PrimeTime®
Fundamentals
User Guide

Version F-2011.06, June 2011

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

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.

Registered Trademarks (®)

Synopsys, AEON, AMPS, ARC, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, Chipidea, CHIPit, CODE V, CoMET, Confirma, CoWare, Design Compiler, DesignSphere, DesignWare, Eclypse, EMBED-IT!, Formality, Galaxy Custom Designer, Global Synthesis, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, LightTools, MAST, MaVeric, METeor, ModelTools, NanoSim, NOVeA, OpenVera, ORA, PathMill, Physical Compiler, PrimeTime, SCOPE, SiVL, SNUG, SolvNet, Sonic Focus, STAR Memory System, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDExplorer are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCSi, VHDL Compiler, VMC, and Worksheet Buffer are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

2. Starting and Using PrimeTime

4. Graphical User Interface

12. Operating Conditions

13. Generating Reports

Preface

This preface includes the following sections:

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *PrimeTime Release Notes* in SolvNet.

To see the *PrimeTime Release Notes*,

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

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select PrimeTime, and then select a release in the list that appears.

About This User Guide

PrimeTime Fundamentals User Guide provides basic information about performing chip-level static timing analysis. It introduces features, concepts, and basic commands of the Synopsys PrimeTime static timing analyzer.

Audience

This user guide is for design engineers who use PrimeTime for static timing analysis.

Related Publications

For additional information about PrimeTime, see the documentation on SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

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

- PrimeTime PX, PrimeTime SI, and PrimeTime VX
- Design Compiler
- IC Compiler
- Library Compiler and Liberty NCX

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

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

To access SolvNet, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

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

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to SolvNet at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Overview

PrimeTime is a full-chip, gate-level static timing analysis tool that is an essential part of the design and analysis flow for today's large chip designs. PrimeTime exhaustively validates the timing performance of a design by checking all possible paths for timing violations, without using logic simulation or test vectors.

This PrimeTime overview chapter includes the following sections:

- PrimeTime Features
- PrimeTime Add-On Features
- Static Timing Analysis Overview
- Learning to Use PrimeTime

Note:

If you encounter an unfamiliar word or phrase, check the Glossary for the definition; however, not all words are defined.

PrimeTime Features

PrimeTime is a full-chip, gate-level static timing analysis tool targeted for complex, multimillion-gate designs. It offers an unsurpassed combination of speed, capacity, ease of use, and compatibility with industry-standard data formats and workflows.

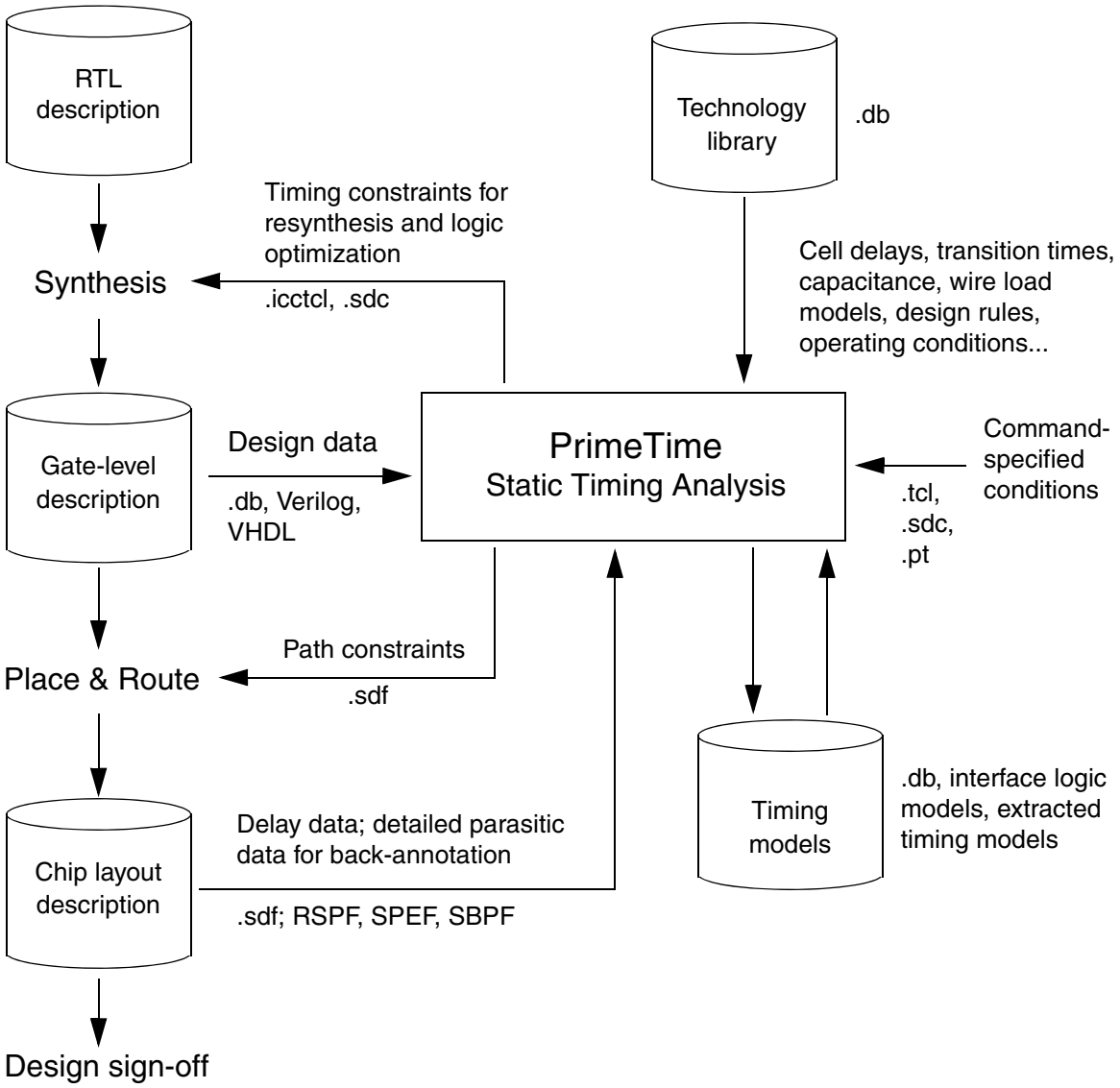
PrimeTime fits ideally into the Synopsys physical synthesis flow because it uses many of the same libraries, databases, and commands as other Synopsys tools such as Design Compiler. It can also operate as a stand-alone static timing analyzer in other design flows. It accepts design information in a wide range of industry-standard formats, including gate-level netlists in .db, Verilog, and VHDL formats; delay information in Standard Delay Format (SDF); parasitic data in Standard Parasitic Exchange Format (SPEF), Synopsys Binary Parasitic Format (SBPF) and Reduced Standard Parasitic Format (RSPF) formats; and timing constraints in Synopsys Design Constraints (SDC) format.

Figure 1-1 shows how PrimeTime is used in a typical synthesis flow. Starting from an RTL design description, a synthesis tool such as Design Compiler generates a gate-level design description. PrimeTime reads this description and verifies the design timing using information provided in the technology library.

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

When the gate-level design is free of timing violations, the designer can proceed to placement and routing. This produces a chip layout database from which accurate delay information or detailed parasitic information can be extracted. This data, when back-annotated on the design in PrimeTime, results in a layout-accurate timing analysis. A successful validation of the circuit timing at this point leads to sign-off of the completed design.

Figure 1-1 Physical Synthesis Flow Using PrimeTime



Note:

The diagram only shows the steps related to timing analysis in the physical synthesis flow. It does not include unrelated steps such as formal verification, scan synthesis, and logic simulation.

Types of Checking Performed

PrimeTime performs the following types of design checking:

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

Analysis Features

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

- Multiple clocks and clock frequencies
- Multicycle path timing exceptions
- False path timing exceptions and automatic false path detection
- Transparent latch analysis and time borrowing
- Simultaneous minimum/maximum delay analysis for setup and hold constraints
- Analysis with on-chip variation (OCV) of process, voltage, and temperature (PVT) conditions
- Case analysis (analysis with constants or specific transitions applied to specified inputs)
- Mode analysis (analysis with module-specific operating modes, such as read mode or write mode for a RAM module)
- Bottleneck analysis (reporting of cells that cause the most timing violations)
- ECO analysis without modifying the original netlist, using inserted buffers, resized cells, and modified nets
- Analysis of crosstalk effects between physically adjacent nets using the PrimeTime SI (signal integrity) option

Timing Models

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

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

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

PrimeTime supports the following types of timing models:

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

Command Interfaces

PrimeTime offers two command environments for timing analysis: pt_shell and the graphical user interface (GUI). The pt_shell interface is a command-line and script-execution environment based on the tool command language (Tcl) programming language, suitable for programmed, computation-intensive tasks. The GUI is a window-based, graphical environment suitable for interactive analysis and for graphically visualizing design data and analysis results.

Figure 1-2 shows a typical pt_shell window and Figure 1-3 shows the top-level GUI window.

Figure 1-2 pt_shell Window

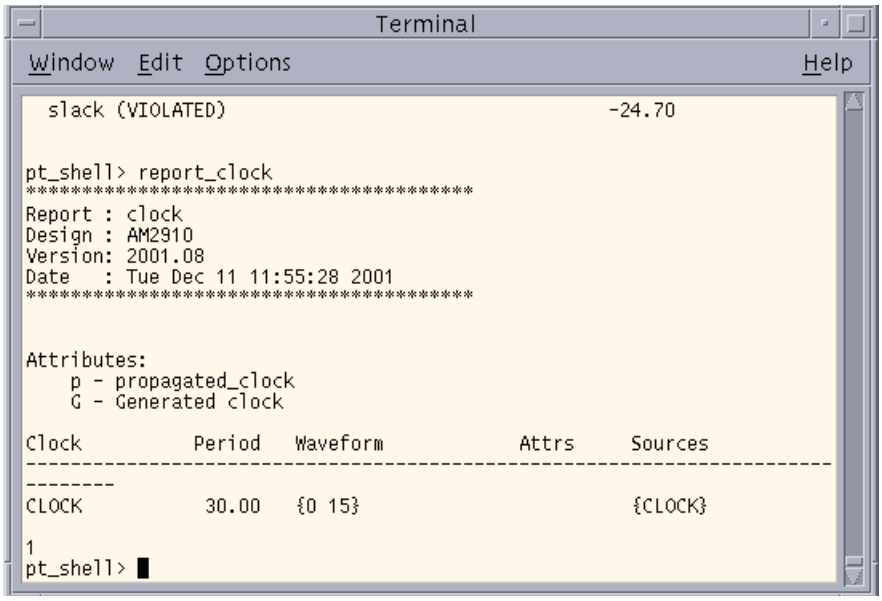
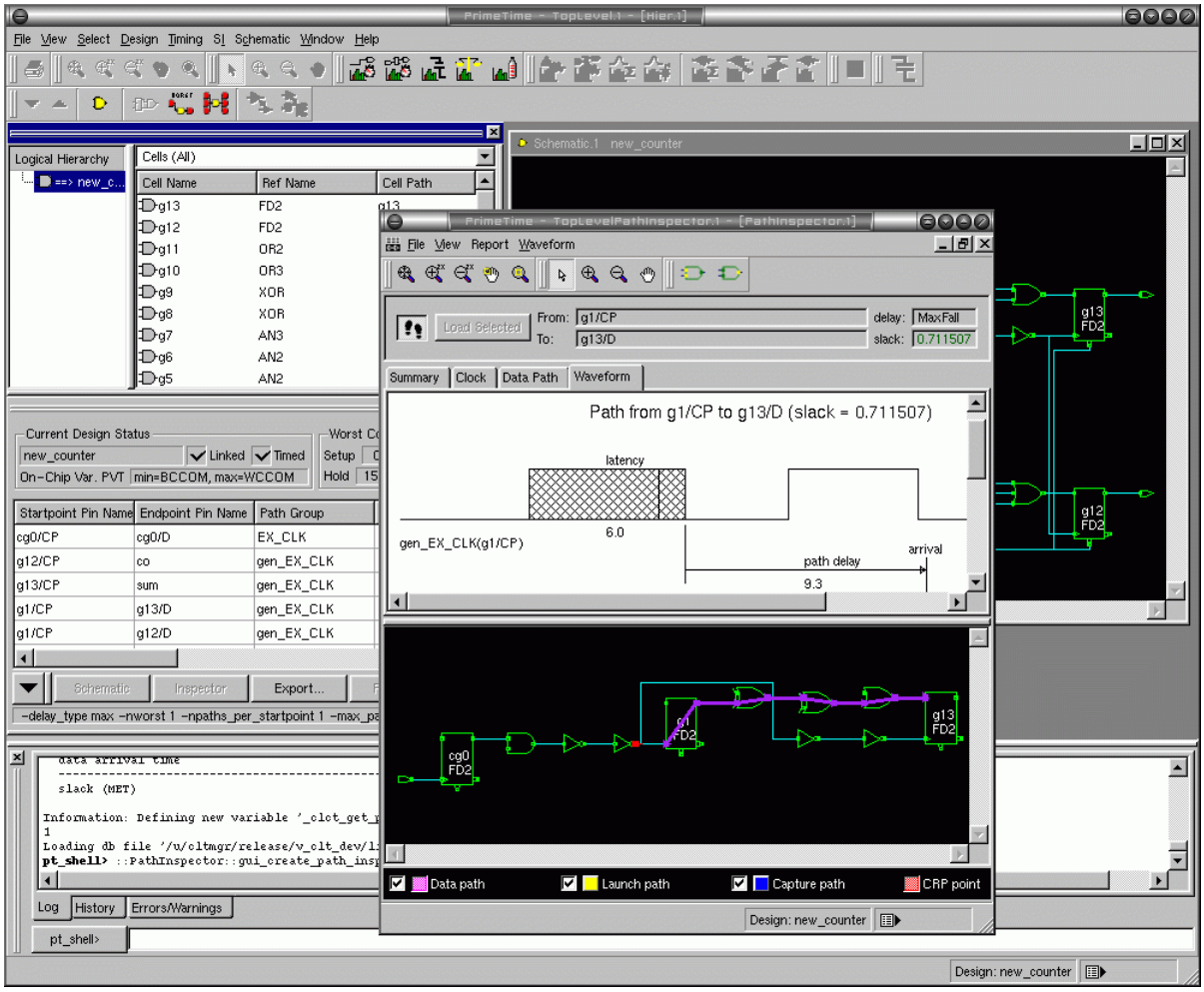


