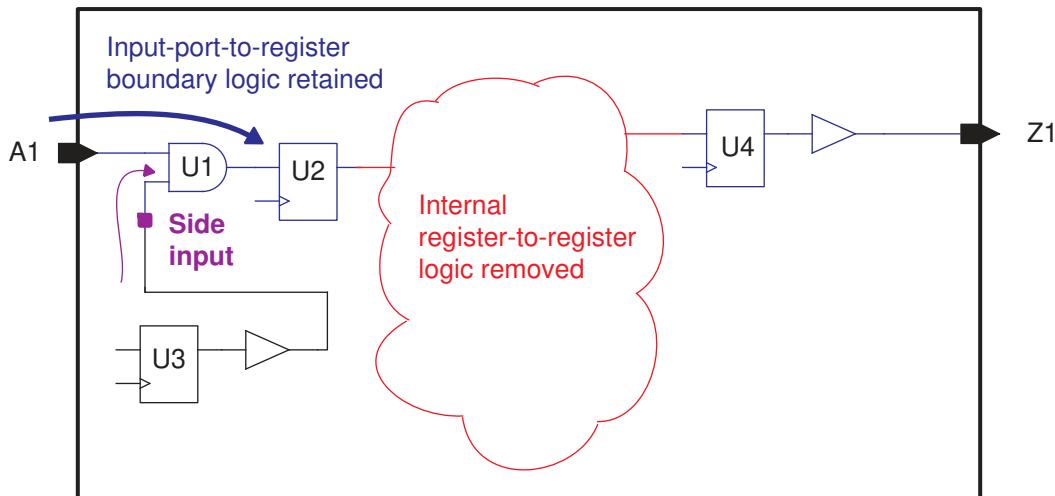
You can also use the `set_port_abstraction` command to report the ports that have been modified and clear previous port abstraction settings.

Side Inputs

In block-level HyperScale analysis, the `write_hier_data` command writes out a block model containing only the information needed for top-level analysis. It retains the logic from an input port to a register and discards the downstream register-to-register logic.

The following diagram shows an example of a side input. This is an input pin of a cell in a path from a block input port to a block interface register, but the pin is not itself in the path. The logic driving the side input comes from an internal register. The entire fanin logic leading to the side input pin is excluded from the model, and the timing information from the removed logic is annotated on the pin to keep its effects on the interface path.

Figure 417 Side Input in Block Model



A side input exists only in the block model used in top-level analysis. For block-level analysis, all the logic is present, so there is no need for side input models. To identify side input pins during top-level analysis, you can query the `is_side_input` pin attribute.

By default, a side input pin captures only the effects on interface paths such as worst slew and crosstalk arrival windows. To capture the full timing details of the removed logic leading up to side input pins, do the following:

```
pt_shell> set_app_var hier_enable_detailed_side_input_path_timing true
```

In the block model, the side input serves as an “internal startpoint” annotated with an “external delay” determined by the removed logic. A timing path exists from the side input pin to register U_2 in the foregoing diagram. In top-level analysis, considering this timing path allows the true worst path leading up to register U_2 to be reported. The true worst path could originate from either the side input or from the top-level logic driving the block input A_1 .

By default, a path that starts from a side input is reported in the same path group as all other paths clocked by the same clock (the clock driving the removed register U3 in this example). If you want side-input paths to be reported in a separate path group, do the following:

```
pt_shell> set_app_var timing_separate_hier_side_inputs true
```

When set the variable is set to true, the `report_timing` command reports the side-input paths in their own path group, named `**HyperScale_side_input**`.

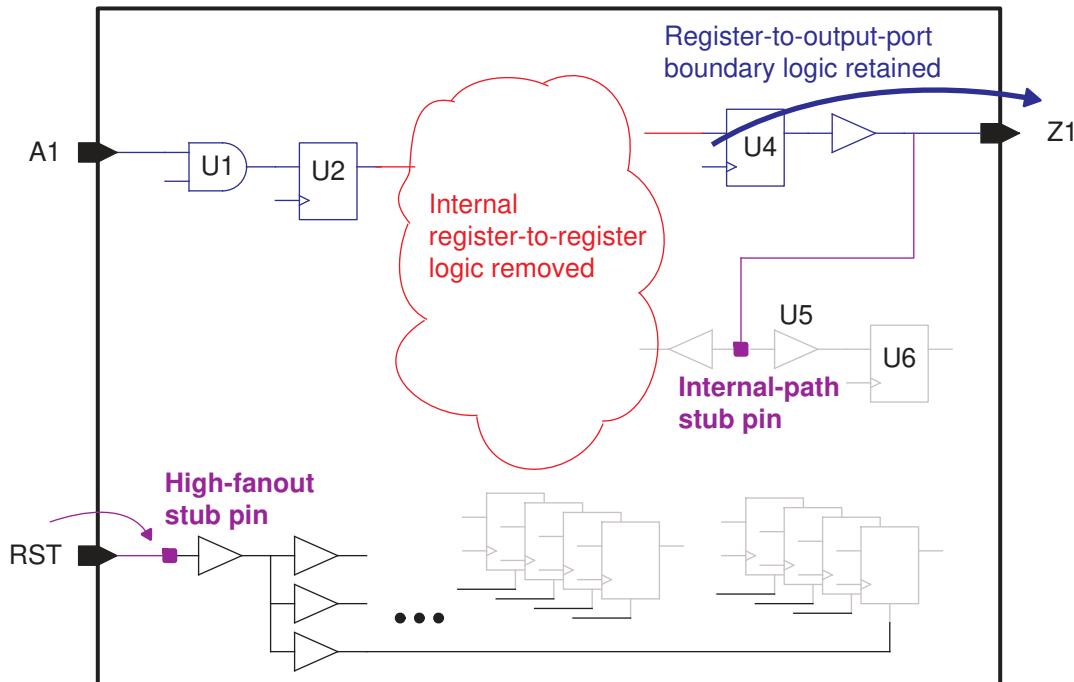
Stub Pins

In block-level HyperScale analysis, the `write_hier_data` command writes out a block model containing only the information needed for top-level analysis. In some cases, it inserts a stub pin to represent the capacitive load or timing of removed logic. There are two types of stub pins created in a block model:

- **Internal-path stub pin** – A cell input pin in the fanout of a path from an interface register to a block output port, but not in that path itself; the pin is in a path that leads to an internal register. The stub pin represents the capacitive load of the removed cells on the preserved net.
- **High-fanout stub pin** – A pin that represents the worst-case required-time constraints at a port or pin with a large transitive fanout, allowing removal of the logic in the large fanout.

The following diagram shows an example of each type of stub pin.

Figure 418 HyperScale Block Model



A stub pin exists only in the block model used in top-level analysis. For block-level analysis, all the logic is present, so there is no need for stub pin models. To identify stub pins during top-level analysis, you can query the `is_stub_pin` attribute.

By default, top-level analysis reports the timing paths that end at high-fanout stub pins, but not internal-path stub pins. The latter paths are seldom affected by block context (only the load on the related output port can affect the path).

To change the default behavior, set the following variables:

- `hier_keep_required_time_mode` – Specifies the types of stub pins that retain their required-time path constraints in the block model: none, internal-path stub pins only, high-fanout stub pins only (the default), or all stub pins.
- `hier_enable_detailed_stub_path_timing` – When set to `true`, captures the full timing details of the logic removed from paths extending beyond internal-path stub pins, instead of using more compact timing data.
- `timing_report_hier_stub_pin_paths` – When set to `false`, suppresses reporting of paths that end at stub pins, instead of reporting them in their own path group, `**HyperScale_stub_default**`.

- `hier_model_enable_unconstrained_port_abstraction` – When set to `false`, prevents abstraction of unconstrained ports.
- `hier_model_port_abstraction_ignore_register_count`
`hier_model_port_abstraction_ignore_register_percentage`

These variables control the thresholds that determine whether a high-fanout net is abstracted. By default, a high-fanout net is abstracted when it fans out to 256 or more registers, and these registers make up at least 1.0 percent of all registers.

Transparent Latch Levels

In latched-based designs, by default HyperScale includes two levels of transparent latches (plus a final capturing latch) in the block model.

To change how many transparent latch levels are kept in the model, set the following variable:

```
pt_shell> set_app_var hier_block_interface_model_latch_level depth
```

The *depth* value is the number of transparent latch levels to include in the model (not counting the final capturing latch). Values from 0 to 5 are allowed.

Block Boundary Modeling

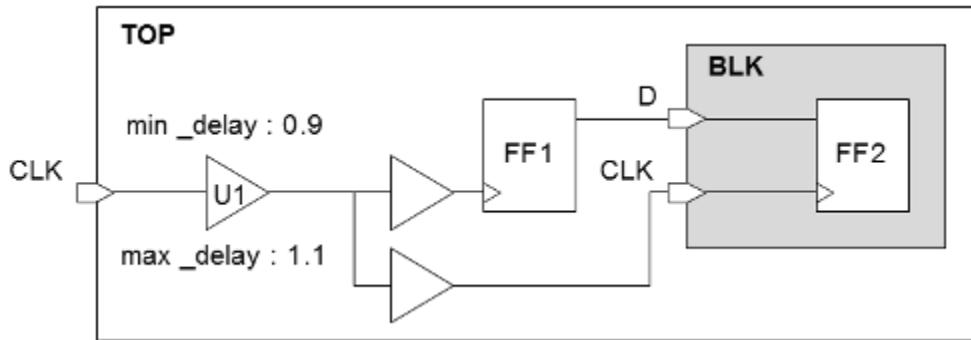
For accurate block-level timing, the tool considers the following boundary information:

- [Clock Reconvergence Pessimism Removal \(CRPR\) Modeling](#)
- [Path-Specific Exception Modeling](#)
- [External Advanced On-Chip Variation \(AOCV\) Effects](#)
- [Clock Latency Modeling](#)

Clock Reconvergence Pessimism Removal (CRPR) Modeling

Consider the design shown in [Figure 419](#).

Figure 419 CRPR advanced boundary modeling

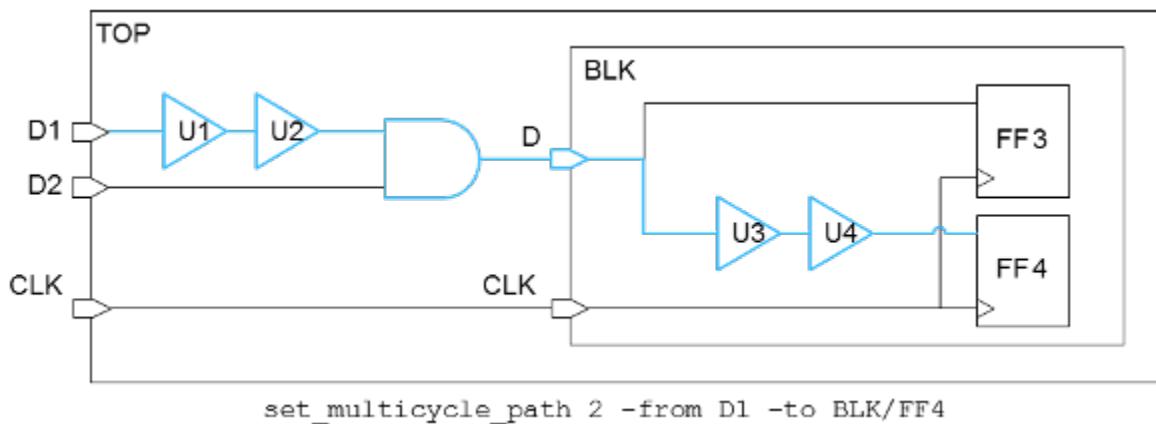


In the top-level analysis, a CRPR credit of 0.2 is applied to the timing path from FF1 to FF2 due to the min/max delay variation in clock buffer U1. In the block-level analysis, while the clock and data arrival times can be described using the `set_input_delay` and `set_clock_latency` commands, there is no way to describe the CRPR relationship between FF2 and the launching flip-flop FF1, which exists outside the block. However, the HyperScale context adjustment fully reproduces the effects of these external CRPR relationships at the block boundary. This CRPR context adjustment includes the static and dynamic components of CRP that can be introduced by on-chip variation, delta delay effects, and static and dynamic IR drop effects.

Path-Specific Exception Modeling

Another category of advanced boundary modeling is path-specific exception modeling. Consider the design shown in Figure 420.

Figure 420 Path-specific exception modeling



In this example, a multicycle path is applied to the long combinational path from input port D1 to flip-flop FF4. This multicycle exception references both top-level and block-level design objects, causing it to be a split exception. In the top-level analysis, this

exception applies to a very specific path. However, this split exception cannot be precisely described in the block-level analysis because the top-level object cannot be referenced from a block-level context. Modeling such an exception would require that the longer external logic delay at input port D is used for the path to flip-flop FF3, but the shorter logic delay is used for the path to flip-flop FF4, and this cannot be modeled with any constraint command. However, the context adjustment in the HyperScale flow fully reproduces the delay behavior of the external logic such that the timing from the same input port to each endpoint reproduces the correct path-specific input delays:

Startpoint: D (input port clocked by CLK)
Endpoint: **FF3** (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)	0.00	0.00
input external delay	3.00	3.00 r
D (in)	0.00 *	3.00 r
FF3/D (DFF)	0.00 *	3.00 r
data arrival time		3.00

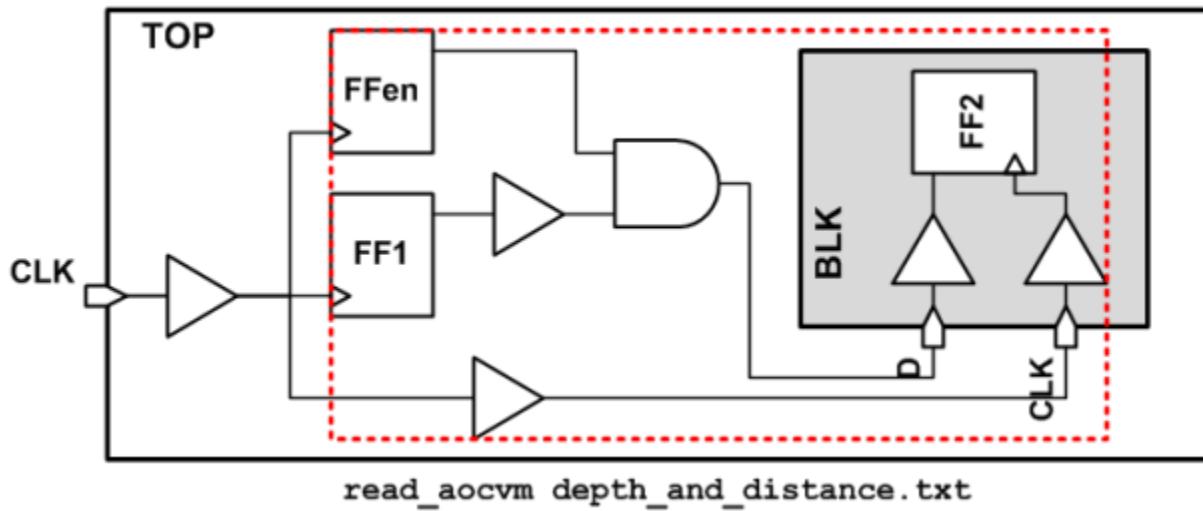
Startpoint: D (input port clocked by CLK)
Endpoint: **FF4** (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point	Incr	Path
clock CLK (rise edge)	0.00	0.00
clock network delay (propagated)	0.00	0.00
input external delay	1.00	1.00 r
D (in)	0.00 *	1.00 r
U3/Y (BUF)	1.00 H	2.00 r
U4/Y (BUF)	1.00	3.00 r
FF4/D (DFF)	0.00	3.00 r
data arrival time		3.00

External Advanced On-Chip Variation (AOCV) Effects

Figure 421 shows an AOCV analysis is performed using two-dimensional depth and distance derating tables. In a standard analysis, the depth and distance characteristics of the external environment cannot be described for the block-level analysis of BLK. However, HyperScale technology can model the depth-and-distance characteristics of the external environment so that the aggregate depths and distances of the combined external and internal path can be computed. The distance modeling even accounts for block rotations where the external environment exists using a transformed coordinate system.

Figure 421 AOCV modeling



These are only three examples of situations where the HyperScale boundary adjustment can replicate the external environment of a block-level analysis with an accuracy that cannot be achieved through Tcl constraint commands. HyperScale technology provides algorithmic analysis continuity across the block boundary that enables the block to be analyzed as it exists in its upper-level context.

Clock Latency Modeling

In both HyperScale and context characterization flows, when you use advanced on-chip variation (AOCV) analysis, you can choose to generate propagated clock latency modeling information using either graph-based or path-based analysis. The default is graph-based analysis, which is accurate for block-level graph-based analysis and conservative for block-level path-based analysis. For higher accuracy using path-based analysis, set the `hier_enable_pba_clock_latency_context` variable to true before you generate the HyperScale block or perform context characterization.

Model-Context Mismatch Handling

The tool handles mismatching between models and context as described in these topics:

- [Context Override](#)
- [Clock Mapping for Side-Input and Stub Pins](#)

Context Override

The tool provides the following HyperScale context override features:

- Detects netlist changes on port nets
- Applies a best-effort context override for each netlist difference found
- Applies context data without a clock if the clock cannot be resolved

To get information on both successful and failed overrides, use the following command:

```
pt_shell> report_context -list_context_override object_list
```

The object list specifies the list of cells, ports, or pins to query; or [current_design] to query all loaded context information.

To get information about context issues for specific objects, query these attributes:

- `is_netlist_dont_override` – true for an anchor leaf pin change on a port net.
- `is_clock_dont_override` – true if at least one context refers to an unmapped clock.

For example,

```
pt_shell> update_timing -full
Information: auto-loading HyperScale timing context data...
Error: Ignore context data at port 'in1' because expected pin 'u1/A' is
      not on the same net . (HS-036)
Error: Ignore context data at port 'in2' because expected pin 'u2/A' is
      not on the same net . (HS-036)
Error: Ignore context data at port 'clk' because expected pin 'cbufal/A'
      is not on the same net . (HS-036)

pt_shell> set bad_ports \
  [get_ports * -filter "is_netlist_dont_override==true"]
in1 in2 clk

pt_shell> update_timing -full
Error: There are unresolved clocks . (HS-004)
Information: auto-loading HyperScale timing context data...
Error: Ignore source latency context for clock 'SYSCLK' at 'clk' because
      no mapped clock found . (HS-036)
Warning: Input delays set without the -clock option will not create
         constrained paths to clocked registers. (UITE-504)
Error: Missing launch clock for arrival context at port 'in1' because no
      clock is mapped with 'SYSCLK'. . (HS-036)
Warning: Input delays set without the -clock option will not create
         constrained paths to clocked registers. (UITE-504)

pt_shell> set bad_ports \
  [get_ports * -filter " is_clock_dont_override==true"]
in1 clk
```

Clock Mapping for Side-Input and Stub Pins

When annotating the arrival and required time on side-input or stub pins, the tool performs the following checks:

- Detects clock mapping errors on the side-input or stub pins
- Issues a message if the launch or capture clock on the side-input or stub pin cannot be resolved
- Skips arrival or required time annotation if clock mapping error detected

A stub pin is a single-pin model that represents a high-fanout or high-fanin port, modeled with only the most critical paths through each output pin connected to the pruned ports.

To detect clock mapping issues, query the `has_model_mismatch_clock` attribute on side-input or stub pins. The attribute is `true` if at least one model arrival or required time refers to unmapped clock. For example,

```
pt_shell> update_timing -full
Error: There are unresolved clocks in block blk. (HS-004)
Error: Missing launch clock for arrival time annotated on pin 'blk/u7/B'
because no clock is mapped with 'CLK2' . (HS-037)

pt_shell> get_attribute [get_pins blk/u7/B] has_model_mismatch_clock
true
```

Controlling the Block Boundary Adjustment

There might be situations where a top-level analysis has previously been run and revised block context information is available; however, it is not desirable to perform the context adjustment in the block-level analysis. One reason could be that the top-level layout has serious timing or layout issues that must be fixed, and it is better to stay with the budgeted constraints until the problems are resolved.

To ignore context data in a HyperScale block-level run, you can use the `set_dont_override` command:

```
set_dont_override
[-include include_types]
object_list
```

`object_list` can be a collection of specific ports, or it can be the `[current_design]` (which ignores the context data for all ports).

By default, all types of context data are ignored. To selectively ignore only certain types of context data, use the `-include` option. [Table 70](#) shows the supported `-include` keyword values.

Table 70 Keywords for the `set_dont_override -include` Option

-include	Data from top-level context to be ignored
<code>arc_delay</code>	Net delay from input port to leaf load pins Net delay from leaf driver pins to output ports
<code>case_value</code>	Case analysis at input ports
<code>pin_transition</code>	Context transitions at input ports

For example,

```
# do not apply case analysis constraints from the top-level context
set_dont_override $my_ports -include {case_value}
```

Note:

The `-include` option specifies the types of data to be *ignored* from the context data (included in the “don’t-override” list), not the types of data to be included when applying the context data.

To determine whether `set_dont_override` has been applied to a particular port, query the `is_user_dont_override` attribute.

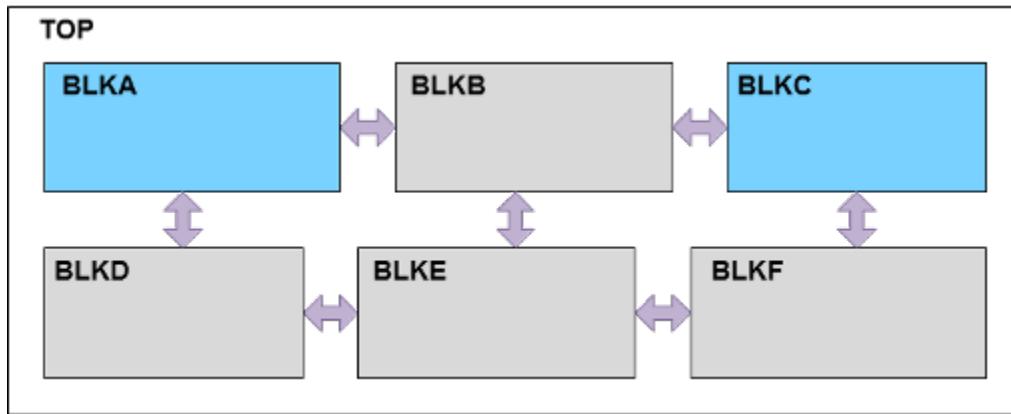
To remove a `set_dont_override` specification, use the `remove_dont_override` command:

```
remove_dont_override [get_ports object_list]
```

If you use the `remove_dont_override` command, the port uses the updated context instead of the original boundary constraints.

As an example, consider a case where other blocks at the top level are not ready, and they exist as black boxes with no timing. [Figure 422](#) shows a HyperScale block-level analysis is being performed for BLKB. Normally, a context adjustment would be performed for BLKB using the available context information from the top-level analysis. However, in this case, BLKA and BLKC are not ready for top-level analysis, so black box models with no timing are being used to satisfy the link requirements.

Figure 422 HyperScale block-level analysis



For the BLKB analysis, to avoid using inaccurate timing data from incomplete blocks BLKA and BLKC, you could globally suppress all context data and revert to the budgeted constraints:

```
pt_shell> set_dont_override [get_ports $all_BLKA_BLKC_ports]
```

A more accurate approach is to ignore context data only on the ports that interface with incomplete blocks BLKA and BLKC, but still get the benefit of context data for BLKE (and other) interface paths:

```
pt_shell> set_dont_override [get_ports $all_BLKA_BLKC_ports]
```

HyperScale Attributes

You can query the following HyperScale-related attributes by using the `get_attribute` command:

- [Attributes for cell objects](#)
- [Attributes for design objects](#)
- [Attributes for pin objects](#)
- [Attributes for port objects](#)

Attributes for cell objects

Table 71 Cell Object HyperScale Attributes

Cell attribute	Description
block_config_name (string)	Specifies the named session for saved HyperScale block data; applies to hierarchical cells for HyperScale top-level runs.

Table 71 Cell Object HyperScale Attributes (Continued)

Cell attribute	Description
block_config_path (string)	Specifies the configuration path of the HyperScale block; applies to hierarchical cells for HyperScale top-level runs.
block_timestamp (string)	Returns a time stamp showing when the hierarchical block was created or modified.
critical_path_max (string)	Specifies for each path group at the block level a structured list containing the name of the path group, the levels of logic of the critical path, length, and setup slack.
critical_path_min (string)	Specifies for each path group at the block level a structured list containing the name of the path group, the levels of logic of the critical path, length, and hold slack.
is_hyperscale_block (boolean)	Returns true if the hierarchical cell is a HyperScale block.
is_mim_context_reference (boolean)	Returns true if the cell is being used as the reference instance during the HyperScale multiply instantiated module (MIM) merging flow.
leaf_cell_count (integer)	Returns the number of leaf-level cells in the hierarchical block that was used to create the abstracted hierarchical instance.
original_ref_name (string)	Specifies the original reference name of hierarchical blocks within the HyperScale model. This attribute applies to hierarchical cells for HyperScale top-level runs. This attribute is needed to apply constraints and LEF files to non-HyperScale subdesigns within HyperScale models at the top level.
pin_count (integer)	Returns the number of pins in the hierarchical block that was used to create the abstracted hierarchical instance.

Attributes for design objects

Table 72 Design Object HyperScale Attributes

Design attribute	Description
context_config_name (string)	Returns the named session of the HyperScale top context applied in HyperScale block-level runs.
context_config_path (string)	Returns the configuration path of the HyperScale top context applied in HyperScale block-level runs.
context_timestamp (string)	Returns the time stamp of the HyperScale top context applied in HyperScale block-level runs.
is_context_available (boolean)	Returns true if the HyperScale top context is available to be applied. This attribute is applicable only for HyperScale block runs. The attribute returns true after link and after a timing update only if the top-level context exists for each configuration.
is_context_loaded (boolean)	Returns true if the HyperScale top context has been loaded and applied during a timing update. This attribute is applicable only for HyperScale block runs. The attribute returns true after a timing update only if the top-level context data was loaded during the timing update, even if the context was not fully applied because of override controls. The attribute is never true before a timing update.
is_hyperscale_block (boolean)	Returns true if the design is a HyperScale block.
is_hyperscale_top (boolean)	Returns true if the design contains one or more HyperScale blocks. For a mid-level HyperScale design, the attributes is_hyperscale_block and is_hyperscale_top are both set to true.
timestamp (string)	Specifies the time stamp for the current HyperScale run. This attribute applies to top and block designs.

Attributes for pin objects

Table 73 Pin Object HyperScale Attributes

Pin attribute	Description
has_model_mismatch_clock (boolean)	Returns true if at least one model arrival or required time refers to an unmapped clock. This attribute applies to a side-input or stub pin.
is_abstracted (boolean)	Returns true if the port is abstracted in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after a timing update at the block level. The attribute returns true for <code>set_port_abstraction -ignore</code> . Ports with <code>is_user_abstracted</code> set to true are a subset of ports with <code>is_abstracted</code> set to true.
is_excluded (boolean)	Returns true if the pin is excluded from timing analysis. The pin is inside the HyperScale block and visible at HyperScale top, and is retained for circuit completion.
is_interface (boolean)	Returns true if the pin is on an interface path inside a HyperScale block and can be seen from the HyperScale top. Query this attribute at the top level.
is_internal_aggressor (boolean)	Returns true if the pin is a dangling aggressor inside the HyperScale block. It is visible at HyperScale top because it belongs to an aggressor net that can affect a victim net in the top-to-block interface. Query this attribute at the top level.
is_model_interface (boolean)	Returns true if the pin is considered to be on the interface of the current design when a HyperScale model is created for the current design. Pins on the interface are written out in the HyperScale model data and visible at the top level when the model is instantiated. Query this attribute at the block level after a timing update.
is_model_internal_aggressor (boolean)	Returns true if the pin is not on the interface of the current design, but the HyperScale model needs to consider its aggressor effects because the net connected with the pin is physically coupled to the interface logic through parasitics. The pin is written in the HyperScale model and is visible at the top level, and contributes to SI analysis of the interface. Query this attribute at the block level after a timing update.
is_model_side_input (boolean)	Returns true if the pin is a side input in a HyperScale block model. This is an input pin of a cell in a path from a block input port to a block interface register, but the pin is not itself in the path. Query this attribute at the block level after a timing update.
is_model_stub (boolean)	Returns true if the pin is a stub pin in a HyperScale block model. This is a pin retained in the model due to its role as a high-fanout timing point or an internal capacitive load in an interface path. Query this attribute at the block level after a timing update.

Table 73 Pin Object HyperScale Attributes (Continued)

Pin attribute	Description
is_side_input (boolean)	Returns true if the pin is inside the HyperScale block and visible at HyperScale top, where the entire fanin (starting from internal registers) is removed, and valid slews, arrivals, or case values are annotated.
is_stub (boolean)	Returns true if the pin is inside the HyperScale block and visible at HyperScale top, where the entire fanout (ending at internal registers) is removed, and valid required times are annotated for accurate slack computation.
is_user_abstracted (boolean)	Returns true if the pin is abstracted because of user-specified settings (<code>set_port_abstraction -ignore</code>) in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after a timing update at the block level. Ports with <code>is_user_abstracted</code> set to true also have <code>is_abstracted</code> set to true.

Attributes for port objects

Table 74 Port Object HyperScale Attributes

Port attribute	Description
is_abstracted (boolean)	Returns true if the port is abstracted in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after a timing update at the block level. The attribute returns true for <code>set_port_abstraction -ignore</code> . Ports with <code>is_user_abstracted</code> set to true are a subset of ports with <code>is_abstracted</code> set to true.
is_clock_dont_override (boolean)	Returns true if context override is not applied because of the context referring to unmapped clocks. This attribute applies only to ports at the block level.
is_netlist_dont_override (boolean)	Returns true if context override is not applied because of an anchor leaf pin change on a port net.
is_user_abstracted (boolean)	Returns true if the port is abstracted because of user-specified settings (<code>set_port_abstraction -ignore</code>) in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after a timing update at the block level. Ports with <code>is_user_abstracted</code> set to true also have <code>is_abstracted</code> set to true.
is_user_dont_override (boolean)	Returns true if context override is not applied because of user-specified settings. This attribute applies only to ports at the block level.

