

Synopsys® Timing Constraints and Optimization User Guide

Version U-2022.12-SP2, March 2023

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2025 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

Related Products, Publications, and Trademarks	12
Conventions	13
Customer Support	13
<hr/>	
1. Introduction to Synthesis Timing	15
Static Timing Analysis	15
Timing Paths	21
Clocks	25
Input and Output Delays	27
Delay Calculation	28
Flip-Flop and Latch Timing Checks	30
Timing Analysis in the Design Flow	34
Synopsys Design Constraint Commands	34
Library Timing Data	38
Design Compiler	40
Ideal Clocking	41
Wire Load Models and Topographical Technology	42
Design Partitioning	43
Path Groups	45
Register Retiming Optimization	47
IC Compiler	48
Design Planning	49
Placement	50
Clock Tree Synthesis	51
Routing	52
Timing Analysis After Routing	53
Synthesis Design Rules	55
<hr/>	
2. Clocks	57
Creating Clocks	57
Clock Network Effects	59
Clock Latency	61
Propagated Latency	61
Ideal Network Latency	62

Contents

Source Latency	63
Clock Uncertainty	64
Ideal Clock Transition Times	67
Reporting Clock Information	67
Multiple Clocks	68
Synchronous Clocks	69
Asynchronous Clocks	72
Exclusive Clocks	72
Clock Sense	74
Pulse Clocks	78
Minimum Pulse Width Checks	80
Clock-Gating Signal Timing Checks	81
Generated Clocks	86
Divide-by-2 Generated Clock	87
Generated Clock Based on Edges	87
Divide-by Clock Based on Falling Edges	88
Shifting the Edges of a Generated Clock	90
Combinational-Only Source Latency Calculation	91
Generated Clock Based on a Non-Unate Master Clock	92
Estimated I/O Latency	94
Calculating I/O Latency for Input Paths	96
Calculating I/O Latency for Output Paths	97
Propagated Clocks	97
 3. Timing Paths	100
Path Groups	100
User Grouping of Paths	101
Weight or Cost Function	101
Critical Range	101
Reporting Path Groups	102
Path Specification Methods	103
Through Arguments	103
Rise/Fall From/To Clock	104
Default Path Delay Constraints	107
Path Delay for Flip-Flops Using a Single Clock	107
Path Delay for Flip-Flops Using Different Clocks	109

Contents

Setup Analysis	110
Hold Analysis	110
Single-Cycle Path Analysis Examples	112
Timing Exceptions	114
False Path Exceptions	114
Maximum and Minimum Path Delay Exceptions	116
Multicycle Path Exceptions	116
Path Timing Margin Exceptions	122
Specifying Exceptions Efficiently	123
Exception Order of Precedence	125
Exception Type Priority	125
Path Specification Priority	125
Reporting Exceptions	127
Removing Exceptions	128
Preset and Clear Timing	128
Data-to-Data Checks	129
Specifying Data-to-Data Checks	130
Generating Timing Reports for Data Checks	132
Data Checks and Clock Domains	133
Library-Based Data Checks	133
<hr/>	
4. Operating Conditions	135
Operating Condition Definitions	135
Setting Operating Conditions	138
Operating Condition Modes	139
Minimum and Maximum Delay Calculations	140
Min-Max Cell and Net Delay Values	141
Setup and Hold Checks	142
Path Delay Tracing for Setup and Hold Checks	142
Setup Timing Check for Worst-Case Conditions	142
Hold Timing Check for Best-Case Conditions	143
Simultaneous Best-Case/Worst-Case Conditions	144
Path Tracing in On-Chip Variation Mode	144
Using Two Libraries for Min-Max Analysis	145
Setting On-Chip Variation Derating Factors	145
Advanced On-Chip Variation Analysis	148
How AOCV Analysis Works	148

Contents

Cell Adjustment Factor	150
AOCV Flow	150
AOCV Data Formats	150
Library-Based AOCV Data	151
File-Based AOCV Data	152
Advanced On-Chip Variation Reporting	154
Fast Path-Based AOCV	154
Parametric On-Chip Variation Analysis	155
How POCV Analysis Works	155
Parametric On-Chip Variation Flow	157
POCV Data Formats	157
Library-Based POCV Data	158
File-Based POCV Data	159
Saving Scenario-Specific POCV Data	161
Enabling Parametric On-Chip Variation Analysis	161
Reading File-Based Parametric On-Chip Variation Data	162
POCV Guard Band	163
Scaling the POCV Coefficients	164
Enabling Constraint Variation	164
Reporting POCV Information	164
Reporting the POCV Analysis Results	166
Voltage and Temperature Scaling Between Libraries	167
Clock Reconvergence Pessimism Removal	168
On-Chip Variation Example	169
Reconvergent Logic Example	170
Setting Clock Reconvergence Pessimism Removal	170
Transparent Latch Edge Considerations	172
Reporting CRPR Calculations	173
<hr/>	
5. Timing Constraints	174
Input Delays	174
Excluding Clocks	177
Reference Pin	178
Output Delays	178
Drive Characteristics at Input Ports	180
Setting the Port Driving Cell	180
Setting the Port Drive Resistance	183

Contents

Setting a Fixed Port Transition Time	183
Removing Drive Information From Ports	183
Port Load Capacitance	184
Ideal Networks	185
Propagation of the Ideal Network Property	185
Creating Ideal Networks	186
Removing Ideal Networks	188
Reporting Ideal Networks	189
Retrieving Ideal Objects	189
Setting Ideal Latency and Ideal Transition Time	190
Case Analysis	191
Reporting Case Analysis	193
Removing Case Analysis	194
Constant Propagation Log File	194
Usage Example	196
Mode Analysis	196
Mode Groups	196
Setting Modes Using Case Analysis	197
Setting Modes Directly on Cells	198
Reporting Modes	199
Wire Load Models	199
Net Capacitance, Resistance, and Area Calculation	200
Choosing a Wire Load Model	201
Wire Load Model Modes	202
Setting a Wire Load Model	203
Local Link Library Usage	204
Setting a Wire Load Selection Group	205
Wire Load Models for Hierarchical Cells	205
Selecting the Wire Load Model Automatically	205
Defining the Minimum Block Size	206
Reporting Wire Load Models	207
Timing Loops	209
Breaking Feedback Loops Manually	210
6. Back-Annotation	212
Back-Annotating Delays and Timing Checks	212

Contents

Delay Calculations	213
Back-Annotating Timing Information From an SDF File	213
Instance Names	214
Cell and Pin Names	214
DESIGN Field Names	214
Supported SDF Constructs	214
Back-Annotating Timing From a Subdesign Timing File	219
Back-Annotating Load Delay	219
Back-Annotating for Worst Timing Delay	219
Back-Annotating Timing Checks	220
Back-Annotating for Conditional Arc Cell Delay	220
Back-Annotation Order of Precedence	221
Examples	221
Writing an SDF File	222
Annotating Delays and Timing Checks From the Command Line	222
Defining Net and Cell Delays	222
Defining Timing Check Values Between Pins	225
Reporting Annotated Values and Checks	226
Reporting Load and Resistance Values	226
Reporting Annotated Delay Values	227
Removing Back-Annotated Values	229
Removing Annotated Delay Values	229
Removing Annotated Timing Checks Between Specific Pins	230
Removing Annotated Resistance or Capacitance Values	231
Removing Back-Annotated Values Command Summary	232
Setting Net Load	232
Setting Net Resistance	234
Setting Pi Model Capacitance and Resistance	235
Back-Annotating Detailed Parasitics	236
RTL Load Annotation With Wire Load Modeling	237
Default Resistance Values	241
RTL Load Buffering	241
Extraction of RTL Loads	241
7. Timing Reports	243
Reporting Commands	243
check_timing Command	246
report_constraint Command	248

Contents

report_timing Command	251
report_timing Report Contents	251
report_timing Command Options	254
Scope of the Design Reported	255
Path Details Reported	256
Delay Type (Min/Max) Reported	258
Number of Paths Reported	259
Other Options	259
report_delay_calculation Command	259
get_timing_paths Command	260
report_clock_timing Command	263
Latency and Transition Time Reporting	263
Skew Reporting	264
Interclock Skew Reporting	266
Clock Timing Reporting Options	266
Summary Report	267
List Report	269
Verbose (Path) Report	270
<hr/>	
8. Graphical User Interface	272
Using the GUI for Timing Analysis	272
Timing Menu Commands	275
IC Compiler Layout Window Timing Menu Commands	276
Path Slack Histogram	277
Path Inspector	279
Path Schematic	282
Path Element Tables	283
Path Delay Profiles	284
Timing Analysis Driver	285
<hr/>	
9. Timing in Latch-Based Designs	291
Time Borrowing in Latch-Based Designs	291
A Simple D Latch	291
About Time Borrowing	293
Time Borrowed and Time Given	295
Balancing Relative Slacks	296
Determining Relative Slack	296

Contents

Optimizing Near-Critical Paths	296
Reducing Delay Costs	299
Automatic Slack Distribution During Optimization	302
Normalized Slack Analysis	302
Enabling Normalized Slack Analysis	303
Reporting Normalized Slack	303
Using Normalized Slack to Adjust the Clock Period	304
Setting Limits for Normalized Slack Analysis	304
Constraining a Latch-Based Design	304
Creating Non-Overlapping Clocks	305
What Are Two-Phase Designs?	305
What Are Non-Overlapping Clocks?	306
Path Timing Report With Borrowing	306
Latch-Based Time-Borrowing Example	310
Linear Block Encoder and Decoder	311
Noisy Channel	312
Linear Block Decoder for Single-Bit Error	312
Linear Block Encoder and Decoder Implementation	312
Setting Constraints on the Linear Block Encoder	316
report_timing Command Output	317
Advanced Latch Timing Analysis	322
Enabling Advanced Latch Analysis	323
Breaking Loops	323
Specifying Loop-Breaker Latches	324
Finding Loop-Breaker Latches	324
Latch Loop Groups	324
Listing Collections of Latch Loop Groups	325
Reporting Latch Loop Groups	325
Specifying the Maximum Number of Latches Analyzed Per Path	326
Timing Exceptions Applied to Latch Paths	326
False Path Exceptions	327
Multicycle Path Exceptions	327
Maximum and Minimum Delay Exceptions	328
Clock Groups	328
Specification of Exceptions on Throughpaths	330
Reporting Paths Through Latches With report_timing	330
Calculation of the Worst and Total Negative Slack	332
A. Static Timing Delay Calculation	334

Contents

Path-Based Timing Optimization	334
Reporting Retain Arcs	336
Reporting Arc Delay Calculations	337
Debugging Delay Calculations Along a Critical Path	338
Reporting Details of a Cell Delay Arc	339
Delay Models	340
Linear Delay Model	340
Slope Delay	342
Intrinsic Delay	343
Transition Time	343
Connect Delay	343
Delay Calculation (Linear) Example	345
CMOS2 Delay Model	346
Edge Rate Delay	349
Intrinsic Delay	349
Transition Time	349
Connect Delay	350
Delay Calculation (CMOS2) Example	351
Nonlinear Delay Model	355
Library Cell Timing Arcs	356
Delay Calculation Example	357
Environmental Scaling	359
Waveform Propagation Using CCS Models	359

About This User Guide

The *Synopsys Timing Constraints and Optimization User Guide* describes the usage of timing constraints and timing analysis in the Design Compiler® and IC Compiler™ tools for the synthesis, optimization, and physical implementation of integrated circuits. Most of the information in this book applies to both the Design Compiler and IC Compiler tools, and much of it applies to the PrimeTime® static timing analysis tool as well.

This manual is intended for logic designers and engineers who use the Synopsys synthesis and physical implementation tools to design ASICs, ICs, and FPGAs. Readers should already be familiar with UNIX or Linux and basic IC design principles.

This preface includes the following sections:

- [Related Products, Publications, and Trademarks](#)
 - [Conventions](#)
 - [Customer Support](#)
-

Related Products, Publications, and Trademarks

For additional information, 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 Vision™
- DesignWare® components
- DFT Compiler
- HDL Compiler™
- Milkyway™ Environment
- PrimeTime
- Power Compiler™

Also see the following related document:

- *Using Tcl With Synopsys Tools*

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.
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 Synthesis Timing

Timing is a major consideration in the synthesis and physical implementation of synchronous digital circuits. Design Compiler and IC Compiler perform static timing analysis to drive the selection of logic gates, the implementation of logic, the placement of logic cells, and the routing of interconnections. Using a signoff timing analysis tool such as PrimeTime is necessary to verify that the completed design will work at the intended clock speed. All of these tools share some of the same timing analysis commands and operating principles, as described in the following sections:

- [Static Timing Analysis](#)
 - [Timing Analysis in the Design Flow](#)
 - [Synthesis Design Rules](#)
-

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 under worst-case conditions. It considers the worst possible delay through each logic element, but not the logical operation of the circuit.

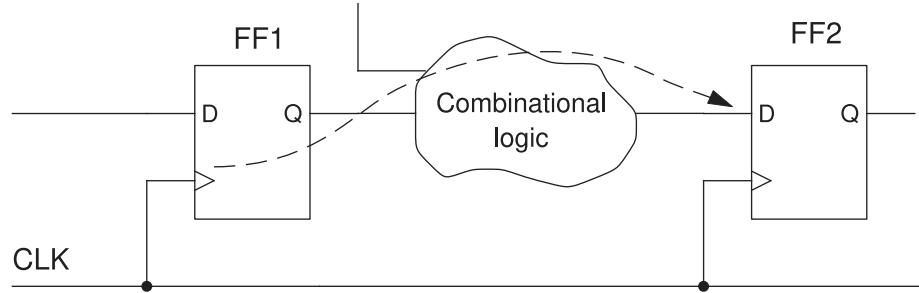
In comparison to circuit simulation, static timing analysis is faster and more thorough. It is faster because it does not need to simulate multiple test vectors. It is more thorough because it checks the worst-case timing for all possible logic conditions, not just those sensitized by a particular set of test vectors. However, static timing analysis checks the design only for proper timing, not for correct logical functionality.

Timing, area, and power constraints drive the operation of synthesis with Design Compiler and physical implementation with IC Compiler. These tools synthesize the netlist and perform physical placement and routing with the goal of making the fastest device, using the least area and power, in the shortest turnaround time that is consistent with the design requirements. These tools perform trade-offs between speed, area, power, and runtime according to the constraints set by the designer. However, a chip must meet the timing constraints to operate at the intended clock rate, so timing is the most important design constraint.

Static timing analysis seeks to answer the question, “Will the correct data be present at the data input of each synchronous device when the clock edge arrives, under all possible

conditions?" This concept is illustrated by the circuit in [Figure 1](#) and the timing diagram in [Figure 2](#).

Figure 1 Timing Path Example



In [Figure 1](#), the dashed arrow represents a timing path. The change in signal data caused by a clock transition at flip-flop FF1 must be propagated to flip-flop FF2 before the following clock edge arrives at FF2, so that the logically processed data can be reliably latched into FF2. The change at FF1.Q might affect the output of the combinational logic cloud at FF2.D, depending on the logic itself, the data value, and the values of any side inputs feeding into the logic. The change at FF2.D, if any, must occur before the next clock edge arriving at FF2.

Figure 2 Setup Check Timing

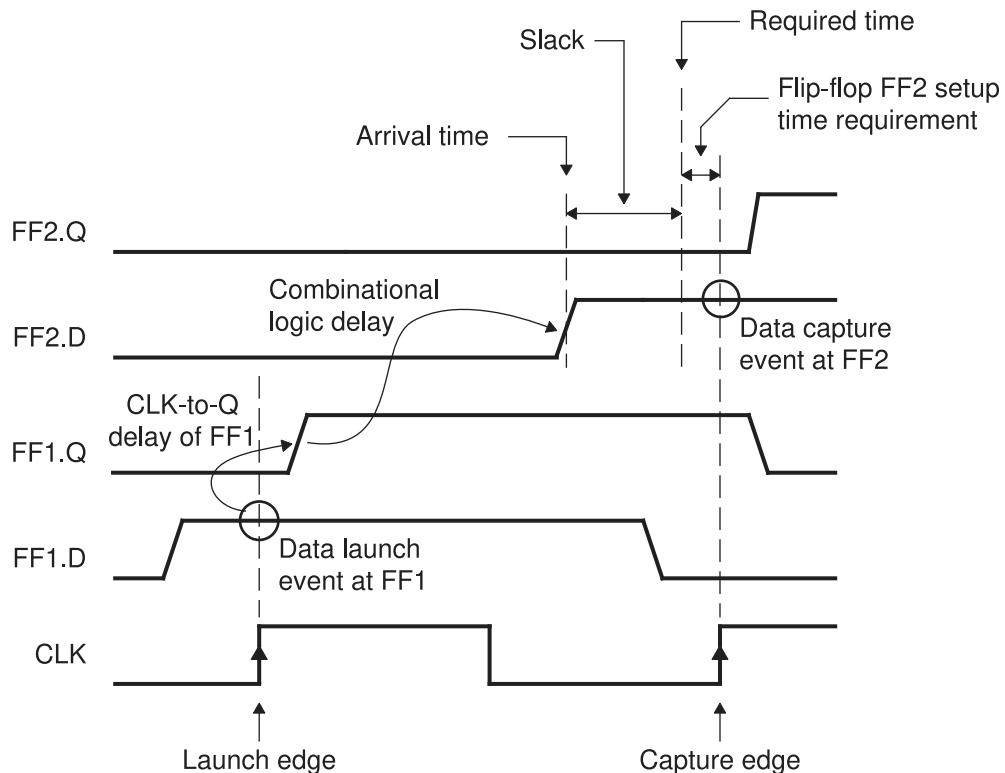


Figure 2 shows the timing for this path. The arrival of a clock edge at FF1 latches the data at the input FF1.D into the flip-flop. It also places that data on the flip-flop output, FF1.Q, after the clock-to-Q delay of the flip-flop. This is called the *launch* event for the timing path.

This signal goes through the combinational logic with some delay. The output of the combinational logic is at the input of the second flip-flop, FF2.D. The time at which the signal value changes here is called the *arrival time* for the path.

The change in value at FF2.D must occur before the arrival of the clock edge arriving at FF2, by at least an amount equal to the setup time requirement for the flip-flop. This latest allowable arrival time is called the *required time* for the path. The latching of data at FF2 is called the *capture* event for the timing path. In this example, the capture event occurs one whole clock cycle after the launch event.

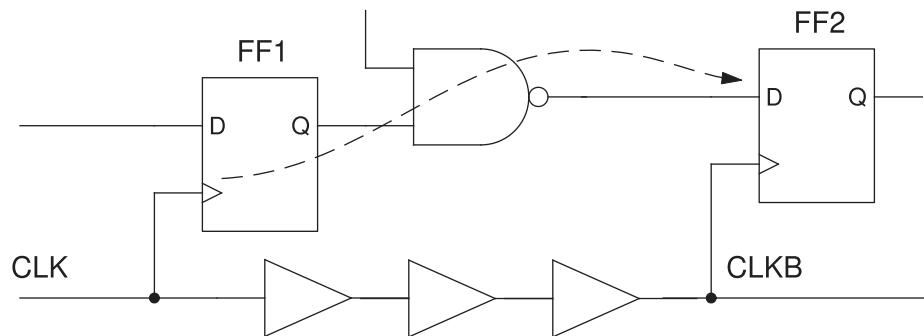
The amount of time by which the timing constraint is met is called the *slack* of the timing check. If the signal arrives earlier than necessary as shown in **Figure 2**, the slack is positive. If the signal arrives exactly at the required time, the slack is zero and the timing constraint is barely met. If the signal arrives later than the required time, the slack is

negative. In all three cases, the amount of slack is the required time minus the arrival time. For example, if the required time is 1.8 ns after the launch clock edge and the arrival time is 1.6 ns after the launch clock edge, the slack is 1.8 minus 1.6, or 0.2 ns, a positive number.

The preceding timing check is called a *setup* check, which verifies that a change in data arrives soon enough before each clock edge at the sequential device. This is the most common type of timing check that drives synthesis and optimization. However, other types of timing checks can be performed as well.

For example, a *hold* check verifies that the data remains valid during and after the arrival of the clock edge at the end of the path by a sufficient amount. A hold violation can occur if the shortest possible combinational delay from launch to capture is very short and the longest delay from the launch clock edge to the capture clock edge is very long. An example is shown in [Figure 3](#) and [Figure 4](#).

Figure 3 Hold Check Timing Path Example



In [Figure 3](#), the timing path has a very short combinational delay from FF1 to FF2, consisting of a single NAND gate. Meanwhile, there is a long delay for the clock signal between the two flip-flops because of the three buffers, possibly made even longer by a large RC delay due to a long route. Therefore, the capture clock signal CLKB arriving at FF2 is significantly delayed with respect to the launch clock CLK at FF1.

Figure 4 Hold Check Timing

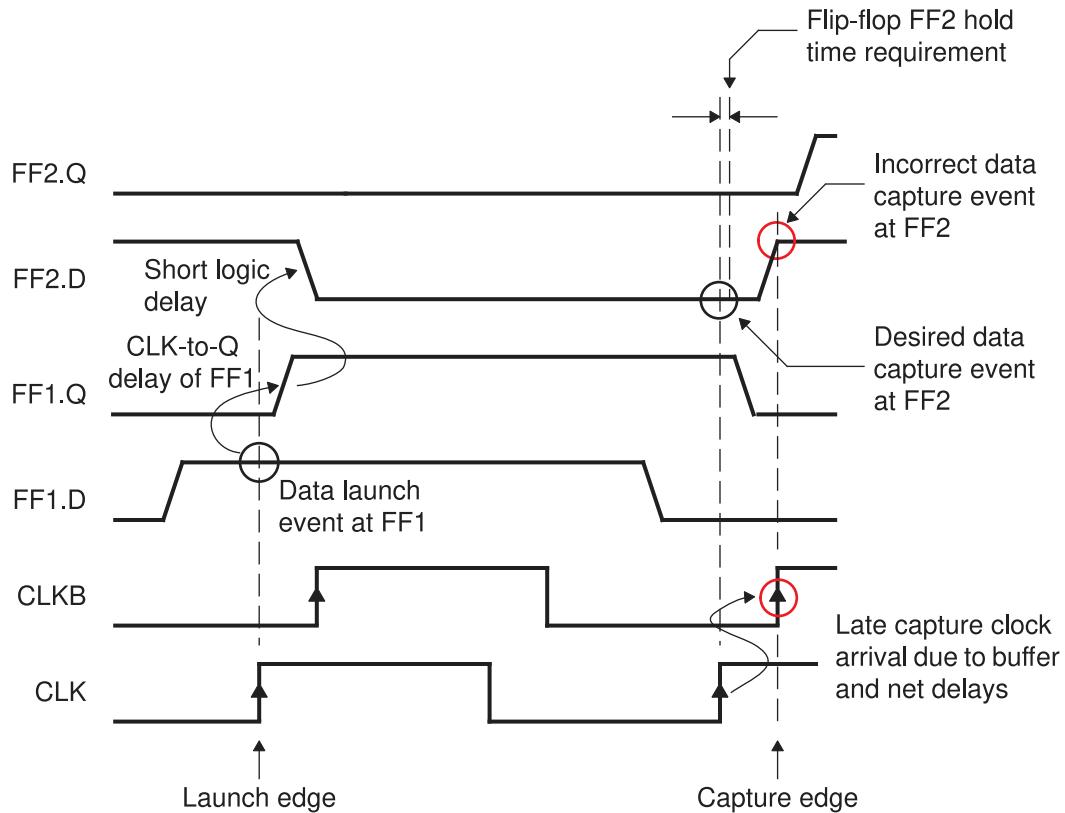


Figure 4 shows the possible timing in this situation. The setup constraint is met easily because the data arrives well before the required setup time. However, the hold constraint is not met because the data at the D input of FF2 is not held long enough after the nominal clock arrival time. The data changes before the delayed clock CLKB has a chance to latch it in. This type of violation can be fixed by shortening the delay in the clock line or by increasing the delay in the data path.

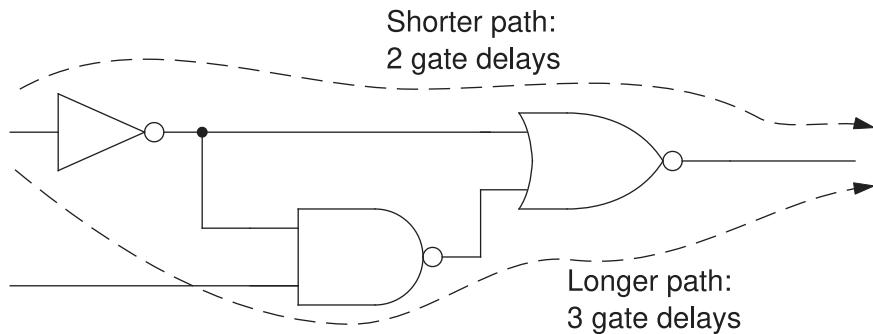
By default, the synthesis or implementation tool fixes setup violations, but not hold violations, because setup requirements are more difficult to satisfy. To have the tool fix hold violations as well as setup violations, use the `set_fix_hold` command. This command sets the `fix_hold` attribute on specified clocks, which directs the tool to check for and fix hold violations during the compile operation.

Each type of timing check considers different worst-case conditions. For example, a setup check considers the longest and slowest possible path through the combinational logic of the data path and the earliest possible arrival of the capture clock edge with respect to the launch clock edge. Conversely, a hold check considers the shortest and fastest possible

path through the combinational logic of the data path and the latest possible arrival of the capture clock edge with respect to the launch clock edge.

[Figure 5](#) shows examples of different pathways that can be taken through the same block of combinational logic. In a data path, a setup check would consider the longer delay through three gates, whereas a hold check would consider the shorter path through two gates.

Figure 5 Multiple Paths Through Combinational Logic

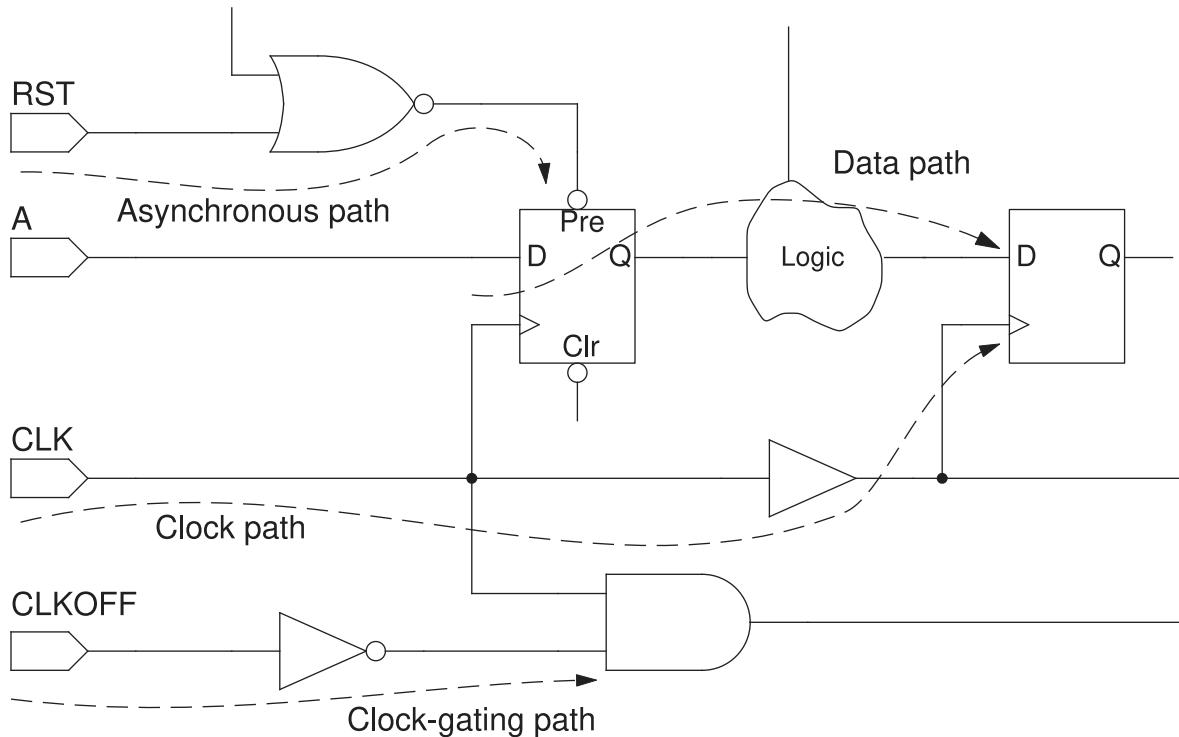


A synthesis, optimization, or analysis tool can perform timing checks on the following types of paths:

- 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
- Data-to-data check (a custom timing check specified with the `set_data_check` command, having specified setup and hold times between data signals)

Some of these path types are shown in [Figure 6](#).

Figure 6 Path Types



A tool can also perform timing checks on the asynchronous preset and clear inputs of sequential devices. A *recovery* check verifies that enough time has passed after the inactivation of an asynchronous control signal before a clock edge occurs. A *removal* check verifies that enough time has passed after a clock edge before the removal of an asynchronous control signal. You enable these timing checks by setting the `timing_disable_recovery_removal_checks` variable to `false` or by setting the `enable_recovery_removal_arcs` variable to `true`. These two variables are logically opposite each other; setting one variable automatically sets the other to the opposite state.

A *clock-gating* check is a setup or hold check performed on the control input of a clock-gating cell. This type of check detects occurrences of clipped clocked edges or spurious clock pulses. The command for specifying clock-gating checks is `set_clock_gating_check`.

Timing Paths

The timing analysis tool finds and analyzes all of the timing paths in the design. Each path has a startpoint and an endpoint. The startpoint is a place in the design where data is

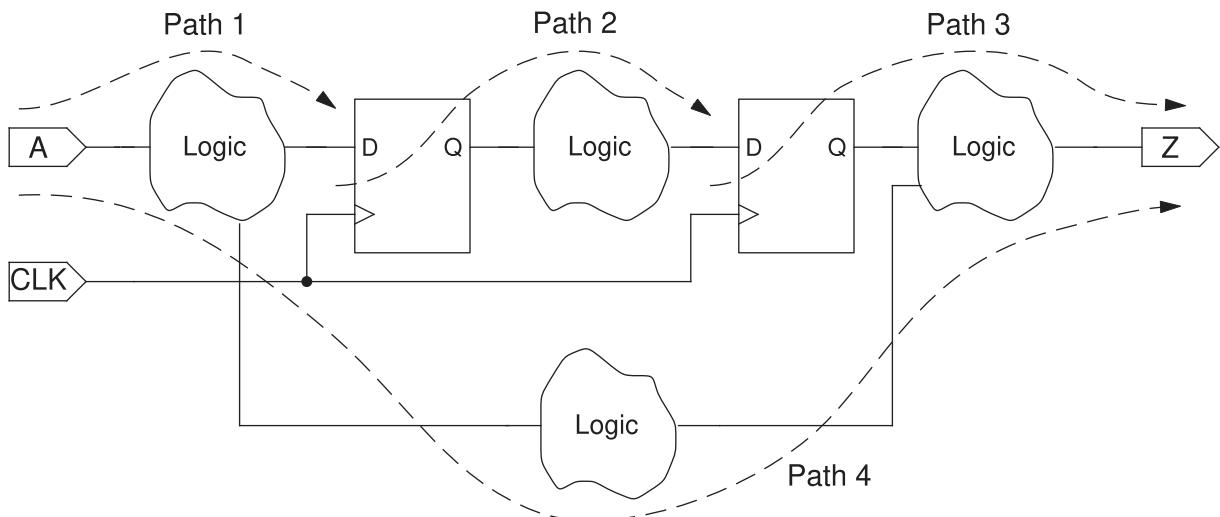
launched by a clock edge. The data is propagated through combinational logic in the path and then captured at the endpoint by another clock edge.

The startpoint of a path is a clock pin of a sequential element or an input port of the design. The arrival of an active clock edge at a startpoint launches data through the path. An input port can be considered a startpoint due to the arrival of data launched by an external source.

The endpoint of a path is a data input pin of a sequential element or an output port of the design. The arrival of an active clock edge at the clock input of the capture device captures data at the end of the path. An output port can be considered an endpoint due to the external capture of the data leaving the output port.

[Figure 7](#) shows an example of a design and its timing paths.

Figure 7 Timing Path Types



Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point:

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

Each path in the design has an associated timing slack. The slack is a time value that can be positive, zero, or negative. The single path having the worst slack is called the *critical path*. This is the path that has the most negative slack, or if there are no timing violations, the one with the smallest slack among all paths in the design. The plural term, *critical paths*, means the set of n paths having the worst slack among all paths in the design.

You can optionally divide the paths of a design into groups so that timing analysis, reporting, and optimization are done separately for the designated *path groups*. For example, you might put the input-to-register, register-to-register, and register-to-output paths into three separate groups because they have different types of timing constraints. By default, there is one path group for each clock used in the design.

The following is a typical path timing report generated by the `report_timing` command in IC Compiler. A similar report is produced by the `report_timing` command in Design Compiler or PrimeTime.

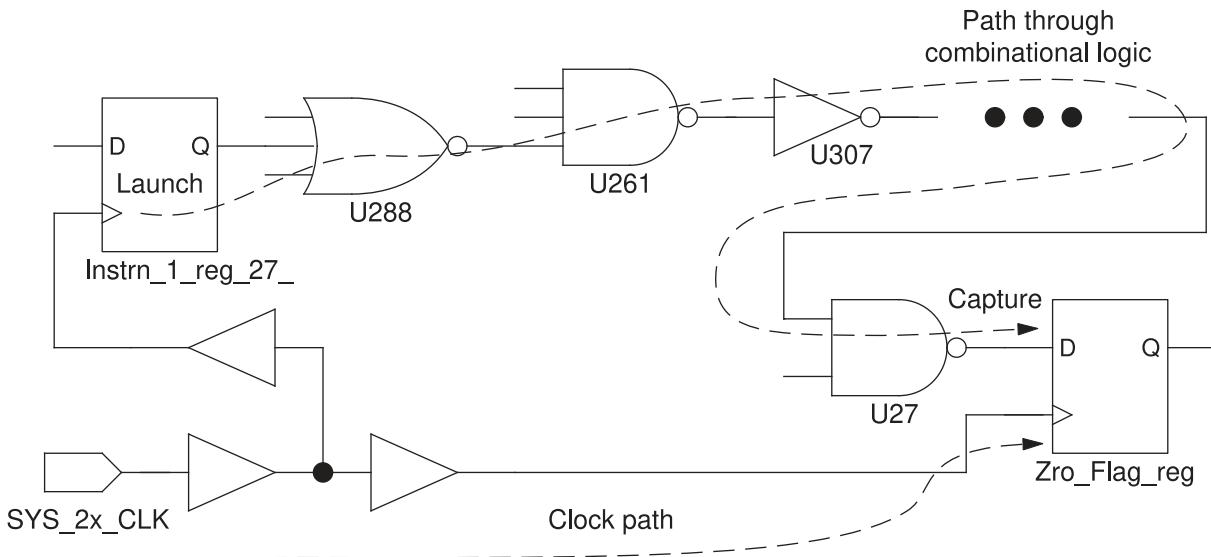
```
Startpoint: I_RISC_CORE/I_INSTRN_LAT/Instrn_1_reg_27_
            (rising edge-triggered flip-flop clocked by SYS_2x_CLK)
Endpoint: I_RISC_CORE/I_ALU/Zro_Flag_reg
            (rising edge-triggered flip-flop clocked by SYS_2x_CLK)
Path Group: SYS_2x_CLK
Path Type: max
```

Point	Incr	Path
clock SYS_2x_CLK (rise edge)	0.00	0.00
clock network delay (propagated)	0.51	0.51
I_RISC_CORE/I_INSTRN_LAT/Instrn_1_reg_27/CP (senrq1)	0.00	0.51 r
I_RISC_CORE/I_INSTRN_LAT/Instrn_1_reg_27/Q (senrq1)	0.62	1.13 f
I_RISC_CORE/I_INSTRN_LAT/Instrn_1[27] (INSTRN_LAT)	0.00	1.13 f
I_RISC_CORE/I_ALU/ALU_OP[3] (ALU)	0.00	1.13 f
I_RISC_CORE/I_ALU/U288/ZN (nr03d0)	0.36 *	1.49 r
I_RISC_CORE/I_ALU/U261/ZN (nd03d0)	0.94 *	2.43 f
I_RISC_CORE/I_ALU/U307/ZN (invbd2)	0.35 *	2.78 r
I_RISC_CORE/I_ALU/U343/Z (an02d1)	0.16 *	2.93 r
I_RISC_CORE/I_ALU/U344/ZN (nr02d0)	0.11 *	3.04 f
I_RISC_CORE/I_ALU/U348/ZN (nd03d0)	0.28 *	3.32 r
I_RISC_CORE/I_ALU/U355/ZN (nr03d0)	0.29 *	3.60 f
I_RISC_CORE/I_ALU/U38/Z (an02d1)	0.15 *	3.75 f
I_RISC_CORE/I_ALU/U40/Z (an02d1)	0.12 *	3.87 f
I_RISC_CORE/I_ALU/U48/ZN (nd02d1)	0.06 *	3.93 r
I_RISC_CORE/I_ALU/U27/ZN (nd02d1)	0.06 *	3.99 f
I_RISC_CORE/I_ALU/Zro_Flag_reg/D (secrq4)	0.00 *	3.99 f
data arrival time		3.99
clock SYS_2x_CLK (rise edge)	4.00	4.00
clock network delay (propagated)	0.47	4.47
clock uncertainty	-0.10	4.37
I_RISC_CORE/I_ALU/Zro_Flag_reg/CP (secrq4)	0.00	4.37 r
library setup time	-0.37	4.00
data required time		4.00

data required time	4.00
data arrival time	-3.99
slack (MET)	0.01

By default, the `report_timing` command reports the worst setup path in each path group. In this example, the logic associated with the reported path is shown in [Figure 8](#).

Figure 8 Timing Path Logic



The report starts by showing the path startpoint, path endpoint, path group name, and path timing check type. In this example, the path timing check type is shown as “max,” which means a maximum-delay or setup check; “min” would mean a minimum-delay or hold check.

A large table shows point-by-point accounting of the delays along the path from the startpoint to the endpoint. The table has columns labeled Point, Incr, and Path. These columns list the points (cell pins) along the path, the incremental contribution to the delay at each point, and the cumulative delay up to that point, respectively. Hierarchical boundary crossings are listed as well, showing zero incremental delay at each crossing.

The asterisk (*) symbols in the Incr column indicate where SDF back-annotated delay values were used for the net delay. The letters “r” and “f” in the Path column indicate the sense of the signal transition, either rising or falling, at that point in the path.

The path starts with the launch clock edge and ends at the data input of the capture device. The “data arrival time” shown in the table is the amount of elapsed time from

the source of the launch clock edge to the arrival of data at the endpoint, taking into consideration the longest possible delays along the path.

After this is the accounting for the required arrival time. The “data required time” shown in the table is the latest allowable arrival time for the data at the path endpoint, taking into account the nominal capture clock edge time, the clock network delay, the clock uncertainty, the least possible delay along the clock path, and the library setup time requirement for the capture device. The required time is subject to adjustment for clock reconvergence pessimism removal (CRPR).

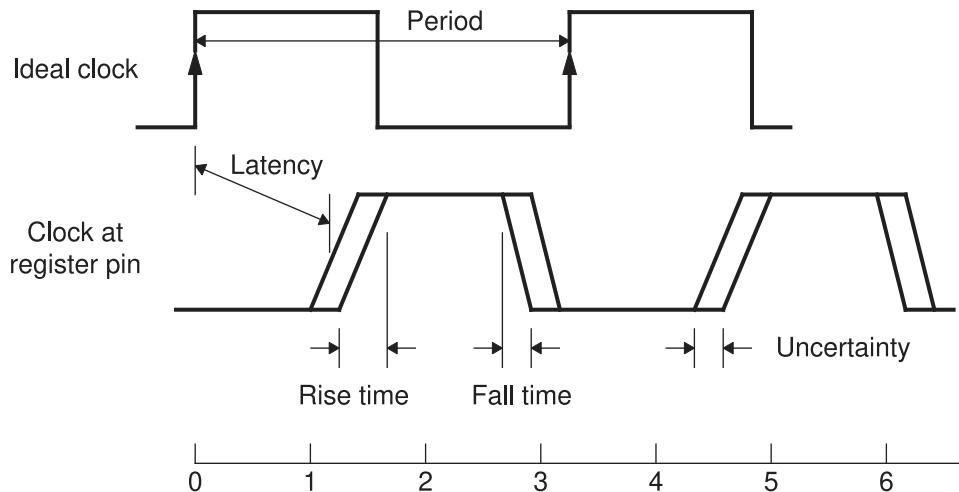
The slack value shown at the end of the report is simply the data required time minus the data arrival time. The slack is the amount of time by which the timing constraint is met, considering the latest possible arrival of data at the endpoint and the earliest possible arrival of the capture clock edge.

In this example, the slack is a very small positive value, which means that the timing constraint is barely met. A negative slack would require a change in the design to fix the violation. For example, a driver in the path could be replaced with a larger cell with a higher drive strength, which would reduce the net delay. On the other hand, a large positive slack would offer opportunities for optimization. For example, drivers in the path could be replaced with slower, smaller cells to reduce the area or with slower, higher-threshold cells to reduce leakage power.

Clocks

To perform timing analysis, you must specify the period of the clock or clocks used in the design and possibly the transition time, waveform, latency, uncertainty, relative skew, and other timing characteristics, as shown in [Figure 9](#). The timing analysis takes all of these clock characteristics into account to determine the worst possible conditions for each type of timing check. The command for specifying clocks is `create_clock`.

Figure 9 Clock Characteristics



If multiple clocks are used in the design, you must specify the timing relationships between those clocks so that the analysis tool can check the timing of a path launched by one clock and captured by another. By default, the tool assumes that after the occurrence of a launch event at a path startpoint, the very next clock edge that can occur at the path endpoint should capture the data at the end of the path. Any exceptions to this basic assumption must be specified as *timing exceptions*. Some examples of timing exceptions are *false paths*, *multicycle paths*, and explicit *minimum-delay paths* and *maximum-delay paths*:

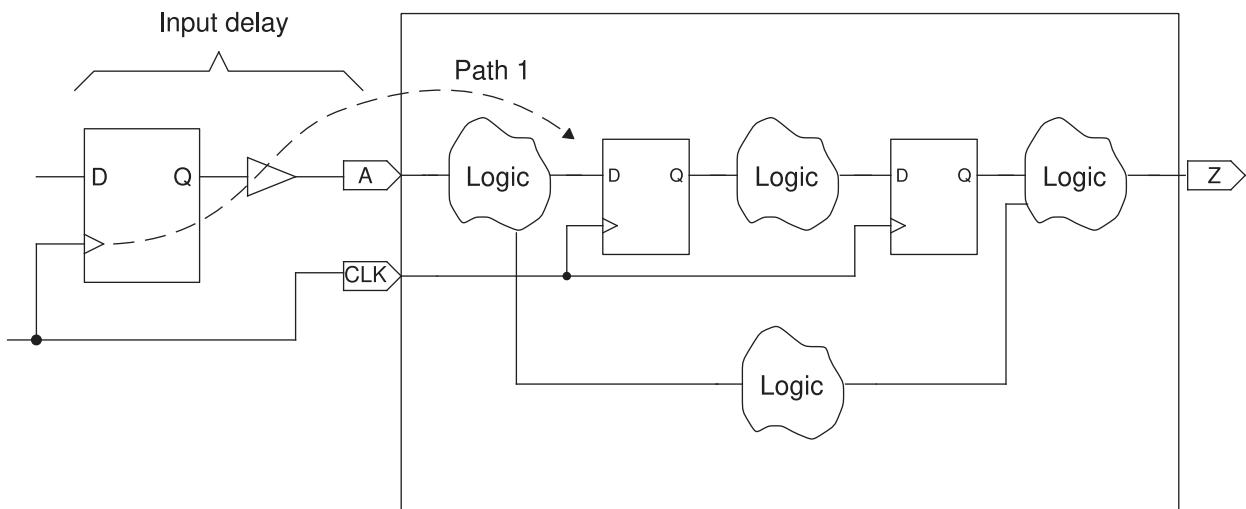
- A *false path* is a path that physically exists in the design but never propagates data from the startpoint to the endpoint due to the logic configuration, expected data sequence, or operating mode.
- A *multicycle path* is a path designed to take more than one clock cycle from launch to capture.
- A *minimum-delay path* or *maximum-delay path* is a path that must meet a delay constraint that you specify explicitly as a time value.

A clock signal typically has a very large fanout to many sequential devices. Therefore, each clock is usually implemented as a branching tree of buffers organized in a multilevel hierarchy. Clock tree synthesis is typically performed at the placement and routing stage, not at the logic synthesis stage, because placement and routing provide the interconnect length information needed to build the tree. To perform timing analysis before clock tree synthesis and layout, the logic synthesis tool must estimate the delays from the clock source to the clock pins of sequential devices.

Input and Output Delays

The analysis tool determines the timing slack for a path by comparing the startpoint-to-endpoint delay with the time span between the launch and capture clock edges. However, for a path starting at an input to the design, the arrival time of the data cannot be determined from the design netlist. This is because the external launch time and logic delays are unknown. Therefore, to analyze the input-to-register timing, you must specify the external timing conditions leading up to the input as an input delay. This concept is illustrated in [Figure 10](#).

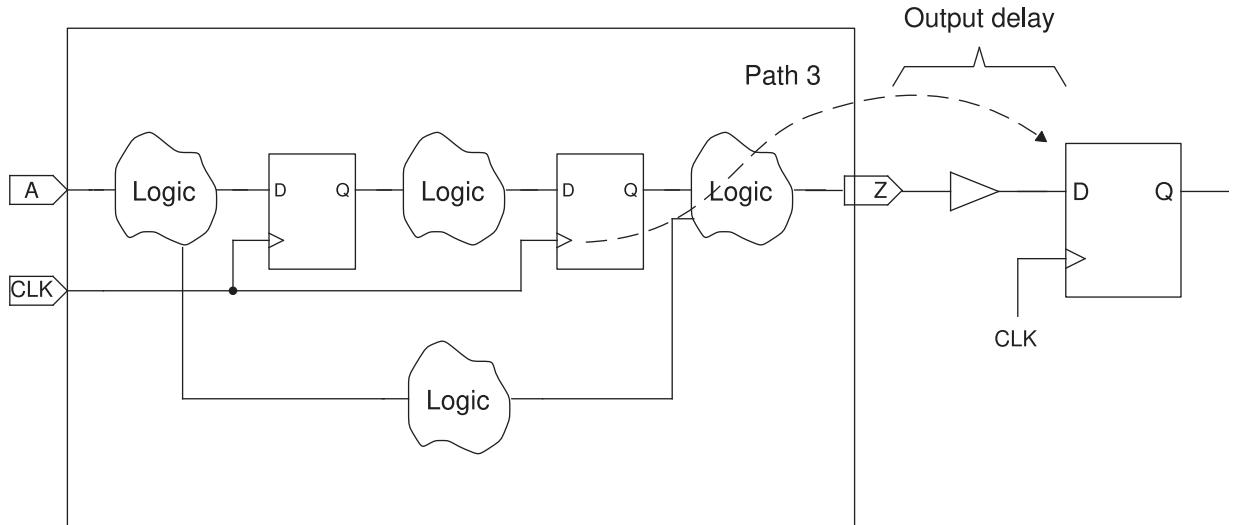
Figure 10 External Input Delay



The design being analyzed is enclosed in the dashed rectangle. The data arriving at input port “A” is launched by an external sequential device clocked by CLK. Then the data passes through some combinational logic before reaching input “A.” To analyze the timing of Path 1, you need to specify the amount of delay from the clock edge to the arrival of data at input “A.” In this example, the input delay is the CLK-to-Q delay of the external flip-flop plus the delay of the external buffer. You can also specify the drive characteristics of the external buffer for a more accurate calculation of the interconnect delay from input “A” to the combinational logic inside the design.

Similarly, for a path ending at an output of the design, the capture time of the data cannot be determined from the design netlist. This is because the capture time and external logic delays are unknown. Therefore, to analyze the register-to-output timing, you must specify the external timing conditions beyond the output as an output delay. This concept is illustrated in [Figure 11](#).

Figure 11 External Output Delay



The data leaving output port “Z” passes through some combinational logic and is captured by an external sequential device clocked by CLK. To analyze the timing of Path 3, you need to specify the amount of delay from the arrival of data at output “Z” to the capture clock edge. In this example, the output delay is the delay of the external buffer plus the setup time requirement for the external flip-flop. You should also specify the load characteristics of the external circuit to accurately account for the interconnect delay at the output “Z.”

The commands for specifying input delays and output delays are `set_input_delay` and `set_output_delay`.

Delay Calculation

To find the slack for a timing path, the analysis tool must determine the arrival time of the launch clock edge, the arrival time of the capture clock edge, and the delay from the startpoint to the endpoint of the path. To find the arrival time of the launch clock edge, the tool calculates the delay from the original source clock to the clock input of the launch flip-flop. Similarly, to find the arrival time of the capture clock edge, the tool calculates the delay from the original source clock to the clock input of the capture flip-flop. It also considers the period of the clock, or if the launch and capture clocks are different, the worst-case offset between those clocks.

To find the cumulative delay along a path, the analysis tool adds the individual cell delays and interconnect delays along the path. The cell delay information is contained

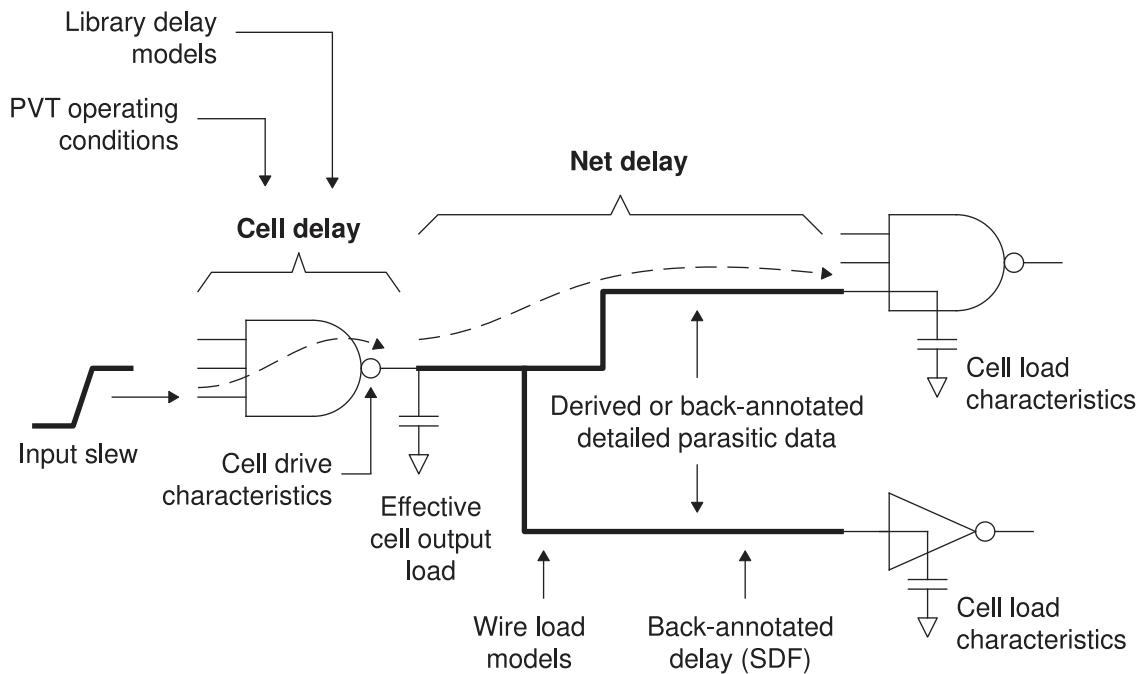
in the library description of each cell. To calculate the interconnect delays, the analysis tool needs to know the characteristics of the driver cell that is driving the net, the load characteristics of the receiver cells driven by the net, and the resistance and capacitance (RC) characteristics of the wires in the net. The interconnect RC characteristics depend on the physical configuration and lengths of the wires, so these characteristics can be determined accurately only after layout has been completed.

For timing analysis performed before placement and routing, the analysis tool must estimate the wire delays. The simplest estimation method is the wire load model, which gets a rough value for the total wire capacitance based on the size of the chip and the fanout of the net. Larger chip sizes and larger fanouts are assumed to result in longer wires and more resistance and capacitance. More advanced wire estimation methods predict cell placement and wire lengths, thereby getting more accurate results without using wire load models. Design Compiler with topographical technology is one such advanced tool.

For timing analysis performed after placement and routing, the analysis tool can extract the RC network accurately from the physical lengths of the wire connections and the known characteristics of the wire material. IC Compiler, for example, extracts RC information accurately from detail routing when available or from global routing where detail routing has not been completed.

The main types of information used in delay calculation are summarized in [Figure 12](#).

Figure 12 Information Used in Cell and Net Delay Calculation



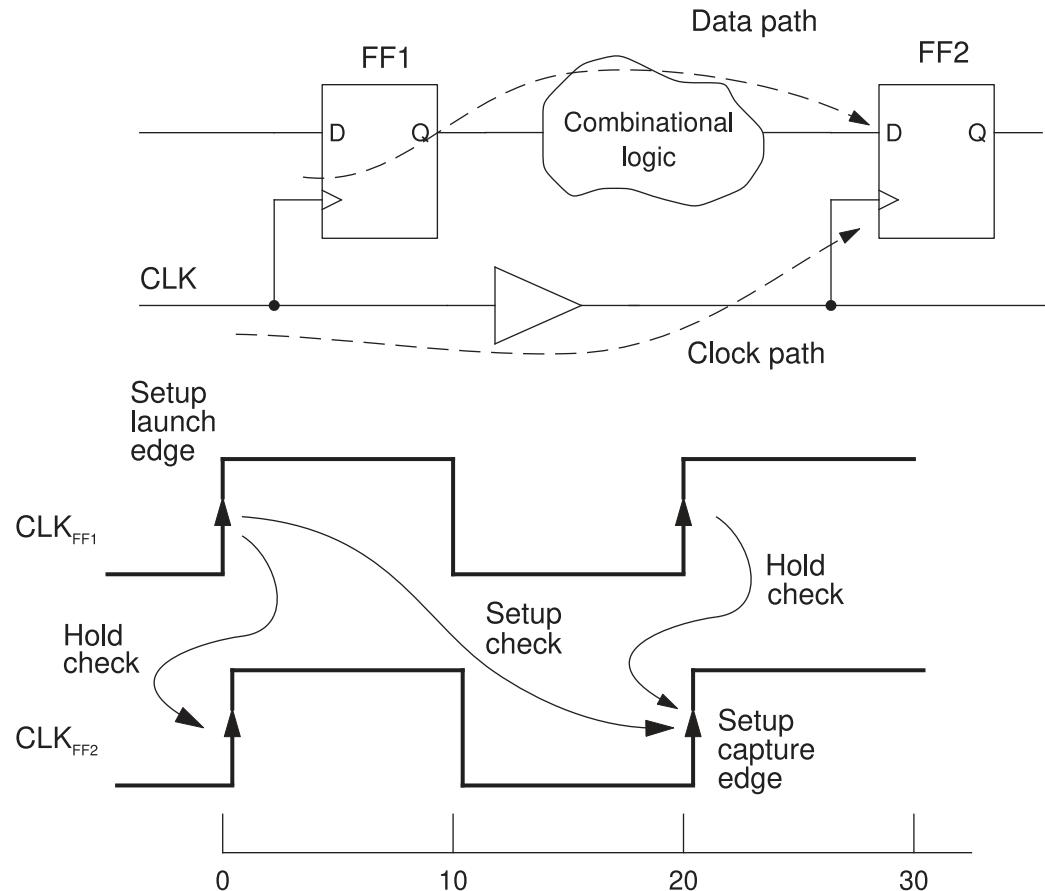
You can view a detailed accounting of the delay calculation for a cell or a net by using the `report_delay_calculation` command.

Flip-Flop and Latch Timing Checks

The method of checking setup and hold timing differs somewhat between flip-flop and latch-based designs. For edge-sensitive flip-flops, the data must be available strictly at the time that the clock edge arrives at the flip-flop. For level-sensitive latches, “borrowing” time from one latch to the next can loosen the timing requirements for certain paths.

[Figure 13](#) shows how a timing analysis tool checks setup and hold constraints for a synchronous design based on flip-flops.

Figure 13 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 by the `create_clock` command 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 the signal is to be propagated through the path in one clock cycle. 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 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.

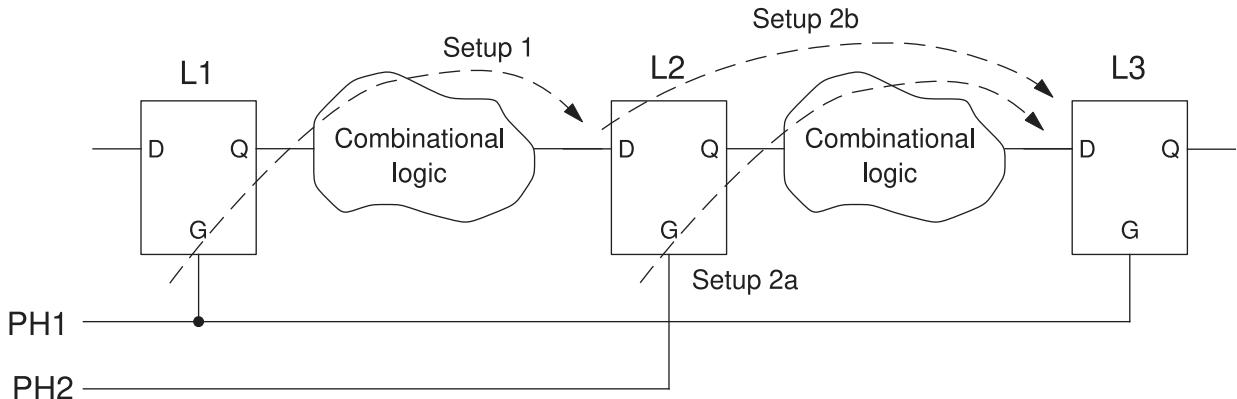
When the tool performs a hold check, it verifies that the data launched from FF1 reaches FF2 no sooner than the capture clock edge for the previous clock cycle. This check ensures that the data that already exists at the input of FF2 remains stable long enough

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.

Latch-based designs typically use two-phase, nonoverlapping clocks to control successive registers in a data path. In these cases, the tool can use time borrowing to lessen the constraints on successive paths.

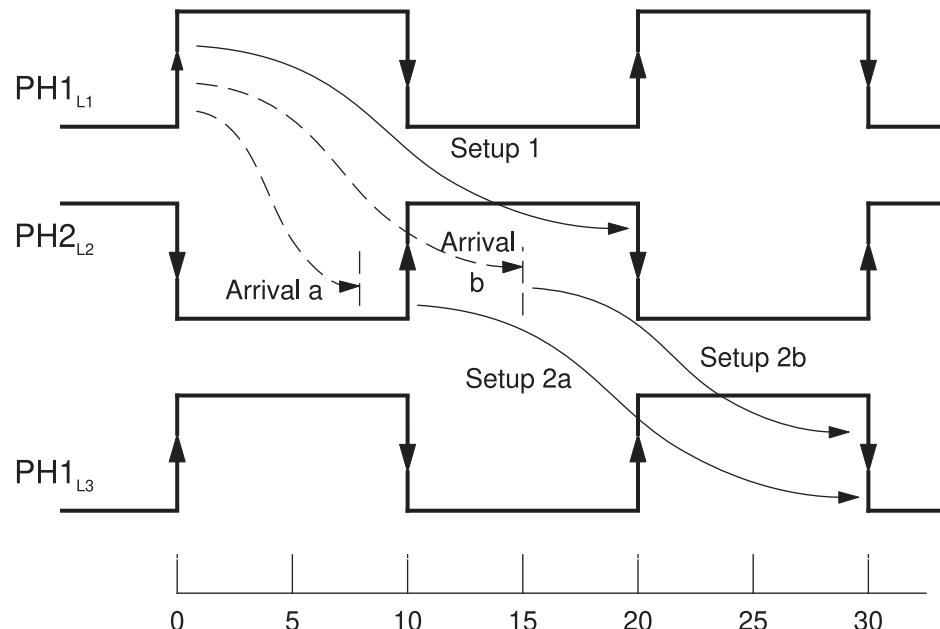
For example, consider the two-phase, latch-based path shown in [Figure 14](#). All three latches are level-sensitive, with the gate active when the G input is high. L1 and L3 are controlled by PH1, and L2 is controlled by PH2. A rising edge launches data from the latch output, and a falling edge captures data at the latch input. For this example, consider the latch setup and delay times to be zero.

Figure 14 Latch-Based Paths



[Figure 15](#) shows how the tool performs setup checks between these latches. For the path from L1 to L2, the rising edge of PH1 launches the data. The data must arrive at L2 before the closing edge of PH2 at time = 20. This timing requirement is labeled Setup 1. Depending on the amount of delay between L1 and L2, the data might arrive either before or after the opening edge of PH2 at time = 10, as indicated by the dashed-line arrows labeled “Arrival a” and “Arrival b” in the timing diagram. Arrival after time = 20 would be a timing violation.

Figure 15 Time Borrowing in Latch-Based Paths



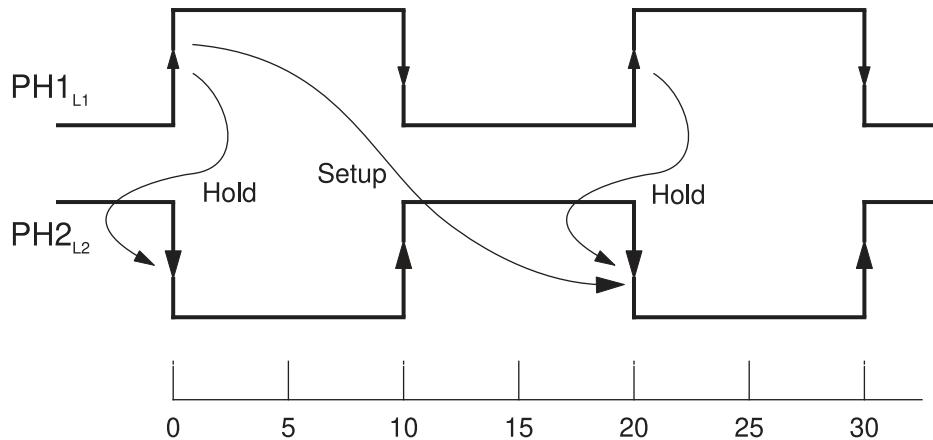
If the data arrives at L2 before the opening edge of PH2 at time = 10 (Arrival a), the data for the next path from L2 to L3 gets launched by the opening edge of PH2 at time = 10, just as a synchronous flip-flop would operate. In this case, no time is borrowed from the second path. This timing requirement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

If the data arrives after the opening edge of PH2 (Arrival b), the first path (from L1 to L2) borrows time from the second path (from L2 to L3). In that case, the launch of data for the second path occurs at the data arrival time at L2, at some time between the opening and closing edges of PH2. This timing requirement is labeled Setup 2b. When borrowing occurs, the path originates at the D pin, rather than the G pin, of L2.

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

To perform hold checking, the tool considers the launch and capture edges relative to the setup check. It verifies that data launched at the startpoint does not reach the endpoint too quickly, thereby ensuring that data launched in the previous cycle is latched and not overwritten by the new data. This is depicted in [Figure 16](#).

Figure 16 Hold Checks in Latch-Based Paths



Timing Analysis in the Design Flow

Timing analysis serves different purposes in different phases of the design flow. In Design Compiler, timing drives the selection of library cells used for synthesis and the allocation of registers between combinational logic in data paths. In IC Compiler, timing drives the placement of cells and the routing of interconnections to minimize delays in the critical paths. In PrimeTime, exhaustive signoff timing analysis is the main purpose of the tool.

These tools all share the same basic delay calculation methods. The timing results are generally consistent between the tools but not always identical. Because PrimeTime is a signoff analysis tool, it performs a more comprehensive and exhaustive analysis to verify correct timing, whereas Design Compiler and IC Compiler perform timing analysis with sufficient accuracy to drive synthesis, physical implementation, and optimization.

Synopsys Design Constraint Commands

Design Compiler, IC Compiler, and PrimeTime share many common timing analysis features. The tools allow you to use the same commands to specify timing constraints and generate timing reports. These commands are known as the Synopsys Design Constraints (SDC). These commands have the same syntax and produce the same effects across all the supported tools. That means you can use the same SDC script to constrain a design in Design Compiler, IC Compiler, PrimeTime, and other tools. The SDC commands can specify design rule constraints and power constraints as well as timing constraints.

In each tool, the `write_sdc` command writes out a script containing a set of SDC commands that specify the current constraints set on the design. In a different tool, you

can use the `read_sdc` command to read in the file and apply the same constraints. The `read_sdc` command works very much like the `source` command, but the `read_sdc` command also checks the script commands for SDC compliance. SDC script files can be used to transfer constraints between Synopsys tools and also certain external tools.

Some Synopsys tools have extended constraint capabilities beyond what is supported by SDC in the form of additional commands or command options. The `write_sdc` command writes out only the SDC-compatible commands, which can be executed in any SDC-compatible tool. The `write_script` command writes out a wider range of commands that set the design attributes.

To help keep track of constraints that have been set, some SDC commands optionally allow a comment string to be associated with the constraint. To add a comment, use the `-comment` option along with a comment string. For example,

```
prompt> create_clock -name "CLK" -period 0.33 -comment "Main clock"
```

The comment is maintained throughout the design, analysis, and implementation flow. Each time you use the `write_sdc` or `write_script` command, the tool writes out the SDC command with the `-comment` option using the same comment string.

To remove all Synopsys Design Constraints from the design, use the `remove_sdc` command.

Table 1 lists the SDC commands used for specifying timing constraints and timing-related design characteristics. For more information about SDC commands, see the *Using the Synopsys Design Constraints Format Application Note*, which is available on SolvNet in the documentation sets for Design Compiler, IC Compiler, and PrimeTime. To access SolvNet, go to the SolvNet Web page at the following address:

<https://solvnet.synopsys.com>

Table 1 SDC Timing Commands

Command	Usage
<code>create_clock</code>	Specifies the clocks used in the design and their characteristics: name, period, waveform, and design location. The timing analyzer needs this information to determine the required data arrival time at each path endpoint.
<code>group_path</code>	Groups a set of paths or endpoints for timing analysis and cost function calculations. Paths within a group are analyzed and optimized separately from other groups. By default, there is one path group per clock.
<code>set_clock_gating_check</code>	Creates one or more clock-gating checks in the design. A clock-gating check is a setup or hold check performed on the control input of a clock-gating cell. This detects occurrences of clipped clocked edges or spurious clock pulses.

Table 1 SDC Timing Commands (Continued)

Command	Usage
<code>set_clock_groups</code>	Specifies groups of clocks that are mutually exclusive or asynchronous with each other. This prevents analysis of a timing path that starts in one clock group and ends in another.
<code>set_clock_latency</code>	Specifies explicitly the source latency or network latency of a clock. This command is typically used before layout, when propagated clocking cannot be used. Source latency is the time the clock signal takes to be propagated from its ideal waveform origin point to the clock definition point in the design. Network latency is the time the clock signal takes to be propagated from the clock definition point in the design to the clock pin of the sequential device. The timing analyzer uses this information to determine clock arrival times in the absence of propagated clocking.
<code>set_sense</code>	Specifies the unateness of a clock signal, either positive or negative, that is propagated past a nonunate point in the clock network. A nonunate point is a place where the sense of the clock signal cannot be determined, such as the output of an exclusive OR gate with the clock signal as one input and an unknown side-input value as the other input.
<code>set_clock_transition</code>	Specifies explicitly the rising or falling transition times of a clock. This command is typically used before layout, when propagated clocking cannot be used. The transition time applies to rising or falling transitions at the clock pins of sequential devices clocked by the specified clock. The timing analyzer uses this information to determine clock transition times in the absence of propagated clocking.
<code>set_clock_uncertainty</code>	Specifies the uncertainty or skew characteristics of a single clock or between two different clocks. For a single clock, simple uncertainty is the maximum difference between successive edges with respect to variation away from the nominal arrival times. For two clocks, interclock uncertainty is the maximum difference or skew between occurrences of clock edges with respect to variation away from the nominal arrival times. The timing analyzer uses this information to determine the worst possible clock arrival times for each timing check.
<code>set_data_check</code>	Creates a custom data-to-data check, also known as a nonsequential constraint, using specified setup and hold time values between the specified data signals. You specify the “from” object or related pin, the “to” object or constrained pin, and the setup or hold value for the check.
<code>set_disable_timing</code>	Disables timing checks and timing optimization for specified cells, pins, or ports. This command removes the affected objects entirely from timing analysis, unlike the <code>set_false_path</code> command, which removes only the timing constraints, and not delay calculation, from the paths. If all paths through a pin are false, <code>set_disable_timing</code> is more efficient than <code>set_false_path</code> .

Table 1 SDC Timing Commands (Continued)

Command	Usage
set_driving_cell	Specifies the name of a library cell that drives one or more input ports of the design. This information allows the timing analyzer to accurately determine the delay cause by the driver, load, and wire characteristics of the net.
set_fanout_load	Specifies the number of external loads in the fanout of one or more output ports of the design. The tool uses this information to enforce the maximum fanout design rule, not for performing timing analysis.
set_ideal_latency	Specifies explicitly the ideal clock latency in the transitive fanout of specified ports or pins. Ideal clock latency is the time it takes for a clock signal to propagate from its ideal waveform origin point to the clock pin of the sequential device in an ideal network. The default ideal latency is zero. The timing analysis tool uses the ideal latency to determine clock arrival times in the absence of propagated clocking.
set_ideal_network	Invokes ideal clocking behavior in the transitive fanout of specified ports, pins, or nets, causing explicitly specified latency and transition times (zero by default) to be used throughout the specified network. Ideal clocking is used before layout, when propagated clocking cannot be used.
set_ideal_transition	Specifies explicitly the rising or falling transition times of signals in the transitive fanout of specified ports or pins. The default ideal transition time is zero. The timing analysis tool uses the ideal transition time in the absence of propagated clocking.
set_input_delay	Specifies the amount of delay from a launch clock edge outside of the design to the arrival of data at an input of the design. This information is necessary to check the timing of signals entering the design inputs.
set_input_transition	Specifies explicitly the rise or fall transition times on input ports of the design for propagated (not ideal) clocking. The timing analyzer uses this information to determine the delays and transition times of signals in the transitive fanout of the input.
set_load	Specifies explicitly the capacitive load on one or more input ports, output ports, or nets. The timing analyzer uses this information to determine the effects of the load on delays and transition times of signals passing through the port or net.
set_operating_conditions	Specifies the operating conditions under which the design is analyzed and optimized. The library defines the operating conditions, each condition consisting of a set of process, temperature, and voltage values. Each cell in the library has a different set of cell timing characteristics for each operating condition. The command selects one or more of these defined operating conditions by name and invokes either best-case/worst-case analysis or on-chip variation analysis.

Table 1 SDC Timing Commands (Continued)

Command	Usage
<code>set_output_delay</code>	Specifies the amount of delay from the departure of data at an output of the design to the capture clock edge outside the design. This information is necessary to check the timing of signals leaving the design outputs.
<code>set_port_fanout_number</code>	Specifies the number of external loads in the fanout of one or more output ports of the design. This allows the timing analyzer to estimate the total wire load of the external devices connected to the output.
<code>set_propagated_clock</code>	Causes network latency to be determined by propagating delays through the clock network for specified clocks or for the transitive fanout of specified ports or pins. Propagated clocking can be used after layout, when detailed net RC information is available.
<code>set_resistance</code>	Specifies explicitly the resistance of one or more nets. The timing analyzer uses this information to determine the effects of the resistance on delays and transition times of signals passing through the net.
<code>set_timing_derate</code>	Applies derating or adjustment factors to specified delays of timing checks. Derating can be used to model the worst-case effects of process, voltage, and temperature variation.
<code>set_wire_load_model</code>	Specifies the types of wire load models used for estimating wire resistance and capacitance before routing has been performed. Used in Design Compiler and PrimeTime; not used in Design Compiler with topographical technology or IC Compiler.

Library Timing Data

To perform timing analysis, the tool needs information about the timing characteristics of the logic cells used in the design, such as the input-to-output delays of combinational logic blocks; clock-to-output delays, setup times, and hold times of sequential blocks; and transition times of cell output signals. The cell timing characteristics are contained in the library description of each cell. Timing analysis tools can read the library data in the following forms:

- Liberty (.lib) format using the `read_lib` command
- Synopsys database (.db) format using the `read_db` or `read_file` command

The cell timing characteristics are typically determined by a characterization tool such as Liberty NCX in combination with a circuit simulator such as HSPICE. Liberty NCX writes out the cell timing data in Liberty (.lib) format, a text format that a person can read. Some timing analysis tools can read .lib files directly with the `read_lib` command.

Libraries in Liberty (.lib) format are typically compiled into Synopsys database (.db) format by Library Compiler. The .db file is a binary format that is more compact and faster to read than the .lib format. A timing analysis tool can read in a .db library directly with the `read_db` command or implicitly via the `link_library` variable setting.

Each cell timing parameter, such as input-to-output delay or output transition time, is a function of the input slew and output load. Accordingly, the delay information is organized in a table of values corresponding to different combinations of input slew and output load. You can view the timing parameter tables in the .lib library file, if available. You can also view the timing data in the analysis tool by using the `report_lib -timing` command. For example,

```
prompt> report_lib -timing cb13fs120_max nd02d1
```

```
...
```

Lookup Table Template:

```
Template_name
-----
del_1_5_7_w
    VARIABLE_1: input_net_transition
    VARIABLE_2: total_output_net_capacitance
    INDEX_1:  0.0150  0.2500  0.6500  1.4000  3.0000
    INDEX_2:  0.0000  0.0070  0.0140  0.0385  0.0805  0.1505  0.3500
...
DELAY: A2, ZN, prop, neg_unate, '', ( , ), ( , ), ( , );
cell_rise ( del_1_5_7_w ) :
    VALUES :  0.0270  0.0480  0.0690  0.1400  0.2620  0.4650
              1.0420  0.0680  0.0990  0.1230  0.1950  0.3160
              0.5190  1.0970  0.1010  0.1510  0.1890  0.2870
              0.4110  0.6140  1.1910  0.1380  0.2080  0.2630
              0.4060  0.5790  0.7920  1.3670  0.1880  0.2860
              0.3620  0.5660  0.8170  1.1270  1.7480
```

The table lists the delay values from the input A2 to the output ZN for a rising signal, for each possible combination of an input transition time and an output capacitive load taken from the index tables. The delay calculator uses interpolation or extrapolation to get delay values for input transition times and capacitive loads between or outside of the index tables. Similar tables specify the output transition times as a function of input transition and output load. The output transition time calculated for a cell becomes the input transition time for the next cell in the timing path.

For a detailed description of the delay calculation for a particular cell instance or net in the design, you can use the `report_delay_calculation` command. For example,

```
prompt> report_delay_calculation -from I_RISC_CORE/I_ALU/U27/A2 \
           -to I_RISC_CORE/I_ALU/U27/ZN
...
Rise Delay

cell delay = 0.0583731
Table is indexed by
(X) input_pin_transition = 0.103374
(Y) output_net_total_cap = 0.00451049
Relevant portion of lookup table:
(X) 0.0150      (X) 0.2500
(Y) 0.0000      (Z) 0.0270      (Z) 0.0680
(Y) 0.0070      (Z) 0.0480      (Z) 0.0990

Z = A + B*X + C*Y + D*X*Y
A = 0.0244          B = 0.1745
C = 2.9088          D = 6.0790

Z = 0.0583731
scaling result for operating conditions
multiplying by 1 gives 0.0583731
...
```

Cell delays vary with operating conditions, so the libraries may have different delay values for different operating condition corners. A corner is a particular combination of voltage, temperature, and process values. Some libraries are designed for scaling so that the analysis tool can obtain accurate delay values at intermediate operating conditions by using interpolation between the defined operating conditions and their corresponding delay values.

Design Compiler

Design Compiler is a synthesis tool that converts a design description at the Register Transfer Level (RTL) to a gate-level netlist. The synthesis process performed by Design Compiler typically consists of the following major steps:

- Read in the RTL description in Verilog or VHDL format.
- Read in the timing, area, and power constraints in Synopsys Design Constraints (SDC) format.
- Generate the design logic using generic Boolean gates, optimize the logic, and map the design into a netlist using the technology-specific gates of the target library.
- Write out the compiled gate-level netlist in .ddc format.

Although the area and power constraints are important, only the timing constraints must be absolutely satisfied for the circuit to operate. Timing drives the logic implementation and selection of gates from the library to ensure that timing path delays do not exceed the applicable clock periods. For example, where a setup delay is found to be too long, Design Compiler might use larger devices with more drive strength to reduce net delays, at the expense of more area.

After you apply the timing constraints to the design, you should check for timing constraint and setup problems with the `check_timing` command. After synthesis is completed, you can report the worst-case paths and analyze them in detail with the `report_constraint` and `report_timing` commands.

Ideal Clocking

At the logic synthesis stage before layout, performing clock tree synthesis is not practical because the wire lengths are unknown and the clock skew is highly sensitive to parasitic interconnect differences. By default, Design Compiler uses ideal clocking, which means zero latency, zero uncertainty, and zero transition time for all clock signals. Latency is the delay from the clock source to the clock register pins, which is zero for ideal clocking, based on the assumption of zero resistance and capacitance of the clock network. All nets and cells in the clock network are automatically marked as `dont_touch`. Ideal networks are not optimized or buffered during synthesis.

Ideal clocking is highly optimistic. To get more accurate timing results, you can specify nonzero latency, uncertainty, and transition times for clock signals to represent the approximate timing values expected for the completed clock network. For example,

```
dc_shell> set_clock_latency 1.2 -rise [get_clocks CLK1]
dc_shell> set_clock_latency 0.9 -fall [get_clocks CLK1]
dc_shell> set_clock_uncertainty -setup 0.65 [get_clocks CLK1]
dc_shell> set_clock_uncertainty -hold 0.45 [get_clocks CLK1]
dc_shell> set_clock_transition 0.34 -rise [get_clocks CLK1]
dc_shell> set_clock_transition 0.30 -fall [get_clocks CLK1]
```

With settings such as these, the clock network is still treated as ideal, but with the specified latency, uncertainty, and skew values instead of zero throughout the clock network.

High-fanout nets other than clocks, such as reset or enable signals, might also require buffer trees that are synthesized at the layout stage rather than in Design Compiler. For these nets, you can specify ideal network behavior and nonzero latency and transition times for synthesis, in anticipation of high-fanout network synthesis in the layout tool. For example,

```
dc_shell> set_ideal_network [get_ports Reset]
dc_shell> set_ideal_latency 1.40 [get_ports Reset]
dc_shell> set_ideal_transition 0.30 [get_ports Reset]
```

Wire Load Models and Topographical Technology

To perform accurate timing analysis, both cell delays and net delays must be known. However, at the prelayout logic synthesis stage, the exact net delays cannot be determined because the wire lengths are unknown. There are two methods you can use to estimate the RC wire characteristics in Design Compiler: wire load models and topographical technology.

A wire load model obtains one parasitic resistance value and one capacitance value for each net, based on the net's fanout. A net having a larger fanout is assumed to have more wires and therefore more resistance and capacitance. Wire load models are supplied in the logic library. The models are based on data extracted from similar designs for similar size, fabricated with the same process technology.

A library typically has several wire load models to be used for different design sizes. As the size of a design increases, standard cells can be placed physically farther apart within that design, which means that wire lengths are typically longer. Some library vendors may use names for their models to represent different design sizes, such as "300kGates," "600kGates," and so on. It is important to select the appropriate wire load model for the design according to size. For example,

```
dc_shell> set_wire_load_model -name 1.6MGates
```

The specified model applies to all nets at the current design level and below. When the design is hierarchical and the blocks are grouped into physical areas of the chip, then smaller individual wire load models can better represent the actual RC values within each lower-level subdesign. For example,

```
dc_shell> set_wire_load_model -name 1.6MGates
dc_shell> set_wire_load_mode enclosed
dc_shell> set_wire_load_model -name 800KGates [get_designs SUB1]
dc_shell> set_wire_load_model -name 200KGates [get_designs B1]
dc_shell> set_wire_load_model -name 100KGates [get_designs B2]
```

Setting the wire load mode to "enclosed" means that the wire capacitance of each net is calculated using the wire load model set on the smallest subdesign that completely encloses that net.

Some libraries support automatic wire load selection based on the area of the design. Usage of this feature can be controlled with the `auto_wire_load_selection` variable.

Wire load models are based on statistical averages and are not specific to the particular design. In ultra-deep submicron designs, wire load models may not provide enough accuracy because of the increased impact of wire parasitics on path delays. For these designs, Design Compiler with topographical technology is recommended.

Design Compiler with topographical technology accurately predicts timing during synthesis without using wire load models. Instead, it uses physical information from the Milkyway database to accurately predict actual wire lengths and obtain more accurate predictions of

actual wire resistance and capacitance values. It uses the design floorplan if available, or it creates its own floorplan if needed, to get the placement information it needs for predicting wire lengths. The topographical mode requires DC Ultra and DesignWare licenses.

To run Design Compiler in topographical mode, use the `-topographical` option when you invoke `dc_shell`:

```
% dc_shell -topographical
...
Initializing...
Starting shell in Topographical mode...
...
dc_shell-topo>
```

The shell prompt is `dc_shell-topo>` in topographical mode. No wire load models are needed; if any are present, they are ignored. When you run the `compile_ultra` command, it automatically invokes the topographical features. The command performs placement in the background to estimate wire lengths accurately. Cell placement based is on an existing floorplan provided in DEF format or a floorplan specified manually with commands such as `set_placement_area`, `set_port_location`, `set_cell_location`, and `create_placement_bounds`.

The physical Milkyway reference libraries contain the physical layout descriptions of the cells in the synthesized netlist, including standard cells, macro cells, and pad cells, which Design Compiler uses for topographical placement during compile operations. The technology file defines the process metal layers, physical design rules, units of resistance, capacitance, and so on. The TLUPlus files define models for calculating ultra-deep-submicron RC parasitic values from extracted wire data.

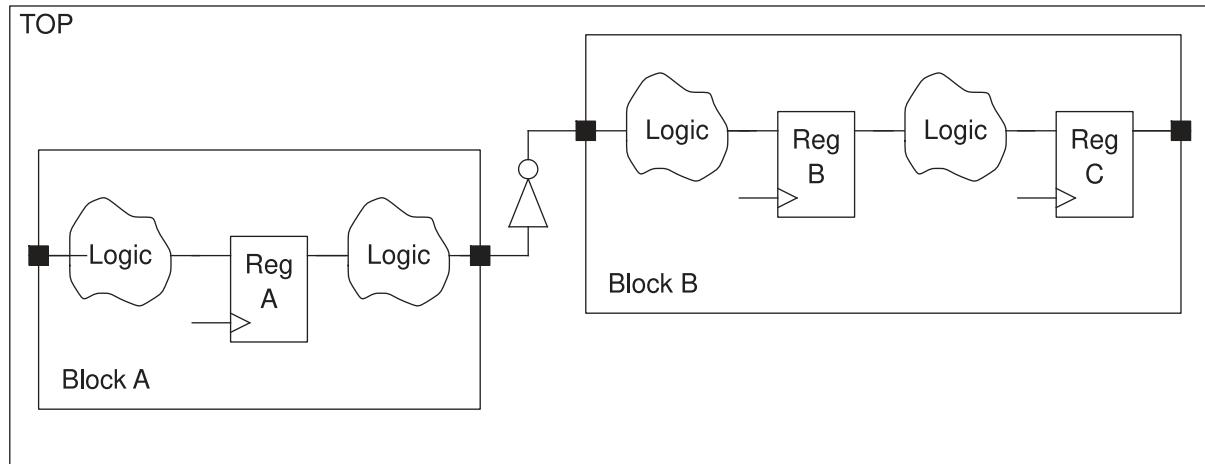
By using this physical information, Design Compiler obtains a placement model that accurately predicts wire lengths and therefore wire RC and delay characteristics. The result is a higher-quality synthesis and fewer design iterations between logic synthesis and physical implementation.

Design Partitioning

A large design is typically divided into a hierarchy of blocks, often organized by function, to break down the design and implementation task into manageable units. To get the best possible results, it is important to partition the design in a manner that allows Design Compiler to optimize the timing and area within the boundaries of each block.

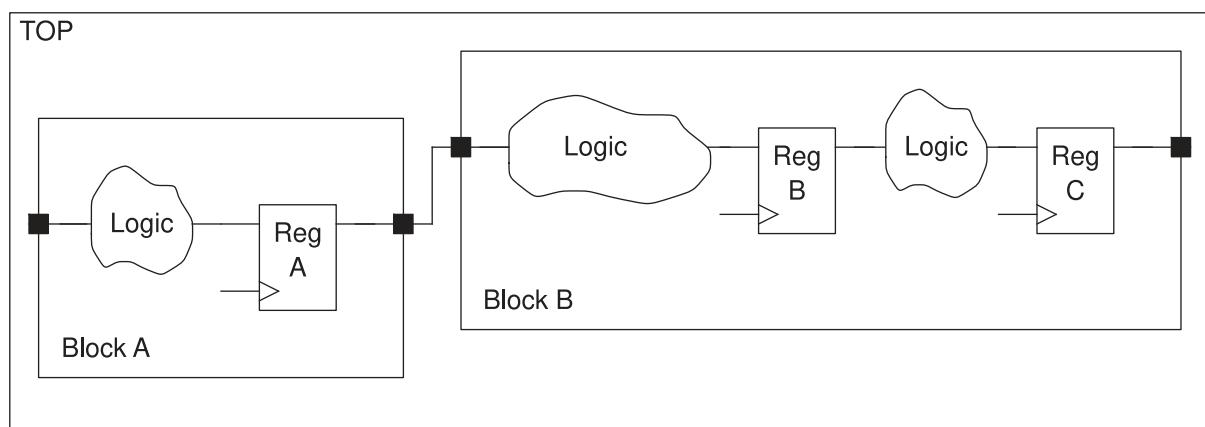
For example, consider the partitioning of logic shown in [Figure 17](#). The combinational logic between Register A and Register B has been divided between Block A and Block B. Some “glue logic,” an inverter, holds the logic together at the top level. Design Compiler must preserve the Block A and Block B pin definitions, so it cannot perform logic optimization across the hierarchical boundary between them. Furthermore, there is no opportunity to optimize away the inverter at the top level.