Figure 1-3 PrimeTime Top-Level GUI Windows



The pt_shell interface is the command environment used most often by experienced PrimeTime users. It is a text-only environment in which you enter commands at a prompt (pt_shell>) and view responses and reports in text form. The pt_shell interface is based on the Tcl scripting language, which means that you can use Tcl features such as procedures, lists, and array processing functions. Routine tasks are typically done by executing scripts or procedures prepared beforehand.

The GUI offers some visual analysis capabilities that are not available in pt_shell. For example, you can view schematics of the design, display clock waveforms, and generate histograms of analysis results, such as path slack, net capacitance, and bottleneck cost. The console within the top-level window lets you enter commands and view the text response, just like pt_shell.

Design Compiler Compatibility

The PrimeTime static timing analysis tool is designed to work well with the most commonly used synthesis tool, Design Compiler. PrimeTime and Design Compiler are compatible in the following ways:

- Both tools use the same technology libraries and read the same design data files in .db and .ddc formats.
- Both tools share many of the same commands, such as the `create_clock`, `set_input_delay`, and `report_timing` commands. Shared commands are identical or very similar in operation.
- Both tools share the same delay calculation algorithms and generally produce identical delay results.
- Timing reports generated by both tools are very similar.
- PrimeTime can capture the timing environment of a synthesizable subcircuit and write this timing environment as a series of Design Compiler commands. You can use the resulting script in Design Compiler to define the timing constraints for synthesis or logic optimization of the subcircuit.
- Both tools support the Synopsys Design Constraints (SDC) format for specifying design intent, including the timing and area constraints for a design.

Although Design Compiler has its own built-in static timing analysis capability, PrimeTime has better speed, capacity, and flexibility for static timing analysis, and offers many features not supported by Design Compiler such as timing models, mode analysis, and internal clocks.

PrimeTime Add-On Features

The PrimeTime Suite contains several add-on tools that you can use with PrimeTime. To use an add-on, you must have the appropriate licenses.

PrimeTime PX

PrimeTime PX is a PrimeTime add-on feature that accurately analyzes full-chip power dissipation of cell-based designs. It provides vector-free and vector-based peak power and average power analysis. The vectors to PrimeTime PX are either RTL or gate-level simulation results in Value Change Dump (VCD) format or Switching Activity Interchange Format (SAIF). PrimeTime PX provides multi-VDD and power domain analysis. It also has an integrated graphical user interface (GUI) for visual power debugging. For more information, see the *PrimeTime PX User Guide*.

PrimeTime SI

PrimeTime SI (signal integrity) is a PrimeTime add-on feature that adds crosstalk analysis capabilities to PrimeTime static timing analyzer. PrimeTime SI calculates the timing effects of cross-coupled capacitors between nets and includes the resulting delay changes in the PrimeTime analysis reports. It also calculates the logic effects of crosstalk noise and reports conditions that could lead to functional failure. For more information, see the *PrimeTime SI User Guide*.

PrimeTime VX

PrimeTime VX is a PrimeTime add-on feature that increases the accuracy of timing analysis by considering the statistical variation and distribution of process parameters. PrimeTime VX accurately determines the timing behavior of a circuit under varying parameters such as channel length, threshold voltage, and interconnect wire thickness. Given a set of variation-aware cell libraries and interconnect data, PrimeTime VX analyzes path timing behavior under all combinations of parameter variations. For more information, see the *PrimeTime VX User Guide*.

Static Timing Analysis Overview

Static timing analysis is a method of validating the timing performance of a design by checking all possible paths for timing violations. PrimeTime checks for violations in the same way that you would do it manually, but with much greater speed and accuracy.

To check a design for violations, PrimeTime breaks the design down into a set of timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints inside the design and at the input/output interface.

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

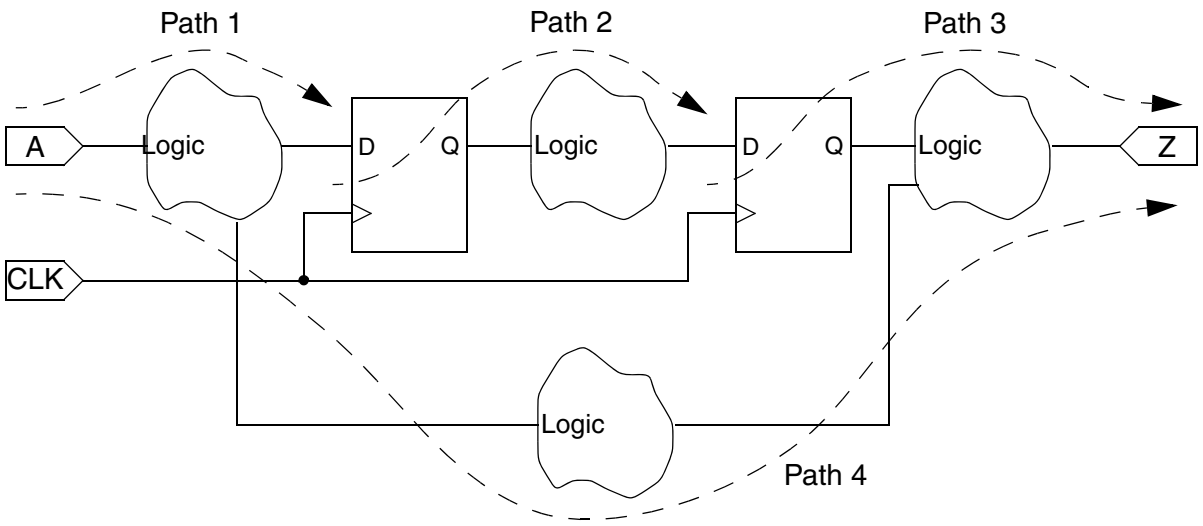
Timing Paths

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

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

Figure 1-4 shows an example of a simple design and the data paths contained in that design.

Figure 1-4 Timing Paths

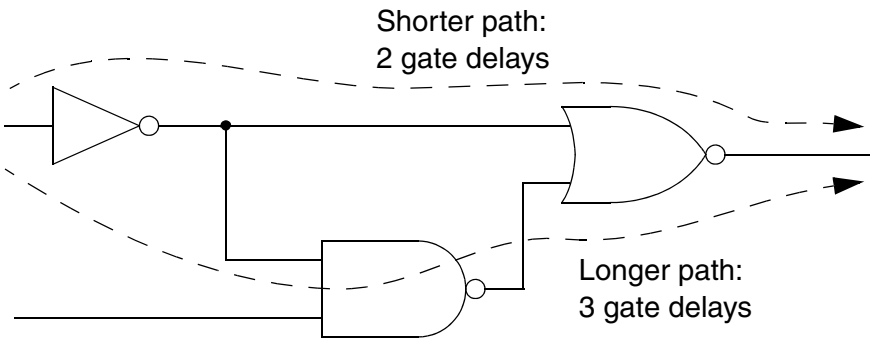


In this figure, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point:

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

A combinational logic cloud might contain multiple paths, as illustrated in Figure 1-5. PrimeTime uses the longest path to calculate a maximum delay or the shortest path to calculate a minimum delay.

Figure 1-5 Multiple Paths Through Combinational Logic

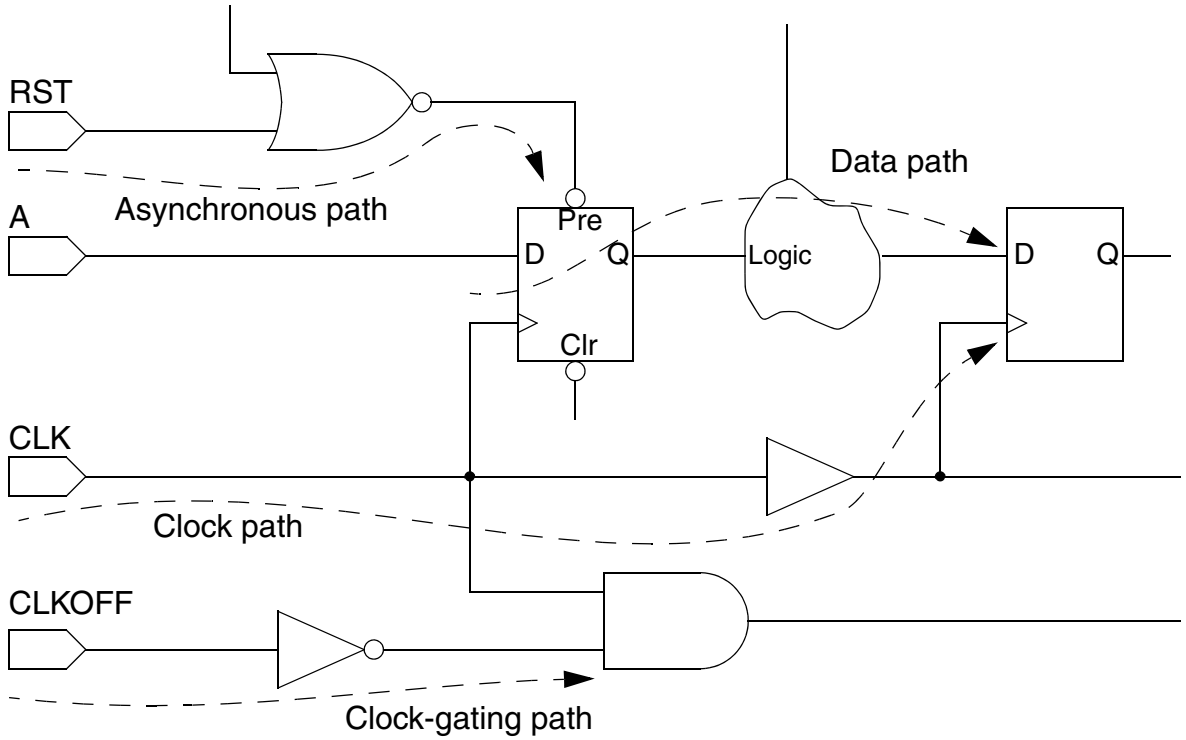


In addition to the data paths just described, PrimeTime considers other types of paths for timing analysis, such as the following:

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

Figure 1-6 shows some examples of these types of paths.