Context Characterization

Context characterization captures the timing context of a block design from its environment in the parent design. The timing context of an instance includes clock information, input arrival times, output delay times, timing exceptions, design rule constraints, propagated constants, wire load models, input drives, and capacitive loads.

To learn about characterizing the context of a block design, see

- [Characterizing the Context of a Block](#)
- [Example of Context Characterization](#)
- [Reporting the Characterized Context](#)
- [Exporting the Characterized Context](#)
- [Writing Physical Information](#)
- [Removing Context Information](#)

- Using Context for Synthesis or Optimizing Constraints
 - Context Characterization Exclusion of Lower-Level Blocks
 - Using Context for Block-Level Timing Analysis
 - Context Handling of Cross-Boundary Timing Exceptions
 - Writing and Reading Context in Binary Format
-

Characterizing the Context of a Block

To characterize and export the context of a specific block from a top-level design, use the following flow:

```
# Read and link the design
read_verilog ...
link_design

# Back-annotate parasitics
read_parasitics ...

# Apply design constraints
read_sdc ...

# Specify cells to be characterized
characterize_context -block blockA -instances U1

# Update the timing information
update_timing

# Report the context information
report_context -nosplit [get_cells U12]

# Export the context information
write_context -format ptsh -output U12_context [get_cells U12]
```

The `characterize_context` command itself does not characterize the block context; instead, it marks the block to be characterized during the next timing update.

Full-netlist blocks, HyperScale blocks, and ETM instances can be characterized.

Block characterization generates and exports the timing and physical information shown in the following table.

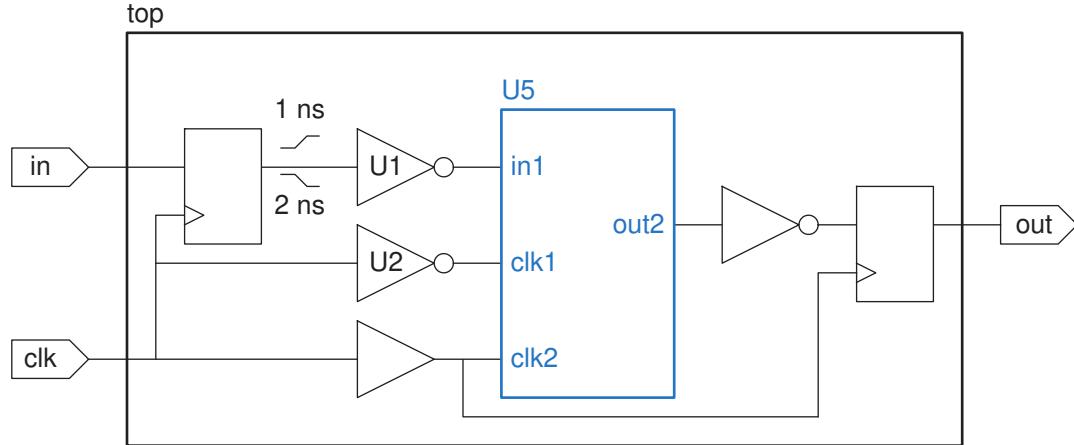
Table 75 Information Generated by Context Characterization

Context data	Description	Output by write_context as these commands
Clocks	Clock waveform, clock signal latency, and clock uncertainty information for all clocks that drive registers or ports that launch data signals to or receive data signals from the block	create_clock, set_clock_latency, set_clock_uncertainty
Input arrival times	Arrival times of data signals at input pins of the block	set_input_delay
Output delay times	Required times of data signals at output pins and their related clocks	set_output_delay
Point-to-point timing exceptions	<ul style="list-style-type: none"> • Timing exceptions that affect the block being characterized: • False paths • Multicycle paths • <code>max_delay</code> and <code>min_delay</code> exceptions between clocked startpoints and endpoints <p>Point-to-point exceptions that cross the boundary of the characterized block are excluded from the written context information, except in <code>gbc</code> (binary context) format.</p>	set_false_path, set_multicycle_path, set_max_delay, set_min_delay
Constant logic values on inputs	Constant logic values that are propagated to input pins of the block being characterized; logic constant values as logic constants and case analysis values as case analysis	set_case_analysis
Input drive strength	Drivers of input pins of the block being characterized; if the input pin is driven by a port that has a linear drive specified, this drive is also characterized	set_drive, set_driving_cell
Port capacitance	Pin capacitance is written as a <code>set_load</code> command. The capacitance is not annotated on the boundary nets. Instead, it is set to zero, and the port has annotated wire capacitance to account for this capacitance.	set_load
Wire load models	Disables the automatic wire load selections and sets the wire load mode to <code>top</code> ; does not support other wire load modes	set_auto_wire_load_selection false set_wire_load_mode top
Design rule constraints	Design rule constraints such as <code>max_capacitance</code> , <code>min_capacitance</code> , <code>max_fanout</code> , <code>min_fanout</code> , <code>max_transition</code> , <code>min_transition</code> , and <code>fanout_load</code>	set_max_capacitance, set_min_capacitance, set_max_fanout, set_max_transition, set_fanout_load

Example of Context Characterization

The following script example sets the port interface attributes and characterizes the U5 block.

Figure 423 Context characterization example



```
current_design top

# clk is a top-level port
create_clock -period 10 -waveform {0 5} [get_ports clk]
set_propagated_clock clk
# in is a top-level input port
set_input_delay -clock clk 1 [get_ports in]
# out is a top-level output port
set_output_delay -clock clk 1 [get_ports out]

# characterize and write context
characterize_context -block blockA -instances U5
update_timing
report_context U5
write_context -nosplit -format ptsh \
    -output U5_context [get_cells U5]
```

Reporting the Characterized Context

To report the timing context derived by the `characterize_context` command, use the `report_context` command. By default, this command reports all context information.

To report the timing-related context information, including clocks and their waveforms, point-to-point timing exceptions, input external delay, and output external delays, for specific instances, use the `-timing` option. For example:

```
pt_shell> report_context -timing U5
```

To report the environment and design rule constraint information, such as wire load models, driving cell information about input pins, and capacitive load on input and output pins, use the `-environment` and `-design_rules` options:

```
pt_shell> report_context -environment -design_rules U5
```

Exporting the Characterized Context

After you use the `characterize_context` command, you export context information to use later in the PrimeTime tool, or other tools such as Design Compiler, by using the `write_context` command. For example, to write the context information for the U5 and U7 instances as a PrimeTime script, use this command:

```
pt_shell> write_context -format ptsh -output U5_context {U5 U7}
```

In this example, U5 and U7 must be different instances of the same block.

The `-format` option specifies the format of the written characterization data:

- `ptsh` for PrimeTime script format (the default)
- `sdc` for Synopsys Design Constraints script format
- `dctcl` for Design Compiler script format
- `gbc` for binary context format

For information about the binary context format, see [Writing and Reading Context in Binary Format](#).

Writing Physical Information

During context characterization of a block design, the `characterize_context` command does not capture the delays and parasitics annotated on the internal nets of the block. To capture and export this information, use the `write_physical_annotations` command.

The following script example characterizes the context of the U2 cell and exports the context and physical information for use in Design Compiler:

```
# Specify the cell U2 to context characterization
characterize_context -block blockA -instances U2

# Export timing context
```

```
write_context -format dctcl -output U2.dctcl U2

# Export delays and parasitics
write_physical_annotations -sdf U2.sdf \
    -parasitics U2.rc -format dctcl -append U2.dctcl U2
```

Removing Context Information

If you no longer need to report or export the context that was previously created by the `characterize_context` command, you can delete the context information by using the `remove_context` command.

For example, to delete the context information for the U5 and U7 instances, use this command:

```
pt_shell> remove_context -instances {U5 U7}
```

Using Context for Synthesis or Optimizing Constraints

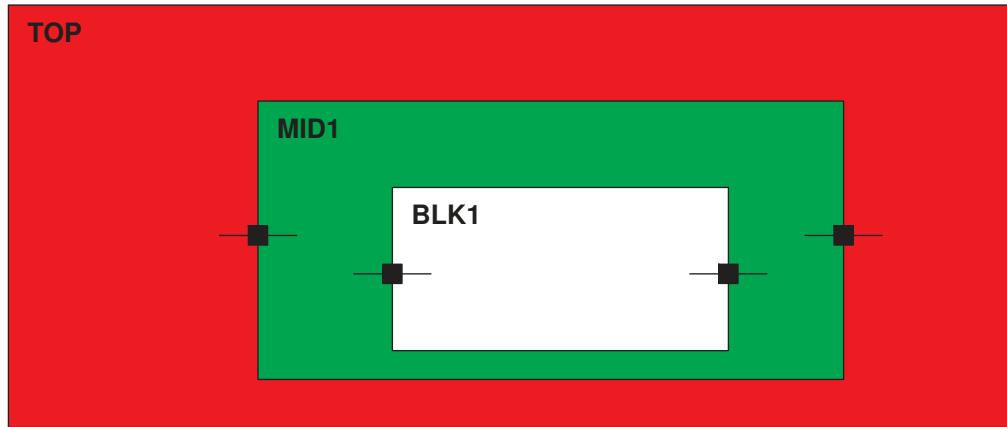
To characterize the context of a block design for setting synthesis or optimization constraints in Design Compiler:

1. In PrimeTime:
 - a. Read the top design.
 - b. Identify the block designs that require timing optimization.
 - c. Characterize the timing context for each block design.
 - d. Generate a Design Compiler script containing the context information.
 2. In Design Compiler:
 - a. Read the block design and script generated in the previous step.
 - b. Perform module-level optimization of the block designs.
-

Context Characterization Exclusion of Lower-Level Blocks

The HyperScale and context characterization flows support context generation of a block or the top level while excluding lower-level blocks. For example, consider the following three-level hierarchical design.

Figure 424 Three-Level Hierarchical Design Example



Mid-Level Context Generation

To generate the context for the mid-level block MID1 while excluding the lower-level block BLK1 (the context of the green part of the diagram), do the following:

```
set hier_context_exclude_sub_block true
characterize_context MID1
characterize_context MID1/BLK1
write_context -format ptsh
```

By setting the `hier_context_exclude_sub_block` variable to true, context characterization excludes the highest characterized subblock, BLK1. During block-level analysis, the subblock is considered a black box, and characterization captures the boundary timing on the I/O of the excluded subblock.

The `write_context` command, aside from the regular I/O timing on the outer boundary of MID1, also writes out the reversed boundary timing of BLK1 to time the remaining portion of MID1 without BLK1:

```
set_output_delay BLK1/in
set_input_delay BLK1/out
...
```

Top-Level Context Generation

To generate the context for the top level of the design while excluding the lower-level block MID1 (the context of the red part of [Figure 424](#)), do the following:

```
set hier_context_exclude_sub_block true
characterize_context MID1
characterize_context MID1/BLK1
```

```
characterize_context -top
write_context -format ptsh
```

When you set the `hier_context_exclude_sub_block` variable to true, context characterization excludes the highest characterized subblock, MID1. During top-level analysis, the subblock is considered a black box, and characterization captures the boundary timing on the I/O of the excluded subblock.

The `write_context` command, aside from the regular top-level context, also writes out the reversed boundary timing of MID1 to time the remaining portion of the top level without MID1:

```
set_output_delay MID1/in
set_input_delay MID1/out
...
```

Using Context for Block-Level Timing Analysis

To characterize the context of a block design for performing block-level timing analysis in PrimeTime:

1. Read the design into PrimeTime.
2. Identify the block designs for timing analysis.
3. Characterize the timing context for each block design.
4. Generate a PrimeTime script containing the timing assertions.
5. Read the block design into PrimeTime.
6. Read the script generated in step 4.
7. Analyze the timing of the block design.

Context Handling of Cross-Boundary Timing Exceptions

The HyperScale and context characterization flows capture cross-boundary timing exceptions during top-level analysis where the exceptions are fully specified, and the `write_context` command writes out this information as part of the block boundary context. For block-level analysis, the tool consumes the data to continue the necessary and proper timing propagation along the affected paths.

The following guidelines apply to the context handling of cross-boundary timing constraints:

- All timing exceptions must be defined in the flat constraints for top-level analysis, including:
 - Top-level-only exceptions
 - Block-internal exceptions
 - Cross-boundary block interface exceptions, in which referenced objects are all clocks or all pins inside the block (specify these constraints as both top-level flat constraints and block-level constraints)
 - Cross-boundary straddling exceptions, in which referenced objects are pins at the top level and block level (these constraints can be specified only as top-level flat constraints)
- Block-level constraints must be a proper subset of these exceptions, and may only include:
 - Block-internal exceptions
 - Cross-boundary block interface exceptions, in which referenced objects are all clocks or all pins inside the block

When you use binary context, you define the cross-boundary straddling exceptions fully as top-level constraints, not block-level constraints. During top-level analysis, the `write_context` command writes out the exception information. For analysis at the block level, the `read_context` command automatically reads in and uses this context-specific constraint information.

To write out the context data in text format (`ptsh`, `dctcl`, or `sdc` script instead of binary), set the following variable before you use the `write_context` command:

```
set_app_var hier_constraints_include_cross_boundary_exceptions true
```

When the top level has cross-boundary timing exceptions, the `write_context` command adds virtual clocks and timing exception commands that fully specify the exceptions. You can see the generated `create_clock` and related timing exception commands in the scripts written out by the `write_context` command.

The following diagram shows various examples of top-level, block-level, and cross-boundary paths. The table below the diagram shows how timing exceptions for these paths are specified and how the `write_context` command handles these exceptions.

Figure 425 Top-Level, Block-Level, and Cross-Boundary Paths

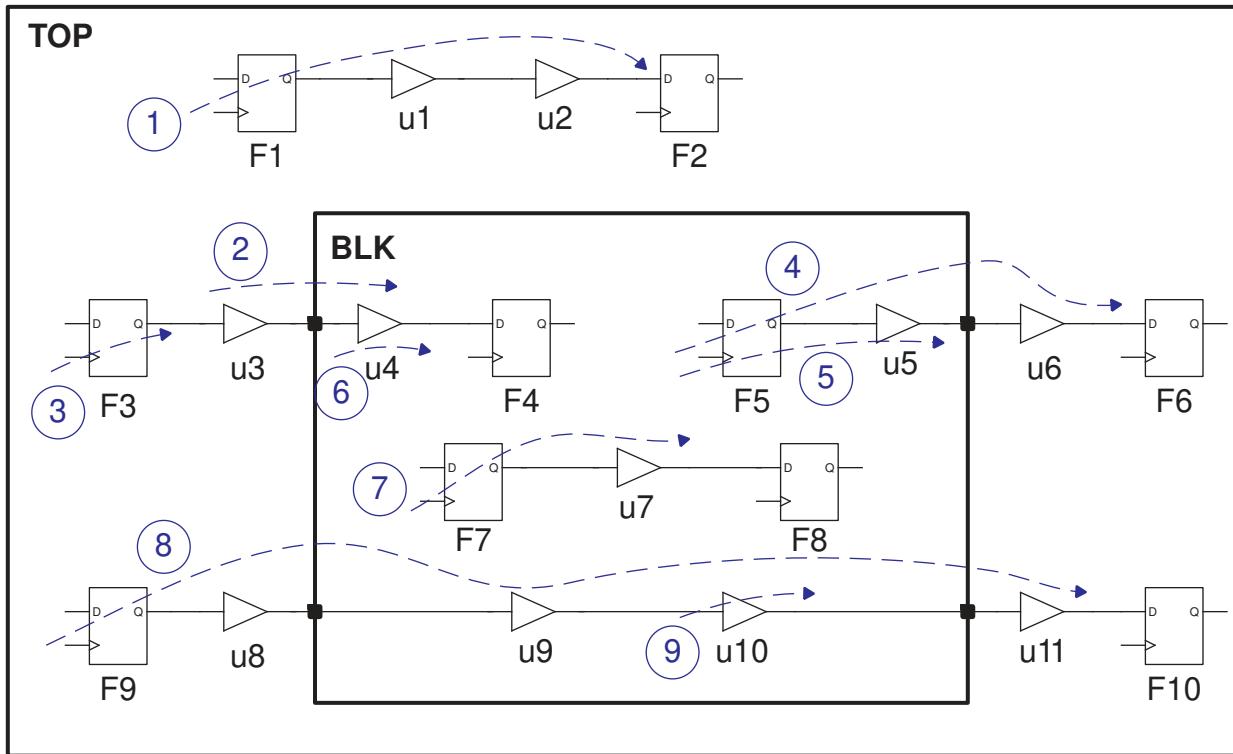


Figure 426 Timing Constraints and write_context

Timing exception path specification (for example, <code>set_multicycle_path -from ...</code>)	Constraint specified by user?			Does <code>write_context</code> write it?	
	Flat analysis	Top analysis	Block analysis	Script format	Binary format
1. <code>-from F1/CP -through u1/A -to F2/D</code>	Yes	Yes	No	No	No
2. <code>-through u3/A -through BLK/u4/Z</code>	Yes	Yes	No	No	Yes
3. <code>-from F3/CP</code>	Yes	Yes	No	No	Yes
4. <code>-from BLK/F5/CP -through u6/A -to F6/D</code>	Yes	Yes	No	No	Yes
5. <code>-from BLK/F5/CP -through BLK/u5/A</code>	Yes	Yes	Yes	Yes	No
6. <code>-through BLK/u4/Z</code>	Yes	Yes	Yes	Yes	No
7. <code>-from BLK/F7/CP -through BLK/u7</code>	Yes	No	Yes	Yes	No
8. <code>-from F9/CP -thr BLK/u9/Z -thr u11/A</code>	Yes	Yes	No	No	Yes
9. <code>-through BLK/u10/A</code>	Yes	Yes	Yes	Yes	No
10. <code>-from CLK1 -to CLK2</code>	Yes	Yes	Yes	Yes	No

Writing and Reading Context in Binary Format

You can write out the characterization context information in binary format instead of a script in PrimeTime, Design Compiler, or SDC format. The binary format includes more types of context information than the script formats, possibly resulting in more accurate block-level timing analysis. Using the binary context format requires a PrimeTime SI license.

The script formats include the following context information for a block:

- I/O boundary conditions
- Block-internal constraints and clock definitions

The binary format includes additional types of context information, such as:

- Clock reconvergence pessimism removal (CRPR) information originating from shared clock path segments outside of the block
- Advanced on-chip variation (AOCV) and parametric on-chip variation (POCV) metrics for paths that cross the block boundary

- State-dependent cross-boundary exceptions, such as a false path exception set on a path that starts at the top level and crosses a hierarchical boundary into a lower-level block
- MUX clock exclusivity information automatically generated after you set the `timing_enable_auto_mux_clock_exclusivity` variable to `true`

To write out the block context in binary format, use the `write_context` command with the `-format gbc` option:

```
pt_shell> write_context BLK_INST -format gbc -output GBC_INST
```

When you write out the context in binary format, the `-output` argument specifies a directory (rather than a text file) as the output of the command. The command writes out the context as a set of binary files in the specified directory.

To read in the binary context information for block-level analysis, use the `read_context` command:

```
pt_shell> read_context ./GBC_INST
```

If you use the binary context format, you still need to apply the block-internal constraints using one of the script formats to ensure consistent clock context information. For example, the following script characterizes and writes out the context for a block:

```
read_verilog TOP_DESIGN.v
link_design

read_parasitics ...

read_sdc ...

characterize_context -block blockA -instances BLK_INST

# Tighten the max delay constraint by 10 percent for block
set_context_margin -max -delay -percent 10.0 -objects BLK_INST

update_timing

report_context -nosplit [get_cells BLK_INST]
write_context BLK_INST -format gbc -output GBC_INST
write_context BLK_INST -format ptsh -output BLK_CONSTR.tcl
```

The following script performs block-level analysis in a new PrimeTime session using the previously written context information:

```
read_verilog BLK.v
link_design

read_parasitics ...

# Source block-internal context script in ptsh format
```

```

source BLK_CONSTR.tcl

# Retain the user-defined timing budget constraints on data bus ports
set_dont_override [get_ports DATA_BUS*]

# Read binary context information from storage directory
read_context ./GBC_INST

update_timing

report_context -nosplit

```

Alternatively, you can use the `set_clock_map` command to explicitly specify the mapping between top-level and block-level clocking.

Binary Context for Multiply Instantiated Modules

To generate the binary context of a multiply instantiated module (MIM), use the following flow:

```

# Characterize the cell
characterize_context -block block_name -instances instance_list
update_timing
...
# Write out the context in binary format (gbc)
write_context block_name -output topDir -format gbc

```

To read and apply the context, use the following flow:

```

link_design block
...
# Load the context
read_context topDir
update_timing

```

Binary Context and Instance Grouping

You can organize multiple instances of the same block into groups to merge the context information for each group, as shown in the following example.

Top-level analysis with block instance grouping:

```

read_verilog blk_A.v
read_verilog top.v
link_design TOP
...
characterize_context -block blkA -name g1 -instances {a1 a2}
characterize_context -block blkA -name g2 -instances {a3 a4}
update_timing
...
write_context {a1 a2 a3 a4} -format gbc $Context_dir

```

Block-level analysis:

```
read_verilog block_A.v
link_design BLK_A
...
read_context $Context_dir -name g1      # Read context for a1 and a2
update_timing
...
```

To automatically merge the constraints for all instances of the same block, use the following command in the top-level analysis session:

```
set_hier_config -enable_mim
```

Reducing Clock Latency Pessimism During Context Merging

By default, context merging uses a “min-of-min and max-of-max” (min/max) approach for the merging, using the smallest of all min-condition values and the largest of all max-condition values. For clock arrivals, this can introduce pessimism in the block-level analysis, especially in register-to-register paths, because crosstalk victim and aggressor windows become wider than in any instance.

To reduce this pessimism, you can use a “largest window” mode for clock arrival merging by setting the following variable:

```
pt_shell> set_app_var \
           hier_context_merge_clock_latency_mode largest_window
```

In this mode, context merging uses the clock input arrivals from the instance with the widest early-to-late latency span. The tool issues a message when this occurs:

```
Information: Merged latency is the largest window for clock "%s".
(HS-059)
```

However, “largest-window” merging can only be applied to clocks that are asynchronous to all other clocks entering the instances.

When the variable is set to `min_max_value` (the default), clock arrivals use min/max merging.

Data input arrivals always use min/max merging.

Context Budgeting

Context budgeting is similar to context characterization, except that the characterized block contexts are adjusted to distribute slack across the blocks. This can prevent blocks from being over-optimized or under-optimized when a set of blocks is optimized in parallel.

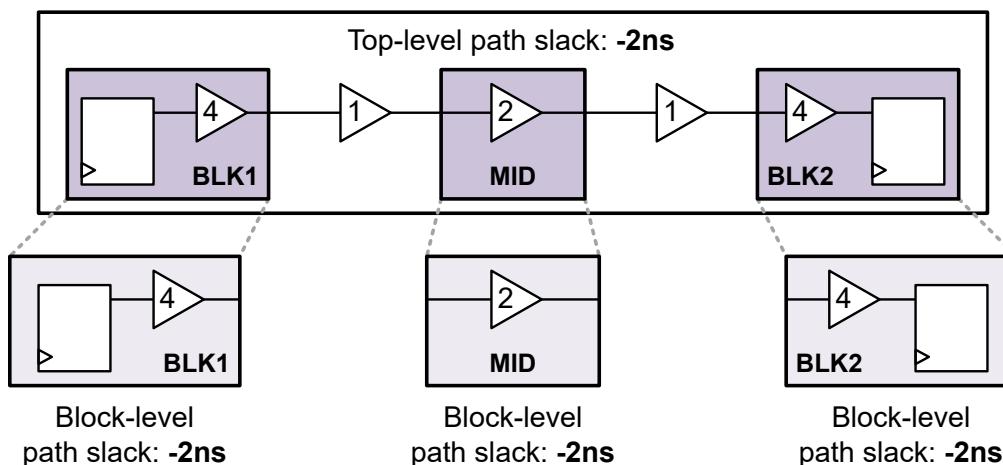
Context budgeting is described in the following topics:

- [Overview of Context Budgeting](#)
- [How Contexts are Adjusted During Budgeting](#)
- [Performing Context Budgeting](#)
- [Reporting the Context Budget](#)
- [Customizing the Context Budget](#)
- [Using Context Budgeting in a HyperScale Flow](#)
- [Limitations](#)

Overview of Context Budgeting

All pins along a critical path share the same slack value. Thus, when a critical path spans multiple blocks and those blocks are characterized with traditional context characterization, each block sees the same slack value along their portion of the critical path.

Figure 427 Context Characterization Replicates Top-Level Slack at Blocks

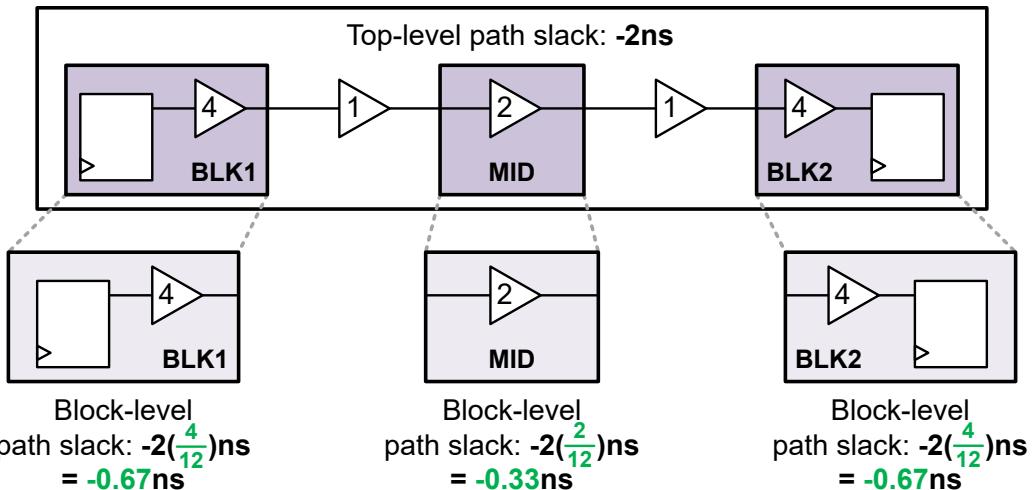


If these blocks are optimized in parallel using their traditionally characterized contexts,

- For failing paths, if each block-level run improves its portion of the path to zero slack, then the full top-level path (across all blocks) becomes over-optimized.
- For passing paths, if each block-level run relaxes its portion of the path to zero slack, then the full top-level path (across all blocks) can change from positive slack to negative slack.

To address this issue, PrimeTime provides a *context budgeting* feature that is similar to context characterization, except that the contexts are adjusted to proportionally allocate top-level path slack across the blocks.

Figure 428 Context Budgeting Allocates Top-Level Slacks Across Blocks



Then, if the blocks are optimized in parallel to their budgeted contexts, the top-level path slack converges to zero by construction because each block contributes only its portion of the required slack change, without over-optimization or under-optimization.

Context budgeting requires a PrimeTime-APX license.

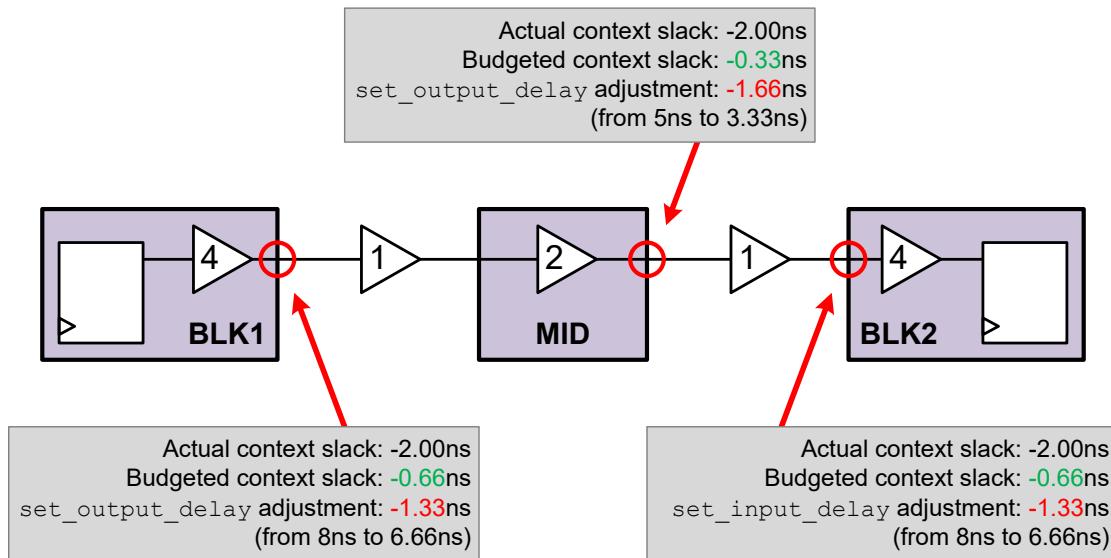
How Contexts are Adjusted During Budgeting

Context budgeting works by performing regular context characterization first, then adjusting those contexts to reallocate slacks across paths.