Figure 17 Poorly Partitioned Blocks



The logic partitioning shown in [Figure 18](#) is much better. The combinational logic between Register A and Register B has been moved entirely into Block B, including the top-level “glue logic.” This partitioning allows Design Compiler to optimize all of the logic together inside block B, possibly resulting in less time delay, smaller area, or both for that portion of the combinational logic. The partition was made at the output of Register A, not at the input of Register B, to give Design Compiler an opportunity to further optimize the logic by using a more complex flip-flop cell. For example, a multiplexing function in the logic could be implemented by using an enable-type flip-flop.

Figure 18 Well-Partitioned Blocks



For better results, avoid dividing combinational logic between different blocks. Instead, keep as much combinational logic together and keep it with the downstream registers, such as Register B in the foregoing example. Where necessary, make a partition at the output of a register, such as the output of Register A in the foregoing example. This simplifies the setting of input and output timing constraints for lower-level block synthesis and provides Design Compiler with nearly a whole clock cycle in which to optimize the register-to-register logic.

Performing good partitioning in the original RTL is the best strategy. However, if the RTL partitioning is not favorable, you can improve the partitioning in Design Compiler by ungrouping and regrouping the logic into a more favorable configuration prior to the compile operation. The `compile_ultra` command, by default, performs automatic repartitioning. You can also manually repartition the design with the `ungroup` and `group` commands.

Path Groups

The timing paths of the design are organized into groups called *path groups* (not to be confused with *logic grouping* described in the foregoing section). By default, there is one path group for each clock in the design. All timing paths clocked by a given clock at the path endpoint belong to that clock's path group. A design that has only a single clock has only one clocked path group, so all clocked paths in the design belong to that group.

All timing paths within a path group are optimized for timing together, starting with the critical path, which is the path having the worst slack within the group. After the critical path is fixed, the next-worst path becomes the new critical path and the target for fixing. The tool continues fixing paths until all paths in the group have zero slack or until a better optimization solution for the current critical path cannot be found. In the latter case, the subcritical paths are not fixed, but are left with timing violations. The subcritical paths are the paths with better slack than the critical path, but that are still in violation.

Paths within a group are optimized and reported separately from other groups. For example, in a design with two clocks, CLK1 and CLK2, there are two path groups. Design Compiler optimizes each path group in turn, starting with the critical path in each group, even if the worst slacks are different between the two groups.

You can optionally divide the paths into groups to control the focus of optimization effort on targeted paths. For example, if you are not sure about the input delay requirements, you can put the input-to-register paths into a separate group. In that case, the input-to-register paths are optimized separately from the other paths and the worst input-to-register violation does not prevent optimization of register-to-register paths clocked by the same clock.

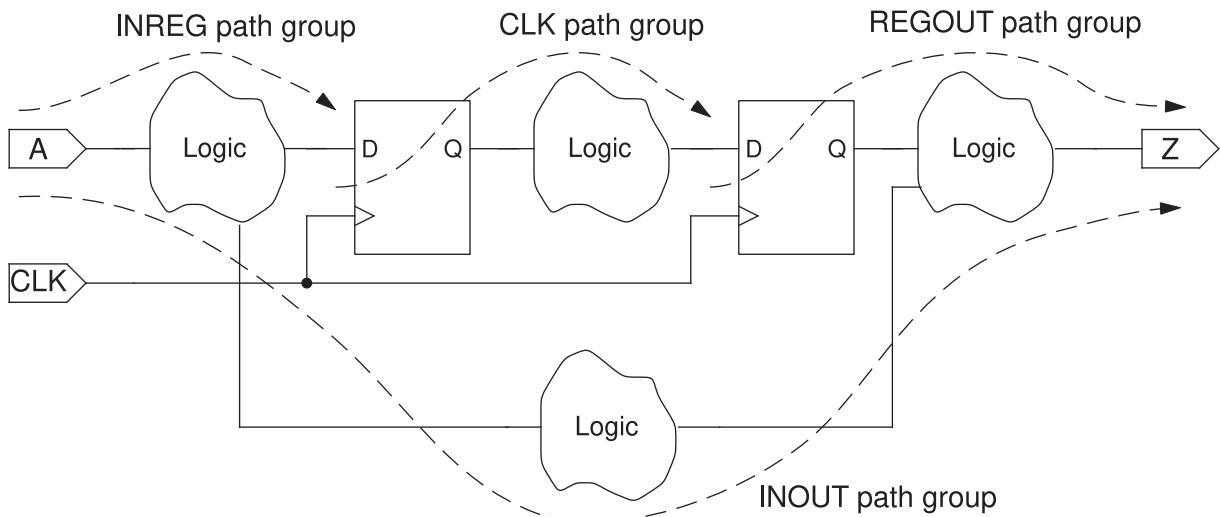
The command for grouping paths is `group_path`. For example,

```
dc_shell> create_clock -name CLK -period 1.67 [get_ports CLK]
dc_shell> group_path -name INREG -from [all_inputs]
```

```
dc_shell> group_path -name REGOUT -to [all_outputs]
dc_shell> group_path -name INOUT -from [all_inputs] -to [all_outputs]
```

By default, all paths clocked by CLK at the path endpoint belong to the CLK path group. In this example, the three `group_path` commands place the input-to-register, register-to-output, and input-to-output paths into separate path groups called INREG, REGOUT, and INOUT, respectively. The register-to-register paths remain in the default CLK path group as demonstrated in [Figure 19](#).

Figure 19 Timing Path Groups



With this path grouping, any problems encountered in optimizing input-related or output-related timing paths will not affect the optimization of register-to-register paths. Furthermore, the `report_timing` command reports the worst path in each path group separately, so you can find out about the worst register-to-register paths separately from the input-related and output-related paths.

You can optionally assign a weight, also known as cost function, to each path group so that Design Compiler applies more effort to optimizing the targeted group. The default weighting value for each path group is 1. The higher the weighting, the higher the effort applied to the group. For example, the following commands assign the input-related and output-related paths as in the previous example, but also specify an effort level of 5 for the register-to-register paths and 2 to the input-to-register paths. The effort level remains at 1 for the other types of paths.

```
dc_shell> create_clock -name CLK -period 1.67 [get_ports CLK]
dc_shell> group_path -name INREG -from [all_inputs] -weight 2
dc_shell> group_path -name REGOUT -to [all_outputs]
```

```
dc_shell> group_path -name INOUT -from [all_inputs] -to [all_outputs]
dc_shell> group_path -name CLK -weight 5
```

Higher weighting for a group means that Design Compiler will attempt to reduce the size of a timing violation for a path in that group at the expense of slack in a path belonging to a lower-weighted group.

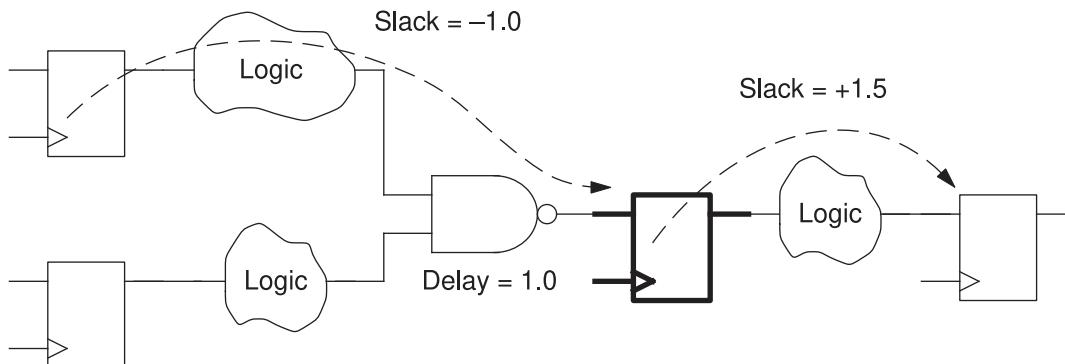
By default, different `group_path` commands follow conventional rules of priority to resolve conflicting path selections, as described in [Path Specification Priority](#). You can optionally set a priority level for a command by using the `-priority` option of the `group_path` command. This assigns a priority to a particular `group_path` command, overriding the default order of priority.

To get information about the current set of path groups, use the `report_path_group` command. To remove a path group, use the `remove_path_group` command. Paths belonging to a removed group are implicitly assigned to the default path group.

Register Retiming Optimization

Another way to meet timing constraints is to reposition registers within the combinational logic. For example, in [Figure 20](#), the timing path on the left has a setup timing violation due to a long combinational logic path, with a total slack of -1.0 for the path. Meanwhile, the short timing path downstream has a positive slack of $+1.5$.

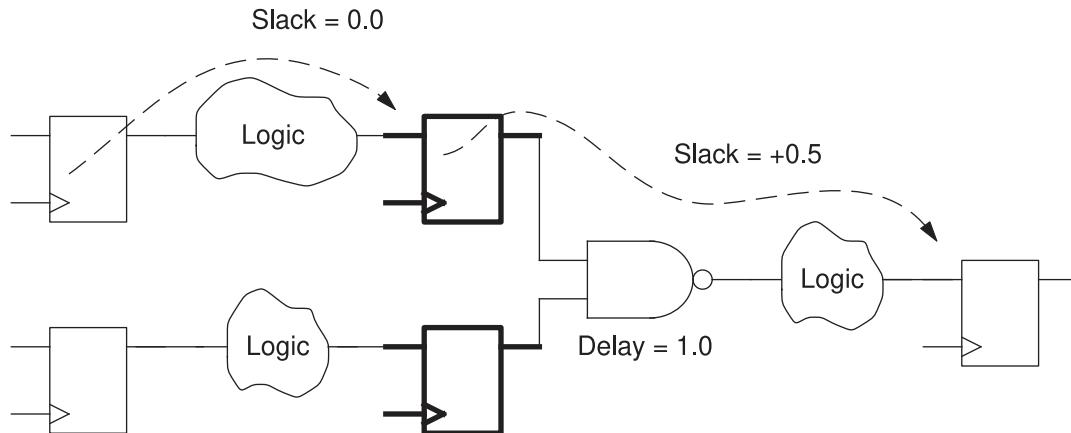
Figure 20 Setup Timing Violation



Design Compiler can fix this violation by changing the implementation as shown in [Figure 21](#). It takes away the register at the output of the NAND gate and replaces it with two new registers at the inputs of the same NAND gate. This produces a logically equivalent circuit that passes setup timing for both paths. The change effectively takes away the delay of the NAND gate from the violating path and gives that delay to the

downstream path. The cost of this timing violation correction is the area of the additional register in the circuit.

Figure 21 Violation Fixed by Register Retiming



Conversely, a circuit like the one shown in [Figure 21](#), but with some available timing slack in the first timing path, can reduce the total area by taking away the two registers at the inputs of the NAND gate and replacing them with a single register at the output of the NAND gate.

Optimization performed by splitting or merging registers and moving those registers through combinational logic, whether for better timing or better area, is called *register repositioning* or *register retiming*. This type of optimization is performed with the command `compile_ultra -retime, balance_registers, or optimize_registers`.

IC Compiler

IC Compiler is a comprehensive chip-level physical implementation tool that supports design planning, placement, clock tree synthesis, routing, test, and design-for-manufacturing (DFM) capabilities. Starting from a gate-level netlist, IC Compiler generates a placed and routed design with clock trees, fully optimized and prepared for manufacturing. An IC Compiler session typically consists of the following major steps:

- Setup
- Design planning
- Placement (`place_opt` command)
- Clock tree synthesis (`clock_opt` command)

- Routing (`route_opt` command)
- Analysis
- Chip finishing

Timing analysis plays an important role in design planning, placement, clock tree synthesis, routing, and postroute analysis. The same timing constraints applied to logic synthesis in Design Compiler should be applied to physical implementation in IC Compiler, including operating conditions, clocks, input drivers, output loads, input/output delays, and timing exceptions. You can also specify physical constraints such as area, utilization, congestion, and layer usage for routing. After you apply the timing constraints, you should check for timing constraint and setup problems with the `check_timing` command.

Design Planning

In the design planning context, floorplanning is the process of sizing and placing hierarchical cells and functional blocks in a manner that makes later physical design steps more effective. Floorplanning in hierarchical flows provides a basis for estimating the timing of the top level. A timing budget allocates the clock cycle time to each block according to the top-level timing estimation.

An effective floorplan helps ensure timing closure in many ways, such as placing blocks to make critical paths short, preventing routing congestion that would lead to longer paths, and eliminating the need for over-the-top routing for noise-sensitive blocks. The challenge is to create a floorplan with good area efficiency while leaving sufficient area for routing.

You can check the timing of the entire design at various stages of the design planning flow by using the `check_timing` and `report_timing` commands. You can also use the `check_fp_timing_environment` command to check the timing budgets assigned to each block.

After the design passes the early timing feasibility checks, you can perform timing budgeting on plan groups by using the `allocate_fp_budgets` command. Timing budgeting generates the SDC constraints and attributes for the individual plan groups. These constraints and attributes are then transferred to the individual soft macro blocks when the hierarchy is committed using the `commit_fp_plan_groups` command.

Clock planning is an optional step in the design planning flow. You can use clock planning to estimate the clock tree insertion delay and skew in a hierarchical design. Clock planning also helps determine the optimal clock pin locations on blocks. Clock planning inserts anchor cells to isolate the clock trees inside the plan group from the top-level clock tree. It then runs fast clock tree synthesis for each plan group to estimate the clock skew and insertion delay. You set up and perform clock planning with the `set_fp_clock_plan_options` and `compile_fp_clock_plan` commands.

Placement

Placement is the process of finding a suitable physical location for each cell in the design. Placement is performed in two stages: coarse placement and legalization.

During coarse placement, IC Compiler determines an approximate location for each cell according to the timing and congestion constraints. The placed cells do not fall on the placement grid and may overlap each other. Large cells, such as RAM and IP blocks, act as placement blockages for smaller, leaf-level cells. Coarse placement is fast and is sufficiently accurate for initial timing and congestion analysis.

During legalization, IC Compiler moves the cells to precisely legal locations on the placement grid and eliminates any overlap between cells. The small changes to cell locations cause the lengths of the wire connections to change, possibly causing new timing violations. Such violations can often be fixed by incremental optimization, for example, by resizing the driving cells.

The `place_opt` command is recommended for performing placement in most situations. This command performs coarse placement, high-fanout net synthesis, physical optimization, and legalization, all in a single operation. In certain applications, you might want to perform placement tasks individually using commands such as `create_placement` and `psynopt`, for a greater degree of control or to closely monitor the results as they are generated.

In the placement process, IC Compiler considers possible trade-offs between timing and congestion. Timing considerations bring cells closer together to minimize wire lengths and therefore wire delays. On the other hand, the occurrence of congestion draws cells farther apart to provide room for the connections. Congestion cannot be ignored entirely in favor of timing because rerouting wires around congested areas will cause an increase in wire lengths and wire delays, thus defeating the value of close placement.

In the `place_opt` command, the `-congestion` option causes the tool to apply more effort to congestion removal, resulting in better routability. However, this option should be used only if congestion is expected to be a problem because it requires more runtime and causes area utilization to be less uniform across the available placement area. If congestion is found to be a problem after placement and optimization, it can be improved incrementally with the `refine_placement` command. Timing, area, and congestion optimization can also be done incrementally with the `psynopt` command.

The `-area_recovery` option of the `place_opt` command allows IC Compiler to recover chip area where there is extra timing slack available. For example, it can resize cells smaller in timing paths where there is a positive timing slack.

Placement is typically done before clock tree synthesis, so the clock network is ideal and does not have a clock buffer tree available for accurate clock network timing analysis. To get more accurate timing results, you should use the same commands as those used in

Design Compiler to specify nonzero latency, uncertainty, and transition times for the clock network. For example,

```
icc_shell> set_clock_latency 1.2 -rise [get_clocks CLK1]
icc_shell> set_clock_latency 0.9 -fall [get_clocks CLK1]
icc_shell> set_clock_uncertainty -setup 0.65 [get_clocks CLK1]
icc_shell> set_clock_uncertainty -hold 0.45 [get_clocks CLK1]
icc_shell> set_clock_transition 0.34 -rise [get_clocks CLK1]
icc_shell> set_clock_transition 0.30 -fall [get_clocks CLK1]
```

As in Design Compiler, the clock network is still treated as ideal, but with the specified latency, uncertainty, and skew values instead of zero throughout the clock network.

By default, the `place_opt` command does not perform clock tree synthesis. However, if the clock tree is relatively simple and if it uses the same routing rules as signal routes, you can perform clock tree synthesis simultaneously with placement. In that case, you can use propagated delays instead of ideal clocking during placement and optimization. To do so, use the `-cts` option in the `place_opt` command.

The `place_opt` command does not perform clock tree synthesis by default. However, it does perform synthesis of high-fanout nets such as reset and scan-enable signals. When skew is an issue with a high-fanout net, you should compile it separately with the `compile_clock_tree -high_fanout_net` command.

Clock Tree Synthesis

By default, clock networks are considered ideal during logic synthesis and physical placement. An ideal clock network consists of a single driver connected to all the sequential devices in the design clocked by that clock, without any buffers. An ideal network can have zero latency, uncertainty, or transition times; or you can specify nonzero latency, uncertainty, and transition times that are uniform throughout the clock network.

Clock tree synthesis is the replacement of an ideal clock network with a hierarchy of buffers with the goal of minimizing latency, skew, and transition times. This is typically done after placement and high-fanout net synthesis and before routing. The `clock_opt` command performs clock tree synthesis, including logic synthesis of the buffer tree, routing of the clock nets, RC extraction from synthesized clock network, and clock tree optimization. You can optionally perform hold time fixing, interclock delay balancing, area recovery, and other clock tree optimization functions.

You can perform the clock tree synthesis in steps for greater control and to closely monitor and analyze the results as they are generated. For example, the following commands perform clock tree synthesis without routing, followed by clock tree optimization without routing, and finally routing of the optimized clock nets:

```
icc_shell> clock_opt -only_cts -no_clock_route
(timing analysis of compiled clock tree here)
...
icc_shell> clock_opt -only_psyn -no_clock_route
```

```
(timing analysis of optimized clock tree here)
...
icc_shell> route_group -all_clock_nets
(timing analysis of fully routed clock tree here)
...
```

You can also choose to perform the clock tree synthesis steps separately using commands such as `compile_clock_tree` and `optimize_clock_tree`. To use propagated rather than ideal latency and transition time, you can use the commands `set_propagated_clock` and `remove_ideal_network`. For details, see the applicable man pages.

Several commands let you specify the parameters for clock tree synthesis before you use the `clock_opt` command. Use the `set_clock_tree_references` command to specify which buffer and inverter cells to use for implementing the clock tree. Use the `set_clock_tree_options` commands to specify the layers used for routing clock nets, the target delay and skew, the maximum number of buffer levels, the maximum capacitance, transition time, and fanout rules for the buffers used in the tree. Use the `set_clock_tree_exceptions` command to apply exceptions to normal clock tree synthesis behavior to specified objects in the design, such as stop pins, nonstop pins, exclude pins, “don’t-buffer” nets, “don’t-size” cells, and size-only cells.

You can apply nondefault routing rules to the routing of clock nets such as shielding, extra width, or extra spacing to reduce the effects of parasitic resistance, electromigration, and crosstalk. You specify nondefault routing rules with the `define_routing_rule` command and then invoke those rules with the `-routing_rule` option of the `set_clock_tree_options` command.

To check and fix hold violations, use the `-fix_hold_all_clocks` or `-only_hold_time` option with the `clock_opt` command. The `-fix_hold_all_clocks` option fixes hold violations for all clocks. The `-only_hold_time` option fixes only hold time violations and does nothing else.

Routing

Routing is the process of defining physical connections to all clock and signal pins through metal paths. The routed paths must meet setup and hold timing requirements, maximum capacitance and transition time design rules, and clock skew requirements. Furthermore, the metal traces must meet physical design rules such as minimum width and minimum spacing.

Cell placement and clock tree synthesis must be completed before routing starts. The power and ground nets must be routed. The estimated congestion must be acceptable and the estimated timing must show the worst slack to be at least zero or near zero. There can be no maximum capacitance or maximum transition violations.

Routing consists of three main stages: global routing, track assignment, and detail routing. In global routing, the router divides the chip area into global routing cells and assigns routes to the available tracks in each routing cell. The router avoids congested cells by

detouring routes around the congested areas. It considers the additional time delay of the longer routes. In track assignment, the router assigns each net to a specific track within each global routing cell and attempts to make long, straight traces with as few vias as possible. In detail routing, the router fixes physical design rule violations such as minimum width and minimum spacing. By default, the clock nets are routed first, then the signal nets.

You can use either Zroute or the classic router to perform routing. Zroute, the default router, is recommended because of its higher quality of results. You can specify the router with the `set_route_mode_options` command.

The `route_opt` command performs global routing, track assignment, detail routing, and route optimization. The command options let you specify the effort level and the types of optimization performed such as wire sizing, cell sizing, area recovery, leakage power, and fixing of hold violations. A higher effort level produces more highly optimized results at the cost of more runtime. You can choose to limit the routing stages performed (global, track, or detail) or perform incremental optimization of a previously routed design.

By default, the router does not check or fix crosstalk conditions. To invoke crosstalk checking and repair during routing, use the following commands:

```
icc_shell> set_si_options -delta_delay true or -static_noise true ...
...
icc_shell> route_opt -xtalk_reduction or -only_xtalk_reduction ...
```

Crosstalk issues can also be reduced by applying nondefault routing rules for clocks and other sensitive signals, enforcing maximum transition rules, reducing congestion, and limiting the extent of area recovery performed during optimization.

Timing Analysis After Routing

After routing, detailed nets are available for accurate RC extraction, thereby producing more accurate delay calculation results. The `report_constraint` command reports timing violations, and the `report_timing` command generates detailed reports on the critical paths.

For a postroute design, by default, the tool computes delays using the Elmore or Arnoldi delay models, based on the detailed parasitic information for the nets in the design. The Elmore model is faster but not as accurate as the Arnoldi model. The asymptotic waveform evaluation (AWE) model is also available as an option; it has most of the speed of the Elmore model and most of the accuracy of the Arnoldi model.

To choose the Elmore model for minimum runtime, use the following command:

```
set_delay_calculation_options -postroute elmore
```

To choose the Arnoldi model for maximum accuracy, use the following command:

```
set_delay_calculation_options -postroute arnoldi -arnoldi_effort high
```

You can set the `-arnoldi_effort` option to `low`, `medium` (the default), `hybrid`, or `high`. [Table 2](#) lists these option settings in order of increasing accuracy and runtime.

Table 2 Arnoldi Postroute Delay Calculation Effort Options

-arnoldi_effort option	Postroute delay calculation models used
<code>low</code>	Asymptotic waveform evaluation (AWE) delay model only
<code>medium</code>	A combination of the Arnoldi and Elmore delay models (the default)
<code>hybrid</code>	A combination of the Arnoldi and AWE delay models
<code>high</code>	Arnoldi delay model only

Faster delay models are appropriate for early design iterations. The high-effort Arnoldi delay model is appropriate for final signoff.

In timing reports generated by the `report_timing` command, the following character symbols appear next to the time value in the “Incr” column to indicate the type of timing model used for delay calculation:

```
& Postroute Elmore
@ Arnoldi
c CCS (composite current source)
a Postroute AWE
* Preroute Elmore or back-annotated delay
w Preroute AWE
```

For a complete and final design, you should use the StarRC signoff extraction tool and a the PrimeTime signoff timing tool. The goal of a signoff tool is to verify with the greatest possible accuracy that the design will work properly with respect to timing. The signoff tools perform extraction and timing analysis more accurately than the tools that perform synthesis, physical implementation, and optimization.

The StarRC tool is a full-chip parasitic RC extraction tool that considers all capacitive interactions between conductors and accurately models physical characteristics of wires such as dishing, erosion, and physical proximity of nearby features. The tool extracts billions of capacitors for a typical design and uses an advanced parasitic reduction method to generate the smallest possible netlist that can achieve accurate timing analysis results.

The PrimeTime tool is a full-chip static timing analyzer that uses the same libraries, gate-level netlist, parasitic RC data, and SDC timing constraints as the Design Compiler and IC Compiler tools. The PrimeTime tool performs a comprehensive analysis with maximum speed and accuracy, producing results that are very consistent with SPICE simulation. The PrimeTime Suite has optional add-on tools that perform crosstalk delay analysis, noise analysis, and power analysis.

The `run_signoff` command performs signoff analysis by invoking the Star RC and PrimeTime tools in the background. The StarRC tool performs signoff-quality parasitic extraction, and the PrimeTime tool performs signoff-quality timing and crosstalk analysis. You can use the signoff costing from these tools to perform optimization on the design with the `signoff_opt` command. The commands for setting up the signoff tools are `set_starrcxt_options` and `set_primestime_options`.

To set up, report, and check the IC Compiler variable settings for consistency with PrimeTime timing analysis, use the following commands:

```
enable_primestime_icc_consistency_settings
check_primestime_icc_consistency_settings
report_primestime_icc_consistency_settings
```

Synthesis Design Rules

Synthesis design rules are technology-specific constraints that must be met for a design to function correctly. These rules include minimum and maximum capacitance, maximum transition time, and maximum fanout. Design rule constraints are different from timing constraints such as clock period, input delay, and output delay, and also different from optimization constraints such as area and power.

Synthesis and optimization tools attempt to meet all constraints placed on a design, but by default, they give highest priority to design rule constraints. Meeting the timing constraints is also a requirement for correct operation, but design rule constraints are given higher priority. Area and power optimization constraints reflect goals that are desirable, but not absolutely required for correct operation.

Design rules are typically specified in the technology, but they can also be specified by commands in the synthesis, optimization, or analysis tools. [Table 3](#) lists the SDC commands used for specifying synthesis design rules. These commands do not directly affect timing analysis or timing optimization because design rules are enforced separately from optimization goals. However, the enforcement of design rules provides a basis for ultimately achieving the timing goals of the design.

Table 3 SDC Design Rule Commands

Command	Usage
<code>set_fanout_load</code>	Specifies the number of external loads in the fanout of one or more output ports of the design. The synthesis or optimization tool uses this information to enforce the maximum fanout design rule.
<code>set_max_capacitance</code>	Specifies the maximum allowed capacitance for specified input ports or all input ports of a design. The synthesis or optimization tool ensures that the loading of the input port is small enough to keep the total capacitance no more than the specified maximum value.

Table 3 SDC Design Rule Commands (Continued)

Command	Usage
set_max_fanout	Specifies the maximum allowed fanout for specified input ports or all input ports of a design. The synthesis or optimization tool ensures that the number of loads connected to the input port is small enough to keep the fanout no more than the specified maximum value.
set_max_transition	Specifies the maximum allowed signal transition time for a net, applied to nets belonging to specified clock groups, specified ports, or the whole design. The synthesis or optimization tool ensures that the transition time is no more than the specified maximum value.
set_min_capacitance	Specifies the minimum allowed capacitance of specified input ports or all input ports of a design. The synthesis or optimization tool ensures that the loading of the input port is large enough to keep the total capacitance at least the specified minimum value.

To remove all SDC constraints from the design, including both timing constraints and design rule constraints, use the `remove_sdc` command.

In some cases, there can be a conflict between design rule constraints and timing constraints. For example, the maximum fanout design rule might require the insertion of buffers in a high-fanout net, thus increasing the path delay and possibly causing timing violations. By default, design rules have a higher “cost priority” than timing constraints. The tool first attempts to meet all design rules without causing timing violations, but if this is not possible, it satisfies the design rules and allows timing violations to occur.

You can optionally change the order of priority with the `set_cost_priority` command. For example, to set the cost priority of timing delay at the top:

```
prompt> set_cost_priority -delay
```

By increasing the cost priority of delay, the tool still attempts to fix design rule violations such as maximum fanout, but not at the expense of delay. Any remaining design rule violations can be addressed in the back-end or physical design. The command also lets you set the relative priority of design rules, as in the following example:

```
prompt> set_cost_priority {max_fanout max_capacitance max_delay}
```

This command sets the cost priority highest for maximum fanout, followed by maximum capacitance, maximum delay, and then by the unlisted design rules. To find out if the constraints are met, use the `report_constraint` command.

2

Clocks

An essential part of timing analysis is to accurately specify clocks and clock effects, such as latency and uncertainty. You can specify, report, and analyze clocks as described in the following sections:

- [Creating Clocks](#)
 - [Clock Network Effects](#)
 - [Multiple Clocks](#)
 - [Clock Sense](#)
 - [Pulse Clocks](#)
 - [Minimum Pulse Width Checks](#)
 - [Clock-Gating Signal Timing Checks](#)
 - [Generated Clocks](#)
 - [Estimated I/O Latency](#)
 - [Propagated Clocks](#)
-

Creating Clocks

You must specify all of the clocks in the design by using the `create_clock` command. This is the command syntax:

```
create_clock
  [source_objects]
  [-period period_value]
  [-waveform edge_list]
  [-name clock_name]
  [-add]
```

The `source_object` is the place in the design where the clock exists, which is usually an input port but can also be a pin inside the design. The analysis tool traces the clock network so that the clock reaches all registers in the transitive fanout of the source object. If you do not specify a source object, you define a *virtual clock*, which is a clock used as

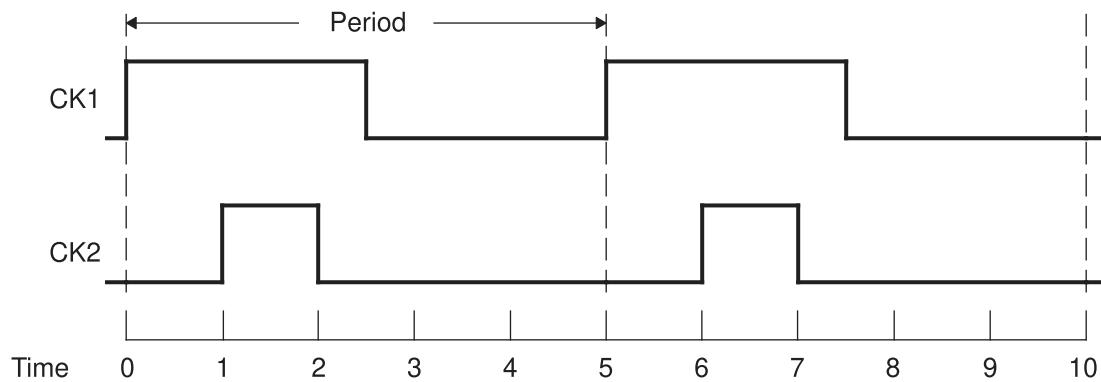
a reference for setting input delays and output delays but does not physically exist in the design.

Each clock should have its period defined with the `-period` option. This is the period of time over which the same waveform repeats end-to-end.

By default, a clock has a 50 percent duty cycle with a rising edge at time zero and a falling edge at one-half the period. If the clock does not have this simple form, you specify the waveform with the `-waveform` option and provide a list of rising and falling edge times within the clock period. For example, the following commands create a clock having the default waveform and another clock with a rising edge at 1.0 and a falling edge at 2.0. The resulting clock waveforms are shown in [Figure 22](#).

```
prompt> create_clock -period 5.0 [get_ports CK1]
prompt> create_clock -period 5.0 -waveform {1.0 2.0} [get_ports CK2]
```

Figure 22 Clock Waveforms



You can optionally specify a name for the clock with the `-name` option. If you do not specify a name explicitly, the clock gets its name from the source object. The `-add` option lets you add multiple clocks to the same source object in the design.

By default, a clock created with the `create_clock` command is ideal. It has zero delay at the source object, zero propagation delay, zero transition time, and zero uncertainty. For an accurate timing analysis, you need to specify clock characteristics such as latency, skew, uncertainty, and transition time. To specify the source latency or network latency of the clock, use the `set_clock_latency` command. To add skew or uncertainty to an ideal clock, use the `set_clock_uncertainty` command. To specify the transition time, use the `set_clock_transition` command. Alternatively, to enable calculation of propagated delay through the network using wire parasitic data, use the `set_propagated_clock` command.

To get a report on clocks that have been defined for the design, use the `report_clock` command. To create a collection of clocks, use the `get_clocks` or `all_clocks` command. For example, to generate a report on the clocks having names starting with PHI1 and a period less than or equal to 5.0, use the following command:

```
prompt> report_clock [get_clocks -filter "period <= 5.0" PHI1*]
```

The `get_clocks` and `get_ports` commands can be used to distinguish between a clock object and port object that share the same name.

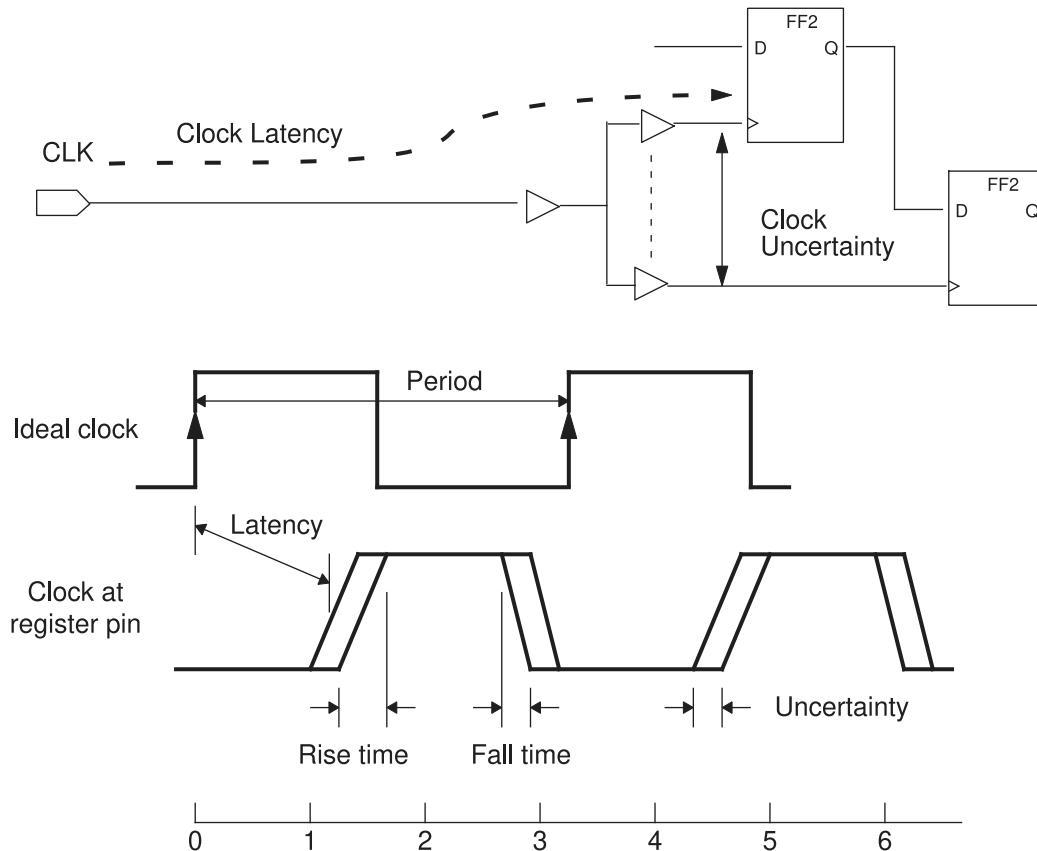
To get a report on the transitive fanout of a clock source, use the `report_transitive_fanout` command. The report lists the successive driver and load pins fanning out from the specified source object. Similarly, to get a report on the transitive fanin of a clock sink, use the `report_transitive_fanin` command. The report shows the successive driver and load pins in the path feeding into the sink object.

To remove a defined clock, use the `remove_clock` command. The `reset_design` command also removes all clocks as well as other design information.

Clock Network Effects

For accurate timing analysis, you must describe the clock network. The main characteristics of a clock network are latency, uncertainty, and transition time. [Figure 23](#) shows the timing effects of clock networks.

Figure 23 Clock Network Effects



Latency is the amount of time it takes for the clock signal to be propagated from the original clock source to the sequential elements in the design. It consists of two components, source latency and network latency. Source latency is the delay from the clock source to the clock definition pin in the design. Network latency is the delay from the clock definition point to the register clock pin.

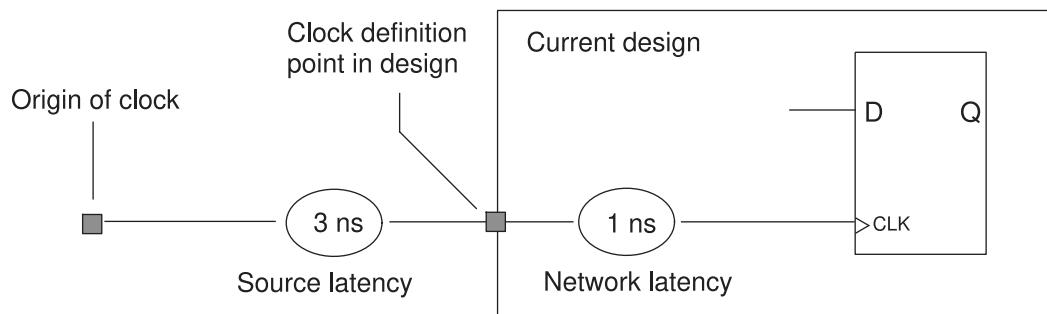
Uncertainty is the maximum difference between the arrival of clock signals at registers in one clock domain or between domains. This is also called skew. The larger the skew, the more difficult it is to meet the timing constraints.

Transition time is the amount of time it takes for a signal to change from logic low to logic high (rise time), or from logic high to logic low (fall time). The transition time of a signal at an input of a cell affects the delay to the output of the cell and the transition time of the output signal.

Clock Latency

Latency is the amount of time it takes for the clock signal to be propagated from the original clock source to the sequential elements in the design, consisting of two components, source latency and network latency. Source latency, also known as insertion delay, is the time it takes for a clock to be propagated from its ideal waveform origin point to the clock definition point in the design. Network latency is the time it takes for a clock to be propagated from the clock definition point in the design to a register clock pin. See [Figure 24](#).

Figure 24 Source and Network Latency



The timing analysis tool provides two methods for representing clock latency. You can do either of the following:

- Allow the tool 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.

Propagated Latency

You can have the tool 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.

To propagate clock network delays and automatically determine latency at each register clock pin, enter

```
prompt> 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 attribute, use the `remove_propagated_clock` command.

Ideal Network 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.

The `set_clock_latency` command sets the latency for one or more clocks, ports, or pins. This is the command syntax:

```
set_clock_latency
[-rise] [-fall]
[-min] [-max]
[-source]
[-early] [-late]
[-clock clock_list]
delay
object_list
```

For example, to set the expected rise latency to 1.2 and the fall latency to 0.9 for CLK, enter

```
prompt> set_clock_latency -rise 1.2 [get_clocks CLK]
prompt> set_clock_latency -fall 0.9 [get_clocks CLK]
```

You must specify the delay value and the list of objects affected by the command. The object list can contain one or more clocks, ports, or pins.

Use the `-rise` or `-fall` option to restrict the latency setting to only rising or only falling edges of the clock. Otherwise, the setting applies to both rising and falling edges.

Use the `-min` or `-max` option to restrict the latency setting to only the minimum operating condition or only the maximum operating condition. Otherwise, the setting applies to all operating conditions.

The `-source` option sets source latency, rather than network latency, on the ports or pins specified in the object list. Use the `-early` and `-late` options in separate commands to specify different early and late latency values, as described in the next section.

The `-clock` option lets you specify the relevant clock when you set the latency on a port or pin and multiple clocks can pass through the port or pin.

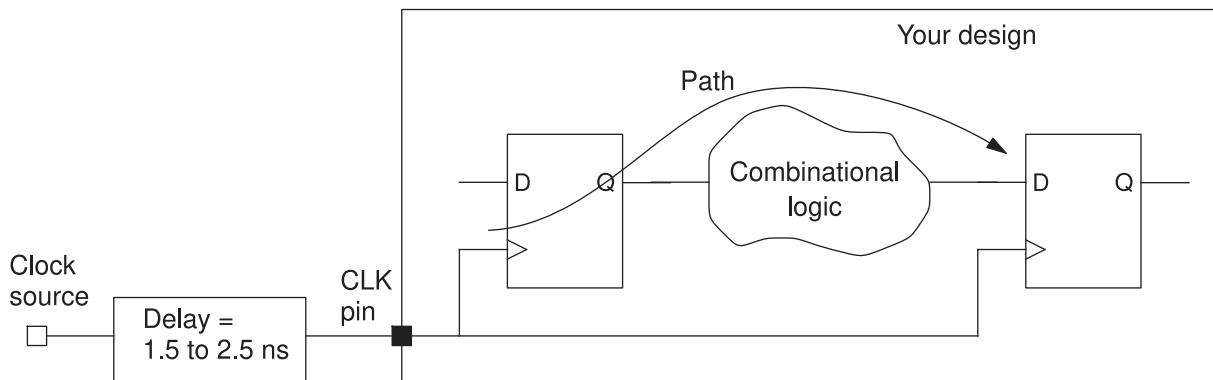
The `remove_clock_latency` command removes user-specified clock network or source clock latency information from specified objects.

Source Latency

Source latency is the latency from the ideal waveform to the source object in the design. You can specify source latency for both ideal and propagated clocks. The total latency at a register clock pin is the sum of the source latency and network latency.

To specify an external uncertainty for source latency, use the `-early` and `-late` options of the `set_clock_latency` command. For example, consider a source latency that can vary from 1.5 to 2.5 ns, as illustrated in [Figure 25](#).

Figure 25 External Source Latency

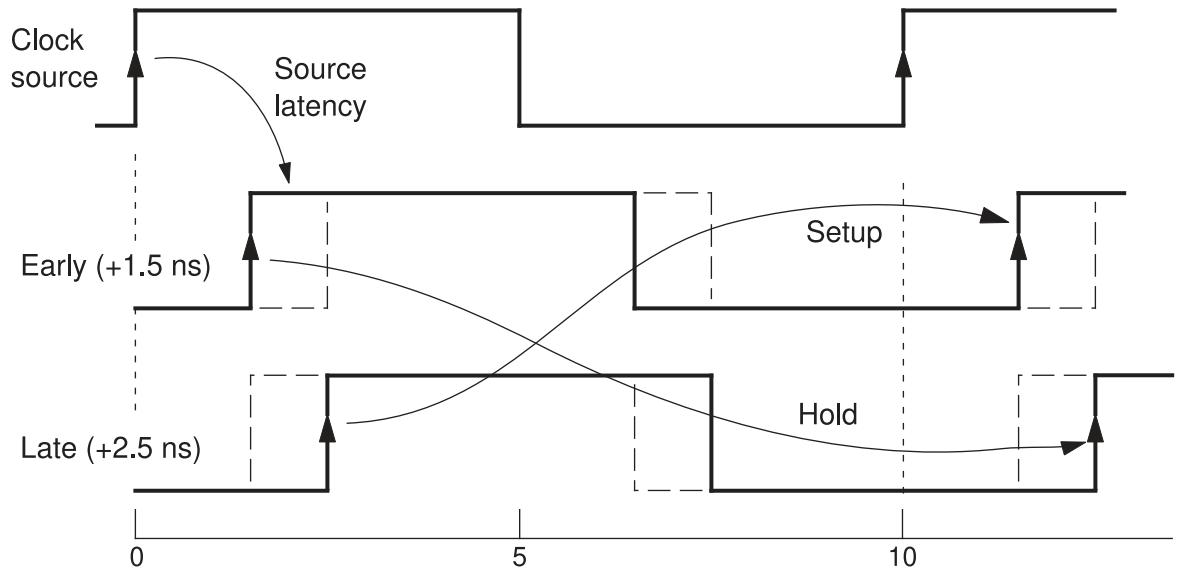


To specify this type of source latency, you can use commands such as the following:

```
prompt> create_clock -period 10 [get_ports CLK]
prompt> set_clock_latency 1.5 -source -early [get_clocks CLK]
prompt> set_clock_latency 2.5 -source -late [get_clocks CLK]
```

The tool uses the more conservative source latency value (either early or late) for each startpoint and endpoint clocked by that clock. For example, for a setup check, it uses the late value for each startpoint and the early value for each endpoint. [Figure 26](#) 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.

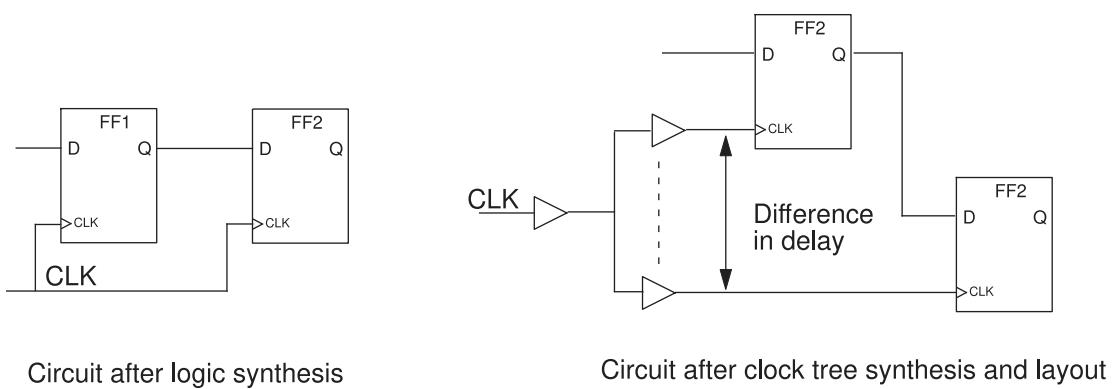
Figure 26 Early/Late Source Latency Waveforms



Clock Uncertainty

Setting clock uncertainty is a way to incorporate a margin of error in the design to account for possible variances in the clock propagation times in the post-layout design. [Figure 27](#) illustrates the concept of clock uncertainty.

Figure 27 Clock Uncertainty



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 arriving at sequential devices. Before clock tree synthesis, clock uncertainty is caused by clock jitter, which is the variation in the clock edge times of the source clock, as well as clock skew, which is the difference in clock arrival times resulting from different propagation delays from the chip's clock pins to different sequential devices in the chip. After clock tree synthesis, with propagated latency, the tool separately accounts for uncertainty resulting from different propagation delays through the clock tree.

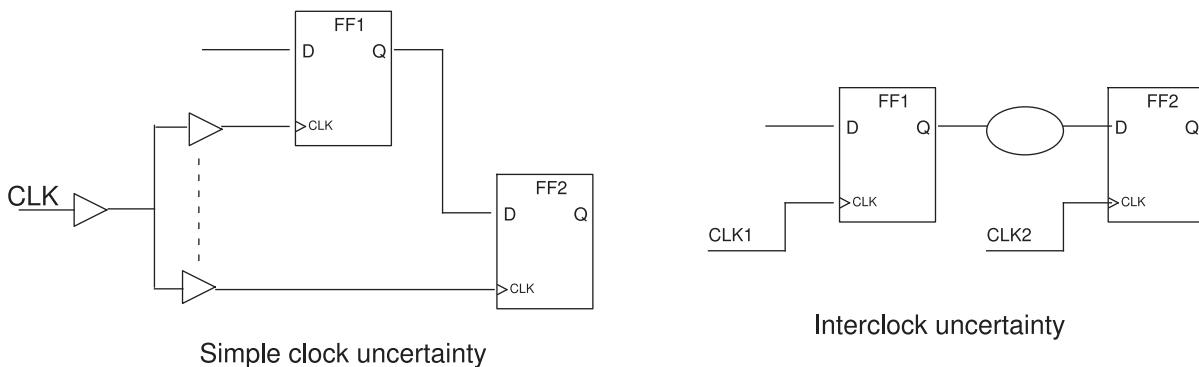
This is the command syntax:

```
set_clock_uncertainty
[object_list |
 -from from_clock | -rise_from from_clock | -fall_from from_clock
 -to to_clock | -rise_to to_clock | -fall_to to_clock]
[-rise] [-fall]
[-setup] [-hold]
uncertainty
```

You must specify the uncertainty time value and either an object list or “from” and “to” clocks. Specifying an object list invokes simple uncertainty for a clock, whereas specifying “from” and “to” clocks invokes interclock uncertainty between the two clocks.

Simple uncertainty is the variation in the generation of successive edges of a clock with respect to the exact, nominal times. Interclock uncertainty is the variation in skew between edges of different clocks. [Figure 28](#) shows the difference between these two types of uncertainty.

Figure 28 Simple and Interclock Uncertainty

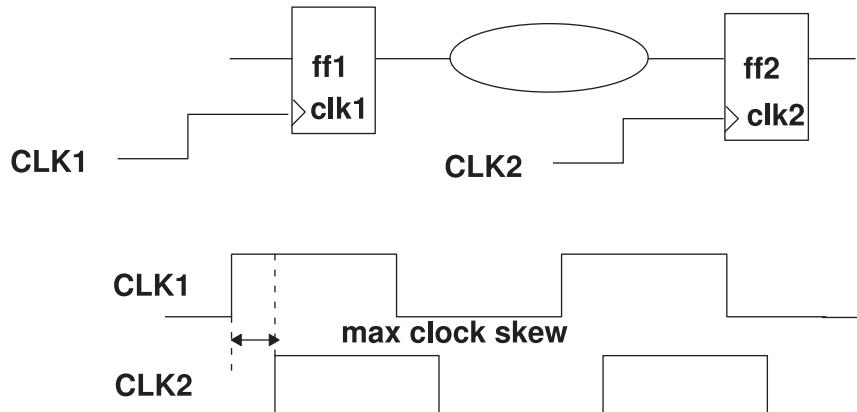


For simple uncertainty, you specify one or more objects, which can be clocks, ports, or pins. The uncertainty value applies to all capturing latches clocked by the specified clock or whose clock pins are in the fanout of the specified ports or pins.

For interclock uncertainty, 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. [Figure 29](#) shows the interclock skew between two clocks defined with the same period and waveform.

Figure 29 Example of Interclock Uncertainty



In performing a setup or hold check, the tool 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.

You can use `set_clock_uncertainty` to model clock network skew and `set_clock_latency -source` to model variations in the source clock delay, such as source clock jitter or off-chip clock skew.

When a path has both simple clock uncertainty and interclock uncertainty, the interclock uncertainty value is used. For example,

```
prompt> set_clock_uncertainty 5 [get_clocks CLKA]
prompt> 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 two clock domains. If you have paths from CLKA to CLKB and from CLKB to CLKA, you must specify the uncertainty for both directions, even if the value is the same. For example,

```
prompt> set_clock_uncertainty 2 -from [get_clocks CLKA] \
    -to [get_clocks CLKB]
prompt> 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

```
prompt> 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

```
prompt> set_clock_uncertainty -setup 0.21 [get_clocks CLK1]
prompt> set_clock_uncertainty -hold 0.33 [get_clocks CLK1]
```

To remove clock uncertainty settings, use the `remove_clock_uncertainty` command.

Ideal Clock Transition Times

For propagated clocks, the tool calculates the clock transition time at each net. For ideal clocks, the default transition time is zero. To specify a nonzero transition time for an ideal clock, use the `set_clock_transition` command. This is the command syntax:

```
set_clock_transition
  transition_time
  [-rise] [-fall]
  [-min] [-max]
  clock_list
```

For example,

```
prompt> 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 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 cancel the effects of this command, use the `remove_clock_transition` command.

Reporting Clock Information

The `report_clock` command displays information about all clocks and generated clocks in the current design. For example,

```
prompt> report_clock
...
Attributes:
  d - dont_touch_network
  f - fix_hold
```

p - propagated_clock
G - generated_clock

Clock	Period	Waveform	Attrs	Sources
PCI_CLK	15.00	{0 7.5}		{pclk}
SDRAM_CLK	7.50	{0 3.75}		{sdram_clk}
SD_DDR_CLK	7.50	{0 3.75}	G	{sd_CK}
SYS_CLK	8.00	{0 4}		{sys_clk}

Generated Clock	Master Source	Generated Source	Master Clock	Waveform Modification
SD_DDR_CLK	sdram_clk	{sd_CK}	SDRAM_CLK	divide_by(1)

Use the `-skew` option of the `report_clock` command to report the uncertainty of the clocks.

Multiple Clocks

By default, when multiple clocks reach a register clock pin, the tool considers all of the clocks simultaneously. Therefore, it considers the interactions between different clocks operating on a given timing path, such as launch by one clock and capture by a different clock.

If some clock interactions are valid and others are not, you can specify the invalid ones by using the `set_false_path` command, as in the following example:

```
prompt> set_false_path -from [get_clocks CK2] -to [get_clocks CK4]
```

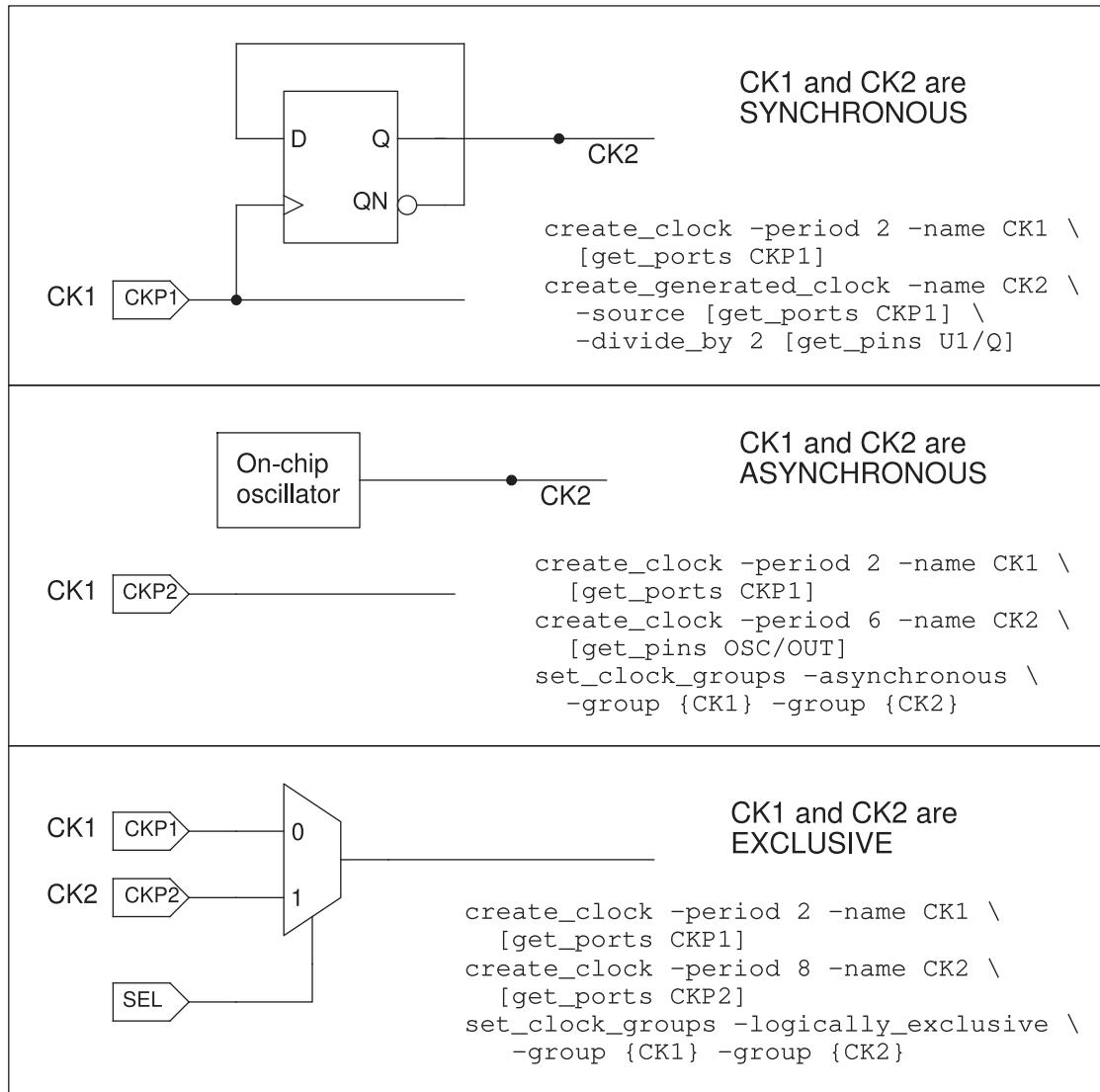
You can also define exclusive clock groups using the `set_clock_groups` command, as described in [Exclusive Clocks on page 72](#).

If the circuit is designed to use only one clock at a time everywhere, you can disable analysis of multiple clocks by setting the following variable:

```
prompt> set_app_var timing_enable_multiple_clocks_per_reg false
```

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 in the design. The relationship between two clocks can be synchronous, asynchronous, or exclusive, as shown by the examples in [Figure 30](#).

Figure 30 Synchronous, Asynchronous, and Exclusive Clocks



For the tool 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

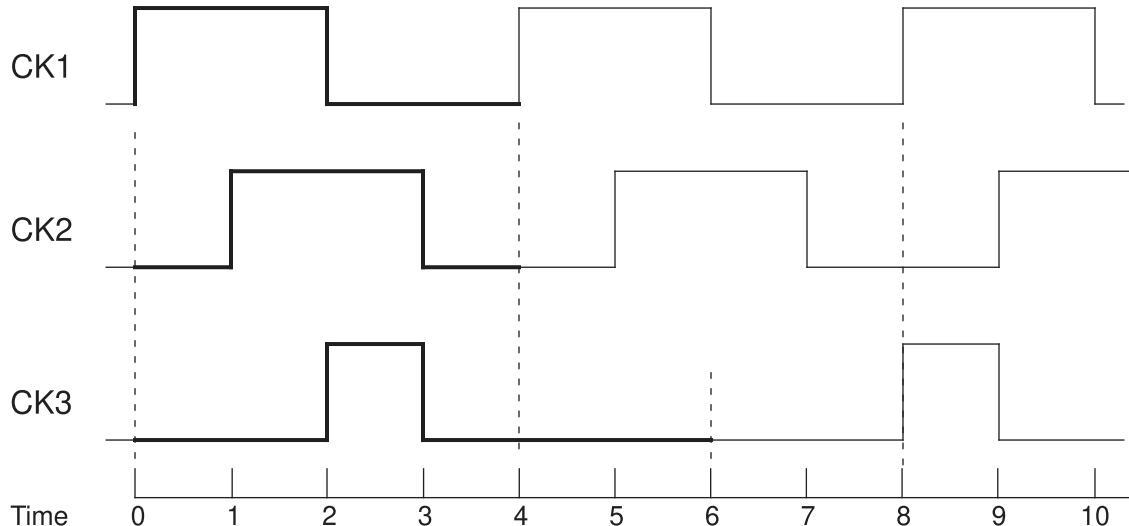
Two clocks are synchronous with respect to each other if they share a common source and have a fixed phase relationship. Unless you specify otherwise, the tool assumes

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:

```
prompt> create_clock -period 4 -name CK1 -waveform {0 2}
prompt> create_clock -period 4 -name CK2 -waveform {1 3}
prompt> create_clock -period 6 -name CK3 -waveform {2 3}
```

The tool creates the clocks as specified in the commands, with the waveforms synchronized as shown in [Figure 31](#). It adjusts the timing relationships further for any specified or calculated latency or uncertainty.

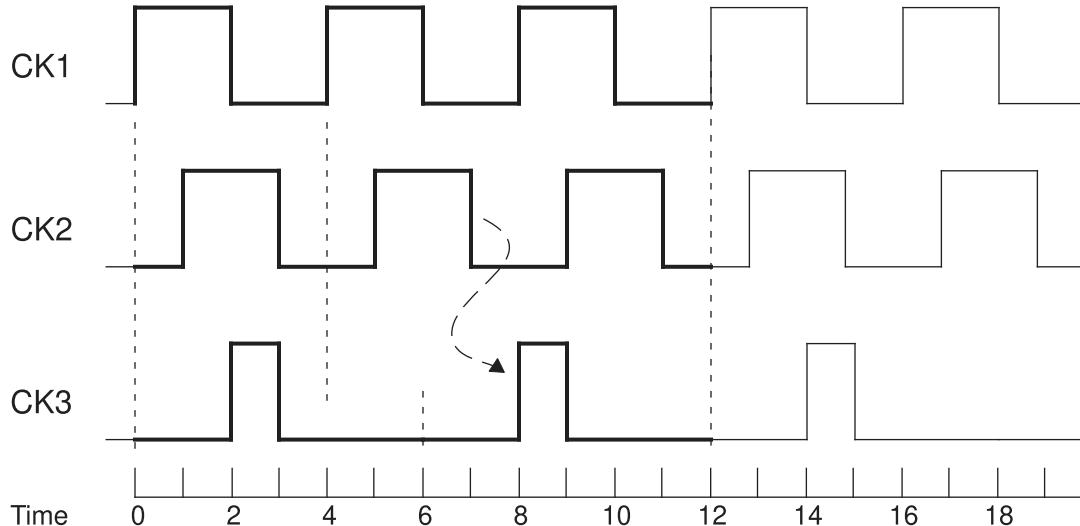
Figure 31 Synchronous Clock Waveforms



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, the tool 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, the tool internally expands the clocks by repeating them over the base period. The resulting clock waveforms are shown in [Figure 32](#). Each expanded clock has a period of 12.

Figure 32 Expanded Clock Waveforms



The tool checks timing paths between all edges in the expanded clocks. For example, the most restrictive setup check between a falling edge of CK2 and a rising edge of CK3 is from time=7 to time=8, as shown by the dashed arrow in Figure 32.

The `report_interclock_relation` command provides information about the common clock period used for paths between registers driven by different clocks. For example,

```
prompt> report_interclock_relation
...
```

From	Period1	Count1	To	Period2	Count2	Common
CK1	4.00	3	CK3	6.00	2	12.00
CK3	6.00	2	CK1	4.00	3	12.00

Each line of the report shows a pair of clocks: one that launches a timing path and one that captures the data at the end of the path. It also shows the periods and period multiplier of each clock, followed by the common multiple of the two periods used for timing analysis.

You should define multiple clocks in a manner consistent with the way they actually operate in the design. To declare clocks that are not synchronous, you can use case analysis or commands such as `set_clock_groups -logically_exclusive` or `set_false_path`.

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 the relationship between two clocks to be asynchronous. In that case, the tool does not check the timing paths launched by one clock and captured by the other clock. This is like declaring a false path between the two clocks. 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 the tool from spending time analyzing 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 be considered.

To declare clocks CK1 and CK2 to be logically exclusive:

```
prompt> set_clock_groups -logically_exclusive -group {CK1} -group {CK2}
```

This causes the tool 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. To find out about clock groups that have been set, use `report_clock -groups`.

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:

```
prompt> set_clock_groups -logically_exclusive \
          -group {CK1 CK2} -group {CK3 CK4}
```

This causes the tool 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,

```
prompt> 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 in the design. For example,

```
prompt> 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:

```
prompt> set_clock_groups -logically_exclusive -name EX1 \  
-group {CK1 CK2} -group {CK3 CK4}
```

Use the `remove_clock_groups` command to remove a clock grouping declaration:

```
prompt> remove_clock_groups -logically_exclusive EX1
```

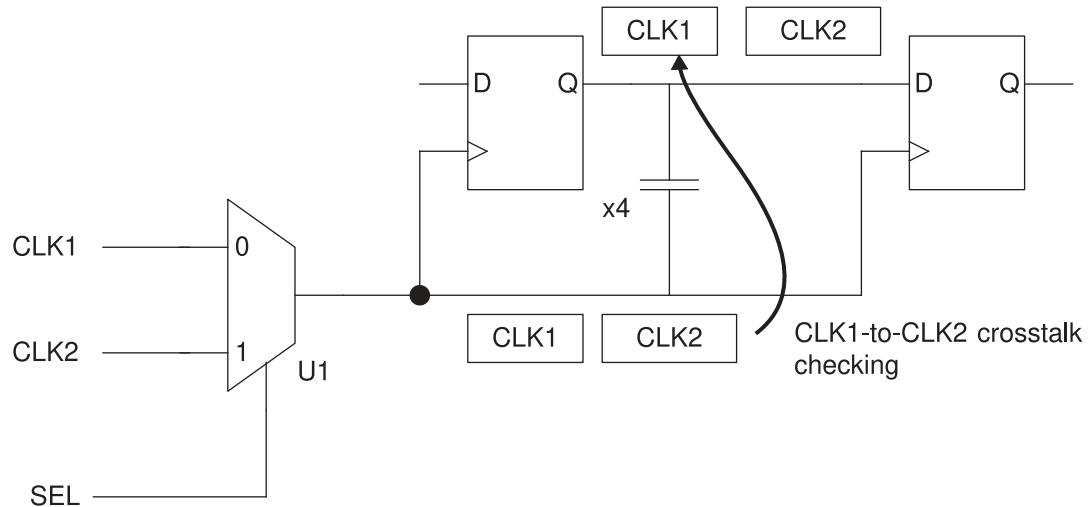
To remove all exclusive clock grouping declarations made with the `set_clock_groups` command:

```
prompt> remove_clock_groups -logically_exclusive -all
```

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 `-logically_exclusive`. This prevents the tool from attempting to perform crosstalk analysis between the clock nets.

For example, consider the circuit shown in [Figure 33](#). Only one of the two input clocks is enabled at any given time.

Figure 33 Circuit With Multiplexed Clocks



To analyze both clocks at the same time, you can define them to be logically exclusive:

```
prompt> set_clock_groups -logically_exclusive \
           -group {CLK1} -group {CLK2}
```

The `-logically_exclusive` option causes the tool to suppress any logical (timing path) checking between CLK1 and CLK2. However, it still computes crosstalk delta delays across coupling capacitor x4 between the two clocks, which is pessimistic because the two clocks are never simultaneously present on the nets. To eliminate this pessimism, define the clocks to be physically exclusive:

```
prompt> set_clock_groups -physically_exclusive \
           -group {CLK1} -group {CLK2}
```

Then the tool ignores any crosstalk between the nets of the physically exclusive clocks, as well as suppressing the logical checking between the clocks. This eliminates the pessimistic analysis of crosstalk between CLK2 and CLK1 across capacitor x4.

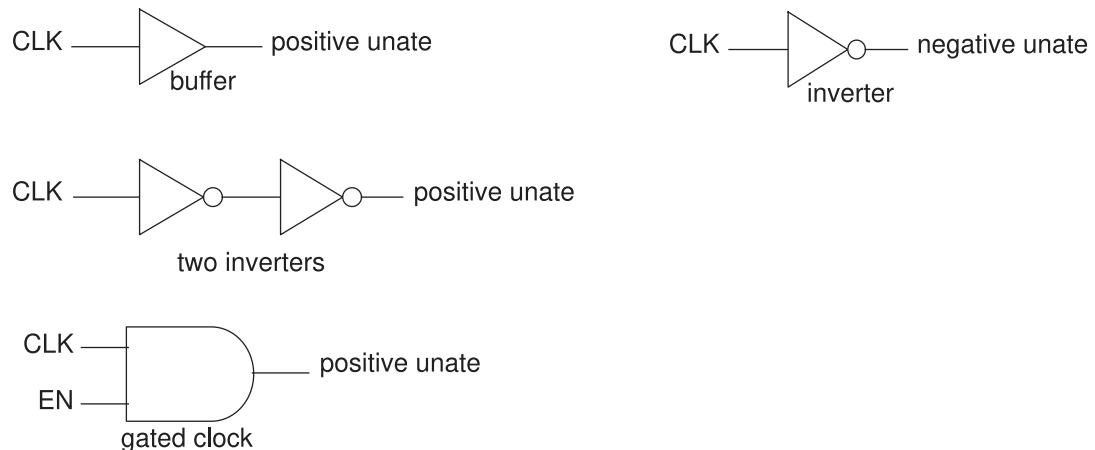
Clock Sense

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. No specific action is necessary 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 can only cause a rising edge at the register clock pin, and a falling edge at the clock source can only cause a falling edge at the register clock pin.

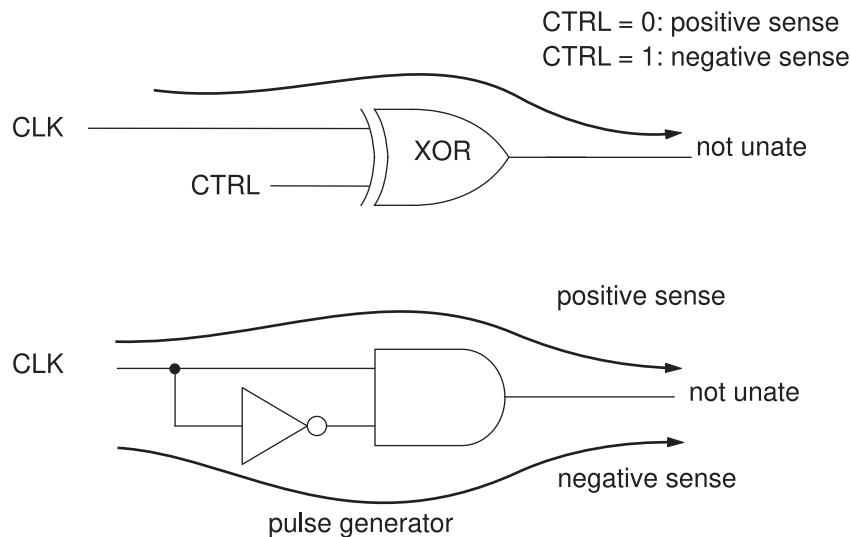
Similarly, a clock signal is “negative unate” if a rising edge at the clock source can only cause a falling edge at the register clock pin, and a falling edge at the clock source can only cause a rising edge at the register clock pin. In other words, the clock signal is inverted. See [Figure 34](#).

Figure 34 Positive and Negative Unate Clock Signals



A clock signal is not unate if the clock sense is ambiguous as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate because there are non-unate arcs in the gate. The clock sense could be either positive or negative, depending on the state of the other input to the XOR gate, as shown in [Figure 35](#).

Figure 35 Non-Unate Clock Signals



The tool considers the output of the pulse generator at the bottom of Figure 35 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, you can specify the sense of a clock signal at a point in the clock path using the `set_sense` command. For example,

```
prompt> set_sense -positive [get_pins xor1.z]
```

This command tells the tool to propagate only the positive unate paths through the output pin of the XOR gate, with respect to the original clock source. From that point onward, the tool keeps track of the sense of the signal through any subsequent buffers or inverters.

The positive unate setting applies to any clock that passes through the specified pin. If multiple clocks can reach that pin, you can restrict the setting to certain clocks, as in the following example:

```
prompt> set_sense -positive \
           -clocks [get_clocks CLK] [get_pins mux1.z]
```

The `set_sense` setting has an effect only in the non-unate part of a clock network. If the command is applied to a pin in a unate section of the clock network and the requested sense disagrees with the actual sense, it generates an error message and the setting is ignored.

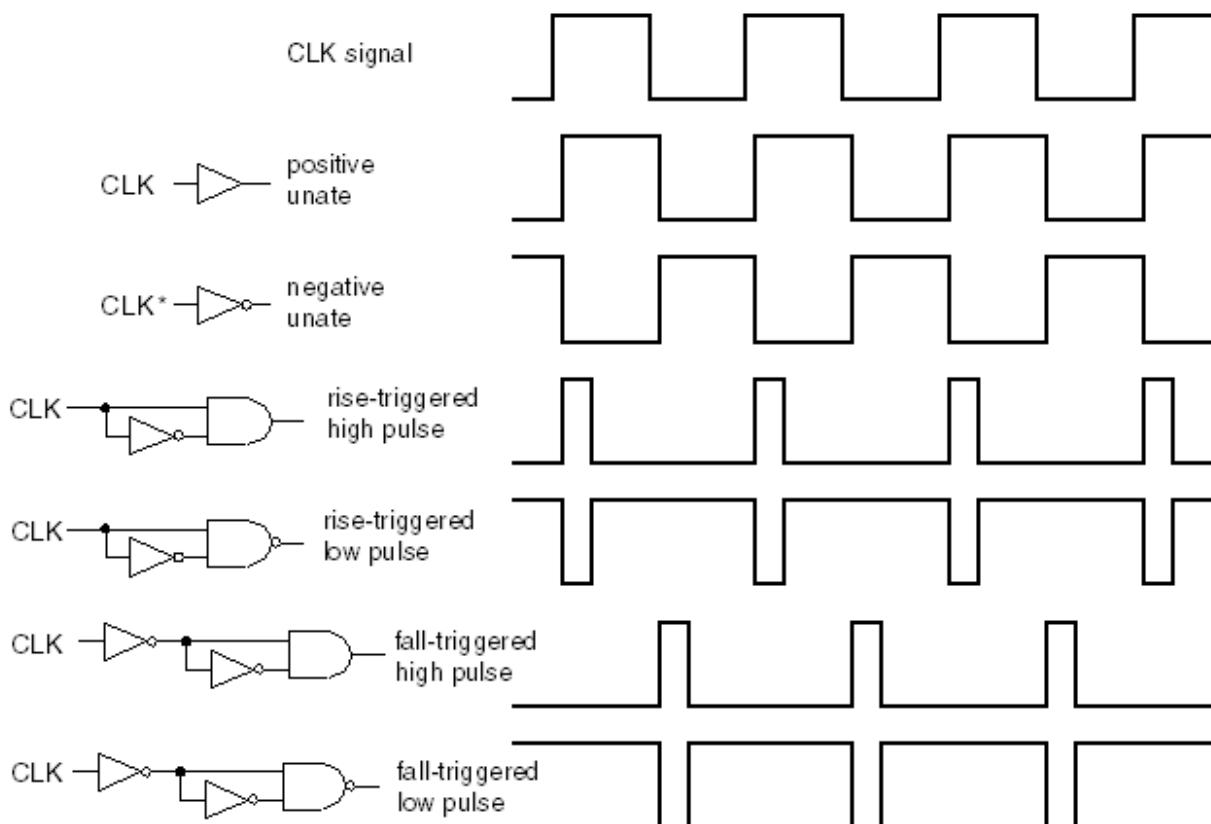
Specifying the clock sense at a point in the design uses one of the following forms of syntax:

```
set_sense -positive object_list
set_sense -negative object_list
```

The object list specifies the pins in the design where the sense is being defined. If multiple clocks pass through the objects, you can specify the clocks affected by the command by using the `-clock` option. To reverse the effects of `set_sense`, use the `remove_sense` command.

[Figure 36](#) shows some examples of clock-modifying circuits and the corresponding clock senses.

Figure 36 Clock Sense Examples



To stop clock propagation forward from a specified pin, use the `-stop_propagation` option of the `set_sense` command. This stops propagation of the clocks specified in the clock list from the specified pins or cell timing arcs in the `object_list`.

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.

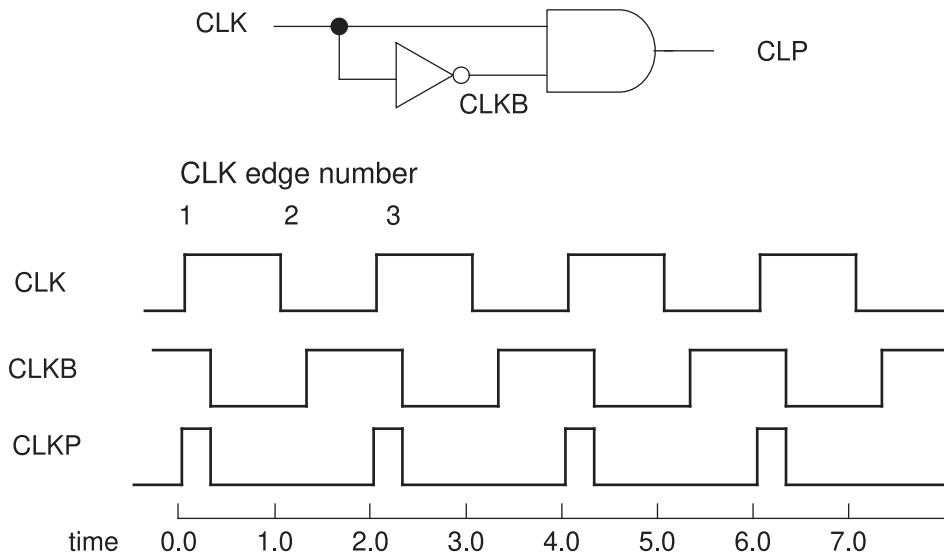
To analyze the timing of a circuit containing pulse clocks, the tool needs information about the timing characteristics of the clock. There are three ways to provide this information:

- 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` 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 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 in the design, using either the `create_generated_clock` or `set_sense` command. Using the `create_generated_clock` command creates a new clock domain at a pulse generation point. Using `set_sense` does not create a new clock domain, but merely specifies the sense for an existing clock downstream from the specified point. For example, consider the pulse clock circuit shown in [Figure 37](#).

Figure 37 Pulse Clock Specified as a Generated Clock



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 CLPK as a generated clock:

```
nt_shell> create_generated_clock -name CLPK -source CLK \
    -edges {1 1 3} [get_pins and2/z]
```

Specifying the generated clock as a pulse clock using repeated edge digits ensures correct checking of delays between the source clock and the pulse clock.

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` command to specify the sense of the existing clock:

```
prompt> set_sense -pulse rise_triggered_high_pulse \
    [get_pins and2/z]
```

This 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 of the source clock downstream from the specified point in the clock network, and that sense is the “rise-triggered high pulse” sense.

In general, the clock sense of a pulse clock can be specified at a location in the design by one of the following forms of syntax:

```
set_sense -pulse rise_triggered_high_pulse object_list
set_sense -pulse rise_triggered_low_pulse object_list
set_sense -pulse fall_triggered_high_pulse object_list
set_sense -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` command. To determine the actual pulse width, the tool considers the different rise and fall latency values at the pulse generator output pin:

$$(\text{high pulse width}) = (\text{fall network latency}) - (\text{rise network latency})$$

$$(\text{low pulse width}) = (\text{rise network latency}) - (\text{fall network latency})$$

You can use the `set_clock_latency` command to specify the latency values (and therefore the pulse width) explicitly for an ideal clock, or you can allow the tool to calculate the propagated latency from the circuit for a propagated clock. For example, to set an ideal pulse width to 0.5 for high pulses, for all registers downstream from pin z of gate and2, and with an overall latency of 0.6, the commands would be:

```
prompt> set_clock_latency -rise 0.6 [get_pins and2.z]
prompt> set_clock_latency -fall 1.1 [get_pins and2.z]
```

Minimum Pulse Width Checks

Minimum pulse width checks are important to ensure proper operation of sequential circuits. The pulse width of the original clock might be reduced due to differences in the propagation paths for rising and falling clock edges. 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 defined. If so, the `report_constraint` command checks for violations of these constraints. To view these constraints, use the `report_lib` command.

To explicitly set the pulse width constraints for specific clocks and clock pins, use the `set_min_pulse_width` command. To check for pulse width violations, you can use either the `report_constraint` or `report_min_pulse_width` command. For example,

```
prompt> report_min_pulse_width
...

```

Pin	Required pulse width	Actual pulse width	Slack	Scenario
ff1/cp(low)	2.00	0.87	-1.13 (VIOLATED)	
ff2/cp(low)	2.00	1.41	-0.59 (VIOLATED)	
ff2/cp(high)	0.60	0.59	-0.01 (VIOLATED)	
ff1/cp(high)	0.60	1.13	0.53 (MET)	

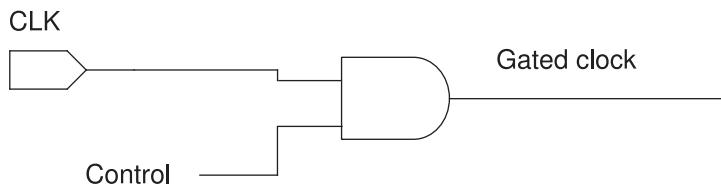
To report the minimum pulse width constraints that have been set, use the `report_min_pulse_width` command. To remove these constraints, use the `remove_min_pulse_width` command.

For more information about setting pulse width constraints, see the man page for the `set_min_pulse_width` command.

Clock-Gating Signal Timing Checks

A gated clock signal occurs when a 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 signal. See [Figure 38](#).

Figure 38 Gated Clock



The tool does not automatically check setup and hold violations on the gating signals of clock-gated cells. It is possible for these signals to undergo transitions while clock pulses are passing through the gating cells. This can lead to both clipped and spurious clock pulses.

To check for such conditions, use the `set_clock_gating_check` command. This is the command syntax:

```
set_clock_gating_check
  [-setup setup_margin]
  [-hold hold_margin]
  [-rise] [-fall]
  [-high [-low]
  [object_list]
```

Using this command, you can specify setup and hold margins to control gating signal transitions. Use the `-setup` option to ensure that the clock-gating input is stable for a given time interval before the clock input of the gating cell changes to a noncontrolling value. Similarly, use the `-hold` option to ensure that the clock-gating signal remains stable for a given time interval after the clock input returns to a controlling value. Used together, the two checks ensure that the clock-gating signal is stable for the entire period of time during which the gated clock input has a noncontrolling value.

If you want only the rising or only the falling delays constrained by the `set_clock_gating_check` command, use the `-rise` or the `-fall` option, respectively. If you do not specify either of these options, both types of delays are constrained.

You can list the design objects on which clock-gating setup and hold margins are to be checked, such as designs, cells, pins, or clock objects. Multiplexer cells are supported. Also, when you specify a clock object, all clock-gating cells in the given clock's network are checked. If you do not specify any design objects, the clock-gating check is applied to the current design.

In the case of certain cells that the tool cannot analyze, you must designate the noncontrolling value of the clock. Use the `-high` or `-low` option to define the noncontrolling clock value as high or low, respectively. You can use these options only with cells or pins.

The tool handles clock-gating checks like other timing constraints and tries to adjust the delays of the logic driving the gating inputs to avoid setup and hold violations. During optimization, Design Compiler can change only the size of cells with clock-gating checks. The logic functions of such cells are preserved and no other logic transformations are allowed.

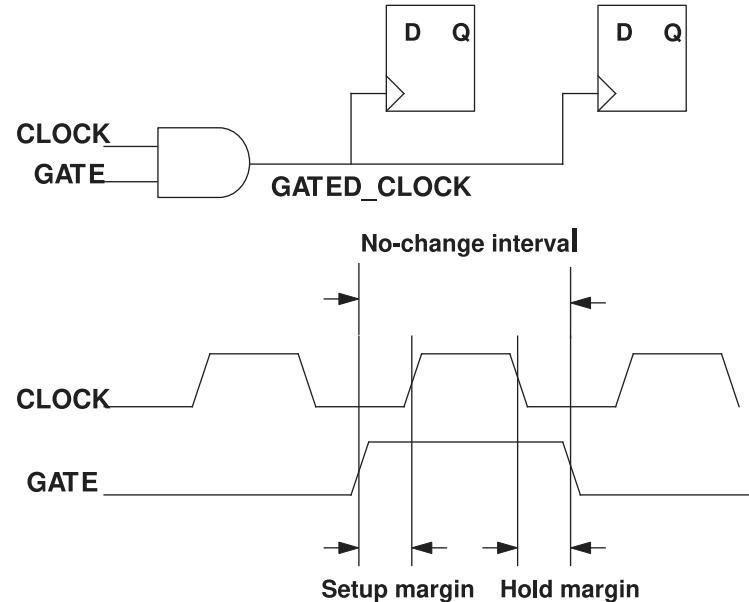
Clock-gating checks can be performed only between a clock signal and a nonclock signal, not between two clock signals or between two nonclock signals.

If you decide that clock-gating checks are no longer needed, you can remove them with the `remove_clock_gating_check` command. This command has the same options as the `set_clock_gating_check` command.

In [Figure 39](#), the `set_clock_gating_check` command can be used to define the setup and hold margins shown at the AND and NAND clock-gating cells. The setup margin is

measured relative to the rising transition of the gating cell's clock input. The hold margin is measured from the falling transition of the clock input.

Figure 39 Setup and Hold Margins for AND and NAND Gates



[Figure 40](#) shows setup and hold margins for OR and NOR clock-gating cells. The setup margin is measured relative to the falling transition of the gating cell clock input, while the hold margin is measured relative to the rising transition of the clock input. [Figure 41](#) shows examples of distorted clock waveforms that might be caused by invalid changes on the clock-gating inputs in the absence of clock-gating checks.