Figure 1-6 Path Types



Delay Calculation

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

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

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

Cell Delay

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

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

Net Delay

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

PrimeTime can calculate net delays by the following methods:

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

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

Constraint Checking

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

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

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

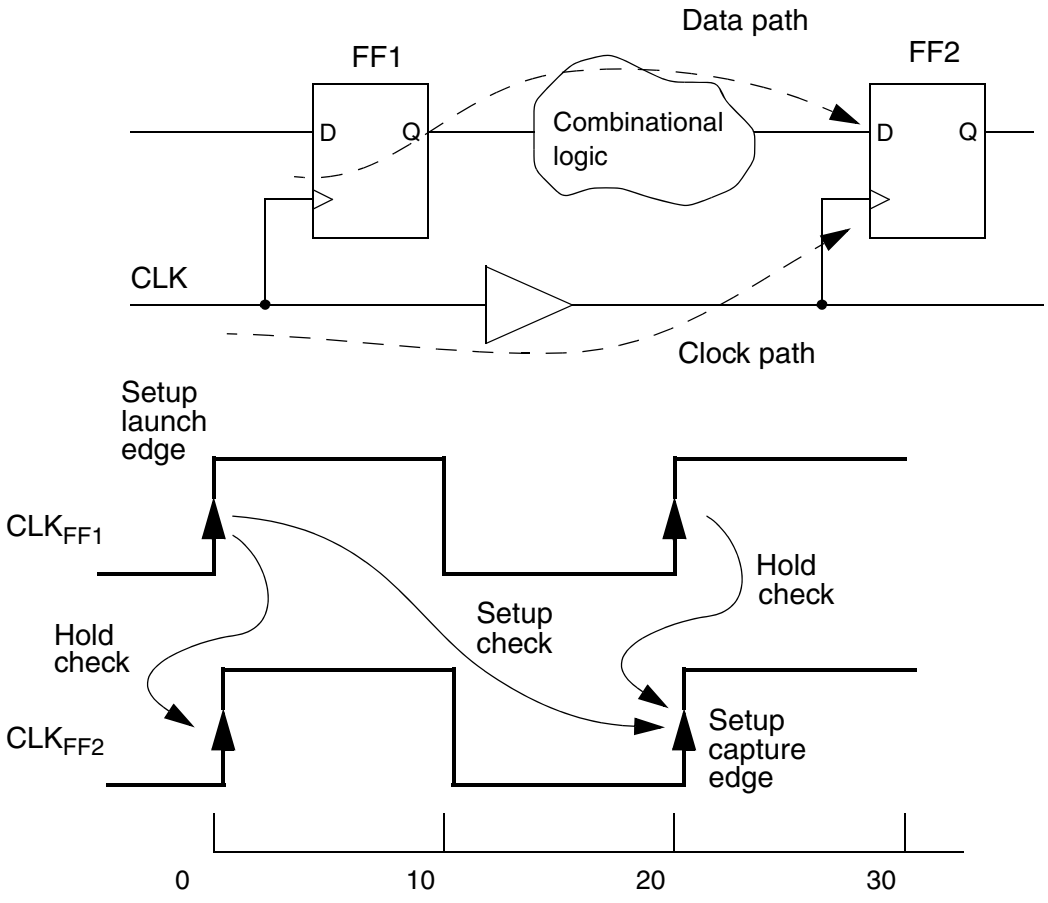
In addition to setup and hold constraints, PrimeTime can also check recovery/removal constraints, data-to-data constraints, clock-gating setup/hold constraints, and minimum pulse width for clock signals.

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

Setup and Hold Checking for Flip-Flops

Figure 1-7 shows how PrimeTime checks setup and hold constraints for a flip-flop in the absence of timing exceptions that apply to the data path.

Figure 1-7 Setup and Hold Checks



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

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

When PrimeTime performs a hold check, it verifies that the data launched from FF1 reaches FF2 no sooner than the capture clock edge for the previous clock cycle. This check ensures that the data already existing at the input of FF2 remains stable long enough after the clock edge that captures data for the previous cycle. For this hold check, PrimeTime considers the shortest possible delay along the data path and the longest possible delay along the clock path between FF1 and FF2. A hold violation can occur if the clock path has a long delay.

Note:

For more examples of setup and hold calculations, including paths that use different clocks at the path startpoint and endpoint, see Chapter 9, “Timing Exceptions”.

Setup and Hold Checking for Latches

Latch-based designs typically use two-phase, nonoverlapping clocks to control successive registers in a data path. In these cases, PrimeTime can use time borrowing to lessen the constraints on successive paths. For example, consider the two-phase, latch-based path shown in Figure 1-8. All three latches are level-sensitive, with the gate active when the G

input is high. L1 and L3 are controlled by PH1, and L2 is controlled by PH2. A rising edge launches data from the latch output, and a falling edge captures data at the latch input. For this example, consider the latch setup and delay times to be zero.

Figure 1-8 Latch-Based Paths

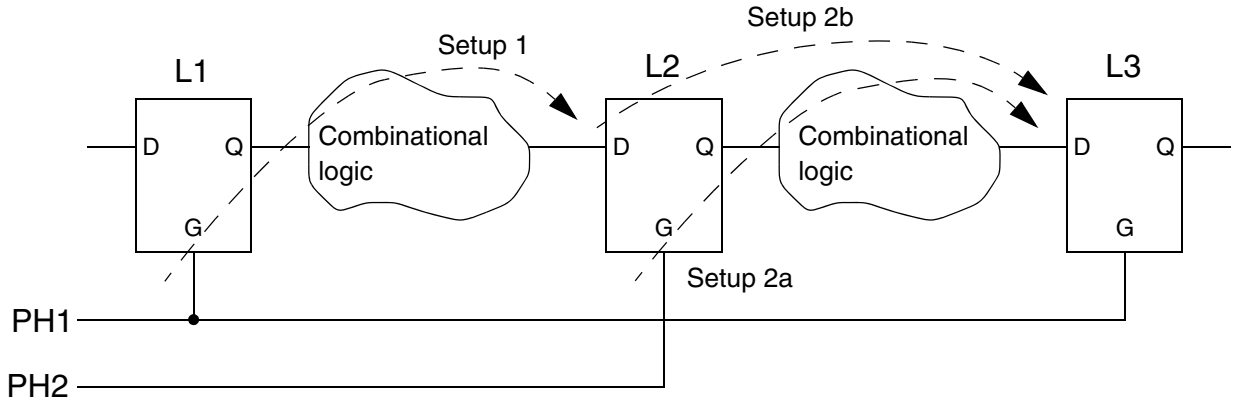
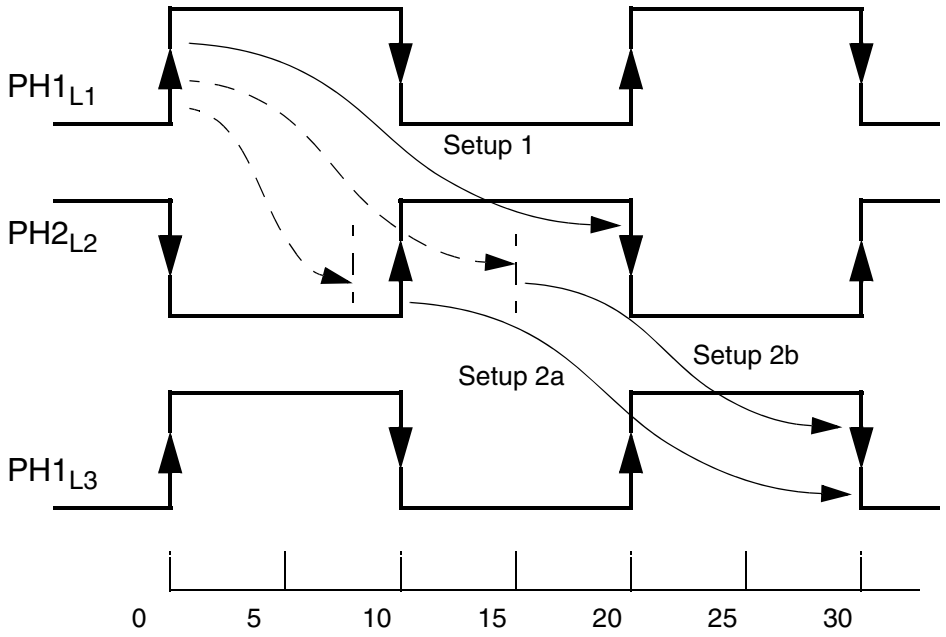


Figure 1-9 shows how PrimeTime performs setup checks between these latches. For the path from L1 to L2, the rising edge of PH1 launches the data. The data must arrive at L2 before the closing edge of PH2 at time=20. This timing requirement is labeled Setup 1.

Depending on the amount of delay between L1 and L2, the data might arrive either before or after the opening edge of PH2 (at time=10), as indicated by the dashed-line arrows in the timing diagram. Arrival after time=20 would be a timing violation.

Figure 1-9 Time Borrowing in Latch-Based Paths



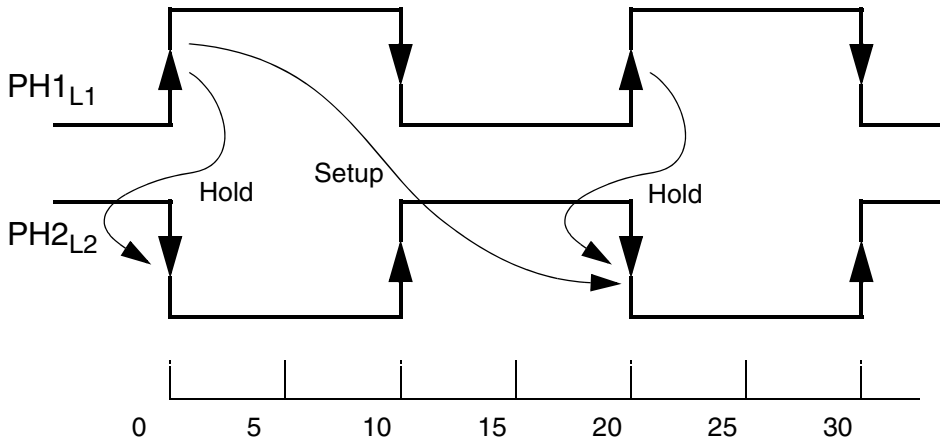
If the data arrives at L2 before the opening edge of PH2 at time=10, the data for the next path from L2 to L3 gets launched by the opening edge of PH2 at time=10, just as a synchronous flip-flop would operate. This timing requirement is labeled Setup 2a.

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

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

To perform hold checking, PrimeTime considers the launch and capture edges relative to the setup check. It verifies that data launched at the startpoint does not reach the endpoint too quickly, thereby ensuring that data launched in the previous cycle is latched and not overwritten by the new data. This is depicted in Figure 1-10.

Figure 1-10 Hold Checks in Latch-Based Paths



Timing Exceptions

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

PrimeTime lets you specify the following types of timing exceptions:

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

Learning to Use PrimeTime

For a basic introduction to using PrimeTime, read this chapter and the next chapter of this manual.

If you need help while using PrimeTime, information is readily available by using the `help` and `man` commands, by consulting the PrimeTime manuals, and by searching through the Web-based SolvNet article database. For more information about getting help, see "PrimeTime Documentation and Online Help" on page 2-13.