The block contexts are adjusted at the following locations:

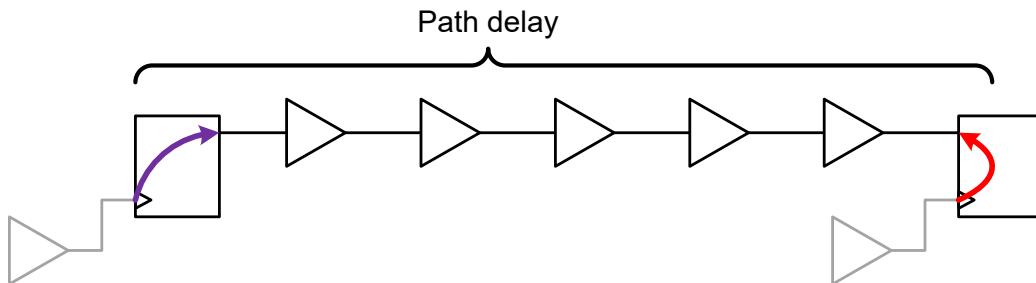
- Block-level register-to-output paths are adjusted at block output ports.
- Block-level input-to-output paths are adjusted at block output ports.
- Block-level input-to-register paths are adjusted at block input ports.

At each adjustment location, the `set_input_delay` or `set_output_delay` constraint is adjusted by the difference between actual slack and budgeted slack. For example,



The path delay used by budgeting is the sum of the following:

- Launching register CLK-to-Q delay
- All combinational data path delay
- Capturing register CLK-to-D constraint arc requirement (setup or hold)



Launch and capture propagated clock delays are not included in the path delay itself. However, they are considered by budgeting because they affect the path slack that is budgeted.

Performing Context Budgeting

The context budgeting flow is similar to the context characterization flow described in [Context Characterization](#), except for a few additional budgeting-specific commands.

The following script highlights the additional commands needed to add budgeting to context characterization:

```
read_verilog top.vg
link_design top top

create_clock -period 10 -waveform {0 5} CLK
set_propagated_clock CLK
set_input_delay -clock CLK 2 [get_ports in]
set_output_delay -clock CLK 2 [get_ports out]

# specify which blocks to characterize/budget
characterize_context -block block1 -instances BLK1
characterize_context -block block2 -instances BLK2

# update the design timing
update_timing

# perform budgeting and report some results
set_timing_budget -mode pin_slack
update_budget
report_budget -through BLK1/z
report_budget -through BLK2/A

write_context -nosplit -format ptsh \
    -output BLKA_context [get_cells BLKA]
```

The `characterize_context` command itself does not characterize the block context; instead, it marks the block to be characterized during the next timing update.

The `update_budget` command adjusts the block contexts as described in [How Contexts are Adjusted During Budgeting](#). It must be run after the design timing is updated.

Reporting the Context Budget

After you run the `update_budget` command to compute your budget, you can use the `report_budget` command to report the budgeted paths through various block boundary pins of interest.

Use the `-through` option of the `report_budget` command to specify a pin through which to report. The specified pin must be a hierarchical pin of a cell previously configured for budgeting by the `characterize_context` command.

The segments for the given path are reported, along with the hierarchical pins that delineate the segments.

```
pt_shell> report_budget -through MID/z
...
Slack: -2.00
```

Required time: 10.00
From clock: CLK
To clock: CLK
Path Type: max rise

Point	Mode	Delay	Weight	Budget
<hr/>				
BLK1/FF/CP				
BLK1/Z	pin_slack	4.00	0.33	3.33
MID/A	pin_slack	1.00	0.08	0.83
MID/Z	pin_slack	2.00	0.17	1.67
BLK2/A	pin_slack	1.00	0.08	0.83
BLK2/FF/D	pin_slack	4.00	0.33	3.33
<hr/>				

If the specified -through pin has a context adjustment applied to it, the report also includes the adjustment computation details after the segment list:

```
MID/A -> MID/Z
Budget = delay + slack * weight = 2.00 + -2.00 * 0.17 = 1.67
Output delay margin = slack + delay - budget = -2.00 + 2.00 - 1.67 = -1.67
Output budget = context + Output delay margin = 3.33
```

The `report_budget` command also provides a `-delay` option that restricts the report path to specific min/max/rise/fall transitions through the specified pin. The default is to report the worst max-delay path through the specified pin.

The `report_budget` command always reports the critical path through the specified -through pin. Subcritical paths cannot be reported, as they are not relevant to the budgeting calculations. Only one -through option can be specified.

Customizing the Context Budget

The default budgeting behavior is to allocate slack proportionally by path segment delay. However, the default budgeting behavior can be customized in a variety of ways.

The following topics describe how the context budget can be customized:

- [Applying Slack Margins](#)
- [Setting the Target Slack of a Block Segment](#)
- [Setting the Exact Delay of a Block Segment](#)
- [Setting the Clock Period Percentage of a Block Segment](#)
- [Setting a Minimum Block Segment Delay](#)
- [Precedence of Context Budget Specifications](#)

Applying Slack Margins

You can use the `-slack_margin` and `-positive_slack_margin` options of the `set_timing_budget` command to add slack margin to the budgeted constraints.

These slack margin values are applied to entire paths (from startpoint to endpoint), not to path segments. They are global margin settings that apply to all paths; they cannot be applied to individual blocks or segments.

These options work as follows:

- The `-slack_margin` option affects violating paths. It always subtracts the specified margin value from the original negative slack value.
- The `-positive_slack_margin` option affects passing paths. It subtracts the specified margin value from the original positive slack value, but the adjustment is limited to reducing the slack to zero (it cannot adjust a passing path into a violating path).

The `-positive_slack_margin` option can only be used together with the `-slack_margin` option.

For example,

```
pt_shell> set_timing_budget \
    -slack_margin 1.00 -positive_slack_margin 0.50
```

Because slack margin values are subtracted, positive margin values tighten the timing requirements.

When you apply a slack margin, the header of the `report_budget` report shows the original slack value (without the margin):

```
Slack: -2.00 (does not include slack margin)
Required time: 10.00
...

```

but the budgeting calculations after that use the margin-adjusted slack value:

report_budget without slack margin					report_budget with -slack_margin of 1.0				
Point	Mode	Delay	Weight	Budget	Mode	Delay	Weight	Budget	
BLK1/FF/CP									
BLK1/Z	pin_slack	4.00	0.33	3.33	slack_margin	4.00	0.33	3.00	
MID/A	pin_slack	1.00	0.08	0.83	slack_margin	1.00	0.08	0.75	
MID/Z	pin_slack	2.00	0.17	1.67	slack_margin	2.00	0.17	1.50	
BLK2/A	pin_slack	1.00	0.08	0.83	slack_margin	1.00	0.08	0.75	
BLK2/FF/D	pin_slack	4.00	0.33	3.33	slack_margin	4.00	0.33	3.00	
		(sum: 12.00)		10.00)		(sum: 12.00)		9.00)	

Setting the Target Slack of a Block Segment

The `-target_slack` option of the `set_timing_budget` command allows you to specify the wanted *budgeted* slack of a block segment.

With this specification applied, the affected block segment does not participate in the weighted distribution of slack. Instead, the segment context is adjusted to precisely achieve the wanted slack, then the surrounding segments are budgeted to make up the balance of the slack.

For example, given the following command:

```
pt_shell> set_timing_budget -target_slack -0.10 MID/A
```

then a resulting budget might be as follows:

Point	Mode	Delay	Weight	Budget
BLK1/FF/CP				
BLK1/Z	pin_slack	4.00	0.40	3.24
MID/A	pin_slack	1.00	0.10	0.81
MID/Z	target_slack_budget	2.00	0.00	1.90
BLK2/A	pin_slack	1.00	0.10	0.81
BLK2/FF/D	pin_slack	4.00	0.40	3.24

The actual block delay through MID is 2.00 but the budgeted delay is 1.90. Thus the MID segment will have a slack of -0.10 in block-level analysis, and optimization will attempt to improve the delay by 0.10. In other words, the `-target_slack` value represents what delay change you want in block-level optimization.

If a `-target_slack` value of 0 is specified, it indicates that you do not want (or expect) any delay change for that segment.

The `-target_slack` option does not need to be set on context adjustment pins. It can be set on block input pins, block output pins, or on the block itself to affect all segments in it.

Setting the Exact Delay of a Block Segment

The `-delay_value` option of the `set_timing_budget` command allows you to specify the exact wanted *budgeted* delay of a block segment.

With this specification applied, the affected block segment does not participate in the weighted distribution of slack. Instead, the segment context is adjusted to precisely achieve the wanted delay, then the surrounding segments are budgeted to make up the balance of the slack.

For example, given the following command:

```
pt_shell> set_timing_budget -delay_value 1.75 MID/A
```

then a resulting budget might be as follows:

Point	Mode	Delay	Weight	Budget
<hr/>				
BLK1/FF/CP				
BLK1/Z	pin_slack	4.00	0.40	3.30
MID/A	pin_slack	1.00	0.10	0.82
MID/Z	delay_value	2.00	0.00	1.75
BLK2/A	pin_slack	1.00	0.10	0.82
BLK2/FF/D	pin_slack	4.00	0.40	3.30

The `-delay_value` option does not need to be set on context adjustment pins. It can be set on block input pins, block output pins, or on the block itself to affect all segments in it.

Setting the Clock Period Percentage of a Block Segment

The `-percent_value` option of the `set_timing_budget` command allows you to specify the wanted *budgeted* delay of a block segment as a percentage of the clock period.

With this specification applied, the affected block segment does not participate in the weighted distribution of slack. Instead, the segment context is adjusted to precisely achieve the specified delay, then the surrounding segments are budgeted to make up the balance of the slack.

For example, given the following command:

```
pt_shell> set_timing_budget -percent_value 15 MID/A
```

with BLK1/FF and BLK2/FF clocked by a 10ns clock, then a resulting budget might be as follows:

Point	Mode	Delay	Weight	Budget
<hr/>				
BLK1/FF/CP				
BLK1/Z	pin_slack	4.00	0.40	2.60
MID/A	pin_slack	1.00	0.10	0.65
MID/Z	delay_value	2.00	0.00	1.50
BLK2/A	pin_slack	1.00	0.10	0.65
BLK2/FF/D	pin_slack	4.00	0.40	2.60

The percentage value is applied to the available clock period from the startpoint to the endpoint, taking the ideal launch/capture clock waveforms and multicycle exceptions into account. The propagated latency skew between the launch and capture clocks is not considered.

The `-percent_value` option does not need to be set on context adjustment pins. It can be set on block input pins, block output pins, or on the block itself to affect all segments in it.

Setting a Minimum Block Segment Delay

You can set the `timing_budget_segment_delay_threshold` variable to a minimum delay value to be used when computing block segment budgets.

For example,

```
pt_shell> set_app_var timing_budget_segment_delay_threshold 2.5
```

This minimum value applies to all block segments: register-to-output, input-to-output, and input-to-register. It does not apply to top-level logic segments.

With the preceding variable setting applied, then a resulting budget might be as follows:

Point	Mode	Delay	Weight	Budget
<hr/>				
BLK1/FF/CP				
BLK1/Z	pin_slack	4.00	0.32	3.36
MID/A	pin_slack	1.00	0.08	0.84
MID/Z	pin_slack	2.50	0.20	2.10
BLK2/A	pin_slack	1.00	0.08	0.84
BLK2/FF/D	pin_slack	4.00	0.32	3.36
<hr/>				

Note that the minimum value is the pre-budgeting value; it is then adjusted by budgeting as needed.

Reporting the Budgeting Customizations

To report the budgeting customizations applied by the `set_timing_budget` command, use the `report_timing_budget` command.

For example,

```
pt_shell> report_timing_budget
```

Point	Budget Allocation Mode	Path Type	Value
<hr/>			
<Global Mode>	slack_margin		1.00
BLK1/Z	delay_value	max	3.00

Precedence of Context Budget Specifications

When multiple specifications apply to the same path segment,

- Pin-level specifications take precedence over block-level specifications.

When multiple specifications are applied to segments along the same path, they are applied in the following order:

1. `-slack_margin` and `-positive_slack_margin` specifications are applied to compute an updated path slack.
2. `timing_budget_segment_delay_threshold` specifications are applied to increase small block segment delays as needed.
3. `-target_slack`, `-delay_value`, and `-percent_value` specifications are applied to compute segment-specific target delays.
4. The remaining segments (without segment-specific specifications) are budgeted proportionally to use the remaining available slack.

Using Context Budgeting in a HyperScale Flow

You can use context budgeting in a HyperScale flow.

When context budgeting is enabled in a HyperScale top-level run,

- The `set_hier_config -block` command automatically selects the block for context characterization; you do not need to use the `characterize_context` command.
You must still use the `characterize_context` command for any non-HyperScale (full-netlist) blocks that you want to include in budgeting.
- The `write_hier_data` command automatically includes the budgeting adjustments in the block context data; you do not need to use the `write_context` command.
You can still use the `write_context` command if you want the block constraints for standalone testing.

For example,

```
# configure HyperScale (blocks automatically included in budgeting)
set_app_var hier_enable_analysis true
set_hier_config -block BLK1 ...
set_hier_config -block BLK2 ...

read_verilog top.vg
link_design top top

create_clock -period 10 -waveform {0 5} CLK
set_propagated_clock CLK
set_input_delay -clock CLK 2 [get_ports in]
set_output_delay -clock CLK 2 [get_ports out]

# update the design timing
update_timing
```

```
# perform budgeting and report some results
set_timing_budget -mode pin_slack
update_budget
report_budget -through BLK1/Z
report_budget -through BLK2/A

# write HyperScale block contexts (includes budgeting adjustments)
write_hier_data ${HS_DIR}/TOP
```

In a HyperScale block-level run, you do not need to enable the context budgeting feature; the budgeted block context is used by default. To use the actual context instead of the budgeted context, use the `-exclude_budget` option of the `read_context` command to ignore the budgeting adjustments:

```
pt_shell> # use actual (non-budgeted) block context
pt_shell> read_context ${HS_DIR}/TOP -name BLK1 -exclude_budget
```

Limitations

Note the following limitations of context budgeting:

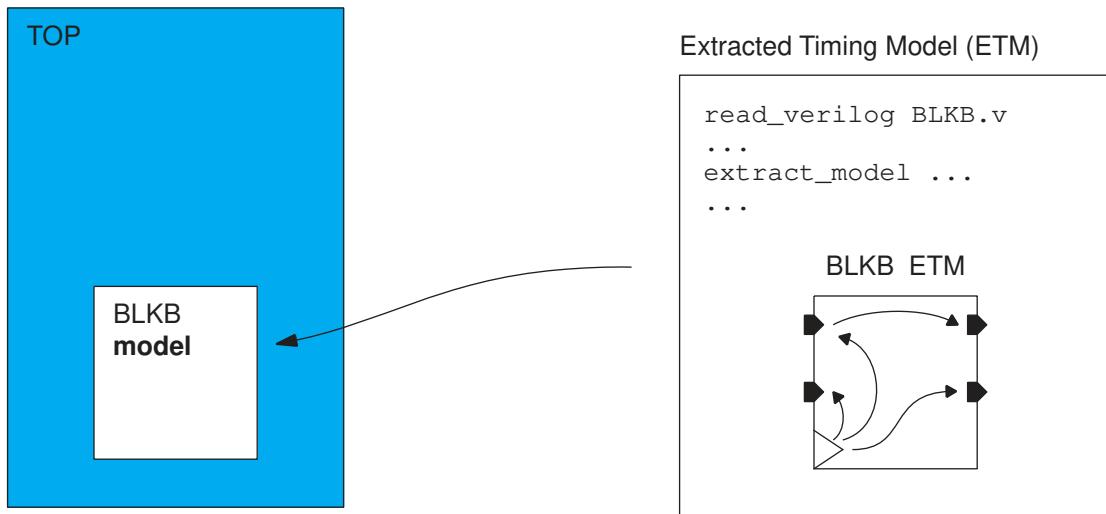
- The `-target_slack` option of the `set_timing_budget` command can only be used on block feedthrough (input-to-output) segments.
- The `update_budget` command can only be run once.

Extracted Timing Model (ETM)

An extracted timing model (ETM) is an abstract representation of a block, generated by the `extract_model` command. The model can be used in place of the block's netlist for timing analysis at a higher level of the design hierarchy, as shown in the following figure.

Figure 429 Extracted Timing Model Analysis

Hierarchical analysis using block models



Using a timing model offers the following benefits:

- Reduces the runtime and memory usage for full-chip analysis in the PrimeTime tool and other tools such as Design Compiler, IC Compiler, and IC Compiler II
- Protects the intellectual property of a netlist-based core

The extracted model is an abstraction of the block interface timing behavior using sequential and combinational timing arcs. The `extract_model` command extracts each timing arc and represents its delay as a function of input transitions and output loads in the form of nonlinear delay model (NLDM) and Composite Current Source (CCS) lookup tables. Therefore, you can use the ETM with different input transition times and different output loads and still get accurate results.

You extract the model by running the `extract_model` command. For example,

```
pt_shell> extract_model -output model2
```

This example generates a timing model for the current design and saves the model as a library cell named `model2.lib.db`. The model serves as a port-to-port leaf cell replacement for the design being modeled, with the boundary net capacitance values approximated inside the model.

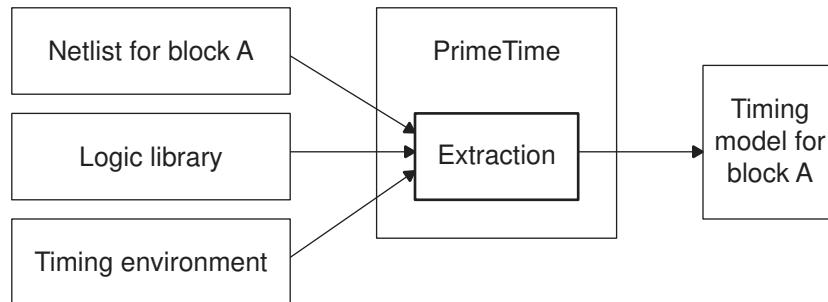
To specify the output format of the timing model, either `.db` or `.lib`, use the `-format` option of the `extract_model` command. The `.db` format can be used directly by most Synopsys tools. Use the `.lib` format when you want to be able to read and understand the timing arcs contained in the model, or for compatibility with a third-party tool that can read Liberty files.

Timing Model Extraction Overview

To generate an extracted timing model, you need the following elements:

- 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 430 Timing model extraction process



Before you generate a model with the `extract_model` command, ensure that the netlist contains no timing violations by using the `check_timing` or `report_constraint` command.

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 431](#) and [Figure 432](#) show an example of a gate-level design named “simple” and the library cell model extracted from the netlist. The generated library cell contains pin-to-pin timing arcs and eliminates the internal logic of the original design.

Figure 431 Gate-Level Netlist

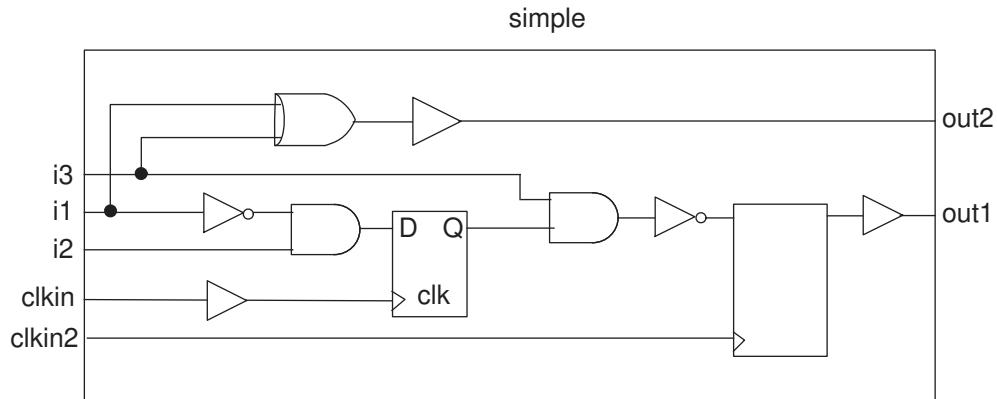
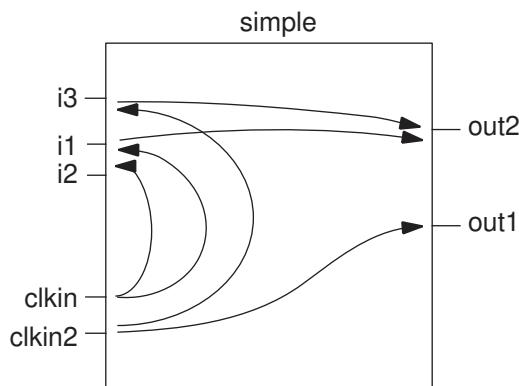


Figure 432 Extracted Library Cell Model



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.

The `extract_model` command generates a static timing model for the current design. To also generate noise or power information in the model, use the `-noise` or `-power` option.

You can create a test design containing an instance of the extracted library cell. To do this, use the `-test_design` option in the `extract_model` command. The test design is named `output_file_name_test.db` and the model file is named `output_file_name_lib.db`.

Usage Flow of ETM

The following sections summarize the block-level and top-level stages of the ETM flow.

Block-Level Analysis: Extraction and Validation

Block-level analysis is performed in conditions described so far:

- The model is extracted for every pair of mode and corner. Then the different modes for each corner are merged for use in PrimeTime analysis.
- As the extracted model should not be optimistic, the constraints at the boundary must cover the range of context in which the model is used. The output load constraints must be specified to cover min and max loading of the output ports. The input port constraints must cover the range of sharpest input transition to slowest input transition.
- In signal integrity analysis
 - For good accuracy, the setting of the `extract_model` engine must consider the size and range of indexes for the extracted timing table.
 - You can make your blocks arrival tolerant by setting infinite arrival windows on inputs.

Here is the block-level usage flow:

```
set link_path ...
set search_path ...
read_verilog ...
read_parasitics ...
load_upf ...
read_sdc block.sdc
update_timing
set hier_modeling_version 2.0
extract_model ... -validate timing
```

The validation step, automated with the `-validate timing` option, is needed to verify that the timing values for each endpoint of the initial netlist and of the extracted model instantiated in the wrapper match.

At the end of the process the summary is displayed after GBA analysis and PBA analysis:

	Totals	Slack	Transition Time	Capacitance	Rules
Passed	66531	13995	17512	35024	-
Failed	5	5	0	0	0
Total	66536	14000	17512	35024	0

Top-Level Analysis

Now the hierarchical blocks are replaced by the library cells created by `extract_model`, and static timing analysis is done at the top level.

```
set link_path ...
lappend link_path mod.lib.db
set search_path ...
read_verilog top
read_db $::pt_model_dir/ETM/modname/mod.lib.db
current_design top
read_parasitics top.spf
read_sdc top.sdc
source mod_inst.pt.gz
update_timing
```

The `mod_inst.pt` script contains “local” constraints such as local timing derating, timing exception on specific pins, and case analysis.

Preparing for Model Extraction

Before performing model extraction, you must perform timing analysis of the block according to the recommended PrimeTime methodology, including the following tasks:

- Check and resolve all warnings and errors in design linking, parasitics reading and constraints loading. Parasitic back-annotation warnings, errors, and library problems can cause accuracy issues.
- Run the `check_timing -verbose` command. Ensure that all warnings and errors are understood, verified, and acceptable for timing analysis and model extraction to proceed. For example, you must resolve unconstrained paths, unclocked registers, and mismatching rail voltages.
- Verify that the timing constraints are correct. Clocks and generated clocks must be defined correctly. To check and ensure the consistency and correctness of the timing constraints for PrimeTime, you can use [Constraint Consistency](#). In addition to constraints such as clocks, exceptions, and case values, you should define data and clock signal arrival time ranges at the block inputs, data and clock signal transition ranges at the block inputs, and the external load ranges at the block outputs. This setup determines the operational and applicable range for the block.

- Check the warnings and errors issued during the timing update. Nearly all the issues affecting the accuracy and validity of timing analysis also affect the quality of the ETM.
- Verify that the design does not have timing violations by using the `report_constraint` command. This is important because the ETM can still be generated for a block with timing violations. Violations inside the ETM are hidden during full-chip analysis.

Variables That Affect ETM Generation

To configure the accuracy and other aspects of generating an extracted timing model (ETM), set the variables in the following table. For a complete list of variables that affect ETM generation, use this command:

```
pt_shell> report_app_var extract_model_*
```

Table 76 Commonly-used variables for ETM generation

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.

Table 76 Commonly-used variables for ETM generation (Continued)

Extraction environment variable	Effect on extract_model command
extract_model_use_conservative_current_slew	Specifies whether to use a more conservative current context slew, which models the netlist behavior better if the netlist is in the worst-slew propagation mode.
extract_model_with_ccs_timing	Specifies whether to extract the model with CCS timing data.
extract_model_with_clock_latency_arcs	Specifies whether to enable modeling of clock tree insertion delay timing arcs in the extracted model.
extract_model_with_non_sequential_arcs	Specifies whether to enable modeling of set_data_check checks and non-sequential library cell checks as nonsequential setup and hold arcs in the extracted model.
timing_clock_gating_propagate_enable	Determines whether data signals that gate a clock are propagated past the gating logic.
timing_disable_clock_gating_checks	When set to true, disables all clock-gating checks and does not extract them to the model.
timing_enable_preset_clear_arcs	Specifies whether to enable or disable preset and clear timing arcs.

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 Model Extraction

After setting up your modeling environment (see [Preparing for Model Extraction](#)), you can extract a timing model:

- [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:

```
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_verilog design.v
```

4. Link your design:

```
pt_shell> link_design design
```

Preparing to Extract the Model

To preparing a model for extraction,

1. Define the wire load models or read the parasitics for the 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 any required false paths or other timing exceptions:

```
pt_shell> set_false_path -from IN4 -to OUT1
```

4. Check the 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:

```
pt_shell> extract_model -output example_model -format {db}
```

If you generate a .lib version of the timing model, you might notice it contains user-defined attributes such as `min_delay_flag` and `original_pin`. For more information about these attributes, search for them in [SolvNetPlus](#).

Timing Model Extraction Details

The following topics describe how the tool extracts simple timing arcs from the timing paths in the original design:

- [Boundary Nets and Internal Nets](#)
- [Paths From Inputs to Registers](#)
- [Paths From Inputs to Outputs](#)
- [Paths From Registers to Outputs](#)
- [Paths From Registers to Registers](#)
- [Clock Paths](#)
- [Transparent Latches and Time Borrowing](#)
- [Minimum Pulse Width and Minimum Period](#)
- [False Paths](#)
- [Clock-Gating Checks](#)

- [Back-Annotated Delays](#)
- [Noise Characteristics](#)

Boundary Nets and Internal Nets

The extracted model does not preserve the boundary nets of the original design. Instead, it factors the boundary net delays into the model.

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.

In most cases, register-to-output delay arcs are not extracted for unconstrained paths (such as from `set_false_path` or `set_sense_stop_propagation` specifications). However, they are extracted for paths that are unconstrained due to the following output-clock-related reasons:

- No `set_output_delay -clock` clock defined for the port
- `set_clock_group -asynchronous` relationship between the register clock and the `set_output_delay -clock` clock

because the external clock configuration might be different in the context where the model is 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.

Transparent Latches and Time Borrowing

The `extract_model` command can extract simple latch structures in the interface logic. The extracted model preserves the borrowing behavior up to two borrowing (transparent) latch levels deep, under the conditions specified at the time of model extraction. The model does not cover all possible borrowing behaviors.

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, the model extraction flow might not support those conditions. In that case, you can still use [HyperScale Analysis](#) for hierarchical analysis.

The Transparent Latch Level Limit

The `-latch_levels` option of the `extract_model` command specifies the maximum number of transparent latch levels from a port considered during model extraction. Beyond this limit, latches are considered nonborrowing (non-transparent) latch cells.

Valid values are from 0 to 2. The default is 2. Values above 2 are accepted but the actual limit remains at 2.

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 setup arcs when an input port goes through borrowing latches to a flip-flop or nonborrowing latch:
 - One setup arc for each borrowing latch
 - One setup arc for the final capturing flip-flop or nonborrowing latch
- It extracts delay arcs when an input port, or a clock connected to a flip-flop or nonborrowing latch, goes through borrowing latches to an output port:
 - One delay arc for the initial driving flip-flop or nonborrowing latch
 - One delay arc for each borrowing latch
- At each port, same-sense arcs are merged to optimize the model.

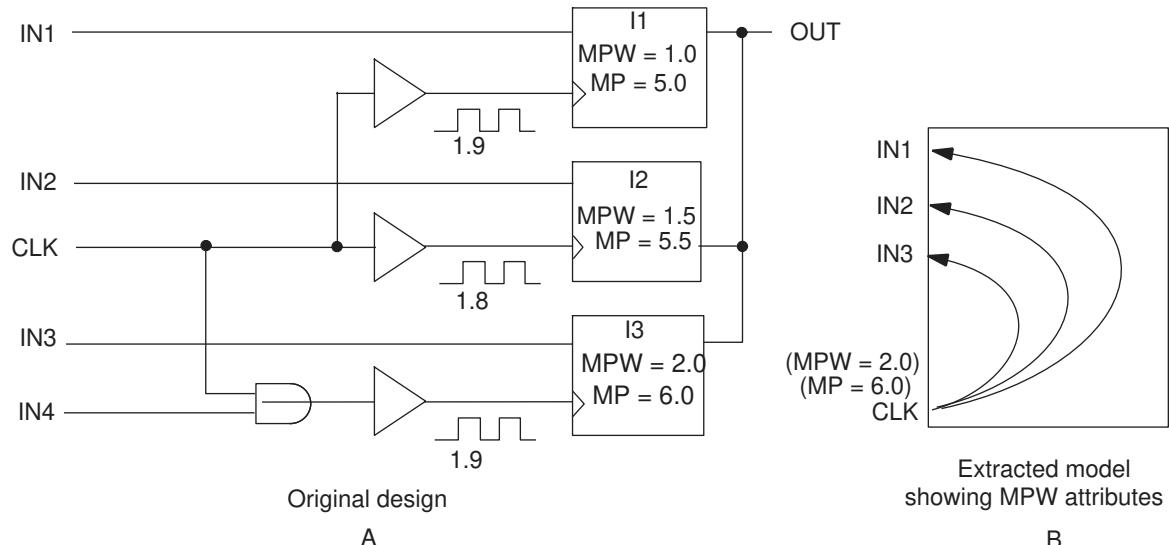
The timing of the extracted model matches that of the original netlist if the specified borrowing behavior at the time of model generation remains valid for the arrival times defined on input ports and clocks when the model is used. The borrowing status (borrowing or not borrowing) must match, although the actual amount of time borrowed can be different.