Figure 40 Setup and Hold Margins for OR and NOR Gates

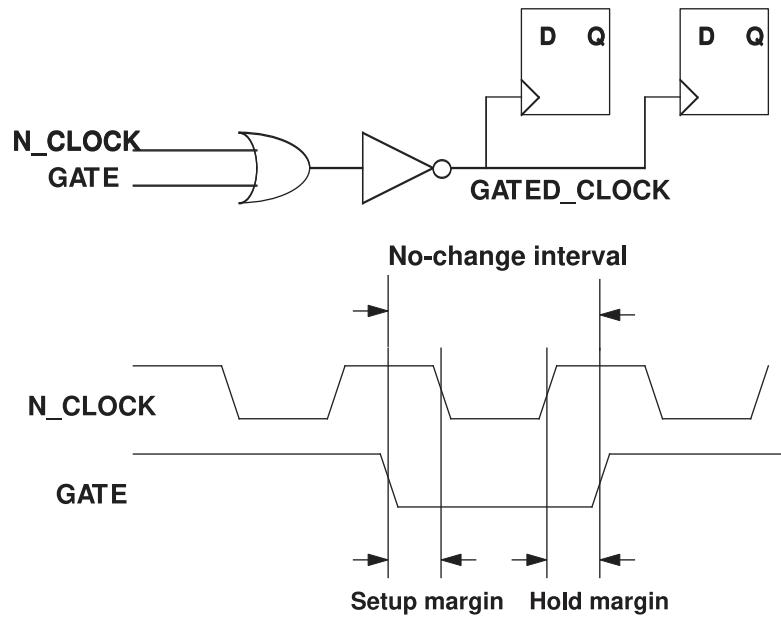
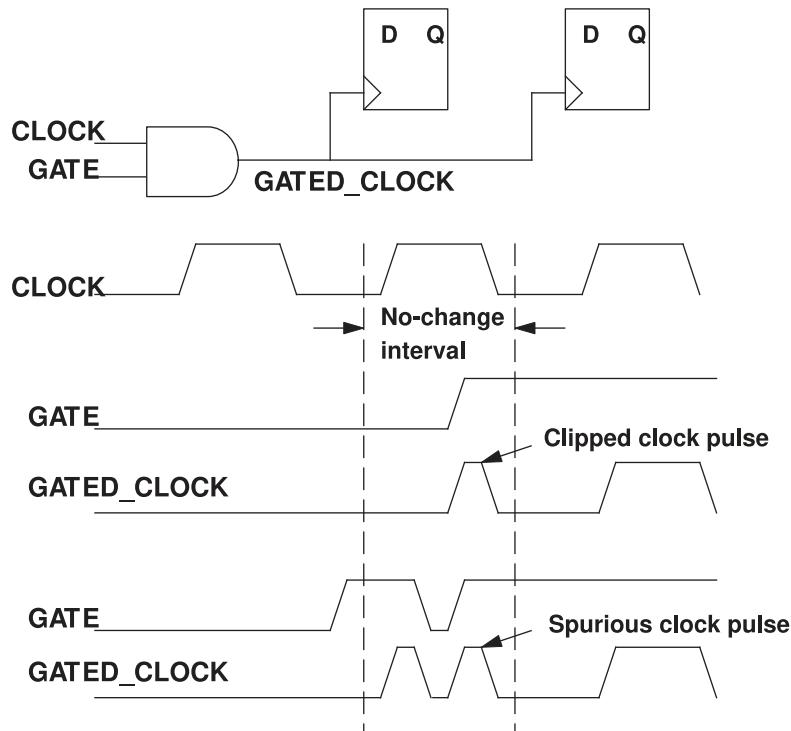


Figure 41 Distorted Clock Waveforms



Example 1

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

```
prompt> set_clock_gating_check -setup 0.2 -hold 0.4 CLK1
```

Example 2

To specify a setup requirement of 0.5 on gate and1, enter

```
prompt> set_clock_gating_check -setup 0.5 and1
```

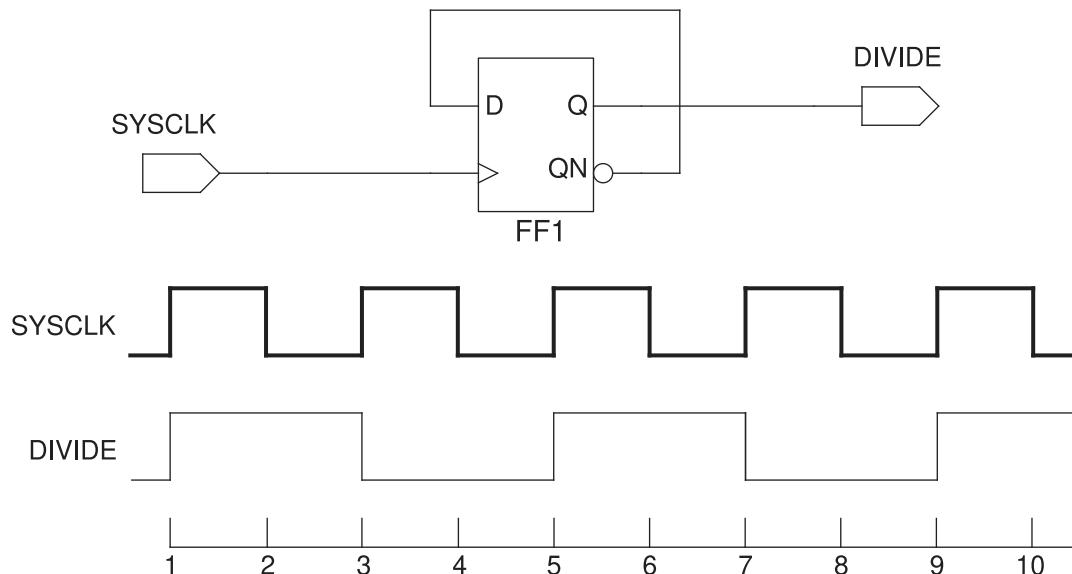
You can disable clock-gating checks on specific cells and pins. Use the `set_disable_clock_gating_check` command to list the cells and pins you want the clock-gating check to ignore. To cancel the effect of this command, use the `remove_disable_clock_gating_check` command.

Generated Clocks

A design might include clock dividers or other structures that produce a new clock from a master source clock. A clock that is generated by on-chip logic from another clock is called a generated clock.

[Figure 42](#) shows an example of a divide-by-2 generated clock. 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 42 Divide-by-2 Clock Generator



The tool does not derive the behavior of the generated clock from the logic, so you must specify the behavior as a separate clock. You can do so with the `create_clock` command. However, there are certain advantages to using the `create_generated_clock` command instead.

The `create_generated_clock` command specifies the characteristics of an internally generated clock in terms of the master clock. You specify one or more source objects in the design where the generated clock exists, a pin where the master clock exists, and the time relationship between the master clock and the generated clock, such as divide-by-2. The tool determines the generated clock characteristics based on the master clock. If the period or latency of the master clock changes, the generated clock also changes accordingly.

To remove a generated clock definition, use the `remove_generated_clock` command.

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 42](#) shown previously, specify 2 as the frequency division factor:

```
prompt> create_generated_clock -name DIVIDE \
           -source [get_ports SYSCLK] -divide_by 2 [get_pins FF1/Q]
```

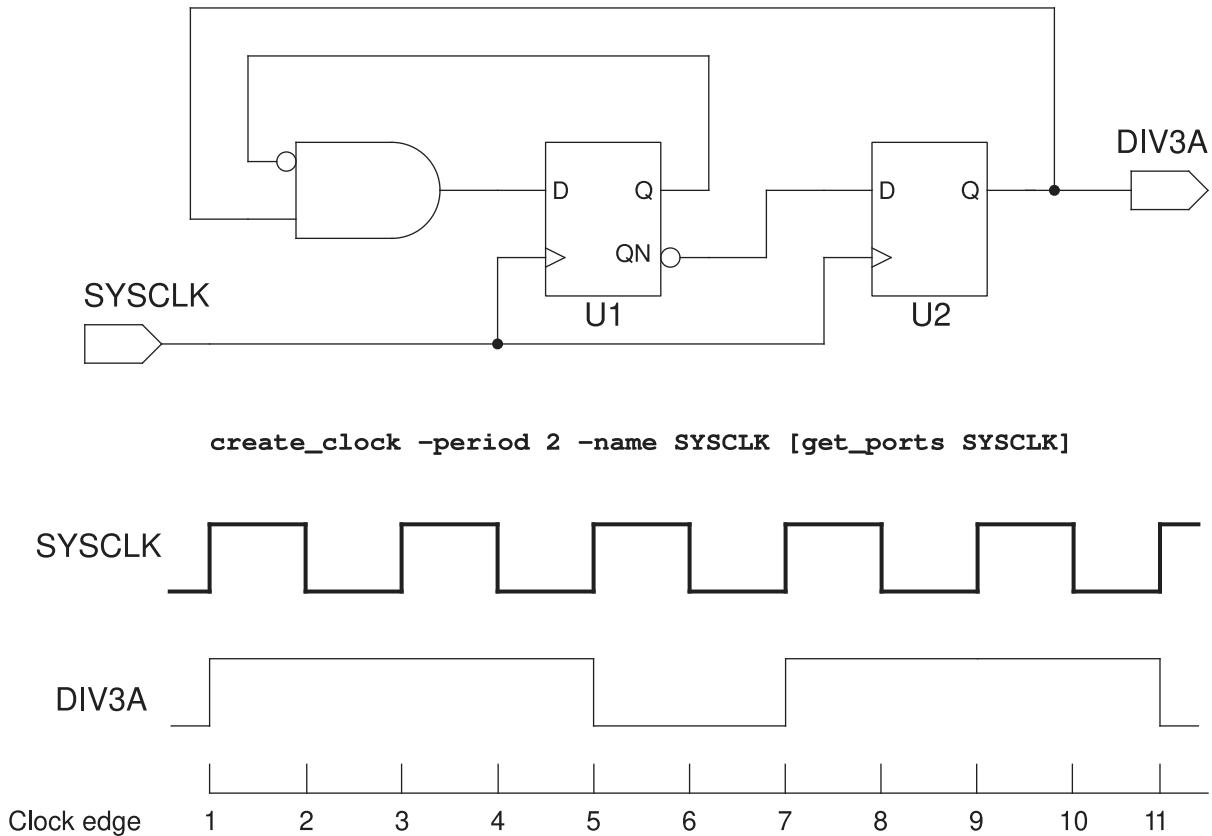
Specify a port or a pin (not a clock name) as the master source from which the new clock is generated. Specify a pin as the creation point for the new generated clock.

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 [Divide-by Clock Based on Falling Edges](#)).

Generated Clock Based on Edges

You can use `create_generated_clock -edges` to specify the generated clock in terms of edges of the master clock waveform on the master pin. For example, consider the clock generator circuit shown in [Figure 43](#).

Figure 43 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, you can enter

```

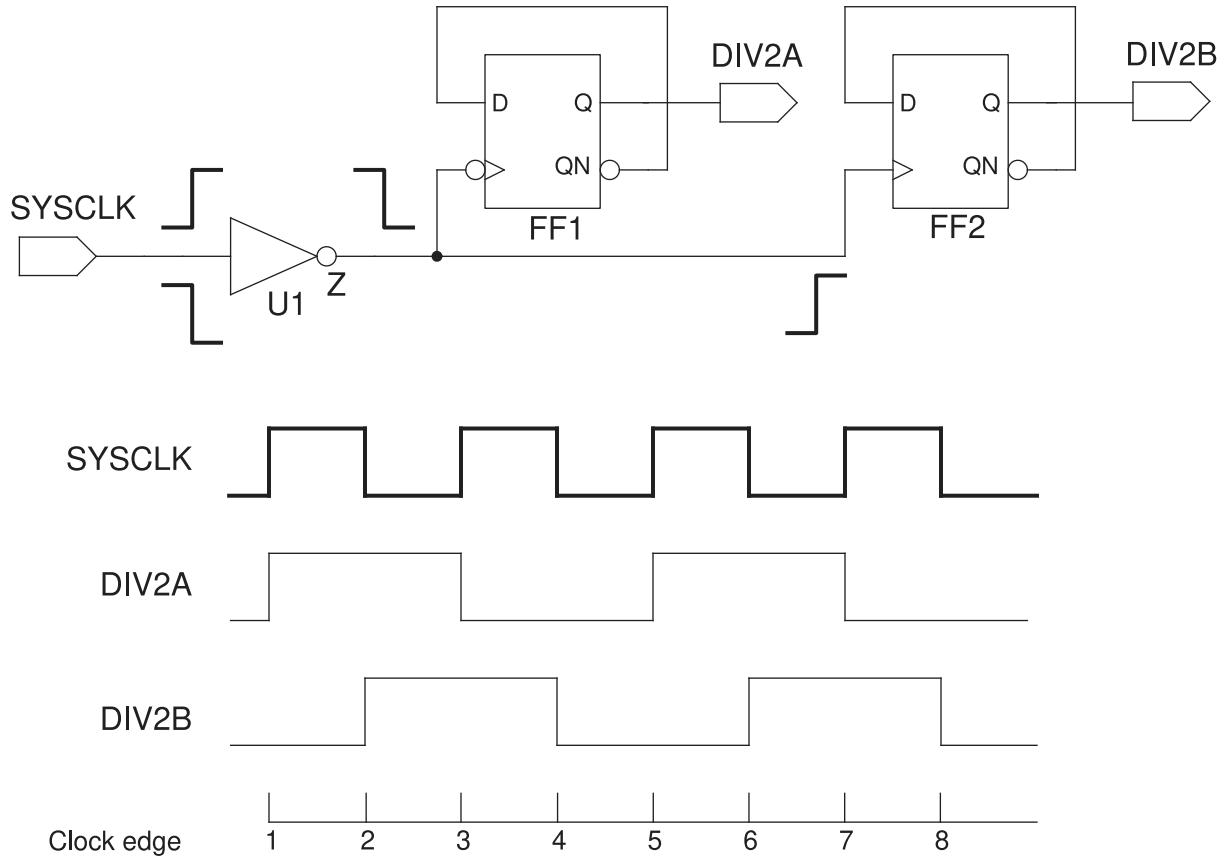
prompt> create_generated_clock -edges { 1 5 7 } \
          -name DIV3A -source [get_ports SYSCLK] [get_pins U2/Q]

```

Divide-by Clock Based on Falling Edges

If you need to create a generated clock based on falling edges at the master clock pin, use the `create_generated_clock` command with the `-edges` option. The generated clock examples in [Figure 44](#) demonstrate how to do this.

Figure 44 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:

```
prompt> create_generated_clock -name DIV2A \
           -source [get_ports SYSCLK] -divide_by 2 [get_pins FF1/Q]
prompt> 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:

```
prompt> create_generated_clock -name DIV2B \
           -source [get_ports SYSCLK] -edges { 2 4 6 } [get_pins FF2/Q]
```

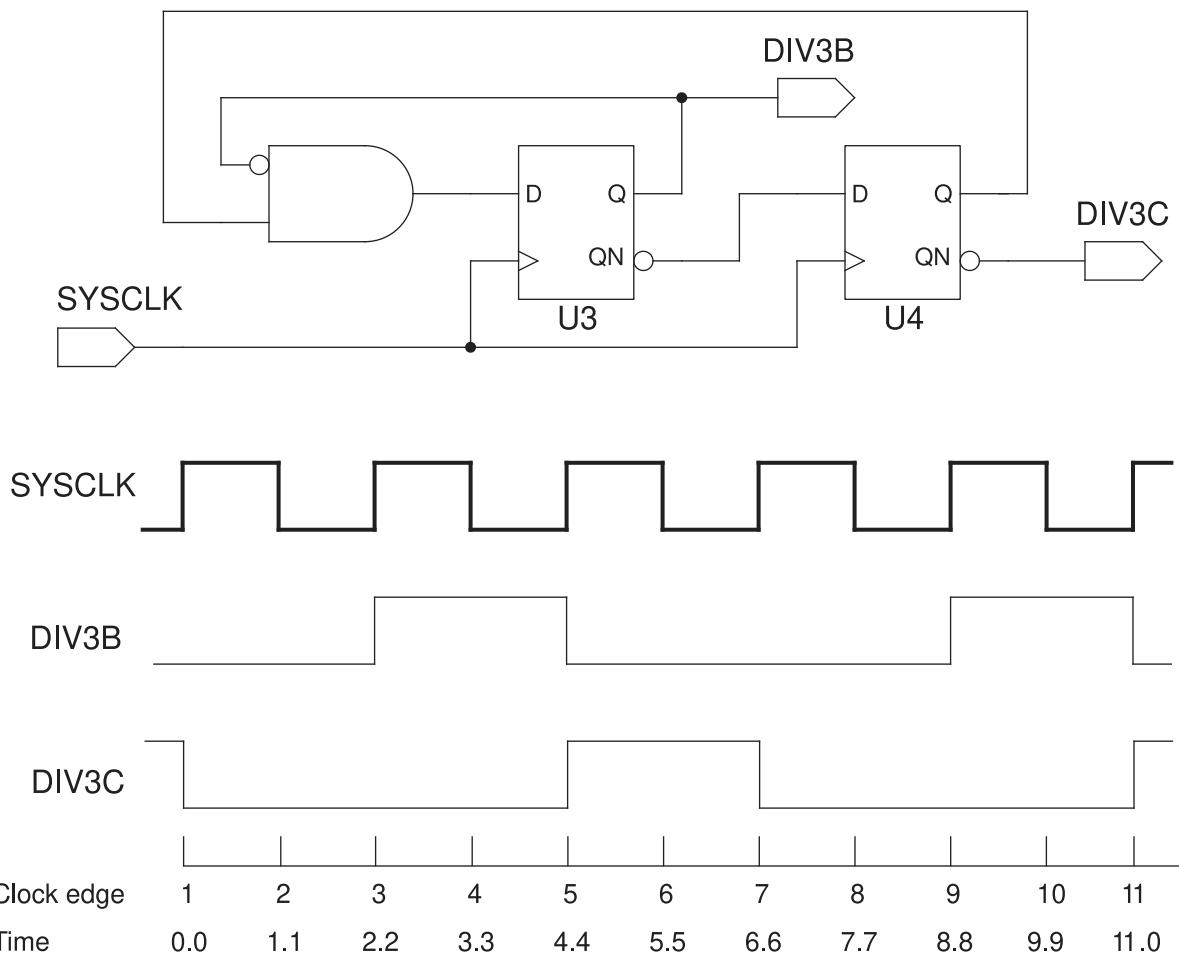
Another way to specify DIV2B is to use a different source pin:

```
prompt> create_generated_clock -name DIV2B \
           -source [get_pins U1/Z] -edges { 1 3 5 } [get_pins FF2/Q]
```

Shifting the Edges of a Generated Clock

You can shift the edges of a generated clock by a specified amount of time. This shift is not considered clock latency. For example, consider the clock generator circuit shown in [Figure 45](#).

Figure 45 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:

```
prompt> create_clock -period 2.2 -name CLK [get_ports SYSCLK]
prompt> create_generated_clock -edges { 3 5 9 } \
    -name DIV3B -source [get_ports SYSCLK] [get_pins U3/Q]
prompt> 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]
```

The `report_clock` command reports the specified clocks as follows:

p - propagated_clock
G - Generated clock

Clock	Period	Waveform	Attrs	Sources
-------	--------	----------	-------	---------

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
-----------------	---------------	------------------	-----------------------

DIV3B	MYCLK	U3/Q	edges(3 5 9)
DIV3C	MYCLK	U4/Q	edges(3 5 9) shifts(2.2 2.2 2.2)

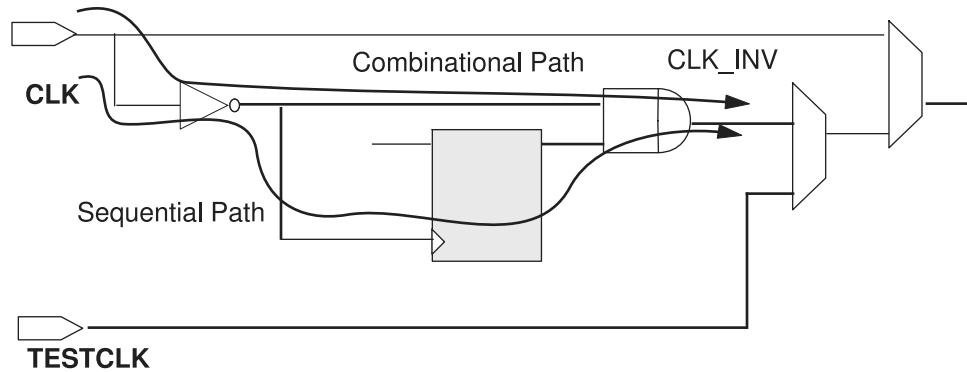
Combinational-Only Source Latency Calculation

During timing analysis, when the tool calculates clock source latency of a generated clock, it traces both combinational and sequential paths between the source pin of a generated clock and the source pin of its master clock. A sequential path is any path containing a register clock pin, a data pin of a transparent latch, or a generated clock definition point other than the final pin in the path.

In some situations, you might want the tool to avoid these sequential paths when it is calculating clock latency. For example, in [Figure 46](#), the generated clock `CLK_INV` is specified by using the following command:

```
prompt> create_generated_clock -name CLK_INV \
    -source [get_ports CLK] -divide_by 1 get_pins A/Y
```

Figure 46 Paths in a Generated Clock Source Network



To have the tool avoid sequential paths, use the `-combinational` option with the `create_generated_clock` command. For example,

```
prompt> create_generated_clock -name CLK_INV \
           -source [get_ports CLK] -divide_by 1 [get_pins A/Y] \
           -combinational
```

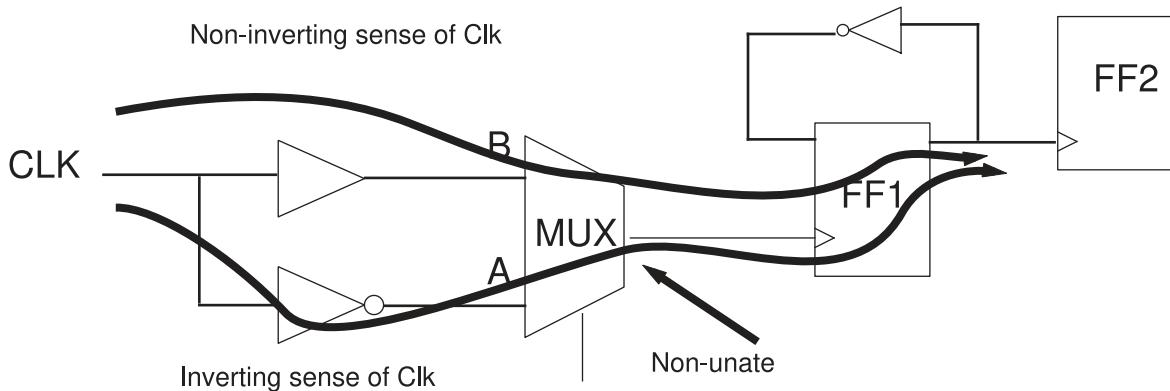
When you specify this option, the generated clock has the same period as the master clock. You can use the `-combinational` option only for those generated clocks that are frequency-divided (`-divide_by 1` option) or frequency-multiplied (`-multiply_by 1`) generated clocks of the master clock.

Generated Clock Based on a Non-Unate Master Clock

A clock signal is not unate if the clock sense is ambiguous as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate because there are non-unate arcs in the gate.

The tool considers the output of the multiplexer in [Figure 47](#) 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 MUX and the inverting path is through the inverter. By default, the tool propagates both positive unate clock paths and negative unate clock paths through the non-unate clock network.

Figure 47 Non-Unate Clock Paths



The clock network starting from the convergent point sees both the positive and negative sense of the original clock. The tool keeps track of clock properties of both senses separately. For example, when the positive sense clock has a waveform of {0 5}, the negative sense clock has a waveform of {5 10}.

In general path analysis, the particular path is constrained by the most constraining sense of the non-unate clock. When you need to generate clocks based on the non-unate master clock (for example, a divide-by-2 clock), new clocks generated by the `create_generated_clock` command are based on the original clock waveform. To generate a clock based on the expanded waveform (inverted clock of the divide-by-2 clock), use the `create_generated_clock -invert` command.

To specify the negative sense before waveform expansion (divide-by-2 clock of the negative unate master clock), you can include the `-preinvert` option with the `create_generated_clock` command. Note that the generated source has to be a pin in the fanout of the non-unate clock in order for the correct waveform expansion to occur. Consider [Figure 47](#). [Example 1](#) shows how to use the `-preinvert` and `-invert` options when a generated clock is driven by a non-unate master clock. [Figure 48](#) shows the resulting waveforms.

Example 1 Specifying Generated Clocks Based on a Non-Unate Master Clock

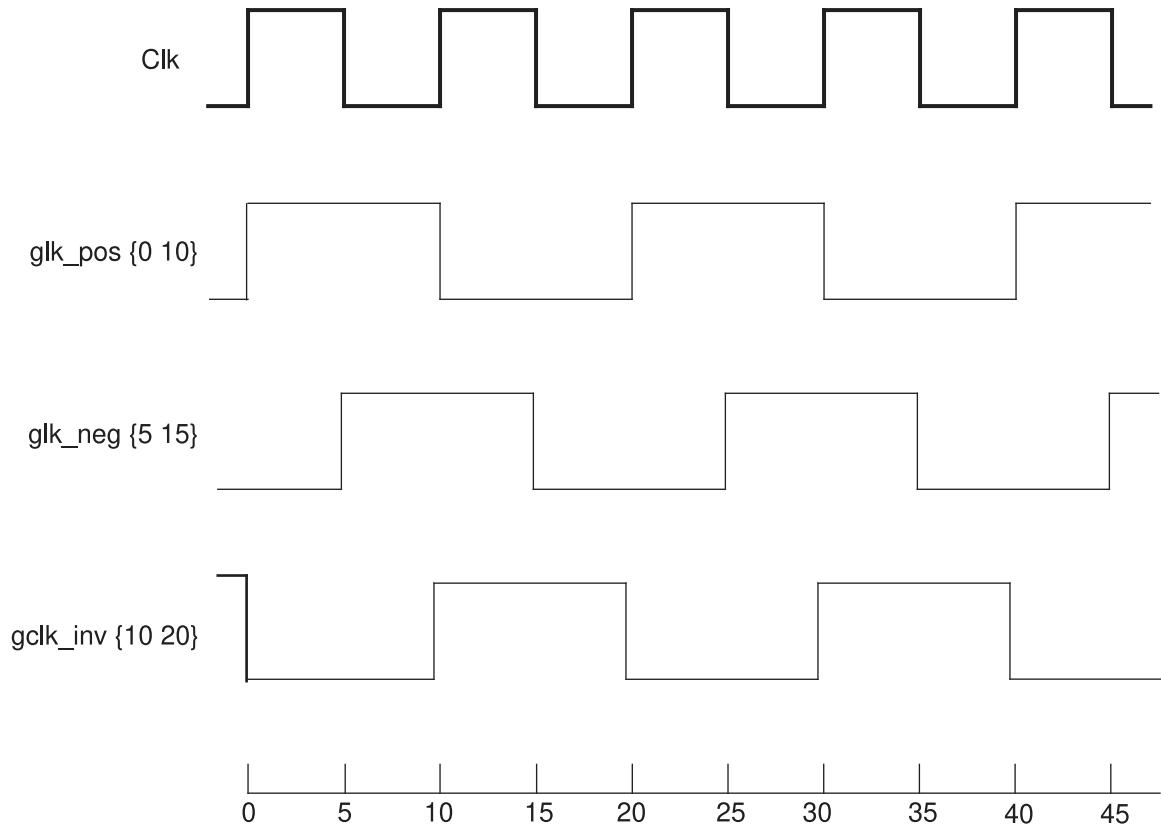
```
create_generated_clock -divide_by 2 -source \
[get_pins FF1/CLK] -name gclk_pos [get_pins FF1/Q]

create_generated_clock -divide_by 2 -source \
[get_pins FF1/CLK] -name gclk_neg \
[get_pins FF1/Q] -preinvert

create_generated_clock -divide_by 2 -source
```

```
[get_pins FF1/CLK] -name glk_inv
[get_pins FF1/Q] -invert
```

Figure 48 Waveforms Resulting From -preinvert and -invert



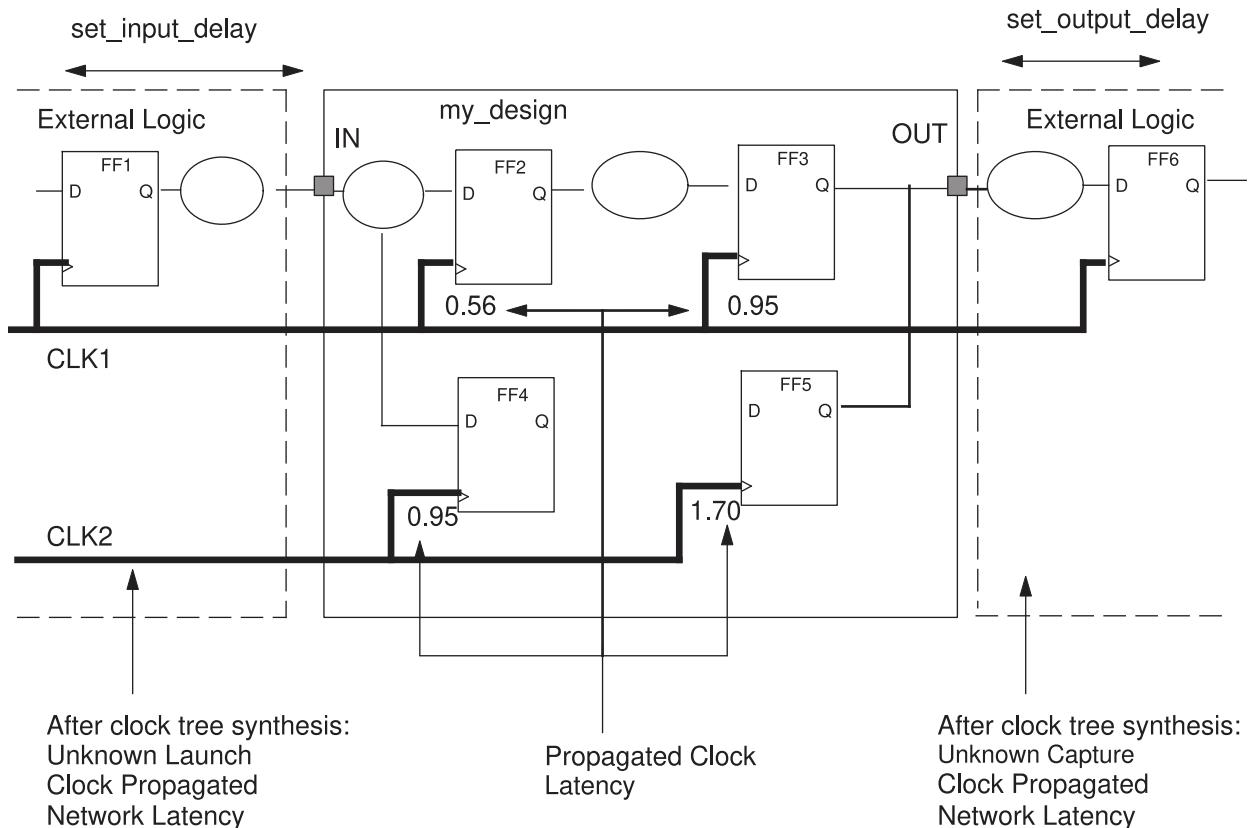
Estimated I/O Latency

You model the timing context of a design within a larger system by using input and output delays. You use the `set_input_delay` command to model the external delay at an input port; you use the `set_output_delay` command to model the external delay from an output port to a sequential device.

Prior to clock tree synthesis, when calculating the arrival time at an input port or departure time from an output port, the tool adds the clock source latency and clock network latency to the external datapath delays. However, after clock tree synthesis, when clocks are defined as propagated, the ideal clock network latency is replaced by propagated latency. Propagated clocks automatically calculate clock network latency within the design;

however, clock network latency external to your design is not known. This network latency at input ports and output ports is known as I/O latency.

Figure 49 I/O Latency



As illustrated in [Figure 49](#), for input paths, the tool uses the propagated network latency for capture clocks in the current design but does not know the propagated network latency for launch clocks. In a similar manner, for output paths, the tool uses the propagated network latency for launch clocks but does not know the propagated network latency for capture clocks.

To account for clock network latency in external delays, you can set the `estimate_io_latency` variable to true. When this variable is set to true, the tool assigns clock network delay to every input or output port. The tool assumes that clock tree synthesis balances network latency across different clocks. Based on this assumption, the tool estimates off-chip clock latency with reference to the synthesized clock tree in the current design.

If you used the `-network_latency_included` option with the `set_input_delay` command or `set_output_delay` command, the tool does not use an estimated I/O latency. Using the `-network_latency_included` option tells the tool that the clock network latency of the related clock is included in the input or output delay. The ideal way to model external network latency is to add the delay to the external delay and use the `-network_latency_included` option. Using the `estimate_io_latency` variable gives a reasonable representation if the clock network latency is balanced across different clocks.

Calculating I/O Latency for Input Paths

For each input delay, the tool locates the registers at the fanout of the input port. For setup analysis, the tool adds the smallest propagated clock network latency for any register in the fanout set to the arrival time. For hold analysis, it adds the largest network latency of any register in the fanout set to the arrival time. The latency that the tool selects can be associated with any clock and not necessarily the clock referenced by the input delay.

For example, in [Figure 49](#), for setup analysis, the tool uses an estimated I/O latency of 0.56, which is the smallest propagated clock network latency for the fanout registers of IN. Use the `report_timing` command to view the estimated I/O latency value. Here is a typical `report_timing` report showing the estimated input latency.

```

Startpoint: in (input port clocked by CLK1)
Endpoint: FF4 (rising edge-triggered flip-flop clocked by CLK2)
Path Group: CLK2
Path Type: max

      Point           Incr      Path
-----+
clock CLK1 (rise edge)        6.00      6.00
clock source latency          1.00      7.00
estimated IO latency          0.56    7.56
input external delay          9.00     16.56 f
in (in)                      0.00     16.56 f
U1/Z (IVA)                   0.35     16.90 r
U2/Z (IVA)                   0.34     17.24 f
U3/Z (IVA)                   0.46     17.70 r
FF2/D (FD1)                  0.00     17.70 r
data arrival time              17.70

clock CLK2 (rise edge)        8.00      8.00
clock network delay (ideal)   4.00     12.00
FF4/CP (FD1)                  0.00     12.00 r
library setup time            -0.80    11.20
data required time             11.20
data arrival time              11.20
-----+
data required time             11.20
data arrival time              -17.70

```

slack (VIOLATED)	-6.50
------------------	-------

Calculating I/O Latency for Output Paths

For each output delay, the tool locates the registers in the fanin of the output port. For setup analysis, the tool adds the largest propagated clock network latency for any register in the fanin set to the data required time. For hold analysis, it adds the smallest network latency of any register in the fanin set to the data required time. The latency that the tool selects can be associated with any clock and not necessarily the clock referenced by the output delay.

For example, in [Figure 49](#), for setup analysis, the tool uses an estimated I/O latency of 1.70, which is the largest propagated clock network latency for the fanin registers of OUT. Use the `report_timing` command to view the estimated I/O latency value. Here is a typical `report_timing` report showing the estimated output latency.

```
Startpoint: FF5 (rising edge-triggered flip-flop clocked by CLK2)
Endpoint: out1 (output port clocked by CLK1)
Path Group: CLK1
Path Type: max
```

Point	Incr	Path
clock CLK2 (rise edge)	8.00	8.00
clock network delay (ideal)	4.00	12.00
FF5/CP (FD1)	0.00	12.00 r
FF5/Q (FD1)	1.42	13.42 f
A1/Z (AN3)	0.85	14.27 f
out1 (out)	0.00	14.27 f
data arrival time		14.27
clock CLK1 (rise edge)	10.00	10.00
clock source latency	3.00	13.00
estimated IO latency	1.70	14.70
output external delay	-3.00	11.70
data required time		11.70
data required time		11.70
data arrival time		-14.27
slack (VIOLATED)		-2.57

Propagated Clocks

Setting a propagated clock is appropriate after layout, when the design has actual clock trees and annotated delay or parasitics. If you do not set a propagated clock, the tool assumes ideal clocking. Ideal clocking means that clock networks have a specified

latency (from the `set_clock_latency` command) or zero latency by default. You specify propagated clock latency after layout; you specify ideal clock latency before layout to provide an estimate of the clock tree.

Thus, after layout, you do not need to estimate the clock tree uncertainty or network latency anymore, although you still need to specify source latency. Additionally, you no longer need to specify the clock transition.

For example, as part of the prelayout process, you might specify the following commands:

```
prompt> set_clock_latency -max 0.8 clk
prompt> set_clock_latency -min 0.3 clk
```

Alternatively, you might use back-annotation to approximate the expected delay, in which case you would back-annotate Standard Delay Format (SDF) values. For example, as part of the pre-layout process, you might specify the following commands:

```
prompt> set_propagated_clock clk
prompt> read_sdf -max_type sdf_max -min_type sdf_min design.sdf
```

The `set_propagated_clock` command specifies propagated clock latency for the listed clocks, ports, pins, and cells. For information about the `read_sdf` command, see [Back-Annotationing Timing Information From an SDF File](#). See the man pages for the complete description of the `set_propagated_clock` and `read_sdf` commands.

Registers clocked by a propagated clock have their edge times skewed by the path delay from the clock source to the register clock pin.

To set a propagated clock,

1. Remove clock transitions, which you set previously with the `set_clock_transition` command. Enter

```
prompt> remove_clock_transition CLKA
```

2. Remove network latency, which you set previously with the `set_clock_latency` command. (Propagated clocks automatically calculate network latency; however, you still need to specify source latency.) Enter

```
prompt> remove_clock_latency CLKA
```

3. Modify the clock uncertainty to model clock jitter but not clock skew. (Propagated clocks automatically calculate skew.) Enter

```
prompt> set_clock_uncertainty 0.1 CLKA
```

4. Set propagated clocks. Enter

```
prompt> set_propagated_clock CLKA
```

Table 4 summarizes the clock-related commands you would use before and after layout.

Table 4 Clock-Related Commands to Use Before and After Layout

Pre-layout commands	Post-layout commands
<pre>create_clock -period 30 -name ClkA Clk set_clock_uncertainty 0.5 ClkA set_clock_transition 0.25 ClkA set_clock_latency -source 4 ClkA set_clock_latency 2 ClkA</pre>	<pre>create_clock -period 30 -name ClkA Clk set_clock_uncertainty 0.1 ClkA remove_clock_transition ClkA set_clock_latency -source 4 ClkA remove_clock_latency ClkA set_propagated_clock ClkA</pre>

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

To remove propagated clocks that you set with the `set_propagated_clock` command, use the `remove_propagated_clock` command.

3

Timing Paths

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 an overview of timing path principles, see [Introduction to Synthesis Timing](#), which includes basic information about path startpoints and endpoints, delay calculation, setup and hold constraints, and time borrowing.

This chapter provides information about timing paths in the following sections:

- [Path Groups](#)
 - [Path Specification Methods](#)
 - [Default Path Delay Constraints](#)
 - [Timing Exceptions](#)
 - [Preset and Clear Timing](#)
 - [Data-to-Data Checks](#)
-

Path Groups

The synthesis tool organizes timing paths into groups. The grouping of paths affects the optimization of the design and the generation of timing reports. By default, there is one path group for each clock in the design. All timing paths clocked by a given clock at the path endpoint belong to that clock's path group.

The tool also creates a path group called ****default**** (including the asterisks). The ****default**** group contains any paths that cannot be categorized by the clock used at the path endpoint, such as feedthrough and asynchronous paths. Paths can be moved to the ****default**** group by using the `remove_path_group` command or the `-default` option of the `group_path` command.

You can specify clock-gating checks by using the `set_clock_gating_check` command. A clock-gating path ends on a combinational element used for gating the clock. By default, clock-gating paths are contained in the path group of the clock being gated. However, you can have the synthesis tool create a separate ****clock_gating_default**** group to contain the clock-gating paths, for both optimization and reporting purposes. To do so, set the variable `timing_separate_clock_gating_group` to true. Using this option matches

the default behavior of PrimeTime with respect to the grouping of clock-gating paths. PrimeTime also creates a path group called `**async_default**`, which is not supported by the synthesis tools.

User Grouping of Paths

All timing paths within a path group are optimized for timing together, starting with the critical path. Paths within a group are optimized and reported separately from other groups. You can use the `group_path` command to assign paths to existing groups or new groups that you create, which lets you control the focus of optimization effort and the reporting of paths. For details, see [Path Groups](#).

To get information about the current set of path groups, use the `report_path_group` command. To remove a path group, use the `remove_path_group` command. Paths belonging to a removed group are implicitly assigned to the default path group.

Weight or Cost Function

You can optionally assign a weight or cost function to each path group so that the synthesis tool applies more effort to optimizing the targeted group. The default weighting value for each path group is 1. You specify a different weighting value by using the `-weight` option of the `group_path` command. The higher the weighting, the higher the effort applied to the group. For example, higher weighting for a group means that the synthesis tool will attempt to reduce the size of a timing violation for a path in that group at the expense of slack in a lower-weighted group. For details, see [Path Groups](#).

Critical Range

A synthesis tool continuously works to improve the timing of paths leading to the most critical endpoint. The most critical endpoint changes as the paths are improved. Sometimes it is not possible for the tool to optimize all paths to zero slack. By default, when the synthesis tool determines that it cannot further improve the most critical endpoint, it stops delay optimization for that path group and proceeds with the next path group, leaving some paths with timing violations.

The default behavior can be changed by setting the critical range of one or more path groups. The critical range specifies the range of timing slacks that the synthesis tool attempts to correct beyond the first critical path that cannot be optimized further. The default critical range is zero, which means that the synthesis tool does not attempt to optimize any more paths in a path group after it stops working on the most critical path.

With a nonzero critical range, the synthesis tool continues working to optimize near-critical paths that have a timing slack within the specified range of the current critical path, up to one additional path per endpoint. It continues to work on path endpoints within the path

group as long as their timing slacks are negative and within the specified range of the first critical path that could not be optimized further.

The larger the critical range, the larger the number of paths that the tool attempts to fix, at the cost of more runtime. If you set the critical value to a large number, the tool attempts to optimize all paths that have timing violations, subject to the limit of one path per endpoint. If you want to optimize multiple paths to the same endpoint, you must group those paths into separate path groups with the `group_path` command.

You can set the critical ranges for individual path groups by using the `-critical_range` option of the `group_path` command. To set a general critical range for all path groups, use the `set_critical_range` command. A specific critical range setting made with the `group_path -critical_range` command has higher priority than a general setting made with the `set_critical_range` command.

Reporting Path Groups

The `report_path_group` command displays information about the path groups in the current design, including the name, weight, critical range, and endpoint clock for each group. For example,

```
prompt> report_path_group
...
          Critical
Group Name    Weight    Range
-----
**default**    1.00     0.00
PCI_CLK        1.00     0.00
SDRAM_CLK      1.00     0.00
SD_DDR_CLK     1.00     0.00
SYS_CLK         1.00     0.00

Path Group PCI_CLK:
  -to      PCI_CLK

Path Group SDRAM_CLK:
  -to      SDRAM_CLK

Path Group SD_DDR_CLK:
  -to      SD_DDR_CLK

Path Group SYS_CLK:
  -to      SYS_CLK
```

1

Path Specification Methods

The `report_timing` command, the `group_path` command, and the timing exception commands (such as `set_false_path`) operate on one or more timing paths that you specify in the command. The most straightforward way to specify paths is to provide the startpoint and endpoint. The startpoint is known as the “from” object and the endpoint is called the “to” object. The following examples demonstrate the command syntax:

```
prompt> report_timing -from PORTA -to PORTZ
prompt> set_false_path -from FFB1/CP -to FFB2/D
prompt> set_max_delay -from REGA -to REGB 12
prompt> set_multicycle_path -setup 2 -from FF4 -to FF5
```

The “from” startpoint object can be a clock, a primary input port, the 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. For the timing exception commands, you can specify a cell, and the tool applies the exception to one path startpoint on that cell.

The “to” endpoint object can be a clock, a primary output port, a sequential cell, the 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. For the timing exception commands, you can specify a cell, and the tool applies the exception to one path endpoint on that cell.

There can be many paths originating from the “from” object and ending at the “to” object. All such paths are selected for action by the command. The `report_timing` command analyzes all the specified path. By default, it reports only the single path having the worst slack among those analyzed. In a command that sets a timing exception, all the selected paths are affected by the command.

You can restrict the command to only to rising edges or only falling edges by using the command arguments `-rise_from`, `-fall_from`, `-rise_to`, and so on. For example, the `-rise_from` option applies only to rising edges at the startpoint object.

Through Arguments

To restrict the scope of paths selected for action, you can use the `-through` option, as in the following example:

```
prompt> report_timing -from PORTA -through FFB1 -to PORTZ
```

This selects the paths that start at PORTA, pass through FFB1, and end at PORTZ.

You can use the `-through` option alone, without `-from` or `-to`, to specify all paths that pass through one or more specific pins, ports, or cells. However, this method is more

computationally intensive, especially in larger designs. Whenever possible, it is better to use `-from` or `-to` (or both) to specify the paths.

You can use multiple `-through`, `-rise_through`, and `-fall_through` arguments in a single command to specify a group of paths. For example,

```
prompt> 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.

Multiple objects in braces mean any one object in the list satisfies the “through” condition. For example,

```
prompt> 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.

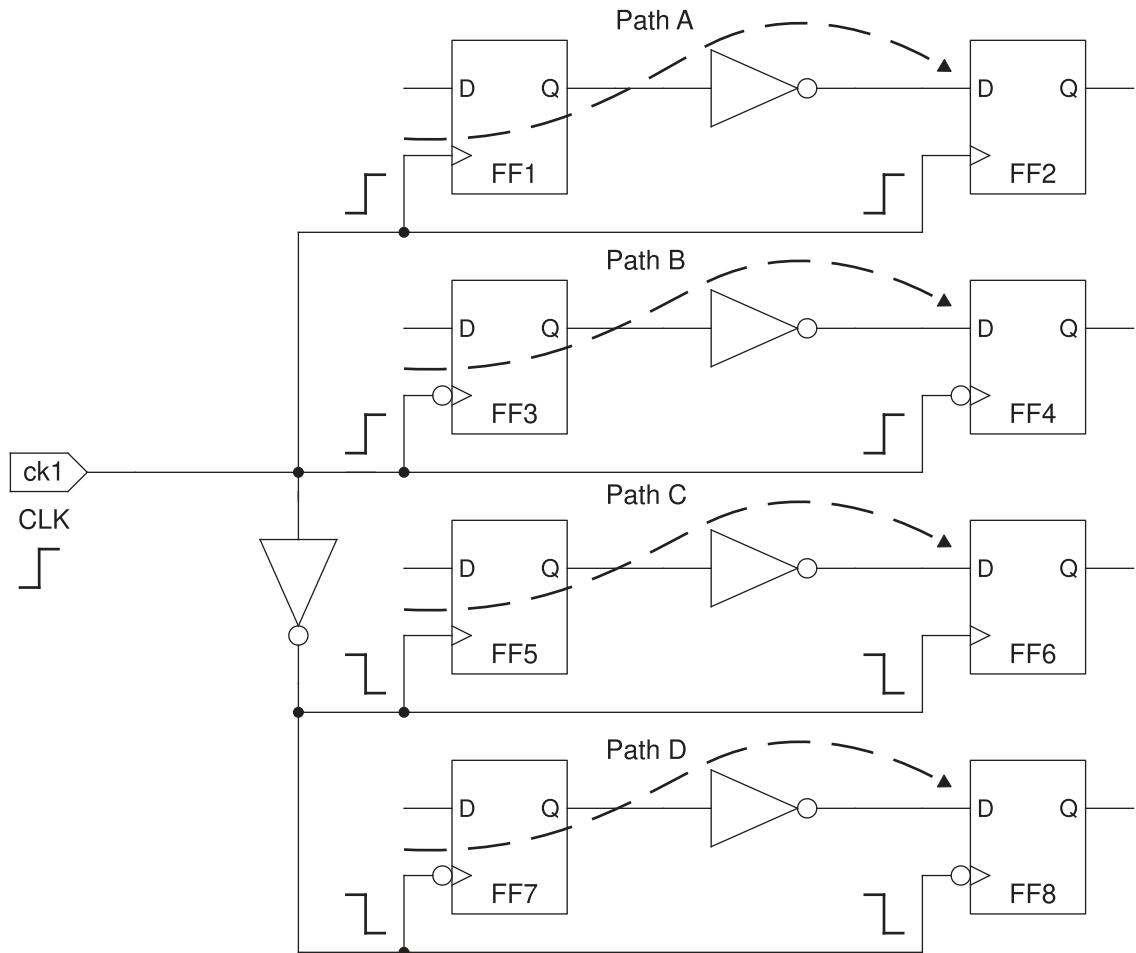
Rise/Fall From/To Clock

You can restrict the command to only to rising edges or only falling edges by using the command arguments `-rise_from`, `-fall_from`, `-rise_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 50](#). FF1, FF2, FF5, and FF6 are rising-edge-triggered flip-flops; and FF3, FF4, FF7, and FF8 are falling-edge-triggered flip-flops. Flip-flops FF1 through FF4 are clocked directly, whereas flip-flops FF5 through FF8 are clocked by an inverted clock signal.

Figure 50 Circuit for Path Specification Examples 1 Through 3



Example 1

```
prompt> set_false_path -to [get_clocks CLK]
```

In Figure 50, all paths clocked by CLK at the path endpoint (all four paths) are declared to be false.

Example 2

```
prompt> set_false_path -rise_from [get_clocks CLK]
```

In Figure 50, 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

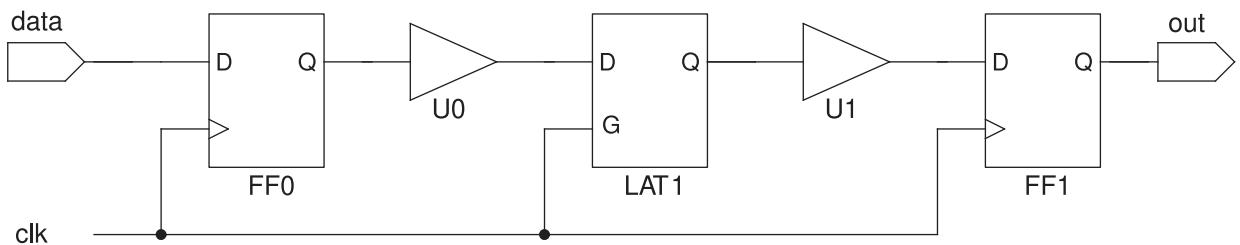
```
prompt> set_false_path -fall_to [get_clocks CLK]
```

In Figure 50, 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 51. The two flip-flops latch data on the rising edge of the clock. The level-sensitive latch opens on the rising edge and closes on the falling edge of the clock.

Figure 51 Circuit for Path Specification Example 4



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`, from `FF0` to `LAT1`, from `LAT1/D` to `FF1`, from `LAT1/G` to `FF1`, and from `FF1` to `out`:

```
prompt> report_timing -rise_from [get_clock clk] -nworst 100
```

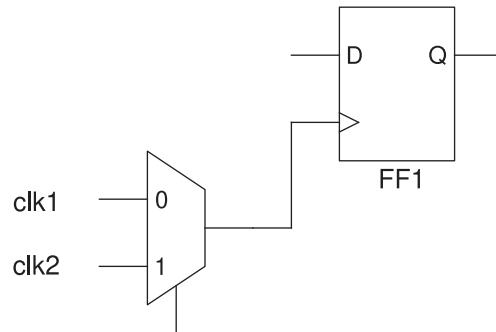
Using the `-fall_to` option in the command instead reports the path with data captured by a falling edge of the clock, which is from `FF0` to `LAT1`:

```
prompt> report_timing -fall_to [get_clock clk] -nworst 100
```

Example 5

In the circuit shown in Figure 52, multiple clocks reach the flip-flop.

Figure 52 Circuit for Path Specification 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. Use the `-through` and `-to` options as follows:

```
prompt> set_false_path -through FF1/D -to [get_clocks clk1]
```

To set a false path ending at the D pin of the flip-flop, but only for paths that capture data on the rising edge of clk1, use the following command:

```
prompt> set_false_path -through FF1/D -rise_to [get_clocks clk1]
```

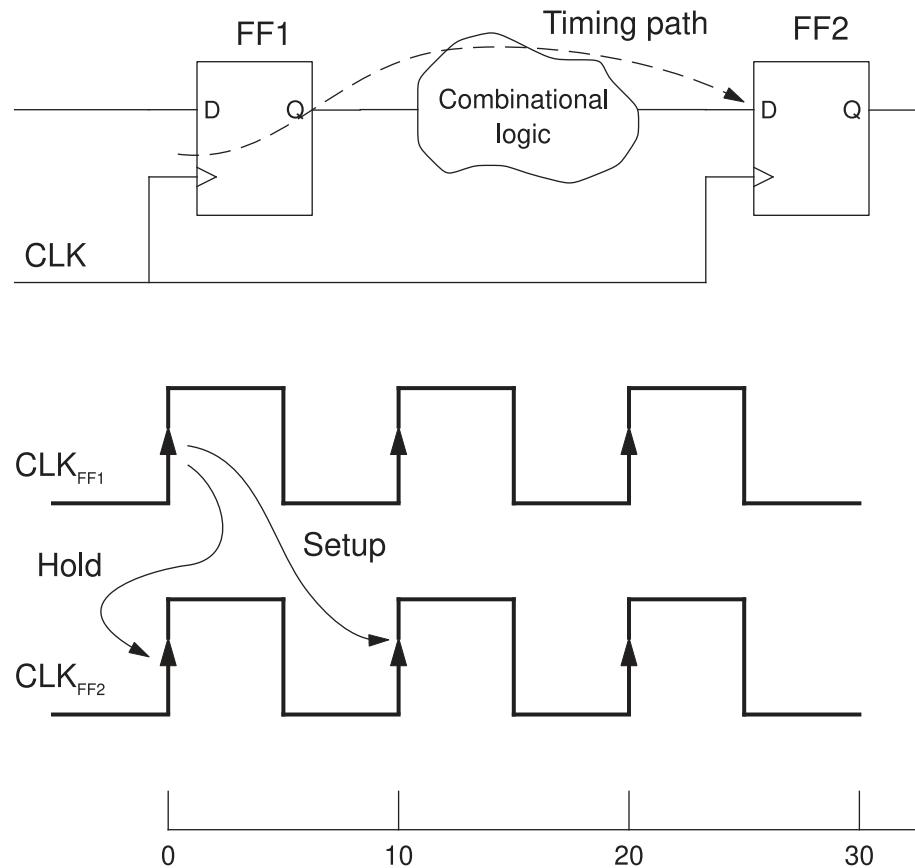
Default Path Delay Constraints

In the absence of timing exceptions, the tool checks for data arrival at the next capture clock edge following the launch event. For a single-clock design, launch occurs on a clock edge and capture occurs at the next clock edge. When the launch and capture clocks are different, the tool considers the nearest possible capture clock edge that can occur after the launch clock edge.

Path Delay for Flip-Flops Using a Single Clock

[Figure 53](#) shows how the tool determines the setup and hold constraints for a path that begins and ends on rising-edge-triggered flip-flops. The two flip-flops are triggered by the same clock. The clock period is 10 ns.

Figure 53 Single-Cycle Setup and Hold for Flip-Flops



The tool 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, the tool 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.

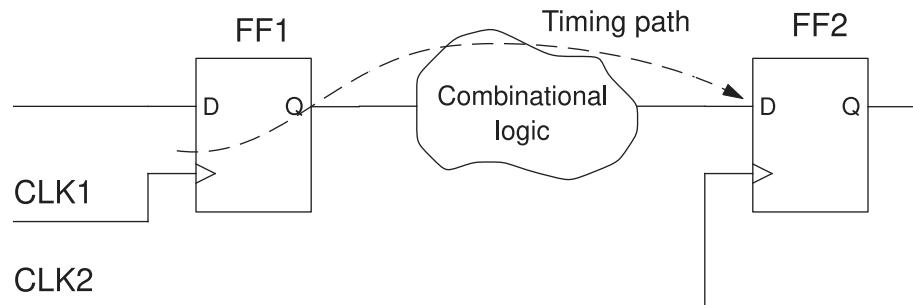
The setup and hold requirements determined by the tool for sequential elements take into account all relevant parameters such as the delay of the combinational logic elements in the path, the setup and hold requirements of the flip-flop elements as defined in the logic library, and the net delays between the flip-flops.

Path Delay for Flip-Flops Using Different Clocks

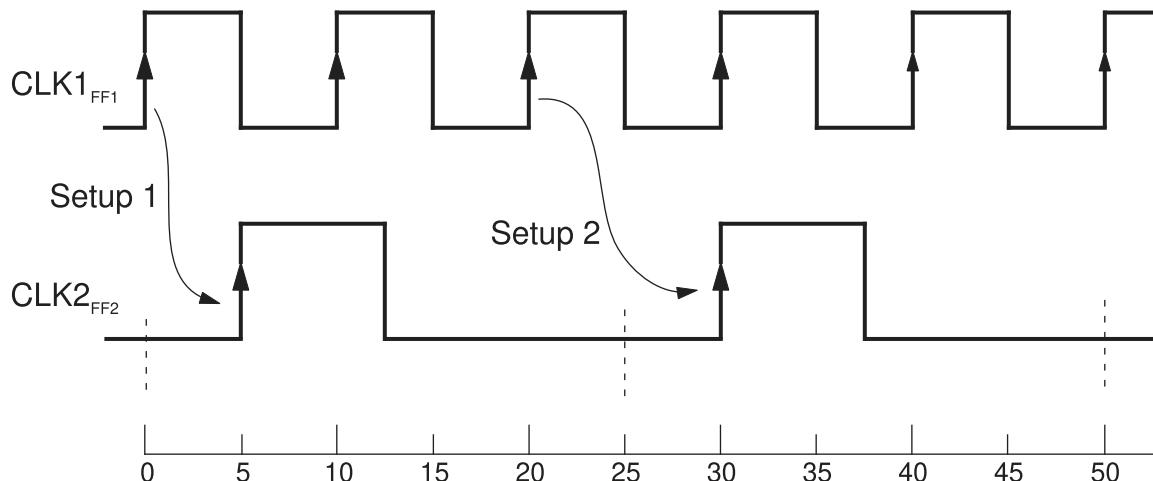
The method of calculating path delays is more complicated when the launch and capture flip-flops belong to different clock domains.

Consider the circuit shown in [Figure 54](#). The flip-flops at the beginning and end of the timing path are clocked by CLK1 and CLK2. The two clocks are declared by the `create_clock` commands shown in the figure.

Figure 54 Setup Constraints for Flip-Flops With Different Clocks



```
create_clock -period 10 -waveform {0 5} CLK1
create_clock -period 25 -waveform {5 12.5} CLK2
```



By default, the tool 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 are asynchronous), then you need to declare this fact by using any of several methods. Otherwise, the tool spends time checking constraints and reporting violations that do not exist in the actual circuit.

Setup Analysis

The tool 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 edge at the destination flip-flop, the tool assumes that the corresponding launch edge is the nearest source clock edge occurring before the capture edge.

In [Figure 54](#), 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

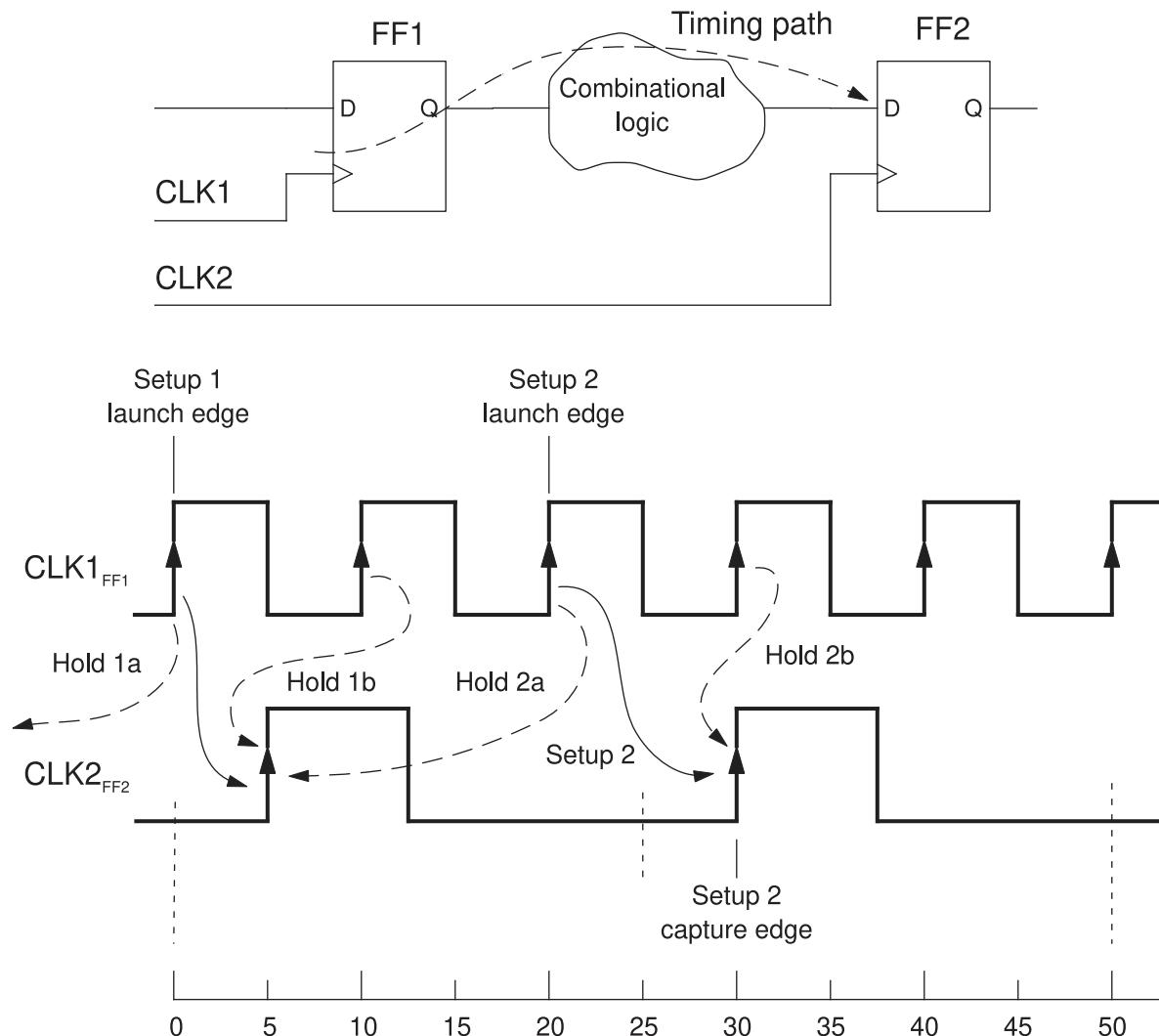
The hold relationships checked by the tool are based on the clock edges adjacent to those used to determine the setup relationships. To determine the most restrictive hold relationship, the tool considers all valid setup relationships, including both Setup 1 and Setup 2 in [Figure 54](#).

For each setup relationship, the tool performs two different hold checks:

- The data launched by the setup launch edge must not be captured by the *previous* capture edge.
- The data launched by the *next* launch edge must not be captured by the setup capture edge.

[Figure 55](#) shows the hold checks performed for the current example. First consider the setup relationship labeled Setup 2. The tool confirms that the data launched by the setup launch edge is not captured by the previous capture edge; this relationship is labeled Hold 2a. It also confirms that the data launched by the next clock edge at the source is not captured by the setup capture edge; this relationship is labeled Hold 2b.

Figure 55 Hold Constraints for Flip-Flops With Different Clocks



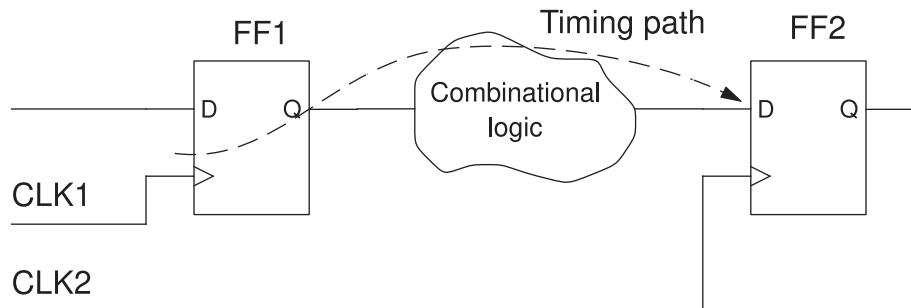
The tool similarly checks the hold relationships relative to the clock edges of Setup 1, as indicated in the figure. The most restrictive hold check is the one where the capture edge occurs latest relative to the launch edge, which is Hold 2b in this example. Therefore, Hold 2b determines the minimum allowed delay for this path, 0 ns.

Single-Cycle Path Analysis Examples

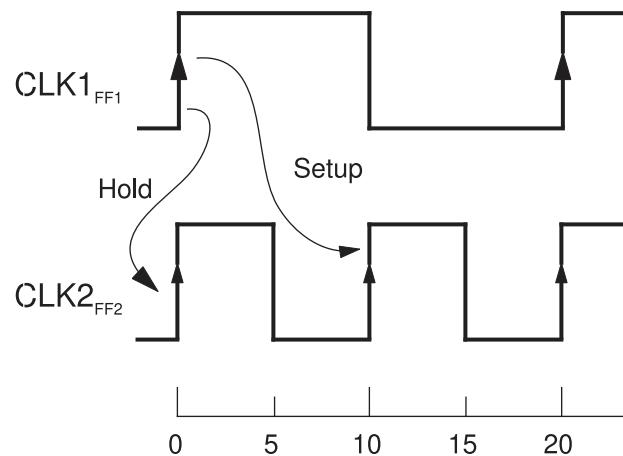
The following examples further illustrate how the tool calculates the delay requirements for edge-triggered flip-flops in the absence of timing exceptions.

In [Figure 56](#), 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 56 Delay Requirements With 20-ns and 10-ns Clocks



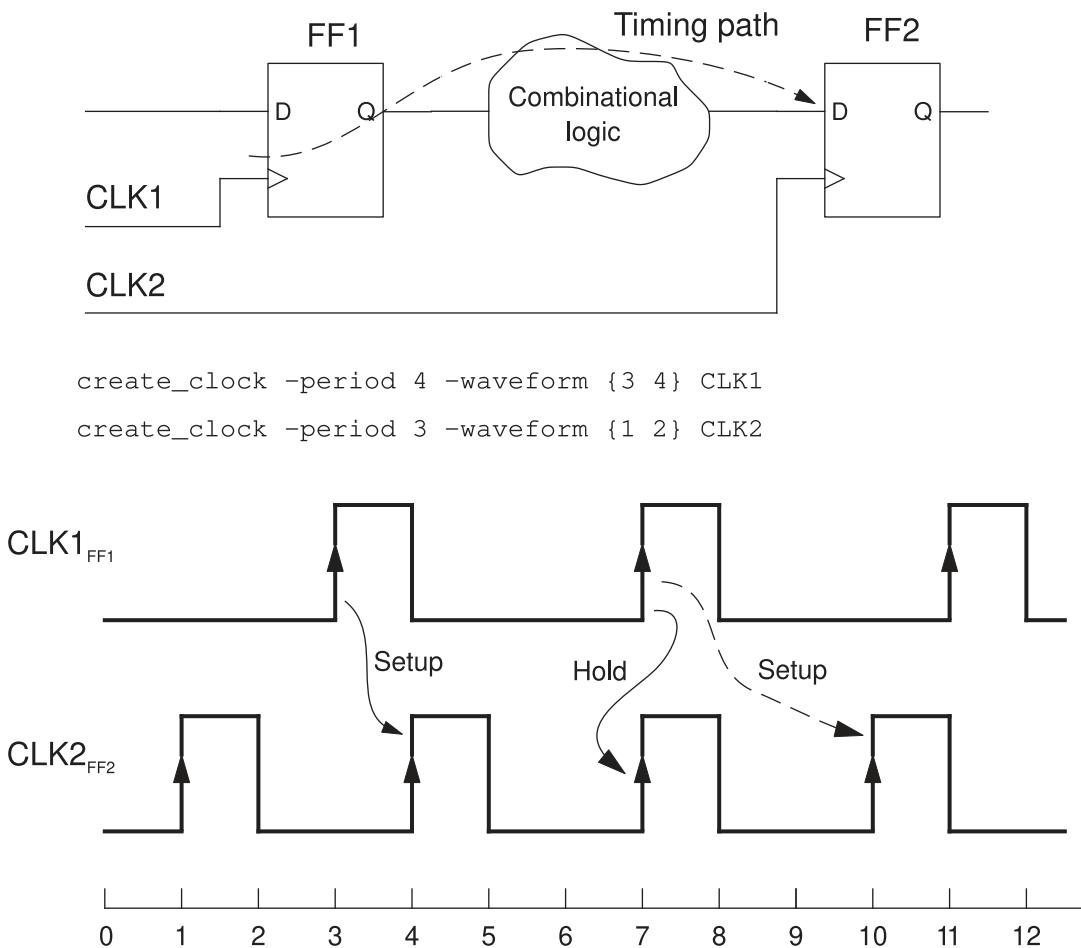
```
create_clock -period 20 -waveform {0 10} CLK1
create_clock -period 10 -waveform {0 5} CLK2
```



In [Figure 57](#), 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 57 Delay Requirements With 4-ns and 3-ns Clocks



Timing Exceptions

The Design Compiler and IC Compiler tools support the following types of timing exceptions:

- False paths. Use the `set_false_path` command to specify a logic path that exists in the design but should not be analyzed. Setting a false path removes the timing constraints on the path.
- Minimum and maximum path delay values. Use the `set_max_delay` and `set_min_delay` commands to override the default setup and hold constraints with specific maximum and minimum delay values.
- Multicycle paths. Use the `set_multicycle_path` command to specify the number of clock cycles required to propagate data from the start to the end of the path.
- Path timing margins. Use the `set_path_margin` command to tighten or loosen the timing requirement for specific paths by a given amount.

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 in the design.

In a timing exception command, be sure to specify valid startpoints and endpoints. A startpoint must be a register clock pin or an input port. An endpoint must be 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_timing_requirements` command. To restore the default timing constraints on a path, use the `reset_path` command.

False Path Exceptions

A false path is a logic path in the design 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:

```
prompt> set_false_path -from [get_pins FFB1/CP] \
           -to [get_pins FFB2/D]
```

Setting a false path is a point-to-point timing exception that removes all timing constraints from a path, which prevents errors from being reported but does not stop delay calculation. This is different from using the `set_disable_timing` command, which disables timing analysis entirely for a specified pin, cell, or port. 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:

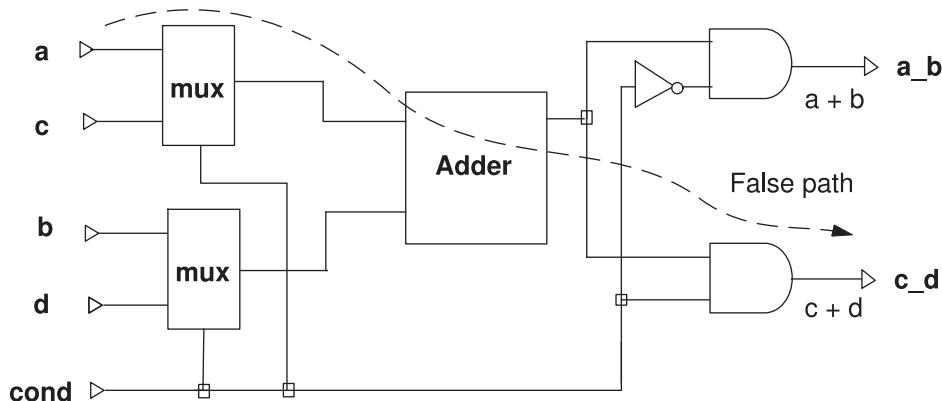
```
prompt> set_false_path -from [get_clocks ck1] \
           -to [get_clocks ck2]
prompt> set_false_path -from [get_clocks ck2] \
           -to [get_clocks ck1]
```

For efficiency, be sure to specify each clock by its clock name, not by the pin name; use the `get_clocks` command rather than `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.

In [Figure 58](#), the adder is shared between the multiplexed inputs a, b, c, and d. The logic of this design prevents the path from port “a” to port “c_d” from being enabled, so it is a false path.

Figure 58 False Path Example



To declare this as a false path and thereby prevent the tool from analyzing this path timing, you could use:

```
prompt> set_false_path -from [get_ports a] -to [get_ports c_d]
```

or, more generally for this example,

```
prompt> set_false_path -from [get_ports {a b}] -to [get_ports c_d]
prompt> set_false_path -from [get_ports {c d}] -to [get_ports a_b]
```

The tool has the ability to detect the presence of certain types of false paths in the design when you use case analysis. To detect false paths, use the `report_timing` command with the `-justify` option.

Maximum and Minimum Path Delay Exceptions

By default, the tool calculates the maximum and minimum allowable path delays by considering the launch and capture 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 the following command:

```
prompt> set_max_delay 12.0 \
           -from [get_cells REGA] -to [get_cells REGB]
```

With this timing exception, the tool ignores the clock relationships. A path delay between these registers that exceeds 12.0 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 the following command:

```
prompt> set_min_delay 2.0 \
           -from [get_cells REGA] -to [get_cells REGB]
```

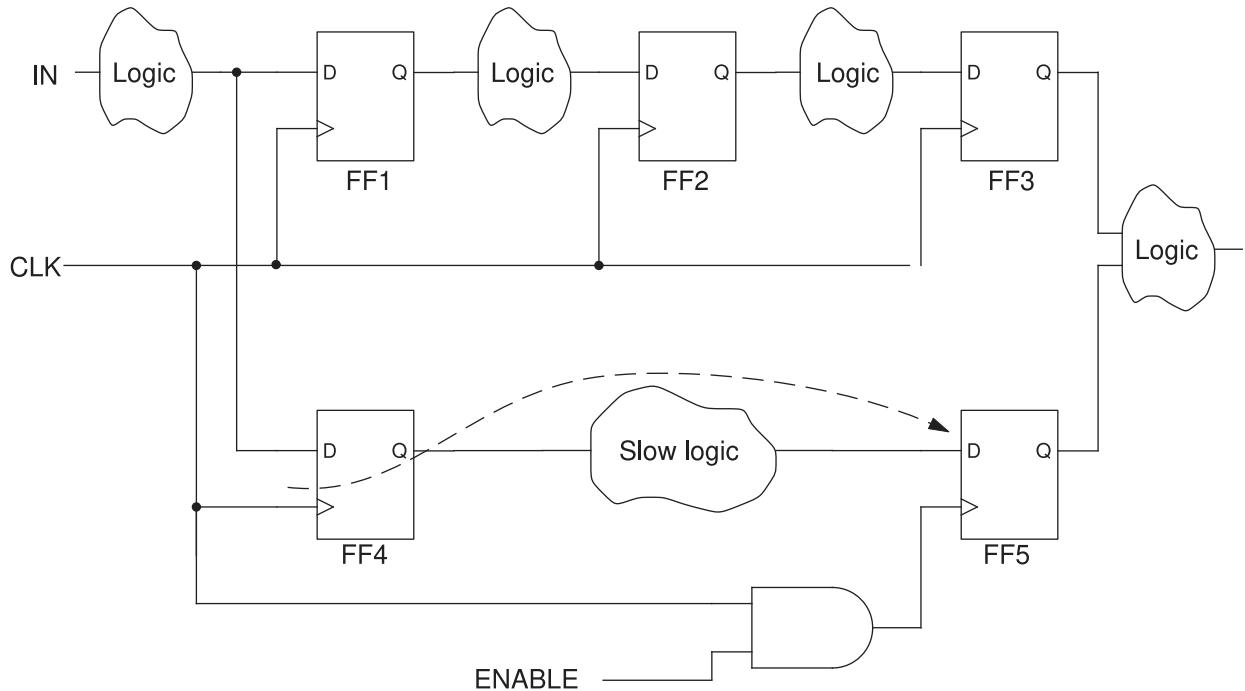
Again, the tool ignores the clock relationships. A path delay between these registers that is less than 2 time units plus the hold requirement of the endpoint register is reported as a timing violation.

Multicycle Path Exceptions

The `set_multicycle_path` command specifies the number of clock cycles required to propagate data from the start of a path to the end of the path. The tool calculates the setup or hold constraint according to the specified number of cycles.

For example, consider the circuit shown in [Figure 59](#). The path from FF4 to FF5 is designed to take two clock cycles rather than one. However, by default, the tool assumes single-cycle timing for all paths. Therefore, you need to specify a timing exception for this path.

Figure 59 Multicycle Path Example



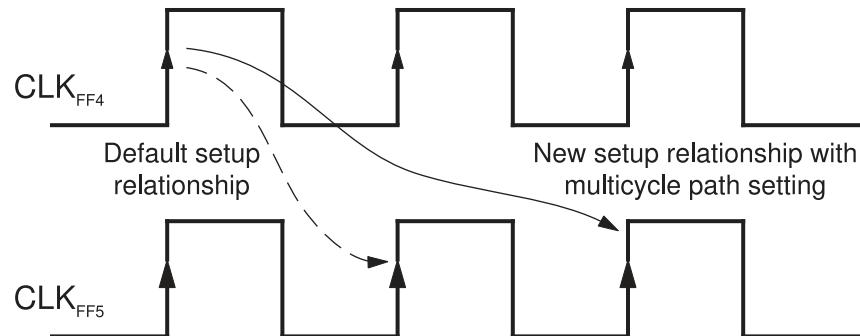
One timing exception method is to specify an explicit maximum delay value with the `set_max_delay` command. However, you might want to use `set_multicycle_path` instead because the maximum delay value is automatically adjusted when you change the clock period.

To set the multicycle path for the design shown in Figure 59, you can use the following command:

```
prompt> 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 60. The second capture edge following the launch edge becomes the applicable edge for the end of the path.

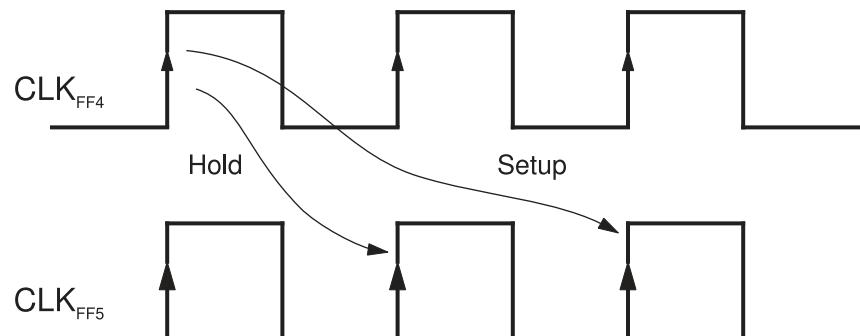
Figure 60 Multicycle Path Setup



```
prompt> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]
```

Changing the setup relationship implicitly changes the hold relationship as well because all hold relationships are based on the valid setup relationships. The tool verifies that the data launched by the setup launch edge is not captured by the previous capture edge. The new hold relationship is shown in Figure 61.

Figure 61 Multicycle Path Hold Based on New Setup



```
prompt> set_multicycle_path -setup 2 \
           -from [get_cells FF4] -to [get_cells FF5]
```

The hold relationship shown in [Figure 61](#) 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 a multicycle hold timing exception.

Although you could use the `set_min_delay` command to specify a particular hold time, it is better to use another `set_multicycle_path` command to move the capture edge for the hold relationship backward by one clock cycle. For example,

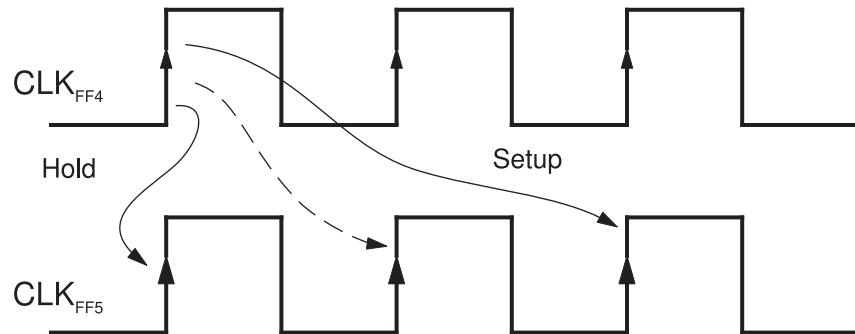
```
prompt> set_multicycle_path -setup 2 \
    -from [get_cells FF4] -to [get_cells FF5]
prompt> set_min_delay 0 -from [get_cells FF4] -to [get_cells FF5]
```

or preferably:

```
prompt> set_multicycle_path -setup 2 \
    -from [get_cells FF4] -to [get_cells FF5]
prompt> set_multicycle_path -hold 1 \
    -from [get_cells FF4] -to [get_cells FF5]
```

[Figure 62](#) 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 62 Multicycle Path Hold Set Correctly



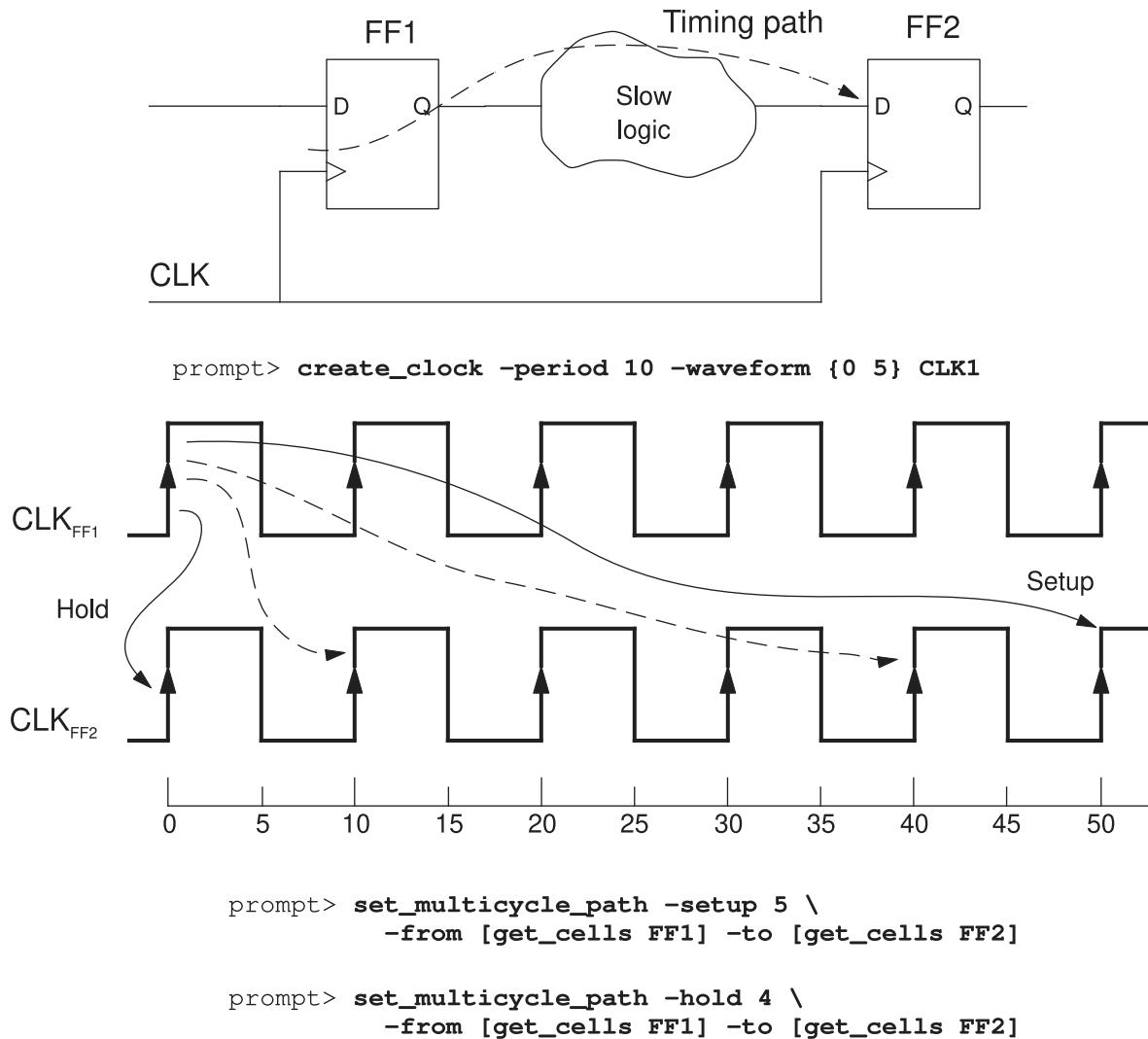
```
prompt> set_multicycle_path -setup 2 \
    -from [get_cells FF4] -to [get_cells FF5]
```

```
prompt> set_multicycle_path -hold 1 \
    -from [get_cells FF4] -to [get_cells FF5]
```

The tool interprets the `-setup` and `-hold` values in the `set_multicycle_path` command differently. The integer value for the `-setup` argument specifies the number of clock cycles for the multicycle path. In the absence of a timing exception, the default is 1. The integer value for the `-hold` argument specifies the number of clock cycles to move the capture edge backward with respect to the default position, relative to the valid setup relationship; the default setting is 0.

The example shown in [Figure 63](#) 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 63 Multicycle Path Taking Five Clock Cycles



With the `set_multicycle_path -setup 5` command, 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, the tool 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 62](#), hold cycles = $2 - 1 - 1 = 0$. For [Figure 63](#), hold cycles = $5 - 1 - 4 = 0$.

You can optionally specify that the multicycle path exception apply only to rising edges or only with falling edges at the path endpoint. If the startpoint and endpoint are clocked by different clocks, you can specify which of the two clocks is considered for adjusting the number of clock cycles for the path.

Path Timing Margin Exceptions

You can make any timing check more restrictive or less restrictive for specified paths by a given amount of time. To do so, use the `set_path_margin` command. For example,

```
prompt> set_path_margin 1.2 -from [get_cells FF4] -to [get_cells FF5]
```

This command makes all setup checks from cell FF4 to cell FF5 more restrictive by 1.2 time units. Specifying a positive margin results in a more restrictive or tighter check, resulting in a smaller or more negative reported slack. Conversely, specifying a negative margin results in a less restrictive or looser check, resulting in a larger or less negative reported slack.