Hands-on training for PrimeTime and other Synopsys products is available from Synopsys Training and Support Services at a variety of times and locations worldwide. For more information, click Training & Support from the following Synopsys Web site address:

http://www.synopsys.com/

2

Starting and Using PrimeTime

PrimeTime is a command-line-driven tool that runs under the UNIX or Linux operating system. You enter commands at the pt_shell prompt and PrimeTime responds with text messages. PrimeTime also offers a graphical user interface (GUI) for displaying design data and analysis results in graphical form.

This chapter concisely explains how to start and use PrimeTime for static timing analysis. It includes the following sections:

Starting PrimeTime

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

Starting a PrimeTime Session

To start pt_shell (the PrimeTime command-line interface), enter the following command at the operating system prompt:

```
% pt_shell
```

PrimeTime automatically checks out a PrimeTime license and displays the initial message and the PrimeTime prompt.

```
                        PrimeTime (R)
          Version E-2010.12 -- December 6, 2010
      Copyright (c) 1988-2010 by Synopsys, Inc.
                ALL RIGHTS RESERVED

This program is proprietary and confidential ...

Initializaing gui preferences ...

pt_shell>
```

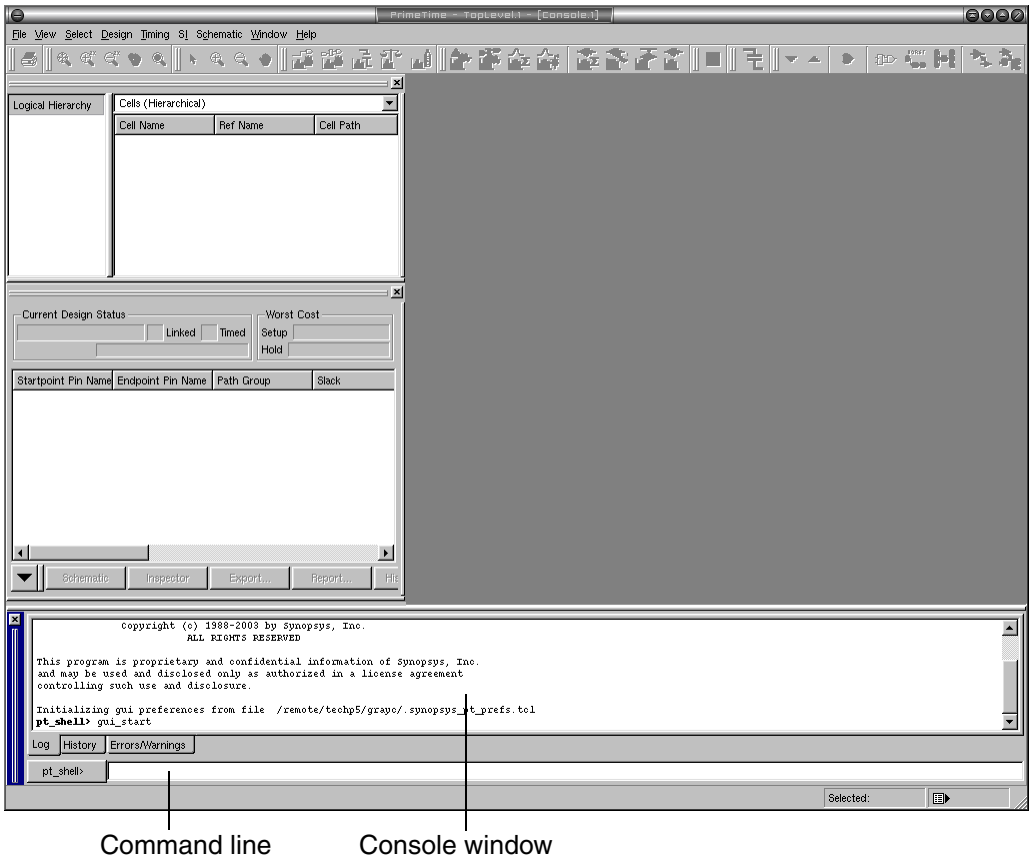
You enter commands at the pt_shell prompt (pt_shell>). PrimeTime responds with text messages in the terminal window below your command entry line.

To start the PrimeTime graphical user interface (GUI) rather than pt_shell, enter:

```
% pt_shell -gui
```

This opens a PrimeTime top-level GUI window like the one shown in Figure 2-1. You can then enter pt_shell commands in the command entry line at the bottom. PrimeTime displays system messages in the rectangular area above the command entry line.

Figure 2-1 Initial PrimeTime Top-Level GUI Window



For more information about using the PrimeTime GUI, see Chapter 4, “Graphical User Interface.”

Ending a PrimeTime Session

Before ending a PrimeTime working session, you might want to save the information set for the current design. To do so, you can use the `save_session`, `write_script`, or `write_sdc` command. For more information, see the man pages for these commands.

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

```
pt_shell> exit
Maximum memory usage for this session: 0.72 MB
CPU usage for this session: 0 seconds
Thank you for using pt_shell!
%
```

Licensing Requirements for Single-Core Analysis

You need a PrimeTime license to start pt_shell or the PrimeTime GUI console window. PrimeTime automatically checks out a license when you start PrimeTime. When you exit from PrimeTime, the license is automatically checked in, allowing others at your site to use the license.

The following additional licenses are optional:

- HDL-Compiler – To read Verilog with the `read_verilog` command and the `-hdl_compiler` option
- VHDL-Compiler – To read VHDL with the `read_vhdl` command and the `-vhdl_compiler` option
- PrimeTime PX – To analyze full-chip power dissipation including multi-VDD and power domain analysis
- PrimeTime SI – To analyze and calculate the logic effects of crosstalk noise

  A PrimeTime SI license is required for multivoltage analysis, except for the following commands and variables that are available in PrimeTime:

  - `set_operating_conditions -object_list`
  - `check_timing -signal_level`
  - `link_path_per_instance`
- PrimeTime VX – To analyze path timing behavior by considering the statistical variation and distribution of process parameters

PrimeTime automatically checks out licenses as needed and checks in licenses when you exit from PrimeTime. To check out or check in a license explicitly, use the `get_license` or `remove_license` command.

Enabling License Queuing

PrimeTime has a license queuing capability that allows your application to wait for licenses to become available if all licenses are in use. To enable this capability, set the SNPSLMD_QUEUE environment variable to `true` in the user environment. Then, on launching pt_shell, the following message is displayed:

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

When you have enabled the license queuing capability, you might run into a situation where two processes are waiting indefinitely for a license that the other process owns. Consider the following scenario:

- Two active processes, P1 and P2, are running PrimeTime.
- You have the following licenses: Two PrimeTime licenses, a PrimeTime SI license and a PrimeTime VX license.

For P1 and P2 to be active, they should have checked out one PrimeTime license each. Assume that P1 has acquired the PrimeTime VX license and P2 has acquired the PrimeTime SI license. If P1 and P2 both attempt to acquire the license held by the other process, both processes wait indefinitely. The SNPS_MAX_WAITTIME environment variable and the SNPS_MAX_QUEUETIME environment variable help prevent such situations. You can use these variables only if the SNPSLMD_QUEUE environment variable is set to `true`.

The `SNPS_MAX_WAITTIME` variable specifies the maximum wait time for the first feature license that you require, for example, starting pt_shell. Consider the following scenario:

You have two PrimeTime licenses both of which are in use and are attempting to start a third pt_shell process. The queuing capability places this last job in the queue for the specified wait time. The default wait time is 259,200 seconds (or 72 hours). If the license is still not available after the predefined time, you might see a message similar to the following:

```
Information: Timeout while waiting for feature ‘PrimeTime’. (PT-017)
```

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

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

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

As you take your design through the PrimeTime flow, the queuing capability might display other status messages as follows:

```
Information: Successfully checked out feature 'PrimeTime SI '. (PT-014)
Information: Started queuing for feature ‘PrimeTime SI'. (PT-015)
Information: Still waiting for feature 'PrimeTime SI'. (PT-016)
```

Startup Files

Each time you start a PrimeTime session, PrimeTime executes the commands contained in a set of PrimeTime startup files. You can put commands into a startup file to set variables, to specify the design environment, and to select your preferred working options.

The name of each startup file is .synopsys_pt.setup. PrimeTime checks for the presence of the file in the following directories:

1. The Synopsys installation setup directory at admin/setup. For example, if the installation directory is /usr/synopsys, the startup file name is /usr/synopsys/admin/setup/.synopsys_pt.setup.
2. Your home directory.
3. The current working directory from which you started PrimeTime.

If more than one of these directories contains a .synopsys_pt.setup file, the files are executed in the order shown above: first in the Synopsys setup directory, then in your home directory, then in the current working directory. Typically, the file in the Synopsys setup directory contains setup information for all users at your site; the one in your home directory sets your personal preferred working configuration; and the one in your current working directory sets the environment for the current project.

To suppress execution of any .synopsys_pt.setup files, use the `-no_init` option when you start PrimeTime.

Command Log File

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

PrimeTime creates the log file in the current working directory and names it pt_shell_command.log. Any existing log file with the same name is overwritten. Before you start a new PrimeTime session, make sure that you rename any log file that you want to keep.

You can specify a different name to use for the command log file by setting the `sh_command_log_file` variable in your startup file. You cannot change this variable during a working session.

Using pt_shell

The PrimeTime shell, pt_shell, is a command-driven, text-only user interface based on the Tcl scripting language. You enter commands at the pt_shell prompt. PrimeTime carries out the action for each command and reports the result in text format.

The PrimeTime command syntax is case-sensitive. Commands, command options, arguments, and variables generally consist of lowercase characters and you must enter them this way at the pt_shell prompt.

Object names in the design are also case-sensitive. For example, the following two commands are not the same because they refer to two different ports, named `Clk` and `CLK`:

```
pt_shell> create_clock -period 20.0 Clk

pt_shell> create_clock -period 20.0 CLK
```

Entering Commands Interactively

When you are working interactively at the pt_shell prompt, you can abbreviate PrimeTime command names and options to the shortest unambiguous string. For example, you can abbreviate the `get_attribute` command to `get_attr`, or the `create_clock -period 5` command to `create_cl -p 5`.

Using command abbreviations is convenient for interactive pt_shell sessions. However, you should avoid using abbreviations in script files because they are susceptible to command changes in later versions of PrimeTime. Such changes could make the abbreviations ambiguous.

When you enter a long command, you can split it across multiple lines using the backslash (\) continuation character. During entry of a long command using backslashes (or in other incomplete input situations), PrimeTime displays a secondary prompt, a question mark, for each additional line of the command. For example,

```
pt_shell> alias my_rep_tim {report_timing  \
          -from {in1 in2} -nworst 6 -delay max}
```

In this user guide, a command that cannot fit on one line is shown on multiple lines using the continuation character. However, for easier reading, the secondary prompt is usually omitted from the examples shown.

Using Command-Line Editing

To enable command-line editing, add the following line to your .synopsys_pt.setup file (the default value of the `sh_enable_line_editing` variable is `false`):

```
pt_shell> set sh_enable_line_editing true
```

Note:

You must set this variable in your .synopsys_pt.setup file. It has no impact from the command line.

Choosing an Editor Mode

To set the command line editor mode to either vi or emacs you can add the following variable to your setup file or enter it directly in your shell (valid values are `emacs` or `vi`; the default is `emacs`):

```
pt_shell> set sh_line_editing_mode editor
```

Using Command-Line Editing

After you have enabled command-line editing, you can use the following types of keystrokes:

- Up and down arrows for history recall
- Left and right arrows to move forward or backward one character
- Home and End keys to move to the beginning or end of the current line
- Emacs line editing keystrokes
- Vi editing keystrokes
- Tab for auto-completion of command, variable, and file names, aliases, and nested commands

To list the editing keys that are active in either emacs or vi mode, use the `list_key_bindings` Tcl command.

Note:

PrimeTime does not save and restore the settings of the `sh_enable_line_editing` and `sh_line_editing_mode` variables. If you restore a previously saved session, these variables retain the settings they had before the restore (for instance, if you had not enabled command-line editing in the saved session, but did have it enabled at the time you restored that session, command-line editing would still be enabled after the restore).