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 attributes on the clock pins in the extracted model, as shown in the following figure.

Figure 433 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 delay effects (asymmetrical rise or fall) and slew propagation along the path are not considered. 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 433 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](#).

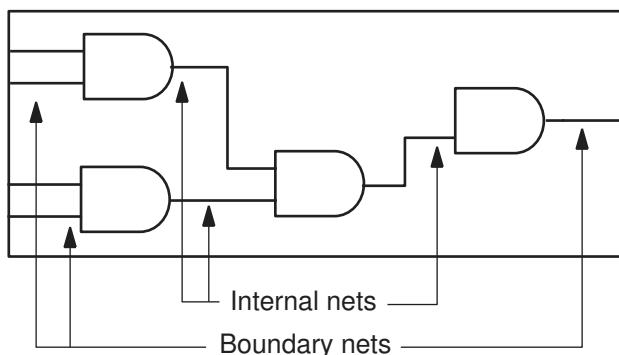
Back-Annotated Delays

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 delays of boundary cells are back-annotated, the delays of the generated model are constant. For example, the delays of paths from inputs are the same for different input signal transition times. Similarly, the delays of paths to outputs are not affected by the input transition time and output load.

The example shown in the following figure demonstrates how the tool handles extraction of back-annotated delays on internal nets and boundary nets. Both internal net delays and boundary net delays are included in the extracted arcs of the model.

Figure 434 Boundary nets and internal nets



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.

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 the tool to ignore the boundary parasitics for timing model extraction, including both detailed and lumped RC information.

Model extraction 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 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 only for this transition time.

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 time at which the SDF is generated. To create some dependence on transition time and capacitive load, remove the annotations on boundary cells by using the `remove_annotated_delay` and `remove_annotated_check` commands before you extract the model.

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 perform hierarchical analysis with crosstalk between modules and between different levels of hierarchy, use HyperScale 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.

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, it

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.

Other Extracted Information

In addition to the timing paths, the model extraction algorithm extracts other types of information:

Modes

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.

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 ports and the maximum capacitance at output ports.

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

The extraction process adjusts timing arc values to reflect the worst-case effect of the `set_multicycle_path` command among all paths between each port pair. Note that this adjustment can result in negative delay arcs.

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 [Variables That Affect ETM Generation](#).

Clock latency arcs

Clock latency arcs are extracted if enabled (see [Variables That Affect ETM Generation](#)). Clock latency arcs are used by other tools to compensate and balance clock tree skews at chip-level. They are also reported by the `report_clock_timing` command in PrimeTime.

User-defined attributes

User-defined attributes defined with the `-export` option of the `define_user_attributes` command are included in the extracted model.

Model Extraction Capabilities

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.

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
extract_model_with_non_sequential_arcs
timing_clock_gating_propagate_enable
timing_disable_clock_gating_checks
timing_enable_preset_clear_arcs
```

Minimum and maximum delay constraints

An explicit minimum or maximum delay constraint set on a timing path is ignored by the extraction process. If you set such a constraint on a combinational cell, the extraction process ignores the timing paths through the cell.

Input and output delays

Model extraction ignores input and output delays set on design ports unless the design contains transparent latches.

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. 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.

Data checks

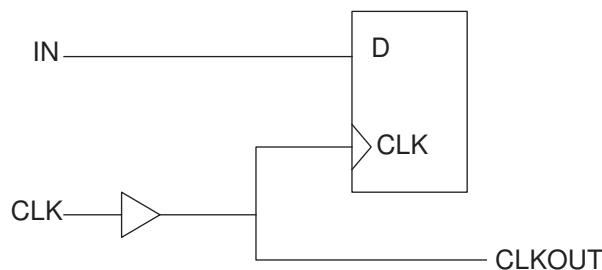
Data checks (both from the `set_data_check` command and from library-level nonsequential check arcs) are extracted to nonsequential check arcs in the model when the `extract_model_with_non_sequential_arcs` application variable is set to `true`. Both pins of the check must be in the interface logic.

During model validation, nonsequential paths and arcs are included in setup/hold timing validation.

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 435](#). 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.

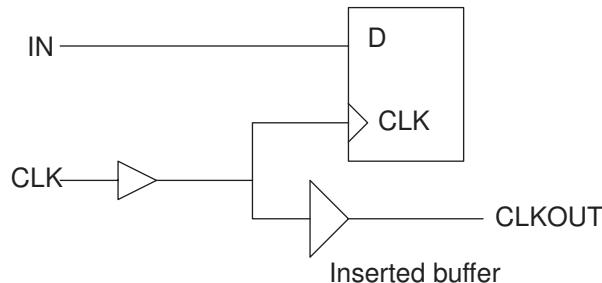
Figure 435 Load-dependent clock network



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 436](#).

Figure 436 Isolating the delay of the clock network



ETM Usage in Other tools

The .lib format is useful for exporting the timing model to external tools. Note that the .lib model captures only the timing behavior, not the logic, of the original netlist.

The Design Compiler tool does not support all the features of extracted timing models, such as mode information. All modes are considered enabled.

Model Extraction Options

You can control the characteristics of the extracted model with the following options:

- [Extracting Models With CCS Data](#)
- [Controlling the Extracted Table Sizes](#)
- [Extracting Clock Latency Arcs](#)
- [Extracting Constants and Case Values](#)
- [Extraction of UPF and PG Data](#)
- [Specifying Model Indexes](#)
- [Specifying Margins for Delay and Constraint Arcs](#)
- [Tapped Output Ports](#)

- [Timing Exceptions](#)
- [Generated Clocks](#)
- [Extracting Internal Pins](#)
- [Multiple Clocks on an Internal Pin](#)
- [Different Operating Corners](#)
- [Extracting Clock-Gating Checks](#)
- [Restricting the Types of Arcs Extracted](#)

Extracting Models With CCS Data

By default, model extraction creates a model with only NLDM delay data, which reduces the extraction runtime and generates a smaller .lib file.

To generate an extracted timing model (ETM) with Composite Current Source (CCS) data,

- CCS model extraction must be enabled:

```
pt_shell> set_app_var extract_model_with_ccs_timing true
```

- Advanced waveform propagation must be enabled:

```
pt_shell> set_app_var delay_calc_waveform_analysis_mode full_design
```

This requires libraries with CCS timing and CCS noise data. See [Waveform Propagation](#) for details.

The resulting ETM contains both CCS timing and noise modeling data, which is needed to use advanced waveform propagation analysis at the higher level of hierarchy where the model is used.

CCS Models and Min/Max OCV Voltages

For multivoltage designs that use on-chip-variation (OCV) min/max voltages, there are additional considerations.

When Library Compiler compiles CCS delay and noise data, it checks for full-rail DC swing responses against the supply voltage, and it reports overshoot or undershoot responses beyond $\pm 5\%$ as violations.

By default, model extraction writes the highest (min-delay) voltage to the model as the supply voltage (defined by the `voltage_map` construct):

```
set_voltage -object_list VDD 0.97 -min 1.03
extract_model ...
```

```
library("myblock") {
    voltage_map("VDD", 1.03);
    pin("A") {
        related_power_pin : "VDD";
        /* max-delay 0.97v CCS timing */
        /* min-delay 1.03v CCS timing */
        /* min-delay 1.03v CCS noise */
    }
}
```

However, this can cause max-delay CCS data, extracted at lower voltages, to issue undershoot violations:

Warning: Line 730, Cell 'top', pin 'A', CCS/CCB noise output DC swing (0, 0.97) is not within 5.0% of rail voltages (0, 1.03) (LBDB-953)

To remedy this, use the `-nominal` option of the `set_voltage` command to define an intermediate voltage within 5% of both the min and max OCV voltages, then set the `extract_model_ccs_keep_voltage_corner` variable to `nominal` to write this nominal value as the supply voltage:

```
set_voltage -object_list VDD 0.97 -min 1.03 -nominal 1.00
set_app_var extract_model_ccs_keep_voltage_corner nominal
extract_model ...
```

```
library("myblock") {
    voltage_map("VDD", 1.00);
    pin("A") {
        related_power_pin : "VDD";
        /* max-delay 0.97v CCS timing */
        /* max-delay 0.97v CCS noise */
        /* min-delay 1.03v CCS timing */
        /* min-delay 1.03v CCS noise */
    }
}
```

Note the following:

- The `set_voltage -nominal` value is used only when writing an ETM model; it is not used for design analysis.
- For timing data, model extraction always writes both min-delay and max-delay data to ensure a bounding analysis.
- For noise data, model extraction writes the noise data corresponding to the supply voltage. When the variable is set to `nominal`, both min-condition and max-condition noise data is written.
- When the variable is set to `nominal`, the min-condition and max-condition voltages must both be within 5% of the nominal voltage. Otherwise, model extraction stops with a MEXT-102 error.

The `extract_model_ccs_keep_voltage_corner` variable can also be set to `min` (the default) and `max`. For details, see the man page.

Controlling the Extracted Table Sizes

To control the quality of the extraction, you can specify the number of points in the nonlinear delay model (NLDM) lookup tables by setting these variables.

Variable	Description
<code>extract_model_num_capacitance_points</code>	Specifies the number of load indexes in the delay tables; the default is 5.
<code>extract_model_num_clock_transition_points</code>	Specifies the number of clock transition indexes in constraint tables; the default is 5.
<code>extract_model_num_data_transition_points</code>	Specifies the number of data input transition indexes in the constraint tables and delay tables; the default is 5.

For the best tradeoff between accuracy and performance, specify the same sizes of the extracted lookup table as the lookup tables of the initial library. If you do not know this information, the recommended table size for good accuracy is 7x7, which you set with the following commands:

```
set_app_var extract_model_num_capacitance_points 7
set_app_var extract_model_num_data_transition_points 7
```

The following variables determine the maximum index value in the tables.

Variable	Default
<code>extract_model_capacitance_limit</code>	64
<code>extract_model_clock_transition_limit</code>	5
<code>extract_model_data_transition_limit</code>	5

You must set these global limits to values consistent with the design rules.

The preceding variables indirectly influence how the ETM selects transition and load values for timing extraction, but they do not directly specify the exact values in the table indexes. To understand how these variable control the number of data input transition indexes in the constraint tables and delay tables, see [SolvNetPlus article 000012824, "How extract_model Determines Slew and Capacitance Index Ranges"](#).

Extracting Clock Latency Arcs

By default, the `extract_model_with_clock_latency_arcs` variable is `false`. If you set this variable to `true`, the `extract_model` command traverses all clock tree paths leading

to both interface and internal register cells, computes the insertion delay of the paths, and creates clock latency arcs in the model.

The most important usage of the clock latency arcs are in the clock tree synthesis (CTS) step in design implementation tools to compensate and balance clock tree skews at the chip level. PrimeTime can also report these clock latency arcs with the `report_clock_timing` command, but these arcs do not affect the setup and hold constraint arcs.

Extracting Constants and Case Values

In general, an ETM describes only the timing for the block interface and contains no functional information. If the `extract_model_write_case_values_to_constraint_file` variable is set to `false` (the default), all logic constant values defined at or propagated to block I/O ports are written into the `.lib` and `.db` files as the function attribute for the pin per Liberty syntax. This occurs regardless of whether the logic value is due to propagation of circuit constants (tie-high, tie-low, etc.) or user-defined case analysis values.

However, some downstream tools that use the ETM might treat the function attribute on the library pin as circuit constants and differently from the case values specified in SDC constraints; this could cause undesired behavior such as logic synthesis. To avoid this, set the `extract_model_write_case_values_to_constraint_file` variable to `true`; this setting causes the following tool behaviors:

- Model extraction differentiates the logic values set or propagated to the I/O boundary of the block as a result of user-defined analysis settings.
- Logic values are written into the ETM constraint file with the proper `set_case_analysis` command.
- Circuit constants are still written in the library as function attribute of the pin.

Extraction of UPF and PG Data

By default, an ETM extracts only block timing information in the library and does not include any power information. This is true even when there is UPF data defined with the timing constraints for netlist timing analysis.

In a multivoltage environment, you can include the UPF information such as rail supply, power domains, isolation information, and PG pin associations with signal pins in the ETM by setting the `extract_model_include_upf_data` variable to `true`. With this variable setting, the `merge_models` command also checks the consistency of library cell PG pin attributes to merge them.

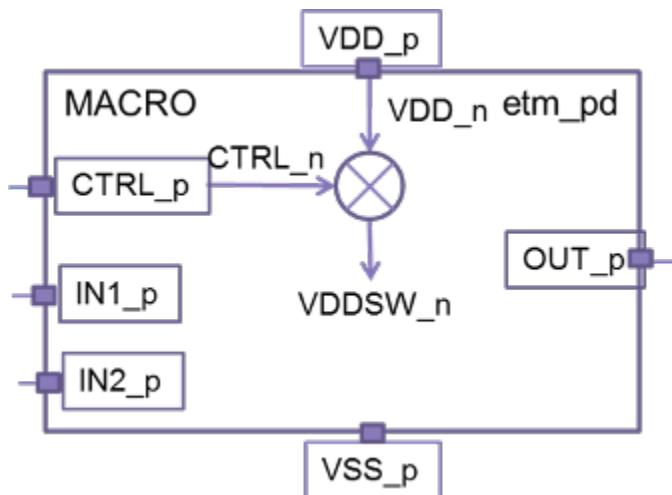
PG Data in an ETM

PrimeTime captures the PG information such as power domain, supply port, supply net, or supply set definition and their connection and power switch as the following Liberty attributes in the ETM:

- pg_pin (including direction, pg_type, voltage_name)
- voltage_map
- related_power_pin or related_ground_pin attributes on signal pins

The following examples use this design to show how UPF information is captured in the ETM.

Figure 437 Simple multivoltage test design



Non-Internal PG Pins in an ETM

Supply nets connected to a top-level supply port are extracted as non-internal PG pins. These are always extracted and are named by the supply port name. The following example shows a supply port definition in the UPF and the corresponding definition in the ETM.

Supply port definition in the UPF	Supply port definition in the ETM
<pre> create_power_domain etm_pd create_supply_port VDD_p create_supply_net VDD_n \ -domain etm_pd connect_supply_net VDD_n \ -ports {VDD_p} </pre>	<pre> pg_pin("VDD_p") { voltage_name : "VDD_p"; pg_type: primary_power; } </pre>

Internal PG Pins in an ETM

These are supply nets that are not connected to a top-level supply port. These are named by the supply net name. Only those supply nets referred on the interface are extracted in the ETM. The following example shows a power switch definition in the UPF and the corresponding definition in the ETM. Switches are only partially supported by PrimeTime. Therefore, it does not extract `pg_function` and `switch_function` automatically. You need to manually specify these Liberty attributes in the generated ETM.

Power switch definition in the UPF	Power switch definition in the ETM
<pre>create_power_switch sw1 \ -domain etm_pd \ -output_supply_port{vout VDDSW_n} \ -input_supply_port {vin VDD_n} \ -control_port {ctrl_small CTRL_n} \ -on_state {full_s vin {!ctrl_small}}</pre>	<pre>pg_pin("VDDSW_n") { voltage_name : "VDDSW_n"; pg_type : internal_power; direction : internal ; * pg_function:"VDD_p"; * switch_function : "!CTRL_p"; }</pre> <p>(* indicates unsupported)</p>

The voltage_map Statement in an ETM

The `voltage_map` statement defines the default voltage values for the cell rail voltage, which can be overridden by `set_voltage` or the operating condition.

set_voltage definition in the UPF	voltage_map statement in the ETM
<pre># Design operating condition name set_operating_conditions WC09 # Set voltages on VDD_n and # VDDSW_n supply nets set_voltage 1.4 -object_list VDD_n set_voltage 0.7 -object_list VDDSW</pre>	<pre>voltage_map ("VDD_p", 1.4); voltage_map ("VDDSW_n", 0.7); voltage_map ("VSS_p", 0.0);</pre>

The related_power_pin and related_ground_pin Attributes in an ETM

An ETM represents the physical connection in the actually implemented block netlist. This applies to both timing data and power data. The `set_port_attributes` and `set_related_supply_net` commands are used to associate supply nets with design ports. The tool uses this information for constraining and checking the design. This supply net information is also used in level shifter insertion by synthesis and design optimization tools, which consume the ETM at top level. It determines if level shifter is required between a port and the logic connected to the port.

Using `set_port_attributes -driver_supply` (for input ports) or `set_port_attributes -receiver_supply` (for output ports) and `set_related_supply_net` specifies the expected supplies that are intended to eventually

power the immediate logic outside of the block. From the block-level implementation point of view, these supplies are not part of the physical design and therefore should not have any effect on model extraction.

The `set_port_attributes -receiver_supply` (for input ports) and `set_port_attributes -driver_supply` (for output ports) commands specify the expected supply that intended to eventually power the inside logic connected to the ports.

If you do not use the `set_port_attributes` and `set_related_supply_net` commands, the tool assumes that ports are internally connected to the supply net of the first leaf level pin in the design.

```
pin(CK) {
    related_power_pin : "VDD_p" ;
    related_ground_pin : "VSS_p" ;
    direction : input ;
    ...
}
```

In real designs, there can be situations where the netlist logic does not have sufficient power information and manual overriding is needed. For example:

- Unconnected port
- Feedthrough wire
- Multidrive or multiload
- Implementation still in progress

Therefore you might want to control or change the `related_power_pin` or `related_ground_pin` attribute in ETMs, in your flows. The `extract_model_upf_supply_precedence` variable controls the precedence between `set_port_attributes -receiver_supply` or `-driver_supply` for input/output ports, `set_related_supply_net`, and netlist logic in determining the `related_power_pin` or `related_ground_pin` attribute of a port.

You can set the `extract_model_upf_supply_precedence` variable to one of the following values:

- `netlist` - This is the default mode in which the `extract_model` command considers only the actual implemented netlist connectivity. The `set_port_attributes` command, `set_related_supply_net` command, and primary supply net are not used.
- `internal` - In this mode, the `set_port_attributes -receiver_supply` (for input ports) and `set_port_attributes -driver_supply` (for output ports) commands override the netlist logic. One possible use of this mode is when the block implementation is still in progress, and you want to extract an ETM that simulates the eventual power connectivity at early stage before level-shifter and isolation cells are actually inserted.

- external - In this mode, the `set_port_attributes -driver_supply` (for input ports) or `set_port_attributes -receiver_supply` (for output ports) or `set_related_supply_net` commands override the netlist logic. This is not the recommended setting as it goes against the essence of modeling. However, you might find it useful if you do not want to change the existing UPF constraints or if you have a custom flow based on `set_related_supply_net`.

The following table summarizes the order of precedence determined by the `extract_model_upf_supply_precedence` variable:

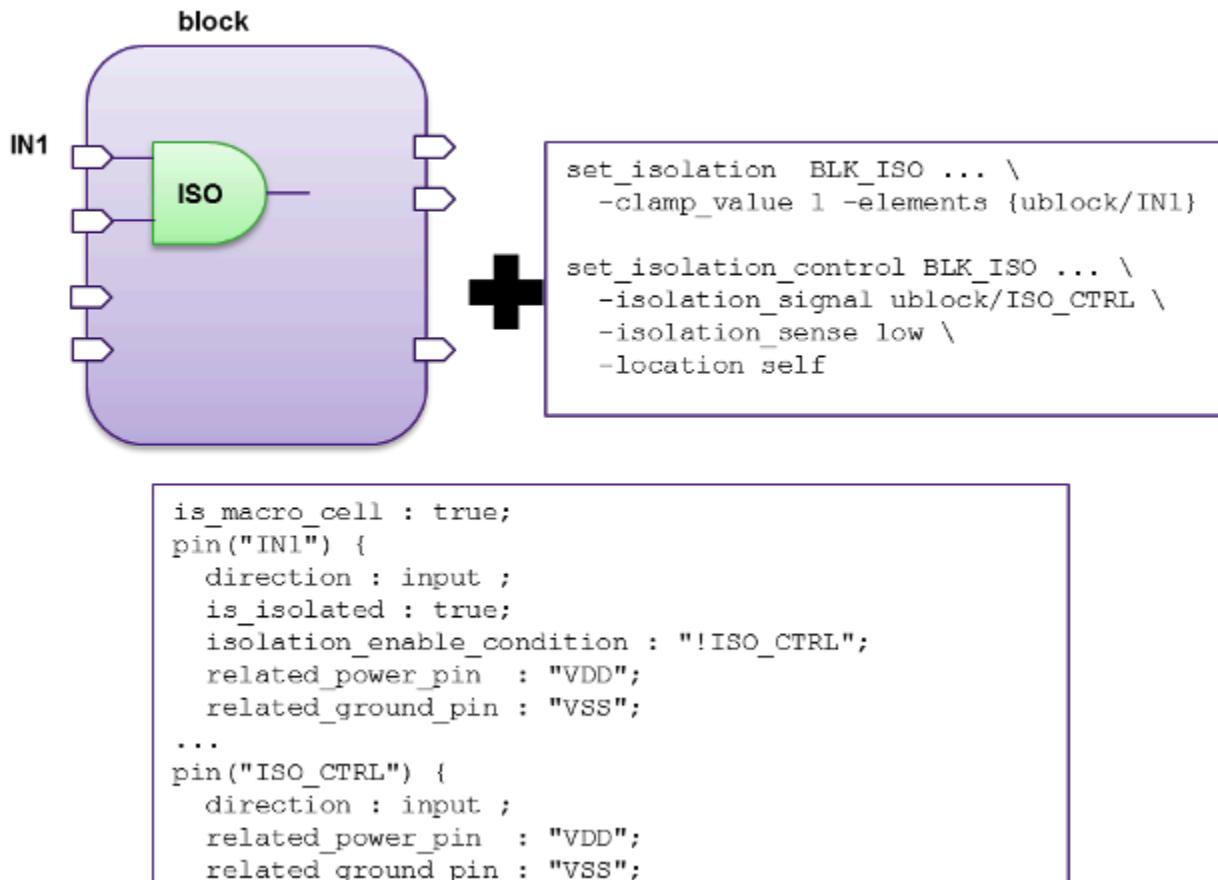
Value	Input port	Output port
netlist	<ul style="list-style-type: none"> • Netlist 	<ul style="list-style-type: none"> • Netlist
internal	<ul style="list-style-type: none"> • <code>set_port_attributes -receiver_supply</code> • Netlist • <code>primary_power</code> and <code>ground_net</code> of the top power domain 	<ul style="list-style-type: none"> • <code>set_port_attributes -driver_supply</code> • Netlist • <code>primary_power</code> and <code>ground_net</code> of the top power domain
external	<ul style="list-style-type: none"> • <code>set_port_attributes -driver_supply</code> • <code>set_related_supply_net</code> • Netlist • <code>primary_power</code> and <code>ground_net</code> of the top power domain 	<ul style="list-style-type: none"> • <code>set_port_attributes -receiver_supply</code> • <code>set_related_supply_net</code> • Netlist • <code>primary_power</code> and <code>ground_net</code> of the top power domain

Isolation Attributes in ETM

PrimeTime extracts UPF isolation related information also into the ETM. This is derived from the `set_isolation` or `set_isolation_control` UPF commands set on the block. The `is_isolated` and `isolation_enable_condition` Liberty attributes indicate the pins that are isolated from the internal logic in the ETM.

The `is_isolated` attribute on an ETM pin indicates usage of isolation cell within the macro to isolate the specified pin from the macro's internal logic. Therefore, upstream tools use the driver supply while buffering the path to the pin. Logic connected to an ETM pin with `isolation_enable_condition` is treated as always-on path. Also, note that the `isolation_enable_condition` is extracted only when the `isolation_signal` specified using `set_isolation_control` is a port and not an internal pin.

Figure 438 Isolation attributes



Retention Registers

No special modeling required for designs with retention registers. The user-specified UPF for the ETM model captures the related supply net of the ETM ports that drive retention save and restore control pins.

Level Shifters

Extracted timing model (ETM) generation handles level shifter cells on boundary paths if you enable UPF extraction by setting the `extract_model_include_upf_data` variable true.

The `extract_model` command models a level shifter with the `input_signal_level` and `input_voltage_range` UPF pin attributes. For example,

```
pin(A) {
    direction : input;
    input_signal_level : "VDD1";
```

```

related_power_pin : "VDD";
related_ground_pin : "VSS";
input_voltage_range (1.1,1.3);
}

```

and similarly for an output pin:

```

pin(out1) {
    direction : output;
    output_signal_level : "VDD1";
    related_power_pin : "VDD";
    related_ground_pin : "VSS";
    output_voltage_range (1.1,1.3);
}

```

Limitations of PG Pin ETM Creation and Merging

ETM creation and merging has the following limitations:

- If a PG-pin Verilog netlist is used, then
 - The `set_port_attributes` command must be used to describe the supplies associated with all ports.
 - The `extract_model_upf_supply_precedence` variable must be set to `external`.
 For more on PG-pin Verilog netlists, see [Netlist-Inferred Supply Connectivity](#).
- For designs with implemented switch cells, `extract_model` captures the switched supply in the generated ETM. But other information such as the PG function and switch function are not captured automatically in the ETM .lib. For an example, see [Internal PG Pins in an ETM](#).

The following attributes are not extracted into the ETM.lib; you must add them manually:

- `pg_function`
- `switch_function`
- `power_down_function`
- `switch_cell_type (fine_grain or coarse_grain)`
- `switch_pin`

Specifying Model Indexes

Extracted timing model (ETM) generation automatically determines the index values for which delay and slew tables are created in the corresponding ETM library files.

To explicitly set the index values at which the libraries tables are generated, run the `set_extract_model_indexes` command.

For example:

```
pt_shell> set_extract_model_indexes -type capacitance \
    {0.11 0.12 0.15 0.18 0.20} \
    -ports [get_ports asynctest_out[1]]
```

The preceding command results in this ETM:

```
pin("asynctest_out[1]") {
    direction : output ;
    ...
    original_pin : dp_ipo540/z;
    timing () {
        related_pin : "clk_d17_ph0" ;
        timing_type : rising_edge ;
        cell_rise( f_itrans_ocap ) {
            index_1 ("0.000000, 0.026049, 0.042800, 0.084400, 0.333900");
            index_2 ("0.110000, 0.120000, 0.150000, 0.180000, 0.200000");
            values ("1.035580, 1.037688, 1.044013, 1.050345, 1.054575", \
                    "1.047846, 1.049955, 1.056279, 1.062611, 1.066841", \
                    "1.054575, 1.056683, 1.063007, 1.069340, 1.073570", \
                    "1.069329, 1.071437, 1.077761, 1.084093, 1.088323", \
                    "1.131341, 1.133449, 1.139773, 1.146106, 1.150335");
        ...
    }
}
```

To remove user-specified index values, run the `reset_extract_model_indexes` command.

Specifying Margins for Delay and Constraint Arcs

To specify a fixed margin to be added to the delay and constraint values during extracted timing model (ETM) generation, run the `set_extract_model_margin` command.

For example:

```
pt_shell> set_extract_model_margin -max 10 -ports [get_ports *]
```

The preceding command results in the following change to the ETM.

Margin value	Resulting ETM
Without user-specified margin	<pre>original_pin : icc_place2129/zN; ... cell_rise(f_itrans_ocap){ index_1 ("0.003800, 0.082400, 0.166200, 0.333900, 1.000000"); index_2 ("0.000450, 0.014980, 0.031580, 0.096976, 0.120100"); values ("1.131896, 1.156789, 1.178753, 1.258480, 1.286219", \ "1.163713, 1.188606, 1.210570, 1.290296, 1.318035", \ ... }</pre>

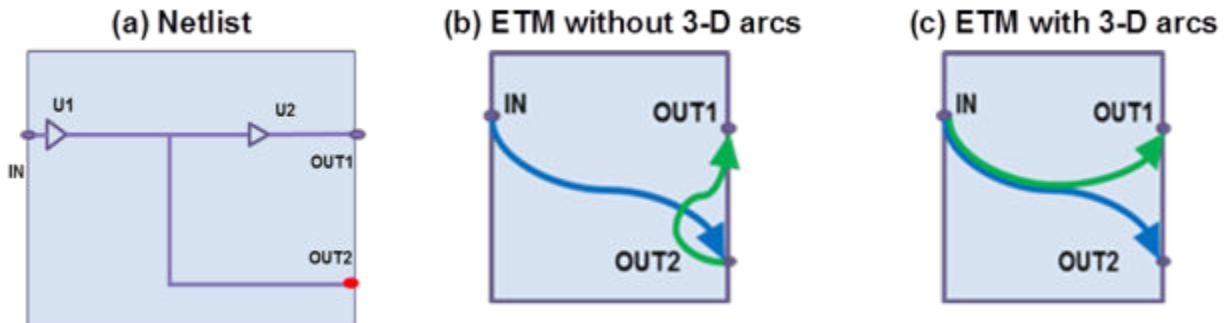
Margin value	Resulting ETM
With user-specified margin of 10	<pre>original_pin : icc_place2129/zN; ... cell_rise(f_itrans_ocap){ index_1 ("0.003800, 0.082400, 0.166200, 0.333900, 1.000000"); index_2 ("0.000450, 0.014980, 0.031580, 0.096976, 0.120100"); values ("11.131897, 11.156789, 11.178753, 11.258480, 11.286219", \ "11.163713, 11.188605, 11.210569, 11.290297, 11.318035", \ ... }</pre>

Tapped Output Ports

A *tapped output* is an output or inout port that is connected to both driver pins and receiver pins.

In the following figure, the OUT2 port is a tapped output port. The load capacitance of OUT2 affects the delay from IN to OUT1. Having such unbuffered ports in the netlist is a poor design technique. PrimeTime issues an MEXT-75 warning message for tapped output ports.

Figure 439 ETM extraction on a netlist with tapped output ports



For NLDM model extraction, by default the tool creates a 3-D arc from IN to OUT1 with OUT2 as the `related_output_load` pin. If downstream tools do not support 3-D arcs, set the `extract_model_with_3d_arcs` variable to `false` to extract a model with 2-D arcs instead.

For CCS model extraction, tapped output ports are always represented by 2-D arcs because CCS does not support 3-D arcs.

Arcs to tapped output ports are automatically kept if they are essential in modeling other ports.

For robust hierarchical modeling, all ports in the design should be buffered so you can isolate the timing of the internal arcs from external loading and model the block more accurately in the ETM.

See Also

- [Extracting Models With CCS Data](#) for details on how to choose between NLDM and CCS model extraction

Timing Exceptions

If you apply timing exceptions such as false paths or multicycle paths to a block, model extraction does the following:

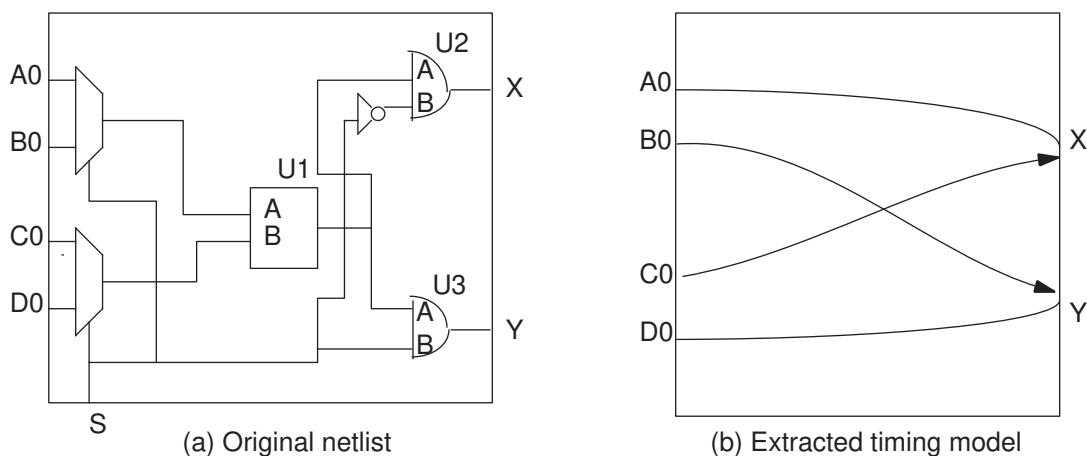
- Takes the exceptions into account if they affect paths on the interface
- Ignores certain paths due to false path exceptions
- Shifts the arcs with respect to the clock for multicycle paths, etc.

If you apply both timing exceptions and mode analysis to a block, the following rules apply to model extraction:

- 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.

For example, [Figure 440](#) shows an original design and an extracted timing model when false paths are present.

Figure 440 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.

If the original design has multicycle paths, the model extractor adjusts the setup, hold, and delay values for the timing arcs to correctly model the multicycle paths.

An ETM is a greatly reduced timing abstraction of the block with timing arcs in a library cell; therefore, it is strongly recommended that you avoid defining complex and detailed exceptions on the block interface with fine grain modifications to path timing behaviors.

It is also recommended that if the exceptions only refer to interface clocks or ports, you do not need to specify them before timing extraction; instead, have the extraction process generate all necessary timing arcs for the interface, then apply these clock-specific or port-specific constraints when the ETM is instantiated at the top level.

Generated Clocks

Generated clocks can be extracted in the ETM and visible at top level analysis when the ETM is instantiated as macro cells. When the generated clocks embedded in the ETM library cell are instantiated at the top level by PrimeTime, use the following naming convention for the generated clocks in the ETM:

instance_name/generated_clock_name

This naming convention should be used in the top-level constraint file to reference generated clocks.

The ETM keeps only the following generated clocks that are relevant to the interface registers:

- Generated clocks that capture signals coming into the block input ports
- Generated clocks that launch signals out through the block output ports
- Feedthrough clock paths that exit the block

The generated clock source network from the master clock source pin to the generated clock source pin is extracted as a set of delay arcs in the ETM. For cases of cascaded generated clocks, for example, a chain of frequency subdivision clocks from the same master clock, it is strongly recommended that the generated clock definitions follow a cascading topology, particularly when the intermediate generated clocks block the original master from directly reaching downstream sources. For example, you can use a simple

cascading of two generated clocks div-by-2 and then div-by-4 from the same master clock, as shown in the following example:

```
create_clock -name MCLK [get_ports MCLKIN] ...
create_generated_clock -name MCLK_div2 -master_clock MCLK -add \
    -divide_by 2 -source MCLKIN [get_pins reg1/Q] ...
create_generated_clock -name MCLK_div4 -master_clock MCLK_div2 -add \
    -divide_by 2 -source reg1/Q [get_pins reg2/Q] ...
```

instead of having both generated clocks referring to the original master MCLK.

PrimeTime cannot perform valid timing analysis for the paths launched or captured by generated clocks whose waveforms or source network latencies cannot be accurately computed. There are multiple reasons for unexpanded generated clocks or untraceable source networks (often resulting in error or warning messages during update timing, such as UITE-461, PTE-075, PTE-021, PTE-022, PTE-023, PTE-025, PTE-103).

These are the typical causes:

- No physical paths exist between a generated clock and its master
- Edges or senses are incorrect given the circuit between a generated clock and its master.
- An ambiguous source network exists between a generated clock and its master, for example, paths that form loops.

If these issues occur for the block interface timing paths, the extracted ETM is incorrect. Therefore, you must fix these issues before model extraction.

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.

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 the clock name, possibly appended with _1, _2, or something similar.

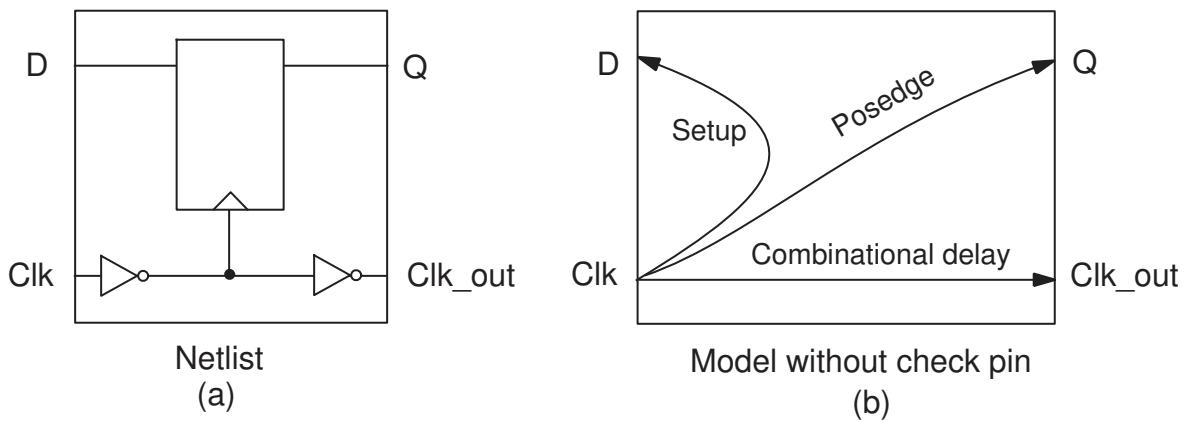
Generated Clocks

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 the generated clock name, possibly appended with _1, _2, or something similar.

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 441](#) shows what the extracted model would look like in the absence of check pins.

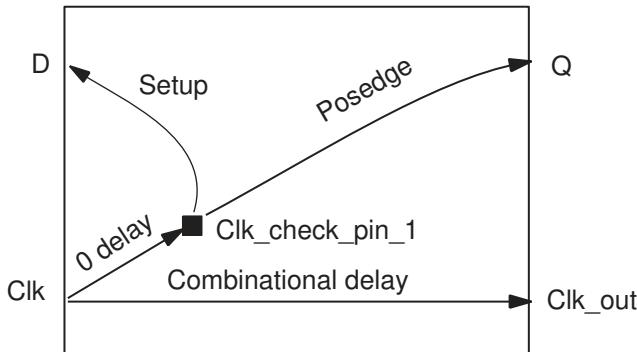
Figure 441 Original circuit and extracted model without check pins



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 441](#), the model extractor creates an extra internal pin in the extracted model as shown in [Figure 442](#). 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 442 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 `extract_model` command creates an internal check pin to handle different paths through the bidirectional pin.

It alters the arcs as follows:

- Creates a zero-delay arc from the INOUT pin to the check pin.
- Moves the constraint arcs that formerly ended at the INOUT pin so that they end at the check pin.

Multiple Clocks on an Internal Pin

A design block often has many different modes of operation that are specified by the set of constraints applied to it. An ETM is generated for a specific mode of constraints, multiple models are generated each for a different mode and optionally merged into one single model with timing arcs sensitized by the specific modes associated them. It is

recommended that you generate separate ETMs for different modes for timing accuracy and flow complexity considerations.

However, to reduce the number of block-level timing runs, you can merge different modes into one set of constraints to perform a single combined timing analysis, therefore generating one ETM covering all the modes. When this happens, often there could be multiple clocks defined on a same source pin or port.

The `extract_model` command handles multiple clocks on the same source pin or port. Multiple generated clocks or primary clocks defined on a source pin inside the block or on a block port result in multiple, unique internal pins and multiple, unique generated clock group definitions in the ETM library. These internal pin definitions are zero delay arcs between the original port/pin, where the clocks were defined to the unqualified pins. The delay and constraint arcs relevant to different clocks are attached to unique internal pins.

Multiple clocks defined on the same block port are more complex because the ETM cannot define multiple pins on the macro boundary. For such cases, there are multiple internal pins created (one for each clock at the port) and zero delay arcs from the clock port to these internal pins. Then there are either `set_sense` or `set_clock_sense` commands to program the clocks propagating through these arcs to separate them.

Different Operating Corners

Similar to different modes of the block, you need to create one ETM model for each corner and use the ETMs for proper matching top-level corner. A corner is usually defined by a specific process corner for the device libraries and parasitics extraction, plus the specific operating conditions such as voltage and temperature.

However, there is no support to merge ETMs from different corners into a single ETM. Instead, either use a different ETM library for different top-level corner analysis, or use the `define_scaling_lib_group` command in PrimeTime to map the different corner ETMs for the proper top-level analysis.

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

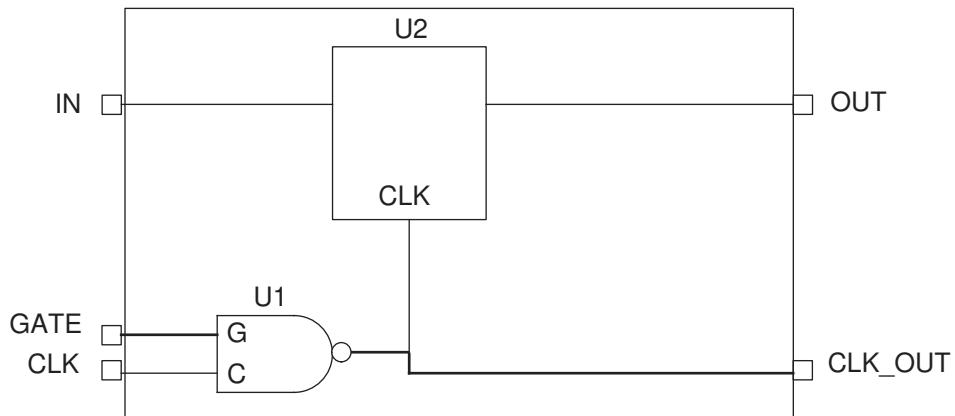
Extraction of clock-gating checks can be enabled or disabled with the `timing_disable_clock_gating_checks` variable before extraction.

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 443](#), 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`.

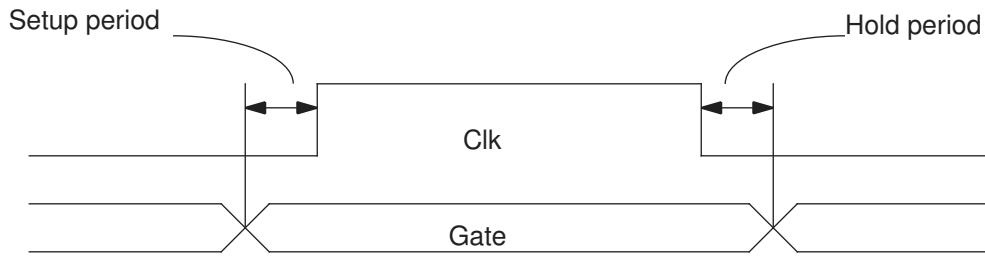
Figure 443 Combinational path through clock-gating logic



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 444](#)).

Figure 444 No-change arc



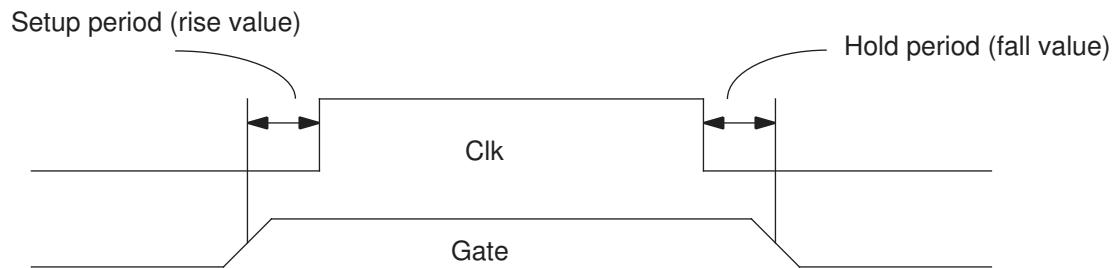
A clock-gating setup check can be combined with its corresponding clock-gating hold check to produce two no-change arcs. One arc is the no-change condition with the clock-gating signal low during the clock pulse, the other is the gating signal high during the clock pulse.

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 445](#).

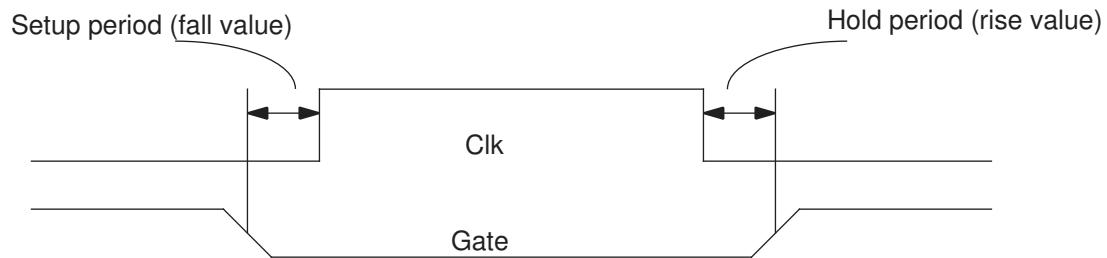
Figure 445 NOCHANGE_HIGH_HIGH arc



- NOCHANGE_LOW_HIGH

Rise table taken from the hold arc rise table. Fall table taken from the setup arc fall table. See [Figure 446](#).

Figure 446 NOCHANGE_LOW_HIGH arc



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

Merging Extracted Models

The following topics describe how to merge multiple ETMs together across operating modes and voltage corners:

- [Model Merging Overview](#)
- [Merging Multiple Modes Into a Mode Group](#)
- [Merging Multiple Modes Into Multiple Mode Groups](#)
- [Merging Models Across Block-Internal Power Supplies](#)
- [Controlling the Model Merging Process](#)

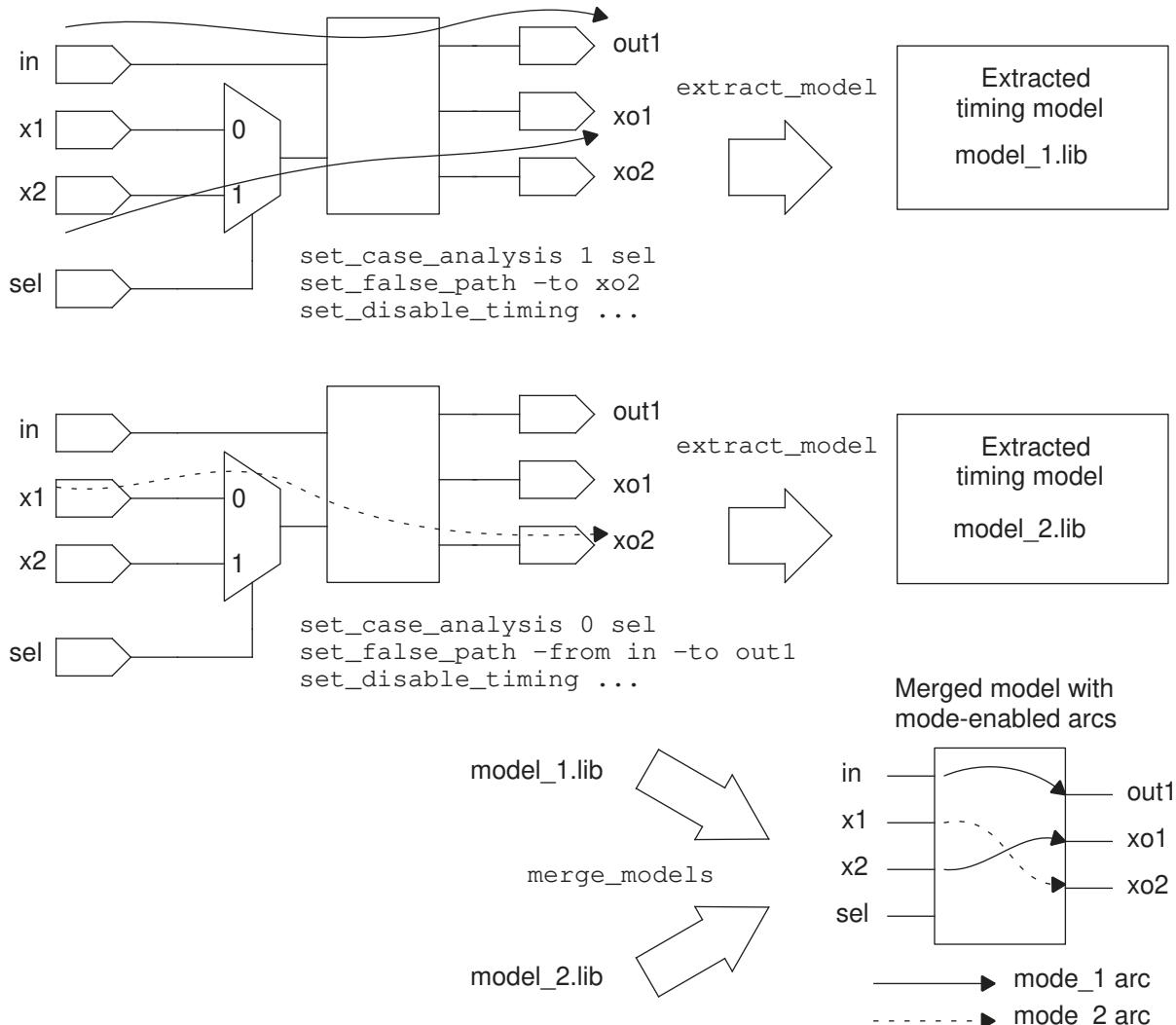
Model Merging Overview

A design can have multiple operating modes, such as functional modes and test modes, each with their own constraints and timing characteristics. However, the `extract_model` command creates an extracted timing model (ETM) for a single operating mode at a time.

You can use the `merge_models` command to merge multiple single-mode ETM .lib files into a single multiple-mode ETM .lib file. The resulting merged ETM can then be set to its available modes by using the `set_mode` command in the PrimeTime tool.

[Figure 447](#) shows an example of a design from which two timing models are extracted, then the two extracted models are merged into a single timing model having two operating modes.

Figure 447 Model extraction and merging example



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.

Note that the `merge_models` command is not a general-purpose Liberty file merger. It supports only the subset of Liberty syntax used in extracted timing models generated by the `extract_model` command in the PrimeTime tool.

Here are some points to consider for model merging:

- To be retained in the merged model, case values reaching output ports 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.

Merging Multiple Modes Into a Mode Group

To create a merged model that merges multiple modes into a single mode group,

- Use the `-group_names` option to specify a single mode group name.
- Use the `-mode_names` option to specify the list of mode names.
- Use the `-lib_files` option to specify the input .lib model files, exactly in the same number and order given in the `-mode_names` option.

For example,

```
# create a merged model with "func" and "idle" modes
merge_models \
    -group_names {OP} \
    -mode_names {func idle} \
    -lib_files {OP_func.lib OP_idle.lib} \
    -output merged_model -format {db lib}
```

The resulting merged model contains mode-specific arcs where the model behaviors differ:

```
library(...) {
    ...
    mode_definition ( OP ) {
        mode_value( func ) {}
        mode_value (idle ) {}
    }
    ...
    pin() {
        timing() {
            mode(OP, "func");
            ...
        }
        timing() {
            mode(OP, "idle");
            ...
        }
    }
}
```

Merging Multiple Modes Into Multiple Mode Groups

To create a merged model with multiple mode groups,

- Use the `-group_names` option to specify the list of mode groups.
- Use the `-mode_names` option *multiple times*, once for each group in the `-group_names` list (and in the same order), to specify the modes in that group.
- Use the `-lib_files` option to specify all input .lib model files, exactly in the order given across all the `-mode_names` options.

For example,

```
# create a merged model that contains MEM and OP mode groups,
# each with two modes
merge_models \
    -group_names {MEM OP} \
    -mode_names {write read} \
    -mode_names {func idle} \
    -lib_files {MEM_write.lib MEM_read.lib \
                OP_func.lib OP_idle.lib} \
    -output merged_model -format {db lib}
```

The resulting merged model contains all specified modes and mode groups:

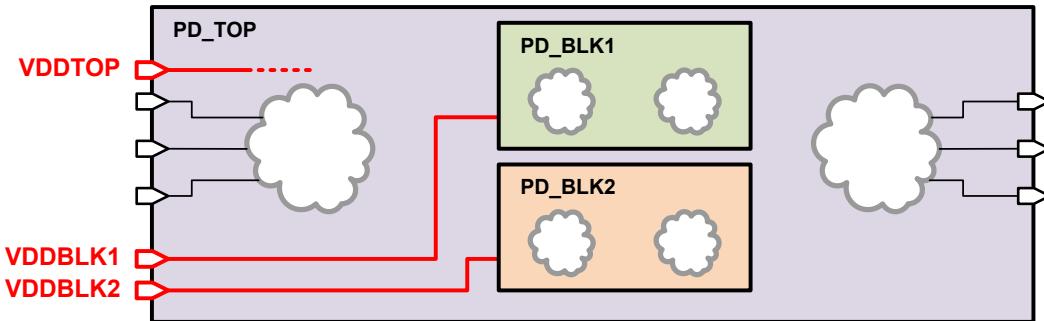
```
library(...) {
    ...
    mode_definition ( MEM ) {
        mode_value( write ) {}
        mode_value( read ) {}
    }
    mode_definition ( OP ) {
        mode_value( func ) {}
        mode_value( idle ) {}
    }
    ...
    pin() {
        timing() {
            mode(MEM, "write");
            ...
        }
        timing() {
            mode(OP, "func");
            ...
        }
    }
}
```

Merging Models Across Block-Internal Power Supplies

For multivoltage blocks with power and ground (PG) supply pins, separate ETMs are created for each combination of the PG pin supply voltages. (When the simultaneous

multivoltage analysis (SMVA) feature is used, the `extract_model` command creates these model variants automatically.)

Consider the following multivoltage design with PG supply pins VDDTOP, VDBBLK1, and VDBBLK2:



If each of the three PG pins can operate at two supply voltages, then ETM extraction is performed at eight different supply voltage combinations (2^*2^*2), resulting in eight ETMs.

When extracting models, PG supply pins can be classified as follows:

- *Boundary supply* - a PG pin that powers logic directly connected to the block boundary, and is thus electrically active at the block boundary
- *Internal supply* - a PG pin that does not power any logic directly connected to the block boundary, and is thus electrically *isolated* from the block boundary

In this example, VDDTOP is a boundary supply and VDBBLK1 and VDBBLK2 are internal supplies. Because internal supplies are electrically isolated from the block boundary, their ETM variants can be merged.

The `-merge_pg_pin` option of the `merge_models` command allows you to merge these model variants. Specify the option once for each internal-supply PG pin to be merged, along with a `high` or `low` keyword indicating whether to keep the PG pin's numerically highest or lowest voltage in the resulting model. For example,

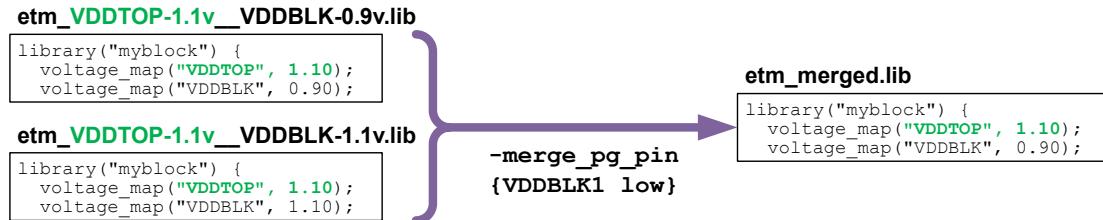
```
# merge internal-supply model variants for VDD_TOP.high
merge_models \
-lib_files [list \
    etm_VDDTOP-1.1v__VDBBLK1-1.1v__VDBBLK2-1.1v.lib \
    etm_VDDTOP-1.1v__VDBBLK1-1.1v__VDBBLK2-0.9v.lib \
    etm_VDDTOP-1.1v__VDBBLK1-0.9v__VDBBLK2-1.1v.lib \
    etm_VDDTOP-1.1v__VDBBLK1-0.9v__VDBBLK2-0.9v.lib] \
-formats lib \
-output etm_VDDTOP-1.1v \
-merge_pg_pin {VDBBLK1 high} \
-merge_pg_pin {VDBBLK2 high}