In a report generated by the `report_timing` command, the accounting for the margin appears as an adjustment in the data required time calculation. For example,

```
prompt> set_path_margin 0.5 -to I_BL_1/shift_reg_0_30/_D
...
prompt> report_timing -to I_BL_1/shift_reg_0_30/_D
...
Path Group: SYS_CLK
  Path Type: max

  Point                                Incr      Path
  -----
clock SYS_CLK (rise edge)                0.00     0.00
clock network delay (propagated)        0.57     0.57
I_BL_1/shift_reg_1_29/_CP (sdcrq1)      0.00     0.57 r
...
I_BL_1/shift_reg_0_30/_D (sdcrq1)       0.00 *    7.85 r
data arrival time                      7.85

clock SYS_CLK (rise edge)                8.00     8.00
clock network delay (propagated)        0.55     8.55
clock uncertainty                       -0.10    8.45
path margin                           -0.50   7.95
I_BLENDER_1/mega_shift_reg_0_30/_CP (sdcrq1) 0.00     7.95 r
library setup time                      -0.18    7.77
```

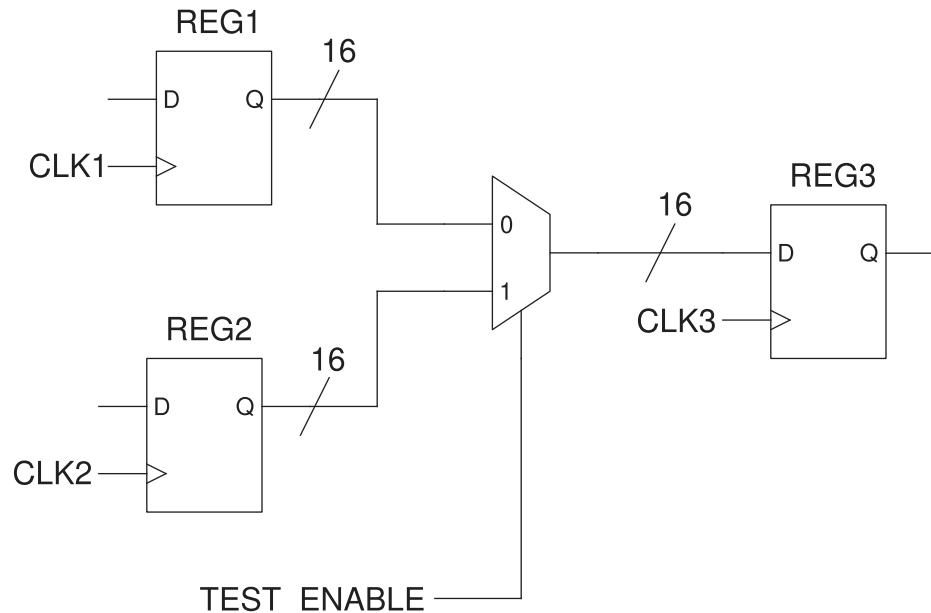
data required time	7.77
data required time	7.77
data arrival time	-7.85
slack (VIOLATED)	-0.08

The specified margin applies to a timing check in addition to any minimum-delay, maximum-delay, and multicycle path exceptions set on the same paths. A false path exception applied to the same path has priority over a path timing margin exception, causing the margin setting to be ignored. For the resolution of other possible combinations of conflicting exceptions, see the man page for the `set_path_margin` command.

Specifying Exceptions Efficiently

In many cases, you can specify the exception paths many different ways. Choosing an efficient method can reduce the runtime. For example, consider the circuit shown in [Figure 64](#). 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 64 Multiplexed Register Paths



To prevent analysis of timing from REG2 to REG3, you can use any of the following methods:

- Use case analysis to consider the case when the test enable signal is 0. See [Case Analysis](#).
- Set an exclusive relationship between the CLK1 and CLK2 clock domains. See [Mode Analysis](#).
- Declare the paths between clock domains CLK2 and CLK3 to be false.

```
prompt> set_false_path \
           -from [get_clocks CLK2] -to [get_clocks CLK3]
```

This method is an efficient way to specify the false paths because the tool 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.

```
prompt> set_false_path -from [get_pins REG2[0]/CP] \
           -to [get_pins REG3[0]/D]
prompt> set_false_path -from [get_pins REG2[1]/CP] \
           -to [get_pins REG3[1]/D]
prompt> ...
```

This method is less efficient because the tool must keep track of timing exceptions for 16 different paths.

- Declare all paths from REG2 to REG3 to be false.

```
prompt> set_false_path -from [get_pins REG2[*]/CP] \
           -to [get_pins REG3[*]/D]
```

This method is even less efficient because the tool 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.

```
prompt> set_false_path -from [get_pins REG2[*]/CP]
```

This method is similar to the previous one. The tool 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 makes 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` option or wildcards.

Exception Order of Precedence

If different timing exception commands are in conflict for a particular path, the exception used 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.

The tool applies the exception precedence rules independently on each path, not each command. For example, suppose that you use the following commands:

```
prompt> set_max_delay -from A 5.1
prompt> 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
- `set_path_margin` and `set_min_delay`, `set_max_delay`, or `set_multicycle_path` 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.

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,

```
prompt> set_max_delay 12 -from [get_clocks CLK1]
prompt> 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 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 higher priority. 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

The `report_timing_requirements` command generates a report on timing exceptions that have been set. For example,

```
prompt> report_timing_requirements
...
Description           Setup      Hold
-----
TIMING EXCEPTION          FALSE     FALSE
    -from    SYS_CLK\
    -to      PCI_CLK
TIMING EXCEPTION          FALSE     FALSE
    -from    PCI_CLK\
    -to      SYS_CLK
...

```

You can reduce the scope of the report by using the path specification options `-from`, `-to`, `-through`, `-rise_from`, `-fall_to`, and so on, to match the path specifiers used when the original exceptions were created.

Exception-setting commands are sometimes ignored for a number of reasons. For example, if a false path is specified from port A to port Z1, but no such path exists between those ports, the exception setting is ignored. To get a report on ignored exceptions, use the `-ignored` option of the `report_timing_requirements` command. For example,

```
prompt> report_timing_requirements -ignored
...
Description           Setup      Hold
-----
NONEXISTENT PATH          FALSE     FALSE
    -from    SYS_2x_CLK\
    -to      PCI_CLK
NONEXISTENT PATH          FALSE     FALSE
    -from    SDRAM_CLK\
    -to      PCI_CLK
...

```

Note:

In the PrimeTime tool, the command to report timing exceptions is `report_exceptions` rather than `report_timing_requirements`.

Removing Exceptions

To remove a timing exception previously set with the `set_false_path`, `set_max_delay`, `set_min_delay`, or `set_multicycle_path` command, use the `reset_path` command.

You control the scope of exception removal in the `reset_path` command by specifying `-from`, `-to`, `-through`, `-rise_from`, `-fall_to`, and so on. The path specification and object must match the original path specification and object used to set the exception. Otherwise, the `reset_path` command has no effect. For example,

```
prompt> set_false_path -from [get_clocks CLK]
...
prompt> reset_path -from [get_pins ff1/CP]
          # ff1 clocked by CLK

prompt> set_false_path -through [get_pins {d/z g/z}]
...
prompt> reset_path -through [get_pins a/z]
          # where a fans out to d
```

In each of these two examples, the object in the `reset_path` command does not match the original object, so the paths are not reset, even though they might have exceptions applied.

You can use the `-reset_path` option in an exception-setting command to reset all of the exceptions set on a path before the new exception is applied. For example,

```
prompt> 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.

The `reset_design` command removes all exceptions from the design.

Preset and Clear Timing

In IC Compiler, using the `-enable_preset_clear_arcs` option of the `report_timing` command enables analysis of preset and clear timing arcs, that is, analysis of a timing arc that passes through the asynchronous preset or clear input of a flip-flop and ends at the Q output of the flip-flop. These timing arcs are reported like other timing conditions. However, by default, the timing information is not used to optimize the design.

Setting the `timing_enable_preset_clear_arcs` variable to true enables analysis of preset and clear timing arcs and also causes IC Compiler to use the timing results for optimization of the design. In that case, if preset or clear timing violations are found, IC Compiler attempts to fix them. When the variable is set to `true`, preset and clear timing

analysis is always performed, whether or not the `-enable_preset_clear_arcs` option is used in the `report_timing` command.

Data-to-Data Checks

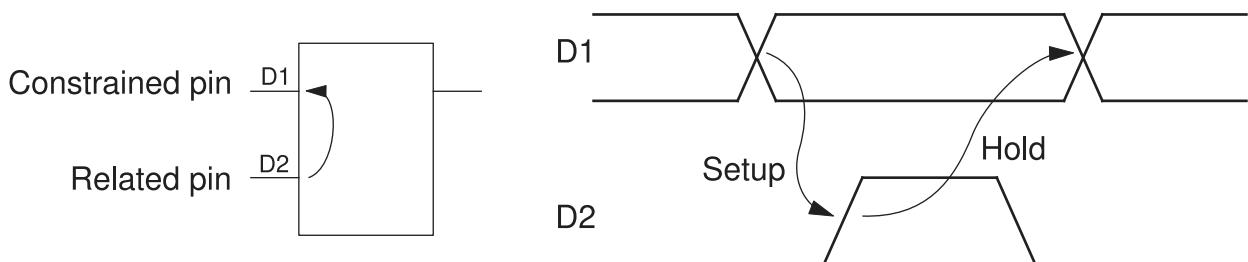
The tool can perform setup and hold checking between two data signals, neither of which is defined to be a clock, at any two pins in the design. A timing constraint between two data (nonclock) signals is called a nonsequential constraint. This feature can be useful for checking the following types of 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

You can define such checks by using the `set_data_check` command, or you can define them for a library cell by setting nonsequential constraints for the cell in Library Compiler. You should use data checks only in situations such as those described previously. Data checks should not be considered a replacement for ordinary sequential checking.

[Figure 65](#) shows a simple example of a cell that has a nonsequential constraint. The cell has two data inputs, D1 and D2. The rising edge of D2 is the active edge that might be used to latch data at D1. Pin D1 is called the constrained pin and pin D2 is called the related pin. In a sequential setup or hold check, pin D2 would be considered the clock pin. However, for any of a number of reasons, it might be desirable to consider the signal at D2 a data signal, and not define it to be a clock.

Figure 65 Simple Data Check Example



In this example, the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge. If these nonsequential constraints are not already defined for the library cell, you can define them in the tool with the `set_data_check` command.

Specifying Data-to-Data Checks

You use the `set_data_check` command to specify that data-to-data checks be performed between two data pins. This is the command syntax:

```
set_data_check
{-from from_object |-rise_from from_object | -fall_from from_object}
{-to to_object | -rise_to to_object | -fall_to to_object}
[-setup | -hold]
[-clock clock_object]
[check_value]
```

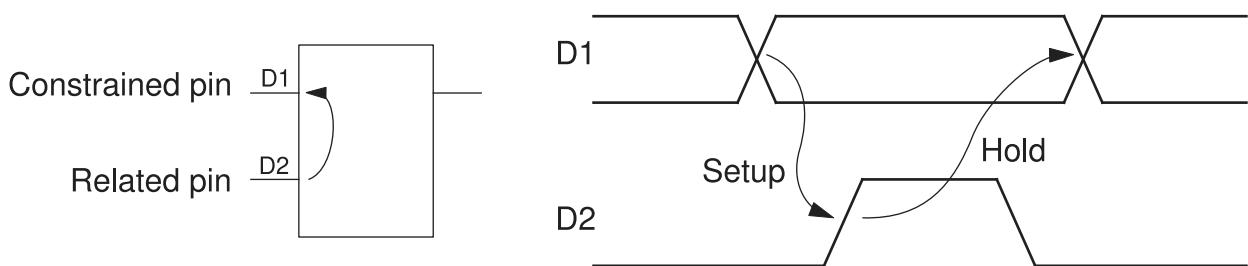
Use the command options as follows:

- To specify a pin or port in the current design as the related pin, use the `-from`, `-rise_from`, or `-fall_from` option.
- To specify a pin or port in the current design as the constrained pin, use the `-to`, `-rise_to`, or `-fall_to` option.
- To specify that the data check value be for setup only or hold only, use the `-setup` option or `-hold` option. Otherwise, the value applies to both setup and hold.
- To specify the name of a single clock that launches the signal for the related pin, use the `-clock` option.

Example 1

In [Figure 66](#), 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.

Figure 66 Data Check on Rising Edge



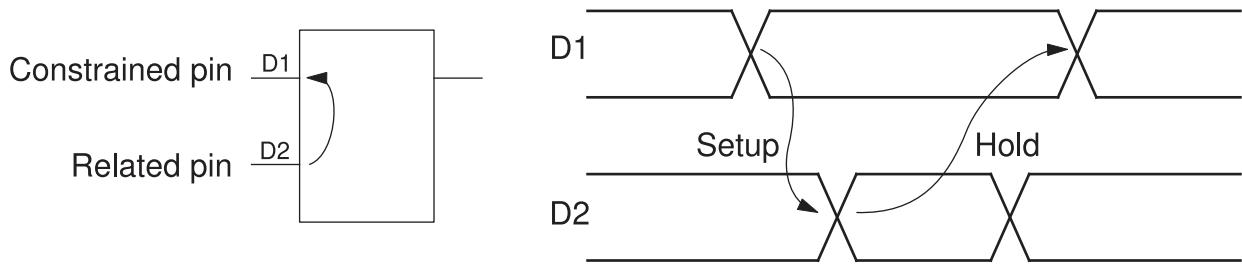
To define these data-to-data checks, use commands similar to the following:

```
prompt> set_data_check -rise_from D2 -to D1 -setup 3.5
prompt> set_data_check -rise_from D2 -to D1 -hold 6.0
```

Example 2

In [Figure 67](#), the data checks apply to both rising and falling edges on the related pin. Use the `-from` option instead of the `-rise_from` or `-fall_from` option.

Figure 67 Data Checks on Rising and Falling Edges



To define these data checks, use the following commands:

```
prompt> set_data_check -from D2 -to D1 -setup 3.5
prompt> set_data_check -from D2 -to D1 -hold 6.0
```

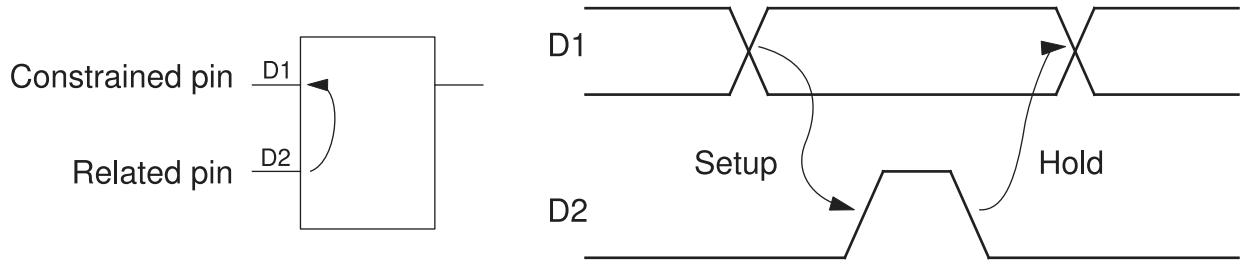
Example 3

You can define a no-change data check by specifying only a setup check from the rising edge and a hold check from the falling edge:

```
prompt> set_data_check -rise_from D2 -to D1 -setup 3.5
prompt> set_data_check -fall_from D2 -to D1 -hold 3.0
```

The tool interprets this as a no-change check on a positive-going pulse. The resulting timing check is illustrated in [Figure 68](#).

Figure 68 No-Change Data Check

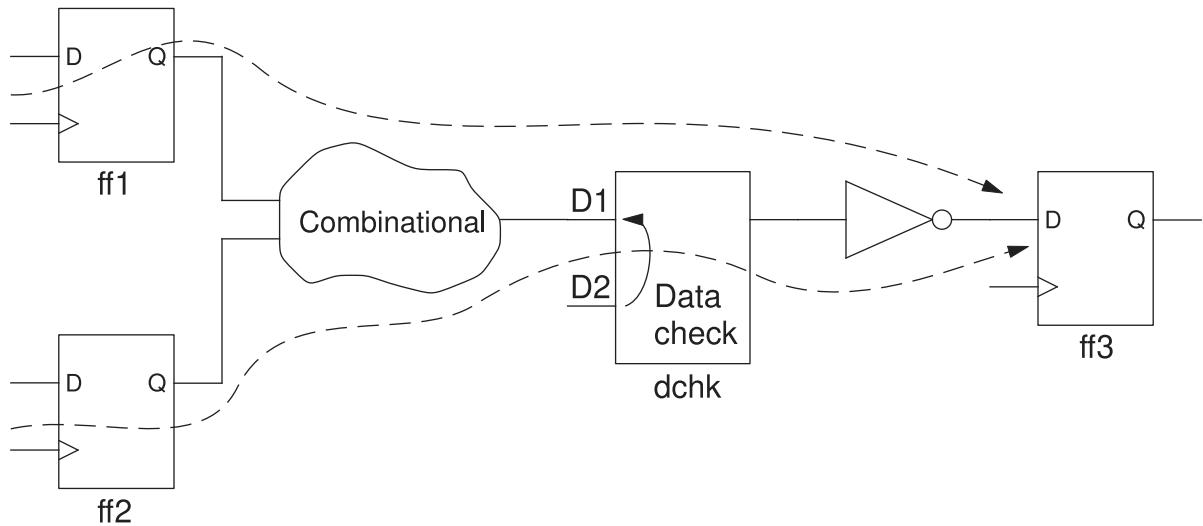


To remove data checks set with the `set_data_check` command, use the `remove_data_check` command.

Generating Timing Reports for Data Checks

Data checks are nonsequential, so they do not break timing paths. For example, in Figure 69, the data check between D1 and D2 does not interrupt the timing paths shown by the dashed-line arrows. If you were to define the signal at D2 to be a clock, the check would be sequential and the paths would be terminated at D1.

Figure 69 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, the tool reports the data checks that apply to the pin. For example, for the

circuit shown in [Figure 69](#), the `report_timing -to dchk/D1` command generates a data check report, whereas `report_timing -through dchk/D1` generates a timing report on ordinary paths that pass through the specified pin and paths that end at the data check pin.

Data Checks and Clock Domains

In a data check, signals arriving at a constrained pin or related pin can come from different clock domains. The tool checks the signal paths separately and puts them into different clock groups, just like in ordinary 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 do this, 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

You can use the `timing_enable_non_sequential_checks` variable to specify that the tool perform data checking for any cell that has nonsequential timing constraints defined in the library cell, as long as the signal at the related pin is not defined to be a clock. If the signal is defined to be a clock, the tool ignores the library-defined nonsequential timing constraints. The variable is set to false by default.

Note:

This behavior is different from that of PrimeTime, which always honors nonsequential arc checks.

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 on defining nonsequential constraints in the library, 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 the 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, you can use the `-clock clock_name` option of the `set_data_check` command.

The `remove_data_check` command does not remove data checks defined in library cells.

4

Operating Conditions

The logic library can specify multiple sets of chip operating conditions (process, voltage, and temperature). Use the `set_operating_conditions` command to choose the operating conditions to use for timing analysis and optimization. In the best-case/worst-case or on-chip variation analysis mode, the tool simultaneously checks the circuit for the two extreme operating conditions, minimum and maximum.

Operating conditions are described in the following sections:

- [Operating Condition Definitions](#)
 - [Setting Operating Conditions](#)
 - [Operating Condition Modes](#)
 - [Minimum and Maximum Delay Calculations](#)
 - [Using Two Libraries for Min-Max Analysis](#)
 - [Setting On-Chip Variation Derating Factors](#)
 - [Advanced On-Chip Variation Analysis](#)
 - [Parametric On-Chip Variation Analysis](#)
 - [Voltage and Temperature Scaling Between Libraries](#)
 - [Clock Reconvergence Pessimism Removal](#)
-

Operating Condition Definitions

The operating conditions of a design include the process, voltage, and temperature parameters under which the chip is intended to operate. The Design Compiler and IC Compiler tools analyze and optimize the design under the conditions you specify.

To see the libraries and operating conditions being used in the current design, use the `report_design` command. To get a list of the operating conditions available in a particular library and to view their characteristics, use either the `report_operating_conditions -library lib_name` command or the `report_lib -operating_condition lib_name` command.

Operating conditions are usually specified in the logic library of the design. Multiple sets of conditions can be specified, such as worst, typical, and best; and commercial, industrial, and military specifications.

The following excerpt from a logic library in Liberty format defines three operating conditions called BEST, TYPICAL, and WORST:

```
library (my_library) {
    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";
    }
}
```

...

For more information about defining operating conditions in libraries, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*.

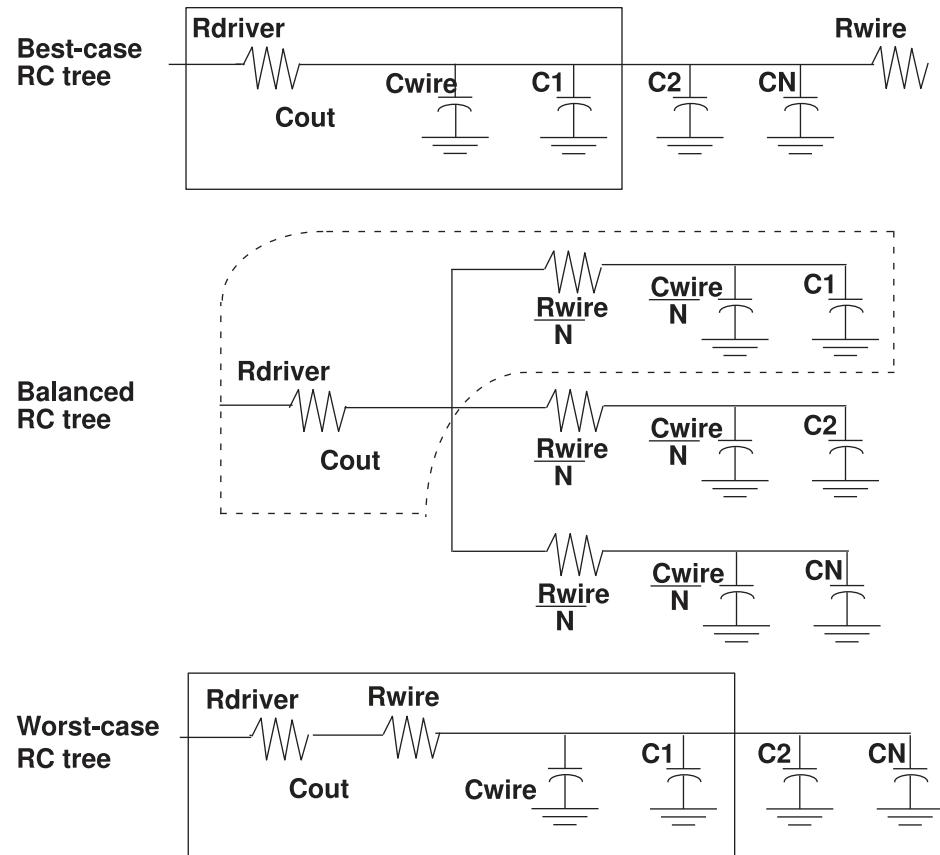
You can create your own operating conditions in Design Compiler or IC Compiler with the `create_operating_conditions` command. For example,

```
prompt> create_operating_conditions -name WC_CUSTOM -library tech1 \
           -process 1.2 -temperature 30.0 -voltage 2.8 \
           -tree_type worst_case_tree
```

This example creates a new operating condition called WC_CUSTOM, which is associated with the existing logic library called tech1. The process, temperature, and voltage values are 1.2, 30.0, and 2.8 respectively. The tree type is set to `worst_case_tree`.

The tree type is the type model used for estimating the distribution of net resistance and capacitance for each net. The `-tree_type` option can be set to `best_case_tree`, `balanced_tree`, or `worst_case_tree`. The tree type models are shown in [Figure 70](#).

Figure 70 RC Interconnect Topologies for Fanout of N



In the best-case tree, the receiver being analyzed is physically close to the driver compared to the other loads, so all the net resistance is assumed to be beyond the capacitive loads, resulting in a larger drive current and the smallest possible delay.

In the balanced tree, all the receivers are the same distance from the driver, so the net resistance and capacitance are evenly distributed between multiple loads, resulting in average delay.

In the worst-case tree, the receiver being analyzed is physically far from the driver compared to the other loads, so all the net resistance is assumed to be between the driver and the capacitive loads, resulting in a smaller drive current and the largest possible delay.

Setting Operating Conditions

In Design Compiler or IC Compiler, the `set_operating_conditions` command specifies the operating conditions for analysis and optimization, so that the tool can use the appropriate set of parameter values in the logic library. This is the syntax of the command:

```
set_operating_conditions
  [-analysis_type bc_wc | on_chip_variation]
  [-min min_condition] [-max max_condition]
  [-min_library min_lib] [-max_library max_lib]
  [-min_phys min_proc] [-max_phys max_proc]
  [-library library_name]
  [-object_list objects]
  [condition_name]
```

If you do not use the `-analysis_type` option, the tool performs analysis at a single operating condition. In that case, you specify only a single condition name. For example,

```
prompt> set_operating_conditions TYPICAL
```

The `-analysis_type` option selects a mode in which the minimum and maximum conditions are analyzed at the same time. You specify the min-max analysis type, either `bc_wc` (best-case/worst-case) or `on_chip_variation`. In the best-case/worst-case mode, the tool uses the maximum operating condition for setup checks and the minimum operating condition for hold checks. In the on-chip variation mode, the tool uses the maximum operating condition for all maximum path delays and the minimum operating condition for all minimum path delays. For example, for a setup check, the on-chip variation mode uses the maximum operating condition for the data path and the minimum operating condition for the capture clock path.

In the best-case/worst-case or on-chip variation mode, use the `-min` and `-max` options to specify the minimum and maximum operating conditions. If the same condition name can be found in multiple libraries, you can use the `-min_library` and `-max_library` options to specify the libraries from which to obtain the operating conditions. If these operating conditions come from the same library, you can use the `-library` option to specify the library name. Use can the `-min_phys` and `-max_phys` options to specify the names of the process resources (technology files) containing the interconnect resistance and capacitance values to be used for minimum and maximum analysis.

The `-object_list` option lets you apply the operating conditions to specified cells or ports in multivoltage applications. By default, the whole design is affected.

To remove operating conditions that have been previously set, use the `set_operating_conditions` command without specifying any operating conditions, or use the `reset_design` command.

Operating Condition Modes

The Design Compiler and IC Compiler tools offer three modes with respect to operating conditions: single, best-case/worst-case, and on-chip variation:

- In the single operating condition mode, the tool uses a single set of delay parameters for the whole circuit, based on one set of process, temperature, and voltage conditions.
- In the best-case/worst-case mode, the tool simultaneously checks the circuit for the two extreme operating conditions, minimum and maximum. For setup checks, it uses maximum delays for all paths. For hold checks, it uses minimum delays for all paths.
- In the on-chip variation mode, the tool 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 does the opposite. This mode models the effects of variation in operating conditions across the chip.

[Table 5](#) and [Table 6](#) show the clock arrival times, delays, operating conditions, and derating used for setup checks and for hold checks under each of the operating condition modes.

Table 5 Timing Parameters Used for Setup Checks

Analysis mode	Launch clock path	Data path	Capture clock path
Single operating condition	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)
Best-case/worst-case mode	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, worst-case operating condition
On-chip variation mode	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

Table 6 *Timing Parameters Used for Hold Checks*

Analysis mode	Launch clock path	Data path	Capture clock path
Single operating condition	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)
Best-case/worst-case mode	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, best-case operating condition
On-chip variation mode	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

A best-case/worst-case or on-chip variation 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, the tool uses the longest path, worst-case operating conditions, latest-arriving clock edge, maximum cell delays, longest transition times, and so on.

You can do best-case/worst-case or on-chip variation 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 Min-Max Analysis](#).

Min-Max Cell and Net Delay Values

Each timing arc in the design can have a minimum and a maximum delay to account for variations in operating conditions. You can specify these values in the following ways:

- Annotate delays from one or two SDF files.
- Have the tool calculate the delay.

To annotate delays from one or two SDF files, do one of the following:

- Use min and max from the SDF triplet.
- Use two SDF files.

To have the tool calculate the delay, do one of the following:

- Use a single operating condition with timing derating factors to model variation.
- Use two operating conditions (best case and worst case) to model the possible on-chip variation.

[Table 7](#) summarizes the usage of minimum and maximum delays from SDF triplet data.

Table 7 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***.

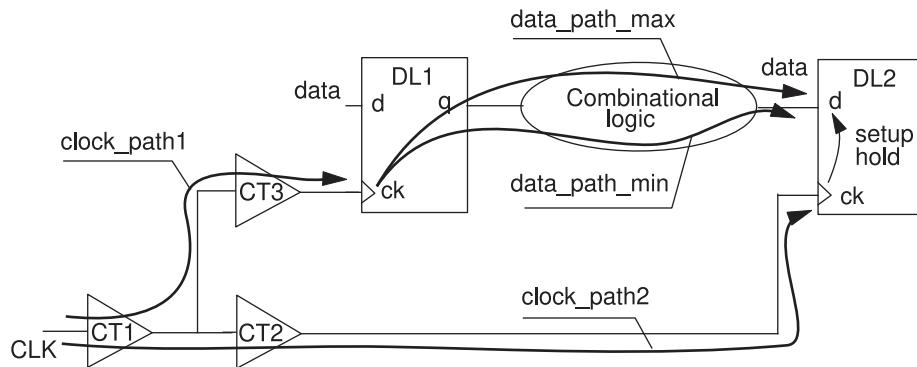
Setup and Hold Checks

This section provides examples of how setup and hold timing checks are done for a single operating condition, for simultaneous best-case/worst-case operating conditions, and for on-chip variation.

Path Delay Tracing for Setup and Hold Checks

[Figure 71](#) shows how setup and hold checks are done.

Figure 71 Design Example Showing Setup and Hold Checks



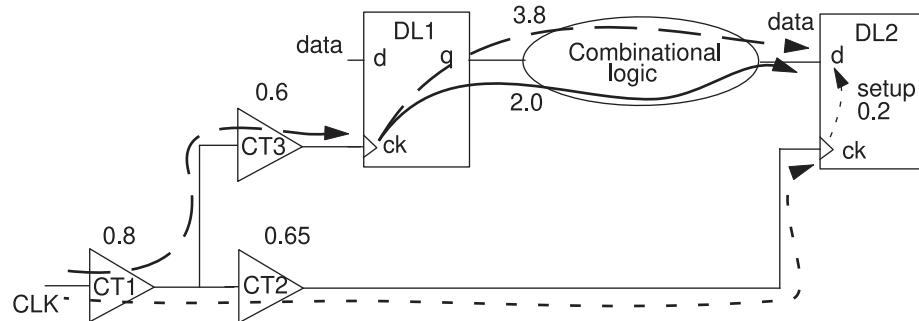
- The setup timing check from pin DL2/ck to DL2/d considers
 - Maximum delay for clock_path1
 - Maximum delay for data path (data_path_max)
 - Minimum delay for clock_path2
- The hold timing check from pin DL2/ck to DL2/d considers
 - Minimum delay for clock_path1
 - Minimum delay for data path (data_path_min)
 - Maximum delay for clock_path2

The `data_path_min` and `data_path_max` values can be different due to multiple topological paths in the combinational logic that connects DL1/q to DL2/d.

Setup Timing Check for Worst-Case Conditions

[Figure 72](#) shows how cell delays are computed for worst-case conditions. To simplify the example, the net delays are ignored.

Figure 72 Setup Check Using Worst-Case Conditions



The tool checks for a setup violation as follows:

$$\text{clockpath1} + \text{datapathmax} - \text{clockpath2} + \text{setup} \leq \text{clockperiod}$$

In the equation,

$$\text{clockpath1} = 0.8 + 0.6 = 1.4$$

$$\text{datapathmax} = 3.8$$

$$\text{clockpath2} = 0.8 + 0.65 = 1.45$$

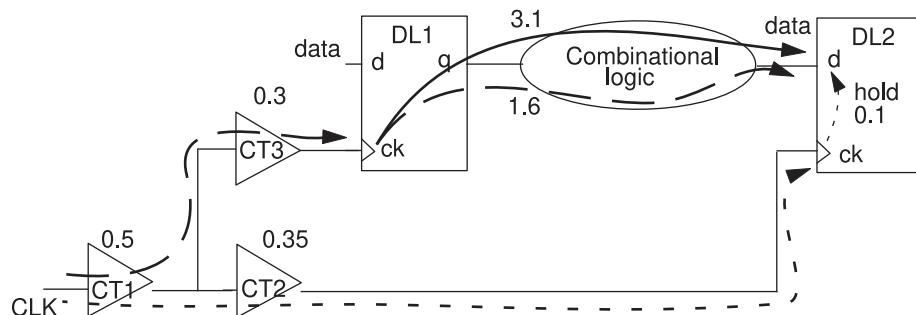
$$\text{setup} = 0.2$$

The clock period must be at least $1.4 + 3.8 - 1.45 + 0.2 = 3.95$.

Hold Timing Check for Best-Case Conditions

Figure 73 shows how cell delays are computed for best-case conditions. Note that the cell delays are different from the delays in Figure 72.

Figure 73 Hold Check Using Best-Case Conditions



The tool checks for a hold violation as follows:

$$\text{CLOCKPATH1} + \text{DATAPATHMIN} - \text{CLOCKPATH2} - \text{HOLD} \geq 0$$

In the equation,

$$\text{clockpath1} = 0.5 + 0.3 = 0.8 \quad \text{datapathmin} = 1.6 \quad \text{clockpath2} = 0.5 + 0.35 = 0.85 \quad \text{hold} = 0.1$$

No hold violation exists because $0.8 + 1.6 - 0.85 - 0.1 = 1.45$, which is greater than 0.

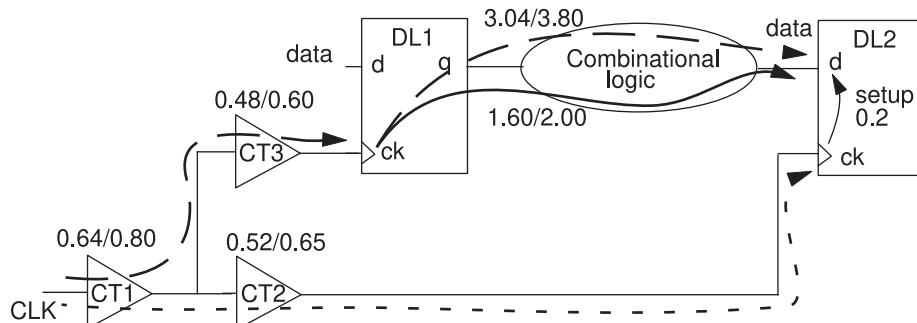
Simultaneous Best-Case/Worst-Case Conditions

In best-case/worst-case mode, the tool uses worst-case conditions for setup checks and best-case conditions for hold checks. The timing reports for setup and hold checks are the same as for [Figure 72](#) and [Figure 73](#).

Path Tracing in On-Chip Variation Mode

In [Figure 74](#), each delay of a cell or a net has an uncertainty because of on-chip variation. For example, you can specify that on-chip variation can be between 80 percent and 100 percent of the nominal delays for worst-case conditions. [Figure 74](#) shows the resulting delays.

Figure 74 On-Chip Variation 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. The tool checks for a setup violation as follows:

$$\text{clockpath1} + \text{datapathmax} - \text{clockpath2} + \text{setup} \leq \text{clockperiod}$$

In the expression,

$$\text{clockpath1} = 0.80 + 0.60 = 1.40 \text{ (at 100% of worst case)}$$

`datapath_max = 3.80 (at 100% of worst case) clockpath2 = 0.64 + 0.52 = 1.16 (at 80% of worst case) setup = 0.2`

The clock period must be at least $1.40 + 3.80 - 1.16 + 0.2 = 4.24$

On-chip variation affects the clock latencies; therefore, you only need it when you are using propagated clock latency. If you specify ideal clock latency, you can have the tool consider on-chip variation by increasing the clock uncertainty values with the `set_clock_uncertainty` command.

Using Two Libraries for Min-Max Analysis

The `set_min_library` command directs the tool to use two technology 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. You 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, the tool 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, the tool uses that timing information for minimum analysis. Otherwise, it uses the information in the maximum library. For more information, see the `set_min_library` man page.

Setting On-Chip Variation Derating Factors

You can have the tool adjust minimum delays and maximum delays by specified factors to model the effects of varying operating conditions. This adjustment of calculated delays is called derating. Derating affects the delay and slack values calculated for the design.

The `set_timing_derate` command specifies the adjustment factors and the scope of the design to be affected by derating. For example,

```
prompt> set_timing_derate -early -cell_delay 0.9
prompt> set_timing_derate -late -cell_delay 1.2
```

The first of these two commands causes all early (shortest-path) delays to be decreased by 10%, such as the capture clock path in a setup check. The second command causes all late (longest-path) delays to be increased by 20%, such as the launch clock path and the data path in a setup check. These changes result in a more conservative analysis than leaving delays at their original calculated values.

The `-early` or `-late` option specifies whether the shortest-path or longest-path delays are affected by the derating factor. Early path delays include the capture clock path for a setup check, and the launch clock path and data path for a hold check. Late path delays include the launch clock path and data path for a setup check, and the capture clock path for a hold check.

The fixed-point value in the command specifies the derating factor applied to the delays. Use a value of less than 1.0 to reduce the delay of any given path or cell check. Similarly, use a derating factor greater than 1.0 to increase the delay of any given path or cell check. Typically, derating factors less than 1.0 are used for early (shortest-path) delays and greater than 1.0 for late (longest-path) delays. This approach results in a more conservative analysis.

The last derating value to be set overrides any previously set values. To reset the derating factor globally to 1.0, use the `reset_timing_derate` command.

Delays are adjusted according to the following formula:

$$\text{delay_new} = \text{old_delay} + (\text{derate_factor} - 1.0) * \text{abs}(\text{old_delay})$$

When the delay is a positive value (the usual case), this equation is reduced to:

$$\text{delay_new} = \text{old_delay} * \text{derate_factor}$$

For negative delay, the delay adjustment equation is reduced to:

$$\text{delay_new} = \text{old_delay} * (2.0 - \text{derate_factor})$$

Delays can be negative in some unusual situations. For example, if the input transition is slow and the output transition is fast for a cell, the time at which the input signal reaches the 50% trip point can be later than the time at which the output signal reaches the 50% trip point. The resulting delay from input to output is negative. A similar situation can occur for a net because of a change in delay caused by crosstalk. In addition, the library hold time requirement for a sequential cell can be negative.

The `-rise` or `-fall` option specifies whether rise delays or fall delays are affected by derating. If neither option is used, both types of delays are affected. The `-clock` or `-data` option specifies whether clock paths or data paths are affected by derating. If neither option is used, both types of paths are affected. A clock path is a path from a clock source to the clock input pin of a launch or capture register. A data path is a path starting from an input port or from the data output of a launch register, and ending at an output port or at the data input of a capture register.

The `-net_delay`, `-cell_delay`, and `-cell_check` options let you specify the functional scope of the design to be affected by derating. The `-net_delay` option derates the delay from a net driver to a net load. The `-cell_delay` option derates the delay from a cell input to a cell output. The `-cell_check` option derates cell timing checks, which are the cell hold and removal times for early derating or the cell setup and recovery times for late derating. Cell checks are setup and hold timing requirements for cells, not delays. If you do not use any of these options, the derating factor applies to both net delays and cell delays, but not cell timing checks.

By default, derating applies to both the static and dynamic components of net delay. Dynamic delay is the change in delay resulting in crosstalk with other signals. When you use the `-net_delay` option, you can optionally use the `-static` or `-dynamic` option to restrict derating to apply to only the static or only the dynamic component of net delay.

If you want only some cells or nets to be affected by derating, you can list them in the `set_timing_derate` command. The list can include library cells, hierarchical cells, leaf-level cells, and nets. In the absence of a list, derating applies to the whole design. To ensure that the setting on an instance is preserved during optimization, the tool sets the `size_only` attribute of the instance to `true`.

If you set a derating factor on a net that crosses a hierarchical boundary, the derating affects the whole net, not just the part of the net listed in the command. Also, if you set a derating factor on a hierarchical cell, the derating factor extends to cells and nets within and below the hierarchy.

In case of conflict between different `set_timing_derate` values specified for different levels of the design, the more detailed command (with the narrowest scope) has precedence. For example, the following commands have decreasing order of precedence:

```

prompt> set_timing_derate -late -cell_delay 1.4 [get_cells inv*]
      # derates a collection of cells in the design

prompt> set_timing_derate -late -cell_delay 1.3 \
      [get_lib_cells class/ND*]
      # derates cells in a collection of cells in a library class

prompt> set_timing_derate -late -cell_delay 1.2
      # derates all cells

prompt> set_timing_derate -late -1.1
      # derates all nets and cells
  
```

If you use the `-derate` option of the `report_timing` command, the timing report shows the derate factors applied to each cell and net.

To obtain a report on the current timing derating factors that have been set, use the `report_timing_derate` command. For example,

```
prompt> report_timing_derate
# reports all derating settings
prompt> report_timing_derate [get_cells U*]
# reports derating settings on all cells named U*
```

Use the `-include_inherited` option to include reporting of derating values inherited from other objects, such as a lower-level cell that inherits its derating values from a higher-level cell. Otherwise, the command reports only the derating values set directly on the specified object.

The `timing_library_derate_is_scenario_specific` variable determines whether the `set_timing_derate` command and the `read_aocvm` command (described in the next section) apply to the current scenario only or to all scenarios for library cell objects. The default is false. Set this variable to true if you want library cell derating settings to be scenario-specific. Changing this variable removes all timing derating values already set.

Advanced On-Chip Variation Analysis

Advanced on-chip variation (AOCV) is an optional method of accuracy improvement in the IC Compiler tool that determines varying derating factors for different paths based on the path lengths and physical distance spans. A path that is longer (has more gates) or covers a greater physical distance tends to have less total variation because the random variations from gate to gate tend to cancel each other out. Accordingly, AOCV applies higher derating values to short paths and lower derating values to long paths.

This method is less pessimistic than a conventional OCV analysis, which relies on constant derating factors that do not consider path-specific metrics. The improved timing accuracy affects both timing reports and design optimization.

The primary purpose of AOCV derating is to improve timing accuracy in the IC Compiler tool after clock tree synthesis has been performed and the clocks have been changed from ideal to propagated.

How AOCV Analysis Works

In the IC Compiler tool, each AOCV data table specifies the derating factors as a function of the number of successive gates in the path (the path depth) and physical distance span of the path, the objects to which the table applies, the signal sense (rising or falling), and the derating type (early or late).

By default, when the `timing_aocvm_enable_analysis` variable is set to `true`, AOCV analysis applies throughout the design, including both clock and data path delays. It uses depth values calculated from both clock network paths and data paths, and applies

separate depth values for launch and capture paths. This behavior is consistent with the default behavior of AOCV analysis in the PrimeTime tool.

You can control the behavior of the AOCV analysis mode by setting the `timing_aocvm_analysis_mode` variable. The following settings are supported:

- `""` (empty string) – AOCV derating applies throughout design. Both clock and data network objects are included in the depth computation. This is the default.
- `clock_network_only` – AOCV derating applies only to delays in the clock network. Constant (OCV) derating applies to data paths, if applied by the `set_timing_derate` command. Otherwise, data path delays are not derated. Depth is based only on objects in the clock network.
- `separate_data_and_clock_metrics` – AOCV derating applies to clock paths and data paths, using depth values calculated separately from objects in the clock paths for clock path delay derating, and objects in the data paths data path delay derating.
- `combined_launch_capture_depth` – AOCV derating applies to clock paths and data paths, using a depth value obtained from launch paths and capture paths combined. This option cannot be combined with the `separate_data_and_clock_metrics` option.

You can combine the `clock_network_only` setting with either of the two other settings, `separate_data_and_clock_metrics` or `combined_launch_capture_depth` (but not both at the same time). For example,

```
icc_shell> set_app_var timing_aocvm_analysis_mode \
    "clock_network_only combined_launch_capture_depth"
```

Before clock synthesis, when ideal clocking is used for timing analysis, the clock network path depth is considered to be zero. In this situation, you can optionally specify an estimated number of gates in typical clock paths so that the tool can use a more realistic path depth for AOCV analysis. To do this, set the `timing_aocvm_ideal_clock_mode` variable to `true`, and set the `timing_aocvm_ideal_clock_depth` variable to the estimated number of gates in each clock network timing path.

When the IC Compiler tool performs timing analysis, it applies the AOCV derating factors to timing arcs based on the content of the AOCV data tables. AOCV derating works with both calculated delays and delays annotated in Standard Delay Format (SDF). You can optionally restrict the usage to just the minimum or just the maximum operating condition.

By default, when both ordinary on-chip variation (OCV) and AOCV derating are specified, the more specific setting has higher priority. You can control this behavior by setting the `timing_aocvm_precedence_compatibility` variable. For details, see the man page.

Cell Adjustment Factor

Some cells might have a larger variation effect than others. You can optionally set an adjustment factor on specified library cells or cell instances for the purpose of counting the number of successive gates in each path. The default factor is 1.0 for all gates.

For example, a buffer is often implemented as two inverters in series. In that case, the buffer cell can be assigned an adjustment factor of 2.0. Then, when counting the number of successive gates in a path, each buffer counts as 2.0 gates.

The library can specify the adjustment factors for each library cell. You can also specify the adjustment factors in the IC Compiler tool using the `set_aocvm_coefficient` command.

AOCV Flow

In the IC Compiler tool, these are the steps to invoke the AOCV feature:

1. If you are using library-based AOCV data, verify that the library contains the necessary data. If you are using file-based AOCV data, create a text file containing the tables of derating values to be applied to objects in the design or to the whole design.
2. Enable AOCV by setting the `timing_aocvm_enable_analysis` variable to `true`.
3. To consider physical distance as well as gate counts, set the `timing_aocvm_enable_distance_analysis` variable to `true`.
4. If you are using file-based AOCV data, read the data tables into the tool with the `read_aocvm` command.

You can report the cells and nets to which AOCV applies and the applied derating values by using the `report_aocvm` command. To remove some or all AOCV derating values from objects in the design, use the `remove_aocvm` command.

AOCV Data Formats

To perform AOCV timing analysis, the IC Compiler tool needs the derating values to apply to objects as a function of the number of successive gates in the path (the path depth) and optionally, the physical distance span of the path. There are two ways to provide this information:

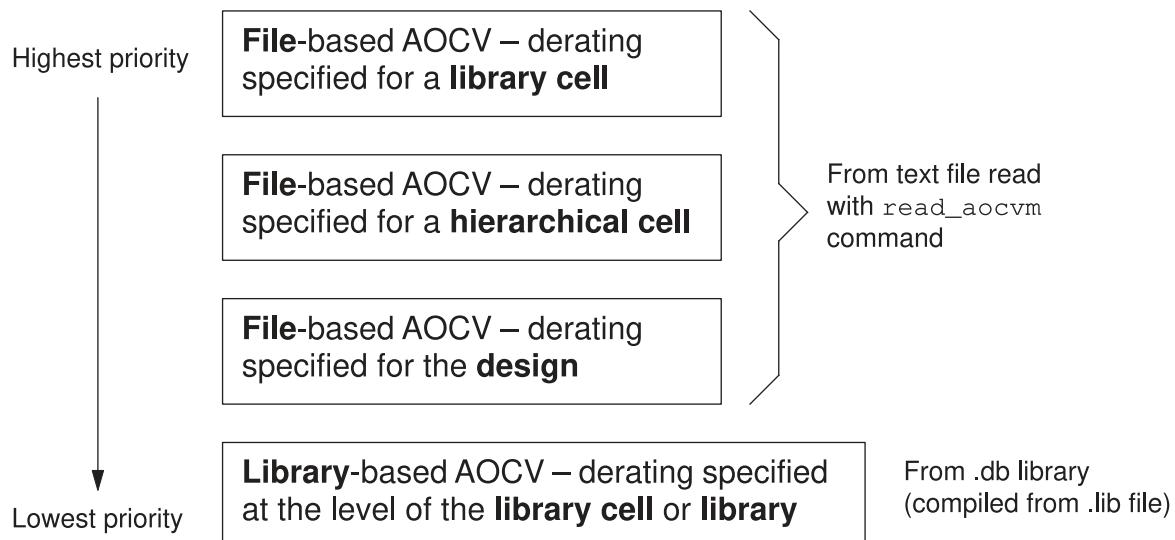
- Library-based AOCV data – The `.db` library specifies the AOCV derating data in a set of tables originally written in Liberty (`.lib`) format. When the IC Compiler tool reads in the `.db` library, it has access to all the AOCV derating data.
- File-based AOCV data – You create a text file containing the derating data tables and you read in the file using the `read_aocvm` command in the IC Compiler tool.

Both library-based and file-based AOCV data can be generated by the Synopsys SiliconSmart characterization tool. For details, see the *SiliconSmart ACE User Guide*, available on SolvNet.

You can use both library-based and file-based AOCV derating data at the same time. In case of any conflict, the file-based data has priority, overriding the library-based data.

The file-based AOCV data can specify the derating data at the level of the library cell, hierarchical cell, or design. In case of conflict, a more specific setting has priority over a more general one. In other words, library cell derating overrides hierarchical cell derating, and hierarchical cell derating overrides design-level derating, as summarized in [Figure 75](#).

Figure 75 AOCV Derating Data Order of Priority



Library-Based AOCV Data

Library-based AOCV derating is specified in Liberty (.lib) syntax and compiled into .db format by the Library Compiler tool, just like other types of cell library data. You can use the Synopsys SiliconSmart characterization tool to generate the depth-based (but not distance-based) AOCV values.

In the .lib file, the derating values are specified in table format as a function of depth or both depth and distance, similar to delay values. For more information about the syntax of library-based AOCV data, see “Advanced OCV Modeling” in the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*, available on SolvNet.

File-Based AOCV Data

For the IC Compiler tool, the AOCV data file specifies the derating values to apply to objects as a function of the path depth, and optionally, the physical distance spanned by the path. Data files produced by the `write_binary_aocvm` command in the PrimeTime tool, both binary and compressed, are accepted by the IC Compiler tool.

A data table containing depth and distance dependency information contains multiple rows of data, each row corresponding to a different distance value. By default, the IC Compiler tool selects the most conservative value from among all rows of distance data. If you want to use only a particular distance row in the table instead, specify that row using the `-distance_row` option of the `read_aocvm` command. To enable full usage of all the distance information, set the `timing_aocvm_distance_analysis` variable to true.

Each line in the data table specifies the derating factors as a function of the number of successive gates in the path (the path depth), the objects to which the table applies, the signal sense (rising or falling), and the derating type (early or late). The tool selects the objects based on the following definitions:

- `object_type` `design` | `cell` | `lib_cell`
- `delay_type` `cell` | `net` | `cell net`
- `path_type` `clock` | `data` | `clock data`
- `object_spec` [*patterns*]

In the `object_spec` definition, the *patterns* field 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 field of the `object_spec` definition, similar to using the `regexp` Tcl command within commands that gather objects, such as `get_cells` and `get_lib_cells`.

You can use any of the options of the related object-gathering command in the patterns field. For example, if the object type is `lib_cell`, you can use any of the arguments of the related `get_lib_cells` command in the patterns field.

The tool can annotate cell and net tables on the design by using the `object_type` `design`. It can also annotate cell tables on library cells by using the `lib_cell` object class and annotate cell and net tables on hierarchical cells by using the `cell` object class.

Note:

To add a comment in any location within the file, use double forward slashes (//).

[Table 8](#) shows the syntax definition for the AOCV file format.

Table 8 AOCV File Format Syntax

Field specifier	Field description
version	AOCV version number.
object_type	design cell lib_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	string
depth	A set of M floating-point values, where M can be zero.
distance	A set of N floating-point values, where N can be zero. Distance values are used only when the <code>timing_aocvm_enable_distance_analysis</code> variable is set to true.
table	<p>A set of N x M floating-point values. There are also the following special cases:</p> <ul style="list-style-type: none"> • If N==0, the table is of size M. • If M==0, the table is of size N. • If M==0 and N==0, the table is of size 1. <p>Linear interpolation is used to determine points that have not been defined in the table. The tool does not extrapolate beyond the lowest or highest values specified in the table.</p>

The following example of an AOCV file sets an early AOCV table for clock nets in the whole design:

```

version          1.0
object_type:    design
rf_type:        rise fall
delay_type:     cell net
derate_type:    early
path_type:      clock
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 a late AOCV table for clock paths, for all hierarchical cells with an operating condition voltage of 1.2 V:

```

version          1.0
object_type:    cell
rf_type:        rise
delay_type:     cell
derate_type:    late
path_type:      clock
object_spec:   * -filter (voltage_max==1.2 && is_hierarchical==true)
depth:          1 2 3
distance:       100 1000
table:          1.21 1.11 1.09 \
                1.23 1.16 1.14

```

Advanced On-Chip Variation Reporting

Internally calculated AOCV derating values are shown in the derate column of the `report_timing` command if you have specified the `-derate` option.

You can use the `report_aocvm` command to display AOCV derate table data. You can show design objects annotated with early, late, rise, fall, cell, or net derate tables. You can also use this command to determine cells and nets that have been annotated or not annotated with AOCV information.

You can get information about timing arcs that start, pass through, or end at a specified cell in the design. For example,

```
icc_shell> report_aocvm -arc_details [get_cells CTS_clk_delay10]
```

This generates a report on the depth metrics and AOCV derating factors of the timing arc associated with the specified cell. For more information about the `report_aocvm` command, see the man page.

Fast Path-Based AOCV

In the IC Compiler tool, by default, AOCV considers the shortest path leading up to each endpoint to determine the derating value for that endpoint. It applies that same derating value to all paths leading up to that endpoint, including longer paths, giving pessimistic results for those paths.

When you use AOCV, you can further increase the accuracy by invoking the fast path-based option of AOCV. This option selects a limited number of worst paths leading up to each endpoint for analysis, and calculates a separate (less pessimistic) derating value for each path in isolation from other paths, based on the actual number of stages in the path. The resulting slack from this calculation is larger (less negative) than the default AOCV calculation. The fast path-based option then executes a `set_path_margin` on each path, which adjusts the slack to the more accurate value for that path.

To apply the fast path-based option, execute the `apply_fast_pba_analysis` command. By default, the command applies the `set_path_margin` commands immediately. Alternatively, you can use the `-check_only file_name` option to write the `set_path_margin` commands to a file, without applying them. You can then examine the file and possibly execute it as a script using the `source` command.

The generated margin values, if applied to the design, are reported by the `report_timing` command. You can see the applied `path_margin` values in the timing report.

Another way to invoke fast path-based AOCV analysis is to enable it with the `set_route_opt_strategy -fast_pba_analysis true` command. In that case, the analysis and timing adjustment occur during execution of the `route_opt -incremental` or `focal_opt` command.

The increased accuracy (reduction in pessimism) comes at the cost of runtime and memory. You can control the tradeoff between accuracy and runtime/memory by setting the analysis options using the `set_fast_pba_analysis_options` command. To remove all the generated margin values, use the `remove_fast_pba_analysis` command.

Parametric On-Chip Variation Analysis

In the Design Compiler Graphical and IC Compiler tools, parametric on-chip variation (POCV) analysis is an optional timing analysis mode that takes a statistical approach to modeling on-chip variation. In this mode, the tool computes arrival times, required times, and slack as statistical distributions rather than fixed minimum and maximum values. The timing results are less pessimistic, providing a better basis for timing optimization, and have better correlation with the PrimeTime tool when POCV analysis is enabled in that tool. You can invoke POCV analysis by making just a few modifications to your existing design flow.

How POCV Analysis Works

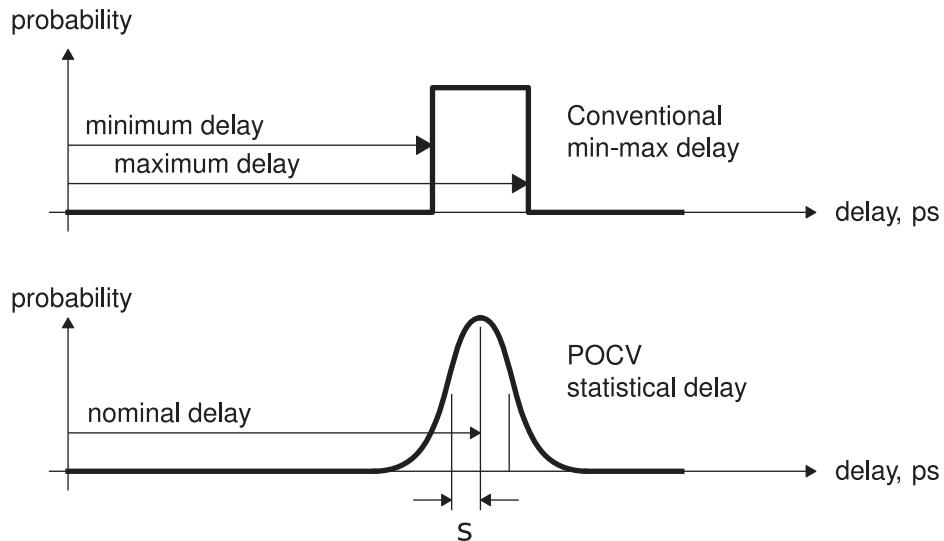
In POCV timing analysis, instead of specifying absolute minimum and maximum delays for each timing arc, the tool calculates the delay as a function of the Gaussian or normal distribution P :

$$\begin{aligned} \text{delay} &= \text{nominal_delay} + s * P \\ &= \text{nominal_delay} + (C * \text{nominal_delay}) * P \end{aligned}$$

The tool gets the delay variation value σ (Greek letter sigma) from the .db library, if that information is available in the library, or it calculates this value from a coefficient C supplied in a POCV data file, where $\sigma = C * \text{nominal_delay}$.

[Figure 76](#) graphically compares conventional min-max delay with POCV statistical delay.

Figure 76 Comparison of Min-Max Delay and POCV Statistical Delay



The two ways of describing delay variation differ in the following ways:

- The min-max delay specifies the absolute minimum and maximum delay values for the timing arc. The actual delay has an equal probability of occurring anywhere between these two extremes, and no chance of occurring beyond these extremes.
- The POCV statistical model specifies a nominal delay value and a variation σ . The delay has the highest probability of occurring at the nominal value and smaller probabilities of occurring farther away from the nominal value. The actual delay has a 99.7 percent chance of falling within 3σ of the nominal delay.

After the tool determines the delay distribution of each timing arc, it can propagate them statistically through timing paths to calculate each arrival time, required time, and slack value as a total nominal value plus a cumulative variation.

To determine the cumulative delay of a path, the tool statistically combines the delay distribution of each stage. This is more accurate than simply adding the worst-case value from each stage. The resulting delay and slack values are more realistic and less pessimistic than values calculated by simple min-max addition.

By default, the final slack reported by the `report_timing` command is the slack at three standard deviations (3σ) below the nominal slack:

$$\text{reported_slack} = \text{nominal_slack} - (3.0\sigma)$$

You can change the sigma multiplier from 3.0 to a larger or smaller value to tighten or loosen the timing constraint.

Parametric On-Chip Variation Flow

Performing POCV analysis consists of the following steps.

1. In a multicorner-multimode (MCMM) flow, enable scenario-specific saving of library derating information:

```
prompt> set_app_var timing_library_derate_is_scenario_specific true
```

2. If you want to save the POCV data in the .ddc or Milkyway database, do the following:

```
prompt> set_app_var timing_save_library_derate true
```

3. Enable POCV analysis:

```
prompt> set_app_var timing_pocvm_enable_analysis true
```

4. If you are applying a single variation coefficient C at the level of the library cell, hierarchical cell, or design, read in the POCV coefficient data file:

```
prompt> read_aocvm coefs.pocv
```

In a multicorner-multimode (MCMM) flow, do this for each scenario.

This step is optional if you are using library cells already characterized for POCV timing variation, as described in the next section.

5. Report the design objects that have been annotated with POCV coefficients:

```
prompt> report_ocvm -type pocvm
```

6. Generate the desired timing reports. To include statistical information (mean and sigma) in a report, use the -variation option:

```
prompt> report_timing -variation -from ... -to ...
```

POCV Data Formats

To perform POCV timing analysis, the Design Compiler Graphical or IC Compiler tool needs the variation σ for each timing arc. There are two ways to provide this information:

- Library-based POCV data – The .db library specifies the variation data in a set of tables. Each table entry specifies the variation σ for a combination of input slew and output load, for a given timing arc in the library cell, similar to the delay tables in the same library. When the tool reads in the .db library, it has access to all the POCV variation data at the timing arc level.
- File-based POCV data – You read in a coefficient data file in plain text format using the `read_aocvm` command. This applies a single coefficient value C for each library

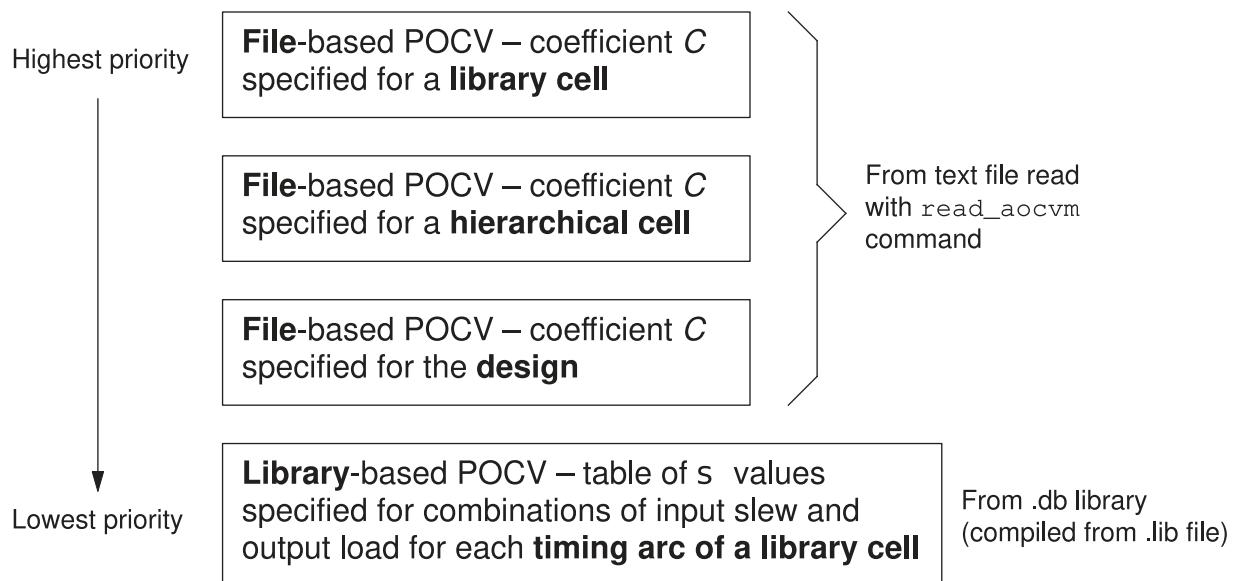
cell, hierarchical cell, or design specified in the text file. The delay variation σ is calculated as $C * nominal_delay$. All the timing arcs of a cell share the same coefficient C , irrespective of input slew and output load.

Both library-based and file-based POCV data can be generated by the Synopsys SiliconSmart characterization tool. For details, see the *SiliconSmart ACE User Guide*, available on SolvNet.

You can use both library-based and file-based POCV coefficient data at the same time. In case of any conflict, the file-based data has priority, overriding the library-based data.

The file-based POCV data can specify the coefficient at the level of the library cell, hierarchical cell, or design. In case of conflict, a more specific setting has priority over a more general one. In other words, a library cell coefficient overrides a hierarchical cell coefficient, and a hierarchical cell coefficient overrides a design-level coefficient, as summarized in [Figure 77](#).

Figure 77 POCV Variation Data Order of Priority



Library-Based POCV Data

Library-based POCV s values are specified in Liberty (.lib) syntax and compiled into .db format by the Library Compiler tool, just like other types of cell library data. You can use the Synopsys SiliconSmart characterization tool to generate the POCV values.

In the .lib file, the variation values σ are specified in table format as a function of input slew and output load for each timing arc in the cell, just like delay values. It is recommended that you use the same set of slew and load index values for delay and POCV data. For each timing table, two POCV data tables are required: one for early delays and another for late delays. [Example 2](#) shows the early and late POCV data tables in a .lib library.

Example 2 POCV Variation Data Table

```
ocv_sigma_cell_rise ("delay_template_7x7") {
    sigma_type :="early";
    index_1("0.0088, 0.0264, 0.0608, 0.1296, 0.2672, 0.5424, 1.0936");
    index_2("0.0010, 0.0024, 0.0052, 0.0108, 0.0221, 0.0445, 0.0895");
    values("0.000476, 0.000677, 0.001075, 0.001870, 0.003438, 0.006626, 0.0129
22", \
    "0.000651, 0.000901, 0.001303, 0.002081, 0.003678, 0.006818, 0.013144", \
    "0.000840, 0.001166, 0.001714, 0.002558, 0.004112, 0.007249, 0.013529", \
    "0.001115, 0.001520, 0.002193, 0.003317, 0.005087, 0.008153, 0.014445", \
    "0.001521, 0.002033, 0.002883, 0.004242, 0.006522, 0.010072, 0.016258", \
    "0.002155, 0.002793, 0.003853, 0.005563, 0.008424, 0.012955, 0.020171", \
    "0.003204, 0.003977, 0.005321, 0.007515, 0.010960, 0.016582, 0.025786");
}

ocv_sigma_cell_rise ("delay_template_7x7") {
    sigma_type :="late";
    index_1("0.0080, 0.02640, 0.06080, 0.12960, 0.26720, 0.54240, 1.09360");
    index_2("0.00100, 0.00240, 0.00520, 0.01080, 0.02210, 0.04450, 0.08950");
    values("0.000476, 0.000677, 0.001075, 0.001870, 0.003438, 0.006626, 0.0129
22", \
    "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");
}
```

For more information about the table syntax, see the *Library Compiler Timing, Signal Integrity, and Power Modeling User Guide*, available on SolvNet.

File-Based POCV Data

The POCV data file specifies the delay variation coefficients C for the design, for hierarchical cells in the design, or library cells used in the design. The tool accepts both binary and compressed data files produced by the `write_binary_aocvm` command in PrimeTime. POCV analysis always applies to both data paths and clock paths.

[Table 9](#) shows the syntax of the POCV data file.

Table 9 POCV Data File Syntax

Field specifier	Field description
version	4.0 (on-chip variation data file version number; must be 4.0 or later)
ocvm_type	pocv (Parametric on-chip variation)

Table 9 POCV Data File Syntax (Continued)

Field specifier	Field description
object_type	design cell lib_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	<i>string</i> (Name of object annotated with POCV coefficient)
coefficient	<i>value</i> (Variation coefficient)

The following example sets the POCV coefficient for falling transitions on the lib28/invx2 library cell:

```
version:      4.0
ocvm_type:   pocv
object_type: lib_cell
rf_type rise: fall
delay_type : cell
derate_type: early
object_spec: lib28/invx2
coefficient: 0.05
```

The following example sets the POCV coefficient for early delays in the whole design:

```
version:      4.0
ovcm_type:   pocv
object_type: design
rf_type:     rise fall
delay_type:  cell net
derate_type: early
object_spec: // Leave object_spec blank if object_type is design
coefficient: 0.05
```

The following example sets the POCV coefficient for late delays for all hierarchical cells with an operating condition voltage of 1.2 V:

```
version       1.0
ovcm_type:   pocv
object_type: cell
rf_type:     rise
delay_type:  cell
derate_type: late
```

```
object_spec: * -filter (voltage_max==1.2 && is_hierarchical==true)
coefficient: 0.07
```

The `object_spec` definition specifies the object name, optionally using an expression that is evaluated based on the attributes of the object, similar to using the `regexp Tcl` command within commands that gather objects, such as `get_cells` and `get_lib_cells`. You can use any of the options of the related object-gathering command in the `patterns` field. For example, if the `object_type` is `lib_cell`, you can use any of the arguments of the related `get_lib_cells` command in the `patterns` field.

To add a comment in any location within the file, use double forward slashes (//).

Saving Scenario-Specific POCV Data

If you are using the multicorner-multimode (MCMM) flow and you have POCV coefficients that were annotated by `read_aocvm` commands specific to each operating condition used to characterize the same library, you can enable saving of file-based POCV information in the .ddc file or Milkyway cell on a per-scenario basis.

To enable per-scenario derating, do the following before you read in the design:

```
prompt> set_app_var timing_library_derate_is_scenario_specific true
```

To enable saving of the POCV data in the .ddc or Milkyway database:

```
prompt> set_app_var timing_save_library_derate true
```

If the scenario-specific variable is set later in the session, then all existing file-based POCV settings and timing derates are removed from the design. Both of these variables affect only library cell information in POCV data files; instance information is not affected.

Enabling Parametric On-Chip Variation Analysis

To enable POCV analysis, set the following variable:

```
prompt> set_app_var timing_pocvm_enable_analysis true
```

In the Design Compiler Graphical tool, setting this variable to true checks out a DC-Extension license.

When this variable is set to true, the `update_timing` command performs POCV timing analysis instead of conventional min-max timing analysis. It calculates nominal delays together with the statistical variation of each delay.

For reporting purposes, the `report_timing` command needs to determine which values in a distribution are considered worst-case. By default, it chooses values at 3.0 standard deviations (3σ) away from the mean. In other words, for an arrival time distribution, the

worst-case early arrival is 3.0 standard deviations below the mean, and the worst-case late arrival is 3.0 standard deviations above the mean.

You can change the number of standard deviations to specify the values in a distribution that are considered worst-case. For example, a lower value of 2.5 reports less worst-case variation and has more relaxed timing constraints. Conversely, a higher value such as 3.5 reports more worst-case variation and enforces more restrictive timing constraints, possibly increasing the number of reported timing violations.

To change the number of standard deviations used to calculate worst-case values, set the `timing_pocvm_corner_sigma` variable. For example,

```
prompt> set_app_var timing_pocvm_corner_sigma 3.1
```

You should set the `timing_pocvm_enable_analysis` and `timing_pocvm_corner_sigma` variables as part of the initial setup for the session, before you load the design.

Where POCV derating information is not available, the tool uses conventional on-chip variation derating, if set with the `set_timing_derate` command. You can modify the default behavior by setting the `timing_ocvm_precedence_compatibility` variable. For details, see the man page for this variable.

Reading File-Based Parametric On-Chip Variation Data

The parametric on-chip variation data file specifies the POCV coefficients to be applied to the design, to hierarchical cells in the design, or to library cells used in the design. Reading this file is optional if you are using library cells already characterized for POCV timing variation. Coefficients that you read in have priority over any conflicting library-based variation data.

To read in the POCV data file, use the `read_aocvm` command. For example,

```
prompt> read_aocvm coefs.pocv
```

If the POCV data file contains incorrect syntax, the tool issues an error message and does not annotate the data.

If you are using wildcards for the library name in the `object_spec` field of your data file (for example, `object_spec: */INV1`), read the POCV data file in the scenario that matches the operating condition used to characterize the library. All cells that match the `object_spec` in the scenario are annotated with the POCV data. Remember that you must set the `timing_library_derate_is_scenario_specific` variable to `true` before you read in the data.

If a cell matches more than one `object_spec`, the last one read in is used. For example, using `read_aocvm` to read in all POCV data files with a wildcard for the library name results in usage of the last set of POCV data to match the `object_spec` field. For this

reason, you might want to consider adding the library name to the `object_spec` field in the POCV data file. For example,

```
object_spec: my_lib/INV1
```

The following script example shows how to read in POCV data files for multiple scenarios:

```
set scenarios [func_ff func_ss]

foreach scenario $scenarios {
    current_scenario $scenario

    if { $scenario == "func_ff" } {
        read_aocvm ./pocv_data/std_cell_func_ff.pocv_coefficients
    } elseif { $scenario == "func_ss" } {
        read_aocvm ./pocv_data/std_cell_func_ss.pocv_coefficients
    } else {
        echo "No POCV data for scenario $scenario"
    }
}
```

You should use the `read_aocvm` command only after the design has been loaded and linked so that libraries are present in the session. If you are saving file-based POCV data for library cells in the `.ddc` file or Milkyway cell, you only need to read in the POCV data once and save the design. In future sessions, opening the design automatically reloads the POCV data.

POCV Guard Band

In the POCV flow, you can specify an additional guard band to model variation effects beyond what is already modeled in the cell library or POCV data file. Applying a guard band increases the POCV derating effect, which makes timing checks more restrictive.

To apply POCV guard bands, use the `set_timing_derate` command with the `-pocvm_guardband` option. For example, the following commands increase the POCV derating by 5 percent for calculation of both early and late delays:

```
icc_shell> set_timing_derate -cell_delay -pocvm_guardband -early 0.95
icc_shell> set_timing_derate -cell_delay -pocvm_guardband -late 1.05
```

Each POCV guard band derating factor applies to both the nominal cell delay (mean) and the cell delay variation (σ). To report the POCV guard band derating factors currently being applied, use the `report_timing_derate -pocvm_guardband` command.

To remove all POCV guard band derating factors that have been set, use the `reset_timing_derate -pocvm_guardband` command.

Scaling the POCV Coefficients

In the POCV flow, you can adjust the POCV variation by a specified factor without making any changes to the cell library or POCV data file. To do this, use the `set_timing_derate -pocvm_coefficient_scale_factor` command.

For example, the following command scales the POCV variation by 3 percent for calculation of both early and late delays:

```
icc_shell> set_timing_derate -cell_delay \
           -pocvm_coefficient_scale_factor -early 0.97
icc_shell> set_timing_derate -cell_delay \
           -pocvm_coefficient_scale_factor -late 1.03
```

The specified scaling factor applies only the cell delay variation (σ), not to the nominal cell delay (mean). To report the POCV scaling factors, use the `report_timing_derate -pocvm_coefficient_scale_factor` command.

To remove all POCV scaling factors, use the `reset_timing_derate -pocvm_coefficient_scale_factor` command.

Enabling Constraint Variation

You can improve the accuracy of POCV analysis with the following optional setting:

- To enable variation for setup and hold constraints, set the `timing_enable_constraint_variation` variable to `true`.

The tool reads the constraint variation data provided in the Liberty Variation Format (LVF) in the library.

The `report_timing` command shows the adjusted mean constraint value. To see the original mean and sigma values, use the `report_delay_calculation` command on the constraint arc.

Reporting POCV Information

To display POCV information for the current scenario, use the `report_ocvm -type pocvm` command. This command reports design objects annotated with all types of variation coefficients (early/late, rise/fall, cell/net). For example,

```
prompt> report_ocvm -type pocvm -nosplit
*****
Report : ocvm
         -type pocvm
Design : top
Scenario(s): func_ff
...
```

```
*****
POCV coefficient:
*****
```

		Fully annotated	Partially annotated	Not annotated
Total	1924	1924	0	0
Leaf cells	1924	1924	0	0
Nets	2317	2317	0	0
	4241	4241	0	0

...

To view annotation information for leaf cells or nets, specify the object with the command:

```
prompt> report_ocvm -type pocvm -nosplit [get_cell i1]
*****
Report : ocvm
    -type pocvm
    object_set
Design : top
Scenario(s): func_ff
...
*****
```

POCV coefficient: 0.0500

Name	Object	Opcond	Process	Path_type	Sense	Delay	Inherited
i1	cell	*	early	clock	rise	cell	lib/INV1
i1	cell	*	early	clock	fall	cell	lib/INV1
i1	cell	*	early	data	rise	cell	lib/INV1
i1	cell	*	early	data	fall	cell	lib/INV1

POCV coefficient: 0.0500

Name	Object	Opcond	Process	Path_type	Sense	Delay	Inherited
i1	cell	*	late	clock	rise	cell	lib/INV1
i1	cell	*	late	clock	fall	cell	lib/INV1

...

To report the POCV variation data for a timing arc in a library cell, use the `-arc_details` option of the `report_ocvm` command:

```
prompt> report_ocvm -type pocvm -arc_details \
    -nosplit [get_lib_cells C9GPLL/IVLLP]
*****
Report : ocvm
    -type pocvm
    object_set
Design : simple1
...
*****
```

```

From pin:          C9GPLL/IVLLP/A
To   pin:          C9GPLL/IVLLP/Z
arc sense:        negative-unate

min rise table

      | Load
Slew  | 0.0040000  0.0160000  0.0640000  0.1800000  0.3200000
-----+
0.0056000 | 0.0247500  0.0623400  0.2140700  0.5729000  1.0125000
0.1016000 | 0.0445700  0.0759000  0.2145000  0.5766700  1.0247999
0.2616000 | 0.0688500  0.1124900  0.2431600  0.5838100  1.0253000
0.5336000 | 0.1009300  0.1565900  0.3031500  0.6198700  1.0306000
1.0696000 | 0.1541400  0.2243000  0.4135300  0.7353700  1.1167001
1.5000000 | 0.1920000  0.2722300  0.4857700  0.8339600  1.2023000

```

```
min fall table
```

```
...
```

To report all cells and nets that are missing POCV data, use the following command:

```

prompt> report_ocvm -type pocvm -nosplit -list_not_annotated
...

```

Reporting the POCV Analysis Results

By default, the report generated by the `report_timing` command after POCV analysis looks just like a report resulting from conventional min-max timing analysis. The reported minimum delays are 3.0 standard deviations below the mean, and the reported maximum delays are 3.0 standard deviations above the mean. Slack values are calculated by comparing the required arrival times with the 3σ minimum and maximum delay values.

To display the POCV variation parameters in the timing report, use the `-variation` option of the `report_timing` command. For example,

```

prompt> report_timing -variation -significant_digits 5 -nosplit
...
      Point           Mean       Sensit      Incr      Path
-----
clock CLK (rise edge)           0.000000  0.000000  0.000000  0.000000
clk (in)                      0.024778  0.000000  0.024778  c 0.024778 r
clk (net)                      0.000000  0.000000  0.000000  0.024778 r
clk_buf/A (INV1)               0.005320  0.000000  0.005320  c 0.030098 r
clk_buf/Z (INV1)               0.029918  0.000796  0.030098  c 0.060196 r
clk_buf (net)                  0.000000  0.000000  0.000000  0.060196 r
clk_buf_2/A (INV4)              0.004087  0.000000  0.004087  c 0.064283 r
clk_buf_2/Z (INV4)              0.030745  0.000818  0.030935  c 0.095077 r
clk_buf_2 (net)                 0.000000  0.000000  0.000000  0.095077 r
ff2/CP (FD1)                   0.010982  0.000000  0.010982  c 0.106059 r
ff2/Q (FD1)                     0.069437  0.001625  0.070189  c 0.176248 f

```

ff_out (net)			0.000000	0.176248	f
inv/A (IV)	0.053112	0.000000	0.053112	c 0.229360	f
inv/Z (IV)	0.011549	0.000321	0.011579	c 0.240939	r
inv (net)			0.000000	0.240939	r
inv_2/A (IV2)	0.059238	0.000000	0.059238	c 0.300177	r
inv ⁷ Z (IV2)	0.016281	0.000425	0.016333	c 0.316510	f
inv_2 (net)			0.000000	0.316510	f
ff3 ⁷ D (FD1)	0.032528	0.000000	0.032528	0.349038	f
data arrival time				0.349038	
clock CLK (rise edge)			1.000000	1.000000	
clk (in)	0.020387	0.000000	0.020387	c 1.020387	r
clk (net)			0.000000	1.020387	r
clk_buf/A (INV1)	0.005320	0.000000	0.005320	c 1.025707	r
clk_buf/Z (INV1)	0.029885	0.000795	0.029705	c 1.055412	r
clk_buf (net)			0.000000	1.055412	r
clk_buf_2/A (INV4)	0.008080	0.000000	0.008080	c 1.063492	r
clk_buf_2/Z (INV4)	0.026440	0.000616	0.026332	c 1.089824	r
clk_buf_2 (net)			0.000000	1.089824	r
ff3 ⁷ CP (FD1)	0.005589	0.000000	0.005589	c 1.095413	r
library setup time			-0.800000	0.295413	
data required time				0.295413	

data required time				0.295413	
data arrival time				-0.349038	

slack (VIOLATED)				-0.053620	

Using the `-variation` option in the `report_timing` command inserts two additional columns into the report, labeled “Mean” and “Sensit.” The “Mean” column shows the nominal incremental delay, whereas the “Sensit” column shows the single-sigma variation in incremental delay.

Voltage and Temperature Scaling Between Libraries

Design Compiler and IC Compiler can use “scaling library groups” to implement voltage and temperature scaling. To get the cell delays at intermediate voltage and temperature conditions, the tool interpolates between the data in separate libraries that have been characterized at different nominal voltage and temperature values. Scaling between the libraries is done during runtime of the tool.

To use this feature, the libraries in the scaling group must contain at least two timing models. The cell data must be consistent with respect to cell names, number and names of pins, and number and names of timing arcs.

To specify the membership of libraries to scaling library groups, use the `define_scaling_lib_group` command. To apply a defined scaling library group to the design or to selected objects in the design, use the `set_scaling_lib_group` command.

To create an intermediate operating condition, use the `create_operating_conditions` command. For example,

```
prompt> define_scaling_lib_group -name group1 \
           { max_v70_t125.db max_v108_t125.db }
prompt> set_scaling_lib_group group1
prompt> set_operating_conditions SLOW_108 -library max_v108_t125
prompt> create_operating_conditions -name SLOW_95 \
           -library max_v70_t125 -process 1 -voltage 0.95 -temperature 125
prompt> set_operating_conditions SLOW_95 \
           -object_list { mid1/bot1 top_i }
```

This example creates a one-dimensional scaling group consisting of two libraries characterized at two extreme voltages, 0.70 and 1.08 volts. It applies this scaling group to the design and selects the `SLOW_108` operating condition from the 1.08-volt library as the default operating condition for the design. Then it creates a new operating condition at 0.95 volts and applies it to two objects in the design. The tool performs interpolation between the .70-volt and 1.08-volt libraries to obtain the timing characteristics at 0.95 volts, and uses that information for timing analysis of the two objects.

Note:

PrimeTime supports the `define_scaling_lib_group` command but not the `set_scaling_lib_group` command. Instead, PrimeTime uses the `link_path_per_instance` variable and `set_min_library` command. In Design Compiler or IC Compiler, the `write_link_library` command writes out commands that set the `link_path_per_instance` variable appropriately in PrimeTime.

To perform both voltage and temperature scaling at the same time, you would use four libraries in the scaling group rather than two, representing the four possible combinations of voltage and temperature extremes: high voltage and high temperature, high voltage and low temperature, low voltage and high temperature, and low voltage and low temperature.

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 on-chip variation analysis or timing derating. Automated correction of this inaccuracy is called clock reconvergence pessimism removal, or CRPR.

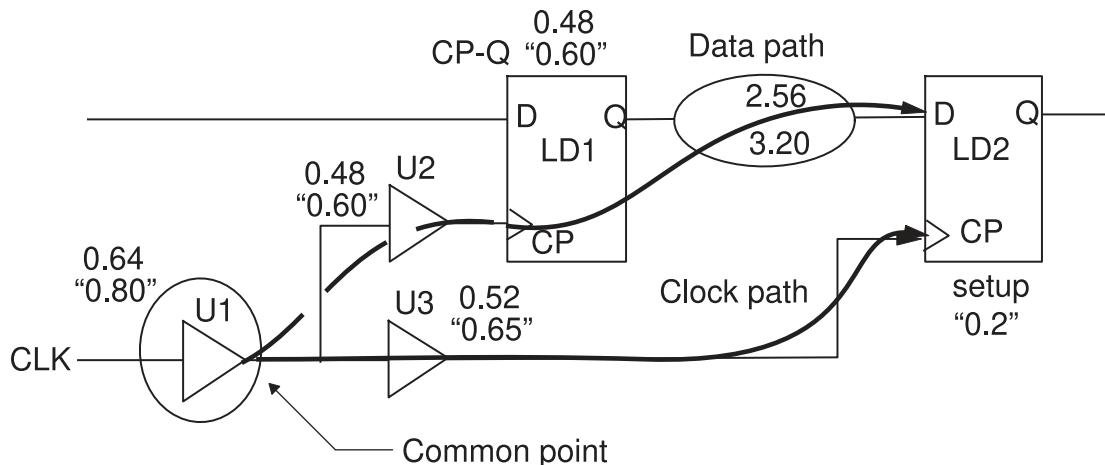
The tool performs clock reconvergence pessimism removal, when enabled, at the same time as regular timing analysis. You need to enable the clock reconvergence pessimism removal before you do timing analysis.

On-Chip Variation Example

Consider the following command sequence for running on-chip variation analysis and the corresponding design in [Figure 78](#):

```
prompt> set_operating_conditions -analysis_type \
          on_chip_variation -min MIN -max MAX
prompt> set_timing_derate -net_delay -early 0.80
prompt> report_timing -delay min
prompt> report_timing -delay max
```

[Figure 78](#) Clock Reconvergence Pessimism Example



[Figure 78](#) shows how the analysis is done. Each delay (considered equal for rising and falling transitions to simplify this example) has a minimum value and a maximum value computed for the minimum and maximum operating conditions.

The setup check at LD2/CP considers the clock path to the source latch (CLK to LD1/CP) at 100 percent worst case, and the clock path to the destination latch (CLK to LD2/CP) at 80 percent worst case.