Changing the pt_shell Prompts

The pt_shell prompts are programmable. By default, the primary prompt is `pt_shell>` and the secondary prompt is a question mark (`?`). To change the prompt, set the `tcl_prompt1` or `tcl_prompt2` variable to the name of a procedure that displays the desired prompt. The procedure cannot take an argument. For example, to make the secondary prompt an asterisk (`*`), do the following:

```
pt_shell> proc prompt2 {} { echo -n "* " }
pt_shell> set tcl_prompt2 prompt2
prompt2
```

To make the primary prompt track the current design and the current instance, use the following procedure. (If there is no current design, the primary prompt is `pt_shell>`.)

```
proc cd_prompt {} {
  global sh_dev_null
  global synopsys_program_name
  set des ""
  redirect $sh_dev_null {set des [current_design]}
  if { $des == "" } {
    echo -n "${synopsys_program_name}> "
    return
  }
  set des [get_object_name $des]
  set inst ""
  redirect $sh_dev_null {set inst [current_instance .]}
  if { $inst == "" } {
    set sep ""
  } else {
    set sep "/"
  }
  set prmpt "${des}${sep}${inst}> "
  echo -n $prmpt
}
pt_shell> set tcl_prompt1 cd_prompt
cd_prompt
TOP> current_instance u1
TOP/u1>
```

Command Scripts

A command script is a text file containing a sequence of pt_shell commands. The setup file .synopsys_pt.setup is an example of a script. Also, the log file generated at the end of a PrimeTime session can be used as a script. You can create your own scripts, or use scripts created by others, to carry out complex or repetitive tasks.

PrimeTime can recognize script files in plain ASCII format, ASCII compressed in gzip format, and ASCII encoded into byte code format by the TclPro Compiler. To execute a script in any of these forms, use the `source` command:

```
pt_shell> source file_name
```

To execute a specified command script upon startup, use the `-f` option when you start PrimeTime:

```
% pt_shell -f file_name
```

By using the features of the Tcl command language, you can create scripts that use variables, loops, and conditional execution. The `if`, `while`, `for`, `foreach`, `break`, `continue`, and `switch` flow control commands determine the execution order of other commands.

Any line of text in a script file that begins with the pound sign (`#`) is a comment, which PrimeTime ignores. Any text from a semicolon and pound sign (`;#`) to the end of a line is also considered comment text.

Running Scripts in Batch Mode

To run PrimeTime in batch mode, use the `-f` option when you invoke PrimeTime to source the script, and redirect the output to a file. For example,

```
% pt_shell -f big_analysis.tcl > big_analysis.out &
```

PrimeTime redirects output and error messages to the specified file (in this example, big_analysis.out), which you can read at any time. If your script contains a syntax error, PrimeTime stops and waits for input unless the `sh_continue_on_error` variable is set to `true`.

Make sure you end the script with the `quit` or `exit` command. Otherwise, the pt_shell prompt does not appear and you do not know when the script has finished executing. If your script does not end with the `quit` command, PrimeTime waits for input. In that case, bring the process to the foreground, then type `quit` or `exit` to end the session.

Command Results

Every PrimeTime command has a result. Many commands result in “1” to indicate success or “0” to indicate failure. For example,

```
pt_shell> create_clock -period 6.67 [get_ports clk1]
1
```

For many other commands, the result is a collection. For example, the result of the `get_ports` command is a collection of ports:

```
pt_shell> get_ports IN*
{"IN1", "IN2", "IN3", "IN4"}
```

The `get_ports IN* command` creates a collection of all ports in the design beginning with the letters `IN`.

In PrimeTime, commands are often nested, one command within another, so that the result of one command is used as an argument for another. Each nested command is enclosed in square brackets (`[]`). This command structure is common for performing operations on design elements.

For example, the `set_input_delay` command sets a timing constraint on one or more specified input ports. You can gather a collection of input ports with the `get_ports` command and then pass the result to the `set_input_delay` command:

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

The `get_ports IN*` command creates a collection, which is then used as the second argument of the `set_input_delay` command. The effect of the entire command is to set the input delay to a specific value for all ports beginning with the letters `IN`.

The output of some commands is a report, such as the `man` or `report_timing` command. By default, PrimeTime issues a long report all at once. To have PrimeTime pause between each screenful of text (like using the `more` command under UNIX), set the page mode variable to `true`:

```
pt_shell> set sh_enable_page_mode true
```

Variables

PrimeTime offers many options that you control by setting variables. To set a variable, you use the `set` command:

```
pt_shell> set variable_name variable_setting
```

When you set a variable, the displayed result is the new setting for the variable. For example,

```
pt_shell> set sh_enable_page_mode true
```

If you attempt to set a variable to an invalid value for that variable, PrimeTime responds with an error message. For example,

```
pt_shell> set sh_enable_page_mode maybe
Error: can't set "sh_enable_page_mode": invalid value:
       use true or false
Use error_info for more info. (CMD-013)
```

If you mistype a system variable name, PrimeTime interprets it as a request to define a new, user-defined variable. In that case, PrimeTime issues a message indicating that a new variable has been created:

```
pt_shell> set sh_enable_pag_mdoe true
Information: Defining new variable 'sh_enable_pag_mdoe'.
  (CMD-041)
true
```

To find out the current setting for a variable, use the `printvar` command. For example,

```
pt_shell> printvar sh_enable_page_mode
sh_enable_page_mode  = "false"
```

You can use the wildcard character (*) to view multiple related settings because related commands often begin with the same prefix. For example, to see a list of shell-related variables:

```
pt_shell> printvar sh_*
sh_arch              = "sparcOS5"
sh_command_abbrev_mode = "Anywhere"
sh_command_log_file  = ""
sh_continue_on_error = "false"
...
```

You can use multiple wildcard characters. For example, to see a list of all clock-related variables:

```
pt_shell> printvar *clock*
create_clock_no_input_delay = "false"
extract_model_clock_transition_limit = "5"
extract_model_num_clock_transition_points = "5"
...
```

The `printvar` command used by itself, without any arguments, lists all the variables and their settings:

```
pt_shell> printvar
arch                  = "sparcOS5"
auto_index            = ""
auto_link_disable     = "false"
auto_wire_load_selection = "true"
...
```

Tcl Interface

The pt_shell interface is based on the Tcl scripting language, like the Tcl shell of Design Compiler (dc_shell-t). This means that you can use features of the Tcl language such as user-defined variables, procedures, conditional execution, lists, and expressions.

The general features of the Tcl language are beyond the scope of the PrimeTime documentation. For this type of information, see any reference book on Tcl. For information about how the features of Tcl are used in pt_shell, see Chapter 14, “Command Interface and Tcl”.

PrimeTime Documentation and Online Help

If you need help while using PrimeTime, information is readily available from several resources. The following sections provide more information about these resources:

- Using the help Command
- Using the man Command
- PrimeTime Manuals
- SolvNet Articles

Using the help Command

The `help` command provides concise information about the PrimeTime commands. You can get a partial or complete list of commands or view the syntax of a particular command.

The `help` command by itself shows a list of all PrimeTime commands, organized by command group. For example,

```
pt_shell> help
 ...
Design Compiler Emulation:
 cell_of, drive_of, filter, find, link, load_of, read_file,
 remove_wire_load_min_block_size

PrimeTime Backward Compatibility:
 set_load, set_drive

Netlist Editing:
 ...

Procedures
 ...

Builtins:
 ...

Default Command Group
 ...
```

You can use wildcard characters to restrict the scope of the command list or to find the name of a command that you cannot remember exactly. For example, to find all commands containing the string “clock”:

```
pt_shell> help *clock*
 all_clocks      # Create a collection of all clocks in design
 clock           # Builtin
 create_clock    # Create a clock object
 create_generated_clock # Create a generated clock object
 derive_clocks   # Create clocks on source pins in design
 ...
```

For a concise description of a particular command, use `help` with just the command name. For example,

```
pt_shell> help set_input_delay
 set_input_delay      # Set input delay on ports or pins
```

To see the full command syntax, including options and arguments, use the `-verbose` option. For example,

```
pt_shell> help -verbose set_input_delay
 set_input_delay      # Set input delay on ports or pins
   [-clock clock_name]  (Relative clock)
   [-reference_pin pin_port_name]
                        (Relative pin or port)
   [-clock_fall]        (Delay is relative to falling edge
                         of clock)
   [-level_sensitive]   (Delay is from level-sensitive latch)
   [-rise]              (Specifies rising delay)
   [-fall]              (Specifies falling delay)
   [-max]               (Specifies maximum delay)
   [-min]               (Specifies minimum delay)
   [-add_delay]         (Don't remove existing input delay)
   ...
```

An alternative method to get the same help is to enter the command name directly and use the `-help` option. For example,

```
pt_shell> set_input_delay -help
Usage: set_input_delay   # Set input delay on ports or pins
   [-clock clock_name]  (Relative clock)
   [-reference_pin pin_port_name]
                        (Relative pin or port)
   [-clock_fall]        (Delay is relative to falling edge
                         of clock)
   ...
```

For more information about a command, see the man page.

Using the man Command

To obtain detailed information about any command, variable, or system message in PrimeTime, use the `man` command. It works much like the UNIX `man` command. At the pt_shell prompt, type the word man followed by the command, variable, or message code. PrimeTime responds with a detailed description. For example, to view the `update_timing` command man page,

```
pt_shell> man update_timing
```

Man Pages for Commands

The man page for a command shows the command syntax, describes each option and argument, describes the how the command operates, provides some examples, and lists some related commands. For example,

```
pt_shell> man read_db
2.  Synopsys Commands   Command Reference        read_db
Copyright 2010 Synopsys, Inc. All rights reserved.

NAME
     read_db        Reads in one or more design or library
                    files in Synopsys database (db) format.

SYNTAX
     string read_db [-netlist_only] [-library] file_names

     list file_names
ARGUMENTS
     -netlist_only  For designs only; ignored for libraries.
                    Indicates that only the netlist is to be
                    read; attributes are not to be read.
                    This can greatly increase performance
                    while reading designs
      ...

DESCRIPTION
      ...
EXAMPLES
      ...

SEE ALSO
      ...
```

Man Pages for Variables

The man page for a variable shows the variable name, variable type (string, list, Boolean, integer, or floating-point number), the default setting, and a description of what the variable controls. For example,

```
pt_shell> man search_path
3.  Attributes and Variables   Command Reference      search_path
Copyright 2010 Synopsys, Inc. All rights reserved.

NAME
        search_path
        Shows  a list of directory names that contain design and library
        files that are specified without directory names.

TYPE
        list

DEFAULT
        "" (empty)

DESCRIPTION
        A list of directory names that specifies which directories
        to search for design and library files that are
        specified without directory names ...

        ...

SEE ALSO
        link_design(2)
        printvar(2)
        read_db(2)
        ...
```

Man Pages for Messages

When PrimeTime issues an informational, warning, or error message, it shows a letter-number code at the end of the message. For example,

```
pt_shell> update_timing
Error: Current design is not defined. (DES-001)
```

The message code is DES-001. To get more information about the warning or error that occurred, use the `man` command with the message code. Be sure to type uppercase letters for the error code. For example,

```
pt_shell> man DES-001
N.  Messages          Command Reference           messages

Copyright 2010 Synopsys, Inc. All rights reserved.

NAME
        DES-001 (error) Current design is not defined.

DESCRIPTION
        The current design is not defined.  Many commands
        require that the current design is set.

WHAT NEXT
        You must read a design database file and link a
        design.
E-2010.12         Copyright (c) 2010 Synopsys, Inc. All rights reserved.
```

PrimeTime Manuals

Information about PrimeTime is available from the following manuals:

- *PrimeTime Fundamentals User Guide*
- *PrimeTime Advanced Timing Analysis User Guide*
- *PrimeTime Modeling User Guide*
- *PrimeTime Suite Quick Reference*
- *PrimeTime PX User Guide*
- *PrimeTime SI User Guide*
- *PrimeTime VX User Guide*
- *PrimeTime Distributed Multi-Scenario Analysis User Guide*

These manuals are available from the Synopsys SolvNet Web page on the Internet at https://solvnet.synopsys.com

To read these manuals in electronic form, you need the Adobe Acrobat Reader (available for free from http://www.adobe.com).