# merge internal-supply model variants for VDD_TOP.low
merge_models \
```

```
-lib_files [list \
    etm_VDDTOP-0.9v__VDDBLK1-1.1v__VDDBLK2-1.1v.lib \
    etm_VDDTOP-0.9v__VDDBLK1-1.1v__VDDBLK2-0.9v.lib \
    etm_VDDTOP-0.9v__VDDBLK1-0.9v__VDDBLK2-1.1v.lib \
    etm_VDDTOP-0.9v__VDDBLK1-0.9v__VDDBLK2-0.9v.lib] \
-formats lib \
-output etm_VDDTOP-0.9v \
-merge_pg_pin {VDDBLK1 low} \
-merge_pg_pin {VDDBLK2 low}
```

The supply voltage of a PG pin is defined by the `voltage_map` construct. Supply voltages are merged as follows:

- For boundary PG pins not being merged, the common PG pin voltage from the input models is used in the output model:



If the voltages do not match, the following error occurs:

Error: Found PG pin 'VDDTOP' in the models defined with different voltage value, cannot merge them. (MODEL-20)

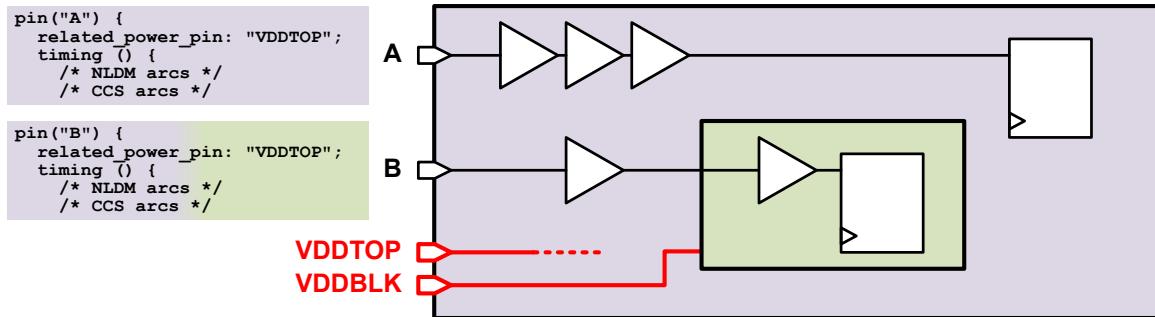
- For internal PG pins being merged, the numerically lowest or highest PG pin voltage across the input models is kept in the output model, per the `low` or `high` keyword specified with the `-merge_pg_pin` option:



A signal pin's `related_power_pin` is the PG pin that powers the logic *directly connected* to it. When the `-merge_pg_pin` option is used, all signal pins should connect directly to logic powered by boundary supplies (although fanout logic can be powered by internal supplies being merged).

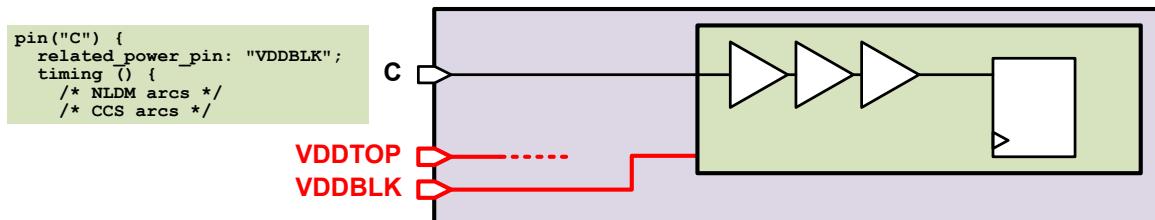
Signal pins are merged as follows:

- For a signal pin powered by a boundary PG pin (per the guideline):



then its arcs from the input models are conservatively merged:

- NLDM tables are merged to create output model arcs, keeping the worst-case value at each table point.
- CCS arcs are copied to the output model as parallel arcs.
- For a signal pin powered by an internal PG pin (against the guideline):



then the tool issues a warning:

Warning: The PG pin VDDBLK1 specified by option -merge_pg_pin is the related pg_pin of pin DATA. (MODEL-54)

and the pin arcs from the input models are merged as follows:

- NLDM tables are merged to create output model arcs, keeping the worst-case value at each table point.
- The CCS arcs from the input model with the numerically lowest or highest PG pin voltage (per low or high specified with the -merge_pg_pin option) are copied to the output model.

When a MODEL-54 warning occurs, the output model is potentially optimistic.

If an internal PG pin is not explicitly merged with the -merge_pg_pin option, then it uses boundary PG pin merging behaviors instead.

For details on how PG pin information is stored in an ETM via `voltage_map` and `related_power` constructs, see [Extraction of UPF and PG Data](#).

You cannot perform mode merging and PG pin merging within the same `merge_models` command; they are mutually exclusive. However, you can use the output models from PG pin merging as input to mode merging.

Controlling the Model Merging Process

Arc merging uses a tolerance value that 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.

You can 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.

Best Practices for ETM Generation

The following topics describe best practices for ETM generation.

Hierarchical Design Style Considerations

The ETM flow is known for providing good runtime and memory benefits at the top-level timing analysis in a hierarchical implementation and timing closure flow. There are many well-established design guidelines to improve the accuracy and convergence of the this flow. Because an ETM is a highly abstracted representation of the block interface timing, you should follow these important best design practices:

- Shield the block sufficiently
 - Avoid over-the-block routes and coupling
 - Space the physical boundary to reduce block-to-block abutted coupling

- Buffer block I/O ports
 - Buffer I/O ports and keep port wires short and shielded
 - Avoid direct wiretapping to output port and exposing block internal to I/O
- Register block I/O signals
 - Keep the interface circuit and constraints simple and register the signals
 - Avoid complex transparent latches at the I/O if possible

These commonly adopted hierarchical design methodologies can dramatically improve the quality of an ETM and achieve the best results. Particularly in an IP reuse flow where blocks are not only highly reused, but also in practically unpredictable chip-level contexts, the using rigorous and conservative design practices is often the best or only option to ensure the quality and the confidence of ETM usage.

Conservative Block Timing Analysis and Model Extraction

An ETM captures the block interface timing as a simple NLDM lookup function of the transition times and capacitive loads seen at the block I/O ports. For combinational and sequential delay paths, the resulting arcs are modeled by tables of two parameters:

- input_net_transition at clock or data input ports
- total_output_net_capacitance at the output ports

For setup and hold constraint paths, the resulting arcs are modeled by tables of two parameters: constrained_pin_transition at the data input ports and related_pin_transition at the clock source pins or ports.

With such great degree of complexity reduction in timing analysis, some accuracy might be lost and cannot be fully recovered unless the full block netlist is used in place of the ETM at the top level. This section provides an overview of how PrimeTime extracts a block into ETM and the recommendations for what you can do to create highly accurate yet conservative models with a focus on understanding how cross-coupling effects are accounted in the ETM.

Block-Internal Paths

When an ETM replaces a block netlist at the top level, all the block-internal register-to-register paths and their timing become invisible. Therefore, you must ensure that the block-internal paths meet timing requirements. This is achieved either by using full block netlist top-level analysis or closing block internal timing before generating the ETM.

Consider these important aspects for block-internal paths:

- Conservatively account for the potential coupling from the block interface paths to the internal paths.
- Check the skews of clocks that reach block boundary ports. This is particularly important for block-internal paths with launch and capture clocks entering through different ports of the block.

Block Interface Paths

During ETM extraction, a number of transition values and load values are chosen to sweep the interface paths. You can choose the number of table index values; while the tool internally determines the exact values to sweep the block interface. Here is a high level description of how ETM extraction accounts for crosstalk delay effects:

- The extraction sweeps the entire delay path with different transition values from the path starting point, which is either a data input port or a clock source pin/port.
- The extraction does a two-dimensional cross sweeping of the last stage with different transition values reaching the stage and the different load values set as the external lumped capacitance at the endpoint, which is always an output port.

This cross sweeping produces all the ETM delay arcs.

- The extraction does a two-dimensional cross sweeping of the constraint arcs with transition values reaching the register data and clock pins.

This cross sweeping produces all the ETM constraint arcs.

- The extraction sweeps all path stages with uncoupled delay computation, using the same calculation engine as specified for timing analysis.
- The bounding crosstalk delays from the current analysis are used.

For details, see [Conservative Modeling of the Block Context](#) and [Constraints for Block Inputs](#).

- By default, PrimeTime SI analysis filters the port stage and all aggressors of the port stage for very small noise bumps. But for a more conservative modeling of signal integrity in ETM, the tool includes all port stage aggressors including the filtered ones. This behavior is controlled by the `si_filter_keep_all_port_aggressors` variable.

When you generate an ETM with the `extract_model` command, the `si_filter_keep_all_port_aggressors` variable is forced to `true`. If necessary, this triggers an implicit `update_timing` command.

Conservative Modeling of the Block Context

With the previous understanding of the model extraction algorithm, it is clear that the quality of block timing analysis (`update_timing`) directly determines the quality and

accuracy of the timing model extracted (`extract_model`) for the same block. Therefore, all the guidelines on taking a conservative block closure approach in a hierarchical analysis flow are also critical to producing high-quality ETMs and achieve top-level timing closure.

In addition to the important best practices in design methodology described earlier, this section provides recommendations on proper PrimeTime usage to further ensure the quality of ETMs, particularly for IP blocks and hierarchical signoff usage of ETM models.

The general recommendation is to follow the worst-case scenario approach that is fundamental to static timing analysis:

- Make exhaustive and conservative assessments of the situations and contexts expected for block usage.

For example, the process corners and the operating voltages and temperatures need to be considered; the margins that need to be applied; the typical drivers and receivers expected on block I/O; timing budgets allocated for the interface; expected clock uncertainties, insertion delays and skews reaching the block, and so on.

- Specify the contexts as ranges of values or conditions in constraints that fully cover these expected instantiation scenarios.

Some ranges and values might need to be different for different corners and modes; covering them with separate timing constraints might be necessary.

- Apply the constraints for both timing analysis (to close the block timing under these contexts) and model extraction (to accurately bound block timing within these contexts).

For each corner and mode of the block, perform a separate analysis and model extraction.

Constraints for Block Outputs

For block outputs, the recommended commands for load ranges are

```
set_load -pin_load -min $min_cap [all_outputs]
set_load -pin_load -max $max_cap [all_outputs]
```

where the `$min_cap` is typically set as 0.0, meaning port is unloaded; `$max_cap` can be conservatively estimated as the input pin capacitance of a typical strong driver, plus the capacitance of a typical length wire allowed by the technology.

Constraints for Block Inputs

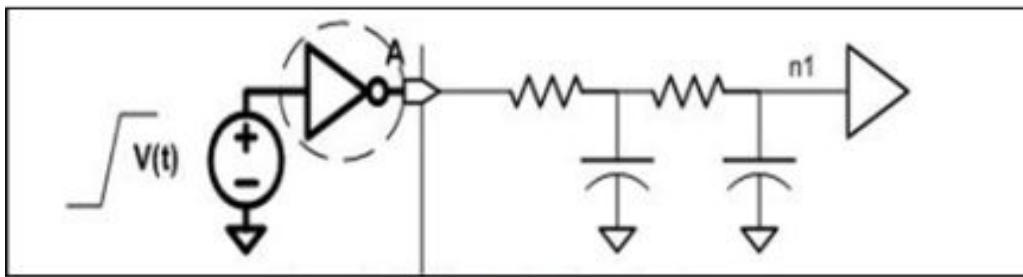
Inputs can be more complex than outputs because they have forward propagation effects for the entire path. The best practice is to use the following:

```
set_driving_cell -min -input_transition_rise/fall $min_slew \
    -lib_cell $strong_driver ...
set_driving_cell -max -input_transition_rise/fall $max_slew \
    -lib_cell $weak_driver ...
```

where the \$min_slew is typically set as 0.0 to model the largest aggressor effects when the port net is coupled to other nets in the block; \$max_slew is typically set as either the largest table index value characterized for the library cell used to buffer the input ports, or the max_transition DRC allowed by the design or technology, to model the worst case victim effects when the port net is coupled inside the block.

Sometimes, even when the max transition set at the input of the driving cell is very large, the resulting transition range at the output of driving cell can still be small and cause the table index ranges in the ETM to be very tight. This is because the driving cell is assumed to be right at the port, as shown in the following figure.

Figure 448 Driving cell set at the input port



If there is a long wire at the top level that connects the driver cell to the block, the actual slew degradation is difficult to model with only the `set_driving_cell` command. A potential way to work around this issue is to set a lumped capacitance on input port which can model long wire effects outside the block and at the same time give the best accuracy for PrimeTime SI analysis. As a result, there is a wider slew range at the input ports and also in the ETM lookup table.

For example:

```
set_driving_cell -min -input_transition_rise 0.010 \
    -lib_cell BUF6X [get_ports A]
set_load -min -pin_load 0 [get_ports A]
set_driving_cell -max -input_transition_rise 1.0 \
    -lib_cell BUF1X [get_ports A]
set_load -max -pin_load 0.020 [get_ports A]
```

In the preceding example, the `set_load -min` is set at 0 for conservative min slews and `set_load -max` is set to a value depending on the `max_transition` DRC rule for the driving cell. i.e the maximum external load allowed without causing a DRC violation for the driver cell.

The resulting largest transition at the output of the driving cell not only determines the largest index in the lookup table, it is also defined as the `max_transition` DRC rule for the library pin. Top level signals reaching the ETM with slower transitions are flagged as DRC violations, even if there is not necessarily timing violations reported.

Sometimes you might need wide transition index ranges in the ETM to accommodate for the potential situation of very long top level wires connecting the block input ports, yet you cannot get satisfactory range with very large max transition and weak driving cells at input ports. Although you can set lumped load to model long top-level wire effects, this hierarchical design practice is not recommended. Particularly for hierarchical signoff with ETMs, a ring of buffer around the peripheral of IP block is the safest approach.

Alternatively, block designers (particularly IP providers) choose to control transition ranges in the ETM tables more directly with:

```
set_input_transition -min $min_slew ...
set_input_transition -max $max_slew ...
```

where the \$min_slew is typically 0.0, and the \$max_slew is the largest index value characterized for the library cell at the input ports, or the max_transition DRC.

Note that during timing analysis and delay calculation, PrimeTime understands `set_input_transition` as an ideal driver with some fixed internal resistance. The resulting port net crosstalk delta delay tend to be more pessimistic comparing to driving cell, which is a more realistic port driver.

Crosstalk delay is also directly related to the alignment of arrival windows between aggressors and victims. Block input arrivals include both data arrival times specified with

```
set_input_delay -min/max ...
```

on the input ports as well as source latencies specified with

```
set_clock_latency -source-min/max -early/late
```

for clock ports. If you have good control and prediction of the arrival times reaching a block, these commands can be used to set up the constraints for the block inputs. If you do not have much knowledge or control of the arrival windows, only the worst case can be assumed for the actual context, it is recommended to set infinite windows for all coupling between the block interface and internal paths. Use the `set_si_delay_analysis -ignore_arrival` command.

Slew Propagation Effects

ETM .lib files use NLDM lookup tables to represent delay, slew and constraint as function of input slews and output loads. To produce these tables, during the extraction, PrimeTime performs sweeping computation of the interface paths with multiple slews simultaneously and for each input port individually, the sweeping algorithm computes the cell, net arc delays and path delays independently for each slew and load sampling point.

By default, there is no worst case slew coming from other ports at convergence fanin pins (such as for an AND gate, the slews from A pin and B pin converge at Z pin). This is very different from the standard block-based static timing analysis (GBA) during `update_timing` where single slews are propagated from all ports simultaneously and merged at convergence pins, a single worst case slew is then propagated downstream to

ensure pessimistic timing analysis. Therefore, from a slew propagation point of view, the ETM algorithm is similar to the more accurate path-based analysis (PBA) algorithm.

When there are convergence between significantly different fanin slews (such as when a relatively slow transition from asynchronous branch combines with relatively fast slews on the interface, or a gating control slew propagates into sharper control clock network), because ETM does not do slew worst casing from different branches, it is possible for ETM path delay to appear less pessimistic than GBA, while still conservative relative to the most accurate path-based analysis timing. This is also the reason the automatic model validation optionally uses path-based analysis to revalidate the ETM timing by removing the potential slew pessimism incurred during `update_timing` GBA propagation.

When this default and potentially more accurate ETM extraction behavior is undesirable, you can force the worst case with context-based slew with the following setting:

```
set extract_model_use_conservative_current_slew true
```

In this nondefault mode of model extraction, the multiple sweeping slews reaching at the convergence point are compared against the worst slew value annotated at the same pin (note this worst case pin slew is determined by the GBA analysis during `update_timing`), the sweeping slews which are considered as less conservative than the worst case pin slew are adjusted if necessary, thereby generating a more conservative model and match the GBA slew propagation better, and provide consistent timing information for the GBA model validation step.

Margin for Block Analysis and Model

Another way to have conservative block-level timing analysis and modeling is to use margins with command:

```
set_timing_derate -early/late ...
```

There are two ways to use derating in ETM flow:

- The recommended approach is to apply slightly extra margin at the block level than targeted full chip timing analysis margin. During ETM extraction, these derating factors are accounted for when the paths are swept and the arc delays are computed. When such pre-derated ETMs are used at top level, they need to be explicitly set with instance specific 1.0 derating to avoid double derating, unless further derating is truly needed.
- The second approach is applicable when block level derating is undesirable for some reason. With this approach, the block ETM is generated without timing deratings and at the top level, you specify the derating for the resulting ETM instances. The results of this approach is more difficult to correlate to the results if the ETM is replaced with the block netlist, because ETM abstracts entire clock and data paths into port to port timing arcs, derating of the ETM timing arc is often different from detailed derating of each stage on the paths.

At advanced process nodes, AOCV and POCV have become more widely adopted as both margin control and the same time a pessimism reduction mechanism. With AOCV, the margins are often defined as part of the library and the actual derating values are not a fixed global value but a function of the logic depth of the timing paths. Shorter path tends to have large margin on the beginning stages and longer paths have smaller margin at deeper stages. PrimeTime `extract_model` supports AOCV.

In a hierarchical flow, during block timing analysis and model extraction, a depth of zero is assumed for all the block ports, even though the block often is instantiated at the chip level with some paths depth before entering the block. This builds in some natural pessimism in both the block-level timing analysis and the ETM generated for the block. Also, during ETM extraction, the depth used by the ETM sweeping is GBA depth computed during `update_timing`. This is one of the reasons the ETM timing is more conservative than PBA. Even though the resulting ETM library contains no AOCV data; the derating effects are already accounted in the delay values of the arcs while sweeping the paths.

POCV models instance delay as a function of a random variable that is specific to the instance. To define the function, there are two types of POCV input data that you can feed into PrimeTime:

- A side file with POCV single coefficient
- A library with POCV slew-load table per timing arc (LVF)

The ETM has separate LVF tables for each timing arc, so you can use a single ETM with LVF tables at the top level to perform analysis and report at different sigma corners. For including constraint variation in the ETM, ensure that you have enabled constraint variation for your analysis run by setting the `timing_enable_constraint_variation` variable to `true`.

Validation of ETM Models

The validation step is a verification of the ETM generation. A timing picture of the initial netlist is taken and compared to the ETM timing pictures. You can compare each path slack or each I/O timing arc delay.

Model Validation Commands

PrimeTime validates the timing models created for a block by comparing the reported timing for the interface paths of the block against the timing of the block ETM instantiated in a wrapper. The wrapper of the ETM .lib cell is a simple netlist where each port of the wrapper is directly connected to the matching pin on the ETM cell. Using the same timing constraints such as clock definitions at the boundary and the same I/O context constraints are critical for validation to work.

There are two essential commands for model validation:

- `write_interface_timing`

This command must be applied to the netlist of the block and to the `test_design` that contains an instance of the ETM library cell. The command generates a timing file with slack or timing arc values. You decide whether to use slack or timing arcs as the criteria for comparison. A comparison using slack is a function of the timing constraints and arc delays. A comparison using timing arc values is independent of timing constraints.

- `compare_interface_timing`

This command compares timing between the netlist and the `test_design` containing an instance of the ETM library cell. A number of command options are available to specify the criteria and the pass/fail tolerance for comparison. A report is generated with failing and matching arcs based on the tolerance.

Automatic Model Validation

Model validation failures for an ETM are often due to the differences in the constraints and context setup between the block netlist and ETM wrapper. To validate the block ETM against the netlist timing, use automatic model validation (AMV).

To enable AMV, use the `extract_model` command with the `-validate timing` option. In addition to extracting the ETM, the command generates a model wrapper netlist with I/O constraints matching those applied to the original block netlist. A background process automatically captures the interface timing of the ETM and compares it against the netlist timing.

To run automatic model validation (AMV), use these commands:

```
set_app_var hier_modeling_version 2.0
extract_model -validate timing ...
```

The `hier_modeling_version` variable is needed because the AMV process creates specific directory structures for the models, constraints, and intermediate outputs.

The following table shows the major control variables for AMV and their defaults.

Variable	Default
<code>model_validation_capacitance_tolerance</code>	0.001
<code>model_validation_output_file</code>	<code>verif.out</code>
<code>model_validation_percent_tolerance</code>	0.00
<code>model_validation_reanalyze_max_paths</code>	1000

Variable	Default
model_validation_timing_tolerance	0.01

The essential mechanism of AMV is the same as the manual model validation described in the next section. You can even replay the steps by manually sourcing the intermediate validation scripts generated by AMV process.

AMV also automatically uses path-based analysis (PBA) when needed on the block netlist interface timing paths to validate the ETM timing. The main reason for doing PBA validation is slew propagation difference described in earlier sections.

Manual Model Validation Flow

This is a typical model validation flow:

1. Read the block-level netlist.

```
read_verilog ...
```

2. Apply block-level timing constraints that must be conservative for min analysis and max analysis. As a result, you have arrival windows for input and ranges for input transitions.

```
read_parasitics -keep_capacitive_coupling -format gpd | spf ...
source constraint.sdc
check_timing
```

3. Perform update_timing and generate timing reports:

```
update_timing
report_timing
extract_model ... -test_design ...
```

The -test_design option is useful for model validation, as it creates a test design with the newly generated library cell instantiated.

4. Use the write_interface_timing command on the design and compare to the initial netlist.

```
write_interface_timing -timing_type slack ... ref_netlist.tim
```

Typically, model validation is performed using slack as this includes the effects of both the timing constraints and timing arcs.

5. Read the test design generated by the extract_model command.

```
remove_design -all
set link_path "* ... model_lib.db"
read_db model_test.db
```

6. Apply block constraints.

```
link
read_parasitics -keep_capacitive_coupling -format gpd | spf ...
...
source constraint.sdc
```

7. Perform `update_timing` and generate timing reports; The timing type should be the same as the timing type used at the block level:

```
update_timing
report_timing
```

8. Write interface timing, and compare slacks with an absolute tolerance of 20 ps if the library unit is in ns:

```
write_interface_timing -timing_type slack ... etm.tim
compare_interface_timing Ref_netlist.tim etm.tim \
    -output comparison.tim -absolute_tolerance 0.020
```

Troubleshooting Model Validation Mismatches

This section references the “netlist design” and the “test design”. The netlist design is the block design from which an ETM is extracted. The `extract_model -test_design` command creates the test design, instantiating the library cell created for the ETM. Use the test design only for model validation.

Operating Conditions and Analysis Modes

You must be careful to use the same conditions of analysis between the block analysis and the test design analysis. Most issues come from setup errors.

Multicycle Path Timing Constraints

Multicycle paths on interface timing paths are not recommended and can be an issue for top-level integration.

When block-level multicycle paths impact interface timing paths, their effects are incorporated directly into the ETM model’s delay and constraint arcs. This adjustment can result in negative delay arcs. The `etm_name_constr.pt` constraint files do not contain any `set_multicycle_path` commands.

Multicycle paths defined for block internal register-to-register paths are not considered by ETM model generation because an ETM only covers interface paths of the block.

Asynchronous Timing Constraints

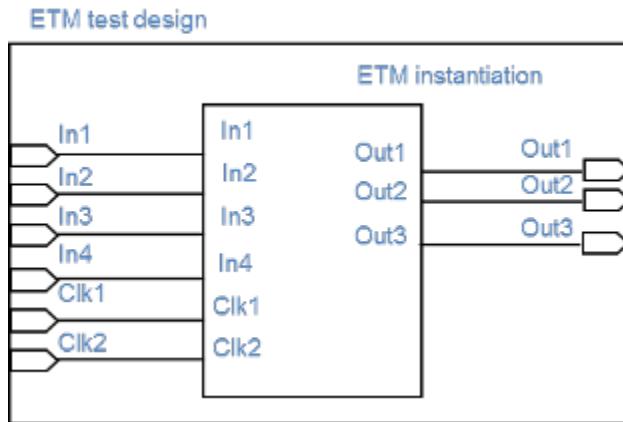
Asynchronous timing constraints specified by exception commands `set_max_delay` and `set_min_delay` are not supported by ETM model extraction and validation.

As explained in earlier sections, complex path specific exceptions are not recommended for block interface paths. Min/max delay values are directly tied with the context arrival times.

Effects of Wrapper Design

The following test design is an instantiation of the ETM library cell with nets connected from the ports to the ETM I/Os.

Figure 449 Test design with ETM instance



The delay to be seen on these nets for verification must be null. To remove the wire load model delay, you must back-annotate a zero capacitance value on all the nets of this test design:

```
set_resistance 0 [get_nets *]
set_load 0 [get_nets *]
```

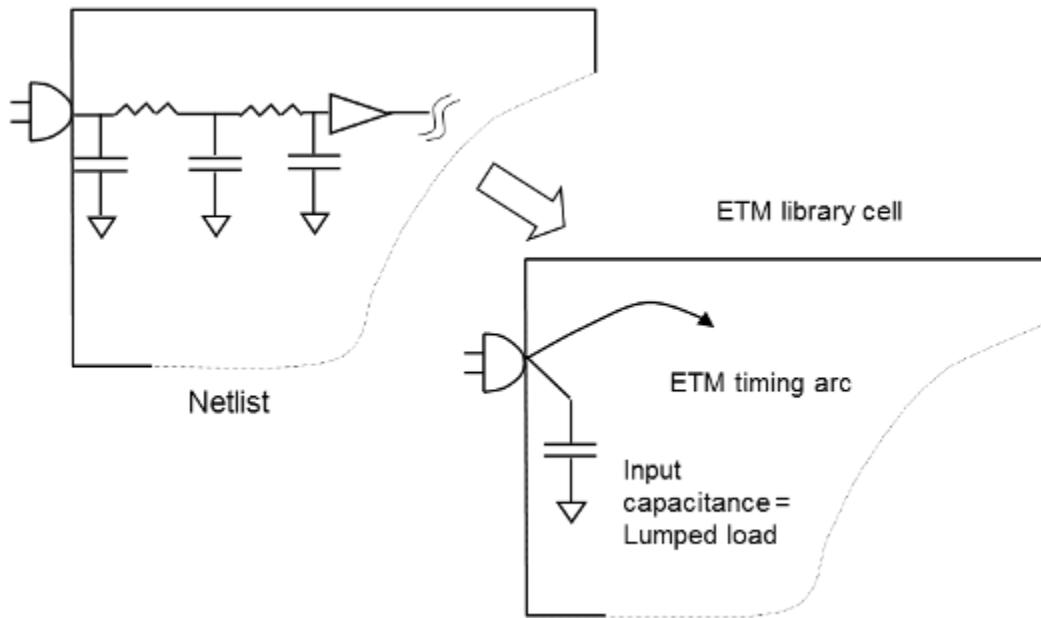
Setup for Outputs

For each primary output, the pin delay is computed using the range of external loads characterized for the driving device.

Inputs and Driving Cells

It is not a trivial task to set up the block netlist and ETM to do an exact comparison. This is particularly the case at the inputs. The `set_driving_cell` command is recommended for more realistic and higher quality timing analysis and model extraction at the block level. However, when driving cells are used in the timing analysis and extraction of block netlist, detailed RC networks exist at the boundary nets; while same driving cell applied to the model wrapper netlist, which only has lumped capacitances at the ETM input pins.

Figure 450 Back-Annotated Boundary Nets



Differences in validation can arise for the same driving cell between a net with distributed parasitics (block netlist analysis) and a lumped capacitance (ETM in wrapper design). The differences include both the driving cell arc delay, as well as the output transition which propagates into the netlist, or used to lookup the NLDM delay tables in ETM differently. Model validation errors caused by this is often small but can become more noticeable when the port net RC is large.

When this problem causes significant validation errors, it often indicates the block port nets are relatively long wires, not sufficiently buffered and subject to significant cross coupling. If you cannot improve the block design, one workaround is to use `set_load_min | -max` on the input ports with a lumped capacitance to model the long wire effects for the block interface, along with `set_driving_cell`.

You can annotate the load equal to the effective capacitance seen from the driving cell in the netlist. To save the effective capacitance value, set the following variable before `update_timing`:

```
pt_shell> set_app_var rc_cache_min_max_rise_fall_ceff true
```

The effective capacitance values on the driving cells are saved in the `cached_ceff_min/max_rise/fall` attributes, which you can query.

Derating

When timing margins are applied in the netlist analysis and included in the ETM timing arcs, the validation constraints must apply no derating in the wrapper and the ETM to avoid double derating.

Also, if AOCV is enabled during netlist timing and model extraction, the validation process should disable AOCV to avoid double derating.

Slew Propagation

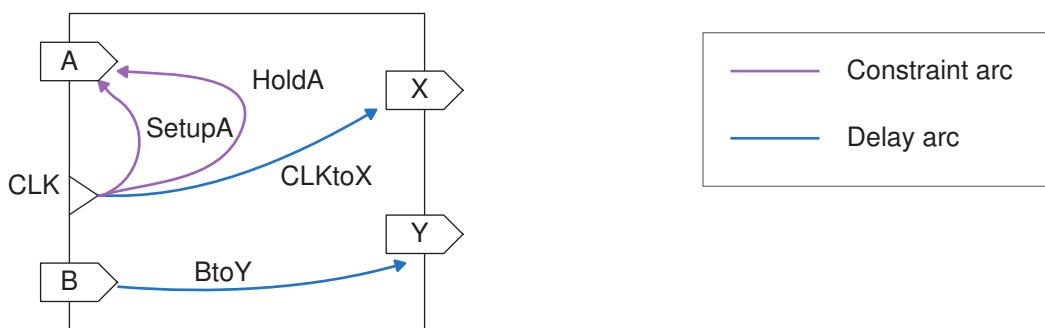
As noted in previous sections, `extract_model` and `update_timing` can take different approaches for worst-case slews at convergence pins. Therefore, graph-based analysis model validation (`write_interface_timing` and `compare_interface_timing`) can show the ETM to be less pessimistic than netlist timing. It is recommended that you use path-based analysis reporting for the netlist path to further verify the ETM timing and eliminate the “false optimism.”

Quick Timing Model (QTM)

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 the quick timing model with a netlist to obtain more accurate timing.

When defining a quick timing model, you use a series of 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.

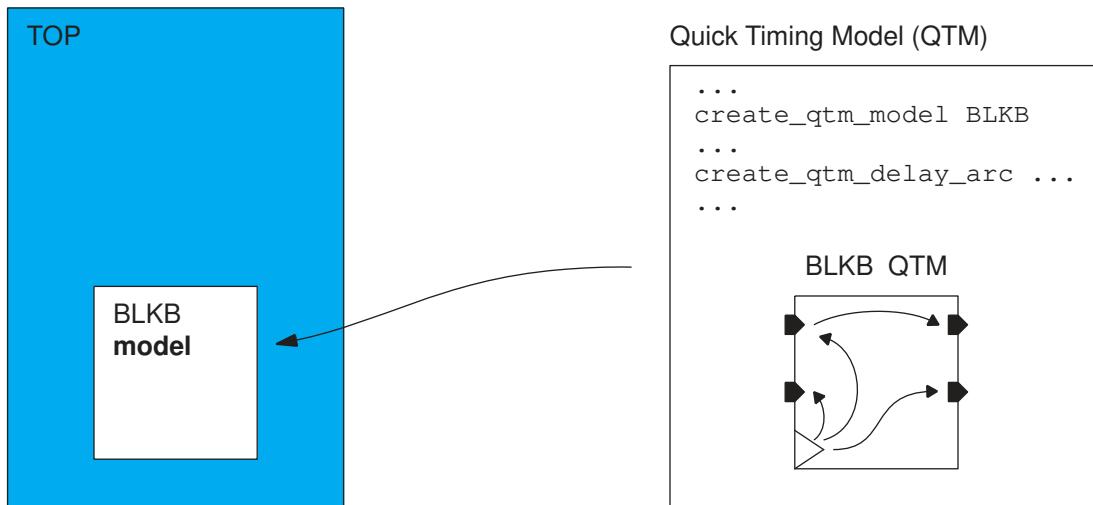
Figure 451 Quick Timing Model Representation of a Block



A quick timing model can be used in place of a block for timing analysis, as shown in the following figure.

Figure 452 Quick Timing Model Analysis

Hierarchical analysis using block models



PrimeTime uses the information in the quick timing model to create a library cell with the appropriate constraint and delay arcs. You can save a quick timing model in the Synopsys .db or .lib format, then instantiate the quick timing model in a design just as you would instantiate a library cell or interface timing specification model. Design Compiler, IC Compiler, and PrimeTime accept designs with instantiated quick timing models.

See Also

- [Defining a Quick Timing Model](#)
- [Instantiating a Quick Timing Model in a Design](#)

Defining a Quick Timing Model

To define a quick timing model:

1. Specify the name of the new quick timing model by using the `create_qtm_model` command.
2. Specify the global model parameters by using these commands:

Command	Global model parameter
<code>set_qtm_technology</code>	Logic library
<code>create_qtm_path_type</code>	Path type

Command	Global model parameter
set_qtm_global_parameter	Flip-flop setup time, hold time, and clock-to-output delay
create_qtm_load_type	Load type
create_qtm_drive_type	Drive type

3. Specify the quick timing model information by using these commands:

Command	Quick timing model information
create_qtm_port	Model port
set_qtm_port_load	Input port capacitance
set_qtm_port_drive	Output port drive
create_qtm_generated_clock	Generated clock
create_qtm_constraint_arc	Setup or hold constraint arc
create_qtm_delay_arc	Clock-to-output or input-to-output delay arc

4. Verify the model by using the `report_qtm_model` command.

5. Save the model by using the `save_qtm_model` command.

See Also

- [Specifying Constraint and Delay Arcs as Path Types](#)
- [Script Example to Create a Quick Timing Model](#)
- [Instantiating a Quick Timing Model in a Design](#)

Specifying Constraint and Delay Arcs as Path Types

When you use the `create_qtm_constraint_arc` or `create_qtm_delay_arc` command, you can specify the constraint or delay arc as a multiple of a path type that was defined by the `create_qtm_path_type` command. For example, you can specify these arcs:

- A setup constraint arc that is equal to the setup time of a D flip-flop in the library.
- An input-to-output delay arc that is equivalent to a path containing five NAND gates with an average fanout of three for each gate.

Using your specifications, PrimeTime calculates constraint and delay times as follows:

- Setup or hold time
 $= \text{number_of_levels} * (\text{delay}/\text{level}) + \text{global setup or hold time}$
- Clock-to-output delay time
 $= \text{number_of_levels} * (\text{delay}/\text{level}) + \text{CLK-to-output delay}$
 $+ \text{load-dependent output delay}$
- Input-to-output delay time
 $= \text{number_of_levels} * (\text{delay}/\text{level}) + \text{load-dependent output delay}$

If the fanout of the input port is high, you can insert a chain of buffers before driving the library cells or split the fanout among several buffers. To account for such delays, add the number of levels when specifying the input arcs.

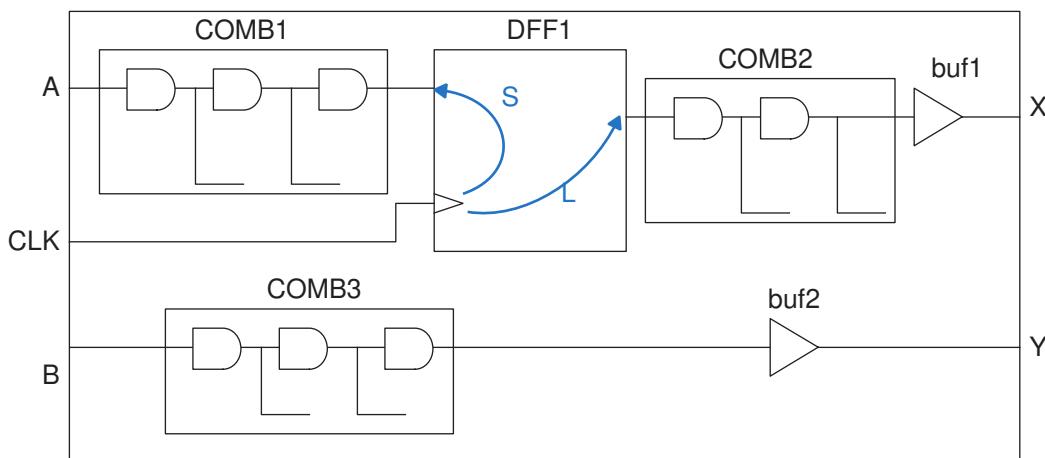
See Also

- [Defining a Quick Timing Model](#)

Script Example to Create a Quick Timing Model

The subsequent script defines this quick timing model example.

Figure 453 Quick timing model example