Although this is a valid approach, the test is pessimistic because clock path1 (CLK to LD1/CP) and clock path2 (CLK to LD2/CP) share the clock tree until the output of U1. The shared segment is called the *common portion*, consisting of just cell U1 in this example. The last cell output in the shared clock segment is called the *common point*, which is the output of U1 in this case.

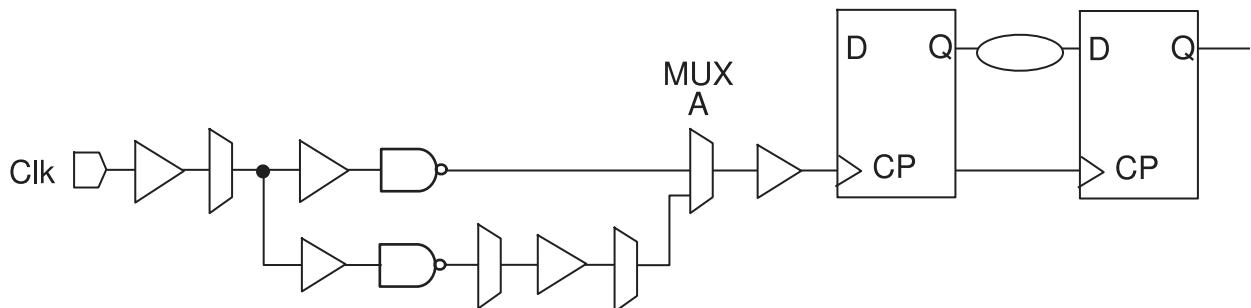
The setup check considers that cell U1 simultaneously has two different delays, 0.64 and 0.80, resulting in a pessimistic analysis in the amount of 0.16. This amount, obtained by subtracting the earliest arrival time from the latest arrival time at the common point, is called the *clock reconvergence pessimism*.

This inaccuracy also occurs in an analogous way for the hold test at the LD2 latch.

Reconvergent Logic Example

[Figure 79](#) shows a situation where clock reconvergence can occur, even in the absence of on-chip variation 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 79 Reconvergent Logic in a Clock Network



Setting Clock Reconvergence Pessimism Removal

To enable clock reconvergence pessimism removal, set the control variable to true before you begin timing analysis, as follows:

```
prompt> set timing_remove_clock_reconvergence_pessimism true
```

By default, the variable is set to false and the feature is disabled. Enabling the feature results in a more accurate (less pessimistic) analysis but causes an increase in runtime and memory usage. Any change in this variable setting causes a complete timing update.

For the on-chip variation example shown in [Figure 78](#), the timing report shows the pessimism removal value as follows:

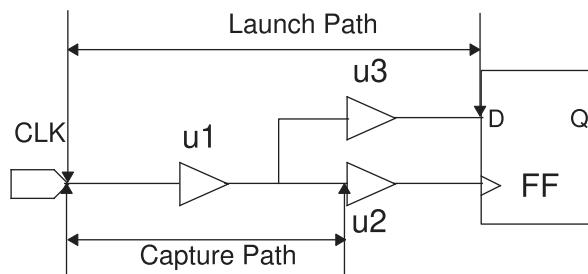
```
prompt> report_timing -delay max
*****
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : my_design
*****
```

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
clock CLK (rise edge)	6.00	6.00
clock network delay (propagated)	1.16	7.16
clock reconvergence pessimism	0.16	7.32
clock uncertainty	0.00	7.32
LD2/CP (FD2)		7.32 r
library setup time	-0.20	7.12
data required time		7.12
data required time		7.12
data arrival time		-5.20
slack (MET)		1.92

Pessimism can also be introduced on paths that fan out from a clock source to the data pin of a sequential device, such as the path shown in [Figure 80](#). (Technically, this is not considered clock reconvergence pessimism because the launching path is considered a data path, not a clock path.)

Figure 80 Timing Path Fanout From Clock Source to Data Pin



To remove pessimism for these paths, set the `timing_crpr_remove_clock_to_data_crp` variable to `true` before you begin timing analysis.

Note:

Calculating CRP for such paths involves analyzing the sequential device associated with each clock to data path separately in the timing analysis. This may cause a severe performance penalty during a timing update.

To specify whether to perform clock reconvergence pessimism removal on clock paths that have opposite-sense transitions in a shared path segment, set the `timing_clock_reconvergence_pessimism` variable. You can set this variable to `normal` (the default setting) or `same_transition`. For the default setting, the tool still performs clock reconvergence pessimism removal with opposite-sense transitions in the shared path segment. In that case, if the two transition types produce different correction values, the smaller of the two values is used. For the `same_transition` setting, the tool performs clock reconvergence pessimism removal only if transitions in the shared path segment have the same sense.

You can set a threshold that allows the tool to ignore small amounts of clock reconvergence pessimism. For computational efficiency, the tool merges multiple points for CRPR calculations when the CRP differences between adjacent points are smaller than the specified threshold. This merging of nodes can reduce CPU and memory usage, without a significant loss of accuracy when an appropriate threshold is set.

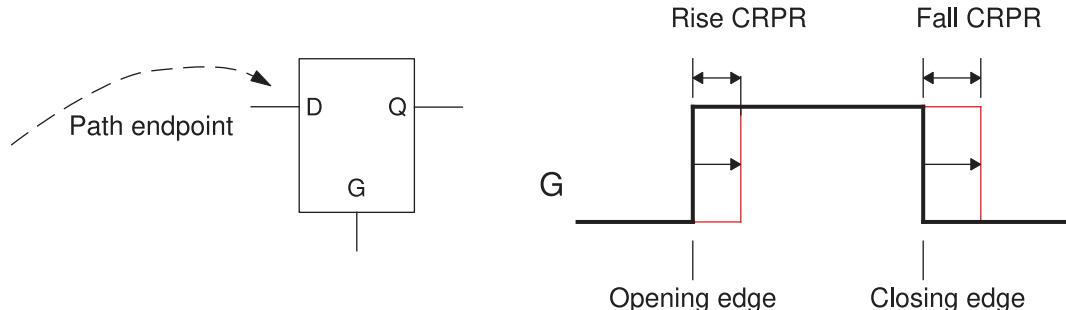
The `timing_crpr_threshold_ps` variable specifies the threshold. The units are picoseconds, irrespective of the time units of the logic library. By default, the variable is set to 20, which allows adjacent nodes to be merged when the difference in clock reconvergence pessimism is 20 ps or less.

For a good balance between performance and accuracy, try setting this variable to one-half the stage delay of a typical gate in the clock network. The stage delay is gate delay plus net delay. You might want to use a larger value during the design phase for faster analysis, and then a smaller value for signoff accuracy.

Transparent Latch Edge Considerations

For a path that ends at a transparent latch, the tool calculates two clock reconvergence pessimism values: one for rising edges and one for falling edges at the common node. The opening and closing clock edges are effectively shifted, each by an amount equal to its corresponding pessimism value, as indicated in [Figure 81](#).

Figure 81 Clock Reconvergence Pessimism Removal 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.

Reporting CRPR Calculations

The `report_crpr` command reports the calculation of clock reconvergence pessimism (CRP) between two register clock pins or ports. You specify the pins of the launch and capture registers, the clock, and type of check (setup or hold). For example,

```
prompt> report_crpr -from [get_pins ffa/CP] \
           -to [get_pins ffd/CP] -setup \
```

The command generates a report showing 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. 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.

5

Timing Constraints

Timing analysis and timing optimization depend on constraints you specify, as described in the following sections:

- [Input Delays](#)
 - [Output Delays](#)
 - [Drive Characteristics at Input Ports](#)
 - [Port Load Capacitance](#)
 - [Ideal Networks](#)
 - [Case Analysis](#)
 - [Mode Analysis](#)
 - [Wire Load Models](#)
 - [Timing Loops](#)
-

Input Delays

The `set_input_delay` command specifies the timing of external paths leading to an input port. The input delay is the arrival time of a transition at an input port relative to a clock edge. The command syntax is:

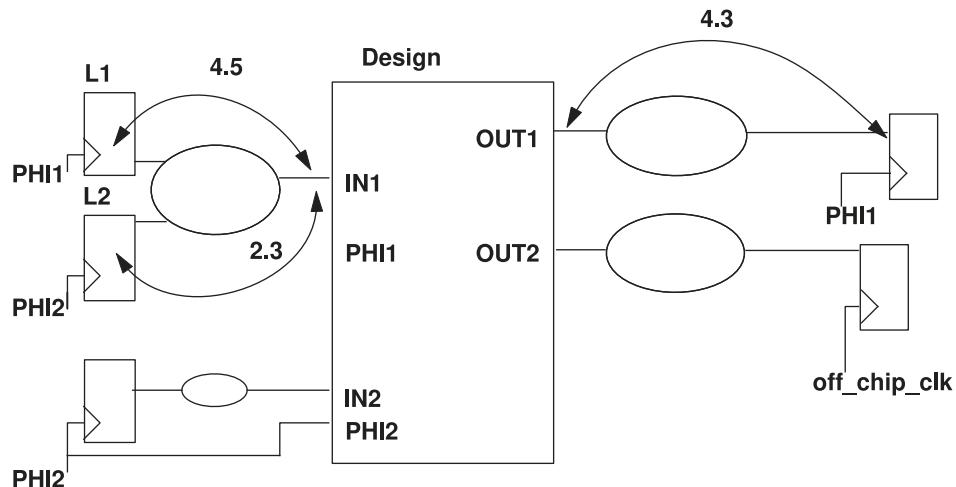
```
set_input_delay
  delay_value
  [-reference_pin pin_port_name]
  [-clock clock_name]
  [-clock_fall]
  [-level_sensitive]
  [-network_latency_included]
  [-source_latency_included]
  [-rise] [-fall]
  [-max] [-min]
  [-add_delay]
  port_pin_list
```

For basic information about specifying input delays and output delays for timing analysis, see [Input and Output Delays](#).

Applying the `set_drive` or `set_driving_cell` commands to the port causes the port to have a cell delay that is the load-dependent value of the external driving-cell delay. To prevent this delay from being counted twice, estimate the load-dependent delay of the driving cell, then subtract that amount from the input delays on the port.

The input delay should equal the path length from the clock pin of the source flip-flop to the output pin of the driving cell, minus the load-dependent portion of the driving cell's delay. For example, see the external path for the L1 clock port to port IN1 in [Figure 82](#).

Figure 82 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. For more information, see the `set_input_delay` man page.

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:

```
prompt> set_input_delay 4.5 -clock PHI1 {IN1}
```

Example 2

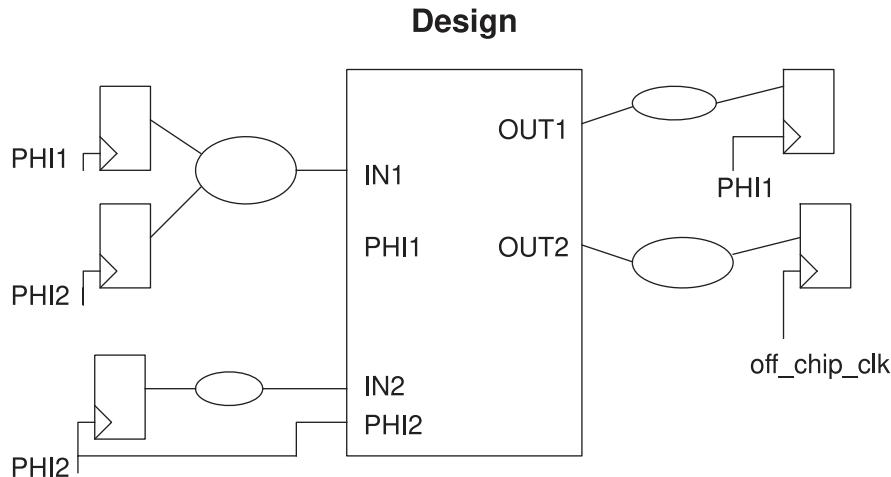
If paths from multiple clocks or edges reach the same port, specify each one using the `-add_delay` option. If you omit the `-add_delay` option, existing data is removed. For example,

```
prompt> set_input_delay 2.3 -clock PHI2 -add_delay {IN1}
```

Example 3

In Figure 83, consider the path to input IN2.

Figure 83 Two-Phase Clocking



Assume that a rising signal on IN2 can occur 4.3 time units after the rising edge of clock PHI2 and a falling signal has a delay of 3.5 units to reach IN2 (the maximum and minimum times are the same in this case). Input delay is defined as

```
prompt> set_input_delay 4.3 -rise -clock PHI2 { IN2 }
prompt> set_input_delay 3.5 -fall -clock PHI2 { IN2 }
```

Now consider the delays leading to input port IN1. Paths from registers are clocked by two different clocks. To fully describe this situation, use the `-add_delay` option to capture input delay information relative to both clocks. Enter

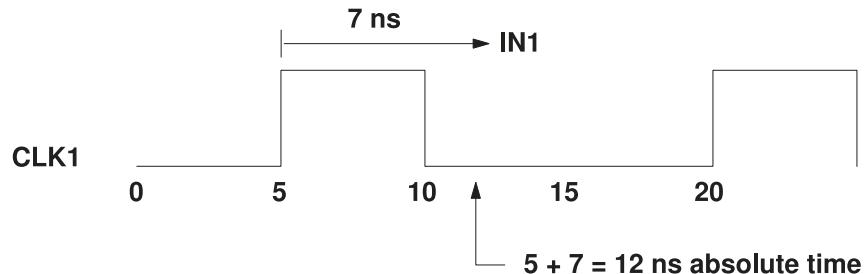
```
prompt> set_input_delay 2.7 -clock PHI1 -add_delay { IN1 }
prompt> set_input_delay 4.2 -clock PHI2 -add_delay { IN1 }
```

Example 4

This example shows a 7-ns input delay defined relative to clock CLK1. The command for defining the delay is

```
prompt> set_input_delay 7 IN1 -clock CLK1
```

Figure 84 Input Delay on CLK1

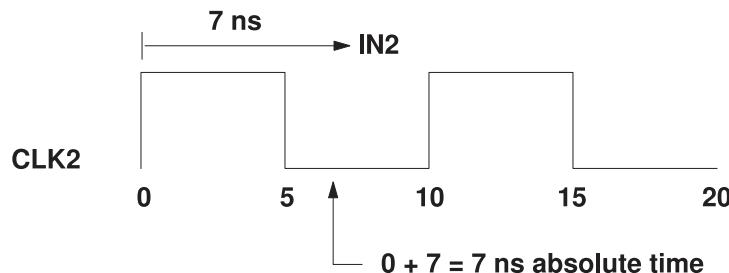


Example 5

This example shows a 7-ns input delay defined relative to clock CLK2. The command for defining the delay is

```
prompt> set_input_delay 7 IN2 -clock CLK2
```

Figure 85 Input Delay on CLK2



If the source of the delay is a level-sensitive latch, use the `-level_sensitive` option. This allows the tool 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 remove input delay information from ports or pins in the current design set using the `set_input_delay` command, use `remove_input_delay`.

Excluding Clocks

The tool considers the input delay on clock source ports or pins as source latency if the clock is propagated. The input delay can be relative to no clock, or to the clock of that source. The source latency value is added to the clock network delay to determine total latency.

A common mistake is to use the following command, which sets delay on all the inputs, including the clock input:

```
prompt> set_input_delay 2 -clock CLK [all_inputs]
```

Instead, use this command:

```
prompt> set_input_delay 2 -clock CLK \  
[remove_from_collection [all_inputs] [get_port CLK]]
```

Use the `set_clock_latency` command with the `-source` option to define the actual source latency, if any.

Reference Pin

The `-reference_pin` option of the `set_input_delay` command lets you specify the input delay relative to a pin in the direct or transitive fanout of a clock rather than relative to the clock itself. For example,

```
prompt> set_input_delay 3.30 -clock CLK \  
-reference_pin u2/z [get_ports d1]
```

The `-reference_pin` option specifies the clock pin or port to which the specified delay is related. If you use this option and propagated clocking is being used, the delay value is related to the arrival time at the specified reference pin, which is clock source latency plus its network latency from the clock source to the reference pin. For an ideal clock network, only source latency is applied.

The pin specified with the `-reference_pin` option should be a leaf pin or port in a clock network. If the `-clock` option is used, the reference pin should be in the direct or transitive fanout of the specified clock source. After you set the input delay or output delay, you can confirm that the reference pin is in the transitive fanout of the specified clock by using the `check_timing` command. If multiple clocks reach the port or pin where you are setting the input delay, and if the `-clock` option is not used, the command considers all the clocks.

The reference pin for a port can be reported with the `report_port -verbose` command.

Output Delays

The `set_output_delay` command specifies output delays. An output delay represents an external timing path from an output port to a register. The maximum output delay value should be equal to the length of the longest path to the register data pin, plus the setup time of the register. The minimum output delay value should be equal to the length of the shortest path to the register data pin, minus the hold time.

The command syntax is:

```
set_output_delay
  delay_value
  [-reference_pin pin_port_name]
  [-clock clock_name [-clock_fall] [-level_sensitive]]
  [-network_latency_included]
  [-source_latency_included]
  [-rise] [-fall]
  [-max] [-min]
  [-add_delay]
  [-group_path group_name]
  port_pin_list
```

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

For basic information about specifying input delays and output delays for timing analysis, see [Input and Output Delays](#).

To remove output delays previously set with 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 removal of output delay values, use the `-clock`, `-clock_fall`, `-min`, `-max`, `-rise`, or `-fall` options.

Example 1

To set an output delay of 4.3 relative to the rising edge of clock PHI1 on port OUT1 (see the figure in [Input Delays](#)):

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

Example 2

If the combinational logic delay from a rising signal on port OUT1 reaches the register in 3.5 time units and there is a 0.3-unit library setup time for that register, the total output delay is 3.8 relative to the clock PHI1 rising edge. The maximum output delay is defined as

```
prompt> set_output_delay 3.8 -rise -clock PHI1 {OUT1}
```

Example 3

If the library hold time is 0.7, the minimum output delay is the minimum path length minus the library hold time ($3.5 - 0.7$). For example, enter

```
prompt> set_output_delay 2.8 -min -rise -clock PHI1 {OUT1}
```

Example 4

In some cases, the relative clock for the output delay does not exist within the subdesign. You can use the `create_clock` command to create a virtual clock that can be used in a `set_output_delay` command. For example, enter

```
prompt> create_clock -period 10 -waveform {3 8} -name off_chip_clk
prompt> set_output_delay 2.5 -clock off_chip_clk {OUT2}
```

Drive Characteristics at Input Ports

You can define the drive capability of the external cell driving each input port. The tool uses this information to calculate the delay for the port and to produce an accurate transition time for calculating cell delays and transition times for the following logic stages.

The `set_driving_cell` command models the external driver at an input port 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 overwrites any setting made by an earlier command. For example, if you use 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.

To obtain information about port settings, use the `report_port` command. This is the command syntax:

```
report_port
[-drive]
[-verbose]
[-physical]
[-only_physical]
[-nosplit]
[-significant_digits digits]
[port_list]
```

By default, the command reports all ports. Specify a port list to restrict the report to only those ports. Use the `-drive` option to get information about the drive capability of the port, the `-physical` option to get the physical location of the port, or `-verbose` to get all types of information about the port.

Setting the Port Driving Cell

The `set_driving_cell` command models the external driver at an input port of the current design as the output of a library cell. The tool can then accurately model the drive capability of the external driver. This is the command syntax:

```
set_driving_cell
[-lib_cell lib_cell_name]
[-library lib]
[-rise] [-fall]
[-min] [-max]
[-pin pin_name]
```

```

[-from_pin from_pin_name]
[-dont_scale]
[-no_design_rule]
[-none]
[-input_transition_rise rtran]
[-input_transition_fall ftran]
[-multiply_by factor]
port_list

```

The `set_driving_cell` command can be used instead of the `set_drive` command to describe driver characteristics an input port. The `set_driving_cell` command works with all delay models. The `set_driving_cell` command removes any corresponding rise or fall drive resistance attributes (from `set_drive`) on the specified ports.

By default, the `set_driving_cell` command takes into account the design rule constraints associated with the specified driving cell, in addition to the drive information used to compute the transition time of the input net. When you specify a driving cell, the design rule constraints are annotated on the input port of the block being synthesized. To use the driving cell only to compute input transition times, use the `-no_design_rule` option to `set_driving_cell`.

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 the driving cell by using the `-input_transition_rise` or the `-input_transition_fall` option. If no input transition is specified, the default is 0.

Table 10 summarizes the port attributes set by the `set_driving_cell` command in Design Compiler.

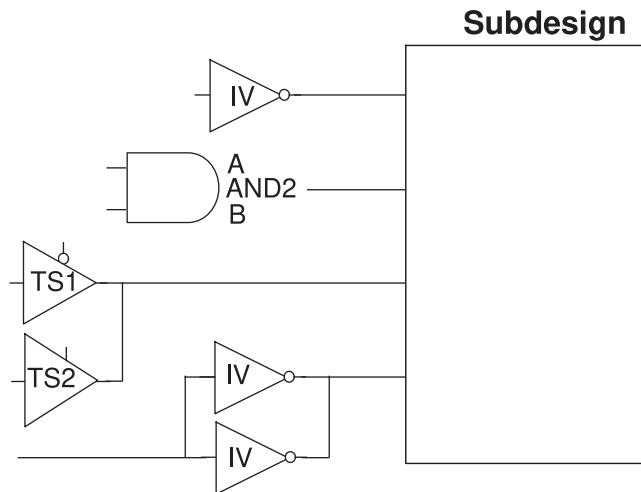
Table 10 Port Attributes Set by the `set_driving_cell` Command in Design Compiler

Attribute name	Attribute type	Value example
<code>driving_cell_rise</code> <code>driving_cell_fall</code>	string	“AN2”
<code>driving_cell_library_rise</code> <code>driving_cell_library_fall</code>	string	“tech_lib”
<code>driving_cell_pin_rise</code> <code>driving_cell_pin_fall</code>	string	“Z”
<code>driving_cell_from_pin_rise</code> <code>driving_cell_from_pin_fall</code>	string	“B”
<code>driving_cell_dont_scale</code>	Boolean	TRUE
<code>driving_cell_multiply_by</code>	float	0.5

Example

[Figure 86](#) shows driving cells set by the following `set_driving_cell` command examples.

Figure 86 Using set_driving_cell



In [Figure 86](#), Port I1 is driven by a single IV cell. Because IV has only one output and one input, define the drive as follows:

```
prompt> set_driving_cell -lib_cell IV {I1}
```

Port I2 is driven by an AND2 cell. If the different arcs of this cell have different transition times, define the drive as follows:

```
prompt> set_driving_cell -lib_cell AND2 -pin Z -from_pin B {I2}
```

Port I3 has two drivers. Assuming that TS1 has the worst rise drive and TS2 has the worst fall drive, define the drive as follows:

```
prompt> set_driving_cell -lib_cell TS1 -rise {I3}
prompt> set_driving_cell -lib_cell TS2 -fall {I3}
```

Port I4 has two parallel drivers. In some technologies, this configuration is valid and reduces transition times by 50 percent. Use the `multiply_by` option, as follows:

```
prompt> set_driving_cell -lib_cell IV -multiply_by 0.5 {I4}
```

Setting the Port Drive Resistance

The `set_drive` command defines the external drive strength as a resistance value for specified input ports. During optimization, the drive of an input port is used to calculate the timing delay to gates driven by that port. The external driver is modeled as the supply voltage connected in series with the specified resistance value.

This is the command syntax:

```
set_drive
  resistance
  [-rise] [-fall]
  [-min] [-max]
  port_list
```

An alternative to using the `set_drive` command is to use `set_driving_cell` or `set_input_transition`. The `set_driving_cell` command is usually the preferred method because it is the most realistic model. The `set_drive` command is useful when you cannot specify a library cell, for example, when the driver is a custom block not in the library. In case of conflict, the command used most recently overrides earlier commands.

The following example sets the rise and fall drives of ports A, B, and C to 2.0

```
prompt> set_drive 2.0 {A B C}
```

The following example sets the drive resistance for all input ports:

```
prompt> set_drive 12.3 [all_inputs]
```

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. The tool uses the specified transition time only in calculating the delays of logic driven by the port. This is the command syntax:

```
set_input_transition
  transition
  [-rise] [-fall]
  [-min] [-max]
  port_list
```

A fixed transition time setting is useful for setting transition times 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.

Removing Drive Information From Ports

The commands shown in [Table 11](#) remove drive information from ports.

Table 11 Commands to Remove Drive Information

To remove this	Use this
Driving cell information from a list of ports	<code>remove_driving_cell</code>
Drive resistance	<code>set_drive 0.0</code>
Input transition	<code>set_input_transition 0.0</code>
Drive data and all user-specified data, such as clocks, input and output delays	<code>reset_design</code>

Port Load Capacitance

To accurately perform timing analysis, the tool needs information about the external load capacitance of nets connected to top-level ports, including pin capacitance and wire capacitance. You can explicitly specify the load capacitance on a port with the `set_load` command. This is the command syntax:

```
set_load value objects
  [-subtract_pin_load]
  [-min]
  [-max]
  [[-pin_load] [-wire_load]]
```

The `-wire_load` and `-pin_load` options are useful when you know the total value of the loads contributed by the external nets. For example, use these options with back-annotated nets or with segmented wire loads where the wire load model is insignificant and you only want the total capacitive load caused by the external net.

If you use both the `-wire_load` and `-pin_load` options, the load value you define is used for both the pin load and wire load of the port. The `-pin_load` option is the default. However, with both the options, you can distinguish between the pin load and wire load portions of the total port load.

Example 1

To specify the external pin capacitance of ports, enter

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

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

Example 2

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

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

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

Ideal Networks

An ideal network is a network of cells, nets, and pins that are exempt from timing updates, timing optimization, and DRC fixing. For objects in an ideal network, the `max_capacitance`, `max_fanout`, and `max_transition` design rules are ignored. As a result, runtime and timing optimization are improved. In addition, the tool does not optimize away the source port or leaf-level pin of the ideal network.

For example, if you define as ideal nets certain high-fanout nets that you intend to synthesize separately, such as scan-enable and reset nets, you can reduce runtime by avoiding unnecessary retiming and unwanted design changes during optimization.

When you specify the source port or leaf-level pin of an ideal network, the nets, cells, and pins in the transitive fanout of this source are treated as ideal objects. Ideal objects have the following properties:

- They are marked as ideal.
- They are not affected by timing updates, delay optimization, or DRC fixing.
- They are assigned ideal timing properties: ideal latency, ideal transition time, and ideal capacitance of zero. You can change the latency and transition values by using the `set_ideal_latency` and `set_ideal_transition` commands, respectively.

The tool disables delay optimization of an ideal network by marking the cells and nets in the network as `dont_touch`. In addition, the tool disables DRC fixing by setting the DRC cost to 0 for the network. The `size_only` attribute is set on the cell that contains or drives the source. This guarantees that the ideal network source is not lost during a compile operation.

Note:

Although clock nets and networks are ideal by default, you should not use the `set_ideal_network` command to modify the properties of these networks. To prevent clock nets from being treated as ideal nets, use the `set_auto_disable_drc_nets -clock false` or `set_propagated_clock` command.

Propagation of the Ideal Network Property

When you specify the source object of an ideal network, all the nets, cells, and pins in the transitive fanout of the source objects are treated as ideal. Any input port or internal pin of the current design can be a source object, except for a pin at a hierarchical boundary.

The tool automatically spreads the ideal network property to the nets, cells, and pins in the transitive fanout of the source object, according to certain propagation rules. The tool propagates the ideal network property across logic gates and hierarchies. Propagation of the ideal network property stops at a sequential cell or a boundary cell (a cell in which the propagation rules are not met). The pins where propagation stops are known as network boundary pins; they are ideal pins.

Propagation of the ideal network property is governed by the following rules:

- A pin is treated as ideal if it is one of the following:
 - A pin specified in the object list of the `set_ideal_network` command
 - A driver pin and its cell is ideal
 - A load pin attached to an ideal net
- A net is treated as ideal if all its driving pins are ideal.
- A combinational cell is treated as ideal if either:
 - All its input pins are ideal, or
 - An input pin is attached to a constant net and all other input pins are ideal. Note that an object with the case analysis attribute is not treated as constant.

Note:

Ideal network propagation can traverse combinational cells, but it stops at sequential cells, even if the sequential cells are connected to ideal clock pins.

If an ideal network overlaps a clock network, the clock timing information, including clock latency and transition values, overrides the ideal timing for the clock portion of the overlapped networks.

Creating Ideal Networks

You use the `set_ideal_network` command to create ideal networks. This is the command syntax:

```
set_ideal_network
  object_list
  [-dont_care_placement]
  [-no_propagate]
```

Specify a list of objects (ports, pins, or nets) as the sources of the ideal network.

If you specify a net, the net's global driver pins or ports are marked as ideal network sources. That is, the ideal network property is applied to the global driver pins or ports of

the specified net. This ensures that the ideal network property is not lost even if the net is optimized away.

To prevent the ideal network from being considered during placement, use the `-dont_care_placement` option.

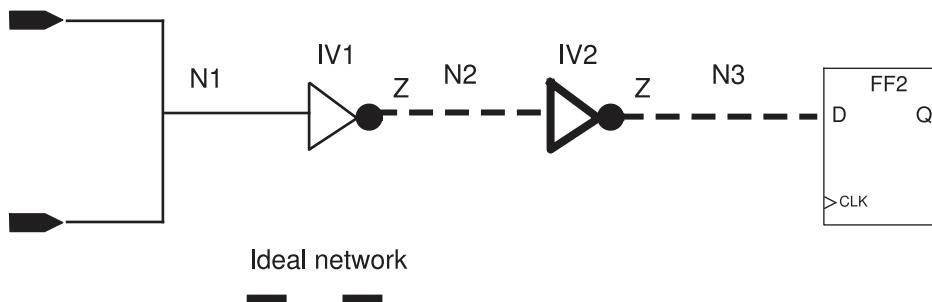
To prevent the ideal network from being propagated through logic gates, use the `-no_propagate` option.

Example 1

Consider the following command and the resulting ideal network in Figure 87.

```
prompt> set_ideal_network [get_pins IV1/Z]
```

Figure 87 Ideal Networks



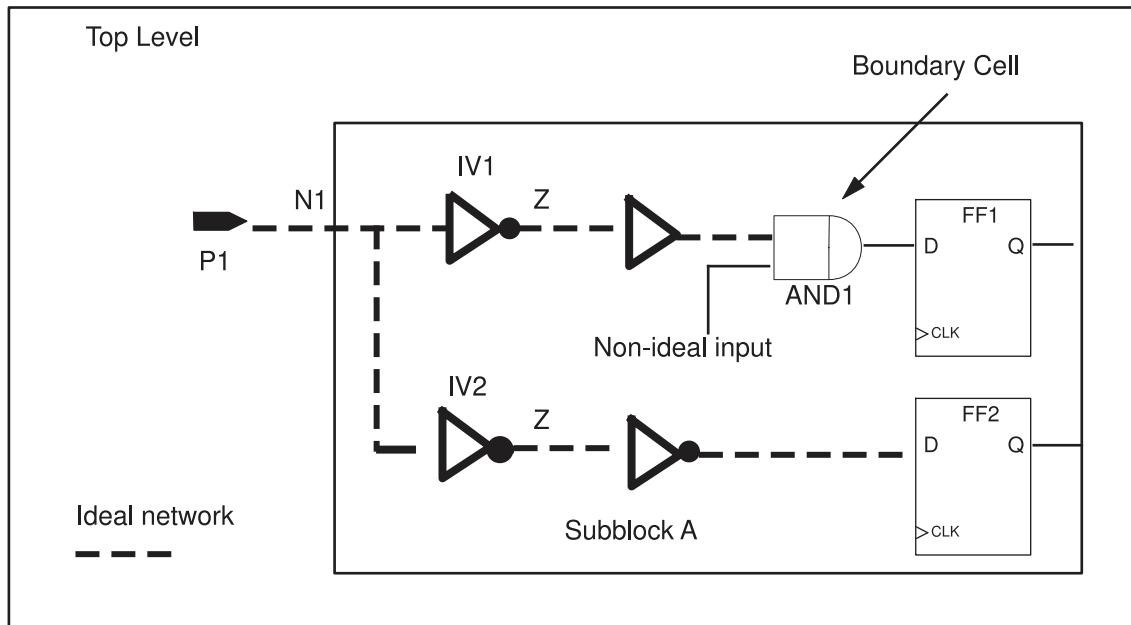
In Figure 87, pin IV1/Z is the source of the ideal network. The ideal network property is set on IV1/Z and propagated along the nets, cells, and pins in the transitive fanout of IV1/Z. In addition, the `dont_touch` attribute is propagated to these nets, cells, and pins. Propagation stops at the sequential cell. In addition, the `size_only` attribute is set on the driver cell IV1 of pin Z.

Example 2

Consider the following command and the resulting ideal network in Figure 88.

```
prompt> set_ideal_network P1
```

Figure 88 Propagation of the Ideal Network



In Figure 88, one of the inputs of the AND gate is not ideal. The ideal network property is set on P1 and propagated along the nets, cells, and pins in the transitive fanout of P1. In addition, the `dont_touch` attribute is propagated to these nets, cells, and pins. The tool propagates the ideal network property across the hierarchical boundary from the top-level block to subblock A. Propagation of the ideal network property stops at the sequential cell FF2 and gate AND1. The `size_only` attribute is not set on any cell because the source P1 of the ideal network is a port.

Removing Ideal Networks

To remove ideal networks and restore objects in the ideal network to their initial, nonideal state, use the following commands:

- `remove_ideal_network`

This command restores the cells, nets, and pins in the ideal network to their initial, nonideal state.

- `remove_ideal_net [get_nets net]`

This command restores the ideal net set by the `set_ideal_network -no_propagate` command to its initial, nonideal state.

- `remove_attribute [get_nets net] ideal_net`

This command restores the ideal net set by the `set_ideal_net` command to its initial, nonideal state.

Reporting Ideal Networks

Use the following commands to get more information about ideal networks:

- `remove_ideal_network`

This command restores the cells, nets, and pins in the ideal network to their initial, nonideal state.

- `report_attribute`

This command reports attributes associated with cells, nets, and pins. If you use this command to report attributes set on an ideal net, it reports attributes set by both the `set_ideal_net` command and the `set_ideal_network -no_propagate` command.

- `report_net`

This command reports net information; ideal nets are indicated by “I”.

Commands such as `report_timing`, `report_cell`, and `report_net` indicate the ideal network property propagated to nonsource objects of an ideal network, as well as the source ports and pins of the network. The `report_attribute` command and the `get_attribute` command, however, indicate the ideal network property only for the source ports and pins of the network. The propagated attribute is not shown.

Retrieving Ideal Objects

Use the following commands to retrieve ideal objects.

- `get_nets -filter "ideal_net == true"`

This command returns ideal nets set by the `set_ideal_net` command or the `set_ideal_network -no_propagate` command.

- `get_pins -filter "ideal_network_source == true"`

This command returns pins that are ideal network sources. Additionally, if you use the `ideal_network_options == 1` filter, the command returns only the ideal network source pins that were set by the `set_ideal_network -no_propagate` command.

- `get_ports -filter "ideal_network_source == true"`

This command returns ports that are ideal network sources. Additionally, if you use the `ideal_network_options == 1` filter, the command returns only the ideal network source ports that were set by the `set_ideal_network -no_propagate` command.

Setting Ideal Latency and Ideal Transition Time

The default latency and transition values for ideal networks and ideal nets is zero. You can override these defaults by using the following commands:

- `set_ideal_latency`
- `set_ideal_transition`

Note:

The timing of ideal networks and ideal nets is updated whenever you execute either of these commands.

You can use these commands to set the ideal latency and ideal transition on the source pin of an ideal net or network and on any nonsource pin of an ideal network. The specified values override any library cell values or net delay values. For ideal networks, the ideal latency and transition values are propagated from the source pins to the network boundary pins.

The total ideal latency at any given point of an ideal network is the sum of the source pin ideal latency and all the ideal latencies of the leaf cell pins along the path to the given point. The ideal transition values specified at the various source and leaf cell pins are independent and noncumulative. The transition for an unspecified input pin is the ideal transition of the closest pin with a specified ideal transition value. This rule applies to boundary pins as well.

The `set_input_delay` command is applicable to ideal networks. This delay is treated as the off-block latency or source latency.

You can remove ideal latency and ideal transition values by using the following commands:

- `remove_ideal_latency`
- `remove_ideal_transition`

These commands remove the ideal latency and ideal transition values from the specified source pins. In the case of ideal networks, the command also removes the ideal latency and transition values from any nonsource pins on which they were set and from any pins to which the ideal property was propagated. The default of zero is restored to the ideal pins.

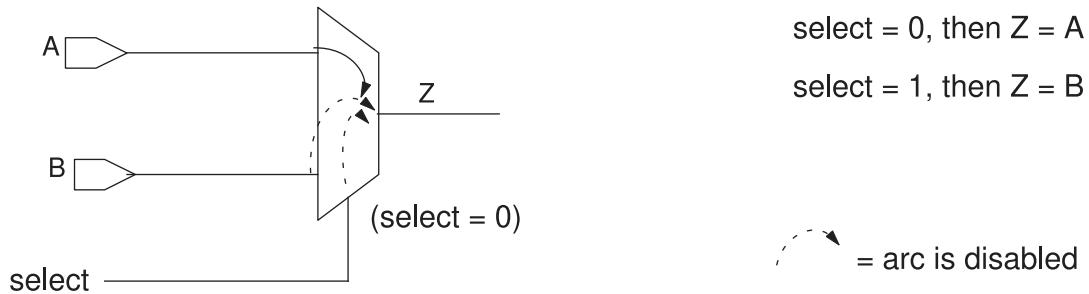
Case Analysis

Case analysis lets you perform timing analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design. Setting a case analysis with a logic constant propagates the constant forward through the design, then disables paths where the constant is propagated. Setting a case analysis with a logic transition, either rising or falling, eliminates certain paths by limiting the transitions considered during analysis.

Example 1

In the multiplexer in [Figure 89](#), logic 0 is set on the select control signal. This disables the arc from B to Z because the constant blocks the data from B to Z.

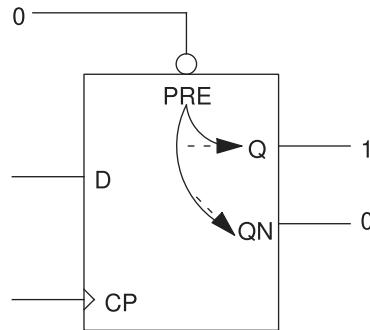
Figure 89 Constant Blocking of Data



Example 2

When case analysis propagates a constant to a sequential element's asynchronous preset or clear pin, the sequential cell outputs also propagate the constant 1 or 0, as shown in [Figure 90](#).

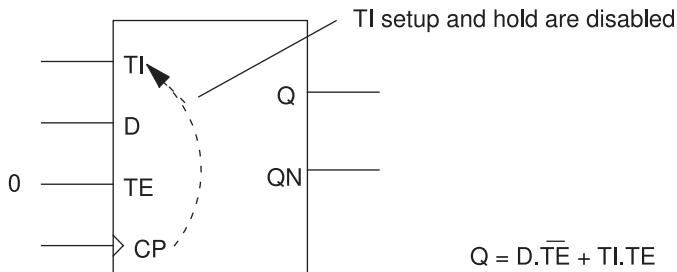
Figure 90 Constant Propagation Through an Asynchronous Input



Example 3

When your design uses scan flip-flops, case analysis is useful for disabling the scan chain. To do this, set the scan mode control pin to the constant value that disables the test mode. The tool propagates the test-mode constant value to the scan-mode pin of each scan flip-flop. Case analysis determines which timing arcs to disable, as shown in [Figure 91](#).

Figure 91 Case Analysis Disabling Timing Checks



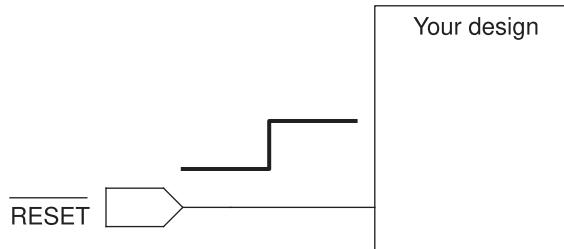
For example, to set case analysis with a 0 value on the `SCAN_MODE` input port, enter

```
prompt> set_case_analysis 0 [get_ports SCAN_MODE]
```

Example 4

Setting a case analysis for a transition can be useful for analyzing the behavior of a circuit for a specific situation. For example, in the circuit shown in [Figure 92](#), a rising-edge transition on the reset input brings the device out of the reset mode.

Figure 92 Case Analysis for a Rising Edge



To analyze the timing of the circuit coming out of reset mode, you are only concerned about rising edges on the reset input, not logic 0, logic 1, or falling edges. To set case analysis in this situation, enter the following syntax:

```
prompt> set_case_analysis rising [get_ports RESET]
```

You can also specify logic constants by using the `set_logic_one` and `set_logic_zero` commands. Case analysis treats these logic constants as case analysis constants for the purposes of timing and costing the design. Also, the tool ties unconnected pins to constants, and these constants can be used to disable arcs based on case analysis.

The tool performs constant propagation if the `set_case_analysis` command has been used or if the `case_analysis_with_logic_constants` variable has been set to true. You can disable propagation of case analysis constants by setting the `disable_case_analysis` variable to true.

To specify whether case analysis values are propagated across sequential cells, set the `case_analysis_sequential_propagation` variable to `always` or `never`. The default is `never`.

Reporting Case Analysis

The `report_case_analysis` command reports the case analysis values. For example,

```
prompt> report_case_analysis
...
Pin name          User case analysis value
-----
test_port          0
U1/U2/core/WR      1
```

Using the `-all` option of the command reports both the built-in logic constant pins and the case analysis values set on the design. For example,

```
prompt> report_case_analysis -all
...
Pin name          User case analysis value
-----
test_port          0
U1/U2/core/WR      1

Pin name          User case analysis value
-----
HRS                1
ALARM              1
MINS               1
```

The `report_disable_timing` command reports timing arcs that have been disabled by various causes, including case analysis. For example,

```
prompt> report_disable_timing
...
Cell or Port      From      To      Flag  Reason
-----
U5                 A         Z       C     A = 0
```

Removing Case Analysis

The `remove_case_analysis` command removes case analysis values. For example, to remove case analysis values from the `test_port` input port, enter the following syntax:

```
prompt> remove_case_analysis [get_ports test_port]
```

To suppress propagating logic constants (including those set with case analysis and pins tied to logic high or low), set the `disable_case_analysis` variable to `true`.

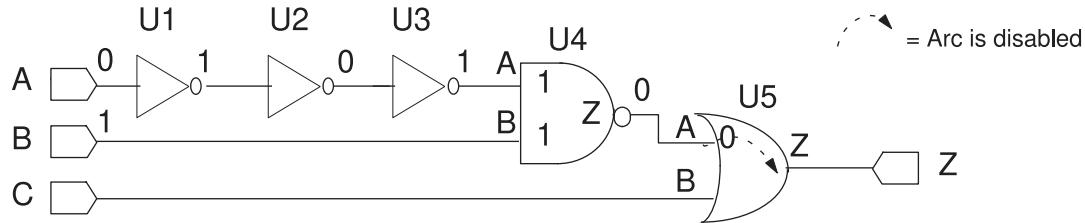
Constant Propagation Log File

When the tool disables a timing arc, knowing the exact origin of the constant value propagated to a given pin of the design might be difficult. If you would like the tool to keep a log file containing the constant propagation information so that you can later track down the point of origin, set the `case_analysis_log_file` variable to a file name. The constant propagation process is logged to the named file.

Example

This command sequence sets case analysis and reports the disabled timing arcs to the log file `my_design_cnst.log` for the example circuit.

```
prompt> set_case_analysis 0 {get_port "A"}
prompt> set_case_analysis 1 {get_port "B"}
prompt> set_case_analysis_log_file my_design_cnst.log
```



In the example, the only arc that the tool disables is the arc in U5 from U5/A to U5/Z. U5/A is at constant value 0 and no constant is propagated to U5/Z. The constant propagation log file `my_design_cnst.log` looks like this:

```
*****
Report : case_analysis propagation
Design : my_design
Version: ...
*****

1.1 Forward propagation on NET pins (level 1)
-----
Propagating logic constant '0' from pin 'A' on net 'A':
> pin 'U1/A' is at logic '0'
Propagation of logic constant '1' from pin 'B' on net 'B':
> pin 'U4/B' is at logic '1'

1.2 Forward propagation through CELLS (level 1)
-----
Cell 'U1' (libcell 'IV') :
  input pin U1/A is at logic '0'
  > output pin 'U1/Z' is at logic '1'

...
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
```

Usage Example

The following script example shows how to use case analysis in the Design Compiler tool.

```
read_verilog design_B_rtl.v
current_design design_B
link
source design_B_constraints.con
set_case_analysis 1 [get_ports {...}]
set_case_analysis 0 [get_ports {...}]
set_disable_case_analysis false
set case_analysis_log_file "design_B_case_analysis.log"
compile
report_case_analysis -all
report_disable_timing
report_timing
...
remove_case_analysis [get_ports {...}]
...
```

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 the tool analyzes while that mode is active. The timing arcs associated with inactive modes are not analyzed. In the absence of any mode settings, all modes are active and all timing arcs are analyzed.

You can set cell modes directly on cell instances with the `set_mode` command. For example, `set_mode READ {U1}` sets the READ mode on cell instance U1. Alternatively, you can enforce an operating mode using case analysis. For example, if a cell U1 has a cell mode called READ which is defined to be active when input RW is 0, setting or propagating a case analysis value of 0 on the U1/RW input activates the READ mode and deactivates all other modes for that cell. The `set_case_analysis 0 [get_pins U1/RW]` command sets a logic 0 directly on the RW pin of U1.

Mode Groups

A library cell with modes can have one or more mode groups defined. Within each group, there are two possible states: all modes enabled (the default), or exactly one mode enabled with all other modes disabled. Often there is only one mode group.

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, the tool makes the corresponding mutually exclusive modes inactive.

For example, specifying the RAM block for the read mode implicitly specifies that the write mode is inactive, irrespective of any setting for the transparent and latched modes. Similarly, specifying the RAM block for the latched mode implies that transparent mode is inactive, irrespective of the read and write mode setting. The two mode groups (read/write and transparent/latched) are independent.

By default, all cell modes are enabled in each cell instance. Using the `set_mode` command, you can set each cell mode group to have one mode enabled and all other modes disabled. When a mode is disabled, all timing arcs in that cell mapped to that mode are disabled.

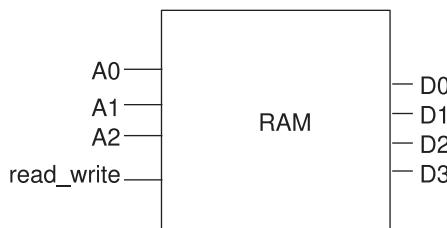
Setting Modes Using Case Analysis

Some library cells and timing models have conditional modes defined in them. This means that a mode selection is invoked by the logical conditions at the cell inputs. For example, the read mode of a RAM cell could be invoked by a logic 0 on the read/write input pin.

When you set the controlling logic value on the input pin using case analysis or when case analysis values are propagated to that pin, the cell is implicitly placed into the appropriate analysis mode.

Example

In this RAM example, the address input pins are A0, A1, and A2; and the data I/O pins are D0, D1, D2, and D3. The RAM cell can be operated in read and write modes.



The `read_write` pin of the cell controls the read/write mode of the RAM. If the RAM is modeled with the following mode and condition association in the Liberty modeling language, you can perform the read and write mode analysis by setting logic values on the `read_write` input pin:

```
cell(RAM) {
    mode_definition(read_write) {
        mode_value(read) {

```

```

        when : "read_write";
        sdf_cond : "read_write == 0";
    }
    mode_value(write) {
        when : "read_write";
        sdf_cond : "read_write == 1";
    }
}
...
}

```

To enable the read mode in the tool, enter

```
prompt> set_case_analysis 0 [get_ports read_write]
```

To enable the write mode in the tool, enter

```
prompt> set_case_analysis 1 [get_ports read_write]
```

Setting or propagating a logic value to the read_write pin implicitly selects the corresponding mode and disables the other mode. Only the timing arcs associated with the active mode are used in the timing analysis.

When no modes in a mode group are selected by case analysis or other mode selection methods, all of the modes are enabled. The default mode is enabled if no mode is selected.

Setting Modes Directly on Cells

To specify the active cell modes directly for cell instances in the design, use the following command:

```
prompt> set_mode cell_mode_list instance_list
```

For example, to set the U1/U2/core cell to read mode, enter

```
prompt> set_mode read U1/U2/core
```

This makes the read mode active and the other modes inactive for the U1/U2/core cell.

In the command, the cell mode list is a list of modes to be made active, no more than one mode per mode group. If the cell has only one mode group, you can list only one mode to be made active. The command must also specify one or more instances in the design to be placed into the specified mode.

To cancel the mode selection and make all modes active for the U1/U2/core cell, enter

```
prompt> reset_mode U1/U2/core
```

Reporting Modes

The `report_mode` command reports on modes that have been defined or set in the design. For example, to get a cell mode report:

```
prompt> report_mode
...

```

Cell	Mode (Group)	Status
Uram1/core (RAM2_core)	read(rw)	ENABLED
	write(rw)	ENABLED
Uram2/core (RAM2_core)	read(rw)	ENABLED
	write(rw)	ENABLED

The report shows the name of each cell instance with a mode setting, the possible mode settings and group names for the cell, the current status of each mode (enabled or disabled).

Wire Load Models

For timing analysis performed before placement and routing, Design Compiler must estimate the wire delays. The simplest estimation method is the wire load model, which gets a rough value for the total wire capacitance, based on the size of the chip and the fanout of the net. Larger chip sizes and larger fanouts are assumed to result in longer wires and more resistance and capacitance. Wire load models are not used in IC Compiler because accurate wire information is available from the layout database.

For logic synthesis, you should use Design Compiler with topographical technology, if available, because it produces better results than wire load models. However, if you are not using Design Compiler in topographical mode, you can use wire load models to estimate the capacitance, resistance, and area of nets before floorplanning or layout. The wire load models provided in the logic library define the fanout-to-length relationships.

Note:

Wire load models are used only with Design Compiler running in standard mode, not topographical mode. Design Compiler with topographical technology and IC Compiler have better wire estimation methods, so wire load models are not used with these tools.

With wire load models, Design Compiler estimates the wire lengths of pin-to-pin connections based on fanout count, and then uses those estimates to calculate the effects of cell placement and wire routing. Fanout is the total number of pins on the net, excluding the driver pin.

The wire load model is defined in the library specification. In Liberty library syntax, the `wire_load` group defines the wire loads. For more information, see the Library Compiler documentation.

Design Compiler determines which wire load model to use for a design according to the wire load model you define with the `set_wire_load_model` command, the wire load indicated by the logic library's `wire_load_selection` group, or the wire load model identified by the `default_wire_load` attribute in the library, in that order. If none of these items are defined, no wire load model is used, and the net resistance, capacitance, and delay values are zero.

Net Capacitance, Resistance, and Area Calculation

To calculate the capacitance of a net, Design Compiler performs the following steps:

1. Determines the fanout of the net.
2. Looks up the length in the wire load model.
3. Calculates the capacitance by multiplying the length by the capacitance coefficient in the wire load model.

Example

Suppose that a wire load model is defined as follows in the library:

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

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

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

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

It calculates the lumped net capacitance by multiplying of the estimated wire length by the capacitance factor 2.0. For a fanout value of 4, the last `fanout_length` defines the wire length as 4.4, resulting in a calculated lumped net capacitance of

$2.0 * 4.4 = 8.8$

It calculates the total net resistance by multiplying the estimated wire length by the resistance factor, yielding a value of

$4.4 * 100.0 = 440.0$

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

Design Compiler calculates the net area by multiplying the estimated wire length by the area factor, yielding a value of

$4.4 * .5 = 2.2$

Design Compiler uses the net area to measure the impact of different optimization possibilities on the total design area of the chip. An accurate net area estimate guides Design Compiler in selecting the best optimization for minimizing area.

Design Compiler uses interpolation between and extrapolation beyond the library-defined wire load values.

Choosing a Wire Load Model

The choice of wire load model depends on how the design is ultimately implemented. Hierarchy restrictions imposed on layout results in more accurate wire load estimates.

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

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

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

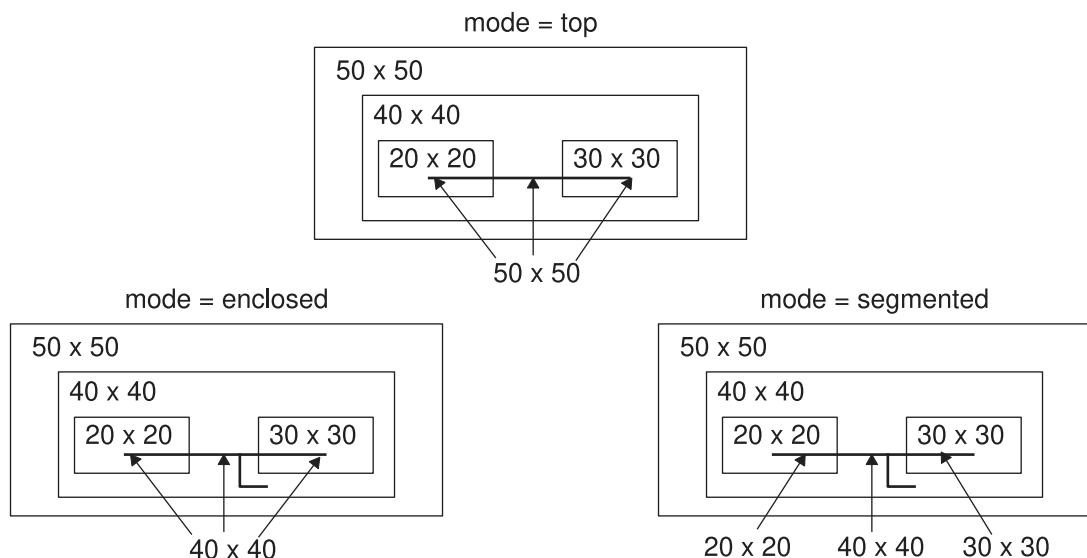
Wire Load Model Modes

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

```
set_wire_load_mode top | enclosed | segmented
```

[Figure 93](#) shows how the span of a net determines the wire load models used in each of the available mode settings.

Figure 93 Wire Load Model Modes



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

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

The “segmented” setting causes the sum of the cell areas of the designs containing the net to be used. Nets crossing hierarchical boundaries are divided into segments (part of the net is outside the module). Each net segment is estimated from the wire load model

of the design containing the segment. If the design enclosing a segment has no wire load model, the design hierarchy is traversed upward until a wire load model is found.

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

Examples

To set the model my_model on a design, enter

```
prompt> set_wire_load_model -name my_model
```

If the current library does not define a default mode, Design Compiler searches the libraries specified by the `link_library` variable.

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

```
prompt> set_wire_load_mode top
```

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

```
prompt> set_wire_load_mode enclosed
```

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

```
prompt> set_wire_load_mode segmented
```

Note:

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

Setting a Wire Load Model

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

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

If you set a wire load model on a design, automatic wire load selection is disabled for that design. In the presence of physical hierarchy, wire loads set on subdesigns are ignored.

The `set_wire_load_model` command does not assume a default wire load mode. You use the `set_wire_load_mode` command to explicitly set the wire load mode before you use the `set_wire_load_model` command.

The syntax is

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

Examples

```
prompt> set_wire_load_model -name "60x60"
Using wire_load model '60x60' found in library 'my_lib'.
```

To remove the `wire_load_model` attribute, use the `remove_wire_load_model` command. For additional information, see the man pages.

Local Link Library Usage

The `set_wire_load_model` command searches the design's local link library before searching the system link library.

During compilation or translation, Design Compiler searches the link library for the default values. During the `update_timing` or `characterize` commands, Design Compiler searches the first library on the link path for the default values. If Design Compiler cannot find them, it uses the following values:

```
wire_load_model:          (none)
wire_load_mode:           TOP
operating_conditions:    (system defaults)
```

Use the following commands to indicate a library name:

```
set_wire_load_model -name model_name -library library_name
set_operating_conditions -library library_name
```

The *library_name* must either be a fully defined file name or exist in memory. If the requested information (wire load or operating conditions) is not in *library_name*, Design Compiler searches the libraries in the link path. When Design Compiler finds the information, it reports the library name. Design Compiler never searches the link library unless you define it as *library_name*.

For more information about link libraries, see the *Design Compiler User Guide*.

Setting a Wire Load Selection Group

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

The `set_wire_load_selection_group` command does not assume a default wire load mode. You can use the `set_wire_load_mode` command to explicitly set the wire load mode to enclosed. This is the command syntax:

```
set_wire_load_selection_group
[-library lib_name]
[-min] [-max]
```

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

Wire Load Models for Hierarchical Cells

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

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

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

Selecting the Wire Load Model Automatically

If no wire load model is defined for a design, Design Compiler automatically selects a default wire load model based on area. The cell area used to select a wire load model for a net is determined by the wire load mode. Design Compiler searches the first library in

the link path (or the first library in the design's `local_link_library`) for a default wire load.

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

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

Note:

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

Defining the Minimum Block Size

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

Example

```
wire_load("05x05") {
    resistance: 0;
    capacitance: 1;
    area: 0;
    slope: 0.186;
    fanout_length(1,0.39);
}
wire_load("10x10") {
    resistance: 0;
    capacitance: 1;
    area: 0;
    slope: 0.311;
    fanout_length(1,0.53);
}
default_wire_load: "15x15";
wire_load_selection() {
    wire_load_from_area (0,100,"05x05");
    wire_load_from_area (150,200,"10x10");
}
```

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

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

The wire load is automatically selected before and during a timing operation and design modification operation such as with the `update_timing`, `report_timing`, or `compile` command.

Design Compiler selects a wire load model before running the `compile` or `translate` command. If the design is not mapped, Design Compiler uses either the default or no wire load model.

Design Compiler resets the wire load model after structuring or mapping and before gate-level optimization, before buffering, and at the end of optimization.

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

Reporting Wire Load Models

You can get information about wire load models for designs, ports, or libraries, using the commands listed in [Table 12](#).

Table 12 Commands for Reporting Wire Load Information

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

The `report_wire_load` command provides a list of all wire load model characteristics set on a design. This command reports the characteristics of a specific wire load model set on a design or library. By default, all wire load models set on the current design are reported.

The `report_wire_load` command is similar to the `report_lib` command, but it also reports the statistics of how the wire load model was created. This is the command syntax:

```
report_wire_load
[-design design_name]
[-name model_name]
[-libraries]
[-nosplit]
```

The `report_wire_load -libraries` command generates a report similar to the following:

```
Wire Loads Report
...
Wire load model: counter_wl
Location : counter (design)
Resistance : 100
Capacitance : 1
Area : 0
Slope : 1.19097
          Average   Standard % Standard
Fanout    Length    Points      Cap Deviation Deviation
-----
 1       1.00      20        1.00     0.00      0.00
 2       2.31      3         2.31     0.27     11.78
 3       3.50      3         3.50     0.24      6.73
 4       4.69      6         4.69     0.45      9.62
 5       5.89      1         5.89     0.11      1.95
-----
Weighted Average Standard Deviation: 3.49

Wire load model: top_cluster/cluster_1_wl
Location : top_cluster/cluster_1 (cluster)
Resistance : 100
Capacitance : 1
Area : 0
Slope : 1
          Average   Standard % Standard
Fanout    Length    Points      Cap Deviation Deviation
-----
 1       1.00      3         1.00     0.00      0.00
-----
Weighted Average Standard Deviation: 0.00

Wire load model: 10x10
Location : my_lib (library)
Resistance : 100
Capacitance : 1
Area : 0
Slope : 0.311
          Average   Standard % Standard
Fanout    Length    Points      Cap Deviation Deviation
```

1 0.53

The report contains the following types of information:

Location

Specifies where the wire load model was found. If a wire load model is on a design and in a library, the design location is indicated.

Resistance, Capacitance, and Area

Reports the values per unit length.

Slope

Reports the slope of the wire load model after the last fanout value.

Points

Specifies the number of points used for a fanout. A high number of points implies that a good population was used to create a statistically more accurate fanout length estimate. If the model was not created from back-annotation, only fanout and length are reported.

Average Cap

Lists the average capacitance, which is directly proportional to length. This is the average capacitance back-annotated and modified by smoothing and trimming.

Standard Deviation

Reports the average difference between each point and the average.

Timing Loops

If a timing loop is discovered during the compilation or timing calculation, the tool displays the message

Information: Breaking a timing loop by disabling timing arcs between pin 'name1' and pin 'name2'.

Timing loops must be disabled ("broken") to time the design. After optimization or timing, automatically disabled arcs are restored.

Commands that can initiate automatic breaking of timing loops include the following:

- `compile`
- `check_timing`
- `update_timing`

- `report_timing`
- `report_disable_timing`

After any of these commands completes its tasks, the disabled arcs are enabled. Thus, in a typical session, feedback loops can be disabled and re-enabled several times.

Breaking Feedback Loops Manually

To report combinational feedback loops, use the `report_timing` command. To manually disable timing on parts of a design to break a combinational feedback loop, use the `set_disable_timing` command. Disabled arcs remain disabled until you remove the `disable_timing` attribute or use the `reset_design` command.

The two types of timing arcs are cell arcs and net arcs. Cell arcs describe the cell's internal pin-to-pin timing and are defined in a logic library cell (component) description. Disabling a cell arc is the same as removing a cell description from the logic library. Net arcs are implicitly defined by connections between cells. Each driver pin has a net arc to each load pin of a net. Disabling a net arc is the same as breaking the connection between a net driver pin and a load pin.

Use the `set_disable_timing` command to break a timing path at the defined cell or pin arcs. The syntax is

```
set_disable_timing object_list
  [-from pin_name -to pin_name]
  [-restore]
```

When you use `set_disable_timing` for pins, the pins and their nets are saved during optimization. This might mean that Design Compiler cannot perform some transformations and optimizations and can lower the quality of results.

Example

```
prompt> set_disable_timing CELL31
prompt> set_disable_timing {U33 U37/A U71/z}
prompt> set_disable_timing {U17} -from A2 -to ZN
```

To restore previously disabled timing arcs, use the `set_disable_timing -restore` command or the `reset_design` command. For example, enter

```
prompt> set_disable_timing -restore {U3 U71/z}
```

The `report_design` command lists the cells and pins whose timing arcs are disabled.

```
prompt> report_design
...
Disabled Timing Arcs:
  Object          Name           From Pin      To Pin
```

```
-----
Cell          U33
Pin           U71/Z
Cell          U17          A2          ZN
```

The `report_disable_timing` command reports the disabled timing arcs in the design, including those disabled by case analysis, by disabling of false net-arcs, by conditional arcs (arcs that have a `when` statement defined in the library), by loop breaking, and by using the `set_disable_timing` command.

To produce a report about disabled timing arcs, use the following command:

```
prompt> report_disable_timing
...
Flags :      c  case-analysis
              f  false net-arc
              C Conditional arc
              l  loop breaking
              u  user-defined

Cell or Port   From   To    Sense          Flag    Reason
-----
U1/U1/reg[0]   E      D    hold_clk_rise  c      D = 0
U1/U1/reg[0]   E      D    setup_clk_rise  c      D = 0
U1/U1/reg[0]   E      E    clock_pulse_width_high  c      D = 0
```

6

Back-Annotation

Back-annotation is the process of reading resistance, capacitance, and delay values from an external file into the tool for timing analysis. Using back-annotation, you can more accurately analyze the circuit timing in the tool after each phase of physical design (floorplanning, placement, global routing, and detail routing).

This chapter has the following sections:

- [Back-Annotating Delays and Timing Checks](#)
 - [Setting Net Load](#)
 - [Setting Net Resistance](#)
 - [Setting Pi Model Capacitance and Resistance](#)
 - [Back-Annotating Detailed Parasitics](#)
 - [RTL Load Annotation With Wire Load Modeling](#)
-

Back-Annotating Delays and Timing Checks

For initial static timing analysis, Design Compiler can estimate net delays based on topographical technology or wire load models. Actual delays depend on the physical placement and routing of the cells and nets in the design.

A floorplanner or router can provide more detailed and accurate delay information, which you can provide to Design Compiler for a more accurate analysis. This process is called delay back-annotation. Back-annotated information often is provided in a Standard Delay Format (SDF) file.

You can read SDF back-annotated delay information in these ways:

- Read the delays and timing checks from an SDF file by using the `read_sdf` command.
- Annotate delays and timing checks from the command line without using an SDF file by using the `set_annotated_delay` command or `set_annotated_check` command. The `set_annotated_delay` command sets the delay values for specified nets and cells in the current design. The `set_annotated_check` command sets an annotated timing check between two or more pins.

Generally, back-annotating from an SDF file can be significantly faster than using the `set_annotated_delay` and `set_annotated_check` commands.

The `set_annotated_delay` and `read_sdf` commands do not perform an exhaustive input validity check; some checking is done during the next timing update. Therefore, to verify that all back-annotation is correct, you can run `update_timing` after you have run the `set_annotated_delay` or `read_sdf` command.

Delay Calculations

Before you use the `set_annotated_delay` and `read_sdf` commands, you need to know how the tool reads load delay. Design Compiler uses the following delay definitions:

Cell delay

The delay between a state transition on an input pin and the resulting state transition on the output pin of a gate.

Load delay

The amount of cell delay introduced by the load of the net being driven, calculated as the loaded cell delay minus what the delay would have been if the net had a total capacitance of zero. Load delay is also known as extra source gate delay.

Connect delay (or net delay)

The delay between the output pin changing state and a subsequent input pin changing state. This is the time-of-flight delay on the net.

Design Compiler correctly interprets SDF files with IOPATH and INTERCONNECT delays as

```
IOPATH = cell delay + load delay
INTERCONNECT = connect delay
```

However, some layout tools define the delays as

```
IOPATH = cell delay - load delay
INTERCONNECT = connect delay + load delay
```

Back-Annotating Timing Information From an SDF File

The `read_sdf` command reads leaf cell net and cell delay values from an SDF version v1.0, v2.0, or v2.1 timing file and annotates the values to the current design. The command syntax is:

```
read_sdf
[-load_delay net | cell]
```

```

[-path path_name]
[-min_type sdf_min | sdf_typ | sdf_max]
[-max_type sdf_min | sdf_typ | sdf_max]
[-worst]
[-min_file min_sdf_file_name]
[-max_file max_sdf_file_name]
sdf_file_name

```

Instance-specific pin-to-pin cell and net delays greater than zero are read from the timing file and annotated on the current design.

Instance Names

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, `design_name` must use VHDL naming conventions. To modify design names, use the `change_names` command.

Cell and Pin Names

Cell and pin names in the SDF file and the current design must be the same. Object renaming does not occur when an SDF file is being read and loaded. If the names do not match, an error message similar to the following appears:

```
Error: Pin 'B1/C1'/'INC1' could not be found. (SDFN-10)
```

DESIGN Field Names

The DESIGN field in the SDF file must have the same name as the subdesign. If the names do not match, an error message similar to the following appears:

```
Error: The SDF file contains delays for the design 'SYSTEM',
they cannot be annotated on design 'BAR'. (SDFN-4)
```

Supported SDF Constructs

Supported constructs are INTERCONNECT and PORT for net delay; IOPATH and COND for cell delay; and SETUP, and HOLD for timing checks. Also, the constructs RECOVERY, REMOVAL, and RETAIN are supported.

The DEVICE construct is parsed but ignored for compilation. (DEVICE is used by VSS.)

Note:

Design Compiler reads only absolute delays with SDF. Incremental delays are not read.

SDF files are case-sensitive. The DIVIDER (hierarchy divider), TIMESCALE (time unit), and DESIGN (design name) constructs are read from the SDF header.

The hierarchy divider specified in the DIVIDER construct must be either “.” or “/”.

The `read_sdf` command scales the timing data from the SDF time unit, as defined in the TIMESCALE construct, to the unit defined in the library. If no time unit is defined in the library, the default time unit is 1 ns.

If the delay statement in the SDF file has two or more timing expressions, the first two expressions are read as the rise and fall values. If the delay statement has only one timing expression, `read_sdf` assumes that the rise and fall times are the same.

SDF does not support internal pins, so the `read_sdf` command ignores timing to internal pins.

Examples

- From this delay statement with one expression,

```
(INTERCONNECT A1/Z A2/B (2:3:4))
```

(2:3:4) is read as rise delay and fall delay.

- From this delay statement with two expressions,

```
(INTERCONNECT A1/Z A2/B (2:3:4) (1:1:2))
```

(2:3:4) is read as rise delay and (1:1:2) is read as fall delay.

- From this delay expression, no rise delay is read and (2:3:4) is read as fall delay.

```
(INTERCONNECT A1/Z A2/B () (2:3:4))
```

- From this delay statement with more than two expressions,

```
(INTERCONNECT A1/Z A2/B (2:3:4) (1:1:2) (4:5:6))
```

(2:3:4) is read as rise delay; (1:1:2) is read as fall delay; and (4:5:6) is not read, as this describes 0-to-Z and 1-to-Z transitions.

Design Compiler does not support all three-states when reading SDF. Only rise (01 and Z1) and fall (10 and Z0) delays are supported; 1Z and 0Z are not supported.

[Table 13](#) summarizes the SDF version v2.1 constructs used by the `read_sdf` command.

Table 13 SDF Constructs Used by the read_sdf Command

SDF construct	SDF 2.1 support
DELAYFILE	Parsed, not used.

Table 13 SDF Constructs Used by the read_sdf Command (Continued)

SDF construct	SDF 2.1 support
SDFVERSION	Used. The SDFVERSION entry is mandatory for SDF 1.0 as well as SDF 2.1 files. The Design Compiler SDF reader parses this entry and, depending on the version number read, automatically invokes the SDF 1.0 parser or the SDF 2.1 parser.
DESIGN	Used. The design name must be the name of the current instance.
DATE	Parsed, not used.
VENDOR	Parsed, not used.
PROGRAM	Parsed, not used.
VERSION	Parsed, not used.
DIVIDER	Used.
VOLTAGE	Parsed, not used.
PROCESS	Parsed, not used.
TEMPERATURE	Parsed, not used.
TIMESCALE	Used.
CELL	Used.
CELLTYPE	Used.
CORRELATION	Parsed, not used.
INSTANCE	Used.
DELAY	Used.
TIMINGCHECK	Used.
ABSOLUTE	Used.
INCREMENT	Not used.
PATHPULSE	Parsed, not used.
GLOBALPATHPULSE	Parsed, not used.

Table 13 SDF Constructs Used by the read_sdf Command (Continued)

SDF construct	SDF 2.1 support
IOPATH	Edge specification on input ports is ignored. 12 values are read; 6 values are used (combinational, enable and disable for rise and fall). The only mapping done is in the case of a single-valued SDF delay entry. This single value gets mapped to both the 01 (rise) and 10 (fall) timing arcs. If the library cell contains arrayed ports, the SDF file must contain bit-blasted entries. A range specification in the SDF file is not supported. The SDF file cannot reference an unindexed composite port name. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when the <code>-worst</code> option is specified. See the <code>read_sdf</code> man page. If the library cell has conditional delays, the SDF IOPATH entry automatically applies to all conditional paths specified in the library cell. IOPATH delays between two output ports are supported although SDF does not allow output to output IOPATH. If the SDF file contains delays from input to output but there is no library timing arc between the two pins, a warning appears and the IOPATH is ignored.
IOPATH	Conditional expressions are ignored by Design Compiler. In case of multiple conditions to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when the <code>-worst</code> option is specified. In case of multiple annotations to the same delay site, the delay value selected can be specified through the SDFPOLICY setup file variable.
PORt	Delays are annotated between all pins in the fanin and the input port. If the SDF file contains INTERCONNECT to the same input port, the interconnect value overrides the port value. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when the <code>-worst</code> option is specified. See the <code>read_sdf</code> man page. 12 values parsed; 2 values supported: 01, 10. The only mapping done is in the case of a single valued SDF delay entry. This single value gets mapped to both the 01 (rise) and 10 (fall) of the timing arc. If the library cell contains arrayed ports, the SDF file must contain bit-blasted entries. A range specification in the SDF file is not supported. The SDF file cannot reference an unindexed composite port name. The input port must be a leaf-cell pin. A warning message appears and the PORT is ignored if the input port is a nonleaf pin.

Table 13 SDF Constructs Used by the `read_sdf` Command (Continued)

SDF construct	SDF 2.1 support
INTERCONNECT	Net delays are annotated between the two given pins. If the SDF file contains PORT to the same input port, the interconnect value overrides the port value. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when the <code>-worst</code> option is specified. See the <code>read_sdf</code> man page. 12 values parsed; 2 values supported: 01, 10. The only mapping done is in the case of a single valued SDF delay entry. This single value gets mapped to both the 01 (rise) and 10 (fall) of the timing arc. If the library cell contains arrayed ports, the SDF file must contain bit-blasted entries. A range specification in the SDF file is not supported. The SDF file cannot reference an unindexed composite port name. Both pins must be leaf-cell pins. A warning message appears and the INTERCONNECT is ignored if either pin is a nonleaf pin.
NETDELAY	Parsed, not used.
DEVICE	Parsed, ignored.
conditional constraints	Parsed, not supported.
SETUP	Edge specification on both ports is supported. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when you specify the <code>-worst</code> option. See the <code>read_sdf</code> man page.
HOLD	Edge specification on both ports is supported. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when you specify the <code>-worst</code> option. See the <code>read_sdf</code> man page.
SETUPHOLD	Edge specification on both ports is supported. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when you specify the <code>-worst</code> option. See the <code>read_sdf</code> man page.
RECOVERY	Edge specification on both ports is supported. In case of multiple annotations to the same delay site, the delay value selected is the last value specified in the SDF file. The worst value is annotated when you specify the <code>-worst</code> option. See the <code>read_sdf</code> man page.
REMOVAL	You can specify removal timing checks with the HOLD statement.
SKEW	Ignored.
WIDTH	Ignored.
PERIOD	Ignored.
NOCHANGE	Ignored.

Table 13 SDF Constructs Used by the read_sdf Command (Continued)

SDF construct	SDF 2.1 support
PATHCONSTRAINT	Ignored.
SUM	Ignored.
DIFF	Ignored.
SKEWCONSTRAINT	Ignored.
C and C++ style comments	Supported.

Back-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.

Back-Annotating Load Delay

Load delay is the contribution to the overall cell propagation delay caused by capacitive loading. Load delays can be written out as part of the cell delay or as part of the net delay. The default assumes that the load delays are included in the cell delays in the timing file being read. Some delay calculators consider load delay to be part of the net delay, and others consider it part of the cell delay. Use the `read_sdf -load_delay` command to specify whether the load delay is part of the net delay or part of the cell delay.

Back-Annotating for Worst Timing Delay

Use the `remove_annotated_delay -all` and `remove_annotated_check` commands before using the `read_sdf -worst` command to annotate the worst timing delay and timing checks of all timing conditions for each pin-to-pin annotation.

This behavior does not depend on the type of timing arc being read (whether the arc is a SETUP arc or a HOLD arc). This is particularly important because worst might lead to the belief that for HOLD timing arcs minimum values will be read. The correct behavior is one where all the values read from are either minimum, typical, or maximum, regardless of the type of timing arc.

Back-Annotating Timing Checks

Setup, hold, recovery, and removal timing checks, when present in the timing file, are used to annotate the current design. The SDF constructs are

For this check	The SDF construct is
setup and hold	SETUP and HOLD
recovery	RECOVERY
removal	REMOVAL

Back-Annotating for Conditional Arc Cell Delay

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 or combination of inputs of the annotated cell. The way the tool annotates these conditional delays depends on whether the logic library specifies conditional arcs.

- If the logic library contains conditional arcs, the corresponding conditional delays specified in the SDF file are annotated on these arcs. The condition strings specified in the logic library and the SDF file must exactly match.
- If the logic library does not contain conditional arcs, the maximum delay or minimum delay, selected from all the SDF conditional arc delays, is annotated.

If your logic library does contain state-dependent delays, using the conditional arc delays from an SDF file is recommended because a more accurate timing analysis can result. Also, if you do an incremental compile after annotating these delays, the annotated conditional delays persist for those cells that are not replaced or removed during optimization. Because these annotated delays usually provide more accurate delay estimations, you can expect improved timing and quality of results.

Note:

When carrying out a timing analysis or incremental compile, you can use the `set_case_analysis` command to select a given condition on an input pin.

For example, if the delay for the timing arc from input pin A to output pin Z of cell U1 depends on the value (1 or 0) at input pin B and you want the A-to-Z delay value associated with 0 at pin B, enter

```
prompt> set_case_analysis 0 [get_pins U1/B]
```

Back-Annotation Order of Precedence

When pin-to-pin timing and resistance and capacitance values are back-annotated, timing information is obtained from the following sources in the following order of precedence:

1. SDF pin-to-pin values
 - The delay information in the SDF is the most accurate.
 - The timing reports have an asterisk (*) on timing arcs read from an SDF file. Note that annotated conditional timing arcs are reported.
 - The SDF data is sufficient if all the timing arcs in the design have a value back-annotated from an SDF file and the design meets timing.
2. Annotated resistance and capacitance values on a net
 - Because lumped resistance and capacitance values are considered, the RC delay numbers are pessimistic and not as accurate as SDF pin-to-pin delays.
 - For the following reasons, you need the resistance and capacitance values:

The technology or place and route library does not contain certain arcs. For example, the logic library has timing arcs for CLK to QB and QB to Q, but the logic library has a CLK-to-Q arc. The tool requires the back-annotated resistance and capacitance values to compute the timing.

Timing is not met, by a small margin, and you want to do in-place optimization. In such a case, you can use the resistance and capacitance values to compute timing with the new cells.

Large timing violations exist, and you want to recompile but with accurate wire load models (using the `create_wire_load` command). The tool uses the capacitance values when it creates the wire load models.

 - Missing resistance or capacitance value is extracted from the wire load model.
3. Wire load models
 - Wire load models are based on historical data; they can be the least accurate.
 - Both resistance and capacitance values are generated, then the delay is calculated.

Examples

This section presents usage examples of the `read_sdf` command.

Example 1

To read the SDF file `cir.sdf` for the design called “cir,” enter

```
prompt> current_design cir
prompt> read_sdf cir.sdf
```

Example 2

To read maximum and minimum delay values from two separate SDF files, enter

```
prompt> read_sdf -min_file min.sdf -max_file max.sdf
```

Example 3

To read timing information of instance u1 of design MULT16 from the disk file mult16_u1.sdf and have the timing annotated on the design MY_DESIGN, enter

```
prompt> current_design MY_DESIGN
prompt> read_sdf -load_delay net -path u1 mult16_u1.sdf
```

Writing an SDF File

You might want to write out the back-annotated delay information so that it can be used for gate-level simulation or some other purpose. You can use the `write_sdf` command to write the delay information in SDF v1.0 or v2.1 format. The default output format is v2.1.

For example, to write an SDF file, enter

```
prompt> write_sdf -version 2.1 mult16.sdf
```

Annotating Delays and Timing Checks From the Command Line

You can annotate delays and timing checks from the command line without using an SDF file by using the `set_annotated_delay` command or `set_annotated_check` command as described in the following sections.

Defining Net and Cell Delays

The `set_annotated_delay` command sets actual net and cell delay values in the current design or the net delay between two or more pins or ports on the same net.

To check the back-annotation completed by the `set_annotated_delay` command, use the `update_timing` command. If the design is hierarchical, link it by using the `link -all` command before you use the `set_annotated_delay` command.

Load delay is the portion of cell delay caused by the capacitive load of the net being driven. Some delay calculators consider load delay part of the net delay and others consider it part of the cell delay.

- If your annotated delay value assumes that load delay is part of the cell delay, use the `-load_delay cell` option.
- If your delay value assumes that load delay is included in the net delay, use the `-load_delay net` option.

By default, load delays are assumed to be in cell delays and appear in `report_timing` path listings.

Delays annotated with this command affect both timing analysis and optimization. Using the `extract_rc` command removes the delay information annotated on nets.

The specified delay value overrides the internally estimated cell and net delay value. If the specified pins are not in the same cell or on the same net, an error message appears when the timing is updated and `delay_value` is discarded for those pins. The `set_annotated_delay` command can be used for pins at lower levels of the design hierarchy. Pins are specified as INSTANCE1/INSTANCE2/PIN_NAME.

To list annotated delay values, use the `report_annotated_delay` command.

To remove the annotated cell or net delay values from a design, use the `remove_annotated_delay` or `reset_design` command.

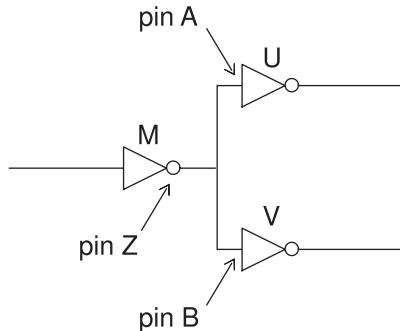
The syntax is

```
set_annotated_delay
  -net|-cell
  [-load_delay net | cell]
  [-rise | -fall]
  [-min] [-max]
  delay_value
  -from from_list -to to_list
  [-worst]
```

Note:

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

Example 1



For this circuit, to annotate a net delay of 4 on the net connected to cell output pin M/Z, enter

```
prompt> set_annotated_delay -net 4 -from M/Z
```

The net delay between M/Z and U/A is 4, and the net delay between M/Z and V/B is also 4.

To annotate a net delay of 5 on the net connected to cell input pin U/A, enter

```
prompt> set_annotated_delay -net 5 -to U/A
```

The net delay between M/Z and U/A is 5, and the net delay between M/Z and V/B is not changed.

To annotate a net delay of 6 between cell output pin M/Z and cell input pin U/A, enter

```
prompt> set_annotated_delay -net 6 -from M/Z -to U/A
```

The net delay between M/Z and u/A is 6, and the net delay between M/Z and V/B is not changed.

Defining different net delays for pins on the same net causes ambiguity, for example, setting delay on a net connecting the output pin Z of cell M to the input pin A of cell U.

```
prompt> set_annotated_delay -net -rise 4 -from M/Z
prompt> set_annotated_delay -net -rise 5 -to U/A
```

In this case, when timing is updated, a warning message appears.

```
(Warning) Overwriting the rise delay between pin 'M/Z' and
pin 'U/A' with (4). (OPT-835)
```

Example 2

To annotate a rise net delay of 12.3 units for minimum delay analysis between the output pins of two circuits, enter

```
prompt> set_annotated_delay -net -rise -min \
           -load_delay net 12.3 -from U1/Z -to U2/A
```

Defining Timing Check Values Between Pins

The `set_annotated_check` command sets an annotated timing check between two or more pins. The syntax is

```
set_annotated_check check_value
  -from from_pins -to to_pins
  -setup | -hold | -removal | -recovery |
  -nochange_high | -nochange_low
  [-rise -fall]
  [-clock clock_check]
  [-worst]
  [-increment]
```

Use the `set_annotated_check` command after place and route for technologies in which the timing check value varies between different instances and the library timing check values do not provide sufficient accuracy. This command automatically links the design if it is not already linked. To incrementally modify (rather than replace) a timing check value, use the `-increment` option.

You can use this command on pins at lower levels of the design hierarchy. Specify pins as `INSTANCE1/INSTANCE2/PIN_NAME`.

To list annotated timing check values, use `report_annotated_check`.

To see the effect of the `set_annotated_check` command for a specific instance, use the `report_timing` command.

Note:

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

Example

To annotate 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, enter

```
prompt> set_annotated_check -setup 2.1 -from u1/ff12/CP -to u1/ff12/D
```

To remove annotated timing check values from a design, use the `remove_annotated_check` or `reset_design` command.

Reporting Annotated Values and Checks

Reports provide information about the implementation of your design. The tool provides commands to report the following:

- Load and resistance values
- Annotated delay values
- Annotated timing checks
- Back-annotated values command summary

Reporting Load and Resistance Values

The `report_net` command displays the net load and resistance values. Annotated values appear in the Attributes column with a “c” or an “r”.

Attributes such as `dont_touch` are displayed for each net. The `dont_touch` attribute can be present on a net as an implicit attribute. This can happen when the `set_dont_touch_network` command is used. All nets in the transitive fanout of a port are affected, but the `dont_touch` setting cannot be removed independently from these nets.

The syntax is

```
report_net
[-nosplit] [-noflat] [-transition_times]
[-only_physical [-verbose]]
[-cell_degradation] [-min] [-connections [-verbose]]
[-physical [-verbose]] [net_list]
[-significant_digits [digits]]
[-max_toggle_rate]
```

Note:

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

Example

```
prompt> report_net
Information: Updating design information... (UID-85)
. .
Attributes:
  d - dont_touch
  c - annotated capacitance
  r - annotated resistance
Net      Fanout  Fanin  Load  Resistance  Pins  Attributes
-----
. .
cell157/n16    1      1     1.00    100.00      2      r
cell157/n17    3      1     3.17    100.00      4      c, r
cell157/n18    2      1     5.36    100.00      3      c, r
```

```
...
-----
Total 13 nets 25      13    31.53   1300.00    38
Maximum        4       1     8.53    100.00     5
Average        1.92   1.00   2.43    100.00   3.22
```

You can also display back-annotated values by using the `report_attribute -application -class net` command. The `load` attribute is used for back-annotated capacitance. The `ba_net_resistance` attribute is used for back-annotated resistance.

Reporting Annotated Delay Values

The `report_annotated_delay` command displays back-annotated data for cells or nets.

The delay values reported are the resulting cell and net delays used in the `report_timing` command and might not be your annotated values if delays were annotated with the `-load_delay net` option of the `read_sdf` command or the `set_annotated_delay` command.