The *PrimeTime Fundamentals User Guide* (the manual you are now reading) introduces basic concepts of static timing analysis and using PrimeTime, including design data, timing paths, clocks, timing exceptions, case analysis, and parasitic back-annotation.

The *PrimeTime Advanced Timing Analysis User Guide* provides information about more advanced topics such as design rule checking, bottleneck analysis, minimum-maximum analysis, on-chip variation (OCV) analysis, and context characterization.

The *PrimeTime Modeling User Guide* describes how to create, validate, and use timing models in PrimeTime, including interface logic models, and extracted models.

The *PrimeTime Suite Quick Reference* is a pocket-sized reference book that lists and briefly describes all the PrimeTime commands and variables. It is a concise summary of the man page text.

The *PrimeTime PX User Guide* describes the PrimeTime PX tool - a static and dynamic full-chip power analysis tool for complex multimillion-gate designs, intended for use within the PrimeTime environment. PrimeTime PX is licensed separately from the main PrimeTime product.

The *PrimeTime SI User Guide* describes the PrimeTime SI (signal integrity) tool and how to perform crosstalk analysis. PrimeTime SI is licensed separately from the main PrimeTime product.

The *PrimeTime VX User Guide* describes the PrimeTime VX tool that you can use to add variation-aware analysis capabilities to PrimeTime. PrimeTime VX is licensed separately from the main PrimeTime product.

The *PrimeTime Distributed Multi-Scenario Analysis User Guide* describes how to create, use, and verify chip-level timing models, including quick timing models, interface logic models, interface timing models, and extracted models.

SolvNet Articles

Users often contact Synopsys customer support with technical questions about Synopsys products, including PrimeTime. The answers to the most frequently asked questions are put into a database that you can access through the Internet. This question-and-answer article database is part of a larger network of Synopsys online services called SolvNet.

To access the SolvNet database,

1. Go to the SolvNet Web page at the following address:

   https://solvnet.synopsys.com

2. When prompted, enter your user name and password. If you do not already have a Synopsys user name and password, click on the link to the registration page.

3. In the SolvNet page, enter your search criteria in the search field and click Search. SolvNet displays a “hit list” showing the titles of articles that match your search criteria.

4. Click the title of interest and read the article.

Here is an excerpt from a typical SolvNet article:

```
Check Calculated Source Latency on Generated Clocks

QUESTION:

When I do a report_timing -path full_clock in PrimeTime,
my generated clock path is not expanded fully. How can I
check the calculated latency on a propagated derived clock
and see which cells it depends on?

ANSWER:

Following is the full_clock timing report from PrimeTime.
The set_propagated_clock command was applied both to the
clock I generated using create_generated_clock ...
```

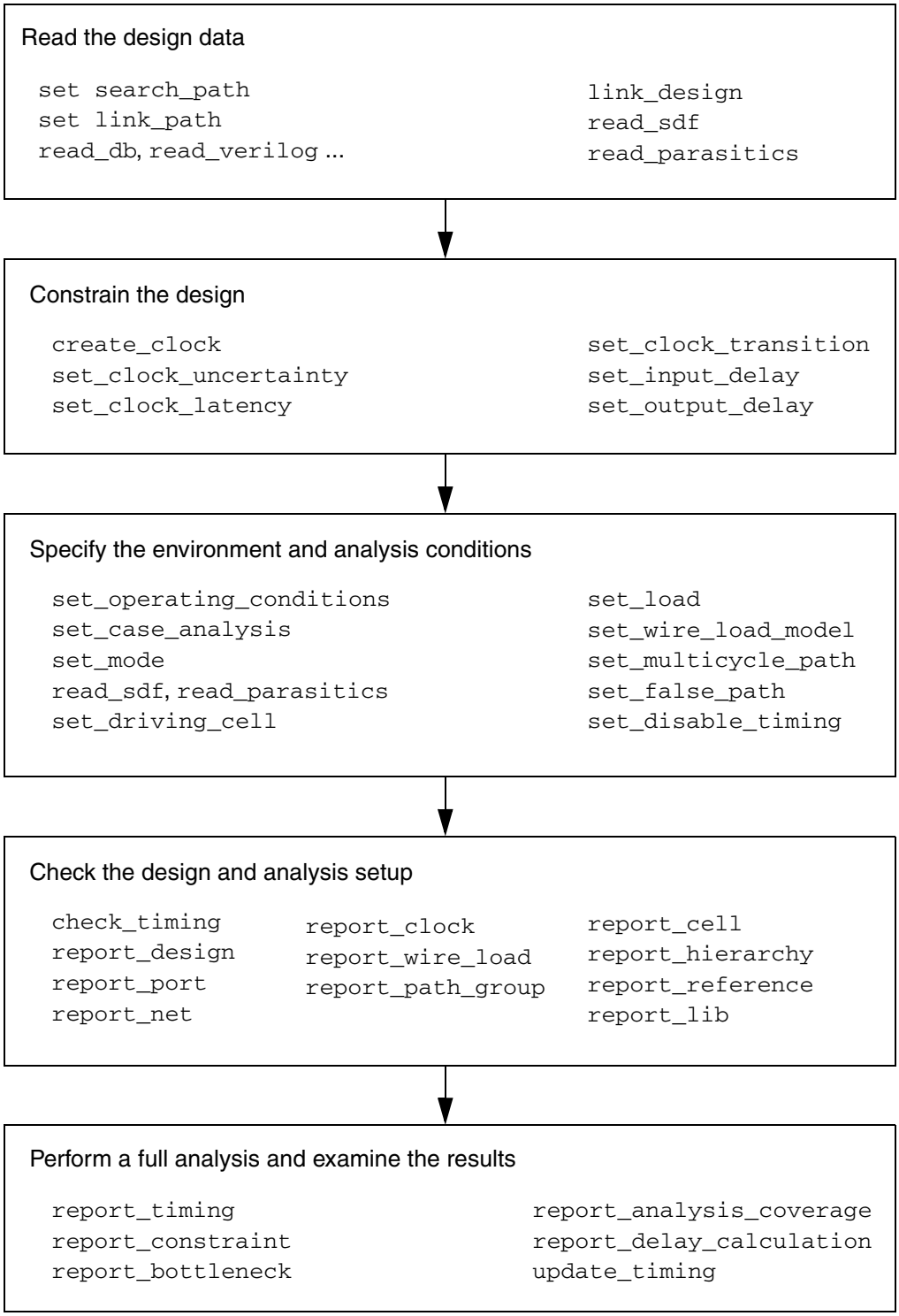
Analysis Flow in PrimeTime

To successfully perform timing analysis for a design, you need to do certain steps in the proper order. A typical analysis session consists of the following steps:

1. Read in the design data (a gate-level netlist and associated technology libraries).
2. Constrain the design by specifying the clock characteristics, input timing conditions, and output timing requirements.
3. Specify the environment and analysis conditions such as operating conditions, case analysis settings, net delay models, and timing exceptions.
4. Check the design data and analysis setup parameters in preparation for a full timing analysis.
5. Perform a full timing analysis and examine the results.

Figure 2-2 shows the basic steps in the PrimeTime analysis flow and some commonly used commands for each step.

Figure 2-2 PrimeTime Analysis Flow



The following sections provide an overview of the basic analysis steps and commands used in PrimeTime. Note that this overview only describes some of the more common work flows and commands. For more complete information, see the later chapters in this manual, the other manuals in the PrimeTime documentation set, and the man pages for the PrimeTime commands.

Reading the Design Data

The first step is to read in the gate-level design description and associated technology library information. PrimeTime accepts design descriptions and library information in .db format and .ddc and gate-level netlists in Verilog and VHDL formats. The corresponding commands for reading design files are `read_db`, `read_ddc`, `read_verilog`, and `read_vhdl`.

After you read in a set of hierarchical design files, the `link_design` command resolves all references between different modules in the hierarchy and builds an internal representation of the design for timing analysis.

Set variables using the `set` command to control the behavior of PrimeTime. The `search_path` variable specifies a list of directories from which to read the design and library files, so that you do not need to specify a full path each time you read in a file. The `link_path` variable specifies where and in what order PrimeTime looks for design files and library files for linking the design. For example,

```
pt_shell> set search_path ". /u/proj/design /u/proj/lib"
pt_shell> set link_path "* STDLIB.db"
```

If layout of the chip has been done, you can get more accurate results by back-annotating the design with detailed delay or parasitic information. To back-annotate the design with delay information provided in an SDF file, use the `read_sdf` command. To back-annotate the design with parasitic capacitance and resistance information, use the `read_parasitics` command. PrimeTime accepts detailed parasitic data RSPF, SPEF, and SBPF formats.

For a large chip design, you can read in and analyze the design in hierarchical stages, using either a bottom-up or top-down approach. For more efficient analysis at higher levels of hierarchy, you can use timing models rather than netlists to represent the lower-level blocks in the hierarchy. You can create a timing model by extracting it from a lower-level netlist or by executing a sequence of PrimeTime commands to generate an approximate “quick timing model.”

Parallel Parasitic Reading

When performing an analysis of a design, the following types of information must be read in before the timing update can take place:

- Netlists using the `read_verilog`, `read_vhdl`, `read_ddc`, and `read_milkyway` commands
- Annotations using the `read_parasitics` and `read_sdf` commands
- Timing constraints and exceptions using the `source` and `read_sdc` commands
- User-defined scripts and custom pre-update reports using the `source` command

Usually, PrimeTime progresses through each command in the analysis script file sequentially, reading the specified information in and continuing with execution after the file has been completely processed. Reading detailed parasitics files using the `read_parasitics` command allows the parsing of parasitics files and back annotation to happen in parallel with other unrelated commands in the script.

To obtain the most runtime benefit from the parallel back annotation of parasitics data, you must write the PrimeTime script in a way that all `read_parasitics` commands immediately follow the design link command in the script file. Doing so ensures having the best throughput of a parasitics-based flow before you start timing analysis with the `update_timing` command.

In addition to the `read_parasitics` command, the parallel loading of parasitics data also covers commands such as the `report_annotated_parasitics`, `complete_net_parasitics`, and `report_annotated_parasitics` commands. Changes made to parasitics data, such as ECO, are also handled.

It is worth noting the change made to the output from the parasitics loading and reporting commands just discussed. To obtain the maximum runtime throughput, these parasitics-related commands have the outputs written to a log file that you can define by using the `parasitics_log_file` variable. The default value is the parasitics_command.log file in the current working directory. You are not required to make changes to the individual command redirections. If left unchanged in the script, these log files appear empty because the command output is written into the file specified by the `parasitics_log_file` variable. For more information about the `read_parasitics` command, see the man page.