```
### Create the model and set global model parameters ###

# Specify the model name
create_qtm_model qtm_example

# Specify the logic library
```

```

set_qtm_technology -library my_lib

# Define a path type; path1 is a 2-input NAND with a fanout of 2
create_qtm_path_type path1 -lib_cell nand21 -fanout 2

# Define a load type
create_qtm_load_type load1 -lib_cell and2

# Define the drive types
create_qtm_drive_type drive1 -lib_cell buf1
create_qtm_drive_type drive2 -lib_cell buf2

# Define the global setup time, which is equivalent to the
# setup time of the DFF1 library cell
set_qtm_global_parameter -param setup \
    -lib_cell DFF1 -clock CLK -pin D

# Define the hold time
set_qtm_global_parameter -param hold -value 0.0

# Define the clock-to-output delay time, which is equivalent to
# the launch time of the DFF1 library cell
set_qtm_global_parameter -param clk_to_output \
    -lib_cell DFF1 -clock CLK -pin Q

### Specify ports, constraint arcs, delay arcs ###

# Create a clock port and set the load
create_qtm_port {CLK} -type clock
set_qtm_port_load {CLK} -value 3

# Create the input ports and set the loads
create_qtm_port {A B} -type input
set_qtm_port_load {A B} -type load1 -factor 2

# Create the output ports and set the drives
create_qtm_port {X Y} -type output
set_qtm_port_drive X -type drive1
set_qtm_port_drive Y -type drive2

# Create the setup and hold constraint arcs
create_qtm_constraint_arc -setup -edge rise -name SetupA \
    -from CLK -to A -path_type path1 -path_factor 2

create_qtm_constraint_arc -hold -edge rise -name HoldA \
    -from CLK -to A -path_type path1 -path_factor 2

# Create the clock-to-output and input-to-output delay arcs
create_qtm_delay_arc -name CLKtoX \
    -from CLK -to X -path_type path1 -path_factor 2

create_qtm_delay_arc -name BtoY -from B \
    -to Y -path_type path1 -path_factor 3

```

```
# Save QTM to qtm_example library cell in the myqtm.db file
save_qtm_model -format db myqtm -library_cell
```

Using the specified quick timing model, PrimeTime calculates the constraint and delay times as follows:

- Setup time of port A relative to CLK

$$\begin{aligned} &= \text{delay of COMB1} + \text{setup time of DFF1} \text{ (denoted by arc S)} \\ &= 3 * (\text{delay of path1}) + \text{global setup time} \end{aligned}$$
- Delay from CLK to output port X

$$\begin{aligned} &= \text{launch time of DFF1} \text{ (denoted by arc L)} + \text{delay of COMB2} \\ &\quad + \text{load-dependent delay of BUF1} \\ &= \text{launch time of DFF1} + 2 * (\text{delay of path1}) + \text{delay of BUF1} \end{aligned}$$
- Delay from input port B to output port Y

$$\begin{aligned} &= \text{delay of COMB3} + \text{load-dependent delay of BUF2} \\ &= 3 * \text{delay of path1} + \text{delay of BUF2} \end{aligned}$$

See Also

- [Defining a Quick Timing Model](#)

Instantiating a Quick Timing Model in a Design

To instantiate a quick timing model in a design:

1. Instantiate the model in your design in the same way you instantiate a leaf library cell.
2. Add the library file name to the `link_path` variable and the path name of the directory containing this file to 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 by using the `link_design` command.
5. Define the clocks and other timing assertions for the design containing the model.
6. Report the timing of the design by using the `report_timing` command.

See Also

- [Defining a Quick Timing Model](#)

23

Using PrimeTime With SPICE

You can compare PrimeTime results with SPICE analysis by using the following capabilities:

- [Simulation Link to Correlate PrimeTime and SPICE Results](#)
 - [Generating a SPICE Deck With the write_spice_deck Command](#)
-

Simulation Link to Correlate PrimeTime and SPICE Results

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:

- [Summary of Simulation Link Commands](#)
- [Correlating Path-Based Uncoupled PrimeTime and SPICE Analysis](#)
- [Correlating Arc-Based Coupled PrimeTime SI and SPICE Analysis](#)
- [Correlating PrimeTime SI and SPICE Noise Analysis](#)
- [Distributing Simulations Across Multiple Machines](#)

Summary of Simulation Link Commands

The following simulation link commands facilitate SPICE analysis in PrimeTime.

Table 77 Summary of simulation link commands

Command	Task
sim_setup_library	Set up the SPICE models for simulation by specifying the library name, subcircuit directory name, and header file name.
sim_setup_simulator	Specify the name of the HSPICE simulator executable, simulator options for comparisons between PrimeTime and the simulator.
sim_validate_setup	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.
sim_validate_path	Perform path-based uncoupled SPICE analysis on a specified path segment and compare the simulation results against the static timing results.
sim_enable_si_correlation	Enable nets for coupled one-and-a-half-stage correlation.
sim_validate_stage	Perform arc-based coupled SPICE analysis on a specified stage or one-and-a-half stage and compare the simulation results against the static timing results.
sim_validate_noise	Compare PrimeTime SI noise analysis results with SPICE results.

Correlating Path-Based Uncoupled PrimeTime and 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 following commands:

```
# Disable use of nonconditional timing arcs between pins that have
# at least one conditional timing arc
set_app_var timing_disable_cond_default_arcs true

# Report all paths through parallel cell arcs
set_app_var timing_report_use_worst_parallel_cell_arc false

# Enable waveform propagation
set_app_var delay_calc_waveform_analysis_mode full_design

# Store waveform data when CCS waveform propagation is enabled
set_app_var timing_keep_waveform_on_points true
```

```
# Disable advanced on-chip variation (AOCV)
set_app_var timing_aocvm_enable_analysis false

# Disable AOCV for SPICE correlation at specific corners
remove_ocvm

# Reset user-specified derate factors
reset_timing_derate
```

- 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
```

- Specify the path to the SPICE simulator. For example:

```
pt_shell> sim_setup_simulator \
           -simulator /abc/tools/bin/hspice \
           -simulator_type hspice
```

- 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.

- 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]
```

- 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 -transition_time $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_time $my_paths
```

The validation report shows the SPICE simulated value (Ref), PrimeTime value (Val), absolute difference (Diff), and percentage difference (%) for delay and transition time.
 For example:

Comparing path number 0 :									
Pin	Sense	Transition time			Delay				%
		Ref	Val	Diff	%	Ref	Val	Diff	
U3/A	f	0.0197	0.01870	-0.00100	-5.12	0.017613	# 0.01749	-0.000117	-0.67
U4/A	r	0.0175	0.01730	-0.00025	-1.45	0.014172	# 0.01404	-0.000129	-0.91
U5/A	f	0.13	0.1371	0.0031	2.33	0.083447	# 0.08412	0.000681	0.82
U5/Z	f	0.239	0.2431	0.00365	1.53	0.131	# 0.1345	0.00267	2.03
Path segment		0.0830	0.08385	0.000842	1.01	0.24713	0.2502	0.00310	1.26

Notes:

- If a path starts at an input port, set up a simple combinational cell, such as a buffer or inverter, as the driving cell on the input port by using the `set_driving_cell` command. For example:


```
pt_shell> set_driving_cell -lib_cell IBUF2 [get_ports in1]
```
- 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.

Correlating Arc-Based Coupled PrimeTime SI and 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. Configure PrimeTime SI analysis by using the following commands:

```
# Enable PrimeTime SI analysis
set_app_var si_enable_analysis true

# Disable use of nonconditional timing arcs between pins that have
# at least one conditional timing arc
set_app_var timing_disable_cond_default_arcs true

# Report all paths through parallel cell arcs
set_app_var timing_report_use_worst_parallel_cell_arc false

# Enable waveform propagation
set_app_var delay_calc_waveform_analysis_mode full_design

# Store waveform data when CCS waveform propagation is enabled
set_app_var timing_keep_waveform_on_points true

# Disable advanced on-chip variation (AOCV)
set_app_var timing_aocvm_enable_analysis false

# Disable AOCV for SPICE correlation at specific corners
remove_ocvm

# Reset user-specified derate factors
reset_timing_derate
```

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 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]
```

Note:

It is not recommended to perform arc-based correlation on stages driven by ports.

6. Enable coupled one-and-a-half-stage correlation for \$net, and perform a timing update. For example:

```
pt_shell> sim_enable_si_correlation \
    [get_nets -of [get_attribute $my_arcs from_pin]]
pt_shell> update_timing -full
```

7. 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 SPICE simulated value (Ref), PrimeTime value (Val), absolute difference (Diff), and percentage difference (%) for delay. For example:

```
pt_shell> sim_validate_stage $my_arcs
```

Pin	Sense	Delay			
		Ref	Val	Diff	%
<hr/>					
U2/Z	f	0.59594 #	0.594371	-0.00156873	-0.26
<hr/>					
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

Correlating PrimeTime SI and SPICE Noise Analysis

To compare PrimeTime SI noise analysis results with SPICE results, run the `sim_validate_noise` command. This command compares the noise bump height and area reported by both tools.

Use the following flow for noise correlation:

```
# Specify the nets on which to compare noise analysis
sim_enable_si_correlation [get_nets -of_objects net_driver_pin]

# Run timing and noise analysis
update_timing -full
update_noise -full

# Specify the timing arcs and run noise correlation
set net_arc [get_timing_arcs -from net_driver_pin -to net_receiver_pin]
sim_validate_noise $net_arc -area [-verbose] \
    -analysis_type above_low | below_high
```

Distributing Simulations Across Multiple Machines

If you need to run simulations for many collection objects, you can use the `sim_setup_distributed` command to distribute their simulations across multiple machines.

For example,

```
pt_shell> sim_setup_distributed \
    -mode grd \
    -submit_command {qsub -V -cwd -P bnormal} \
    -sim_per_job 2

pt_shell> sim_validate_path $twelve_paths
```

The `-sim_per_job` option specifies the number of collection object simulations per submitted job. The total number of submitted jobs depends on the number of collection objects given to the simulation command. (The sample spaces of variation simulations are not split across jobs.)

When the distributed simulation jobs complete, the tool prints a job summary:

Summary:

```
Total number of simulations      : 12
Simulation per farm job        : 2
Total number of farm jobs       : 6
Completed number of farm jobs  : 4
Failed number of farm jobs     : 0
Timed out farm jobs            : 2
-----
```

The following commands support distributed simulation:

- `sim_validate_noise`
- `sim_validate_path`
- `sim_validate_setup`
- `sim_validate_stage`
- `sim_analyze_path`

The `sim_setup_distributed` command provides additional options to limit queued jobs and retry failed jobs. For details, see the man page.

Note the following limitation of the `sim_setup_distributed` command:

- The `-from` and `-to` options of the `sim_validate_path` command are not supported.

Generating a SPICE Deck With the `write_spice_deck` Command

To generate a SPICE deck, run 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

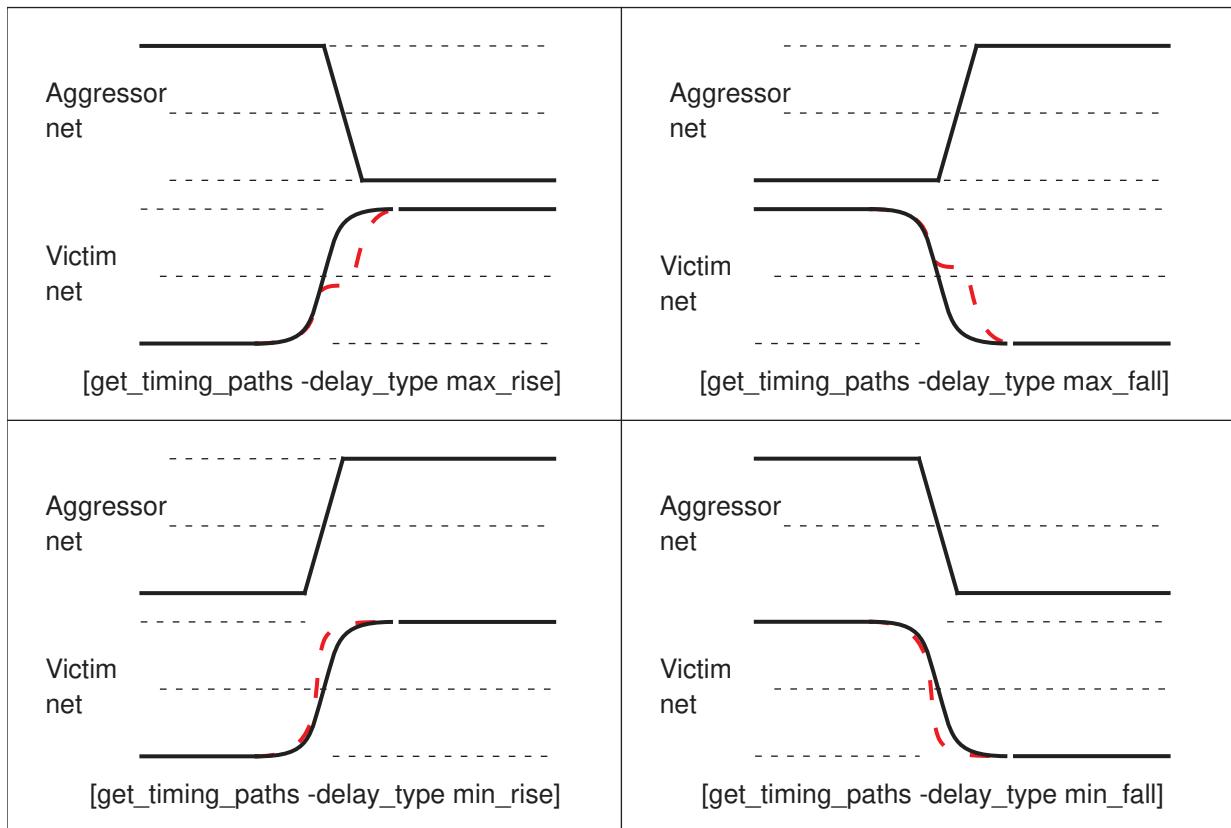
To analyze crosstalk delay effects (but not static noise effects) for a timing path, generate a SPICE deck that represents the timing path. Use the `get_timing_paths` command to collect paths for the `write_spice_deck` command. For example:

```
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 the following figure. 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 454 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:

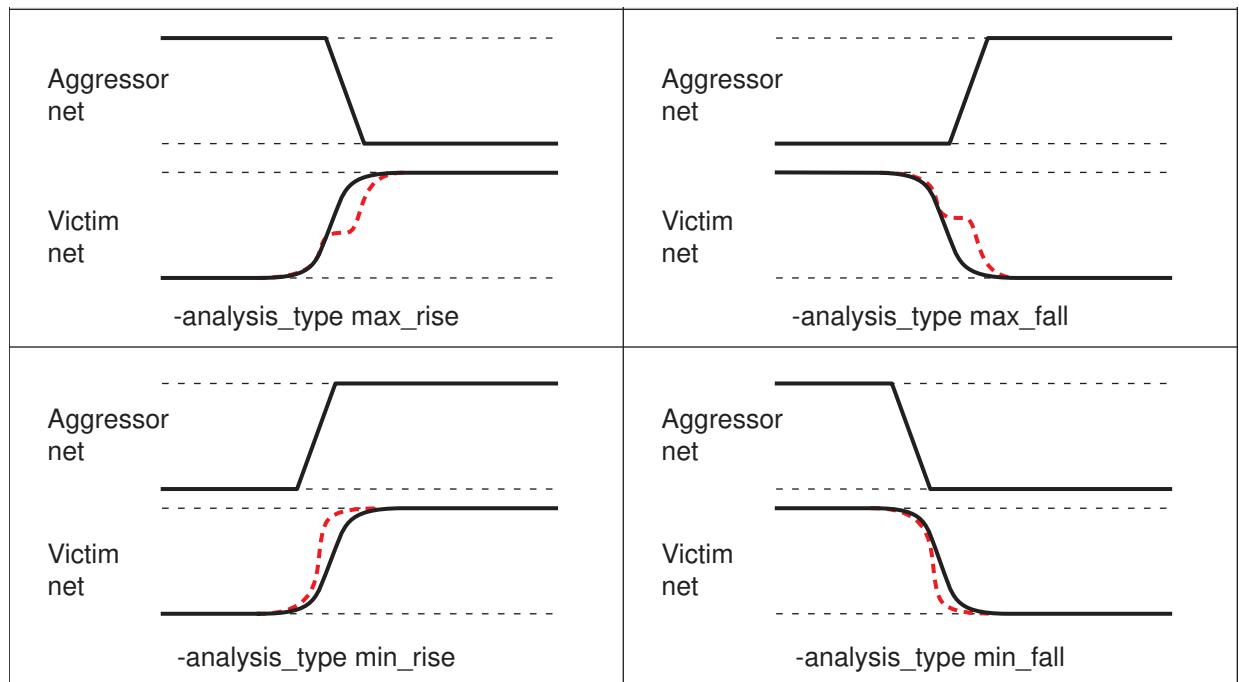
```
pt_shell> write_spice_deck -header header.spi \
    -analysis_type above_high \
    -output ../SIMUL_BEYOND_HIGH/new_general.spi \
    -logic_one_voltage 1.5 -logic_zero_voltage 0.0 \
```

```
-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:

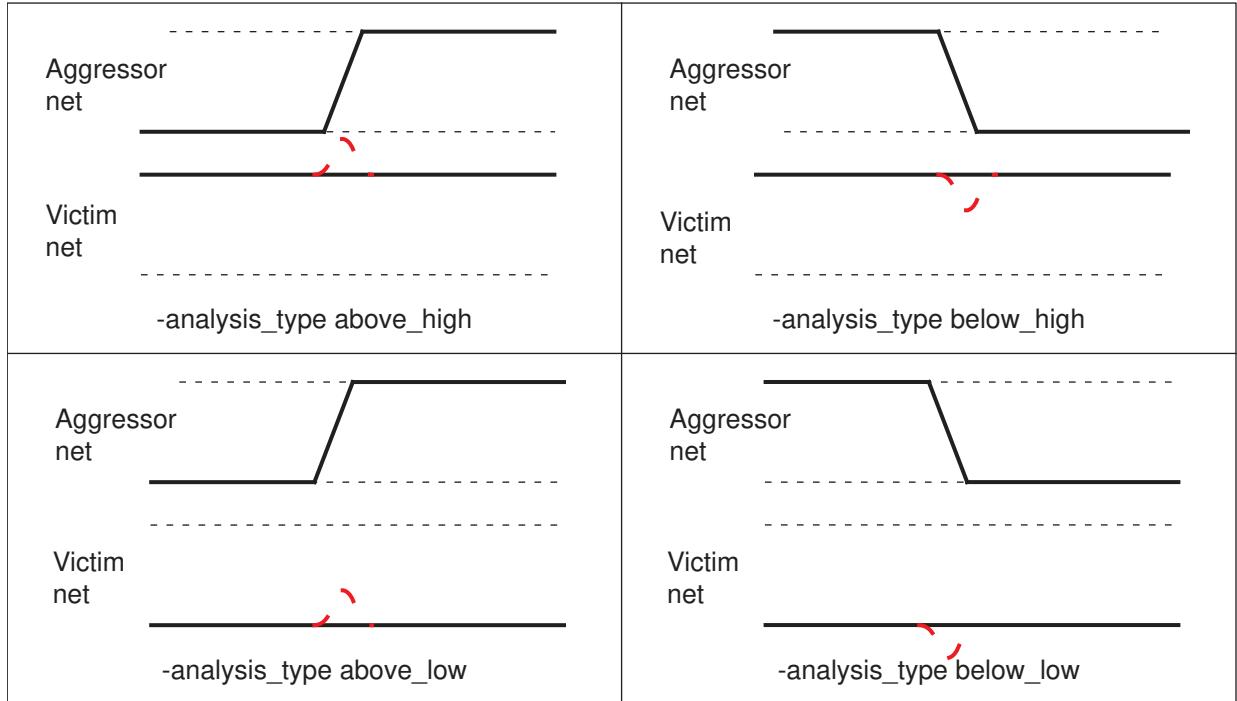
- For crosstalk delay analysis, use `max_rise`, `max_fall`, `min_rise`, or `min_fall`.

Figure 455 Crosstalk delay transitions for arcs



- For static noise analysis, use `above_high`, `below_high`, `above_low`, or `below_low`.

Figure 456 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 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 : 3;
            }
        }
    }
}
```

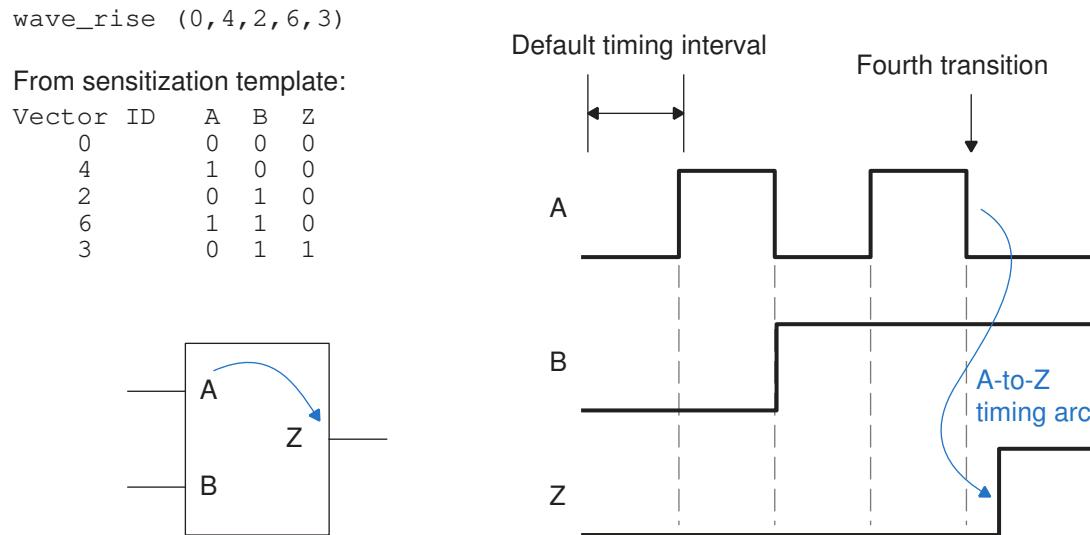
The sensitization template `2in_1out` defines eight vectors. Each vector defines a logic value for the cell pins in the order corresponding to the `pin_names` list.

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.

The following figure 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 457 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.

The following example shows the output file generated by this command and the applicable input SPICE subcircuit definitions.

Example 90 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
```

Chapter 23: Using PrimeTime With SPICE
 Generating a SPICE Deck With the `write_spice_deck` Command

```
* 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 91 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.

24

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 From Design Compiler](#)
 - [Sharing Design Compiler and PrimeTime Scripts](#)
 - [Path Groups in Design Compiler and PrimeTime](#)
-

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 From Design Compiler

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]
```

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 Format \(SDC\) 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 Format \(SDC\) 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
source 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 Format (SDC) Script Files

Synopsys Design Constraints (SDC) format 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

To check the SDC file and verify that it is syntactically and semantically correct, run 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*, available on [SolvNetPlus](#).

Checking the Consistency of Units

To check that the units in an SDC file are consistent with the main library units, run the `set_units` command:

```
set_units -time 1ns -capacitance 1nF -current 1mA \
           -voltage 1V -resistance 1kOhm -power 1W
```

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 inconsistent units from different constraint files.

Path Groups in Design Compiler and PrimeTime

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.

A

Deprecated Features

The following features are currently deprecated and will be obsoleted in a future release:

- [HyperScale Distributed Analysis](#)
 - [Legacy Path-Based SMVA](#)
-

HyperScale Distributed Analysis

The documentation sections for this feature are:

- [From Full-Chip Flat to HyperScale Distributed Analysis](#)
 - [Flat-Like Top-Level Reporting Using Restored Sessions](#)
 - [From Distributed Analysis to Block-Level Analysis](#)
 - [MIM Merging in Distributed HyperScale Analysis](#)
 - [HyperScale Attributes](#)
-

From Full-Chip Flat to HyperScale Distributed Analysis

If you have an existing script that performs full-chip flat analysis, you can easily reuse the same script to perform HyperScale hierarchical analysis. You only need to add a few lines to enable hierarchical analysis and specify the hosts.

The PrimeTime tool automatically runs a separate analysis for each lower-level block, generates a HyperScale model of each block, and runs a top-level analysis using the generated models. This method is called HyperScale distributed analysis.

Here is an example. The lines in boldface are added to the full-chip flat analysis script.

```
# Enable hierarchical analysis
set_app_var hier_enable_distributed_analysis true

# Set working directory for analysis data
set_app_var hier_distributed_working_directory $nfs_dir

# Optional: override default distributed hierarchical partitioning
set_hier_config ...
```

```

# Specify hosts for distributed analysis
set_host_options -num_processes $nProc1 -max_cores $mCore1 \
    -submit_command "..."
set_host_options -num_processes $nProc2 -max_cores $mCore2 \
    -submit_command "..."
...
start_hosts

# Run full-chip flat analysis script
read_verilog blk_A.v
read_verilog blk_B.v
read_verilog top.v
link_design TOP
...
update_timing
...