The syntax is

```
report_annotated_delay [-cell] [-net] [-summary] [-nosplit]
```

Example 1

This example shows that the counter design has only one cell delay annotated between pin CO/A and pin CO/Z.

```
prompt> report_annotated_delay -cell
...
Cell Name      From      To      Rise      Fall
-----
CO            A        Z     100.00  100.00
```

Example 2

This example shows that the counter design has one net with annotated values. Net h has a lumped capacitance (50.00) and a lumped resistance (200.00). A net delay of 200 is annotated between two of the pins on net h. Net h has no annotated timing between pin ffc/QN and pin r/B.

```
prompt> report_annotated_delay -net
...
Net  Name      From      To      Rise      Fall      Load      Res.
-----
h    ffc/QN    w/A     200.00  200.00    50.00   200.00
h    ffc/QN    m/B     200.00  200.00    50.00   200.00
h    ffc/QN    r/B                  50.00   200.00
```

Reporting Annotated Timing Checks

The `report_annotated_check` command displays back-annotated timing checks on the current design, including the data rise and fall and the clock edge.

To list annotated delays, use the `report_annotated_delay` command (described earlier in this chapter).

The syntax is

```
report_annotated_check [-nosplit]
```

Example

```
prompt> report_annotated_check
...*
```

Cell Name	From	To	Rise	Fall	Timing Check
U1	c	cdn	0.16	0.00	recovery_clock_rising
U1	cn	cdn	0.33	0.00	removal_clock_rising

Post-layout optimization and in-place optimization remove the annotated timing checks when they become invalid (when a net connected to the cell with timing checks is modified).

Reporting Back-Annotated Values Command Summary

[Table 14](#) summarizes the commands for displaying back-annotated values in the current design.

Table 14 Summary of Commands for Reporting Back-Annotated Values

To report this	Use this
Timing information for current instance	<code>report_timing</code>
Back-annotated data for cells or nets	<code>report_annotated_delay</code>
Annotated timing checks	<code>report_annotated_check</code>
Net load and resistance values for current instance or current design	<code>report_net</code>
Attributes and their values associated with a cell, net, pin, port, instance, or design	<code>report_attribute</code>

Removing Back-Annotated Values

To remove delays read and annotated with `read_sdf`, use `reset_design` or `remove_annotated_delay`. The `compile` command also removes annotated delays (except delays in cells and nets that have the `dont_touch` attribute). To remove timing checks annotated with `read_sdf`, use the `remove_annotated_check` command. This section describes the following topics:

- Remove annotated delay values
- Remove annotated timing checks between specified pins
- Remove annotated resistance and capacitance values
- Remove back-annotated values command summary

Removing Annotated Delay Values

The `remove_annotated_delay` command removes annotated delay values between two pins or all annotated delay values in the same cell.

Both rise and fall annotated delays are removed. The `remove_annotated_delay` command can be used on pins at lower levels of the design hierarchy. These pins are specified as INSTANCE1/INSTANCE2/PIN_NAME.

If the current design is hierarchical, link the design by using the `link -all` command before you use the `remove_annotated_delay` command.

Use the `-from` and `-to` options in the same manner you used them to set annotated values.

For example,

To remove this	Use this
An annotated delay set with <code>set_annotated_delay -from</code>	<code>remove_annotated_delay -from</code>
An annotated delay set with <code>set_annotated_delay -to</code>	<code>remove_annotated_delay -to</code>
An annotated delay set with <code>set_annotated_delay -from -to</code>	<code>remove_annotated_delay -from -to</code>

A timing arc must exist between the pins in *from_list* and the pins in *to_list*, or Design Compiler will not remove any annotated delay and a warning will appear:

```
prompt> remove_annotated_delay -from ffd/Q -to m/B
Warning: There is no timing arc between pin 'ffd/Q' and pin
'm/B'. (OPT-834)
```

When the tool removes delays, informational messages appear:

```
Information: Removing delays annotated to pin 'w/A'. (OPT-831)
Information: Removing annotated delays from pin 'ffc/QN' to pin 'w/A'.
(OPT-830)
```

If no annotated delay values exist on a net or cell specified in the `remove_annotated_delay` command, no warning or informational message appears.

The syntax is

```
remove_annotated_delay -all | -from pin_list | -to pin_list
```

Note:

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

Examples

To remove a delay value from a net annotated with the command

```
prompt> set_annotated_delay -net delay_value -from ffd/CP -to m/B
enter
```

```
prompt> remove_annotated_delay delay_value -from ffd/CP -to m/B
```

To remove all annotated delays from the current design, enter

```
prompt> remove_annotated_delay -all
Information: Removing all annotated delays from design
'counter'. (OPT-804)
```

Removing Annotated Timing Checks Between Specific Pins

The `remove_annotated_check` command removes annotated timing check information between specific pins.

Removes annotated timing checks between the specified pins. Data rise and fall and clock rise and fall annotated checks are removed by default. The `remove_annotated_check` command can be used for pins at lower levels of the design hierarchy. These pins are specified as INSTANCE1/INSTANCE2/PIN_NAME.

If the design is not already linked, the `remove_annotated_check` command links it automatically.

The `remove_annotated_check` command removes annotated timing checks set by the `set_annotated_check` command and removes setup, hold, recovery, or removal annotated timing checks set by the `read_sdf` command.

The syntax is

```
remove_annotated_check
  -all | -from from_pins | -to to_pins
  [-rise |-fall] [-clock rise | fall]
  [-setup] [-hold] [-recovery] [-removal]
```

Note:

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

Example

To remove an annotated setup check between pins u1/u2/CP and u1/u2/D, enter

```
prompt> remove_annotated_check -setup -from u1/u2/CP -to u1/u2/D
```

To specify a rising or falling clock edge, enter the following command sequence:

```
prompt> remove_annotated_check -clock rise \
  -from [get_pins "U1/cn"] -to [get_pins "U1/sdn"]
prompt> remove_annotated_check -clock fall \
  -from [get_pins "U1/cn"] -to [get_pins "U1/sdn"]
```

You can use the `reset_design` command to remove back-annotated values in the design, but this removes all attributes and constraints from the design.

Removing Annotated Resistance or Capacitance Values

You can also use the `remove_attribute` command to remove resistance or capacitance back-annotation from specified nets in the design.

- To remove annotated capacitance, remove the `load` attribute.
- To remove annotated resistance, remove the `ba_net_resistance` attribute.

Examples

To remove the back-annotated resistance on net U1/U2/Net3, enter the following command:

```
prompt> remove_attribute [get_nets U1/U2/Net3]ba_net_resistance
```

To remove all annotated capacitances on all nets, enter the following command:

```
prompt> remove_attribute [get_nets *) load
```

To remove the annotated capacitance on net foo, enter the following command:

```
prompt> remove_attribute [get_nets "foo"] load
```

Removing Back-Annotated Values Command Summary

Table 15 summarizes the commands for removing back-annotated values.

Table 15 Summary of Commands for Removing Back-Annotated Values

To do this	Use this
Remove annotated values between two pins or all annotated values from the current design	<code>remove_annotated_delay</code>
Remove annotated timing checks between specific pins	<code>remove_annotated_check</code>
Remove annotated resistance	<code>remove_attribute load</code>
Remove annotated capacitance	<code>remove_attribute ba_net_resistance</code>
Remove all user-specified objects and attributes, except those defined using <code>set_attribute</code>	<code>reset_design</code>

Setting Net Load

To replace estimated wire capacitance values with actual values determined by an external tool, create a file containing one `set_load` command for each net, as in the following example:

```
set_load 2.739 INPUT7
set_load 2.101 net204
set_load 3.433 cell133/n28
set_load 1.007 FF21_Q
...
```

You then include the file as a Design Compiler command script. In some systems, your place and route tool can produce this file.

Use the following procedure to replace the estimated net loads with actual values:

1. Edit the file generated by your place and route tool to use the following line format:

```
set_load load_value net_name [-subtract_pin_load]
```

For example,

```
set_load 0.052 cell1109/n29
set_load 0.052 net251
set_load 0.052 net266
```

```
set_load 0.2152 net798
set_load 0.1052 net800
set_load 0.1052 net802
set_load 0.3052 net803
set_load 0.052 net804
```

- Run the script file by using the `source` command.

```
prompt> source load_file_name
```

- Display the resulting annotated nets with the `report_net`, `report_attribute`, or `report_annotation_delay` command.

The tool calculates total load for nets as follows:

```
total_net_load = pin_capacitance + wire_load
```

Pin capacitance is the sum of all capacitance values of all pins on a net, as defined in the logic library description. Wire load is the wire capacitance for a given net, usually estimated by Design Compiler from the wire load model.

You can back-annotate the `wire_load` parameter of the total net load equation by using the `set_load` command.

The syntax for the `set_load` command is:

```
set_load
load_value object_list
[-min] [-max]
[-subtract_pin_load]
[[-pin_load] [-wire_load]]
```

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

Example

```
prompt> set_load 6.53 cell157/n15
Performing set_load on net 'cell157/n15'.
```

Sometimes physical design tools produce capacitance reports that associate wire load with the name of a pin connected to the net. Use the `all_connected` command with the `set_load` command to identify the net to which the pin is connected, and set a load value on it. For example,

```
prompt> set_load 6.53 [all_connected cell157/inv2/z]
Performing set_load on net 'cell157/n15'.
```

CPU overhead is involved in processing the previous command. A design with 100,000 nets can take a long time to process.

Sometimes a net name equals a port name. In such cases, use the following command:

```
prompt> set_load 6.53 [get_nets cell157/n15]
```

The `all_connected`, `find`, and `get_nets` commands are described in the *Design Compiler User Guide*.

Setting Net Resistance

The `set_resistance` command defines the wire resistance for nets in an optimized design. This command overwrites the Design Compiler internally estimated net resistance values.

Resistance is estimated on the basis of the tree type used in the current operating conditions.

- For a balanced tree, the resistance annotated is assumed to be balanced across all loads. The resistance (R) from the driver to each of the N loads is R/N .
- For a worst-case tree, the resistance from the driver to each load is assumed to be R .
- For a best-case tree, resistance is ignored (not used). The net delay is zero, resulting in optimistic delay values.

The syntax is

```
set_resistance
  resistance_value
  [-min] [-max]
  object_list
```

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

Examples

To set a resistance of 200 units on nets a and b, enter

```
prompt> set_resistance 200 {a b}
```

To set a resistance of 300 units on the net U1/U2/Net3, enter

```
prompt> set_resistance 300 U1/U2/Net3
```

Sometimes physical design tools produce resistance reports that associate values with the name of a pin connected to the net. Use the `all_connected` command as an argument of the `set_resistance` command to identify the net to which the pin is connected. For example,

```
prompt> set_resistance 6.53 [all_connected(cell157/inv2/z)]
```

CPU overhead is involved in processing the previous command. A design with 100,000 nets can take a long time to process. In such cases, use the `get_nets` command as an argument for the `set_resistance` command. For example,

```
prompt> set_resistance 6.53 [get_nets cell157/n15]
```

Because the `set_load` and `set_resistance` commands verify the design links each time you run them, you can significantly reduce the runtime of a script if you link the design and disable the autolink feature at the beginning of the script. After the script has finished running, enable the autolink feature again.

For example, enter the following command sequence:

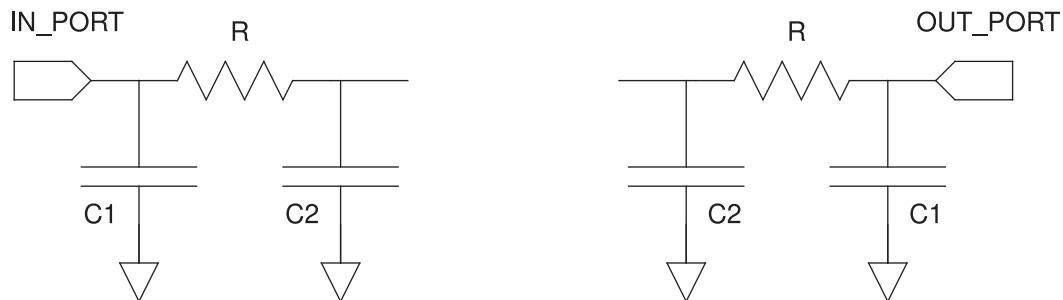
```
prompt> set auto_link_disable true
prompt> source load_res.file
prompt> set auto_link_disable false
```

By default, the `dctcl source` command does not echo the script commands to the screen. When you do not echo the script commands to the screen, the script runs faster and you can easily find messages about unfound nets in the output list.

Setting Pi Model Capacitance and Resistance

An alternative RC network loading model for ports is the pi model. This model contains two capacitors and one resistor, arranged like the Greek letter π , as shown in [Figure 94](#).

Figure 94 Pi Network Model for Input and Output Ports



To apply the pi loading model for a port, use the `set_pi_model` command. For example,

```
prompt> set_pi_model -resistance 200 -capacitance {5 10} [get_ports CLK]
```

The `-resistance` argument specifies the resistance value R and the `-capacitance` argument specifies the two capacitance values, C1 and C2, in units of the technology file.

The pi model overrides any net resistance specified by the `set_resistance` command and any port loads specified by the `set_load` command.

If you use the `-receiver` option of the `set_pi_model` command, the specified capacitance value represents the load of the receiver external to the design and connected to the output port. This value is added to the near-to-port capacitance of the pi model to obtain the total capacitance on the output port. It does not affect input ports.

The pi model is considered part of the RC network of the net of the specified port. The capacitance values of the pi model are counted as part of the total wire load and not as part of the port load. You can use the `report_net` or `report_port` command to view the resistance and capacitance values of the net or port.

To view the pi model on ports, use the `report_pi_model` command. To reset the pi model, use the `remove_pi_model` command.

Back-Annotating Detailed Parasitics

You can read net parasitic data generated by an external tool so that the timing analyzer can calculate delays from that data. Design Compiler can estimate wire delays by using wire load models or topographical technology. However, if you have detailed parasitic data available from an external tool, Design Compiler can use that information to accurately calculate net delays.

This is the `read_parasitics` command syntax in Design Compiler:

```
read_parasitics
  [-syntax_only]
  [-elmore | -arnoldi]
  [-increment]
  [-pin_cap_included]
  [-net_cap_only]
  [-complete_with zero | wlm]
  [-path path_name]
  [-strip_path path_name]
  [-quiet | -verbose]
  [-dont_write_to_db]
  [file_list]
```

This command can read parasitic data in SPEF, DSPF, or RSPF format.

The `-elmore` option makes the tool create an RC tree and back-annotate delay estimated from the parasitic data based on the Elmore delay model. The `-arnoldi` option does the same, but it uses the Arnoldi delay model instead. If neither `-elmore` nor `-arnoldi` is specified, the RC tree is created without estimating or back-annotating delays.

If you have multiple parasitic data files, add either `-elmore` or `-arnoldi` only to the last `read_parasitics` command, as shown in the following script example, to save runtime.

```
current_design A
read_parasitics A1.spf
read_parasitics A2.spf
read_parasitics A3.spf -arnoldi
```

The `read_parasitics` command options of the Design Compiler and IC Compiler tools are somewhat different. This is the syntax in the IC Compiler tool:

```
read_parasitics
[-format SPEF | SBPF]
[-syntax_only]
[-max_file max_file_name]
[-min_file min_file_name]
[-keep_capacitive_coupling]
[-triplet_type min | typ | max]
[file_list]
```

This command can read parasitic data in SPEF or SBPF format. IC Compiler uses its internal delay calculator to determine the delays resulting from the net parasitics. You can optionally read two files, one for minimum and one for maximum operating conditions.

For more information, see the `read_parasitics` man page for the applicable tool.

To remove some or all annotated delays, use the `remove_annotated_delay` command.

RTL Load Annotation With Wire Load Modeling

Design Compiler can models nets by using wire load models. Wire load models can be inaccurate for modeling very long nets. For interblock top-level nets this inaccuracy can be large. You can use the `set_load` command during reoptimize-design to annotate a load value on a net. During a compile, however, the annotated load is lost because the compile does not retain this information. Use RTL load annotation to retain this information.

With RTL load annotation, you can specify more accurate delays for long nets in the design, resulting in more realistic interconnect delays. RTL load annotation enables you to annotate a load value larger than the load suggested by wire load models on top-level interblock nets in the design.

The RTL load annotation is retained throughout the synthesis process and is used during structuring, mapping, and optimization. The net with the annotated load can undergo optimization and still retain the annotated load value. After placement is completed in the design cycle, the annotated loads can be replaced by more accurate models. Nets without any annotation or with optimized logic continue to use statistical wire load models for optimization.

RTL Load Connections

The RTL load annotation overrides the statistical wire loads for selected long nets during the compile. The annotated loads work with unmapped logic and DesignWare components. They can be used during structuring, mapping, and placement-independent optimization.

The value to be annotated can be obtained from

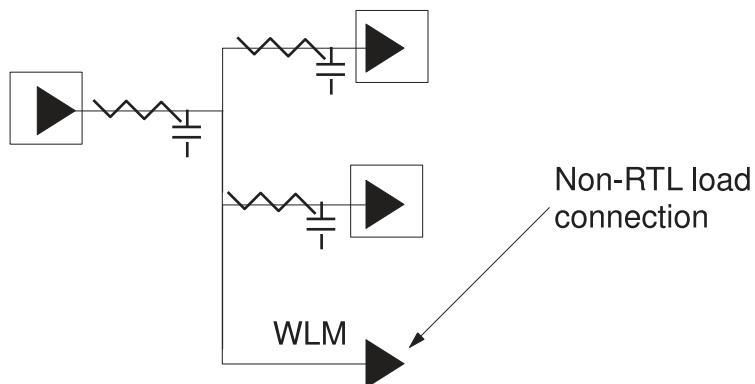
- Initial estimates
- Initial place and route
- Initial floorplan

The RTL load can be annotated on

- Ports and pins
- Nets between hierarchical ports and pins
- Nets between pins of `dont_touch`, `size_only`, or `black_box` cells

The annotated load is retained on the top level and on hierarchical ports and pins of `dont_touch`, `size_only`, or `black_box` gates. The RTL load annotation is on the net but stored on the pins. An annotated net must have at least one RTL load connection. See [Figure 95](#).

Figure 95 Setting an RTL Load



Using RTL Load Annotation

To annotate the load and resistance values on the nets, do the following steps:

1. Read in a design that has long nets. The design can be in RTL or .ddc format.
2. To annotate a load or resistance value on a net, use the `set_rtl_load` command.
If you are annotating a value on a net, be sure the net has at least one RTL load connection.

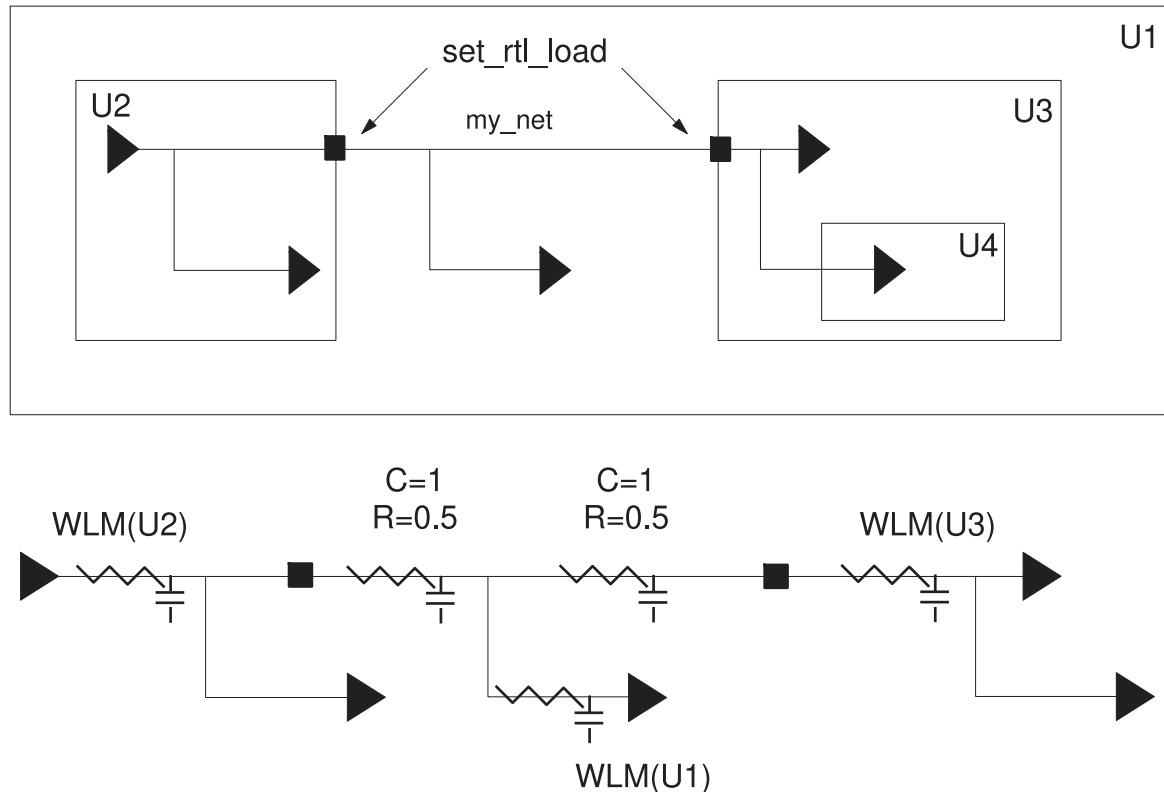
```
set_rtl_load
[-capacitance cvalue] [-resistance rvalue]
[-min] [-max] pin_net_list
```

The `set_rtl_load` command sets an RTL load value for capacitance and resistance on pins, ports, and nets. The command sets total capacitance (C) for a net and distributes it equally among RTL load connections. The resistance value (R) is applied equally to all connections.

```
prompt> set_rtl_load my_net -capacitance 2 -resistance 0.5
```

[Figure 96](#) shows how the capacitance value is distributed equally.

Figure 96 Calculating With RTL Loads



3. To see if the annotated value has been applied on the cell or net, use one of the following commands:

For a report on the RTL load values on cells, use

```
prompt> report_cell -connections -verbose
```

For a report the RTL load values on nets, use

```
prompt> report_net -connections -verbose
```

To write RTL load commands for the current design to a script, use the `write_rtl_load` command.

4. To remove the annotated load and resistance, use the `remove_rtl_load` command or the `reset_design` command.

- To remove an RTL load value for capacitance and resistance from pins, ports, and nets, use the following syntax:

```
remove_rtl_load [-all] [pin_net_list]
```

- To remove all user-specified objects and attributes from the current design, use

```
prompt> reset_design
```

Default Resistance Values

Specifying the resistance values directly for the `set_rtl_load` command is not required. Instead, you can calculate the resistance value as a constant factor multiplied by the capacitance. You choose the factor by setting the `rtl_load_resistance_factor` variable. For instance, if the RTL load capacitance of a pin is set to 4, the RTL load resistance is not set, and the `rtl_load_resistance_factor` variable is set to 0.5, a resistance of 2 is used. The factor is applied to each annotated pin of a net individually, not to the net's capacitance as a whole. Default library units are used throughout. The default is zero.

RTL Load Buffering

When you are using RTL load buffering, remember the following points:

- If buffers are added, the annotated load remains attached to the RTL load annotated pins.
 - Location and placement information is not taken into account during buffering.
 - RTL load buffering is intended for early synthesis approximation only.
-

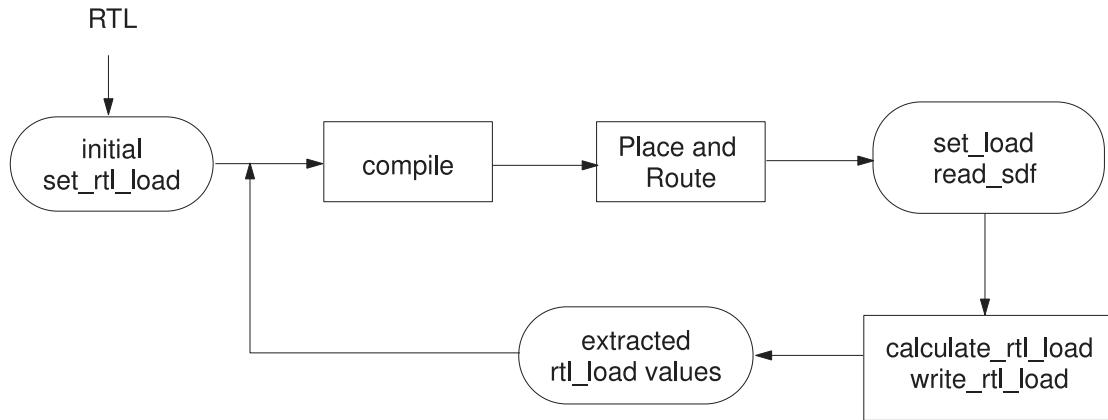
Extraction of RTL Loads

RTL loads are extracted only on nets and pins that you specify. Calculation of RTL loads is performed as follows:

- Capacitance from wire load modeling for random logic blocks is subtracted from the values specified by the `set_load` command.
- The remaining capacitance from the `set_load` command is distributed equally to user-specified pins.
- Resistance values are generated to try to match the timing delays from SDF.

See [Figure 97](#) for more information about RTL load extraction.

Figure 97 Extracting an RTL Load



The `calculate_rtl_load` command converts back-annotated layout information from `set_load`, **SDF**, and annotated delay to `rtl_load` values for RTL optimization.

7

Timing Reports

Design Compiler and IC Compiler can generate a wide range of reports that provide information about the design contents and analysis results. This chapter describes the most commonly used reports.

- [Reporting Commands](#)
 - [check_timing Command](#)
 - [report_constraint Command](#)
 - [report_timing Command](#)
 - [report_delay_calculation Command](#)
 - [get_timing_paths Command](#)
 - [report_clock_timing Command](#)
-

Reporting Commands

The reporting commands `check_timing`, `report_constraint`, `report_timing`, and `report_clock_timing` are commonly used in synthesis and physical implementation flows to get information about the timing of the design.

The `check_timing` command checks for constraint problems such as undefined clocking, undefined input arrival times, and undefined output constraints. These constraint problems could cause you to overlook timing violations. For this reason, the `check_timing` command is recommended whenever you apply new constraints such as clock definitions, I/O delays, or timing exceptions.

After the design is fully constrained, the `report_constraint` command tells you whether the design meets the timing, area, power, and design rule constraints. It reports the existence of any violations and shows the amount by which each constraint is violated, information about the design object that is the worst violator, and the weighted cost of the violation. The verbose mode of the command gives you detailed information about the violations.

The `report_timing` command provides detailed, point-by-point information on the clock paths and data paths that have the worst slack, along with the delay and slack calculation.

You can control the scope of the design that is reported, the number of paths to report, and the types of path information that are included in the report. This information is very helpful in determining the cause of timing violations and how to fix them. The `get_timing_paths` command lets you create collections of timing paths for further analysis and processing.

The `report_clock_timing` command reports the clock latency, transition time, and skew characteristics at specified clock pins of sequential elements in the network. You specify the type of report you want, the scope of the design to analyze, and any desired filtering or ordering options for the report. The tool gathers the requested clock information and reports it in the specified order. The report is useful for debugging latency and skew problems in the clock network.

Table 16 lists the commands commonly used to check timing constraints, display timing analysis results, and report timing-related functions in Design Compiler and IC Compiler.

Table 16 Timing Reporting Commands

Command	Description
<code>all_connected</code>	Creates a collection of objects connected to a net, port, or pin
<code>all_critical_cells</code>	Creates a collection of critical cells
<code>all_critical_pins</code>	Creates a collection of critical pins
<code>all_fanin</code>	Creates a collection of pins or cells in transitive fanin
<code>all_fanout</code>	Creates a collection of pins or cells in transitive fanout
<code>all_inputs</code>	Creates a collection of input ports
<code>all_outputs</code>	Creates a collection of output ports
<code>all_registers</code>	Creates a collection of register cells or pins
<code>check_design</code>	Checks the design for consistency
<code>check_timing</code>	Checks for timing constraint problems; see check_timing Command
<code>get_timing_paths</code>	Creates a collection of timing paths; see get_timing_paths Command
<code>report_annotated_delay</code>	Reports annotated delays; see Reporting Annotated Delay Values
<code>report_attribute</code>	Reports attributes of objects
<code>report_case_analysis</code>	Reports case analysis settings; see Reporting Case Analysis
<code>report_cell</code>	Reports cells in the design
<code>report_clock</code>	Reports clocks defined in the design; see Reporting Clock Information

Table 16 Timing Reporting Commands (Continued)

Command	Description
report_clock_gating	Reports clock-gating cells and registers created by Power Compiler
report_clock_gating_check	Reports clock-gating timing checks
report_clock_timing	Reports timing attributes of clock networks: latency, skew, transition; see report_clock_timing Command
report_clock_tree	Reports structural and timing characteristics of compiled clock trees
report_constraint	Reports constraints and constraint violations: timing, design rules, power; see report_constraint Command
report_delay_calculation	Reports calculation of a cell or net timing delay; see Library Timing Data
report_design	Reports design attributes: libraries, register type, operating conditions, wire load models, input/output delays
report_design_lib	Reports contents of design libraries
report_disable_timing	Reports timing arcs that have been disabled by case analysis, conditional arcs, loop breaking, or by the <code>set_disable_timing</code> command
report_hierarchy	Reports the design hierarchy
report_lib	Reports library information, including timing-related information; see Library Timing Data
report_net	Reports nets of the design
report_port	Reports ports of the design
report_qor	Reports quality of results: slack, cell count, and area
report_reference	Reports hierarchical cell references
report_timing	Reports design timing (paths, slack, violations); see report_timing Command
report_timing_derate	Reports timing derate settings; see Setting On-Chip Variation Derating Factors
report_timing_requirements	Reports timing exceptions; see Reporting Exceptions
report_transitive_fanin	Reports pins in the transitive fanin to a specified sink object
report_transitive_fanout	Reports pins in the transitive fanout from a specified source object

Several of the reporting commands have a `-significant_digits` option, which you can use to specify the number of digits to the right of the decimal point displayed in the report. This setting affects only the display of values in the report, not the precision used internally for calculations. The `report_default_significant_digits` variable specifies the default number of significant digits displayed by all of the reporting commands.

The remaining sections of this chapter describe some of the more commonly used timing-related reporting commands. For more information, see the man page for the command.

check_timing Command

Paths that are incorrectly constrained might not appear in the violation reports, possibly causing you to overlook paths with violations. For this reason, the `check_timing` command is recommended to check any new constraints such as clock definitions, I/O delays, or timing exceptions.

The `check_timing` command checks for constraint problems such as undefined clocking, undefined input arrival times, and undefined output constraints. In addition, it 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 new constraints using commands such as `create_clock`, `set_input_delay`, and `set_output_delay`.

The syntax is

```
check_timing
  [-overlap_tolerance minimum_distance]
  [-override_defaults check_list]
  [-include check_list]
  [-exclude check_list]
  [-multiple_clock]
  [-retain]
```

Here is an example of a typical `check_timing` report:

```
prompt> check_timing

Information: Checking generated_clocks...

Information: Checking loops...

Information: Checking no_input_delay...

Information: Checking unconstrained_endpoints...

Warning: The following end-points are not constrained for maximum delay.

End point
-----
```

```
sd_CK
sd_CKn
```

By default, the `check_timing` command performs several types of constraint checking and issues a report like the one shown in the foregoing example. 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 more information, see the `check_timing` man page.

Table 17 lists and briefly describes the conditions that the `check_timing` command can detect. The types of checking that are performed by default are marked with an asterisk in the first column.

Table 17 Issues Detected by the check_timing Command

Potential problem	Report results
<code>clock_crossing</code>	Checks interactions between multiple clocks. If a path is launched by one clock and captured by another, the clocks are reported. For each pair of interacting clocks, the report shows an asterisk if all the paths are false paths or a pound sign (#) if some of the paths are false paths.
<code>clock_no_period</code>	Issues a warning if a clock does not have a period specified.
<code>data_check_multiple_clock</code>	Issues a warning if multiple clocked signals reach a data check register reference pin.
<code>data_check_no_clock</code>	Issues a warning if no clocked signal reaches a data check register reference pin. In that case, no setup or hold checks are performed on the constrained pin.
<code>gated_clock</code>	Issues a warning about any gated clocks found.
<code>generated_clock*</code>	Checks the generated clock network and reports any of the following conditions: a source pin is not the clock source, a generated clock definition point has no path to the source, multiple generated clocks form a loop, a generated clock does not have both <code>-master</code> and <code>-add</code> options (even if by itself), or if a generated clock source pin has an incoming clock that has no outgoing generated clock propagating it.
<code>generic</code>	Issues a warning about any unmapped cells in the design, which have zero delay.
<code>ideal_timing</code>	Issues a warning about any ideal transition or latency set on a non-ideal pin.
<code>loops*</code>	Issues a warning about any combinational feedback loops found.
<code>multiple_clock</code>	Issues a warning if multiple clocks reach a register pin.

Table 17 Issues Detected by the *check_timing* Command (Continued)

Potential problem	Report results
<code>net_no_driving_info</code>	Issues a warning if a net has coupled parasitics but does not have a driving pin or there is no timing arc on the driver pin.
<code>no_driving_cell*</code>	Issues a warning if a port is connected to a net that has parasitic data, and the port does not have a driving cell.
<code>no_input_delay*</code>	Issues a warning if no clock-related delay is specified for an input port, and there is no default clock (the <code>timing_input_port_default_clock</code> variable is set to false).
<code>partial_input_delay*</code>	Issues a warning about any port that has only the minimum delay or only the maximum delay specified by the <code>set_input_delay</code> command.
<code>pulse_clock_cell_type*</code>	Issues a warning about any cell instance that has a mismatched pulse type defined in the library.
<code>retain</code>	Issues a warning about any retain timing-check value that is larger than its corresponding delay value.
<code>unconstrained_endpoints*</code>	Shows unconstrained register data pins or primary outputs that are not marked as false paths.
<code>object_collection</code>	Checks if a constraint does not use the required <code>get_*</code> command to collect the required design objects.
<code>related_clock</code>	Issues a warning if two related clocks (having same origin or source) have any dependency between them.
<code>multicycle_path</code>	Issues a warning if the launch clock is slower than the capture clock for a <code>set_multicycle_path</code> command.
<code>create_clock</code>	Issues a warning if there is a <code>create_clock</code> definition specified in the fanout of another <code>create_clock</code> definition.
<code>set_case_analysis</code>	Checks if there are conflicting values for the same signal applied by the <code>set_case_analysis</code> command.

* Asterisk indicates a type of check that is enabled by default

report_constraint Command

The `report_constraint` command produces a summary of the constraint violations in the design, including the amount by which each constraint is violated, information about the design object that is the worst violator, and the weighted cost of the violation.

The default report displays brief information about the worst evaluation for each constraint in the current design and the overall cost. For example,

```
prompt> report_constraint
...


| Group           | (max_delay/setup) | Cost | Weight | Weighted Cost |
|-----------------|-------------------|------|--------|---------------|
| CLK             | 0.00              | 1.00 | 0.00   |               |
| default         | 0.00              | 1.00 | 0.00   |               |
| max_delay/setup |                   |      |        | 0.00          |


| Group          | (critical_range) | Total Slack | Neg Endpoints | Critical Cost |
|----------------|------------------|-------------|---------------|---------------|
| CLK            | 0.00             | 0           | 0             | 0.00          |
| default        | 0.00             | 0           | 0             | 0.00          |
| critical_range |                  |             |               | 0.00          |


| Group          | (min_delay/hold) | Cost | Weight | Weighted Cost |
|----------------|------------------|------|--------|---------------|
| CLK            | 0.00             | 1.00 | 0.00   |               |
| default        | 0.00             | 1.00 | 0.00   |               |
| min_delay/hold |                  |      |        | 0.00          |


| Constraint      | Cost            |
|-----------------|-----------------|
| max_transition  | 0.01 (VIOLATED) |
| max_fanout      | 0.00 (MET)      |
| max_capacitance | 0.00 (MET)      |
| max_delay/setup | 0.00 (MET)      |
| critical_range  | 0.00 (MET)      |


```

The command reports all types of design constraints, such as area and power constraints, in addition to timing constraints. These are the command options that are relevant to timing analysis:

```
report_constraint
[-all_violators]
[-verbose]
[-max_capacitance] [-min_capacitance]
[-max_delay] [-min_delay]
[-max_transition]
[-min_pulse_width]
```

You can optionally restrict the scope of the report by specifying the appropriate options in the command, such as `-max_delay` to report only maximum delay constraint violations.

Use the `-verbose` option to get detailed information about the object that caused the worst violation or came closest to causing a violation. For example,

```
prompt> report_constraint -max_capacitance -max_transition -verbose
...
Net: n234

max_transition      2.00
- Transition Time   2.01
-----
Slack              -0.01 (VIOLATED)

List of pins on net "n234" with transition violations :
-----
          Required      Actual
          Transition    Transition   Slack
-----
PIN :     U52/Z        2.00        2.01      -0.01 (VIOLATED)

Net: I_PCI_TOP/I_PCI_READ_FIFO/we_n

max_capacitance    0.17
- Capacitance      0.12
-----
Slack              0.06 (MET)
```

Getting a verbose report on a timing constraint such as maximum delay produces a timing report similar to that generated by the `report_timing` command. The report shows the timing for the path that caused the worst violation or the smallest slack for each path group.

Use the `-all_violators` option to get a summary listing of all violators, including the path endpoints that caused timing violations. For example,

```
prompt> report_constraint -all_violators
...
max_delay/setup ('SYS_CLK' group)

          Required      Actual
          Path Delay  Path Delay   Slack
-----
Endpoint
-----
I_BLENDER_1/mega_shift_reg_2_30_/D      8.50        8.63 r      -0.13 (VIOLATED)
I_BLENDER_1/mega_shift_reg_1_30_/D      8.52        8.61 r      -0.09 (VIOLATED)
I_BLENDER_1/s4_op2_reg_30_/D            8.53        8.61 r      -0.08 (VIOLATED)
I_BLENDER_1/mega_shift_reg_0_30_/D      8.52        8.60 r      -0.07 (VIOLATED)
I_BLENDER_1/s4_op1_reg_30_/D            8.52        8.59 r      -0.07 (VIOLATED)
I_BLENDER_1/s4_op2_reg_31_/D
```

I_BLENDER_1/s4_op1_reg_31_/D	8.54	8.61 f	-0.07	(VIOLATED)
	8.54	8.57 f	-0.02	(VIOLATED)
<hr/>				
max_transition				
Net	Required Transition	Actual Transition	Slack	
n234	2.00	2.01	-0.01	(VIOLATED)
PIN : U52/Z	2.00	2.01	-0.01	(VIOLATED)

report_timing Command

The `report_timing` command provides detailed, point-by-point timing information for the paths that have the worst slack. You can control the scope of the design that is reported, the number of paths to report, and the types of path information to include in the report. The information in the report can help you determine how to fix the violations.

report_timing Report Contents

The following is a typical path timing report generated by the `report_timing` command.

```

prompt> report_timing
...
Startpoint: I_RISC_CORE/I_INSTRN_LAT/Instrn_1_reg_27_
(rising edge-triggered flip-flop clocked by SYS_2x_)
Endpoint: I_RISC_CORE/I_ALU/Zro_Flag_reg
(rising edge-triggered flip-flop clocked by SYS_2x_CLK)
Path Group: SYS_2x_CLK
Path Type: max

Point                                Incr      Path
-----
clock SYS_2x_CLK (rise edge)          0.00      0.00
clock network delay (propagated)     0.51      0.51
I_RISC_CORE/I_INSTRN_LAT/Instrn_1_reg_27_/CP (senrq1) 0.00      0.51 r
I_RISC_CORE/I_INSTRN_LAT/Instrn_1_reg_27/_Q (senrq1)  0.62      1.13 f
I_RISC_CORE/I_INSTRN_LAT/Instrn_1[27] (INSTRN_LAT)    0.00      1.13 f
I_RISC_CORE/I_ALU/ALU_OP[3] (ALU)      0.00      1.13 f
I_RISC_CORE/I_ALU/U288/ZN (nr03d0)   0.36 *    1.49 r
I_RISC_CORE/I_ALU/U261/ZN (nd03d0)   0.94 *    2.43 f
I_RISC_CORE/I_ALU/U307/ZN (invbd2)   0.35 *    2.78 r
I_RISC_CORE/I_ALU/U343/Z (an02d1)    0.16 *    2.93 r
I_RISC_CORE/I_ALU/U344/ZN (nr02d0)   0.11 *    3.04 f
I_RISC_CORE/I_ALU/U348/ZN (nd03d0)   0.28 *    3.32 r
I_RISC_CORE/I_ALU/U355/ZN (nr03d0)   0.29 *    3.60 f
I_RISC_CORE/I_ALU/U38/Z (an02d1)     0.15 *    3.75 f

```

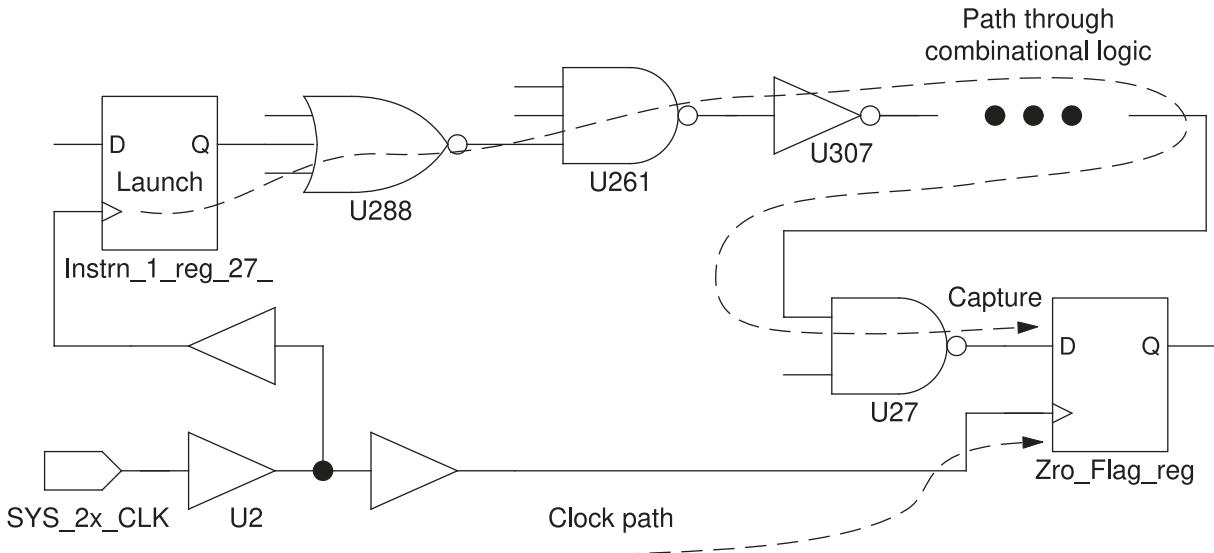
I_RISC_CORE/I_ALU/U40/Z (an02d1)	0.12 *	3.87	f
I_RISC_CORE/I_ALU/U48/ZN (nd02d1)	0.06 *	3.93	r
I_RISC_CORE/I_ALU/U27/ZN (nd02d1)	0.06 *	3.99	f
I_RISC_CORE/I_ALU/Zro_Flag_reg/D (secrq4)	0.00 *	3.99	f
data arrival time		3.99	
clock SYS_2x_CLK (rise edge)	4.00	4.00	
clock network delay (propagated)	0.47	4.47	
clock uncertainty	-0.10	4.37	
I_RISC_CORE/I_ALU/Zro_Flag_reg/CP (secrq4)	0.00	4.37	r
library setup time	-0.37	4.00	
data required time		4.00	

data required time		4.00	
data arrival time		-3.99	

slack (MET)		0.01	

In this example, the logic associated with the reported path is shown in [Figure 98](#).

Figure 98 Timing Path Logic



The report starts by showing the path startpoint, path endpoint, path group name, and path timing check type. In this example, the path timing check type is shown as “max,” which means a maximum-delay or setup check; “min” would mean a minimum-delay or hold check.

The table shows point-by-point accounting of the delays along the path from the startpoint to the endpoint. The table has columns labeled Point, Incr, and Path. These columns list

the points (cell pins) along the path, the incremental contribution to the delay at each point, and the cumulative delay up to that point, respectively. The table also lists the hierarchical boundary crossings, showing zero incremental delay at each crossing.

The *Incr* column shows the amount of incremental delay and can also show a symbol to indicate the dominant source of annotated delay information. In the earlier example, the asterisk (*) symbols indicate SDF back-annotated delay values. [Table 18](#) shows the annotation symbols that can be displayed and the corresponding sources of back-annotated delay information.

Table 18 Incr Column Symbols

Symbol	Source of timing information
H	Hybrid annotation; dominant sources are different for the included cell and net
^	Ideal network latency annotation
*	Back-annotation using SDF or preroute Elmore extraction
&	RC network back-annotation using Elmore delay calculation
\$	RC pi back-annotation
+	Lumped RC back-annotation
c	Arnoldi delay calculation with accurate CCS
@	Arnoldi delay calculation
~	Object has tool-inferred operating condition
(blank)	Wire-load model or no back-annotation

The letters “r” and “f” in the Path column indicate the sense of the signal transition, either rising or falling, at that point in the path.

The path starts with the launch clock edge and ends at the data input of the capture device. The “data arrival time” shown in the table is the amount of elapsed time from the source of the launch clock edge to the arrival of data at the endpoint, taking into consideration the longest possible delays along the path.

After this is the accounting for the required arrival time. The “data required time” shown in the table is the latest allowable arrival time for the data at the path endpoint, taking into account the nominal capture clock edge time, the clock network delay, the clock uncertainty, the least possible delay along the clock path, and the library setup time requirement for the capture device.

The required time is subject to adjustment for clock reconvergence pessimism removal (CRPR). When this feature is enabled, the tool checks for the presence of a shared segment between the launch clock path and capture clock path, such as buffer U2 in the foregoing circuit diagram. The calculated slack is pessimistic if different delay values are used in the common segment for launch and for capture. This pessimism is removed by the CRPR function. For details, see [Clock Reconvergence Pessimism Removal on page 168](#).

The slack value shown at the end of the report is simply the data required time minus the data arrival time. The slack is the amount of time by which the timing constraint is met, considering the latest possible arrival of data at the endpoint and the earliest possible arrival of the capture clock edge.

You can control the level of detail provided in the report by using the `report_timing` command options. For example, to show incremental cell and net delays separately rather than combined, use the `-input_pins` option. To show the point-by-point delays in the clock paths as well as in the data path, use the `-path full_clock_expanded` option.

report_timing Command Options

The `report_timing` command offers a large number of options to control the scope of the design that is reported, the number of paths to report, and the types of path information to include in the report. This is the full syntax of the command:

```
report_timing
  [-to to_list] [-rise_to to_list] [-fall_to to_list]
  [-from from_list] [-rise_from from_list] [-fall_from from_list]
  [-exclude exclude_list]
    [-rise_exclude exclude_list] [-fall_exclude exclude_list]
  [-through through_list]
    [-rise_through through_list]
    [-fall_through through_list]
  [-path_type
    full | end | only | short | start | full_clock | full_clock_expanded]
  [-delay_type min | min_rise | min_fall | max | max_rise | max_fall]
  [-nworst paths_per_endpoint]
  [-max_paths max_path_count]
  [-input_pins]
  [-nets]
  [-transition_time]
  [-crosstalk_delta]
  [-capacitance]
  [-effective_capacitance]
  [-lesser_path max_path_delay]
  [-greater_path min_path_delay]
  [-slack_greater_than greater_slack_limit]
  [-slack_lesser_than lesser_slack_limit]
  [-loops]
  [-enable_preset_clear_arcs]
```

```
[-significant_digits digits]
[-nosplit]
[-unique_pins]
[-start_end_pair]
[-physical]
[-attributes]
[-sort_by group | slack]
[-normalized_slack]
[-group group_name]
[-derate]
[-temperature]
[-voltage]
[-trace_latch_borrow]
[-scenario scenario_list]
```

By default, the `report_timing` command by itself, without any options, reports the single worst maximum-delay (setup) path in each path group. Therefore, the number of paths reported is equal to the number of path groups. By default, there is one path group per clock used in the design.

Scope of the Design Reported

The `-from`, `-through`, and `-to` options restrict the scope of the report to only those paths that start, pass through, and end at the specified objects in the design. The options `-rise_from`, `-fall_to`, and so on, further restrict the scope to only those paths that have a rising-edge signal at the startpoint, a falling-edge signal at the endpoint, and so on. The specified object can be a clock, a port, or a pin of a launch or capture cell.

For example, the following command restricts the scope of the report to only those paths that start and end at specified objects:

```
prompt> report_timing -from [get_pins reg08/CP] -to [get_clocks CLK1]
```

This command reports the single path having the worst maximum-delay (setup) slack that starts with a rising-edge signal at the CP pin of cell reg08 and ends at a register or port clocked by CLK1.

You can use (or not use) any combination of `-from`, `-through`, and `-to` type options. For details, see [Path Specification Methods](#).

To restrict the scope of the report to a particular path group, use the `-group` option. For example, to report only the paths in the CLK1 path group,

```
prompt> report_timing -group CLK1
```

By default, paths are divided into groups according to the endpoint clock. For example, a design that has clocks named CLK1 and CLK2 has two path groups named CLK1 and CLK2. You can also specify your own path grouping with the `group_path` command. For details, see [Path Groups](#).

To exclude all paths from the report that start from, pass through, or end at specified pins, ports, nets, or cells, use the `-exclude`, `-rise_exclude`, or `-fall_exclude` option. For example, to exclude all paths containing the net n234 from reporting, use the following command:

```
prompt> report_timing -exclude [get_nets n234]
```

Path Details Reported

By default, each line of the path report shows the point along the path, the incremental contribution to the delay of the point, and the cumulative delay up to that point. Each point includes both the cell and net delay from the previous point. For example,

```
prompt> report_timing
...
Point           Incr      Path
-----
...
I_BLENDER_1/mult_50/U701/CO (ad01d2)    0.24      4.93 f
I_BLENDER_1/mult_50/U698/CO (ad01d0)    0.26      5.19 f
I_BLENDER_1/mult_50/U699/CO (ad01d0)    0.29      5.48 f
...
```

To display the net names associated with the timing points, use the `-nets` option:

```
prompt> report_timing -nets
...
Point           Fanout     Incr      Path
-----
...
I_BLENDER_1/mult_50/U701/CO (ad01d2)    0.24      4.93 f
I_BLENDER_1/mult_50/n842 (net)          1         0.00      4.93 f
I_BLENDER_1/mult_50/U698/CO (ad01d0)    0.26      5.19 f
I_BLENDER_1/mult_50/n843 (net)          1         0.00      5.19 f
I_BLENDER_1/mult_50/U699/CO (ad01d0)    0.29      5.48 f
I_BLENDER_1/mult_50/n844 (net)          1         0.00      5.48 f
...
```

To display net delays separately from the cell delays, use the `-input_pins` option, which lists the cell input pins as well as the cell output pins as timing points:

```
prompt> report_timing -input_pins -significant_digits 5
...
Point           Incr      Path
-----
...
```

Chapter 7: Timing Reports

report_timing Command

```
I_BLENDER_1/mult_50/U701/CI (ad01d2) 0.00005 4.69329 f
I_BLENDER_1/mult_50/U701/CO (ad01d2) 0.24126 4.93454 f
I_BLENDER_1/mult_50/U698/CI (ad01d0) 0.00005 4.93459 f
I_BLENDER_1/mult_50/U698/CO (ad01d0) 0.25960 5.19420 f
I_BLENDER_1/mult_50/U699/CI (ad01d0) 0.00005 5.19424 f
I_BLENDER_1/mult_50/U699/CO (ad01d0) 0.29044 5.48468 f
...

```

The incremental delay leading up to a cell input is a net delay. The incremental delay leading up to a cell output is the cell delay.

To display the signal transition time at each timing point, use the `-transition_time` option:

```
prompt> report_timing -transition_time
...
```

Point	Trans	Incr	Path
<hr/>			
...			
I_BLENDER_1/mult_50/U701/CO (ad01d2)	0.13	0.24	4.93 f
I_BLENDER_1/mult_50/U698/CO (ad01d0)	0.21	0.26	5.19 f
I_BLENDER_1/mult_50/U699/CO (ad01d0)	0.24	0.29	5.48 f
...			

Similarly, use the `-capacitance` option to display the net capacitance, `-physical` to display the physical locations of the driver and load pins, and so on. You can combine multiple options of this type in a single command. The tool increases the number of columns in the report as needed.

The `-path` option specifies what type of timing points are included in the path report. It can be set to `short`, `full`, `full_clock`, `full_clock_expanded`, `only`, or `end`. The default report type is `full`.

A `full_clock` report is like a `full` report, except that it also reports the timing points in the clock paths leading up to the launch and capture devices. A `full_clock_expanded` report is like a `full_clock` report, except that it also reports the timing points between a primary source clock and a generated clock. For example,

```
prompt> report_timing
...
Startpoint: I_BLENDER_1/mega_shift_reg_3_24_
(rising edge-triggered flip-flop clocked by SYS_CLK)
...
Point                                              Incr      Path
-----
clock SYS_CLK (rise edge)                         0.00      0.00
```

Chapter 7: Timing Reports

report_timing Command

```

clock network delay (propagated)          0.83      0.83
I_BLENDER_1/mega_shift_reg_3_24_/CP (sdcrq1) 0.00      0.91 r
I_BLENDER_1/mega_shift_reg_3_24_/Q (sdcrq1) 0.40      1.23 r
...
.

prompt> report_timing -path full_clock

Startpoint: I_BLENDER_1/mega_shift_reg_3_24_
(rising edge-triggered flip-flop clocked by SYS_CLK)
...
Point                                         Incr      Path
-----
clock SYS_CLK (rise edge)                  0.00      0.00
sys_clk (in)                            0.07      0.07 r
I_BLENDER_1/U483/Z (ora21d4)            0.34      0.41 r
I_BLENDER_1/buffd4_G2B1I2/Z (buffd4)    0.20      0.60 r
I_BLENDER_1/bufbd7_G2B2I2/Z (bufbd7)   0.22      0.83 r
I_BLENDER_1/mega_shift_reg_3_24_/CP (sdcrq1) 0.00      0.83 r
I_BLENDER_1/mega_shift_reg_3_24_/Q (sdcrq1) 0.40      1.23 r
...
.
```

A **short** report shows only the startpoint and endpoint, omitting the intermediate points. An **only** report shows the data path only and omits the required-time and slack calculation.

An **end** report shows only the endpoint, delay, required time, and slack in a single line, but all endpoints are reported instead of just the single worst path per path group. The **-path end** option is typically used with the **-nworst** option to restrict the report length. For example,

```

prompt> report_timing -path end -max_paths 5
...
Endpoint                               Path Delay      Path Required      Slack
-----
I_BLENDER_1/mega_shift_reg_2_30_/D (sdcrq1) 8.64 r        8.51      -0.13
I_BLENDER_1/s4_op2_reg_30_/D (sdnrq1)     8.63 r        8.54      -0.09
I_BLENDER_1/mega_shift_reg_0_30_/D (sdcrq1) 8.61 r        8.52      -0.09
I_BLENDER_1/mega_shift_reg_1_30_/D (sdcrq1) 8.62 r        8.53      -0.09
I_BLENDER_1/s4_op1_reg_30_/D (sdnrq1)     8.61 r        8.53      -0.08
.
```

Delay Type (Min/Max) Reported

By default, the **report_timing** command reports the path or paths having the worst maximum-delay (setup) slack resulting from the longest delays and earliest required times. To get a report on the paths having the worst minimum-delay (hold) slack instead, use the

`-delay min` option. In that case, the command looks for the shortest delays and latest required times. The minimum-delay slack is the arrival time minus the required time.

You can use the `-delay min_rise`, `-delay max_fall`, and similar options to restrict the scope of the report to just rising or just falling edges of the data signal at the path endpoint.

Number of Paths Reported

The `-nworst` and `-max_paths` options control the number of paths reported. The `-nworst` option specifies the number of paths reported per endpoint. The `-max_paths` option specifies the number of paths reported per path group.

For example, the following command reports the eight worst maximum-delay paths per path group, but no more than two paths per endpoint:

```
prompt> report_timing -nworst 2 -max_paths 8
```

If the three worst maximum-delay paths in a path group have the same endpoint, only the first two are reported; the third-worst path reported must have a different endpoint in the path group.

The `-slack_lesser_than` and `-slack_greater_than` options restrict the report to paths that have slack values within the specified range. For example, the following command lists up to 50 worst maximum-delay paths per path group that have a slack less than 0.5:

```
prompt> report_timing -path end -max_paths 50 -slack_lesser_than 0.5
```

The command reports the paths in order of slack, starting with the worst path, and continues until 50 paths have been reported or until the slack exceeds 0.5 time units.

Other Options

The `report_timing` command has several more options not described here. For details, see the `report_timing` command man page.

report_delay_calculation Command

For a detailed description of the delay calculation for a particular cell instance or net in the design, you can use the `report_delay_calculation` command. For example,

```
prompt> report_delay_calculation -from I_RISC_CORE/I_ALU/U27/A2 \
           -to I_RISC_CORE/I_ALU/U27/ZN
...
Rise Delay

cell delay = 0.0583731
Table is indexed by
(X) input_pin_transition = 0.103374
```

```
(Y) output_net_total_cap = 0.00451049
Relevant portion of lookup table:
(X) 0.0150      (X) 0.2500
(Y) 0.0000      (Z) 0.0270      (Z) 0.0680
(Y) 0.0070      (Z) 0.0480      (Z) 0.0990

Z = A + B*X + C*Y + D*X*Y
A = 0.0244          B = 0.1745
C = 2.9088          D = 6.0790

Z = 0.0583731
scaling result for operating conditions
multiplying by 1 gives 0.0583731 ...
```

In the `report_delay_calculation` command, you must specify a “from” pin and a “to” pin. The two specified pins must be either an input pin and an output pin of a cell instance or the driver pin and a load pin of a net. The command gives a detailed report on the calculation of the delay from the “from” pin to the “to” pin, including library data used, operating conditions, and transition times.

get_timing_paths Command

You can use the `get_timing_paths` command to create a collection of paths for custom reporting and other processing. You can assign these timing paths to a variable or pass them into another command. From the collection variable, you can query the collection to obtain the path attributes such as the slack, timing points, and intermediate delays.

Use the `foreach_in_collection` command to iterate among the paths in the collection. The `index_collection`, `copy_collection`, `add_to_collection`, and `remove_from_collection` commands are not applicable to timing path collections. You can use the `get_attribute` command to obtain information about the paths.

The following example shows how to create a path collection, set a variable to that collection, and extract information about the path from the variable.

```
prompt> set mypath [get_timing_paths -delay_type max -group SYS_CLK]
{path1}
prompt> get_attribute $mypath endpoint
{"I_BLENDER_1/mega_shift_reg_2__30_/D"}
prompt> get_attribute $mypath endpoint_clock_pin
{"I_BLENDER_1/mega_shift_reg_2__30_/CP"}
prompt> get_attribute $mypath arrival
8.45935
prompt> get_attribute $mypath slack
0.0432777
```

The following example shows how to iterate through a path collection to extract attributes from each path in the collection.

```

prompt> set mypaths [get_timing_paths -delay_type max \
                     -group SYS_CLK -nworst 2 -max_paths 10]
{path1 path2 path3 path4 path5 path6 path7 path8 path9 path10}
prompt> foreach_in_collection iter $mypaths {
    ? query_objects [get_attribute $iter endpoint]
    ? query_objects [get_attribute $iter startpoint]
    ? echo [get_attribute $iter slack]
    ?
}
{I_BLENDER_1/mega_shift_reg_2__30_/D}
{I_BLENDER_1/mega_shift_reg_3__24_/CP}
0.0432777
{I_BLENDER_1/mega_shift_reg_2__30_/D}
{I_BLENDER_1/mega_shift_reg_3__24_/CP}
0.0475168
{I_BLENDER_1/mega_shift_reg_1__30_/D}
{I_BLENDER_1/mega_shift_reg_2__0_/CP}
0.0796604
...

```

The first command creates a collection containing the ten worst maximum-delay (setup) timing paths clocked by SYS_CLK, with no more than two paths per endpoint. The next command iterates through the collection using “iter” as the iteration variable, and extracts the endpoint, startpoint, and slack of each path.

Note:

When iterating over a collection, to avoid excessive runtime, avoid writing scripts that use a large number of `current_design` commands in a loop.

[Table 19](#) shows the attributes supported on timing paths.

Table 19 Attributes Supported on Timing Paths

Attribute	Type
<code>clock_uncertainty</code>	float
<code>endpoint</code>	string
<code>endpoint_clock</code>	string
<code>endpoint_clock_close_edge_type</code>	string
<code>endpoint_clock_close_edge_value</code>	float
<code>endpoint_clock_is_inverted</code>	Boolean
<code>endpoint_clock_is_propagated</code>	Boolean
<code>endpoint_clock_open_edge_type</code>	string
<code>endpoint_clock_open_edge_value</code>	float

Table 19 Attributes Supported on Timing Paths (Continued)

Attribute	Type
endpoint_clock_pin	string
endpoint_hold_time_value	float
endpoint_is_level_sensitive	Boolean
endpoint_output_delay_value	float
endpoint_recovery_time_value	float
endpoint_removal_time_value	float
endpoint_setup_time_value	float
object_class	string
path_group	string
path_type	string
points	string
slack	string
startpoint	string
startpoint_clock	string
startpoint_clock_is_inverted	Boolean
startpoint_clock_is_propagated	Boolean
startpoint_clock_latency	float
startpoint_clock_open_edge_type	float
startpoint_clock_open_edge_value	float
startpoint_input_delays_value	float
startpoint_is_level_sensitive	Boolean
time_borrowed_from_endpoint	float
time_lent_to_startpoint	float

One attribute of a timing path is the points collection. A point corresponds to a pin or port along the path. Iterate through these points using the `foreach_in_collection` command

and get the attributes on them using the `get_attribute` command. [Table 20](#) shows the attributes supported for a point of a timing path.

Table 20 Attributes Supported for Points of a Timing Path

Attribute	Type
arrival	string
object	string
object_class	string
rise_fall	string
slack	string

To create a collection of clock groups in the design, use the `get_path_groups` command.

For more information, see the man pages for the collection commands (`man collections`), the `get_timing_paths` command, and the `foreach_in_collection` command.

report_clock_timing Command

The `report_clock_timing` command reports the clock latency, transition time, and skew characteristics at specified clock pins of sequential elements in the network. In the command, you specify the type of report you want (latency, transition time, single-clock skew, interclock skew, or summary), the scope of the design to analyze, and any desired filtering or ordering options for the report. The tool 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 the tool. This information is maintained for all clock pins of sequential devices in the design.

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,

```
prompt> report_clock_timing -type latency -to U1/CP \
           -hold -capture -rise
```

In this example, the tool 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`, the tool 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,

```
prompt> report_clock_timing -type latency \
           -to {U1/CP U7/CP} -hold -capture -rise
```

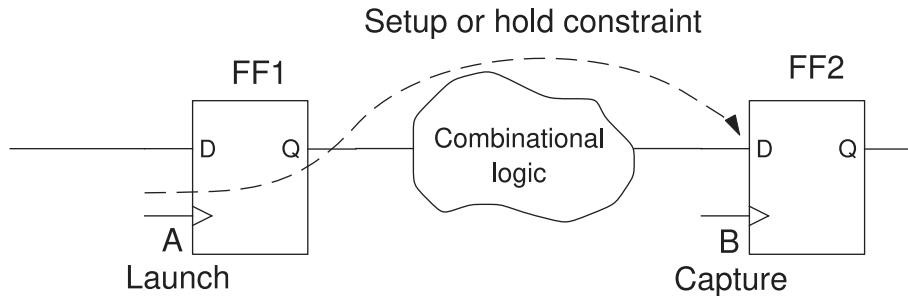
When a pin named in the “to” list is not a clock pin of a sequential element, the tool replaces that pin with the set of clock pins in the transitive fanout of the named pin. Here, “clock pin” means the clock pin of a flip-flop or the gate pin of a latch.

Skew Reporting

To get a single-clock skew report, use the `report_clock_timing -type skew` command. This generates a report on skews between pins clocked by the same clock. To get a report on skews between pins clocked by different clocks as well as by the same clock, use the `report_clock_timing -type interclock_skew` command, as described in the section [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 illustrated in [Figure 99](#). The tool 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 99 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, the tool 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 data used
- Role of the sequential device in the constraint relationship: launch or capture

If clock reconvergence pessimism removal is enabled, the tool takes it into account for the skew calculation. For details, see [Clock Reconvergence Pessimism Removal](#).

You can optionally restrict the scope of the report by specifying “from” and “to” pins in the design. For example, to get a report on the clock skew for a setup path between a negative-level-sensitive launch latch and a rising-edge-triggered capture flip-flop, you would use a command like this:

```
prompt> report_clock_timing -from latch/G -to ff/CP \
           -type skew -setup
```

The tool 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. The tool 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, the tool uses the set of all sequential device clock pins in the design that communicate with the 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 get a report on 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 the following information:

- The skew between pin A, clocked by CLK1, and pin B, clocked by CLK2
- The worst local skew between all sequential devices clocked by CLK1 and all sequential devices clocked by CLK2
- The 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 the tool must analyze, 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 100](#). 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 100 Clock Network Timing Report Types

Clock network report

- Summary report (minimum and maximum values for each clock)
`report_clock_timing -type summary`
- List report
 - Skew report (pin pairs and associated skew, one clock per pair)
`report_clock_timing -type skew -nworst 5`
 - Interclock skew report (pin pairs and associated skew, any clocks)
`report_clock_timing -type interclock_skew -nworst 5`
 - Pin report (single pins and associated latency, transition time)
`report_clock_timing -type transition -nworst 5`
- Verbose report (detailed timing along clock paths)
`report_clock_timing -from U1/CP -to U5/CP -verbose -type skew`

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,

```
prompt> report_clock_timing -type summary -clock [get_clocks CLK1]
...
Clock: CLK1
-----
Maximum setup launch latency:
  f2_2/CP           6.11    rp+-
Minimum setup capture latency:
  f1_2/CP           1.00    rpi-+
Minimum hold launch latency:
  f1_2/CP           1.00    rpi-+
Maximum hold capture latency:
  f2_2/CP           6.11    rp+-
Maximum active transition:
  13_2/G            0.13    rpi-
Minimum active transition:
  12_3/G            0.00    rp-
Maximum setup skew:
  f2_2/CP           rp+-
  f2_1/CP           4.00    rp+-
Maximum hold skew:
  f3_3/CP           rpi-+
```

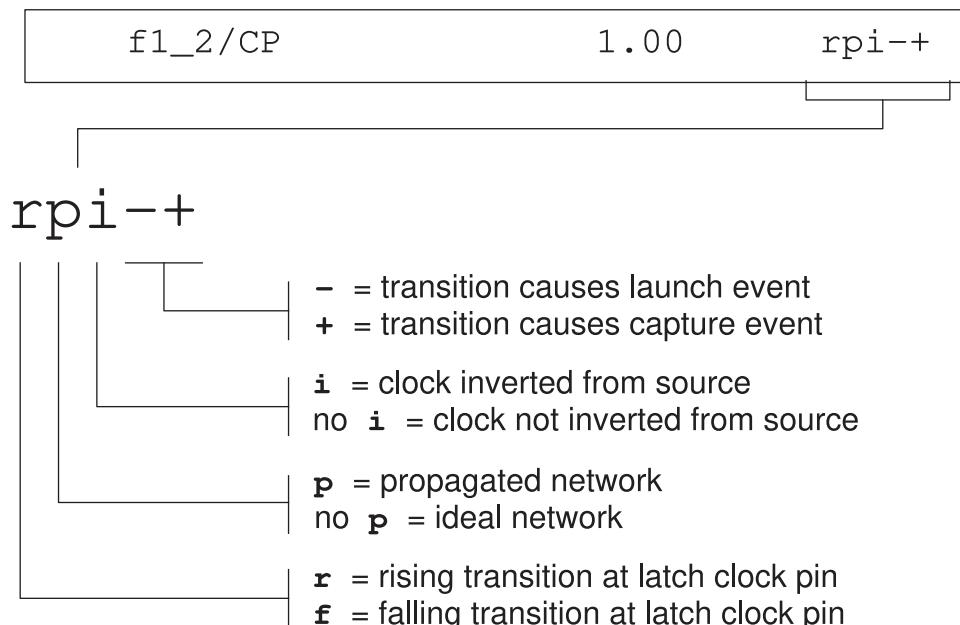
f2_2/CP	3.01	rp-+
---------	------	------

The report shows the following information for each minimum and maximum latency, transition time, and skew value:

- Pins at which the minimum or maximum value occurred
- Minimum or maximum time value
- Conditions under which the minimum or maximum time value occurred

The string of characters in the rightmost column shows the conditions under which the minimum or maximum time value occurred, following the conventions shown in [Figure 101](#).

Figure 101 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,

```
prompt> 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 the tool gather, filter, and display a collection of clock pins according to a specified attribute of interest.

There are two types of lists: 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.

An example of a command to generate a skew report is as follows:

```
prompt> 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 Pin	Latency	CRP	Skew
<hr/>			
f2_2/CP	6.11		rp-+
f2_1/CP	2.01	-0.10	rp-+
<hr/>			
12_2/G	4.11		rp-
f1_2/CP	1.00	-0.10	rpi-+
<hr/>			
f2_2/CP	6.11		rp-+
13_3/G	3.01	-0.10	rp-
<hr/>			

The rightmost column shows the conditions under which the reported values occurred at each corresponding pin, using the codes shown in [Figure 101](#).

Here is an example of a command used to generate an interclock skew report:

```
prompt> report_clock_timing -type interclock_skew \
      -nworst 12 -setup -include_uncertainty_in_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
<hr/>			
f2_2/CP	6.11		rp-+

```
f2_1/CP          2.01      0.11      4.21      fp-+
...

```

The report shows the 12 largest skew values between clock pins anywhere in the design, 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 of startpoints and endpoints can serve as a warning about a potentially long runtime to generate the rest of the report.

To show the names of the two clocks for each entry in the interclock skew report, use the `-show_clocks` option of the command.

Here is an example of a command to generate a pin report:

```
prompt> 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
<hr/>					
13_2/G		0.10	4.00	4.10	rpi-
f3_1/CP		0.11	3.00	3.11	rp-+
12_1/G		0.10	4.00	4.10	rpi-
f3_3/CP		0.10	3.00	3.10	rpi-+
13_3/G		0.11	3.00	3.11	rp-
<hr/>					

The rightmost column shows the conditions under which the reported values occurred, using the codes shown in [Figure 101](#).

To get a list of the largest latency values rather than transition times, use `-type latency` instead of `-type transition`.

Verbose (Path) 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,

```
prompt> 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
-----
clock source latency      0.00      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

clock source latency      0.11      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

endpoint clock latency      6.11
startpoint clock latency     -3.00
clock reconvergence pessimism   -0.10
inter-clock uncertainty      0.21
-----
skew                  3.22

```

A command using the **-latency** or **-transition** option rather than the **-skew** option produces a similar report, except that only one clock path is reported rather than two.

To include information in the path report about the net delays, net names, net capacitance, cell and net attributes, or physical locations of pins in the path, use the **-input_pins**, **-nets**, **-capacitance**, **-attributes**, or **-physical** option. These options are similar to those available in the **report_timing** command.

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.

To restrict the report to specific scenarios, use the **-scenarios** option of the **report_clock_timing** command.

8

Graphical User Interface

The graphical user interface (GUI) provides a way to view design data and timing analysis results in graphical form, including schematics, layout views, histograms, data tables, and reports. Using the GUI for timing analysis is described in the following sections:

- [Using the GUI for Timing Analysis](#)
 - [Path Slack Histogram](#)
 - [Path Inspector](#)
 - [Timing Analysis Driver](#)
-

Using the GUI for Timing Analysis

The graphical user interface (GUI) of the Design Compiler and IC Compiler tools can help you visualize and understand the nature of timing problems in the design, including the type, number, magnitude, and locations of the paths, cells, and nets that are causing timing problems. Using the GUI, you can do the following:

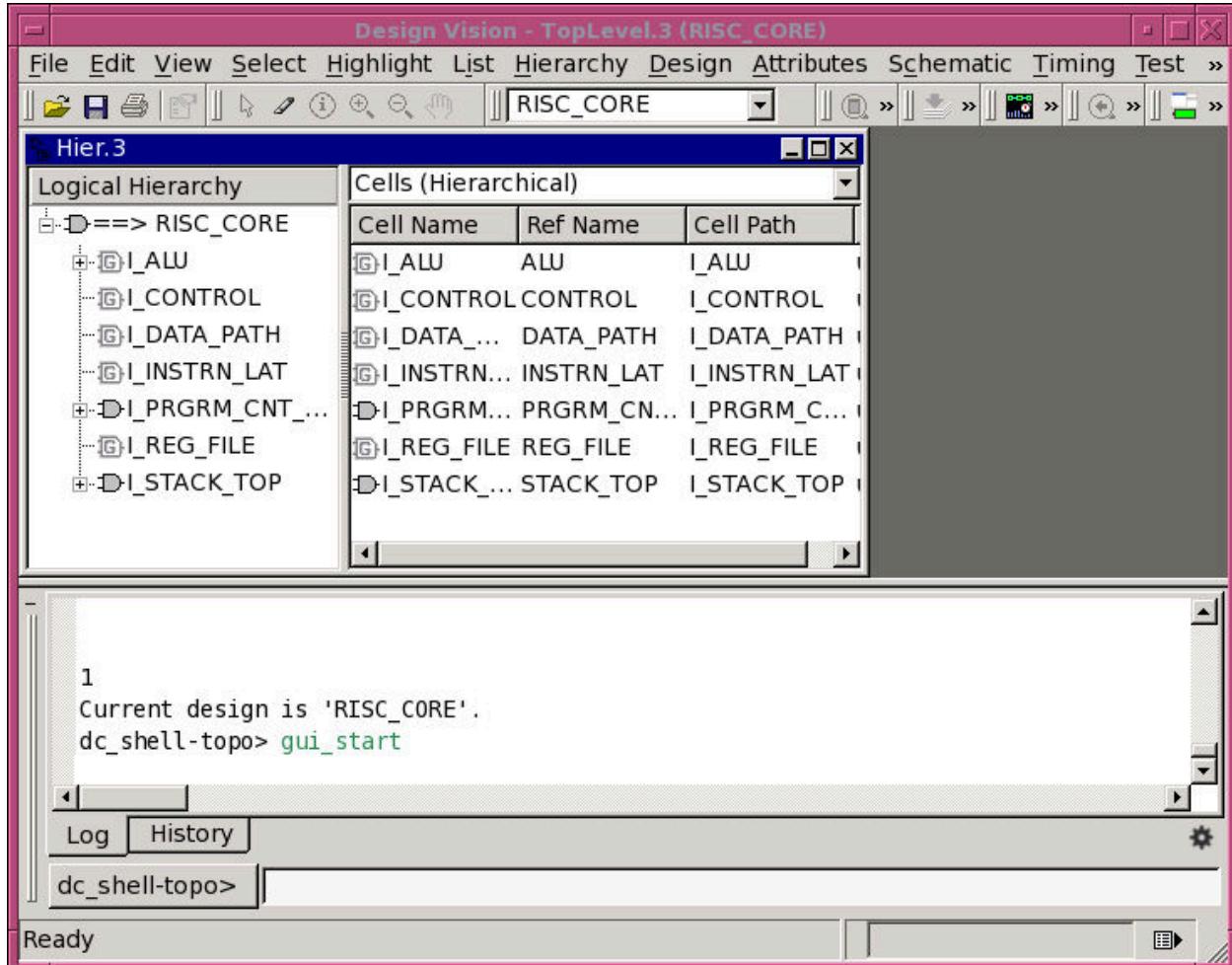
- Browse the logic hierarchy graphically
- Generate path slack histograms
- Select and examine timing paths using text, schematics, spreadsheets, and delay profiles
- Display the physical locations of problem paths, cells, and nets in the IC Compiler layout window
- Examine timing path collections for custom trend analysis

To start the GUI from the tool's shell prompt, use the `gui_start` command:

```
prompt> gui_start
```

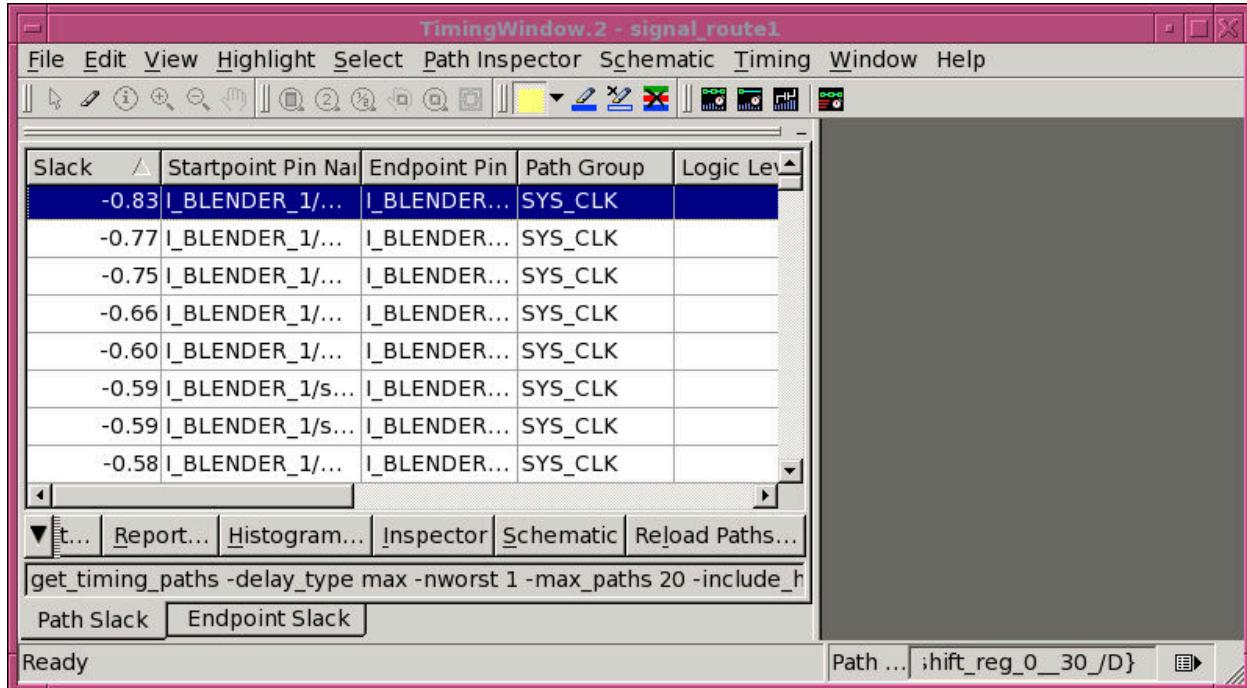
In Design Compiler, starting the GUI opens the Design Vision main window, from which you can view the design hierarchy and perform a wide range of synthesis functions, including timing analysis. [Figure 102](#) shows the Design Vision window.

Figure 102 Design Vision (Design Compiler GUI) Window



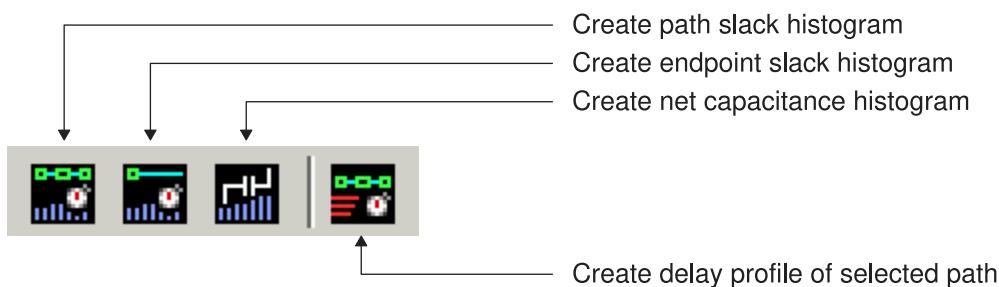
In IC Compiler, starting the GUI opens the IC Compiler main window, from which you can perform a wide range of physical implementation functions. You can display the physical layout and perform timing analysis in separate windows called the layout window and timing window. To open the timing window from the main window, choose Window > New Timing Analysis Window, or from the layout window, choose Timing > New Timing Analysis Window. [Figure 103](#) shows the IC Compiler Timing Analysis window.

Figure 103 IC Compiler Timing Analysis Window



In the Design Vision window or IC Compiler timing analysis window, you can use the buttons shown in Figure 104 to open a path slack histogram, endpoint slack histogram, net capacitance histogram, or path delay profile. Additional timing analysis commands are available in the tool's Timing menu.

Figure 104 Timing Analysis Buttons



Online help on using the GUI is available in the GUI itself. In the Design Compiler GUI, choose Help > Online Help, or in the IC Compiler GUI, choose Help > IC Compiler Online Help. For general information on using the GUI, see the *Design Vision User Guide* in the Design Compiler documentation set or the *IC Compiler Implementation User Guide* in the IC Compiler documentation set on SolvNet (<https://solvnet.synopsys.com>).

To stop the GUI while still keeping the original shell session, use the `gui_stop` command or the File > Close GUI menu command.

Timing Menu Commands

Table 21 lists and briefly describes the commands available in the Timing menu of the top-level Design Vision window and in the IC Compiler timing analysis window.

Table 21 Design Vision and IC Compiler Timing Menu Commands

Menu command	Action taken
Timing > New Path Analyzer (Design Vision only)	Displays the path analyzer window, which shows timing path collections for custom trend analysis.
Timing > New Path Inspector (Design Vision only)	Displays the path inspector window, which shows a wide range of information about the selected path, including a complete path schematic, clocking information, and timing information.
Timing > Timing Analysis Driver (Design Vision only)	Displays the timing analysis driver, which lists the timing paths having the worst slack in spreadsheet form. You can sort, filter, and modify the data in the table and examine any particular path in detail.
Timing > Path Slack	Displays a histogram of slack values for a set of paths that you specify by startpoint, endpoint, number of paths, group name, and delay type.
Timing > Slack Histogram of Selected Logic	Displays a histogram of slack values for the paths whose endpoints are currently selected in a logic schematic.
Timing > Slack Histogram of Selected Paths	Displays a histogram of slack values for the currently selected paths.
Timing > Endpoint Slack	Displays a histogram of slack values for a specified range of slack, reporting no more than one path per endpoint.
Timing > Net Capacitance	Displays a histogram of net capacitance values for a specified range of capacitance values.
Timing > Capacitance of Selected Nets	Displays a histogram of net capacitance values for the currently selected nets.
Timing > Path Profile View	Displays path delay profiles for the currently selected paths.
Timing > Check Timing	Runs the <code>check_timing</code> command, which checks the current design for timing problems such as unconstrained endpoints, no input delay specified, or errors involving multiple clocks.
Timing > Report Timing Path	Runs the <code>report_timing</code> command. You specify the command options in a dialog box. In the generated timing report, you can click the intermediate points on the path to highlight them in the layout and schematic.

Table 21 Design Vision and IC Compiler Timing Menu Commands (Continued)

Menu command	Action taken
Timing > Report Timing Requirements	Runs the <code>report_timing_requirements</code> command, which reports the timing exceptions that have been set. You specify the command options in a dialog box.
Timing > Report Clock Skew	Runs the <code>report_clock_skew</code> command, which reports the rise delay, fall delay, and uncertainty of the clock network.
Timing > Report Clock Tree	Runs the <code>report_transitive_fanout_clock_tree</code> command, which reports the transitive fanout from all the clock source pins and ports, listing the drivers and loads of the clock trees.
Timing > Report Path Group	Runs the <code>report_path_group</code> command, which lists the path groups and shows their respective weight and critical range settings.
Timing > Report Wire Load (Design Vision window in non-topographical mode)	Runs the <code>report_wire_load</code> command, which reports the characteristics of the wire load models in the design or library.

IC Compiler Layout Window Timing Menu Commands

[Table 22](#) lists and briefly describes the commands available in the Timing menu of the IC Compiler layout window.

Table 22 IC Compiler Layout Window Timing Menu Commands

Menu command	Action taken
Timing > Timing and Optimization Setup	Opens an interactive dialog box that lets you set a wide range of timing and optimization options such as synthesis design rules, cost priority, high-fanout synthesis options, power optimization parameters, and extraction parameters.
Timing > Set SI Options	Opens a dialog box that lets you set the crosstalk analysis and optimization options, including whether to enable crosstalk prevention and whether to consider delta delay, static noise, minimum delta delay, and timing windows.
Timing > Static Noise Analysis	Opens a static noise analysis window that displays the victim nets, aggressor nets, and noise parameters such as slack, peak noise, and individual aggressor noise contribution.
Timing > Write Parasitics	Runs the <code>write_parasitics</code> command. A dialog box lets you specify the parameters for the generated file, including the format (SPEF or SBPF) and file name.
Timing > Color By Delta Delay	Changes the layout view to display a color-coded map of crosstalk delta delays on nets.