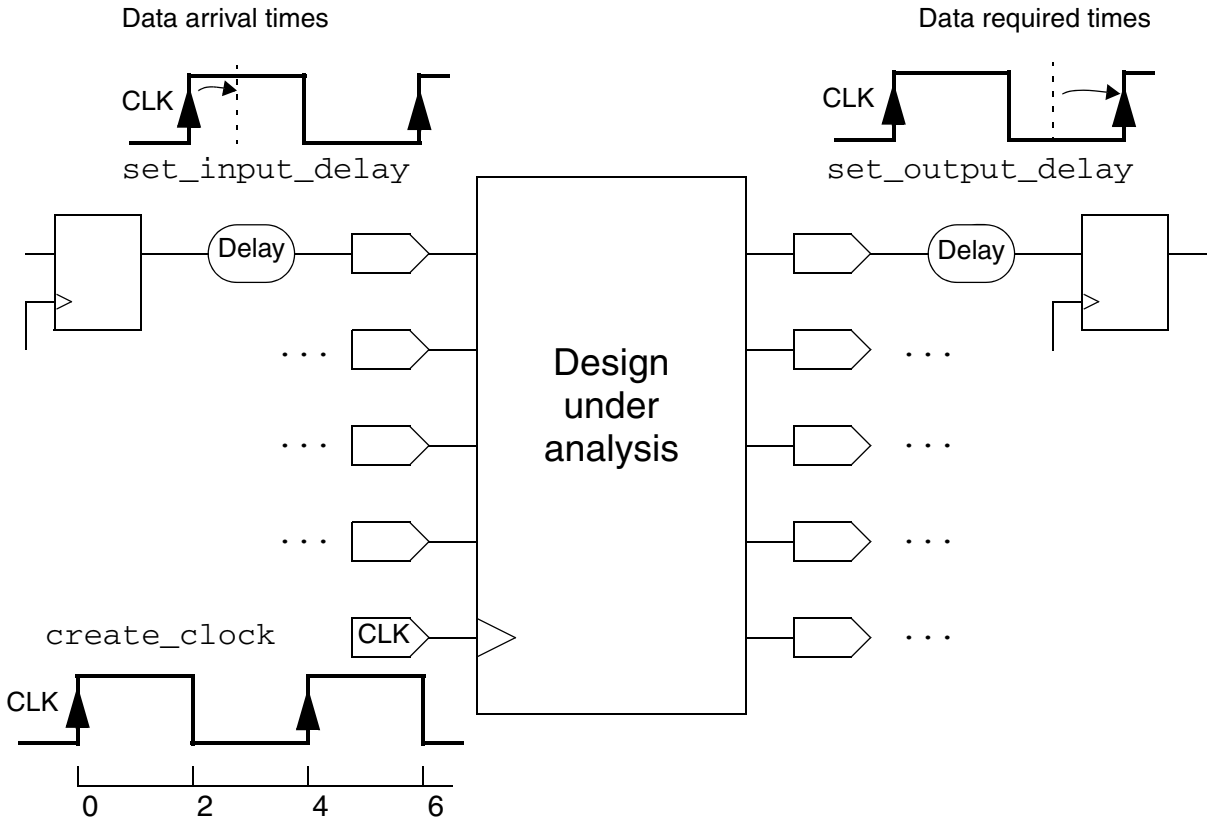
Constraining the Design

To perform timing analysis, PrimeTime needs information about the design-level timing constraints:

- Clock characteristics
- Arrival times of signal transitions at each input port
- Required times of signal transitions at each output port

These design-level timing constraints (also called timing assertions) are illustrated in Figure 2-3.

Figure 2-3 Design-Level Timing Constraints



Specifying Clocks

The `create_clock` command specifies the characteristics of a clock, including the clock name, source, period, and waveform. PrimeTime needs this information in order to perform the constraint checking on clocked sequential elements. PrimeTime can handle timing checks between multiple clocks used in the design, even if they have different periods and phases, as long as the clocks are synchronous with respect to each other.

You can further specify the clock network characteristics by using `set_clock_uncertainty`, `set_propagated_clock`, `set_clock_transition`, `set_clock_latency`, and so on. PrimeTime takes these characteristics into account to determine the worst-case timing conditions for each constraint check.

Some designs have internally generated clocks, such as clock dividers. You can use the `create_generated_clock` command to specify the characteristics of each generated clock and its dependency on a master clock. If you later change the master clock (for example, by setting a new period), PrimeTime automatically adjusts the characteristics of the generated clock accordingly.

Specifying Input Timing Conditions

To do constraint checking at the inputs of the design, PrimeTime needs information about the arrival times of signals at the inputs. The `set_input_delay` command specifies the minimum and maximum amount of delay from a clock edge to the arrival of a signal at a specified input port. PrimeTime uses this information to check for timing violations at the input port and in the transitive fanout from that input port.

Specifying Output Timing Requirements

To do constraint checking at the outputs of the design, PrimeTime needs information about the timing requirements at the outputs. The `set_output_delay` command specifies the minimum and maximum amount of delay between the output port and the external sequential device that captures data from that output port. This setting establishes the times at which signals must be available at the output port in order to meet the setup and hold requirements of the external sequential element.

Specifying the Environment and Analysis Conditions

PrimeTime allows you to specify the operating environment and the conditions for timing analysis. For example, you can do the following actions:

- Specify the process, temperature, and voltage operating conditions, as characterized in the technology library
- Apply case analysis and mode analysis to restrict the operating modes of the device under analysis

- Specify driving cells at input ports and loads at output ports
- Specify timing exceptions for paths that do not conform to the default behavior assumed by PrimeTime
- Specify the wire load model or back-annotated net information used to calculate net delays

Minimum and Maximum Operating Conditions

Semiconductor device parameters can vary with conditions such as fabrication process, operating temperature, and power supply voltage. The ASIC vendor typically characterizes the device parameters in the laboratory under varying conditions, and then specifies the parameter values under different sets of conditions in the technology library.

In PrimeTime, the `set_operating_conditions` command specifies the operating conditions for analysis, so that PrimeTime can use the appropriate set of parameter values in the technology library.

PrimeTime offers three analysis modes with respect to operating conditions, called the single, best-case/worst-case, and OCV modes.

In the single operating condition mode, PrimeTime 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, PrimeTime 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. This mode lets you check both extremes in a single analysis run, thereby reducing overall runtime for a full analysis.

In the OCV mode, PrimeTime 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 data path and minimum delays for the clock path. For a hold check, it uses minimum delays for the data path and maximum delays for the clock path.

An optional analysis technique called clock reconvergence pessimism removal (CRPR) increases the accuracy of OCV analysis by ensuring that each individual circuit element effectively uses either the minimum or maximum delay, but not both, for any given setup or hold check.

Case Analysis and Mode Analysis

A chip design usually has different operating modes, such as normal operating or “mission” mode, test mode, scan mode, reset mode, and so on. Typically, you analyze the timing for each mode separately.

To place the design into a specific operating mode, you can use a technique called case analysis. You apply a constant value or a specific transition to an input port with the `set_case_analysis` command. For example, if the test mode is activated by an active-low input pin, you can set that input to 0 for analysis in test mode, or set it to 1 for analysis in mission mode. If there is an active-low reset input, you can analyze the circuit timing for a reset operation by applying a high-to-low transition on the reset pin.

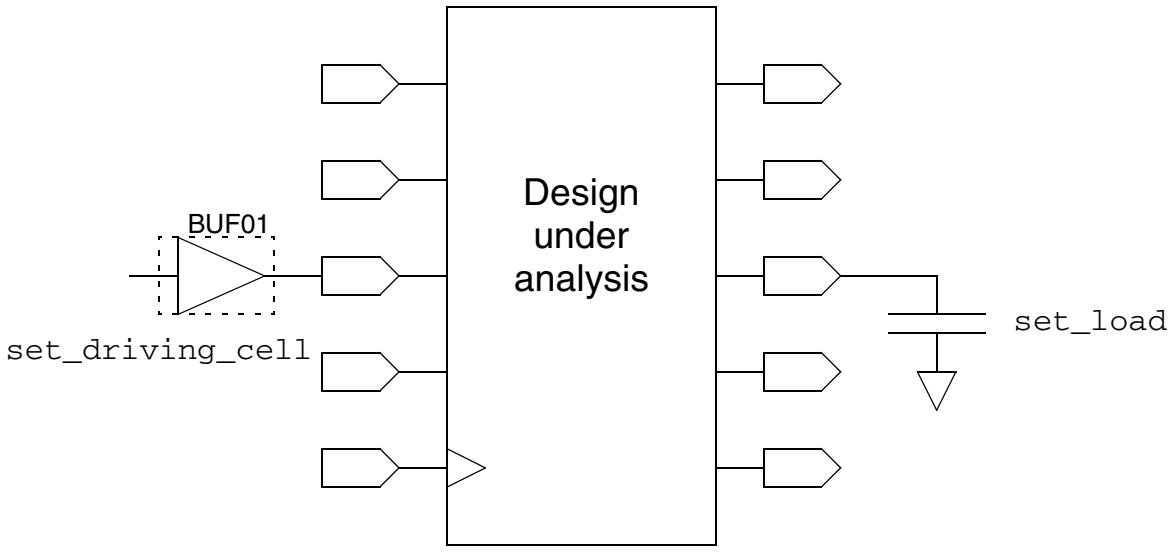
In some designs, library cells can have operating modes. For example, a RAM cell might have a read mode and a write mode, each with its own timing characteristics. If a cell has operating modes defined in it, you can restrict the scope of the analysis by using the `set_mode` command. This command lets you specify the current operating mode for a cell, thereby eliminating from consideration the timing constraints of the inactive modes.

Driving Cells and Port Loads

The external driver that drives an input port has impedance and parasitic load characteristics that can affect the signal timing. To more accurately take these effects into account, you can use the `set_driving_cell` command. This command specifies the name of a library cell that is presumed to be driving the input port. Using a library cell as a driver allows PrimeTime to more accurately calculate the port delay times and transition times, especially for library cells having delays with nonlinear dependence on capacitance.

You can use the `set_load` command to specify the amount of capacitance on a port or net, allowing PrimeTime to more accurately calculate the effects of the load on port or net delay. Figure 2-4 shows how PrimeTime can use a specified driving cell or output load to analyze the timing of a design.

Figure 2-4 Driving Cells and Output Load Examples



Timing Exceptions

For a valid timing analysis, you need to specify the paths that are not intended to operate according to the default setup/hold behavior assumed by PrimeTime. These exceptions include false paths, multicycle paths, and paths that must conform to constraints that you specify explicitly with the `set_min_delay` or `set_max_delay` command. You can also eliminate paths from timing consideration by using the `set_disable_timing` command.

By using the `report_timing -false` command, you can have PrimeTime automatically detect false paths based on the logic configuration of the design.

Wire Load Models and Back-Annotated Delay

To accurately calculate net delays, PrimeTime needs information about the parasitic loads of the wire interconnections. Before placement and routing have been completed, PrimeTime estimates these loads by using wire load models provided in the technology library. The `set_wire_load_model` command specifies which wire load model to use for the current analysis.

After placement and routing, you should back-annotate the design with detailed net delay information using the `read_sdf` command or detailed parasitic resistance and capacitance information using the `read_parasitics` command. This information overrides the wire load models, allowing PrimeTime to perform a layout-accurate timing analysis.

Checking the Design and Analysis Setup

Before you begin a full analysis, it is a good idea to check the characteristics of the design such as the hierarchy, library elements, ports, nets, cells; and the analysis setup parameters such as clocks, wire load models, input delay constraints, and output delay constraints.

The `check_timing` command checks for constraint problems such as undefined clocking, undefined input data arrival times, and undefined output data required times. 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.

When the check_timing command finds potential problems, it reports them in the following manner:

```
Information: Checking 'no_clock'.
Warning: There are 4 register clock pins with no clock.
Information: Checking 'no_input_delay'.
Information: Checking 'unconstrained_endpoints'.
Information: Checking 'generic'.
Information: Checking 'latch_fanout'.
Warning: There are 2 level-sensitive latches which fanout to themselves.
Information: Checking 'loops'.
Warning: There are 6 timing loops in the design.
Information: Checking 'generated_clocks'.
```

Warnings reported by the check_timing command do not necessarily indicate true design problems. To obtain more information, you can use a variety of report commands to get information about the characteristics of the design and the timing constraints that have been placed on the design.

Use the -no_input_delay option to warn if there are no clock related delays specified on an input port. If there is no input delay specified on an unclocked input port, the check_timing command does not generate warnings.

These are some of the more common report commands:

- report_design - Lists the attributes of the design, including the chosen operating conditions, wire load information, and design rules.
- report_port - Lists the ports and shows port information such as the direction, pin capacitance, wire capacitance, input delay, output delay, related clock, design rules, and wire load information.
- report_net - Lists the nets and shows net information such as the fanin, fanout, capacitance, wire resistance, number of pins, net attributes, and connections.
- report_cell - Lists the cells used in the design and cell information such as the library name, input names, output names, net connections, cell area, and cell attributes.
- report_hierarchy - Generates a hierarchical list of submodules and leaf-level cells in the current design.
- report_reference - Generates a list of hierarchical references, showing for each submodule the reference name, unit area, number of occurrences, total area, and cell attributes.
- report_lib - Generates a report on a specified library showing the time units of the library, capacitance units, wire load information, defined operating conditions, logic trip-point thresholds, and names of library cells.