# Flat-like reporting from top level
report_timing -path_type full_clock_expanded \
    -from IN1 -to U2/reg78 ...

```

When hierarchical analysis is enabled (when the `hier_enable_distributed_analysis` or `hier_enable_analysis` variable is set to `true`), one PrimeTime-ADV license is required for each 32 cores requested for analysis. You need to set up the hosts and set the hierarchical configuration before design linking. The tool automatically assigns the lower-level blocks for execution on the available hosts.

The block-level Verilog netlist and parasitic data are not needed for any blocks (or sub-blocks inside) represented as HyperScale block models. However, if your existing script performs top-level full-flat timing analysis by reading in all the top-level and block-level full Verilog netlists and parasitics, you can still keep your original script. There is no need to remove the commands that read in the block-level Verilog and parasitics.

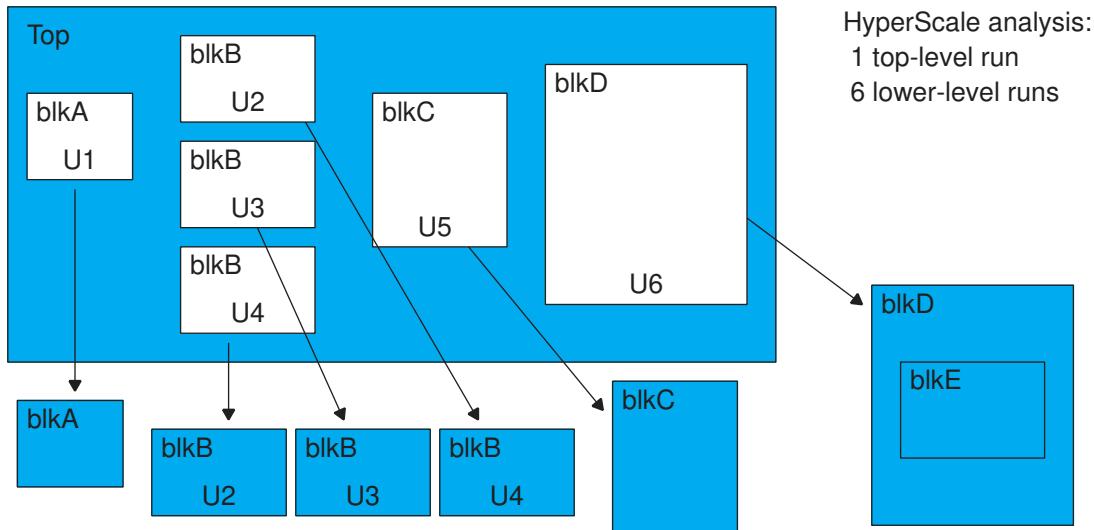
After you enable the distributed analysis flow (set `hier_enable_distributed_analysis` true) and properly set the HyperScale configuration (use `set_hier_config`), the tool automatically links with the HyperScale model data and does not link the full block-level Verilog, even if read. It also automatically loads the reduced parasitics from the HyperScale data and skips reading the full block-level parasitics, even if directed to do so by the script.

However, to further improve the runtime and memory usage, you can remove the commands in the script that read in the block-level Verilog.

Default Partitioning for Distributed Analysis

By default, the distributed analysis method puts each block at the top level into its own partition, as shown in the following example.

Figure 458 Default Partitioning in Distributed Analysis



Overriding the Default Partitioning in Distributed Analysis

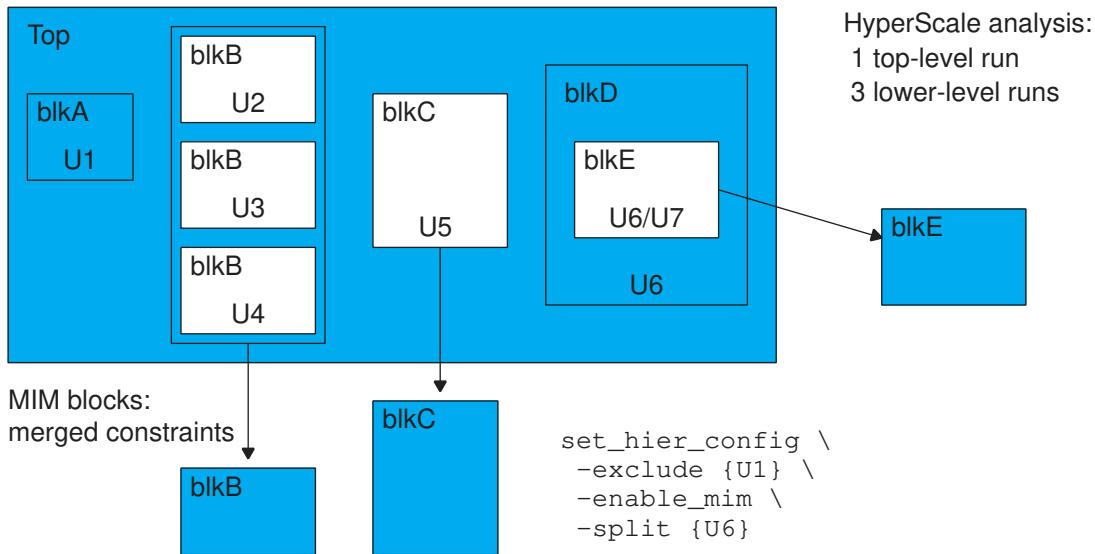
To override the default partitioning, use the `set_hier_config` command. For example,

```
set_hier_config \
    -enable_mim \
    -exclude {U1} \
    -split {U6}
```

Merge context for MIM blocks
Keep U1 (blkA) with top level
Split U6 (blkD) one level down

The following diagram shows the resulting changes to the design partitioning.

Figure 459 User-Directed Distributed Analysis Partitioning Example 1

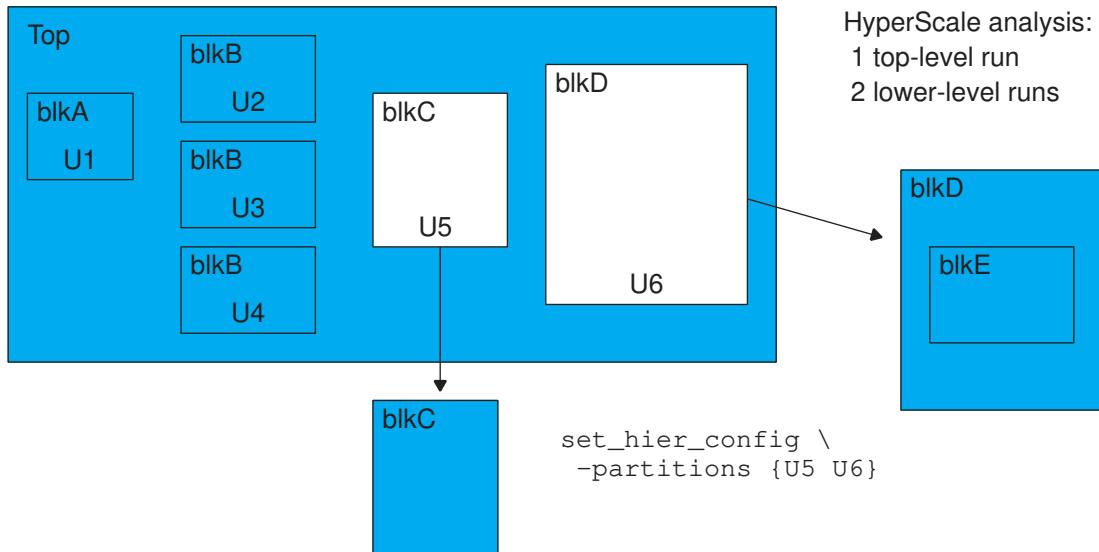


To assign only certain large blocks for separate analysis and keep all other blocks at the top level, use the `set_hier_config` command with the `-partitions` option. For example,

```
set_hier_config \
-partitions {U5 U6}      # Analyze only U5 and U6 as separate blocks
```

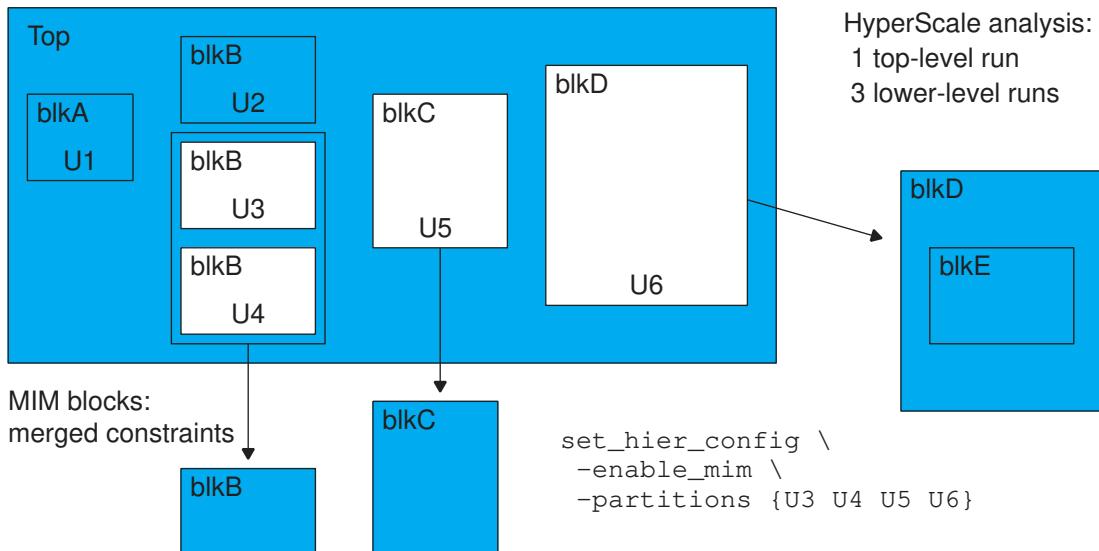
The following diagram shows the resulting changes to the design partitioning.

Figure 460 User-Directed Partitioning Example 2



You cannot use the `-partitions` option with the `-exclude` or `-split` option. However, you can use the `-partitions` option with the `-enable_mim` option to selectively partition a subset of the multiply instantiated modules, as shown in the following example.

Figure 461 User-Directed Partitioning Example 3



Host Resource Affinity

You can explicitly assign larger blocks to the hosts with more CPU and memory resources by using the `set_host_options` and `set_hier_config` commands. For example, the following commands define the host resources and analyze block instances P1, P2, and P3 as HyperScale blocks using the specified resources.

```
set_host_options -name RA -num_processes 1 -max_cores 2 ... host1
set_host_options -name RB -num_processes 1 -max_cores 1 ... host2
set_host_options -name RC -num_processes 1 -max_cores 1 ... host3
start_hosts
report_host_usage -verbose
...
set_hier_config -partitions {P1 P2} -affinity {RA RB} ...
set_hier_config -partitions {P3} -affinity {RC} ...
...
```

The tool assigns partitions P1 and P2 for analysis on resources RA and RB, and partition P3 for analysis on resource RC. The larger of partition P1 or P2 (based on number of cells) is analyzed on the larger of resources RA and RB (based on allocated memory).

Attribute Access in Lower-Level Blocks

In distributed analysis, when you query timing attributes of objects in lower-level blocks, the response can take some time because the attributes are not immediately available from the top level. For faster response, you can gather the attribute data in a cache.

For example, the following script efficiently gathers and uses slack attribute data from hierarchical pins:

```
cache_distributed_attribute_data -class pin \
    -attributes {min_rise_slack max_rise_slack}

set mypins [get_pins -hierarchical *]
foreach_in_collection pin $mypins {
    set slack1 [get_attribute $mypins min_rise_slack]
    set slack2 [get_attribute $mypins max_rise_slack]
    ...
}
```

To remove the cached attribute data:

```
purge_distributed_attribute_data -class pin \
    -attributes {min_rise_slack max_rise_slack}
```

Distributed Analysis Top-Only Analysis Flow

The top-only distributed analysis flow reduces compute resource usage when you are interested in timing results only for the top level. To choose this flow:

```
pt_shell> set_hier_config -enable_top_only_flow ...
```

In this flow, HyperScale distributed analysis works only on the top-level part of the design. Timing reports are available only for “active objects,” which are the ports, cells, nets, and pins of the top design. To find out whether an object is active, query its `is_active` attribute from the distributed analysis manager:

```
pt_shell> get_attribute [get_pins U123/ZN] is_active
true
```

In the top-only distributed analysis flow, the tool releases compute resources for block partitions after a timing update, which prevents further timing updates. To retain the compute resources after timing updates in the top-only mode, set the following variable:

```
pt_shell> set hier_enable_release_resources_in_top_only_flow false
```

Flat-Like Top-Level Reporting Using Restored Sessions

If you have already analyzed the block-level and top-level timing in HyperScale analysis sessions, you can restore the sessions into a new top-level session and report timing paths from that level, including timing paths that cross hierarchical boundaries. You can do this without incurring the cost of another top-level analysis.

For example, suppose that you previously ran separate HyperScale analyses for blkA, blkB, and the top level:

```
-----
# Analysis session for block blkA
...
read_context $topDir
...
save_session $blkAsession

-----
# Analysis session for block blkB
...
read_context $topDir
...
save_session $blkBsession

-----
# Analysis session for top level
...
save_session $topSession
```

To restore these sessions and generate timing reports from the top level, do the following:

```
...
set_app_var hier_enable_analysis true

# Read existing HyperScale session data
set_hier_config -session -block blkA -path $blkAsession
```

```

set_hier_config -session -block blkB -path $blkBsession
set_hier_config -session -top -path $topSession

# Generate full-chip timing report and noise report
report_timing -path_type full_clock_expanded -from IN1 -to BLKA/reg2 ...
report_noise ...
report_constraint ...
report_analysis_coverage ...
report_qor ...

```

After you restore the block-level and top-level sessions, timing reporting works the same as in a HyperScale distributed analysis session.

From Distributed Analysis to Block-Level Analysis

The HyperScale distributed analysis method automatically runs block-level analysis for each block in the background. After you run a HyperScale distributed analysis, you can easily rerun block-level analysis using the automatically saved block-level session data.

For example, suppose that you have already ran a distributed analysis:

```

set_app_var hier_enable_distributed_analysis true
set_app_var hier_distributed_working_directory $nfs_dir
set_host_options -num_processes $nProc1 -max_cores $mCore1 \
    -submit_command "..."
set_host_options -num_processes $nProc2 -max_cores $mCore2 \
    -submit_command "..."
...
start_hosts

# Run full-chip flat analysis script
read_verilog ...
...
update_timing
...
save_session $DH_Session

```

When you save the distributed analysis session, the tool saves the top-level and block-level data into separate subdirectories. In a new PrimeTime session, you can restore just the session data for one block or just the top level, and separately reanalyze that part of the design.

For example,

```
-----
# Restore separate session for block blkA
set_app_var hier_enable_analysis true
restore_session $DH_Session/blkA
report_timing ...
```

```
-----
# Restore separate session for block blkB
set_app_var hier_enable_analysis true
restore_session $DH_Session/blkB
report_timing ...

-----
# Restore separate session for top level
set_app_var hier_enable_analysis true
restore_session $DH_Session/top
report_timing ...
```

By default, the `restore_session` command restores all partitions of the original session. To save time and memory, you can selectively restore only the HyperScale partitions of interest by using the `-partitions` option of the `restore_session` command.

By default, a restored distributed analysis session uses the same host option settings as the original session, as defined by the original `set_host_options` commands. However, you can override the original host settings by running the `set_host_options` command with new settings before the `restore_session` command. For example,

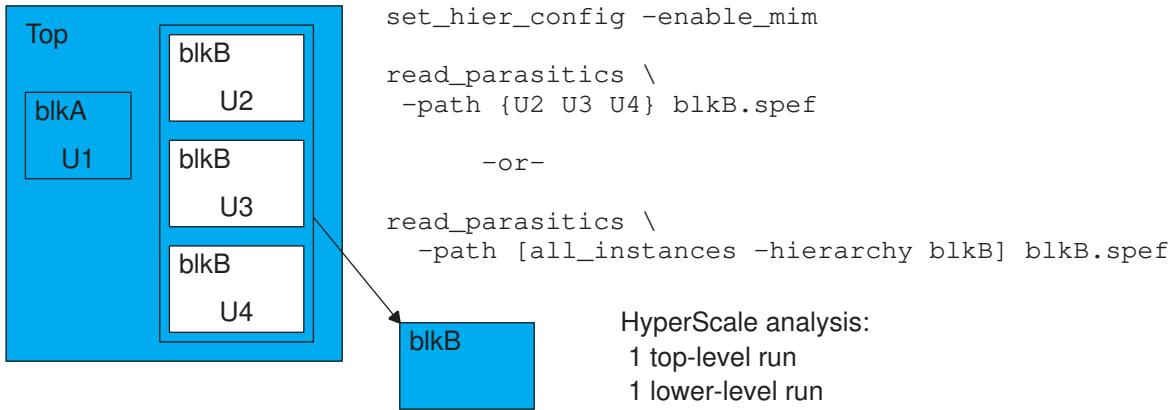
```
set_host_options -num_processes 2 -max_cores 4 myCore3 ...
restore_session -partitions {P2} my_session_dir
```

Any usage of the `set_host_options` command in the current session invalidates all host option settings of the original session.

MIM Merging in Distributed HyperScale Analysis

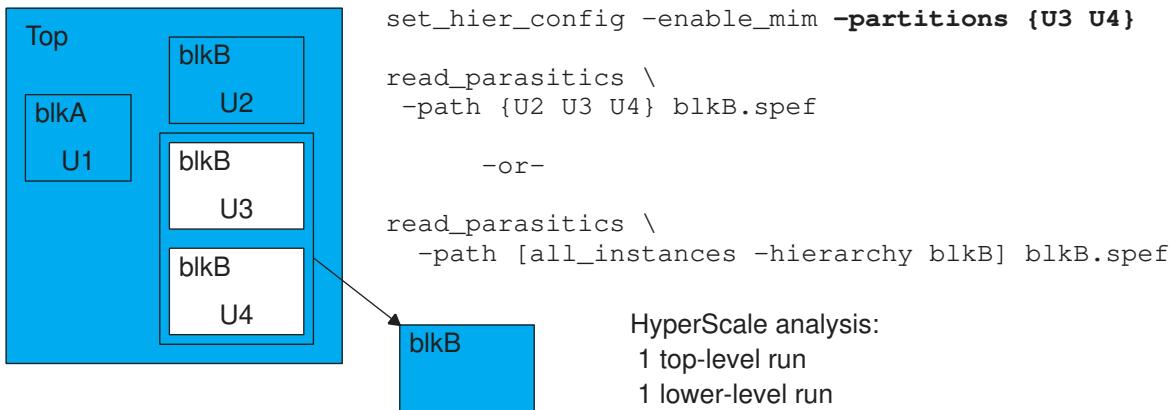
In the full-chip distributed analysis flow, you can perform MIM merging by using the `set_hier_config` command with the `-enable_mim` option, as shown in the following example.

Figure 462 MIM Merging for All Instances of a Block



If the hierarchical context for one block of a MIM set is very different from the rest, you might want to analyze that block with the top level. In that case, you explicitly specify the partitioning by using the `set_hier_config` command with the `-partitions` option, as shown in the following example.

Figure 463 MIM Merging for Some Instances of a Block



In this example, reading parasitics into instance U2 is not required, but to prevent possible errors in a full MIM merging configuration, You should read the parasitics into all instances of a MIM block.

HyperScale Attributes

Table 78 Cell Object HyperScale Attributes

Cell attribute	Description
is_active (boolean)	Returns true if the cell is considered an active object in HyperScale top-only distributed analysis (<code>set_hier_config -enable_top_only_flow</code>).

Table 79 Pin Object HyperScale Attributes

Pin attribute	Description
is_active (boolean)	Returns true if the pin is considered an active object in HyperScale top-only distributed analysis (<code>set_hier_config -enable_top_only_flow</code>).

Table 80 Port Object HyperScale Attributes

Port attribute	Description
is_active (boolean)	Returns true if the port is considered an active object in HyperScale top-only distributed analysis (<code>set_hier_config -enable_top_only_flow</code>).

Legacy Path-Based SMVA

Note:

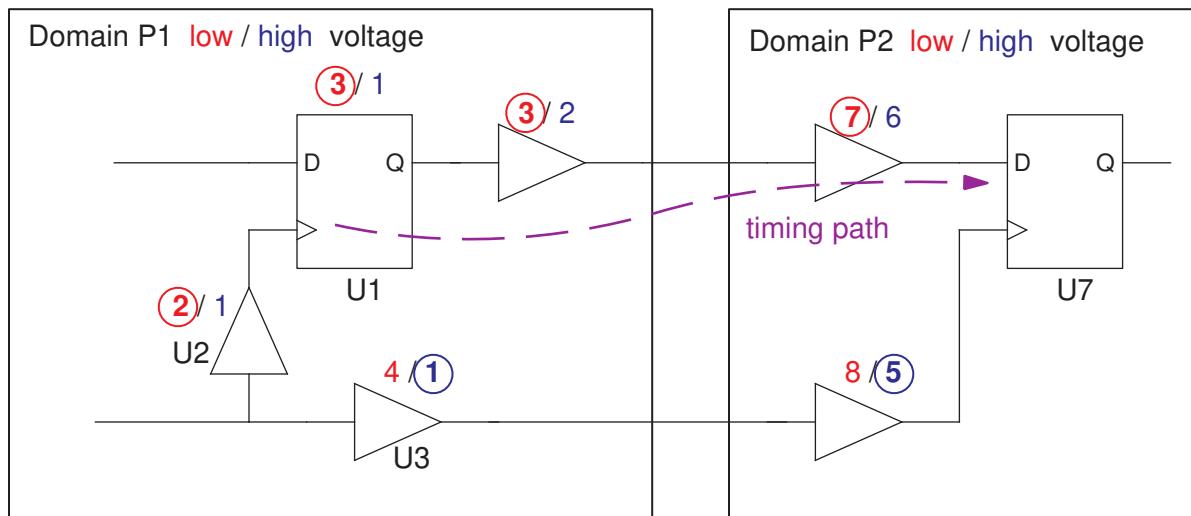
This is the original documentation for the legacy path-based-only SMVA feature, used when the `timing_cross_voltage_domain_analysis_mode` variable is set to `legacy`.

The current SMVA functionality, documented at [SMVA Graph-Based Simultaneous Multivoltage Analysis](#), supports both PBA and GBA 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 464, 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 464 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.

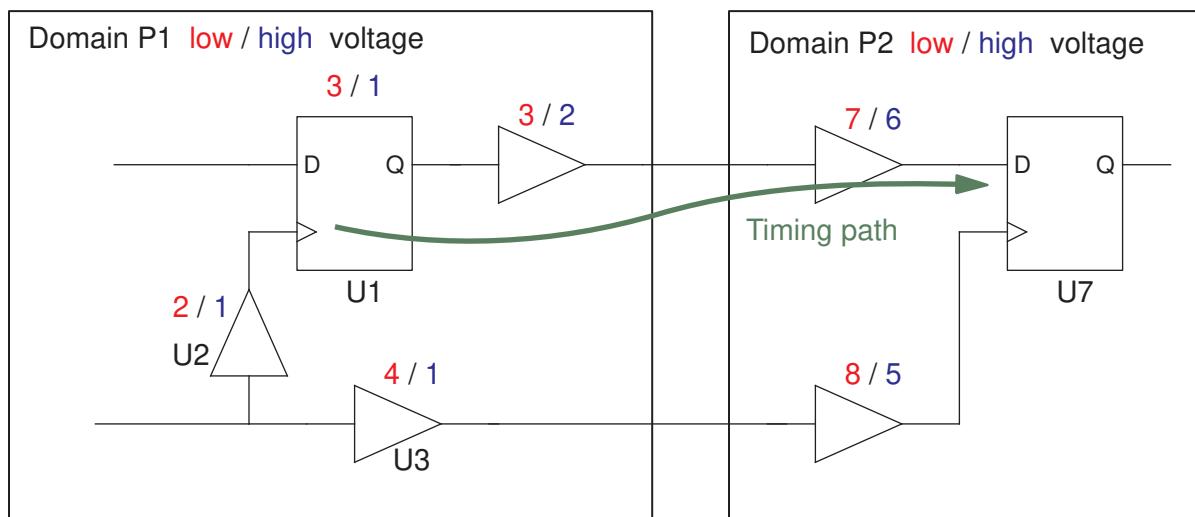
If timing violations are found in this analysis, you could reduce this pessimism by analyzing the design multiple times using every possible combination of fixed supply voltages, for example, with V1/V2 set to low/low, low/high, high/low, and high/high. However, a large number of power domains results 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 465](#) 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 465 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.

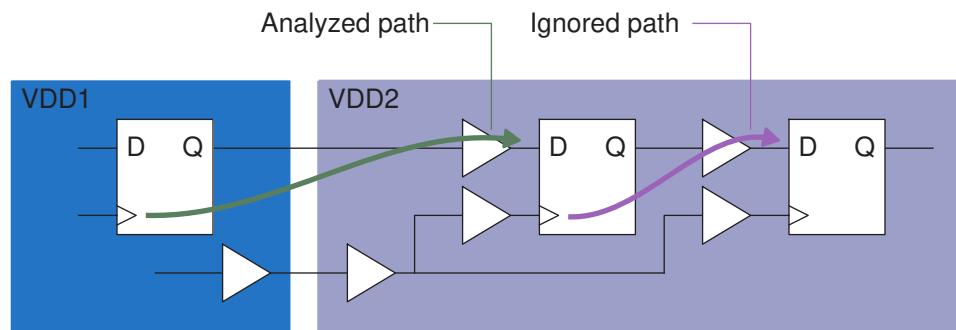
If the variable is set to `true`, during the next timing update, PrimeTime identifies paths that traverse multiple power domains and subsequently limits the reporting to only those

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:

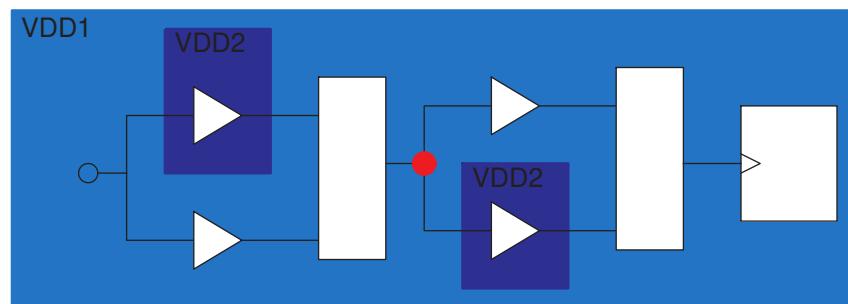
- Identifies and ignores paths that cross power domains before the CRPR common point, as shown in [Figure 466](#)

Figure 466 Analyzing Only Paths Affected by Domain Crossings



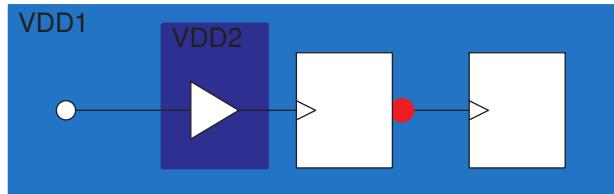
- Identifies reconvergent clock paths, as shown in [Figure 467](#)

Figure 467 Clock Path Reconverges



- Improves pessimism reduction for sequential clock networks when the path crosses power domains, as shown in [Figure 468](#)

Figure 468 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 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_dерate -early 0.92
set_timing_dерate -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
```

Appendix A: Deprecated Features

Legacy Path-Based SMVA

```
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 -pbamode path ...
```

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 with 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.

baseline image

In DMSA analysis, the image that is produced by combining the netlist image and the common data files for a scenario.

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 process 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.

command focus

In DMSA analysis, the 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.

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 tool. 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.

critical region

In HyperTrace accelerated PBA, the set of slack-critical (violating) pins in the design. The slack criticality threshold is typically zero, but it is configurable.

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 image

In DMSA analysis, the image that is automatically saved to disk when there are more scenarios than hosts, and the worker process must switch to work on another scenario.

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 Linux.

current session

In DMSA analysis, the current set of scenarios selected for analysis.

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 `async_default` paths.

database (.db) 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

A Synopsys 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](#).

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 a 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.

graph-based refinement

In HyperTrace accelerated PBA, the process of performing selective signal merging within the critical region to improve accuracy while maintaining the runtime and memory advantages of a graph-based representation.

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.

HyperGrid

A PrimeTime feature to analyze very large designs in a single run by parallelizing the analysis across multiple worker processes running on different hosts. The entire design can be explored and reported with the same commands used in a regular PrimeTime analysis.

HyperScale

A PrimeTime hierarchical analysis method that analyzes the block-level and top-level portions of the design using separate runs and accurately handles the timing interfaces across hierarchical boundaries.

HyperTrace

A PrimeTime technology that accelerates exhaustive path-based analysis (PBA). The critical region of the timing graph is identified. Then, graph-based refinement applies selective signal merging to improve accuracy within the critical region. This refined timing is then used to drive the exhaustive PBA search.

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 (ILM)

An obsolete type of timing model that has been superseded by HyperScale hierarchical analysis.

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 process 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.

logic library

A database containing information about the functional and timing characteristics of circuit elements used in a design, also called the .lib logic library when specified in Liberty syntax. Logic libraries are typically provided by an ASIC vendor.

Liberty

A standard format for specifying cell library information such as delays, timing constraints, and power. The SiliconSmart library characterization tool generates text files in Liberty (.lib) format. The Library Compiler tool reads in Liberty files and generates compiled cell library (.db) files.

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 in Liberty (.lib) format and compiles the description into a database in .db format for synthesis and analysis.

LVF (Liberty Variation Format)

A standard format in Liberty syntax for specifying cell variation parameters such as the mean and standard deviation (sigma) for cell delays.

manager process

In DMSA and HyperGrid distributed analysis, the PrimeTime process invoked by the user that orchestrates one or more worker processes.

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.

nonlinear delay model (NLDM)

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.

partition

In HyperGrid distributed analysis, part of a large design that has been split into multiple pieces (partitions), for parallelized analysis across multiple machines.

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 you define a procedure, you can run it 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 where you can 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, 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.

scenario

In DMSA analysis, a specific combination of operating conditions and operating modes under which to analyze a design.

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.

session

In a general context, refers to the currently running analysis session in the PrimeTime tool. Specifically, it can also refer to the data directory created and read by the `save_session` and `restore_session` commands, respectively.

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.

signal pin

A pin that is not a PG pin, such as a clock, data, or reset pin.

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.

task

In DMSA and HyperGrid distributed analysis, a self-contained piece of work defined by the manager process for a worker process to execute.

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.

transition time

The amount of time it takes for a signal to change from low to high or from high to low.

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.

worker process

In DMSA and HyperGrid distributed analysis, one of a set of processes that performs distributed analysis work. The worker processes are controlled by the manager processes.