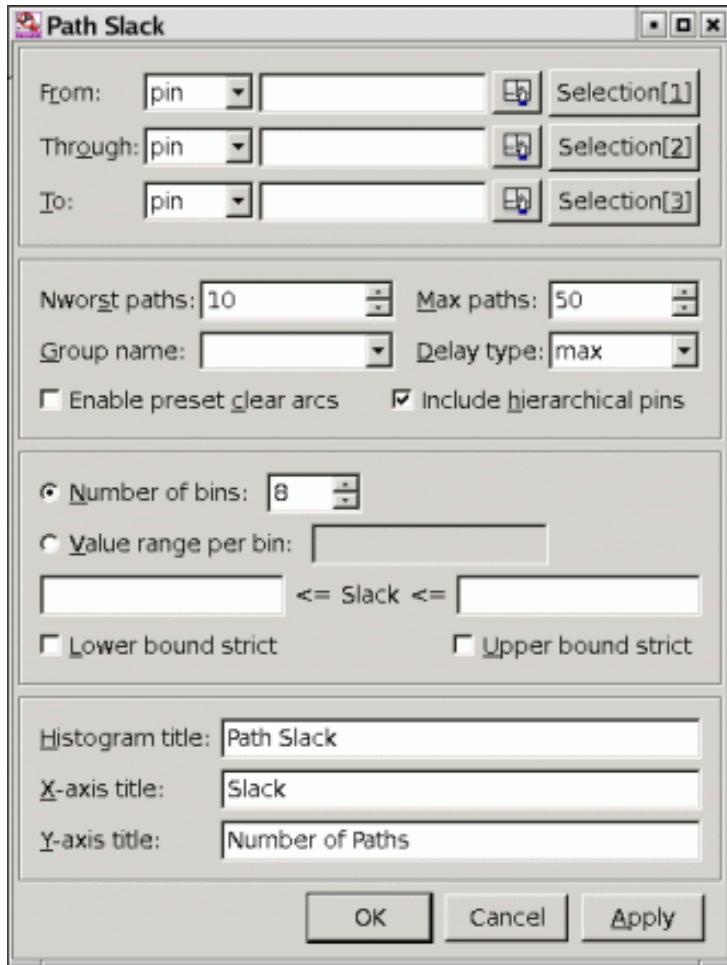
Table 22 IC Compiler Layout Window Timing Menu Commands (Continued)

Menu command	Action taken
Timing > Color By Static Noise	Changes the layout view to display a color-coded map of crosstalk static noise values on nets.
Timing > Color By Imported Paths	Changes the layout view to display a color-coded map of timing paths in the design. You import one or more <code>report_timing</code> path reports from a file or by copying and pasting into a dialog box.
Timing > New Timing Analysis Window	Opens a new timing analysis window containing a list of timing paths in table form, from which you can generate schematics, histograms, reports, and path delay profiles.

Path Slack Histogram

A path slack histogram shows the distribution of slack values for timing paths in the design. To generate a path slack histogram, use the Timing > Path Slack or Timing > Endpoint Slack menu command in the Design Vision main window or the IC Compiler timing analysis window, or use the equivalent toolbar buttons shown in [Using the GUI for Timing Analysis](#). Executing the command opens a dialog box in which you can enter the analysis options, as shown in [Figure 105](#).

Figure 105 Path Slack Histogram Dialog Box

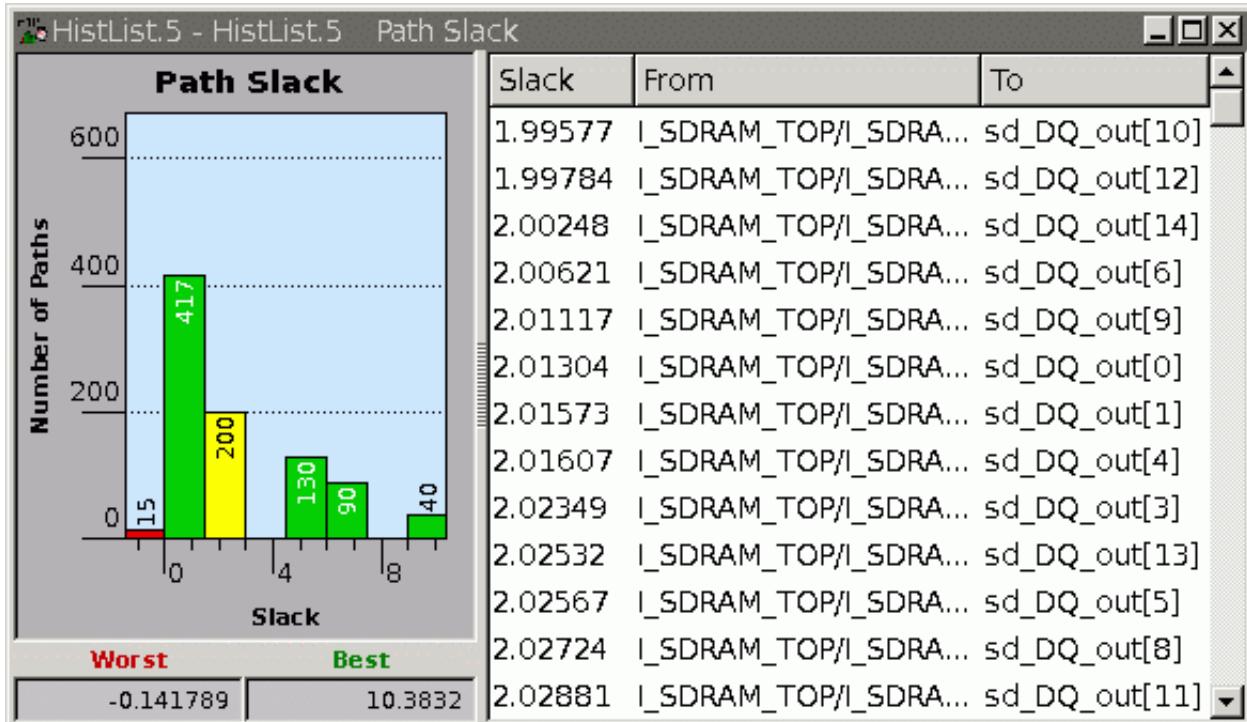


Path slack histograms and endpoint slack histograms are quite similar. Both show the numbers of paths falling within specified ranges of slack values. These are the differences:

- The path slack histogram lists “from-to” paths, possibly including multiple paths to the same endpoint. The number of paths per endpoint (Nworst paths) displayed and the total number of paths per path group (Max paths) displayed are specified in the dialog box.
- The endpoint slack histogram lists all timing path endpoints in the design, showing only the single worst slack per endpoint.

Figure 106 shows an example of a path slack histogram.

Figure 106 Path Slack Histogram



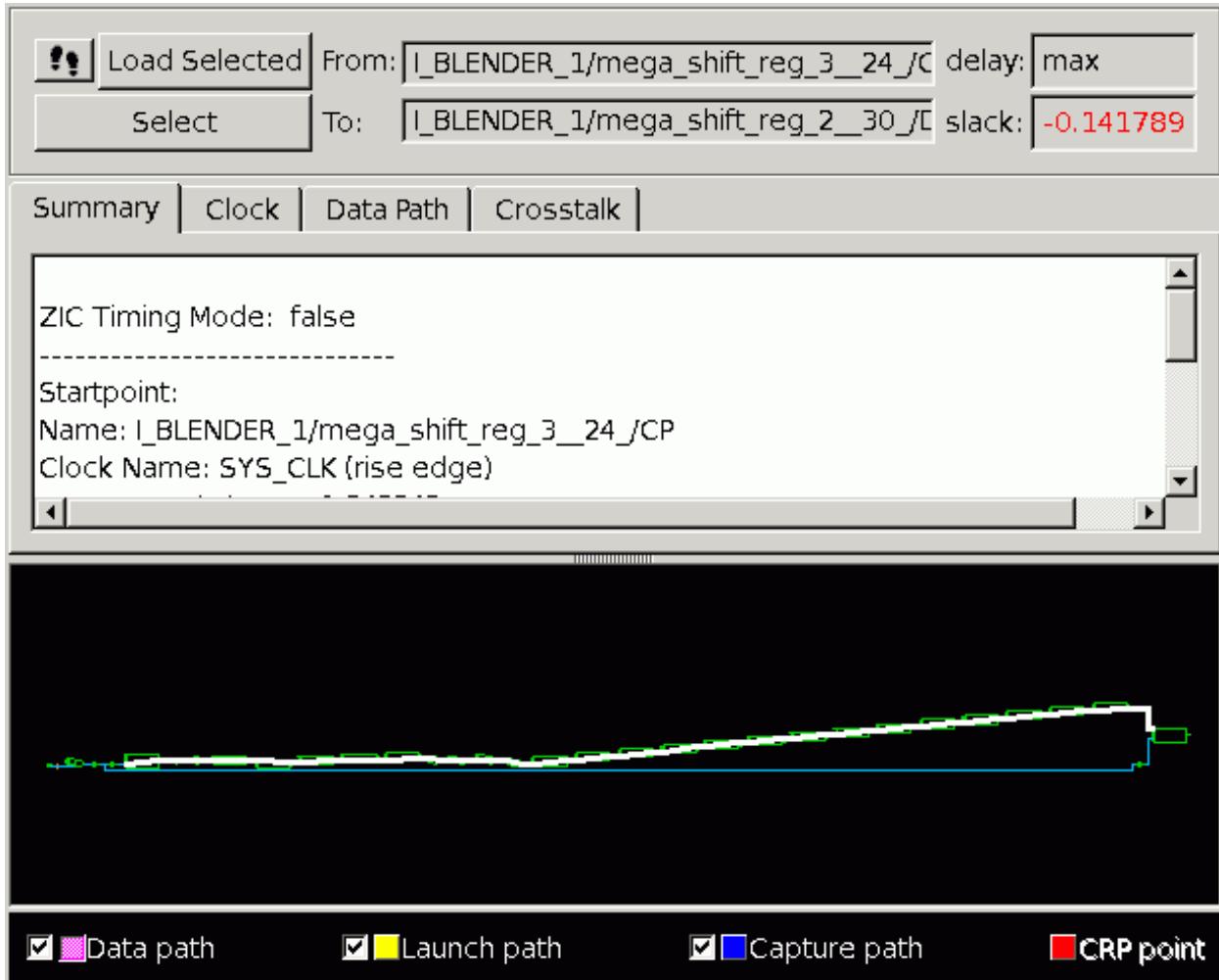
The bars of the histogram show the number of paths that fall into eight ranges (eight bins) of slack values. The leftmost bin, which contains 15 paths, is displayed in red because the slack is negative. The bins with positive slack are displayed in green. The currently selected bin is highlighted in yellow. The slack values, startpoints, and endpoints of the paths in the selected bin are listed in the table on the right, sorted by slack.

To sort the list, click the applicable header at the top of the table. To get detailed information about a path in the list, right-click it and choose Path Schematic, Path Inspector, Timing Paths Report, or Properties. To select a path in the list, click it; this highlights the path in the list, in the schematic view, and in the layout view.

Path Inspector

The path inspector is a tool that displays detailed information about a selected path, including a complete path schematic, clocking information, and timing information. A series of tabs lets you select the type of information you want to examine such as path element tables, a bar-graph delay profile, or path element list. [Figure 107](#) shows a typical path inspector window in IC Compiler. (The path inspector in Design Compiler is somewhat different; for details, see the *Design Vision User Guide*.)

Figure 107 Path Inspector



In the tab panel section, you can click a tab to display the desired information about the path. [Figure 108](#) and [Figure 109](#) show the types of information you can display.

Figure 108 Path Inspector Tabs

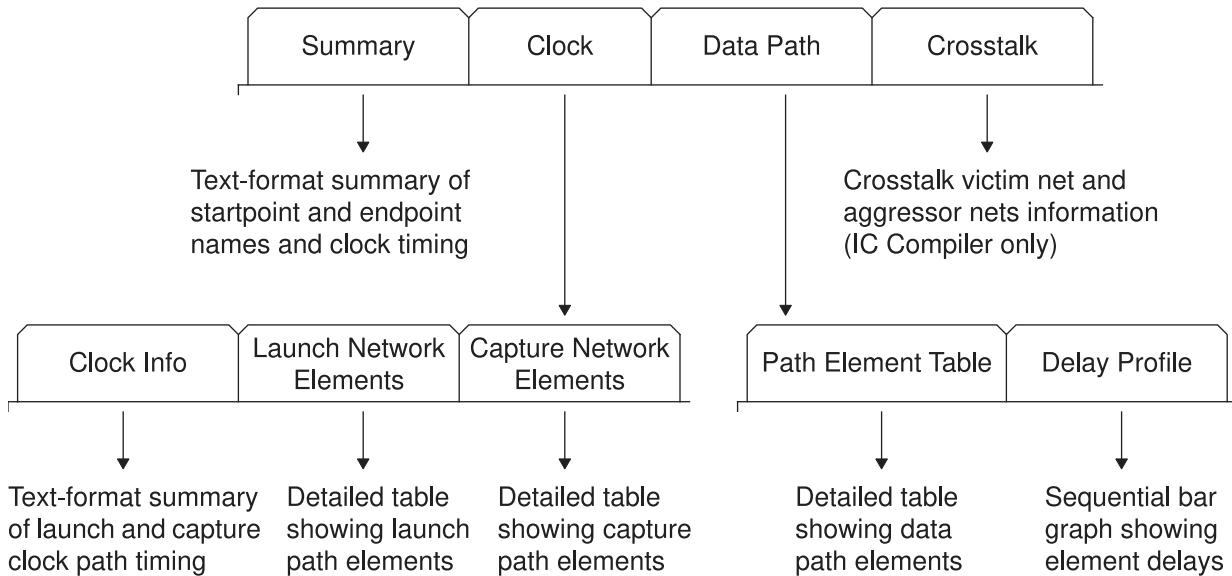
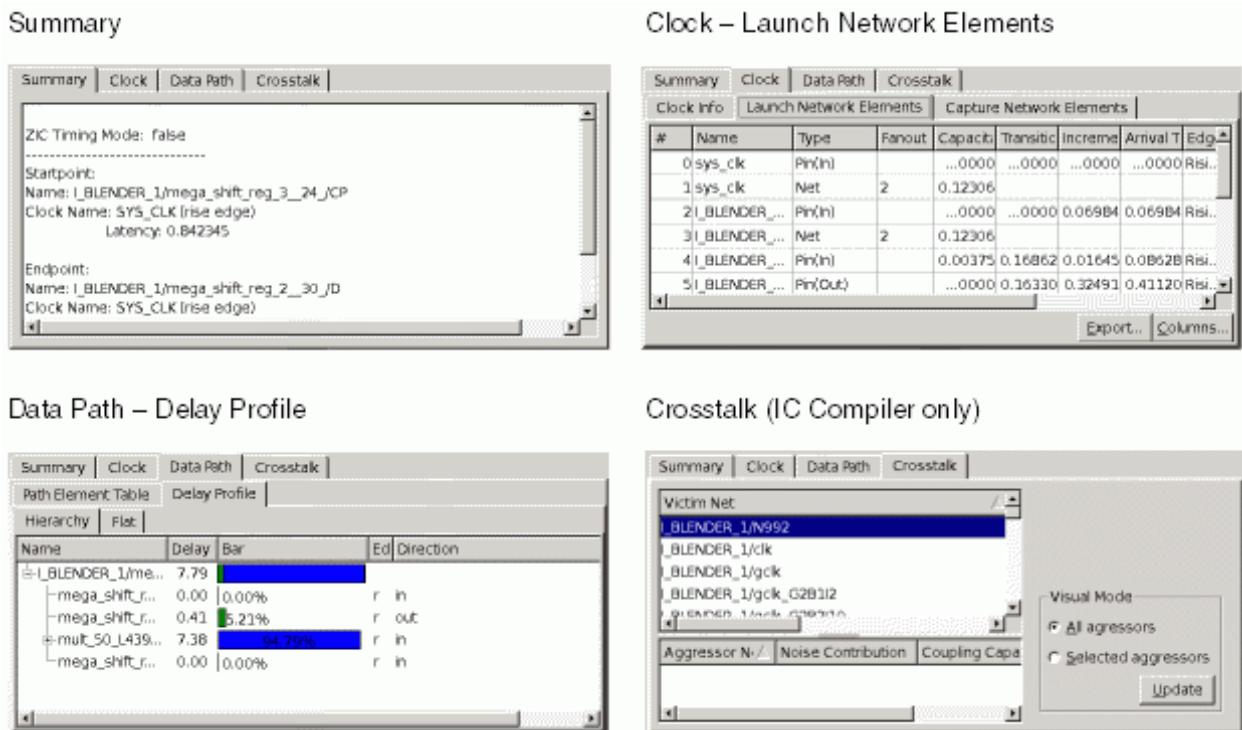


Figure 109 Path Inspector Tab Panel View Examples



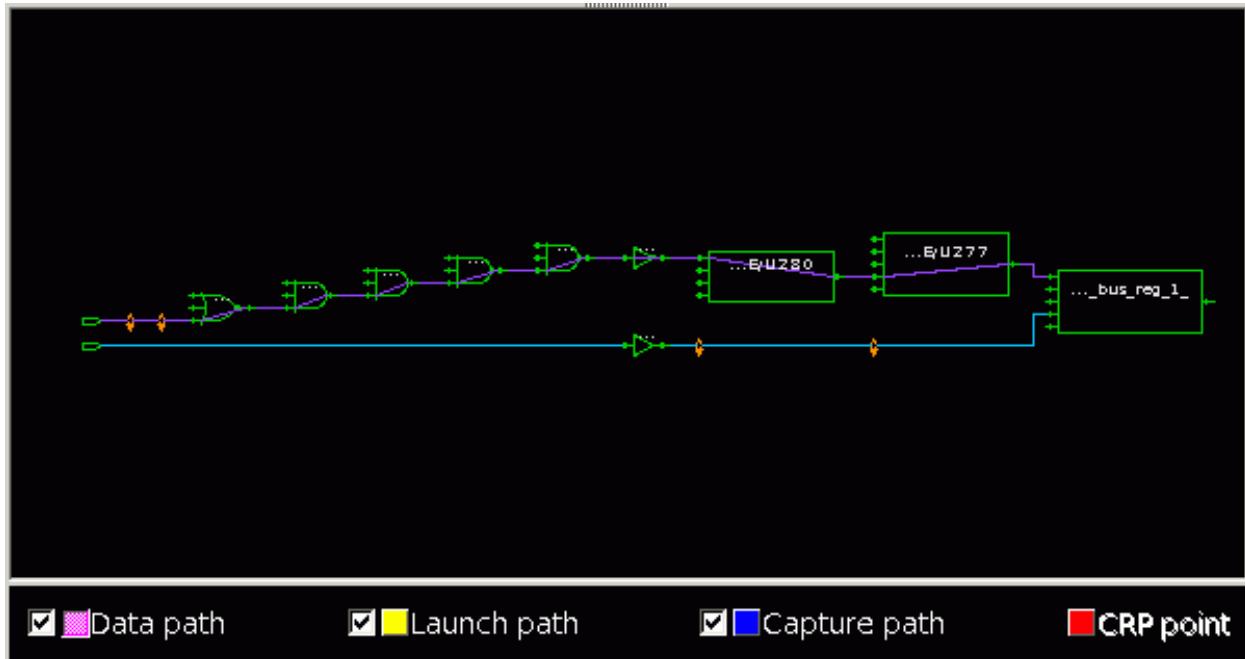
To use the path inspector, first select the path of interest, for example, by choosing Select > Paths From/Through/To or by selecting from a path list in a slack histogram. Then, in Design Vision, choose Timing > Path Inspector. In the IC Compiler timing window, choose Path Inspector > Show Path Inspector. This opens a new path inspector window.

Path Schematic

The schematic in the path inspector of IC Compiler shows the clock launch path and clock capture path. The launch path includes the circuit elements from the input port of the design to the clock pin at the data launch point. Similarly, the capture path shows the circuit elements leading up to the clock pin at the data capture point.

You can choose to display or not display the data path, launch path, and capture path by checking or unchecking the boxes at the bottom of the path schematic. You can also add or remove a colored, point-to-point overlay line to indicate the data path, launch path, or capture path by clicking the purple, yellow, or blue button at the bottom. By default, the data path highlight line is displayed in purple and the others are not, as shown in [Figure 110](#).

Figure 110 Path Schematic in Path Inspector



If the schematic includes the clock reconvergence pessimism (CRP) common point in the launch and capture paths, that point is highlighted with a red dot. You can turn on or turn off the display of this dot by clicking the red button at the bottom.

To get more information about any pin, cell, or net in the schematic, rest the pointer on that object. A pop-up InfoTip displays information such as the object name, capacitance, transition arrival time, and slack.

Path Element Tables

A path element table in IC Compiler provides a detailed listing of elements in a data path, launch path, or capture path. A path element table for a data path is shown in [Figure 111](#).

Figure 111 Path Element Table for a Data Path

The screenshot shows a software interface for viewing path element tables. At the top, there are tabs for "Summary", "Clock", "Data Path" (which is selected), and "Crosstalk". Below these are two sub-tabs: "Path Element Table" and "Delay Profile". A dropdown menu shows "All path elements". The main area is a table with the following data:

#	Name	Reference	Type	Fanout	Capacitance	Transition Time	Increment
0	pperr_n_in		Pin(In)		...0000	...0000	...0000
1	pperr_n_in		Net	3	0.06625		
2	I_PCI_TOP/p...	PCI_TOP	Pin(In)		...0000	...0000	0.03804
3	I_PCI_TOP/p...		Net	3	0.06625		
4	I_PCI_TOP/I...	PCI_CORE	Pin(In)		...0000	...0000	...0000
5	I_PCI_TOP/I...		Net	3	0.06625		
6	I_PCI_TOP/I...	nr03d0	Pin(In)		0.00375	0.19549	0.00393
7	I_PCI_TOP/I...	nr03d0	Pin(Out)		...0000	...9166	0.13859
8	I_PCI_TOP/I...		Net	1	0.00462		

At the bottom are buttons for "Export...", "Report...", and "Columns...".

To display a path element table, use the tabs in the tab panel section of the path inspector as follows:

- Clock > Launch Network Elements for the launch path
- Clock > Capture Network Elements for the capture path
- Data Path > Path Element Table for the data path

By default, elements are listed in path order, from startpoint to endpoint for a data path, or from input port to launch/capture clock pin for a launch path or capture path. To sort the

table according to some other criterion such as incremental delay or capacitance, click the applicable header in the table. An additional click on the same header button resorts the path list in ascending or descending order. You can resize the header buttons horizontally to adjust the widths of the display fields.

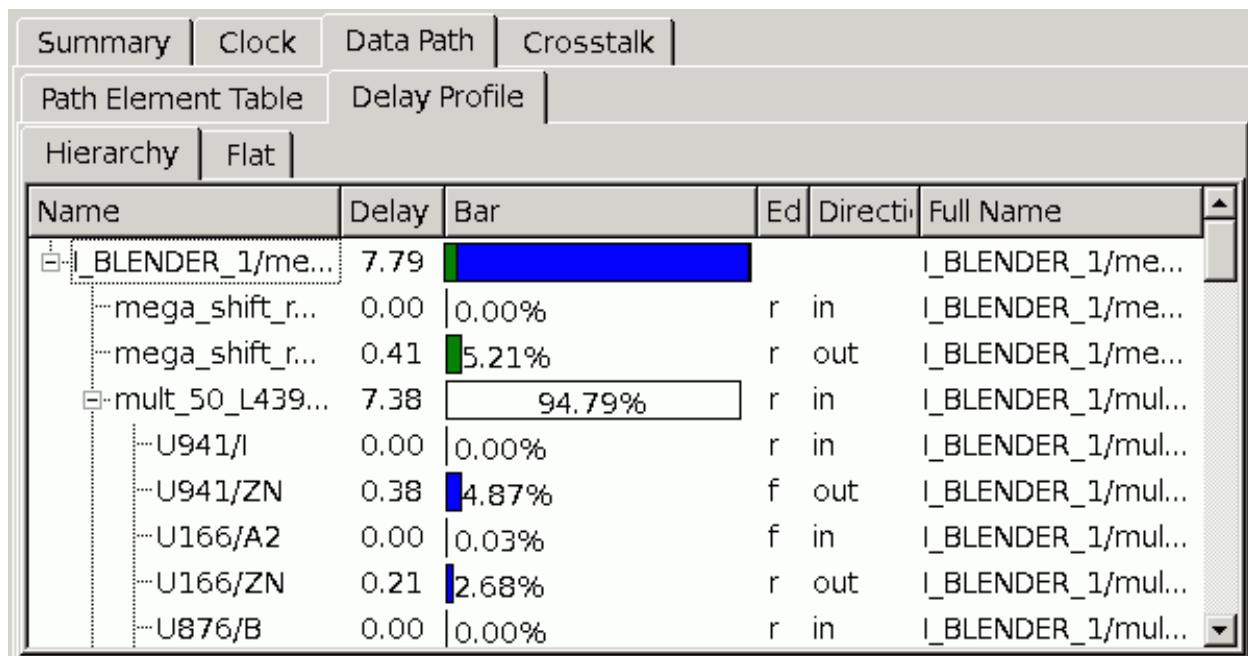
In a data path element table, a pull-down menu at the top of the list lets you specify whether to list nets only, input pins only, output pins only, or some combination of these path elements.

You can select an object in the list by clicking on it. The selected row is highlighted, and the object is also highlighted in any visible schematic or layout view.

Path Delay Profiles

A path delay profile in IC Compiler shows the cell-by-cell and net-by-net distribution of delay along a path in the form of a bar graph, allowing you to visualize the relative importance of different cells and nets contributing to the total delay. To view a delay profile, choose the Data Path > Delay Profile tabs in the tab panel section of the path inspector. See [Figure 112](#).

Figure 112 Hierarchical Path Delay Profile



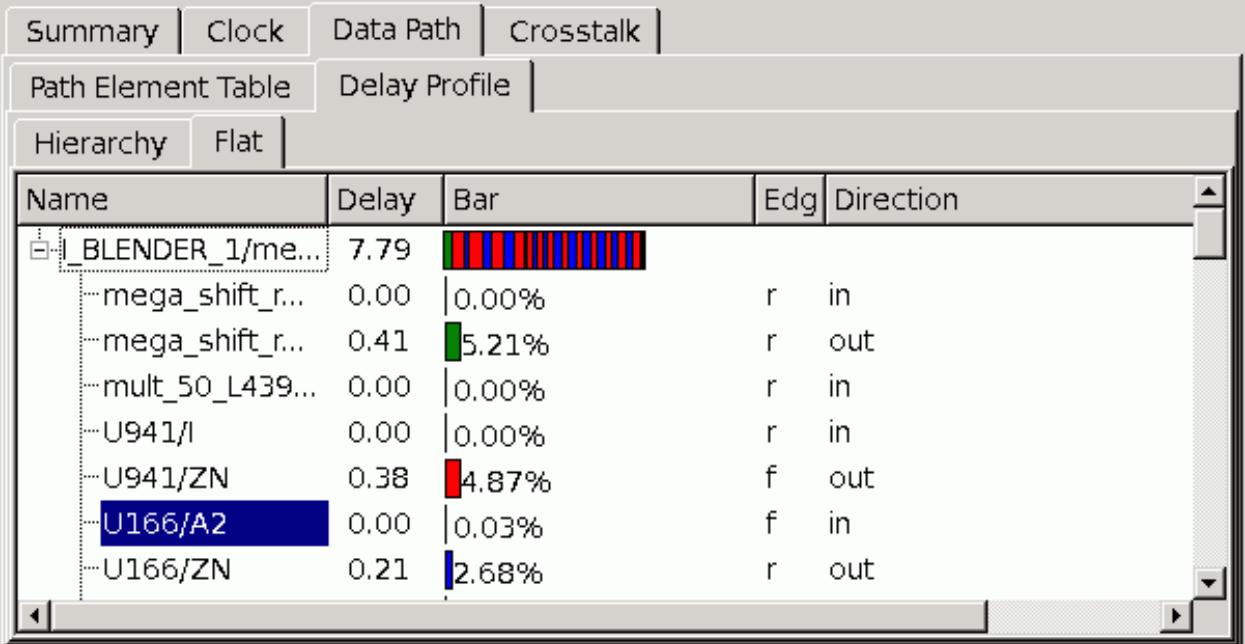
The types of path profiles are: hierarchical, leaf-level pin-to-pin delay, leaf-level cell and net delay, and aggregate cell/net delay. The initial (default) plot is hierarchical. To change the profile type, use the pull-down menu above the graph.

In the bar at the top, each colored segment shows the path delay contribution of each top-level cell and net in the path, starting at the left for the path startpoint and ending at the right for the path endpoint. The individual, color-coded segments are displayed below the full-path profile. The columns adjacent to the plot show the local pin name, delay contribution, edge type (r for rise or f for fall), the signal direction with respect to the pin (in or out), and the full name of the pin.

The default display is a hierarchical profile. The bar at the top shows the top-level cell delays. You can use the plus [+] and minus [-] buttons to traverse the hierarchy and view the delays at different levels.

To view a flat profile instead, choose the Flat tab. In that case, all the leaf-level delays are displayed in the top bar. When you expand the view by clicking the plus [+] button, the individual delays from all hierarchical levels are displayed below it, as shown in [Figure 113](#).

Figure 113 Flat Path Delay Profile



The screenshot shows a software interface for timing analysis. At the top, there are tabs: Summary, Clock, Data Path, Crosstalk, Path Element Table, Delay Profile, Hierarchy, and Flat. The Delay Profile tab is active. Below the tabs is a hierarchical tree view under the Path Element Table tab, showing a node for '_BLENDER_1/me...' which is expanded to show its children: mega_shift_r..., mega_shift_r..., mult_50_L439..., U941/l, U941/ZN, U166/A2, and U166/ZN. To the right of the tree is a table with the following data:

Name	Delay	Bar	Edg	Direction
_BLENDER_1/me...	7.79	███████████		
mega_shift_r...	0.00	0.00%	r	in
mega_shift_r...	0.41	5.21%	r	out
mult_50_L439...	0.00	0.00%	r	in
U941/l	0.00	0.00%	r	in
U941/ZN	0.38	4.87%	f	out
U166/A2	0.00	0.03%	f	in
U166/ZN	0.21	2.68%	r	out

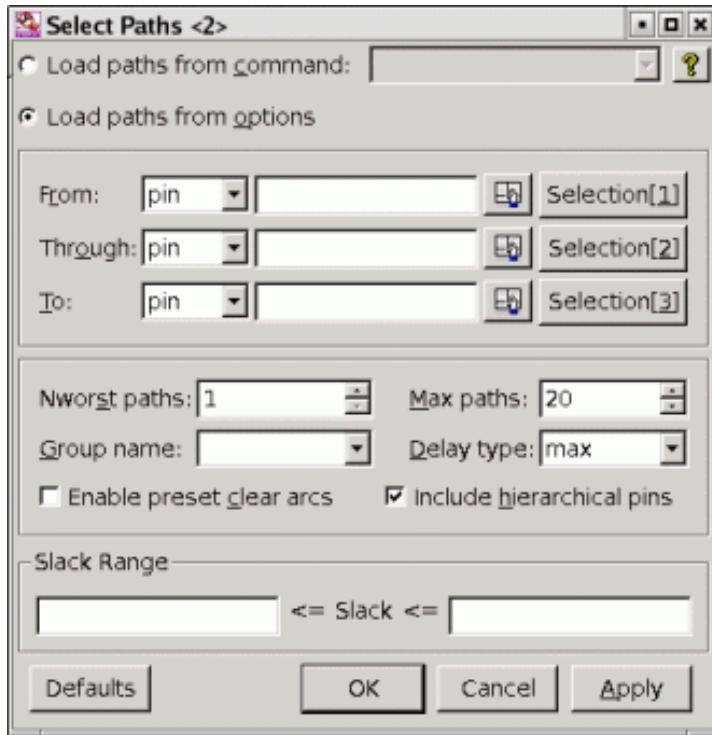
Select a pin in the list by clicking on it. The selected pin is highlighted in the list and also in any visible schematic or layout view.

Timing Analysis Driver

The timing analysis driver is a GUI panel that lists the timing paths having the worst slack in spreadsheet form. You can sort, filter, and modify the data in the table and examine any particular path in detail using `report_timing` reports, schematics, or the path inspector.

To open a timing analysis driver panel in Design Vision, choose Timing > Timing Analysis Driver. In the IC Compiler main window, choose Window > New Timing Analysis Window, or from the IC Compiler layout window, choose Timing > New Timing Analysis window. A dialog box lets you choose which paths to display. See [Figure 114](#).

Figure 114 Timing Analysis Driver Path Selection Dialog Box



By default, the timing analysis driver lists the 20 worst-slack maximum-delay (setup) paths per clock group, listing no more than one path per endpoint. In the dialog box, make any desired changes to the default listing parameters. You can use either “Load paths from options” and fill in the fields of the dialog box, or choose “Load paths from command” and select or enter a command to get exactly the paths that you want, such as `get_selection` or `get_timing_paths`. Click OK to close the dialog box and generate the path list.

Note:

The Design Vision and IC Compiler timing analysis driver panels are slightly different. The Design Vision timing analysis driver panel is described here, but almost all the features described are the same in the IC Compiler GUI.

[Figure 115](#) shows an example of a timing analysis driver window.

Figure 115 Timing Analysis Driver Window

The screenshot shows a Windows-style application window titled "TimingAnalysisDriver.2 - Timing Analysis Driver". The main area is a table with columns: Slack, Startpoint Pin, Endpoint Pin N, Path Group, Endpoint Ck, Arrival, Required, and Requested. The table lists eight paths, all starting from "sel_combo" and ending at various dout[18] through dout[12]. The "Arrival" column shows values like 0.00000, 0.66418, and 0.5. The "Required" column shows values like 8.9, 0.5, and 0.5. The "Path Group" column shows "clk" for all rows. The "Endpoint Ck" column shows "0.00000" for all rows. The "Requested" column shows values like 8.9, 0.5, and 0.5. The "Startpoint Pin" column shows "sel_combo" for all rows. The "Endpoint Pin N" column shows "dout[18]", "dout[17]", "dout[16]", "dout[15]", "dout[14]", "dout[13]", and "dout[12]" respectively. The "Slack" column shows values like -0.22235 and -0.16418. The table has scroll bars on the right and bottom.

Slack	Startpoint Pin	Endpoint Pin N	Path Group	Endpoint Ck	Arrival	Required
-0.22235	coeff_q_reg...	mul_result_r...	clk	0.00000	9.13888	8.9
-0.16418	sel_combo	dout[18]	clk	0.00000	0.66418	0.5
-0.16418	sel_combo	dout[17]	clk	0.00000	0.66418	0.5
-0.16418	sel_combo	dout[16]	clk	0.00000	0.66418	0.5
-0.16418	sel_combo	dout[15]	clk	0.00000	0.66418	0.5
-0.16418	sel_combo	dout[14]	clk	0.00000	0.66418	0.5
-0.16418	sel_combo	dout[13]	clk	0.00000	0.66418	0.5
-0.16418	sel_combo	dout[12]	clk	0.00000	0.66418	0.5

At the bottom of the window, there is a toolbar with buttons: Schematic, Inspector, Histogram..., Report..., Export..., Columns..., and Reload Paths... . Below the toolbar, a command line box contains the text:

```
get_timing_paths -delay_type max -nworst 1 -max_paths 20 -include_hierarchical_pins
```

The text box at the very bottom shows the command used to generate the list of paths. The table shows the following characteristics of each path: Slack, Startpoint Pin Name, Endpoint Pin Name, Path Group, Endpoint Clock Skew, Arrival, Required, Startpoint Clock Name, and Endpoint Clock Name.

By default, the paths are listed in order of increasing slack. To sort the listed paths by some other parameter, click the corresponding header button. An additional click on the same header button resorts the path list in ascending or descending order. You can resize the header buttons horizontally to adjust the widths of the display fields.

To select a path, click its row in the table. The row is highlighted in blue and the selected path is also highlighted in any displayed schematic or layout view. The Shift and Ctrl keys work in the conventional manner to allow multiple selection of table entries.

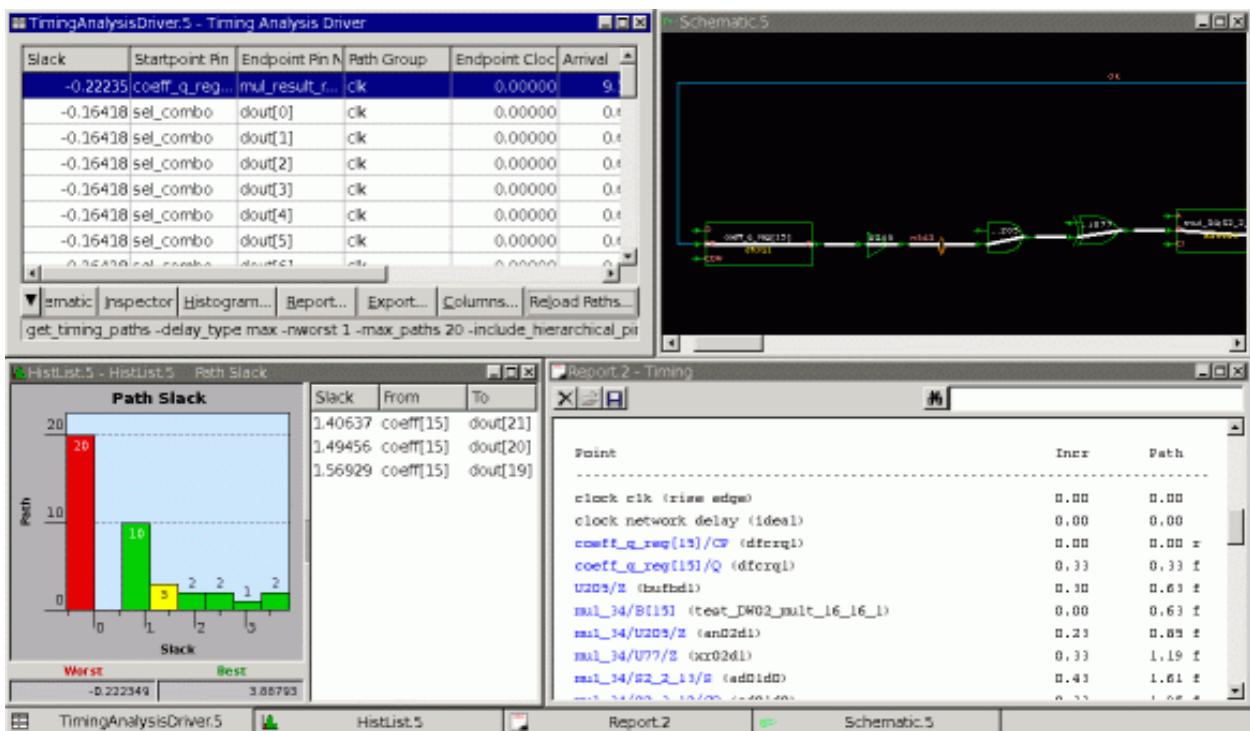
The buttons along the bottom let you perform the following actions:

- Schematic: Open a new schematic window showing the selected path or paths.
- Inspector: Open a path inspector window to get more information about the selected path.
- Histogram: Generate a histogram for the data in one column of the timing path table.
- Report: Generate a timing report (`report_timing` command) for the selected path.

- Export: Save the whole timing path table into an ASCII file, one line per path, with column data delimited by commas. The file can be read into any spreadsheet program.
- Columns: Specify the types of data displayed in the columns of the timing path table and the ordering of those columns.
- Reload Paths: Load a new set of paths from the design that meet the criteria that you specify with the `get_timing_paths` command or by filling in the dialog box options.

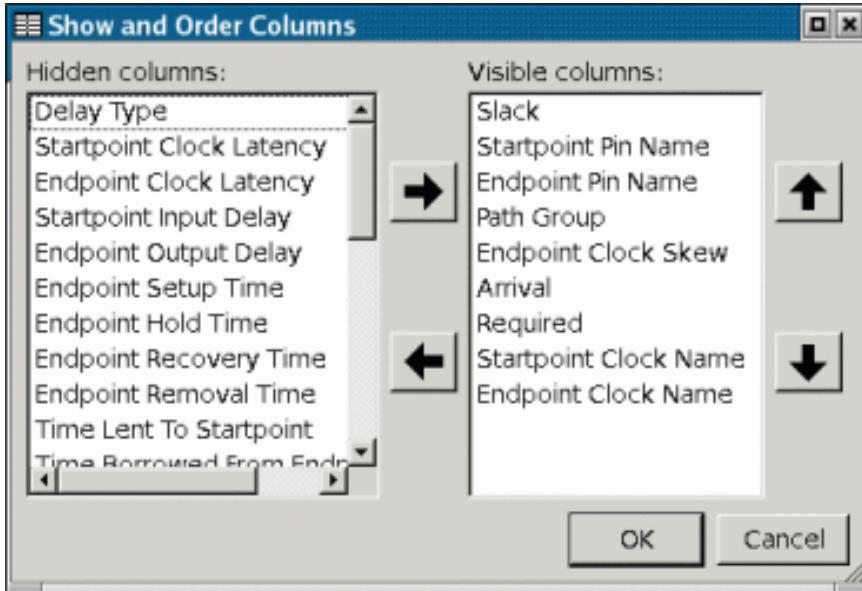
Figure 117 shows an example of a path schematic, a path slack histogram, and a timing report opened from the timing analysis driver.

Figure 116 Analysis Windows Opened From Timing Analysis Driver



To control the types of parameters shown in the timing analysis driver table and their ordering across the table, click the Columns button. This opens the Show and Order Columns dialog box, as shown in [Figure 117](#).

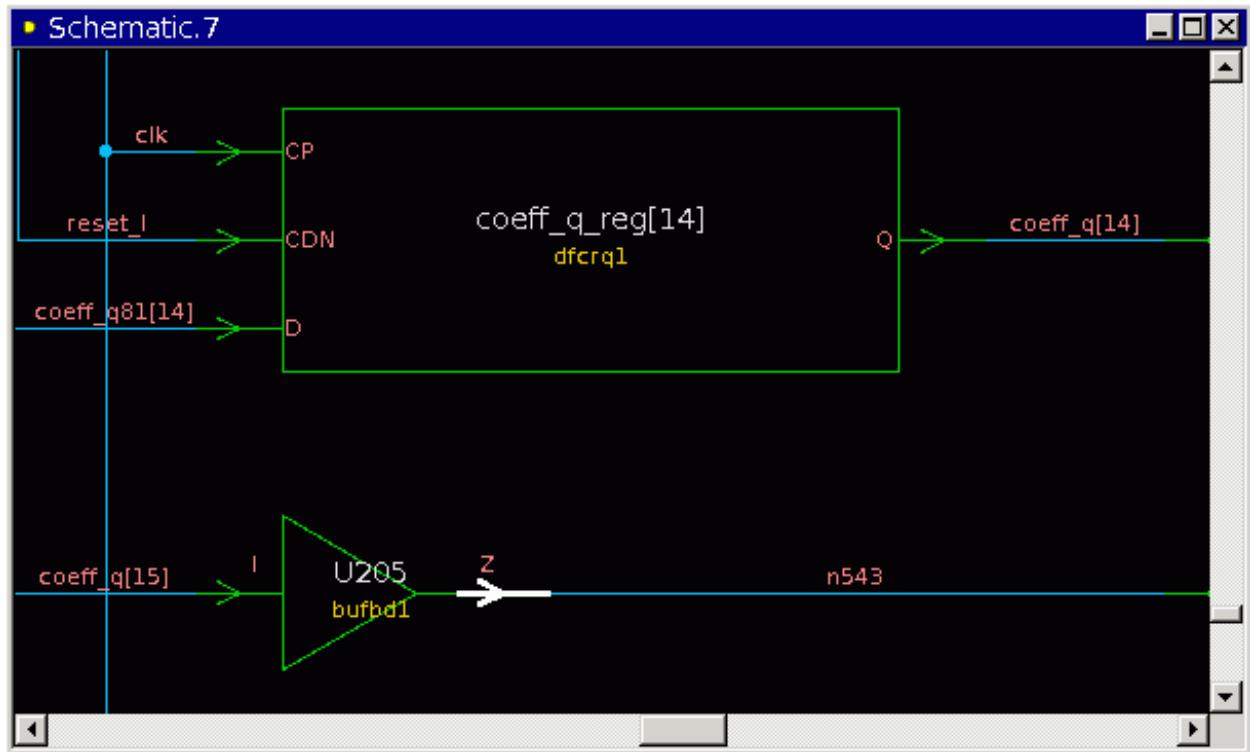
Figure 117 Show and Order Columns Dialog Box for Timing Paths



The list on the right shows the path parameters currently displayed in the table, in order from leftmost column to rightmost column. To control their order across the table, select the desired parameters and then click the up and down arrows to move those parameters within the list. The list on the left shows all the possible path parameters that are not currently displayed. To change the displayed parameters, select the applicable parameters and use the left and right arrow buttons to remove or add entries in the “Visible columns” list. Then click OK.

In the `report_timing` report window, the incremental timing points are highlighted in blue. You can click any of these points to display a schematic of the design with the timing point highlighted. In [Figure 118](#), the timing point U205/Z is highlighted.

Figure 118 Schematic Window Opened From Timing Report



9

Timing in Latch-Based Designs

A latch is a simple, 1-bit level-sensitive memory device. In simulation, a signal holds its value until the output is reassigned. In hardware, a latch implements this holding-of-state capability. Time borrowing can be used in latch-based designs to balance the slack in successive latch path stages, and thereby optimize near-critical paths and reduce delay costs.

This chapter includes the following sections:

- [Time Borrowing in Latch-Based Designs](#)
 - [Constraining a Latch-Based Design](#)
 - [Path Timing Report With Borrowing](#)
 - [Latch-Based Time-Borrowing Example](#)
 - [Advanced Latch Timing Analysis](#)
-

Time Borrowing in Latch-Based Designs

The analysis tool uses time borrowing to analyze the timing of level-sensitive latches. A latch is transparent for the duration of the active clock pulse, meaning that the data appearing on the input of the latch is propagated directly to the output of the latch. If the path leading to the data pin of a latch is too long, one cycle can borrow from the next cycle to extend the time during which the latch is enabled.

A Simple D Latch

To illustrate the processes involved in time borrowing, this section uses a simple D latch. This section includes example VHDL and Verilog code for the D latch and shows the inference report generated when either set of code is compiled.

When you infer a D latch, make sure you can control the gate and data signals from the top-level design ports or through combinational logic. Controllable gate and data signals ensure that simulation can initialize the design.

[Example 3](#) gives the VHDL code for a D latch.

Example 3 VHDL Template for a D Latch

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_latch is
    port (GATE, DATA: in std_logic;
          Q : out std_logic );
end d_latch;

architecture rtl of d_latch is
begin
infer: process (GATE, DATA) begin
    if (GATE = '1') then
        Q <= DATA;
    end process infer;

end rtl;
```

[Example 4](#) gives the Verilog code for a D latch.

Example 4 Verilog Template for a D Latch

```
module d_latch (GATE, DATA, Q);
    input GATE, DATA;
    output Q;
    reg Q;

    always @ (GATE or DATA)
        if (GATE)
            Q = DATA;

end module
```

An inference report contains the information the code compiler passes on to the analysis tool about the inferred devices.

The following table shows the verbose inference report generated for a D latch, using either the VHDL code in [Example 3](#) or the Verilog code in [Example 4](#). Although the coding styles differ, both inferences are the same, and thus they produce the same inference report.

Table 23 Inference Report for a D Latch

Register name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
Q_reg	Latch	1	-	-	N	N	-	-	-

```
Q_reg
-----
reset/set: none
```

For details to help you understand the inference report for Verilog code or VHDL code, see the HDL Compiler documentation.

About Time Borrowing

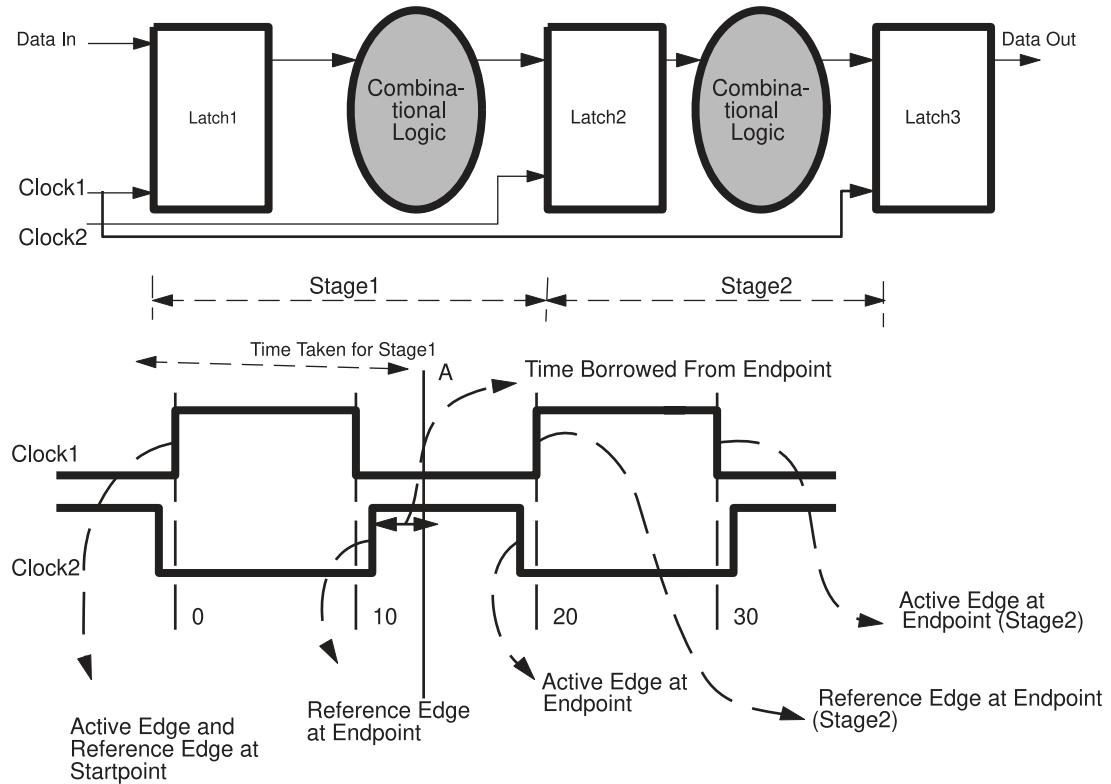
If the path leading to the data pin of a latch is too long, time can be borrowed from the next cycle to enable the latch for a certain duration. This section uses the two-stage latch-based design shown in [Figure 119](#) to explain the concepts that underlie the time-borrowing process.

Each stage of the design consists of a block of combinational logic with a latch on either side. The startpoint of a stage is defined by the latch that precedes the combinational logic block; the endpoint of a stage is defined by the latch that follows the combinational logic block. Thus, as indicated in the bottom illustration of [Figure 119](#), the endpoint of one stage is also the startpoint of the next stage.

In the bottom illustration of [Figure 119](#), the dotted line labeled “Active Edge...at Startpoint” denotes the edge of the clock that renders the startpoint latch transparent—Latch1 in the top illustration of the same figure is the startpoint latch.

The dotted line labeled “Active Edge at Endpoint” denotes the edge of the clock before which the data should stabilize at the input of the endpoint latch—Latch2 in the top illustration of the figure is the endpoint latch.

Figure 119 General Structure of a Latch-Based Design



The reference edge at startpoint—marked by the same dotted line that shows the active edge at startpoint—is the edge of the clock that serves as the reference base for making timing calculations for the startpoint latch, Latch1. The reference edge at endpoint is the edge of the clock that serves as the reference base for making timing calculations for the endpoint latch, Latch2.

In this design, the combinational logic block between Latch1 and Latch2 has more delay than the delay between Latch2 and Latch3. To resolve this discrepancy, the first stage can borrow time from the second clock cycle. In this event, the second clock cycle is left with less time to accommodate the combinational logic block between Latch2 and Latch3.

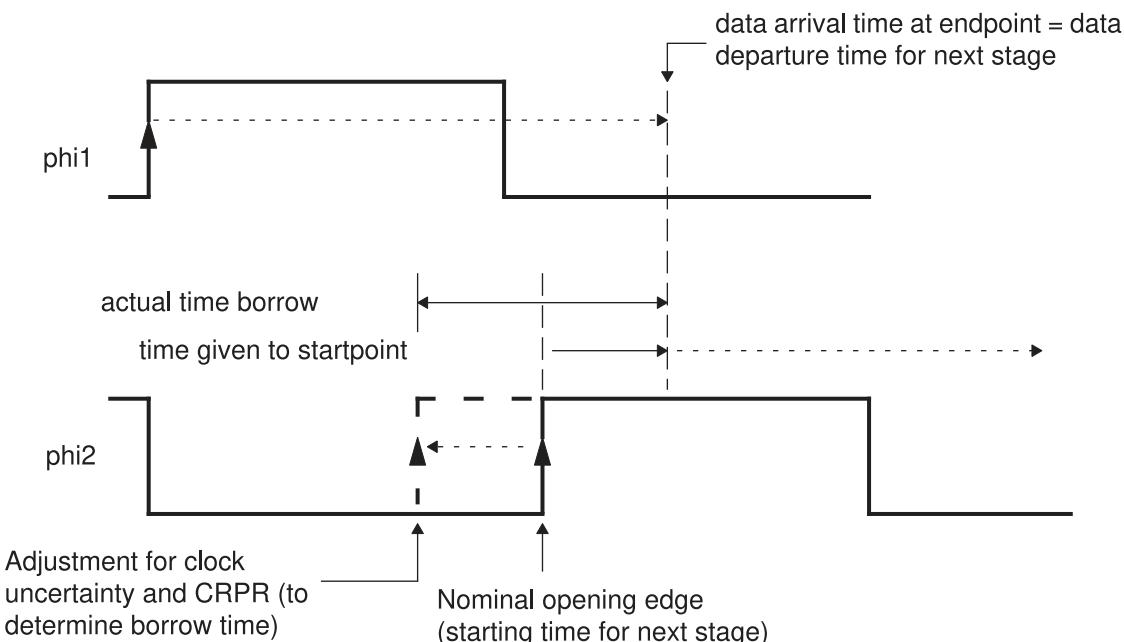
Depending on the requirement, the combinational block between Latch1 and Latch2 can use the time period from the reference edge at startpoint (Latch1) to the line labeled AB; this is the borrowed time Stage1 gets. Initially, this is 50 units of the time used (whether milliseconds, nanoseconds, or another unit of time). Giving Stage1 the borrowed time results in the next combinational block—that is, Stage2—having the time period from AB onward. Thus, the starting point for the Stage2 combinational block becomes AB.

Time Borrowed and Time Given

The tool calculates the amount of time borrowed in relation to the arrival time of the opening clock edge at the latch. The tool 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, the tool 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 120](#).

Figure 120 Borrow Time and Time Given to Startpoint in Transparent Mode



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.

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

Balancing Relative Slacks

This section describes how to balance the relative slacks in two sides of a latch to cause near-critical paths to be optimized and reduce costs.

This section includes the following topics:

- [Determining Relative Slack](#)
- [Optimizing Near-Critical Paths](#)
- [Reducing Delay Costs](#)

Determining Relative Slack

The relative slack of a path is the absolute (negative) slack of the path to which the path belongs, minus the (negative) slack of the critical path in the same path group:

$$(\text{relative slack}) = (\text{absolute slack of path}) - (\text{slack of critical path in path group})$$

The relative slack is a positive number unless the path in question is also the critical path of the path group, in which case the relative slack is zero.

Optimizing Near-Critical Paths

This section explains the relative slacks of both stages of the example latch-based design, based on the initial amount of time borrowed by Stage1 from the Clock2 cycle—that is, before the relative slacks are balanced. Then it explains how to balance these relative slacks.

Relative Slacks of Both Stages Before Balancing

In considering the latch-based design example for whose two stages the relative slacks are shown before balancing, in [Figure 121](#), assume the following values to be true:

- The time borrowed from the endpoint in Stage1 is 50.
- The critical path's slack in the path group for Clock1 is -70.
- The critical path's slack in the path group for Clock2 is -40.

In this scenario, neither Stage1 nor Stage2 is the critical path.

Relative slack is calculated as described in the section [Determining Relative Slack](#). This is how the relative slacks resolve for both stages:

- Relative slack in Stage1 is

$$0 - -40 = 40$$

where

- 0 is the absolute slack
- -40 is the critical path's slack
- 40 is the relative slack

- Relative slack in Stage2 is

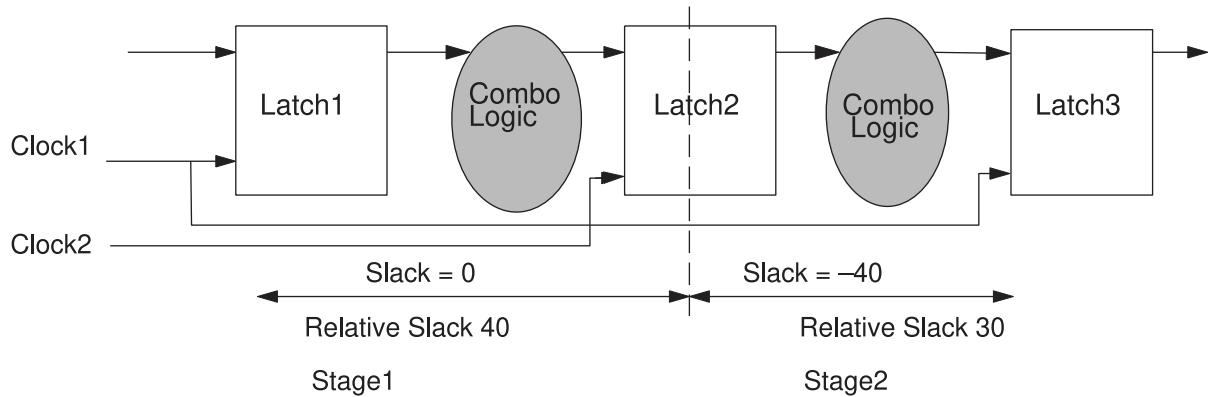
$$-40 - -70 = 30$$

where

- -40 is the absolute slack
- -70 is the critical path's slack
- 30 is the relative slack

The Stage1 relative slack, which is 40, and the Stage2 relative slack, which is 30, are not balanced. To achieve an absolute slack of 0 for Stage1, the absolute slack for Stage2 must be increased—that is, worsened. The tool will not attempt to optimize the Stage1 path even if a critical range is set. [Figure 121](#) shows both stages before the relative slack is balanced.

Figure 121 Both Stages Before Relative Slack Balancing (for Optimizing Near-Critical Paths)



Relative Slacks After Balancing

To attempt to balance the relative slacks, the time borrowed from the Clock2 cycle and given to Stage1 is reduced to 45. This produces an absolute slack of -35 for Stage2 and an absolute slack of -5 for Stage1.

Relative slack is calculated as described in the section [Determining Relative Slack](#). This is how the relative slacks now resolve for both stages:

- Relative slack in Stage1

$$-5 - -40 = 35$$

where

- 5 is the absolute slack
- -40 is the critical path's slack
- 35 is the relative slack

- Relative slack in Stage2 is

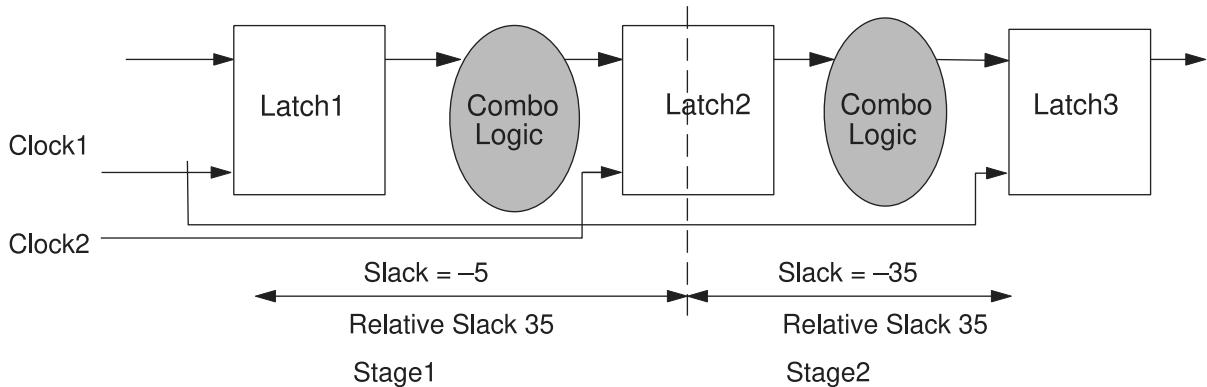
$$-35 - -70 = 35$$

where

- -35 is the absolute slack
- -70 is the critical path's slack
- 35 is the relative slack

This modification of the amount of time borrowed results in Stage1 and Stage2 relative slacks that are balanced. The tool performs this resolution automatically. [Figure 122](#) depicts the result of balancing the relative slacks.

Figure 122 Both Stages After Balancing Relative Slacks (for Optimizing Near-Critical Paths)



Given that Stage1 now has a negative slack, -5 , if you set a critical range that is more than 5 , the tool will optimize the path. Therefore, balancing relative slacks helps in optimizing near-critical paths.

Reducing Delay Costs

This section explains how to reduce delay costs by balancing the relative slacks of both stages of a latch-based design.

Relative Slacks Before Balancing

In considering the latch-based design example for whose two stages the relative slacks are shown in [Figure 121](#) before they are balanced, assume the following values to be true:

- The time borrowed from the endpoint in Stage1 is 60 .
- The critical path in the path group for Clock1 has a slack of -70 .
- The critical path in the path group for Clock2 has a slack of -40 .

Assume, also, that the slacks in some of the near-critical paths in the Clock1 path group are -60 , -50 , and -40 .

Here is how the relative slacks resolve for both stages:

- Relative slack in Stage1

$$-0 - -40 = 40$$

where

- 0 is the absolute slack in Stage1
- -40 is the critical path's slack in path group Clock2
- 40 is the relative slack
- Relative slack in Stage2 is

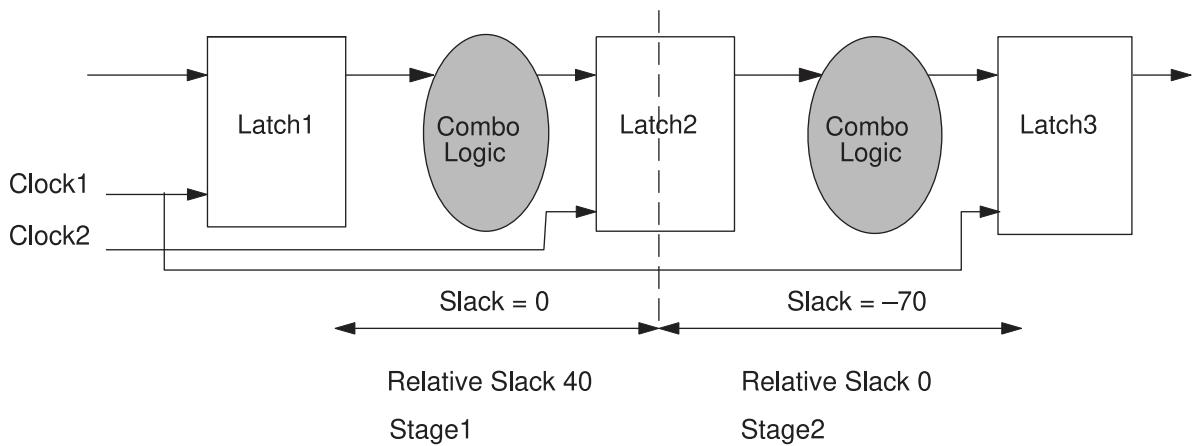
$$-70 - -70 = 0$$

where

- -70 is the absolute slack in Stage2
- -70 is the critical path's slack in path group Clock2
- 0 is the relative slack

As shown in [Figure 123](#), the relative slack in Stage1 is -40 and the relative slack in Stage2 is 0. This results in Stage2 being the critical path of its path group. The resulting delay cost is 110, given a critical path slack in path group Clock1 of 70 and a critical path slack in path group Clock2 of 40.

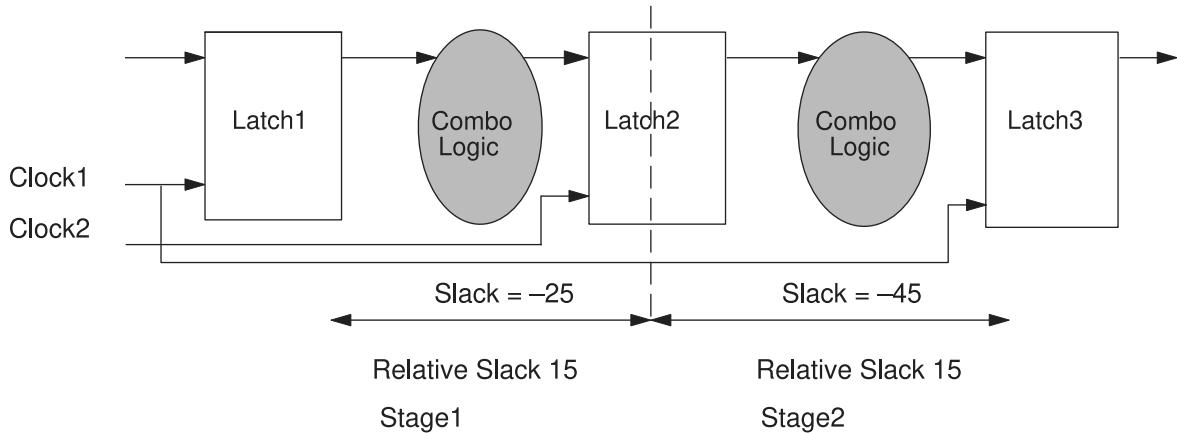
Figure 123 Both Stages Before Balancing Relative Slacks (for Reducing Delay Costs)



Relative Slacks After Balancing

In an attempt to balance the relative slacks of both stages, the time borrowed by Stage1 from the Clock2 cycle is reduced from 60 to 35. Ultimately, this reduces the delay costs from 110 to 100, as shown in [Figure 124](#).

Figure 124 Result of Balancing Relative Slacks for Reducing Delay Costs



Reducing the time borrowed by Stage1 produces the following balanced relative slacks for both stages:

- Relative slack in Stage1

$$-25 - -40 = 15$$

where

- 25 is the absolute slack in Stage1
- -40 is the critical path's slack
- 15 is the relative slack

- Relative slack in Stage2 is

$$-45 - -70 = 0$$

where

- -45 is the absolute slack in Stage2
- -60 is the critical path's slack
- 15 is the relative slack

The critical slack in path group Clock1 is now 60, and the critical path's slack in the path group of Clock2 is now 40.

The relative slack of both stages is now 15, as shown in [Figure 124](#). If the relative slacks had not been balanced, the path in Stage2 would have been treated as the critical path and the near-critical path with a slack of -60 would not have been optimized. Balancing the relative slacks causes the previous near-critical path to become the critical path,

subjecting it to optimization, which is desirable. Moreover, you can easily optimize the previous critical path by setting the proper critical range.

Automatic Slack Distribution During Optimization

By default, the `place_opt` and `route_opt` commands, while performing timing optimization, focus on the worst negative slack at path endpoints. This is a result of “greedy” time-borrowing behavior in which latch time borrowing matches the D pin arrival time, producing zero slack at borrowing latches.

To improve the quality of results, you can optionally have the optimization process distribute the negative or positive slack evenly across all stages of a multistage transparent latch path. This gives the tool an opportunity to work on the earlier stages of transparent borrowing paths, thereby avoiding over-constraining or under-constraining individual stages in each path. Also, spreading positive path slack might improve hold fixing and area during optimization.

To enable slack distribution, set the `timing_enable_slack_distribution` variable to `true`; the default is `false`. Using slack distribution might increase the runtime and memory usage. Note that this variable only affects optimization performed by the `place_opt` and `route_opt` command, not timing reports generated by the `report_timing` command.

Slack distribution, when enabled, overrides automatic time borrowing controlled by the `disable_auto_time_borrow` variable. Automatic time borrowing distributes only the worst negative slack across certain multistage latch paths.

Normalized Slack Analysis

The maximum frequency for a path depends on the delay of the path and the number of clock cycles that a path takes. 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.

The path slack value alone does not completely describe the effect of the path on the clock frequency of the design because some paths take a single clock cycle while others take multiple cycles. For example, [Table 24](#) describes two paths with different slack values. The path with the worse slack (Path B) has a smaller effect on the required clock period. Changing the period to 505 ps increases the allowed length of Path B to 2020 ps, enough to overcome the negative slack of -20 ps.

Table 24 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

Table 24 A Path With Worse Slack Can Have a Smaller Normalized Slack (Continued)

	Path A (Single-segment path)	Path B (Multicycle path)
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
Clock period needed to pass timing	510	505

The allowed propagation delay for a path depends on the number of cycles allowed for the path. The number of cycles depends on the number of latches traversed, as well as the clocks involved in the intermediate latches, and any multicycle paths in path segments along the path.

The tool can help find the paths with the greatest effect on the clock frequency by calculating the normalized slack, a metric for timing paths. The tool calculates the normalized slack for a path as follows:

$$\text{normalized slack} = [\text{path slack}] / [\text{allowed propagation delay for path}]$$

The tool computes the allowed propagation delay for the path using ideal clock edges; it 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.

Enabling Normalized Slack Analysis

To enable normalized slack analysis, perform the following steps before running timing analysis:

1. Enable advanced latch timing analysis:

```
set_app_var timing_enable_through_paths true
```

2. Enabled normalized slack reporting:

```
set_app_var timing_enable_normalized_slack true
```

Reporting Normalized Slack

After running normalized slack analysis, the `report_timing` command can sort paths by normalized slack, as well as find the paths with the worst normalized slack. To do this, specify the `-normalized_slack` option for the `report_timing` command.

Normalized slack values can be much smaller than absolute slack values, so you might need to use the `-significant_digits` option to display enough fractional digits to see the small normalized slack values.

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

$$\Delta\text{period} = -[\text{normalized slack}] \times [\text{period}]$$

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:

$$\Delta\text{period} = -[\text{worst normalized slack}] \times [\text{period}]$$

To find the worst normalized slack, specify the `-normalized_slack` option the `report_timing` command with no other options.

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.

Setting Limits for Normalized Slack Analysis

Normalized slack analysis can potentially take a lot of runtime and memory. You can reduce the impact 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:

```
prompt> set_app_var timing_max_normalization_cycles clock_cycles
```

The default is 4.

When the tool reports the normalized slack for a path, the normalized delay is shown. The normalized delay is the lesser of the allowed propagation delay and the limit, which is the launch clock period multiplied by the `timing_max_normalization_cycles` value.

Constraining a Latch-Based Design

Design Compiler assumes that all external registers are positive edge flip-flops unless they are explicitly created as level-sensitive latches.

Use the following commands to infer external registers:

- `set_input_delay delay_value -clock clock_name input_port`
- `set_output_delay delay -clock clock_name output_port`

These commands set the input delay on input and output ports relative to a clock cycle. The *delay_value* argument specifies the path delay, expressed in units of time consistent with the logic library used during optimization. The *clock_name* argument specifies the clock the stipulated delay pertains to. The *input_port* argument specifies the input port in the current design to which *delay_value* is assigned. The *output_port* argument specifies the output port in the current design to which *delay_value* is assigned.

Specify the `-level_sensitive` option for these commands to define external registers as active-high level-sensitive latches, as shown in the following examples:

```
prompt> set_input_delay 4 -clock CLK2 -level_sensitive [all_inputs]
```

Specifying the `-level_sensitive` option allows the tool to derive the setup and hold relationships for paths from this port, with the presumption that it is a level-sensitive latch.

Specify both the `-level_sensitive` and `-clock_fall` options to define external registers as active-low level-sensitive latches, as shown in the following example:

```
prompt> set_output_delay 3 \
           -clock CLK2 -level_sensitive -clock_fall [all_outputs]
```

Specifying the `-clock_fall` option stipulates that the delay is relative to the falling edge of the clock.

Creating Non-Overlapping Clocks

This section describes non-overlapping clocks, which are used by most two-phase designs. It also explains two-phase designs.

You use the `create_clock` command to create two non-overlapping clocks. Here are examples showing how to specify the `create_clock` command:

```
prompt> create_clock -name \
           CLK1 -period 15 -waveform {9, 14} [get_ports CLOCK1]
prompt> create_clock -name \
           CLK2 -period 15 -waveform {2, 7} [get_ports CLOCK2]
```

What Are Two-Phase Designs?

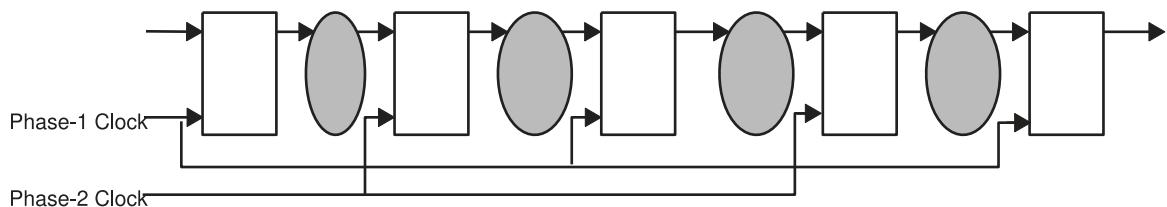
Most latch-based designs use two-phase non-overlapping clocks. These designs are referred to as two-phase designs.

The following requirements apply to two-phase designs:

- All paths originating at a phase-1 latch should terminate at a phase-2 latch. A phase-1 latch is a latch clocked by a phase-1 clock.
- All paths originating at a phase-2 latch should terminate at a phase-1 latch.

Consequently, successive stages of latches should be clocked by alternative clocks, as illustrated in [Figure 125](#).

Figure 125 Two-Phase Design

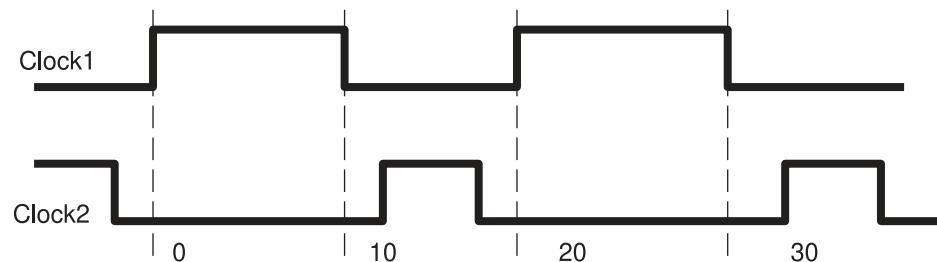


What Are Non-Overlapping Clocks?

Non-overlapping clocks are clocks that don't make a latch transparent simultaneously.

[Figure 126](#) shows two non-overlapping clocks, Clock1 and Clock2.

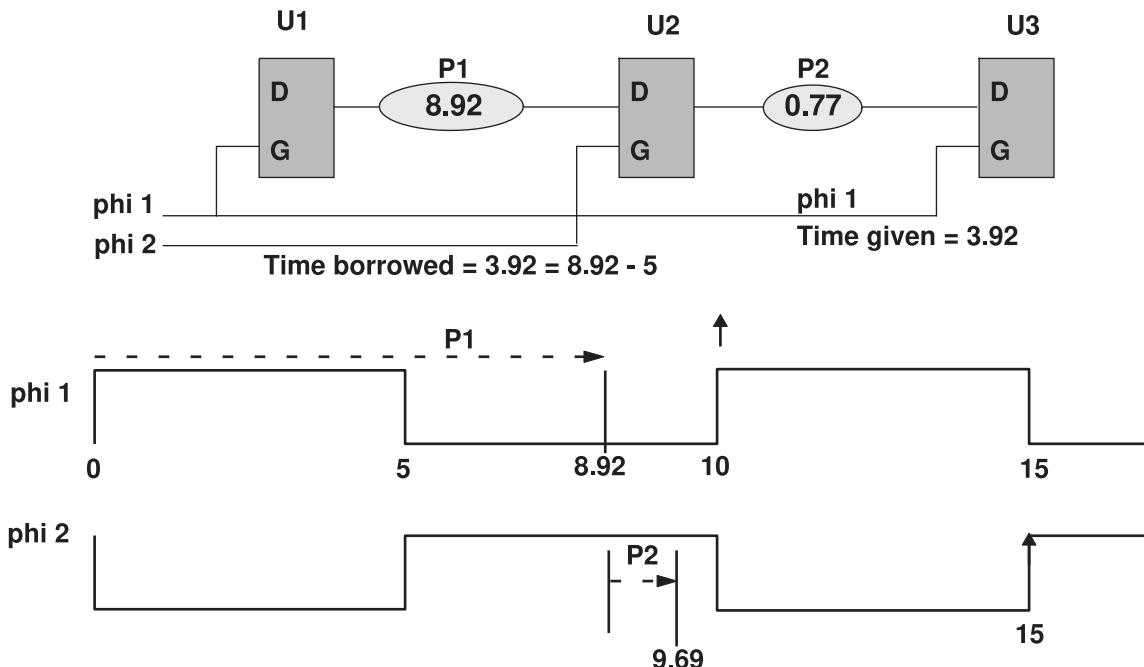
Figure 126 Non-Overlapping Clocks



Path Timing Report With Borrowing

[Figure 127](#) shows a latch-based design using a simple two-phase clocking scheme. The timing report follows.

Figure 127 Latch-Based Design



U1, U2, and U3 are positive level-sensitive latches (active when G=1). P1 and P2 are combinational logic paths. Assuming a library setup time of zero for the latches and path lengths of P1 = 8.92 and P2 = 0.77, it appears that there is a violation at U2. Path P1 is longer than the rising edge of phi2. However, the latch enable time of phi2 is 5, and path P1 can use this time if path P2 has enough slack.

Note:

Although one latch-to-latch path might be greater than the period of the clock, any two successive paths must be less than twice the period for a single-phase design.

Example 5 shows the timing report for the latch-based design. In this report, **bold** type helps you locate the time-borrowing information.

Example 5 Timing Report for a Latch-Based Design

```
*****
Report: timing
        -path short
        -delay max
        -max_paths 1
Design: time_borrow
Version: Y-2006.06
```

Chapter 9: Timing in Latch-Based Designs
 Path Timing Report With Borrowing

```
Date :Mon May 1 2006
*****
Wire Loading Model Mode: enclosed
Startpoint: U1 (positive level-sensitive latch clocked by Phi1)
Endpoint: U2 (positive level-sensitive latch clocked by Phi2)
Path Group: Phi2
Path Type: max
Point           Incr      Path
-----
clock Phi1 (rise edge)      0.00      0.00
clock network delay (ideal) 0.00      0.00
U1/G (LATCH)                0.00      0.00 r
U1/Q (LATCH)                0.57      0.57 r
...
U2/D (LATCH)                8.35      8.92 r
data arrival time            8.92
clock Phi2 (rise edge)      5.00      5.00
clock network delay (ideal) 0.00      5.00
U2/G (LATCH)                0.00      5.00 r
time borrowed from endpoint 3.92      8.92
data required time           8.92
-----
data required time           8.92
data arrival time            -8.92
-----
slack (MET)                 0.00
Time Borrowing Information
-----
Phi2 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 pulse width             5.00
library setup time            -0.49
-----
max time borrow               4.51
actual time borrow             0.00
-----
```

For a positive level-sensitive latch, Design Compiler uses the rising edge as the reference edge.

- For the U1 to U2 path, a setup violation appears at U2, because the signal arrives at 8.92 and the clock arrives at 5.00. Because the U2 latch is transparent, 3.92 ns can be borrowed from the following U2 to U3 path and still meet the setup time at the U2 falling edge. The first timing path shows 3.92 ns being borrowed. This is reported in the timing report for the first path.
- For the U2 to U3 path, time must be added to compensate for the time borrowed, so 3.92 ns is added to the U2 launch time. This is reported in the timing report for the second path. Because the P2 path has enough slack, neither path is a violation.

In some cases, you might not want time borrowing enabled for all or part of a design. You can limit or disable time borrowing by using the `set_max_time_borrow` command. The syntax is

```
set_max_time_borrow time_limit object_list
```

The `set_max_time_borrow` command places a `max_time_borrow` attribute of a specified value on clocks, latch cells, data pins, or clock (enable) pins to constrain the amount of time borrowing for level-sensitive latches. To meet delay targets, time borrowing prevents automatic use of all or part of the enabling clock pulse on a latch. If the attribute is set on a cell and the cell is replaced during optimization, the attribute is moved from the cell to the enable pin.

To prevent time borrowing, specify a time limit of 0.0.

The `max_time_borrow` attribute is ignored when it is placed on clocks that affect no latches, on pins other than latch enable pins, and on cells other than latches.

The increase that can occur in timing analysis runtime for multifrequency designs containing level-sensitive latches occurs because the tool considers all dominant pulse relations for paths where there can be time borrowing. These considerations can be time-

consuming if the clocks are of very different frequencies because there are numerous pulse relations between such clocks.

To prevent time borrowing for the entire design, enter the following command:

```
prompt> set_max_time_borrow 0.0 [all_clocks]
```

To limit time borrowing to 1.2 units for U1/G and U2/G:

```
prompt> set_max_time_borrow 1.2 {U1/G U2/G}
```

To undo a `set_max_time_borrow` command, use the `remove_attribute` command. For example, to undo the previous commands, enter the following command sequence:

```
prompt> remove_attribute [all_clocks] max_time_borrow  

prompt> remove_attribute {U1/G U2/G} max_time_borrow
```

Latch-Based Time-Borrowing Example

This section provides an example of a design for a linear block encoder. The design is composed of the following parts:

- Linear Block Encoder
- Noisy Channel
- Linear Block Decoder for Single Bit Error

The linear block encoder and decoder design, presented in this section, is a latch-based design that uses time borrowing. This section describes the design before giving the code that implements it.

This section includes these topics:

- [Linear Block Encoder and Decoder](#)
- [Noisy Channel](#)
- [Linear Block Decoder for Single-Bit Error](#)
- [Linear Block Encoder and Decoder Implementation](#)
- [Setting Constraints on the Linear Block Encoder](#)
- [report_timing Command Output](#)

Linear Block Encoder and Decoder

[Example 11](#) gives the VHDL code that implements the linear block encoder and decoder. Before you review the code, read this background information about the encoder and decoder implementation.

Here is how the linear block encoder and decoder design works: Each 4-bit message word, denoted as a row vector or 4-tuple

$$D = (d_1, d_2, d_3, d_4)$$

is transformed to a code word C, 7 bits in length

$$(C = (c_1, c_2, \dots, c_7))$$

To achieve this, the linear block encoder adds 3 parity bits to each 4-bit message.

The value of C is generated from the matrix multiplication equation in [Example 6](#).

Example 6 Equation i: Matrix Multiplication

$$C = DG \quad \dots (i)$$

where G is the generator matrix of the code.

[Example 7](#) shows the generator matrix equation.

Example 7 Equation ii: Generator Matrix

$$G = [I_4 \mid P]_{4 \times 7} \quad \dots (ii)$$

where

- I_4 is an identity matrix of order 4.
- P is an arbitrary matrix of order 4 by 3, known as a parity matrix.

The VHDL code example includes these processes:

- P_MATRIX_IN

This process reads the P matrix and generates the G matrix and the HT matrix. The HT matrix is a transposition of the parity check matrix.

- DATA_READ

This process reads in the 4-bit message word.

- LINEAR_BLOCK_CODE

This process generates the Linear Block Code (C) according to equation i, the matrix multiplication equation in [Example 6](#).

Noisy Channel

The noisy channel adds a 1-bit error to the code word C and generates the signal R. The signal is received by the decoder in the format represented by Equation iii in [Example 8](#).

Example 8 Equation iii: Signal R

$$R = C + E \quad \dots \text{(iii)}$$

where E is a 7-bit word with any one of the first 4 bits equal to 1 and all other bits equal to 0. The bit that is set to 1 is the erroneous bit.

Linear Block Decoder for Single-Bit Error

A parity check matrix H is associated with the generator matrix G. [Example 9](#) shows the parity check matrix.

Example 9 Equation iv: Parity Check Matrix

$$H = [PT \mid I_3]_{3 \times 7} \quad \dots \text{(iv)}$$

In the equation given in [Example 10](#), where S is a 3-bit word known as the error syndrome of R for nonzero E, S is one of the rows of the matrix H^T , depending on the erroneous bit of R.

Example 10 Equation v: Error Syndrome

$$\begin{aligned} S &= RHT \quad \dots \text{(v)} \\ S &= 0, \text{ if } E = 0; \end{aligned}$$

That is, if there is an error in the i^{th} bit in R, then the syndrome is the i^{th} row of the H^T matrix. Thus, by comparing the syndrome with the rows of H^T , the error can be detected and the data word D can be recovered.

The SYNDROME_GEN process in the VHDL code shown in [Example 11](#) generates the syndrome according to equation v, the error syndrome equation.

The DECODER process compares the syndrome with the rows of H^T and recovers D.

Linear Block Encoder and Decoder Implementation

[Example 11](#) shows the VHDL code that implements the linear block encoder and decoder described in the preceding sections.

Example 11 VHDL Code for Linear Block Encoder and Decoder

```
library IEEE, SYNOPSYS, WORK;
use IEEE.std_logic_1164.all;
use SYNOPSYS.attributes.all;
```

Chapter 9: Timing in Latch-Based Designs
 Latch-Based Time-Borrowing Example

```

use IEEE.std_logic_arith.all;
use IEEE.std_logic_misc.all;
use WORK.MATRIX_RELATED.all;

entity LINEAR_DECODER is
    port ( DATA_IN : in DATA_WORD;
           ERROR_WORD : in CODE_WORD;
           P_MATRIX : in CHECK_MATRIX;
           CLOCK1, CLOCK2, WRITE_ENAB : in std_logic;
           DATA_OUT : out DATA_WORD );
end LINEAR_DECODER;

architecture BEHAVIORAL of LINEAR_DECODER is

signal D : DATA_WORD;
signal P : CHECK_MATRIX;
signal C, E, R, R1 : CODE_WORD;
signal S : SYNDROME_WORD;
signal G : GENERATOR_MATRIX;
signal H_tran : PARITY_CHECK_MATRIX_TRAN;

begin
    P_MATRIX_IN : process (P_MATRIX, WRITE_ENAB)
    variable G_temp : GENERATOR_MATRIX;
    variable H_tran_temp : PARITY_CHECK_MATRIX_TRAN;
    begin
        if(WRITE_ENAB = '1') then
            P <= P_MATRIX;
            E <= ERROR_WORD;
            for I in K downto 1 loop
                for J in K downto 1 loop
                    if ( I = J ) then
                        G_temp(I)(J) := '1';
                    else G_temp(I)(J) := '0';
                    end if;
                end loop;
                for J in N downto K+1 loop
                    G_temp(I)(J) := P_MATRIX(I)(J-K);
                end loop;
            end loop;
            for I in K downto 1 loop
                for J in N-K downto 1 loop
                    H_tran_temp(I)(J) := P_MATRIX(I)(J);
                end loop;
            end loop;
            for I in N downto K+1 loop
                for J in N-K downto 1 loop
                    if ( I-K = J ) then
                        H_tran_temp(I)(J) := '1';
                    else H_tran_temp(I)(J) := '0';
                    end if;
                end loop;
            end loop;
        end if;
    end process;
end;

```

Chapter 9: Timing in Latch-Based Designs
 Latch-Based Time-Borrowing Example

```

        G <= G_temp;
        H_tran <= H_tran_temp;
    end if;
end process P_MATRIX_IN;
DATA_READ : process ( DATA_IN, CLOCK1 )
begin
    if ( CLOCK1 = '1' ) then
        D <= DATA_IN;
    end if;
end process DATA_READ;

LINEAR_BLOCK_CODE : process ( D, CLOCK2 )
variable C_temp : CODE_WORD;
begin
    if ( CLOCK2 = '1' ) then
        for I in C'range loop
            C_temp(I) := '0';
            for J in D'range loop
                C_temp(I) := C_temp(I) xor ( D(J) and G(J)(I) );
            end loop;
        end loop;
        C <= C_temp;
    end if;
end process LINEAR_BLOCK_CODE;

CHANNEL : process ( C, CLOCK1)
begin
    if ( CLOCK1 = '1' ) then
        for I in C'range loop
            R(I) <= C(I) xor E(I);
        end loop;
    end if;
end process CHANNEL;
SYNDROME_GEN : process ( R, CLOCK2 )
variable S_temp : SYNDROME_WORD;
begin
    if ( CLOCK2 = '1' ) then
        for I in S'range loop
            S_temp(I) := '0';
            for J in R'range loop
                S_temp(I) := S_temp(I) xor ( R(J) and H_Trans(J)(I) );
            end loop;
        end loop;
        S <= S_temp;
        R1 <= R;
    end if;
end process SYNDROME_GEN;

DECODER : process ( S, CLOCK1, R1 )
variable temp,temp1 : CODE_WORD;
variable J : integer;
begin
    if ( CLOCK1 = '1' ) then

```

Chapter 9: Timing in Latch-Based Designs

Latch-Based Time-Borrowing Example

```

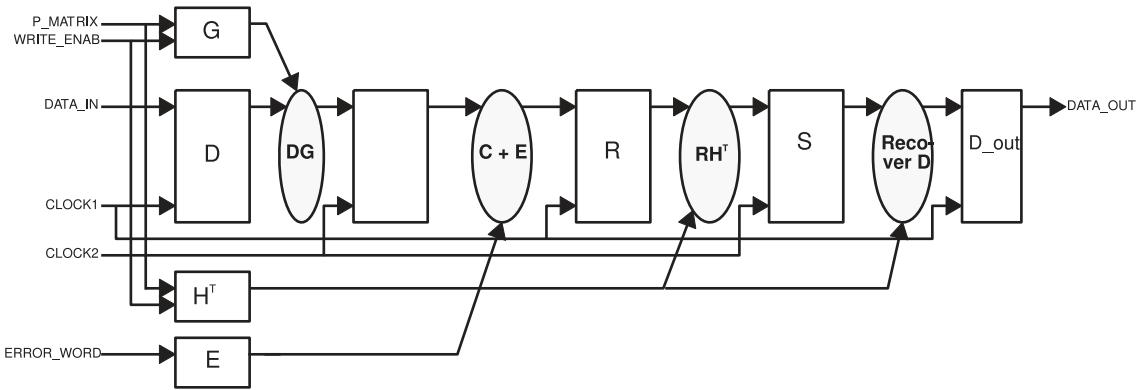
J := 0;
for I in H_Tran'range loop
    if (S = H_Tran(I)) then
        J := I;
    end if;
end loop;
for I in temp'range loop
    if ( I = J ) then
        temp(I) := '1';
    else
        temp(I) := '0';
    end if;
end loop;
for I in R1'range loop
    temp1(I) := temp(I) xor R1(I);
end loop;
for I in K downto 1 loop
    DATA_OUT(I) <= temp1(I);
end loop;
end if;
end process DECODER;
end BEHAVIORAL;
library IEEE, SYNOPSYS;
use IEEE.std_logic_1164.all;
use SYNOPSYS.attributes.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_misc.all;

package MATRIX_RELATED is
    constant K : integer := 4;
    constant N : integer := 7;
    type DATA_WORD is array(K downto 1) of std_logic;
    type CODE_WORD is array(N downto 1) of std_logic;
    type CHECK_WORD is array(N-K downto 1 ) of std_logic;
    type SYNDROME_WORD is array(N-K downto 1) of std_logic;
    type GENERATOR_MATRIX is array(K downto 1) of CODE_WORD;
    type PARITY_CHECK_MATRIX_TRAN is array(N downto 1) of
SYNDROME_WORD;
    type CHECK_MATRIX is array(K downto 1) of CHECK_WORD;
end MATRIX_RELATED;

```

[Figure 128](#) shows the LINEAR_DECODER synthesis produced from the code shown in [Example 11](#).

Figure 128 LINEAR_DECODER Synthesis



Setting Constraints on the Linear Block Encoder

You can use the command sequence shown in [Example 12](#) to set constraints for the linear block encoder and decoder.

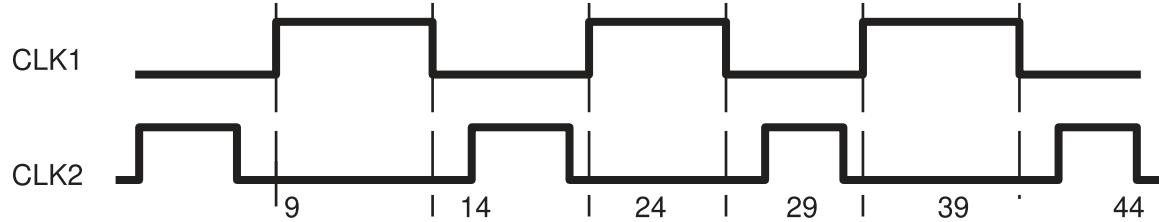
Example 12 Setting Constraints on the Linear Block Encoder

```

set_wire_load_min_block_size "05x05"
set_wire_load_model -library "lsi_10k"
set_operating_conditions -library "lsi_10k" "WCCOM"
set_drive drive_of(lsi_10k/IV/Z) [all_inputs]
set_load load_of(lsi_10k/IV/A) [all_inputs]
set_load 0 find(port, CLOCK*)
create_clock -name CLK1 -period 15 -waveform {9, 14} \
    [get_ports CLOCK1]
create_clock -name CLK2 -period 15 -waveform {2,7} \
    [get_ports CLOCK2]
set_input_delay 4 -clock CLK2 -level_sensitive [all_inputs]
set_output_delay 3 -clock CLK2 -level_sensitive \
    [all_outputs]
set_input_delay 0 [get_ports CLOCK*]
  
```

[Figure 129](#) shows the two-phase clocks for the linear block encoder and decoder produced by the preceding constraints.

Figure 129 Two-Phase Clocks for the LINEAR_DECO



report_timing Command Output

The following example shows a portion of the information produced by the `report_timing` command:

```
prompt> report_timing -delay max -max_paths 20
```

```
Information: Updating design information... (UID-85)
Automatic time borrowing...
```

```
*****
Startpoint: R_reg[3] (positive level-sensitive latch clocked
by CLK1)
Endpoint: S_reg[2] (positive level-sensitive latch clocked
by CLK2)
Path Group: CLK2
Path Type: max

      Point           Incr      Path
-----
clock CLK1 (rise edge)      9.00      9.00
clock network delay (ideal) 0.00      9.00
R_reg[3]/G (LD1P)          0.00      9.00 r
R_reg[3]/Q (LD1P)          2.29     11.29 f
...
S_reg[2]/D (LD1P)          7.16     18.45 r
data arrival time           18.45
                               18.45

clock CLK2 (rise edge)      17.00     17.00
clock network delay (ideal) 0.00     17.00
S_reg[2]/G (LD1P)          0.00     17.00 r
time borrowed from endpoint 0.09     17.09
data required time           17.09
                               17.09

data required time           17.09
data arrival time            -18.45
                               -18.45

slack (VIOLENATED)          -1.36
```

```

Time-Borrowing Information
-----
CLK2 pulse width          5.00
library setup time        -0.60
-----
max time borrow           4.40
actual time borrow        0.09
-----

Startpoint: S_reg[2]
(positive level-sensitive latch clocked by CLK2)
Endpoint: DATA_OUT_reg[3]
(positive level-sensitive latch clocked by CLK1)
Path Group: CLK1
Path Type: max

      Point           Incr     Path
-----
clock CLK2 (rise edge)    2.00    2.00
clock network delay (ideal) 0.00    2.00
time given to startpoint 0.09    2.09
S_reg[2]/D (LD1P)        0.00    2.09 r
S_reg[2]/Q (LD1P)        2.89    4.98 r
...
DATA_OUT_reg[3]/D (LD1)   10.32   15.30 r
data arrival time         15.30

clock CLK1 (rise edge)    9.00    9.00
clock network delay (ideal) 0.00    9.00
DATA_OUT_reg[3]/G (LD1)   0.00    9.00 r
time borrowed from endpoint 4.25    13.25
data required time        13.25

data required time        13.25
data arrival time -15.30

-----
slack (VIOLATED)          -2.05

```

```

Time-Borrowing Information
-----
CLK1 pulse width          5.00
library setup time        -0.40
-----
max time borrow           4.60
actual time borrow        4.25

```

Figure 130 summarizes the `report_timing` output shown in the previous example. The cost is $3.77 - 2.41 + 1.36$ — and the relative slacks in paths II, III and IV are, respectively,

0, 0.36, and 0.11. These relative slacks are almost balanced. Note, however, that the absolute slacks in paths II, III, and IV are not balanced. The absolute slacks are, respectively, -1.36, -2.05, and -1.25.

Figure 130 Timing Report Using Automatic Time Borrowing

Path Group	CLK1	CLK2	CLK1	CLK2
Critical Slack in Path Group	-2.41	-1.36	-2.41	-1.36
Slack in This Path	2.55	-1.36	-2.05	-1.25
Relative Slack in This Path	4.96	0	0.36	0.11

The diagram illustrates a timing analysis for four paths (PATH I, PATH II, PATH III, and PATH IV). The paths are represented by vertical columns of three rectangular boxes each. The top row contains unlabeled boxes, the middle row contains labeled registers (R_reg[3], S_reg[2], and DATA_OUT_reg[3]), and the bottom row contains unlabeled boxes. Horizontal arrows indicate signal flow from left to right between the boxes. Input signals enter from the left. CLK1 is applied to the first box of PATH I and the second box of PATH II. CLK2 is applied to the first box of PATH II and the second box of PATH III. The output of PATH I is connected to the input of PATH II. The output of PATH II is connected to the input of PATH III. The output of PATH III is connected to the input of PATH IV. The outputs of PATH IV and PATH III are connected to the inputs of the middle row boxes (R_reg[3], S_reg[2], and DATA_OUT_reg[3]). The outputs of the middle row boxes are connected to the inputs of the bottom row boxes. The bottom row boxes have arrows pointing to the right, indicating they are the final output stages.

Slacks in two paths that are in different path groups cannot be compared, but relative slack, which is the measure of the criticality of any path, can always be compared.

[Figure 131](#) presents the timing report of the linear encoder and decoder design without the auto time-borrowing feature.

Figure 131 Timing Report Without Using Automatic Time Borrowing

Path Group	CLK1	CLK2	CLK1	CLK2
Critical Slack in Path Group	-3.77	-1.36	-3.77	-1.36
Slack in This Path	2.55	0	-3.38	-1.36
Relative Slack in This Path	6.32	1.36	0.39	0
	PATH I	PATH II	PATH III	PATH IV

As shown in the figure, the cost is $5.13 - 3.77 + 1.36$ —and the relative slacks in paths II, III and IV are, respectively, 1.36, 0.39, and 0. These relative slacks are not balanced. Note that the absolute slacks in paths II, III, and IV are also not balanced. The absolute slacks are, respectively, -1.36, -2.05, and -1.25.

The `disable_auto_time_borrow` variable determines whether the `report_timing` command and other commands perform automatic time borrowing. When the variable is set to false (the default), automatic time borrowing occurs, which balances the slack along back-to-back latch paths to reduce the overall delay cost. Allocating slack throughout the latch stages can improve optimization results.

When the variable is set to true, no slack balancing occurs during time borrowing. This means that the first paths borrow enough time to meet the constraint until the maximum time borrow value is reached, as set by the `set_max_time_borrow` command. This behavior is consistent with the PrimeTime tool.

Results of Use of Automatic Time Borrowing

Use of automatic time borrowing, as shown in the report for the design example, reduces the delay cost from 5.13 ($3.77 + 1.36$) to 3.77 ($2.41 + 1.36$).

Here are some comparisons of delay cost with and without use of automatic time borrowing:

Without use of automatic time borrowing,

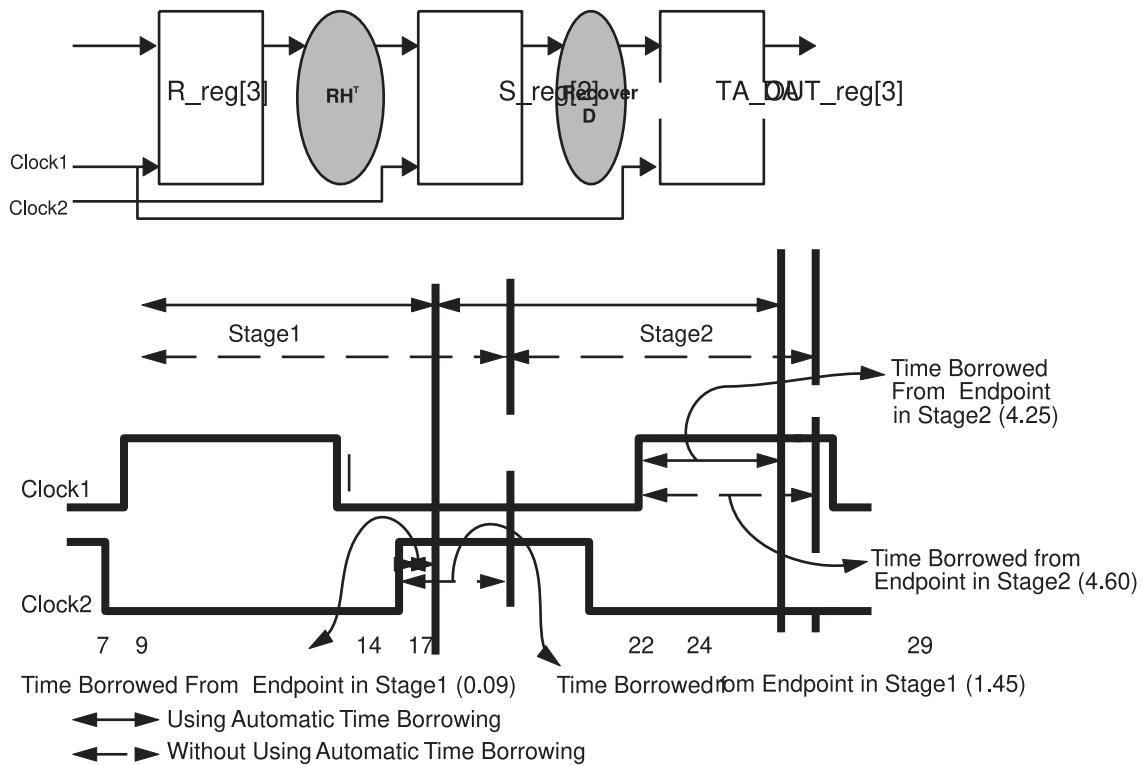
- The path S_reg[3] to DATA_OUT_reg[2] with a slack of -3.77 was the critical path in the path group CLK1.
- The path DATA_OUT_reg[2] to DATA_OUT[1] with a slack of -1.36 was the critical path in the path group CLK2.
- The critical slack in group CLK1 is -3.77, and the critical slack in CLK2 is -1.36.

With use of automatic time borrowing,

- The slack in the path from S_reg[3] to DATA_OUT_reg[2] is reduced to -2.41, from -3.77.
- For the path group CLK2, the slack in the path from DATA_OUT_reg[2] to DATA_OUT[1] is reduced to -1.25 from -1.36. The path from R_reg[3] to S_reg[1] with a slack of -1.36 is now the critical path in group CLK2. (Without use of automatic time borrowing, this path had 0 slack.)
- The critical slack in CLK1 is -2.41, and the critical slack in CLK2 is -1.36.

[Figure 132](#) shows how automatic time borrowing tried to balance the relative slacks in Paths II, III, and IV, which are shown in [Figure 130](#) and [Figure 131](#).

Figure 132 Balancing Relative Slacks by Reducing Borrowed Time



Advanced Latch Timing 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 leakage optimization on latch designs.

You can optionally use advanced latch timing analysis, 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 leakage and design optimization.

Table 25 summarizes the commands and variables for advanced latch analysis.

Table 25 Commands and Variables for Advanced Latch Analysis

Command or variable	Usage
<code>timing_enable_through_paths</code>	Enables advanced latch analysis
<code>set_latch_loop_breakers</code>	Breaks latch loops at specified points, overriding the default points chosen by the tool
<code>get_latch_loop_groups</code>	Returns a list of collections of pins, each collection containing the data pins of a latch loop group
<code>report_latch_loop_groups</code>	Reports information about latch data pins involved in loops of transparent latches
<code>timing_through_path_max_segments</code>	Specifies the maximum number of successive latch path segments analyzed per path

For details about 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 With `report_timing`](#)
- [Calculation of the Worst and Total Negative Slack](#)

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 the tool. The tool tries to select a small set of loop-breaker latches, based on the connectivity of the design. The tool does not consider arrival values when selecting loop-breaker latches.

The `report_timing` command does not find paths through loop-breaker latches. For each `report_timing` command, the tool reports the worst timing path based on path specifiers that does not pass through a loop-breaker latch.

Specifying Loop-Breaker Latches

To manually specify loop-breaker latches, use this command:

```
set_latch_loop_breakers -pin pin_list
```

You should specify a particular latch as a loop breaker if

- The latch is not on a critical path
- The latch is part of a path that is not a loop, but the tool might detect a latch loop (such as a register file with a read and write port, which are never used simultaneously)
- The latch is used with a pulse clock or has only a small transparency window
- There are other latches in a latch loop that should not be treated as loop breakers

Finding Loop-Breaker Latches

To find the loop-breaker latches, query the `is_latch_loop_breaker` attribute on the pins of sequential cells. This attribute is set to `true` for the D pin of a loop-breaker latch. For example, to create a collection of loop-breaker latch pins:

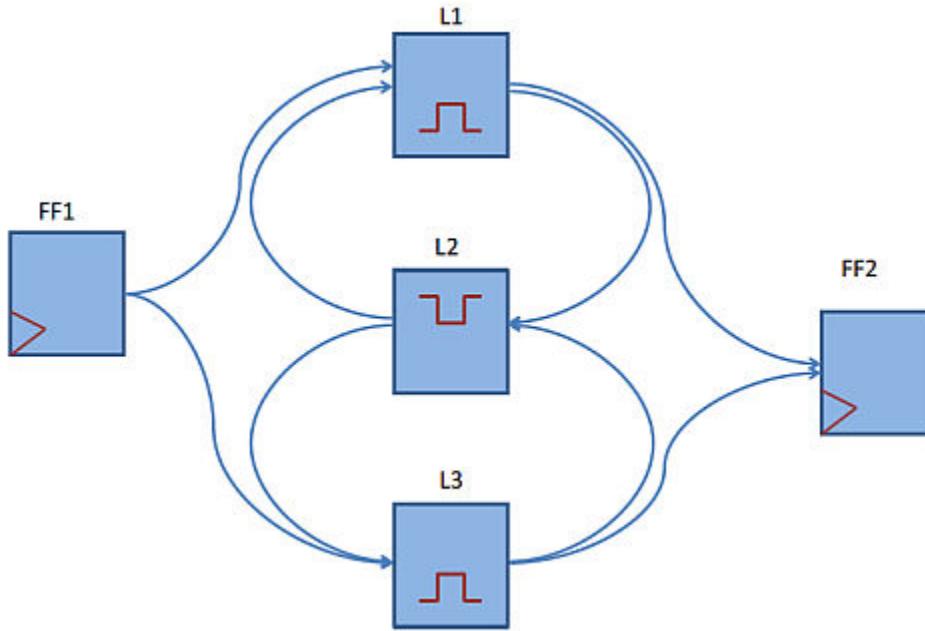
```
get_pins -hierarchical * -filter "@is_latch_loop_breaker"
```

Latch Loop Groups

A single latch can be part of multiple latch loops. The set of all intersecting latch loops is called a *latch loop group*. Every latch in the latch loop group is involved in at least one loop with every other latch in the group.

In [Figure 133](#), 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 133 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 in the design. 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 13](#) 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 13 Report of Latch Loop Groups

```
prompt> 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_breakers
a - user requested to avoid with set_latch_loop_breakers -avoid
Latch Latch Loop Attributes
D pin Group
-----
DUT/Latch1/D NA pu
DUT/Latch2/D 1 b
DUT/Latch3/D 1
DUT/Latch4/D 2 bu
DUT/Latch5/D 2 a
```

If a latch D pin is not in a loop, the latch loop group column indicates “NA”. It is possible for latch D pins that are not in a loop to be path breakers. If you do not use the `-loop_breakers_only` option, the tool does not report these pins.

Specifying the Maximum Number of Latches Analyzed Per Path

By default, the tool reports a maximum of five successive latches per path. The tool limits the maximum length of each path by introducing additional loop-breaker latches even where there are no loops.

To override the default behavior and consider a different maximum number latches per path, set the `timing_through_path_max_segments` variable to the desired number. Set a higher number increase the analysis accuracy for long paths. Set a lower number to decrease the runtime.

A setting of zero means no limit on the number of latches analyzed per path. If this setting causes excessively long runtimes, you should revert to the default setting or use some other reasonable setting.

Timing Exceptions Applied to Latch Paths

When using advanced latch timing analysis, timing exceptions must be satisfied within one path segment. All `-from`, `-through`, and `-to` specifiers must be met in one path segment to be applied.

The following sections describe the application of timing exceptions when running advanced latch timing analysis:

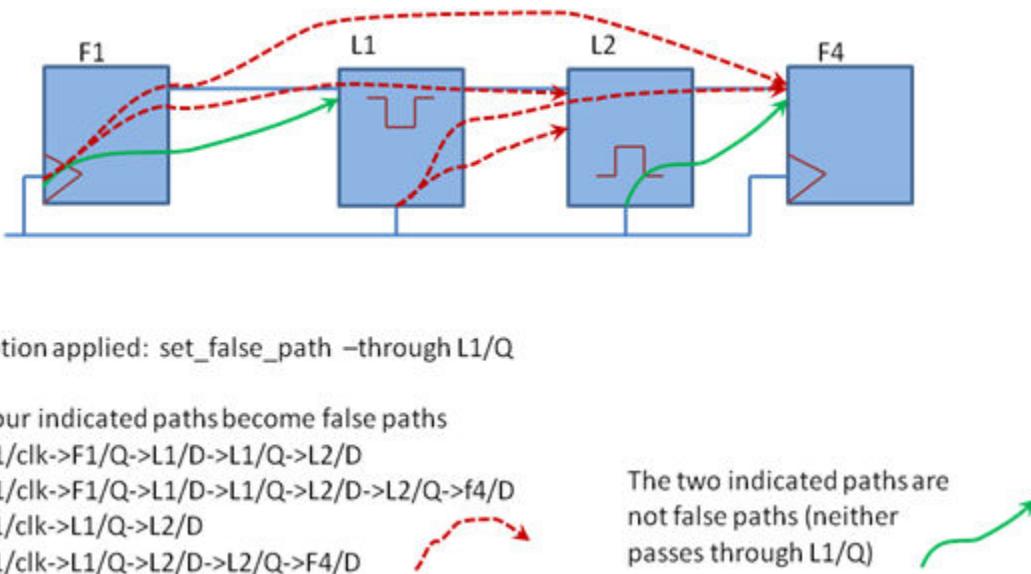
- [False Path Exceptions](#)
- [Multicycle Path Exceptions](#)
- [Maximum and Minimum Delay Exceptions](#)
- [Clock Groups](#)
- [Specification of Exceptions on Throughpaths](#)

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 134](#) shows an example of a false path exception.

Figure 134 False Path Exceptions on Throughpaths

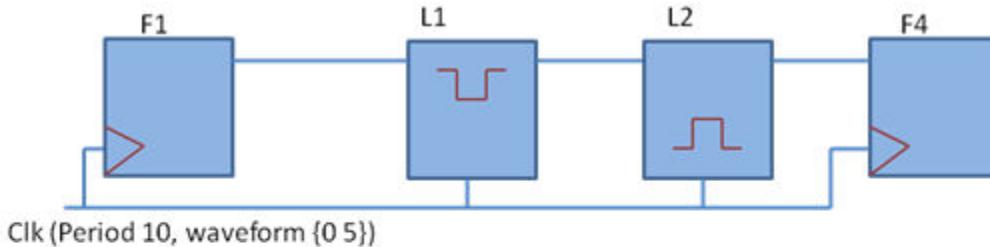


Multicycle Path Exceptions

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 135](#) shows examples of multicycle path exceptions.

Figure 135 Multicycle Path Exceptions Affect the Downstream Latch Windows



Exceptions applied	F1 Launch	L1 Window	L2 Window	F4 Required
No exception	0	5 - 10	10 - 15	20
set_multicycle_path 2 –through L1/Q	0	5 - 10	20 – 25	30
set_multicycle_path 2 –through L1/Q set_multicycle_path 2 –through L2/Q	0	5 – 10	20 – 25	40

In [Figure 135](#), 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 impact normalized slack by changing the allowed propagation delay of paths.

Maximum and Minimum Delay Exceptions

The tool supports maximum delay exceptions for throughpaths when the endpoint of the maximum delay is the D pin of a latch. The exceptions apply a maximum delay to the segment only, overriding the normal pulse relation for the segment. Throughpaths for which the maximum delay exception is not satisfied are not affected, even if they pass through the D pin. For throughpaths that are affected by the exception, the exception affects the local constraint at the latch but does not affect path recovery or normalized slack calculations for the path. Minimum delay exceptions ending at the D pin of a latch affect local hold time constraints for single-segment paths. Throughpaths are not affected because hold checks are not performed for throughpaths.

Clock Groups

Use clock groups to indicate that two clocks do not communicate. You cannot use clock-to-clock false path exceptions to do this because they do not work for all throughpaths;

the false path exception does not conform to the rule that the exception must start and become satisfied within one segment.

Note:

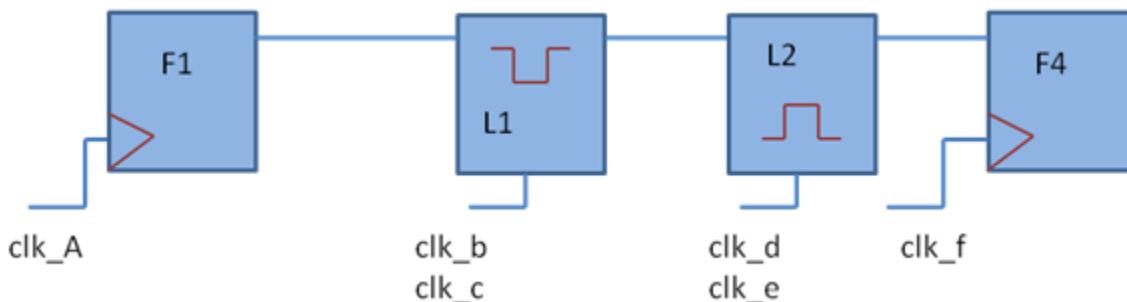
You can use clock-to-clock false paths for hold checks because hold checks are not applied on throughpaths.

The tool applies clock-to-clock exceptions only if the throughpath has a segment that satisfies the exception. For example, if the first intermediate latch is clocked by clkA, and the second intermediate latch is clocked by clkB, then `set_false_path -from clkA -to clkB` causes the throughpath to be false.

Clock groups are taken into account for throughpaths. Throughpaths are not constrained if the throughpath relies on two clocks that are exclusive. This is true even if the throughpath relies on the exclusive clocks only at intermediate latches.

When several clocks arrive at the clock pin of a latch, as shown by the circuit example in [Figure 136](#), potentially distinct throughpaths can be created. Exclusive clock groups are honored by preventing some of the potential throughpaths.

Figure 136 Circuit With Multiple Clocks at Latches



In [Figure 136](#), if no clock groups are set, the following paths are valid from F1/clk to F4/D:

```
F1/clk -> L1/D(clk_b) -> L1/Q -> L2/D(clk_d) -> F4/D
F1/clk -> L1/D(clk_b) -> L1/Q -> L2/D(clk_e) -> F4/D
F1/clk -> L1/D(clk_c) -> L1/Q -> L2/D(clk_d) -> F4/D
F1/clk -> L1/D(clk_c) -> L1/Q -> L2/D(clk_e) -> F4/D
```

Setting the following clock groups reduces the number of valid paths:

```
set_clock_groups -exclusive -group {clk_b} -group {clk_d}
set_clock_groups -exclusive -group {clk_c} -group {clk_e}
```

The following valid paths remain:

```
F1/clk -> L1/D(clk_b) -> L1/Q -> L2/D(clk_e) -> F4/D
F1/clk -> L1/D(clk_c) -> L1/Q -> L2/D(clk_d) -> F4/D
```

Using Clock-to-Clock False Path Exceptions for Setup Only

For clocks that are exclusive, use the `set_clock_groups` command.

If you want to avoid setup checks between two clocks, but you also want to keep the hold checks, use the `set_false_path` exception for a partial solution:

```
set_false_path -from [get_clocks clkA] -to [get_clocks clkB] -setup
```

Single-segment paths launched from clkA and captured by clkB are affected by the exception. Throughpaths are not affected unless there is a segment in the throughpath that satisfies the exception. For example, if the launch clock of the path is clkA, and the first intermediate latch is clocked by clkB, the exception is applied to the throughpath. However, for throughpaths launched by clkA and captured by clkB, but have an intermediate latch clocked by other clocks, the exception is not applied.

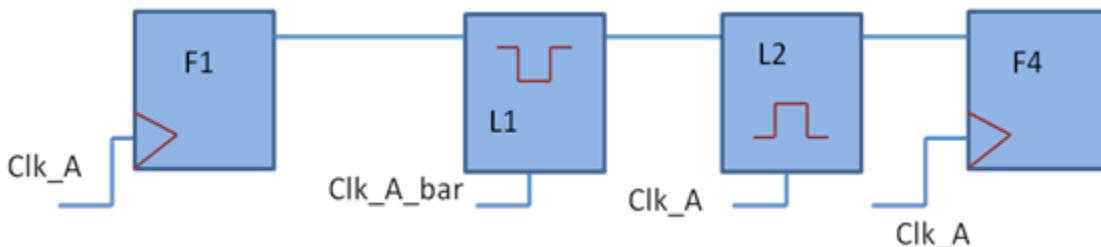
Specification of Exceptions on Throughpaths

Exception specifications that use multiple path specifiers should only use specifiers that are satisfied within one path segment. It is incorrect to specify exceptions with specifiers that are not satisfied in one segment.

For the circuit example in [Figure 137](#), the following exceptions are incorrectly specified and would 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 137 Incorrectly Specified Exceptions on Throughpaths



Exceptions that are incorrectly specified have no effect on timing; to list these exceptions, use `report_exceptions -ignored`.

Reporting Paths Through Latches With `report_timing`

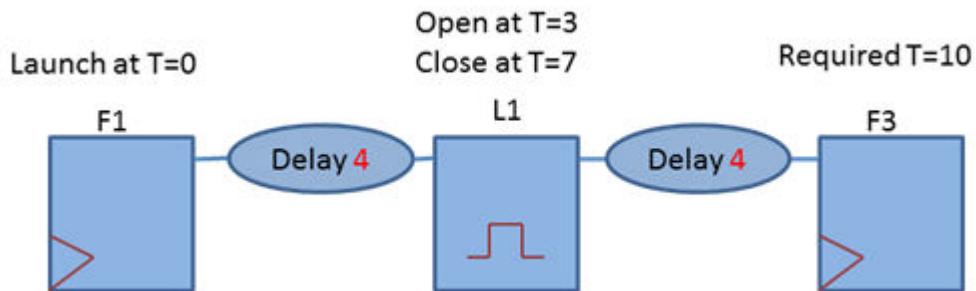
In circuits with latches, the transitive slack provides an estimate of the needed timing improvement for paths through a certain pin. You can find the critical path that determines the transitive slack at a specified pin with the `report_timing` command:

```
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 leakage, the same report indicates the timing path that limits sizing operations at the specified pin.

The report for throughpaths includes a description of each intermediate transparency window. See the report in [Example 14](#), which corresponds to the circuit in [Figure 138](#).

Figure 138 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 14 Timing Report of Paths Through Latches

```
prompt> 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](#).

Calculation of the Worst and Total Negative Slack

The following pin attributes show 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, but not on combinational pins.

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 (for example, `max_fall_slack` or `max_slack`) considers 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 in the design.

The tool calculates the total negative slack (TNS) with the following equation:

TNS = for all timing endpoints in the design, summation of

`maximum(0, minimum(max_rise_local_slack, max_fall_local_slack))`

A

Static Timing Delay Calculation

The timing analyzer of the tool provides accurate static timing information for all timing paths in the current design, allowing the tool to optimize the design and improve the slack of critical paths. A delay model allows the calculation of all delays, including propagation and interconnect delays (net transit time).

The timing analyzer computes each gate and interconnect delay, then traces critical paths, calculating minimum and maximum arrival times to points of interest. The timing analyzer uses the critical path values to evaluate design constraints and create timing reports.

The calculation of delays is described in detail in the following sections:

- [Path-Based Timing Optimization](#)
 - [Reporting Retain Arcs](#)
 - [Reporting Arc Delay Calculations](#)
 - [Delay Models](#)
 - [Waveform Propagation Using CCS Models](#)
-

Path-Based Timing Optimization

The timing analyzer calculates minimum and maximum path delay costs during optimization and allows the tool to make optimization decisions that improve delay cost. The timing analyzer provides fast timing updates as the design is changed during optimization and has advanced features to support the following:

- Multiphase and multifrequency clocking
- Automatic time borrowing for latch-based designs
- User-specified multicycle paths and false paths
- Specific point-to-point path delay requirements

You define port signal timing and clock waveforms, and the timing analyzer determines the required maximum and minimum path delays for each timing path in the design. These requirements are compared with the actual path delay to determine slack, which is the

difference between the actual and required arrival times. You can display design timing analysis results with the `report_timing` command.

The timing analyzer represents a netlist as a directed graph in which nodes in the graph correspond to the pins in the logic network. Edges between nodes (timing arcs) represent two types of connections:

- Net delay – interconnect delay between a driver pin and a load pin in its fanout.
- Cell delay – timing delay between an input pin and an output pin of a cell.

Delay analysis is the calculation of each timing arc's value, which is either a cell delay or a net delay. Rise and fall delay values for a timing path are calculated by addition of the timing arc values, as follows:

positive unate timing arc

Combines rise delays with rise delays and fall delays with fall delays. Examples are an AND gate cell delay and an interconnect (net) delay.

negative unate timing arc

Combines incoming rise delays with local fall delays and incoming fall delays with local rise delays. An example is a NAND gate.

non-unate timing arc

Combines local delay with the worst-case incoming delay value. Non-unate timing arcs are present in logic functions whose output value change cannot be predicted by the direction of the change on the input value. An example is an XOR gate.

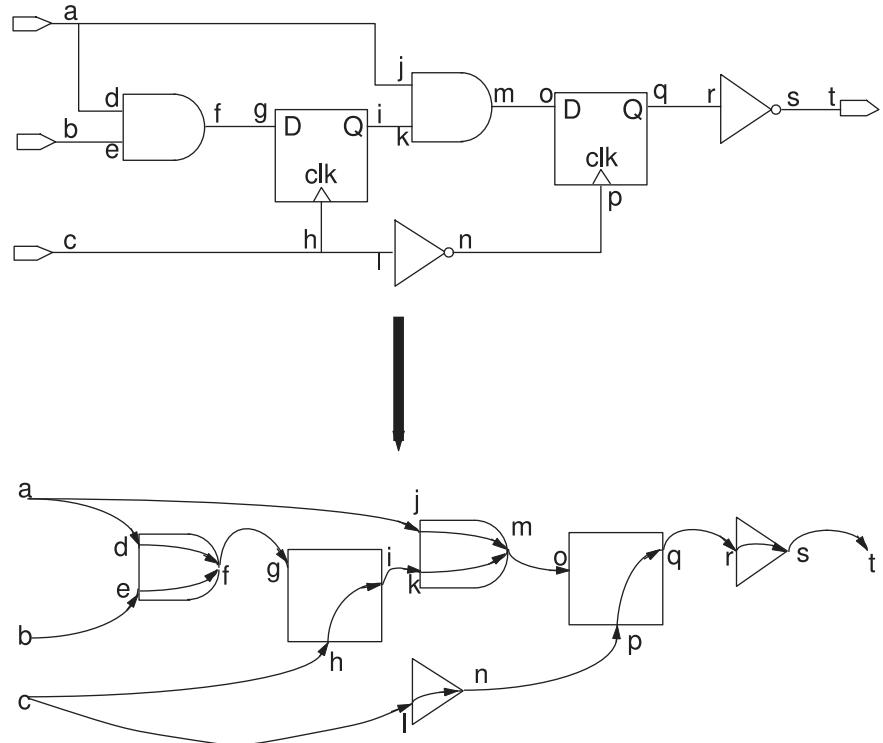
Because the delay attributes are associated with a timing arc and not with a single pin, both minimum and maximum paths between two pins can be modeled.

Note:

The timing analyzer does not store delay equation parameters in terms of specific units. The only restriction the timing analyzer places on the values used is that the units must form an internally consistent system. The recommended units are nanoseconds (delay), picofarads (load), and kilohms (resistance).

[Figure 139](#) shows how a logic network is converted to a timing graph.

Figure 139 Converting a Logic Network to a Timing Graph



The timing arcs in flip-flops are from the clock input (clk) to the data output (Q). No direct timing arcs exist from the D input to the Q output of a flip-flop.

Reporting Retain Arcs

The tool can load retain arcs for timing models from library files, annotate the retain arcs from SDF input files, and report on 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 a 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. It is simply 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, whereas the `write_sdf` command does not.

Reporting Arc Delay Calculations

The `report_delay_calculation` command reports the details of a timing arc delay calculation. The details include cell arcs (from an input pin to an output pin) as well as net arcs (from a source pin to a load pin).

The syntax is

```
report_delay_calculation
  -from from_pin -to to_pin
  [-min|-max]
  [-crosstalk]
  [-from_rise_transition value]
  [-from_fall_transition value]
  [-derate]
  [-nosplit]
```

If a cell timing arc exists between the given pins, the details of the cell arc delay calculation are provided. If the given pins are part of the same net, the details of the net arc delay calculation are provided.

The following conditions constitute an error (no delay information is reported):

- The logic library was not loaded in source format with the `read_lib` command.
- No cell or net delay arc exists between the given pins.
- An undefined pin is specified.
- The pins are associated with a nonleaf cell.
- More than one pin is defined with the `-from` or `-to` option.
- The arc between the defined pins is not supported.

The format of the output varies on the basis of the delay model type and the interconnect delay tree type (for net delay arcs). If delay values have been back-annotated for the defined arc, the annotated delay is given instead of the calculated delay.

Example

This example shows the output for a cell arc and a net arc. The library is `generic_cmos`, and the tree type is `balanced_case_tree`.

From pin: U28/A
To pin: U28/Z

```
arc type : cell
arc sense : unate
Input net transition times: Dt_rise = 0.1458, Dt_fall = 0.0653
```

```
Rise Delay computation:
rise_intrinsic      0.48 +
rise_slope * Dt_rise 0 * 0.1458 +
rise_resistance * (pin_cap + wire_cap) / driver_count
0.1443 * (2 + 0) / 1
rise_transition_delay : 0.2886
-----
Total               0.7686
```

```
Fall Delay computation:
fall_intrinsic      0.77 +
fall_slope * Dt_fall 0 * 0.0653 +
fall_resistance * (pin_cap + wire_cap) / driver_count
0.0523 * (2 + 0) / 1
fall_transition_delay : 0.1046
-----
Total               0.8746
```

```
Net arc output
  From pin:          U28/Z
  To Pin:            U29/A
```

arc type: net
Wire Loading Model Mode: top

Design	Wire Loading Model	Library
RDC_GENERIC	BASIC_ONE	basic
Operating Conditions: BASIC_WORST	Library: basic	
Balanced case tree		
equation : (r_wire/load_count) * (c_pins + c_wire/load_count)		
(0 / 1) * (1 + 0 / 1)		
delay rise, fall : 0 , 0		

Debugging Delay Calculations Along a Critical Path

A typical use of the `report_delay_calculation` command is to assist in debugging the delay calculations along a critical path.

To locate the critical path, use `report_timing -path full -input_pins`. You get a listing similar to the following example.

Point	Incr	Path
input external delay	0.00	0.00 f
i2 (in)	0.00	0.00 f
cell1/i2 (lower1)	0.00	0.00 f
cell1/C/B (AN2)	0.00	0.00 f
cell1/C/Z (AN2)	0.82	0.82 f
cell1/o1 (lower1)	0.00	0.82 f
cell2/i1 (lower2)	0.00	0.82 f
cell2/C/A (IV)	0.00	0.82 f
cell2/C/Z (IV)	0.38	1.20 r
cell2/o1 (lower2)	0.00	1.20 r
o1 (out)	0.00	1.20 r
data arrival time		1.20

The `-input_pins` option causes the input pins to be listed in the path in addition to the output pins, thereby providing information on net delays as well as cell delays.

Reporting Details of a Cell Delay Arc

Here are some examples of how to print details of a cell delay and net delay arcs.

- To print details of a cell delay arc, use the `report_delay_calculation` command by defining the input and output pin of a leaf cell along the path. For example, enter

```
prompt> report_delay_calculation -from cell1/C/B -to cell1/C/Z
```

- To print the details of a net delay arc, give the command a driver and a load pin on the same net along the path (the pins must be associated with leaf cells). For example, enter

```
prompt> report_delay_calculation -from cell1/C/Z -to cell2/C/A
```

This example is valid because both cells are leaf cells.

- The following command is not valid because there is no net delay arc associated with `from_pin` (it is not on a leaf cell):

```
prompt> report_delay_calculation -from cell2/i1 -to cell2/C/A
```

The operating conditions and wire load are taken into account when generating the report data but timing ranges are not because timing ranges typically apply to an entire path (as opposed to a single timing arc).

Delay Models

The timing analyzer uses the timing parameters and the environment attributes described in a logic library to calculate timing delays for designs.

The types of delay analysis are

- Linear (`generic_cmos`)
- CMOS2 (`cmos2`)
- Nonlinear (`table_lookup`)

Note:

The capacitance of all pins on an interconnected wire contributes to the delay. In this section, references to pins or sums over pins include all pins, both driver and input, unless stated otherwise. In many libraries, however, the delay contribution due to the capacitance of an output pin is assigned to other delays and the capacitance is set to zero. For example in [Figure 141](#), the calculation uses capacitance values for the three input pins and none for the output pin. This produces the correct result when the library includes a capacitance of zero for the output pin.

Linear Delay Model

The linear delay equation for computing gate delay values is the sum of four terms:

slope delay (D_s)

The delay incurred from an input pin to an output pin because of a slow logic transition at the input pin.

intrinsic delay (D_I)

The built-in delay from an input pin to an output pin.

transition time (D_T)

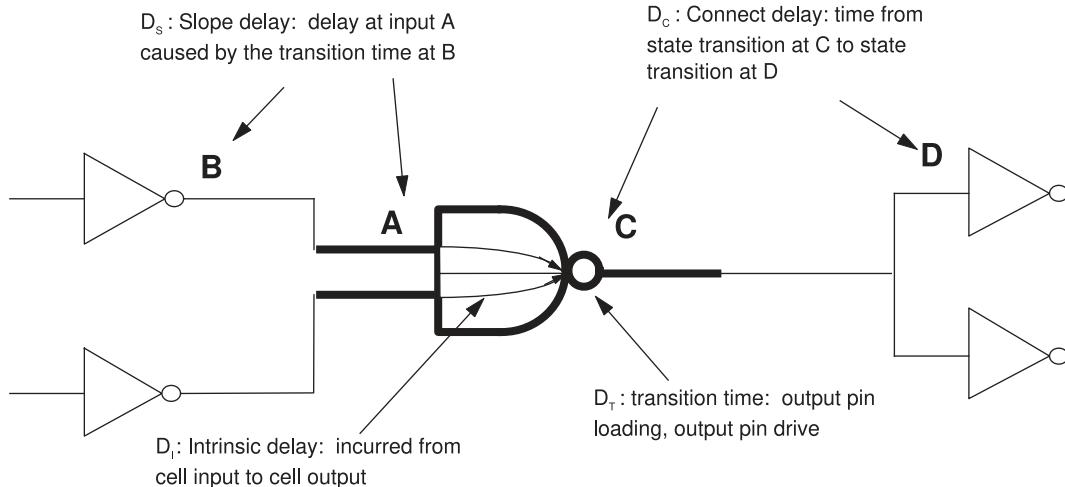
The time a state change takes to complete on a net. This can be constrained as a design rule (`max_transition`) and can also be used as a parameter in delay calculations for cells in the fanout of the net.

connect delay (D_C)

The time-of-flight delay—the time a logic transition takes to propagate through an interconnect network.

[Figure 140](#) shows a diagram of the four terms in the delay equation.

Figure 140 Delay Equation Terms and Timing Arcs (Linear)



The timing analyzer does not store delay equation parameters in terms of specific units. The only restriction the timing analyzer places on the values used is that the units must form an internally consistent system. Synopsys engineers suggest a system of nanoseconds (delay), picofarads (load), and kilohms (resistance).

Example

This example shows the rise and fall delay equations.

$$D_{rise} = D_{slope-rise} + D_{intrinsic-rise} + D_{transition-rise} + D_{connect-rise}$$

$$D_{fall} = D_{slope-fall} + D_{intrinsic-fall} + D_{transition-fall} + D_{connect-fall}$$

The parameters of these equations are

$$D_{slope-rise} = D_{T_{previous_stage}} \times S_{rise}$$

$$D_{slope-fall} = D_{T_{previous_stage}} \times S_{fall}$$

The $D_{T_{previous_stage}}$ value is determined by the arc type.

For rise: arc_type $D_{T_{previous_stage}}$ positive unate rise negative unate fall nonunate max(rise,fall)

For fall: arc_type $D_{T_{previous_stage}}$ positive fall negative unate rise nonunate max(rise,fall)

$$S_{rise} = \text{input rise slope sensitivity}$$

$$S_{fall} = \text{input fall slope sensitivity}$$

$D_{intrinsic-rise}$ = intrinsic rise delay from library

$D_{intrinsic-fall}$ = intrinsic fall delay from library

For non-piecewise:

$$D_{transition-rise} = R_{rise} (C_{pins} + C_{wire}) / (\text{number of non-three-state drivers})$$

$$D_{transition-fall} = R_{fall} (C_{pins} + C_{wire}) / (\text{number of non-three-state drivers})$$

For piecewise:

$$D_{transition-rise} = R_{drivei-rise} \times (C_{pins} + C_{wire}) + Y_{adj_i} / (\text{number of non-three-state-drivers})$$

$$D_{transition-fall} = R_{drivei-fall} \times (C_{pins} + C_{wire}) + Y_{adj_i} / (\text{number of non-three-state-drivers})$$

$$D_{connect-rise} = D_{connect-fall} = \begin{cases} \text{if tree type is best case: } 0.0 \\ \text{if tree type is worst case: } R_{wire} (C_{wire} + C_{pinvp22}) \\ \text{if tree type is balanced: } R_{wire}/N (C_{wire}/N+C_{pin}) \end{cases}$$

R_{wire} = wire resistance

C_{pins} = sum of all pin capacitances on net

C_{pin} = individual pin capacitance value

N = number of pins being driven

$D_{T-rise_previous_stage}$ = transition rise delay at previous stage

$D_{T-fall_previous_stage}$ = transition fall delay at previous stage

Slope Delay

The slope delay of an element, D_S , represents the gate delay resulting from the ramp time of the input signal. A slower input transition results in more slope delay.

$$D_S = D_{Tprevious} \times S$$

$D_{Tprevious}$

The transition time of the previous gate (see [Transition Time](#)). The appropriate transition direction is selected based on the unateness of the timing arc being evaluated.

S

The slope delay factor for the specified timing arc (slope sensitivity).

Intrinsic Delay

The intrinsic delay of an element, D_I , is the portion of the total delay that is independent of the element's usage. This is the fixed (or zero-load) delay of an element.

The total intrinsic delay is calculated by scaling these constant values by their corresponding k-factors (see [Environmental Scaling](#)).

Transition Time

Transition time D_T is the delay the capacitive load on a gate's output pin introduces. It represents the time it takes the output to switch from one logical state to another. The transition time is computed in one of two ways, depending on the logic library:

Linear transition time

$$D_T = R_{drive} \times \left(\sum_{pins} C_{pin} + C_{wire} \right)$$

Piecewise linear transition time

$$D_T = R_{drivei} \times \left(\sum_{pins} C_{pin} + C_{wire} \right) + Y_{adj}$$

In the piecewise linear model, the resistance value (R_{drivei}) and a constant term (Y_{adj}) can vary with different loading conditions. The `piece_type` statement in the logic library determines how the appropriate resistance and constant are selected. The selection is done on the basis of one of the following criteria:

- Total net length
- Total output capacitance
- Output pin capacitance
- Output wire capacitance

A `piece_define` statement in the library determines the correlation between resistance and constant term values and the `piece_type`. In the case of parallel drivers, the arc transition time is divided by the number of non-three-state drivers.

The total transition time is calculated by scaling the constant values by their corresponding k-factors (see [Environmental Scaling](#)).

Connect Delay

The connect delay, D_C , is the time it takes the voltage at an input pin to change after the transition of the driving output pin. Connect delay is also called the time-of-flight delay (the time it takes for a waveform to travel along a wire). The way this delay is calculated is

important in the analysis of interconnect network delay. The timing analyzer supports three cases for an estimated interconnect topology (tree type).

- Best case (`best_case_tree`) models the case where the load pin is physically adjacent to the driver. All wire capacitance is incurred, but none of the wire resistance must be overcome. The best-case connect delay is calculated from the following equation. Because R_{wire} is always 0 in this case, the resulting D_C is always 0.

$$D_{Cbest} = R_{wire}(C_{wire} + C_{pin}) = 0$$

- Balanced case (`balanced_tree`) models the case where all load pins are on separate, equal branches of the interconnect wire. Each load pin incurs an equal portion of the wire capacitance and wire resistance.

$$D_{Cbalanced} = \frac{R_{wire}}{N} \left(\frac{C_{wire}}{N} + C_{pin} \right)$$

- Worst case (`worst_case_tree`) models the case where the load pin is at the extreme end of the wire. Each load pin incurs both the full wire capacitance and the full wire resistance.

$$D_{Cworst} = R_{wire} \left(C_{wire} + \sum_{pins} C_{pin} \right)$$

The three previous equations are used for calculating both the rise and fall delays. The components of these equations are described in the following paragraphs. Where applicable, use the `rise_` parameter for calculating the rise delay and the `fall_` parameter for calculating the fall delay.

R_{wire}

Estimated wire resistance on the net determined by the wire load model. Wire length is computed with a global estimation function whose parameter is the number of fanout pins on the net being estimated.

C_{wire}

The estimated wire capacitance for the net attached to the head of the timing arc for which the delay value is being computed. Wire length is computed with the actual number of fanout pins on the net being estimated and the `fanout_length` specifications in the `wire_load` group. The estimated value is scaled by the capacitance factor.

C_{pin}

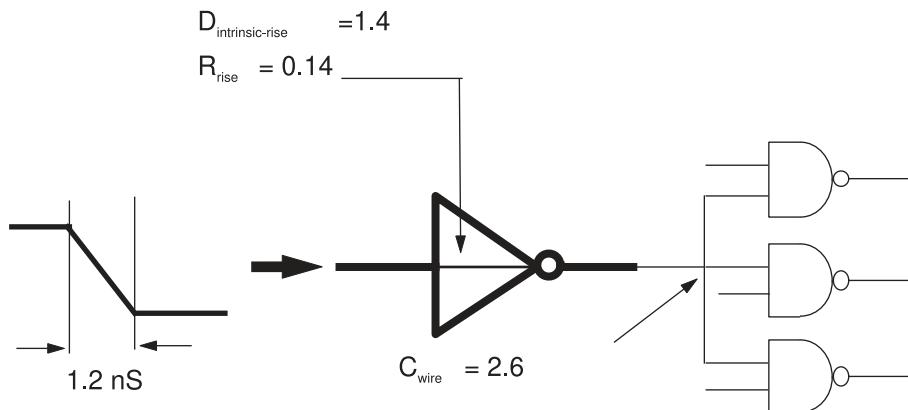
Capacitance values for the load pin.

Total connect delay is calculated by scaling the constant values by their corresponding k-factors (see [Environmental Scaling](#)).

Delay Calculation (Linear) Example

Figure 141 shows the rise delay values for an inverter.

Figure 141 Delay Values for an Inverter



The inverter input pin is driven by a falling signal with a transition time (D_T) of 1.2 ns. The inverter fans out to three NAND gates, each with an input pin capacitance of 1.1. The inverter has an intrinsic rise delay of 1.4, a rise slope sensitivity of 0.02, and an output rise resistance of 0.14. Assuming a best-case RC-interconnect tree type and an estimated interconnect wire capacitance of 2.6, the rise delay is 2.25.

The following is the rise delay calculation for the inverter shown in Figure 141:

$$D_{intrinsic-rise} = 1.4$$

$$S_{rise} = 0.02$$

$$R_{rise} = 0.14$$

$$C_{pins} = 3 * (1.1) = 3.3$$

$$C_{wire} = 2.6$$

$$D_{T-fall_previous_stage} = 1.2$$

$$D_{connect-rise} = 0.0 \text{ for a best case RC-tree}$$

$$\begin{aligned} D_{rise} &= \text{Intrinsic} + \text{Slope} + \text{Transition} + \text{Connect} \\ &= 1.4 + (1.2 * 0.02) + (0.14)(3.3 + 2.6) + 0.0 \\ &= 1.4 + 0.02 + 0.826 + 0.0 \\ &= 2.25 \end{aligned}$$

CMOS2 Delay Model

The CMOS2 delay model is similar to the linear delay model except that it uses delay due to edge rate instead of slope delay. The CMOS2 delay equation for computing gate delay values is the sum of four terms:

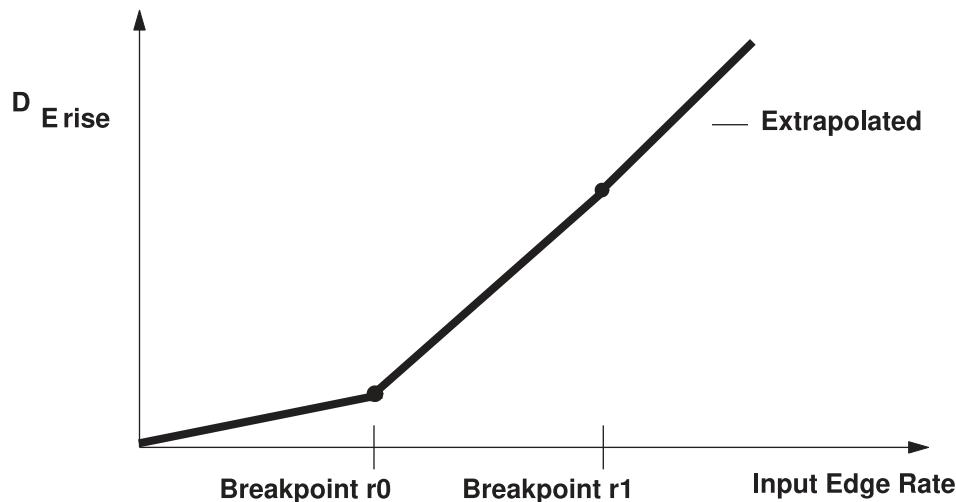
intrinsic delay (D_I)

The built-in delay from an input pin to an output pin.

delay due to input edge rate (D_E)

The delay incurred at an input pin due to edge rate. At each input pin, the timing analyzer computes the actual edge rate. The edge rate can be different for different pins on the net. Also, the cell delay can have a two-piece dependency on input edge rate. See [Figure 142](#) for an example of an input rise dependency.

Figure 142 Rise Dependency on Edge Rate (CMOS2)



transition delay (D_T)

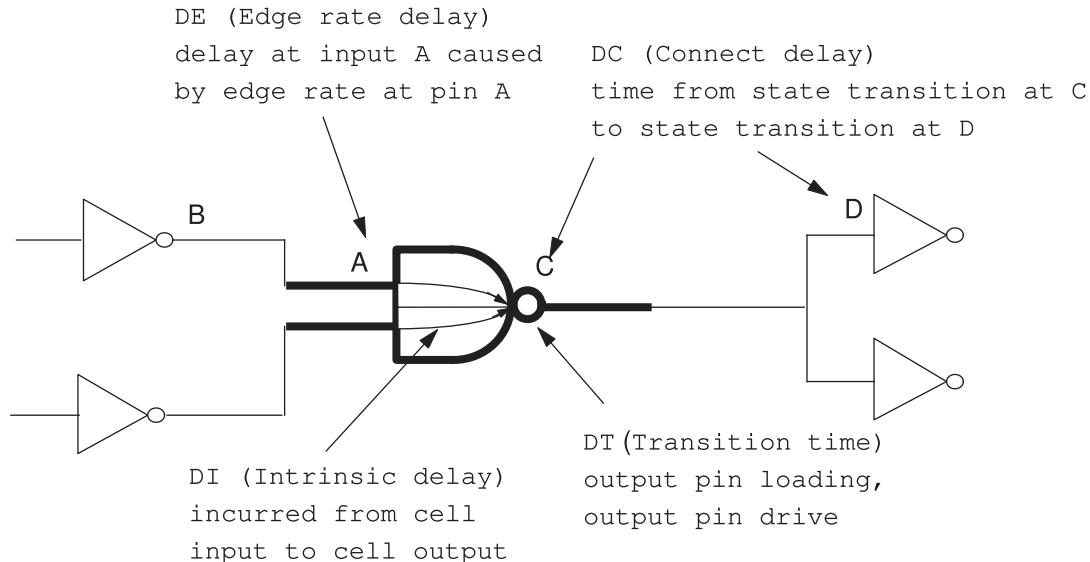
The time it takes to change logic value due to loading at an output pin (output resistance times load).

connect delay (D_C)

The time-of-flight delay—the time it takes a logic transition to propagate through an interconnect network.

[Figure 143](#) shows a diagram of the four terms in the delay equation.

Figure 143 Delay Equation Terms and Timing Arcs (CMOS2)



The timing analyzer does not store delay equation parameters in terms of specific units. The only restriction the timing analyzer places on the values used is that the units must form an internally consistent system. Synopsys engineers suggest a system of nanoseconds (delay), picofarads (load), and kilohms (resistance).

Example

This example shows the rise and fall delay equations.

- For positive unate arcs:

$$\begin{aligned} D_{rise} &= D_{edge-rise} + D_{intrinsic-rise} + D_{transition-rise} \\ D_{fall} &= D_{edge-fall} + D_{intrinsic-fall} + D_{transition-fall} \end{aligned}$$

- For negative unate arcs:

$$\begin{aligned} D_{rise} &= D_{edge-fall} + D_{intrinsic-rise} + D_{transition-rise} \\ D_{fall} &= D_{edge-rise} + D_{intrinsic-fall} + D_{transition-fall} \end{aligned}$$

- For non-unate arcs:

$$\begin{aligned} D_{rise} &= D_{edge-max} + D_{intrinsic-rise} + D_{transition-rise} \\ D_{fall} &= D_{edge-max} + D_{intrinsic-fall} + D_{transition-fall} \end{aligned}$$

Equation Parameters

The parameters of these equations are

Appendix A: Static Timing Delay Calculation

Delay Models

```

Dedge-rise = edge_rate_sensitivity_r0 x
    min(edge_rate_breakpoint_r1 - edge_rate_breakpoint_r0,
        edge_rate_rise_delay - edge_rate_breakpoint_r0) +
    edge_rate_sensitivity_r1 x max(0, edge_rate_rise -
        edge_rate_breakpoint_r1)
Dedge-fall = edge_rate_sensitivity_f0 x
    min(edge_rate_breakpoint_f1 - edge_rate_breakpoint_f0,
        edge_rate_fall_delay - edge_rate_breakpoint_f0) +
    edge_rate_sensitivity_f1 x max(0, edge_rate_fall -
        edge_rate_breakpoint_f1)
edge_rate_sensitivity_r0 = (arc) rising edge-rate sensitivity of first piece
edge_rate_sensitivity_r1 = (arc) rising edge-rate sensitivity of second piece
edge_rate_breakpoint_r0 = (input pin) first breakpoint on rising edge-rate
    sense curve
edge_rate_breakpoint_r1 = (input pin) second breakpoint on rising edge-rate se
nse curve
edge_rate_rise_delay = edge_rate_rise +
    edge_rate_load_rise x (interconnect and input pin
    capacitance - reference capacitance of the output pin
    driving the net + (estimated connect delay/rise
    resistance of driving cell))
edge_rate_rise = the zero-load rise edge rate for a driver pin
edge_rate_sensitivity_f0 = (arc) falling edge-rate sensitivity of first piece
edge_rate_sensitivity_f1 = (arc) falling edge-rate sensitivity of second piece
edge_rate_breakpoint_f0 = (input pin) first breakpoint on falling edge-rate se
nse curve
edge_rate_breakpoint_f1 = (input pin) second breakpoint
    on falling edge-rate sense curve
edge_rate_fall_delay = edge_rate_fall +
    edge_rate_load_fall x (input net capacitance -
    reference capacitance + (estimated connect delay/fall
    resistance of driving cell))
edge_rate_fall = the zero-load fall edge rate for a driver pin
edge_rate_load_fall = the load dependent falling edge rate capability for a dr
iver pin
edge_rate_load_rise = the load dependent rising edge rate capability for a dri
ver pin
Dintrinsic-rise = Intrinsic rise delay from library
Dintrinsic-fall = Intrinsic fall delay from library

```

For non-piecewise:

$$\begin{aligned} D_{transition-rise} &= R_{rise} \times (C_{pins} + C_{wire}) \\ D_{transition-fall} &= R_{fall} \times (C_{pins} + C_{wire}) \end{aligned}$$

For piecewise:

$$\begin{aligned} D_{transition-rise} &= R_{drivei-rise} \times (C_{pins} + C_{wire}) + Y_{adj_i} \\ D_{transition-fall} &= R_{drivei-fall} \times (C_{pins} + C_{wire}) + Y_{adj_i} \end{aligned}$$

$$D_{connect-rise} = \{ \begin{array}{l} \text{if tree type is best case: } 0.0 \\ \text{if tree type is worst case: } (R_{wire} (C_{wire} + C_{pins}) \times K_{rc_rise}) \\ \text{if tree type is balanced: } (R_{wire}/N (C_{wire}/N + C_{pin}) \times K_{rc_rise}) \end{array} \}$$

$$D_{connect-fall} = \{ \begin{array}{l} \text{if tree type is best case: } 0.0 \\ \text{if tree type is worst case: } (R_{wire} (C_{wire} + C_{pins}) \times K_{rc_fall}) \\ \text{if tree type is balanced: } (R_{wire}/N (C_{wire}/N + C_{pin}) \times K_{rc_fall}) \end{array} \}$$

R_{wire} = Wire resistance

C_{pins} = Sum of all pin capacitances on net
 C_{pin} = Individual pin capacitance value

N = Number of pins being driven

K_{rc_rise} = Library multiplication factor

Edge Rate Delay

The delay due to the input edge rate of an element, D_E , represents the delay incurred at an input pin due to edge rate.

Intrinsic Delay

The intrinsic delay of an element, D_I , is the portion of the total delay that is independent of the element's use. This is the fixed (or zero-load) delay of an element.

Total intrinsic delay is calculated by scaling these constant values by their corresponding k-factors (see [Environmental Scaling](#)).

Transition Time

Transition time D_T is the delay introduced by the capacitive load on a gate's output pin. It represents the time it takes the output to switch from one logical state to another.

Transition time is computed in one of two ways, depending on the logic library:

Equation 1 Linear Transition Time

$$D_T = R_{drive} \times \left(\sum_{pins} C_{pin} + C_{wire} \right)$$

Equation 2 Piecewise Linear Transition Time

$$D_T = R_{drivei} \times \left(\sum_{pins} C_{pin} + C_{wire} \right) + Y_{adj_i}$$

In the piecewise linear model, the resistance value (R_{drivei}) and a constant term (Y_{adj_i}) can vary with different loading conditions. The `piece_type` statement in the logic library determines how the appropriate resistance and constant are selected. The selection is done on the basis of one of the following criteria:

- Total net length
- Total output capacitance
- Output pin capacitance
- Output wire capacitance

A `piece_define` statement in the library determines the correlation between resistance and constant term values and the `piece_type`.

Total transition time is calculated by scaling the constant values by their corresponding k-factors (see [Environmental Scaling](#)).

Connect Delay

Connect delay, D_C , is the time it takes the voltage at an input pin to change after the transition of the driving output pin. Connect delay is also called the time-of-flight delay (the time it takes for a waveform to travel along a wire). The way this delay is calculated is important in the analysis of interconnect network delay. The timing analyzer supports three cases for an estimated interconnect topology (tree type).

- Best case (`best_case_tree`) models the case where the load pin is physically adjacent to the driver. All wire capacitance is incurred, but none of the wire resistance must be overcome. The best-case connect delay is calculated from the following equation. Because R_{wire} is always 0 in this case, the resulting D_C is always zero.

$$D_{Cbest} = R_{wire}(C_{wire} + C_{pin}) = 0$$

- Balanced case (`balanced_tree`) models the case where all load pins are on separate, equal branches of the interconnect wire. Each load pin incurs an equal portion of the wire capacitance and wire resistance.

$$D_{Cbalanced} = \frac{R_{wire}}{N} \left(\frac{C_{wire}}{N} + C_{pin} \right)$$

- Worst case (`worst_case_tree`) models are different for the CMOS2 model than they are for other models (the resistance is divided by the fanout N). The load pin is at the extreme end of the wire. Each load pin incurs both the full wire capacitance and a portion of the wire resistance.

$$D_{Cworst} = \frac{R_{wire}}{N} \left(C_{wire} + \sum_{pins} C_{pin} \right)$$

These equations are used for calculating both the rise and fall delays. Where applicable, use the `rise_` parameter for calculating the rise delay and the `fall_` parameter for calculating the fall delay. Descriptions of the components in the equation follow.

R_{wire}

Estimated wire resistance on the net determined by the wire load model. Wire length is computed with a global estimation function whose parameter is the number of fanout pins on the net being estimated. The estimated value is scaled by the resistance factor.

C_{wire}

Estimated wire capacitance for the net attached to the head of the timing arc for which the delay value is being computed. Wire length is computed with the actual number of fanout pins on the net being estimated and the `fanout_length` specifications in the `wire_load` group. The estimated value is scaled by the capacitance factor.

C_{pin}

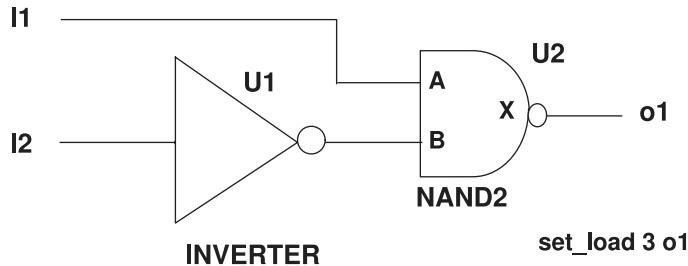
Capacitance values for the load pin.

Total connect delay is calculated by scaling the constant values by their corresponding k-factors (see [Environmental Scaling](#)).

Delay Calculation (CMOS2) Example

This example shows the cell delay across a NAND2 gate from pin B to pin X.

```
wire_load_model: MEDIUM
tree_type: balanced_case
operating_conditions: nominal
port_edge_rate: 0.0
```



The previous driver, an INVERTER gate, has the following characteristics:

```
cell(INVERTER) {
    area : 1;
    pin(X) {
        function : "A'";
        direction : output;
        edge_rate_rise : 0.12;
        edge_rate_fall : 0.13;
        edge_rate_load_rise : 4.5;
        edge_rate_load_fall : 2.5;
        timing() {
            edge_rate_sensitivity_r0 : 0.20;
            edge_rate_sensitivity_f0 : 0.10;
            edge_rate_sensitivity_r1 : 0.15;
            edge_rate_sensitivity_f1 : 0.05;
            intrinsic_rise : 0.10;
            intrinsic_fall : 0.12;
            rise_resistance : 2.0;
            fall_resistance : 1.0;
            related_pin : "A";
        }
    }
    pin(A) {
        direction : input;
        capacitance : 0.10;
    }
}
```

```
    }
}
```

NAND2 Gate Library Description

```
cell(NAND2) {
    area : 1;
    pin(X) {
        function : "(A B)'";
        direction : output;
        edge_rate_rise : 0.24;
        edge_rate_fall : 0.14;
        edge_rate_load_rise : 5.4;
        edge_rate_load_fall : 3.4;
        timing() {
            intrinsic_rise : 0.34;
            intrinsic_fall : 0.24;
            rise_resistance : 3.4;
            fall_resistance : 1.4;
            edge_rate_sensitivity_r0 : 0.24;
            edge_rate_sensitivity_f0 : 0.14;
            edge_rate_sensitivity_r1 : 0.14;
            edge_rate_sensitivity_f1 : 0.04;
            related_pin : "A";
        }
        timing() {
            intrinsic_rise : 0.34;
            intrinsic_fall : 0.24;
            rise_resistance : 3.4;
            fall_resistance : 1.4;
            edge_rate_sensitivity_r0 : 0.24;
            edge_rate_sensitivity_f0 : 0.14;
            edge_rate_sensitivity_r1 : 0.14;
            edge_rate_sensitivity_f1 : 0.04;
            related_pin : "B";
        }
    }
    pin(A) {
        direction : input;
        capacitance : 0.10;
    }
    pin(B) {
        direction : input;
        capacitance : 0.10;
    }
}
```

Library Global Values

The library global values are

```
delay_model : cmos2;
time_unit : "1ns";
```

Appendix A: Static Timing Delay Calculation

Delay Models

```

default_max_transition : 12.00;

default_edge_rate_breakpoint_r0 : 0.500;
default_edge_rate_breakpoint_r1 : 3.000;
default_edge_rate_breakpoint_f0 : 0.500;
default_edge_rate_breakpoint_f1 : 3.000;
default_reference_capacitance : 0.000;
default_setup_coefficient : 1.0;
default_hold_coefficient : 1.0;
default_rc_rise_coefficient : 1.0;
default_rc_fall_coefficient : 1.0;

wire_load("MEDIUM") {
    resistance : 0.00003;
    capacitance : 0.0001;
    area : 0;
    fanout_length(1,800);
    fanout_length(6,2800);
    fanout_length(7,3200.0);
    fanout_length(14,6000.0);
    fanout_length(15,6500.0);
    slope : 400;
}

```

For this example, a balanced-case tree type is used with the MEDIUM wire load model, and nominal operating conditions are assumed. Net N1 connects the driver, an INVERTER pin X, to the NAND2 pin B.

This net has the following characteristics:

```

pin_capacitance :      0.1
wire_capacitance :     0.08
total_capacitance :   0.18
wire_resistance :     0.024

```

Consider the rising cell delay across the NAND2 (B to X). The rising delay on NAND2 pin X requires a fall transition on the input B.

$\text{rise_cell_delay} = D_e(\text{input falling}) + D_i_{\text{rise}} + D_t_{\text{rise}}$

To determine the rise cell delay:

$\text{drive_pin} = u1/X \text{ (INVERTER)}$

The effective capacitance noted by the previous driver is used to calculate the actual falling edge rate at B (ERFD). The fall_resistance is from the previous driver, the INVERTER cell.

```

c_eff = total_net_cap - drive_pin_cap
       - drive_pin_reference_capacitance
       + fall_connect_delay / drive_pin_fall_resistance
c_eff = 0.18 - 0 - 0 + 0.00432 / 1
c_eff = 0.18432

```

Appendix A: Static Timing Delay Calculation

Delay Models

```

ERFD = driver_edge_rate_fall + driver_edge_rate_load_fall * c_eff
ERFD = 0.13 + 2.5 * 0.18432
ERFD = 0.5908

```

The delay due to input edge rate is a two-segment, piecewise linear dependence on ERFD. The breakpoints are found in the library. Calculating the delay due to edge rate (De_fall) is as follows:

```

De_fall = edge_rate_sensitivity_f0 * min(bkpt_f1-bkpt_f0,ERFD-bkpt_f0) +
          edge_rate_sensitivity_f1 * max(0.0, ERFD-bkpt_f1)
De_fall = 0.14 * min(3-0.5,0.5908-0.5) + 0.04 * max(0.0, 0.5908-3)
De_fall = 0.012712

```

The intrinsic rise of the NAND2 is from the library.

```

Di_rise = rise_intrinsic
Di_rise = 0.34

```

The timing analyzer computes the NAND2 transition time, using the rise_resistance of this cell and the capacitance at net O1, which the NAND2 drives.

```

Dt_rise = rise_resistance * net_cap
Dt_rise = 3.4 * 3.08
Dt_rise = 10.472

```

To compute the total cell delay,

```

rise_cell_delay = De + Di_rise + Dt_rise
rise_cell_delay = 0.012712 + 0.34 + 10.472
rise_cell_delay = 10.8247

```

The following is the full timing report:

Design	Wire Loading Model	Library
TWO_INV	MEDIUM	cmos2_c
Startpoint: I1 (input port)		
Endpoint: O1 (output port)		
Path Group: (none)		
Path Type: max		
Point	Incr	Path
input external delay	0.00	0.00 r
I1 (in)	0.00	0.00 r
u1/A (INVERTER)	0.00	0.00 r
u1/X (INVERTER)	0.20	0.20 f
u2/B (NAND2)	0.00	0.21 f
u2/X (NAND2)	10.82	11.03 r
O1 (out)	0.07	11.11 r
		11.11
data arrival time		

Nonlinear Delay Model

The nonlinear delay model stores vendor-specific delay information in the logic library in the form of lookup tables. This model supports a close correlation between nonlinear vendor delay models and the timing analyzer calculations.

Delay analysis involves calculating total delay, which comprises cell and connect delay.

$$D_{\text{total}} = D_{\text{cell}} + D_c$$

D_{cell}

The delay contributed by the gate itself, typically measured from the 50 percent input pin voltage to the 50 percent output pin voltage.

D_c

The connect delay. It is either calculated by using the `tree_type` attribute in the `operating_conditions` group and the selected wire load model, or read in from an SDF file as in the standard delay equation. Connect delay is calculated by the same method used in other delay equations.

The CMOS nonlinear timing model supports two methods of computing D_{cell} . Although you can mix the two methods in a logic library, you typically specify only the method that best correlates to the characterized library data.

The timing analyzer can compute D_{cell} directly by performing table lookup and interpolation in a cell delay table provided in the library or it can compute D_{cell} by using the propagation and transition tables:

$$D_{\text{cell}} = D_{\text{propagation}} + D_{\text{transition}}$$

$D_{\text{propagation}}$

A typical measurement for $D_{\text{propagation}}$ is the time from the 50 percent input pin voltage until the gate output just begins to switch, for example the 10 percent output voltage. Thus, when a $D_{\text{transition}}$ value defined from the 10 percent to 50 percent output voltage is added to $D_{\text{propagation}}$, the result is a 50 percent input to 50 percent output cell delay.

$D_{\text{transition}}$

The time required for the output pin to change state. This is sometimes referred to as the output ramp time.

$D_{\text{transition}}$ is the time between two reference voltage levels on the output pin. These levels can be 20 percent to 80 percent or 10 percent to 50 percent, for example. $D_{\text{transition}}$ is computed by performing table lookup and interpolation.

If cell delay tables are provided for a timing arc, the total delay equation used is

$$D_{\text{total}} = D_{\text{cell}} + D_c$$

If propagation delay tables are provided instead, the total delay equation becomes

$$D_{\text{total}} = D_{\text{propagation}} + D_{\text{transition}} + D_c$$

When transition tables in the library are indexed by input pin transition, the transition or slew of a gate's input pin affects the transition of an output pin. As cells are being swapped and evaluated during optimization, this input or output transition effect can cause delay changes to ripple outside of the cells local to the change. Early in the design flow, you might not need to account for this ripple effect.

Library Cell Timing Arcs

The following tables are defined for each library cell delay timing arc:

- Rise propagation
- Cell rise
- Fall propagation
- Cell fall
- Rise transition
- Fall transition

Note:

Every delay arc can have propagation tables or cell tables, but not both. Also, every delay arc must have transition tables.

Each delay table is indexed by one through three of the following six variables:

- `input_net_transition`
- `output_net_length`
- `total_output_net_capacitance`
- `related_out_total_output_net_capacitance`
- `output_net_pin_cap`
- `output_net_wire_cap`

The values stored in the table are the propagation or transition time value. Interpolation is used to determine values between points. Breakpoints in the table can be arbitrarily defined. Each table maintains its own breakpoints, independent of other tables. Tables of

different dimensions, breakpoints, and index variables can be intermixed within one logic library.

The following tables are defined for each library-cell-constraint timing arc, as with setup and hold:

- `rise_constraint`
- `fall_constraint`

Each constraint table can be indexed by one through three of the following two variables:

- Constrained pin transition time
- Related pin transition time

This time the values stored in the table are the constraint values. The constraint tables allow setup and hold values to vary depending on the transition at the D-pin (constrained pin) and the clock pin (related pin) of a latch.

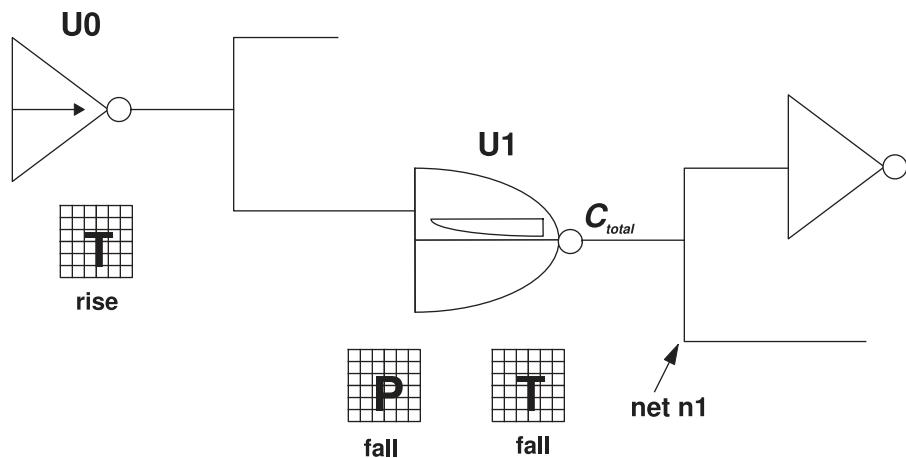
Because much of the information stored in a table is common to other tables, table templates and inheritance are supported. See the Library Compiler documentation for more details.

To use the nonlinear delay model, activate a logic library that specifies the nonlinear delay model.

Delay Calculation Example

[Figure 144](#) illustrates the process for determining the fall delay across a timing arc of cell U1.

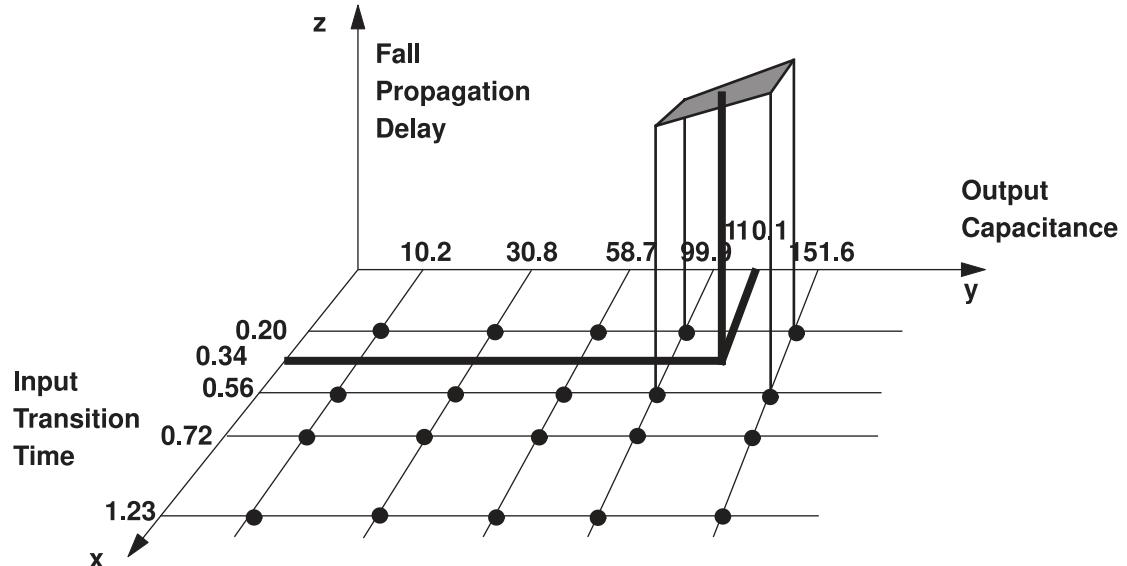
Figure 144 Nonlinear Delay Calculation Example



Examine the fall propagation table: a two-dimensional table indexed by total output capacitance and input transition time. The total output capacitance is the sum of pin and wire capacitance on net n1 (C_{total} in [Figure 144](#)). Input transition time is determined by evaluation of the transition time at the previous gate U0. Because the arc under evaluation in U1 is negative unate, the rise transition table at U0 is evaluated to determine the U1 input transition time. Assume that C_{total} is 110.1 and the input transition time is 0.34. Use these two values to index into the U1 fall propagation table.

[Figure 145](#) shows two-dimensional interpolation.

Figure 145 Two-Dimensional Interpolation



In [Figure 145](#), the black dots represent points defined in the table. The four points with heights shown in the z-axis are the neighboring points chosen for interpolation. The shaded area is the surface used for interpolation.

The fall transition time is computed by evaluation of the fall transition table. In this case, the table is a one-dimensional table based on total output capacitance. The output capacitance value (110.1) was calculated previously. Through simple linear interpolation within the table, the fall transition time is determined.

The fall propagation delay is then added to the fall transition time to compute the cell delay for a falling transition across the given timing arc of cell U1.

If a cell delay table is defined in the logic library for U1 instead of a propagation table, the table evaluations occur in the same way, but the transition result is not included when calculating the cell delay for U1.

Environmental Scaling

When calculating total delay, the timing analyzer individually scales each scalable parameter of D_{total} . Each scalable component of the total delay has its own global parameters to model the effects of variation in process, temperature, and voltage on the nominal case.

The following scaling factor is applied to individual components of the delay equation:

$$(1 + \Delta_V K_V) (1 + \Delta_T K_T) (1 + \Delta_P K_P)$$

Δ_V = Change in voltage from library-specified nominal value.

K_V = Scale factor for change in voltage.

Δ_T = Change in temperature from library-specified nominal value.

K_T = Scale factor for change in temperature.

Δ_P = Change in process from library-specified nominal value.

K_P = Scale factor for change in process.

Each component of the delay equation can have different values for K_V , K_T , and K_P , allowing it to be scaled independently of the other components. See the Library Compiler documentation for more details on environmental scaling.

Waveform Propagation Using CCS Models

Advanced geometry nodes at 28 nm and below can be significantly affected by waveform distortion effects. The IC Compiler tool offers an advanced, gate-level waveform propagation analysis mode that captures the effects of such distortion on the next stage delay, including the Miller effect and long tail effect.

The waveform propagation mode uses the CCS timing models in the .db libraries and uses propagated piecewise linear waveforms in place of simplified equivalent waveforms. It produces more accurate results when the voltage waveform shapes differ significantly from the library characterization driver waveform shapes. The CCS modeling data must be available in the .db libraries to get the more accurate results.

You control this feature in the IC Compiler tool with the `timing_waveform_analysis_mode` variable. The default setting is disabled. Set this variable to `clock_network` to apply CCS-based waveform propagation analysis to the clock network only, or `full_design` to apply it to the whole design.

To further improve accuracy when waveform propagation is enabled, you can optionally use CCS noise models as well as CCS timing models. To do this, set the `timing_enable_ccsn_waveform_analysis` variable to `true`.