- `report_clock` - Generates a report on the clocks defined for the design, showing for each clock the name, period, rise and fall times, and timing characteristics such as latency and uncertainty.
- `report_wire_load` - Shows the wire load models set on the current design or on specified cells.
- `report_path_group` - Generates a report on the path groups in the design. PrimeTime organizes timing paths into groups based on the conditions at the path endpoints.

Performing a Full Analysis

After you load and constrain the design and specify the conditions for analysis, you can perform a full analysis and examine the results. The following commands are often used for timing analysis:

- report_timing
- report_constraint
- report_bottleneck
- report_global_slack
- report_analysis_coverage
- report_delay_calculation

report_timing

The `report_timing` command is perhaps the most flexible and powerful PrimeTime analysis command. It provides general or more information about the timing of the whole design, a group of paths, or an individual path. The command options let you specify the types of paths reported, the scope of the design to search for the specified paths, and the type of information included in the path reports.

By default, the `report_timing` command by itself, without any options, reports the single path with the worst setup timing slack in each path group. For example,

```
pt_shell> report_timing

****************************************
Report : timing
Design : FP_SHR
****************************************

Operating Conditions:
Wire Loading Model Mode: top
```

```
  Startpoint: a (input port)
  Endpoint: c_d (output port)
  Path Group: default
  Path Type: max

  Point                                 Incr       Path
-----------------------------------------------------------
  input external delay                 10.00      10.00 r
  a (in)                                0.00      10.00 r
  m1/Z (MUX21H)                         1.00      11.00 r
  u1/S (FA1)                            1.00      12.00 r
  c_d/Z (AN2)                           1.00      13.00 r
  c_d (out)                             0.00      13.00 r
  data arrival time                               13.00

  max_delay                            15.00      15.00
  output external delay               -10.00       5.00
  data required time                               5.00
-----------------------------------------------------------
  data required time                               5.00
  data arrival time                              -13.00
-----------------------------------------------------------
  slack (VIOLATED)                                -8.00
```

This report includes the following information about each reported path:

- Path startpoint, endpoint, and intermediate points
- Incremental and cumulative delay at each point along the path
- Final data arrival time at the endpoint
- Time at which the data must arrive at the endpoint to meet the constraint (“data required time”)
- Setup timing slack (required time minus arrival time)

In the `report_timing` command, the `-delay_type` option specifies the type of timing checks to report. You can set the delay type to maximum using the `max` value for setup checks, minimum using the `min` value for hold checks, maximum rise using the `max_rise` value for setup checks having a rising-edge transition at the path endpoint, and so on.

The `-from`, `-to`, `-through`, `-rise_through`, and other options specify the scope of the design to be reported. For example, using `-rise_from {get_ports IN1}` causes the command to report only the paths that begin with a rising-edge transition at input port IN1. The design objects you specify with these options can be ports, pins, nets, or clocks.

Additional options let you specify the number of worst-slack paths reported, the slack threshold that triggers path reporting, and the level of detail included in the path reports. You can specify whether to include transition times, capacitance values, and net information in the path reports. Other command options allow you to automatically find false paths, true paths, or justification vectors for reported paths.

You can use the `-start_end_pair` option to report paths for each pair of startpoints and endpoints, based on connectivity. Unlike other options for the `report_timing` command, this option sorts paths based on endpoints, not slack. This option can lead to long runtime. For more information about this or any other option, see the `report_timing` man page.

report_constraint

The `report_constraint` command reports the results of constraint checking done by PrimeTime. You can obtain information such as the location of the worst violation, the amount by which the constraint is met or violated, and the calculation of the delays used for checking the constraint.

In the `report_constraint` command, you specify the type of checking you want reported and the level of detail to be reported. The types of checking you can report include maximum delay (setup), minimum delay (hold), and design rules, such as maximum capacitance, maximum transition time, maximum fanout, minimum pulse width, clock-gating checks, recovery checks, and removal checks.

A typical way to use this command is to get a detailed report on all constraint violations in the design. For example, the following report shows a total of four violations (two setup violations and two design rule violations):

```
pt_shell> report_constraint -all_violators -verbose

****************************************
Report : constraint
         -all_violators
         -verbose
Design : led
Version: 1997.01-development
Date   : Fri Aug  2 17:12:18 1996
****************************************

Startpoint: b (input port)
Endpoint: z5 (output port)
Path Group: default
Path Type: max
```

```
Point                                          Incr       Path
-------------------------------------------------------------
input external delay                           0.00       0.00 r
b (in)                                         0.00       0.00 r
U5/Z (IV)                                      1.32       1.32 f
U3/Z (NR2)                                     3.35       4.67 r
U18/Z (AO6)                                    0.73       5.40 f
U22/Z (AO4)                                    1.42       6.82 r
z5 (out)                                       0.00       6.82 r
data arrival time                                         6.82

max_delay                                      6.50       6.50
output external delay                          0.00       6.50
 data required time                                       6.50
-------------------------------------------------------------
data required time                                        6.50
data arrival time                                        -6.82
-------------------------------------------------------------
slack (VIOLATED)                                         -0.32

Startpoint: c (input port)
Endpoint: z3 (output port)
Path Group: default
Path Type: max

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

max_delay                                      6.50       6.50
output external delay                          0.00       6.50
data required time                                        6.50
-------------------------------------------------------------
data required time                                        6.50
data arrival time                                        -6.57
-------------------------------------------------------------
slack (VIOLATED)                                         -0.07


  Net: a

  max_transition         1.00
- Transition Time        1.26
------------------------------
  Slack                -0.26  (VIOLATED)
```

```
  Net: a

  max_fanout             5.00
- Fanout                 7.00
------------------------------
  Slack                 -2.00  (VIOLATED)
```

report_bottleneck

The `report_bottleneck` command generates a report on the timing bottlenecks in the design. When a design has a large number of timing violations, bottleneck analysis can help you find the places in the design most likely to benefit from design changes, such as gate resizing or resynthesis under new constraints.

To report bottleneck information for the current design, arrival totals and slacks must be available throughout the design, not just at endpoints. The optimal flow (in terms of CPU usage) to use this command within PrimeTime is as follows:

1. Set the `timing_save_pin_arrival_and_slack` variable to `true`.
2. Update timing by using the `update_timing` command.

   Note:
   If you intend to use the `report_bottleneck` command as part of your flow, it is recommended that you set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update.
3. Use the `report_bottleneck` command to report bottleneck information for the current design.

A bottleneck is a point in the design that contributes to multiple timing violations. By default, the `report_bottleneck` command lists the 20 worst leaf-level cells in the design with respect to bottleneck cost. The bottleneck cost of a cell is the number of paths through the cell that cause timing violations.

Here is an example of a bottleneck report:

```
pt_shell> report_bottleneck

 ...
                                                  Bottleneck
Cell                        Reference             Cost
-----------------------------------------------------------
ipxr/ir1/i2/i0/u3           DFF1A                 39656.00
ipxr/ir1/U14                INV2                  39654.00
ipxr/ir1/U16                INV8                  39654.00
ipxr/ir1/i2/i0/u4           DFF1A                 31806.00
 ...                          ...                   ...
```

The report lists the cell instances in order, starting with the cell having the highest bottleneck cost. You can then examine the fanin and fanout logic of each cell to determine the possible cause of the timing bottleneck. Next, decide whether to try a different cell or resynthesize the design using new constraints.

report_global_slack

The `report_global_slack` command displays the slack of specified pins and ports. By default, all pins, except those of hierarchical cells and port of the design, are reported.

To report slacks throughout the current design, use this command as follows:

1. Set the `timing_save_pin_arrival_and_slack` variable to `true`. By setting this variable to `true`, current design arrival totals and slacks are available on all pins, not just at endpoints.

   Note:
   Setting this variable to `true` after a timing update, causes the design to time again. To avoid this additional cost, you should set the variable to `true` before the first timing update.

2. Use the `report_global_slack` command to report global slack information for the current design.

The following example generates a report of all slack violations in the current design:

```
pt_shell> report_global_slack -significant_digits 4 -nosplit

****************************************
Report : report_global_slack
Design : tc1
****************************************

Max_Rise      Max_Fall      Min_Rise      Min_Fall   Point
-----------------------------------------------------------
-1.4869       -1.6330        1.5669        1.7130   ff2/D
-1.4869       -1.6330        1.5669        1.7130   in2/Z
-1.6330       -1.4869        1.7130        1.5669   in2/A
-1.6330       -1.4869        1.7130        1.5669   in1/Z
```

The `report_global_slack` command is primarily used when performing pin slack post-processing and analysis outside of PrimeTime.

report_analysis_coverage

The `report_analysis_coverage` command generates a report showing the types of checks being done (setup, hold, minimum pulse width, recovery, removal, and so on). Also, for each type of check, it shows the number and percentage of checks that meet the constraint, violate the constraint, and are not tested for the constraint. For example, a small test design might produce a report like this:

```
Type of Check    Total    Met        Violated     Untested
---------------------------------------------------------
setup             5       0 (  0%)   3 ( 60%)     2 ( 40%)
hold              5       3 ( 60%)   0 ( 0%)      2 ( 40%)
---------------------------------------------------------
All Checks       10       3 ( 30%)   3 ( 30%)     4 ( 40%)
```

The report shows the numbers of constraints not tested due to false paths, disabled timing, case analysis, and mode analysis. It can also reveal constraints not tested due to setup mistakes such as unconstrained inputs. You can get more coverage information by using the `-status_details` option.

report_delay_calculation

The `report_delay_calculation` command generates a detailed report on how PrimeTime calculates the delay for a specified cell or net timing arc. This information is useful for debugging a timing problem or verifying the timing data in a technology library. The reported delay calculation takes into account the operating conditions and wire load models.

Here is an example of a delay calculation report:

```
pt_shell> report_delay_calculation -from BLK1/A -to BLK1/Z

****************************************
Report : delay_calculation
Design : led
Version: 2005.06
Date   : Thu Jun 2 18:14:42 2005
****************************************

From :                                BLK1/A
To :                                  BLK1/Z

arc type :                            cell
arc sense :                           inverting
Input net transition times:           Dt_rise = 0, Dt_fall = 0

Rise Delay computation:
rise_intrinsic                        0 +
rise_slope * Dt_fall                  0 * 0 +
rise_resistance * (pin_cap + wire_cap) / driver_count
0.2 * (0 + 0.53) / 1
```

```
rise_transition_delay               0.106
-----------------------------------------
Total                               0.106
Fall Delay computation:
fall_intrinsic                      0 +
fall_slope * Dt_rise                0 * 0 +
fall_resistance * (pin_cap + wire_cap) / driver_count
0.15 * (0 + 0.53) / 1
fall_transition_delay               0.0795
-----------------------------------------
Total                               0.0795
```

Note:

The delay calculation feature is sometimes disabled in the ASIC vendor's technology library in order to protect the vendor's confidential delay calculation information.

Timing Analysis Updates

After you make a change that affects the timing of the design (for example, changing the clock period), PrimeTime does not necessarily update the analysis information immediately. It waits until you use a command that requires up-to-date timing information, such as `report_timing`. You can explicitly request a timing update at any time with the `update_timing` command.

By default, PrimeTime minimizes the time it spends on a timing update by analyzing only the parts of the design that are affected by changes since the previous timing update. You can override this default behavior and update the entire delay by using the `-full` option of the `update_timing` command.

Fixing Timing Violations

When PrimeTime reports a timing violation, you should examine the violation report to determine whether it is a true violation, and not a condition such as a false path or an incorrectly specified constraint. The various "report" commands can help you determine the precise cause of each violation.

PrimeTime lets you temporarily change the design in certain ways, without modifying the original netlist, so you can easily test the timing effects of those changes. To make these changes, use the `insert_buffer`, `size_cell`, and `swap_cell` commands.

To fix a timing problem, you generally need to resynthesize part or all of the design using new timing constraints. To resynthesize a submodule in the design, you can capture the timing environment of that submodule with the `charcterize_context` command. This command creates a script that can be used in Design Compiler to specify the timing conditions for resynthesis.

Saving and Restoring Single-Core Sessions

The `save_session` and `restore_session` commands allow you to save the current state of a PrimeTime session and restore the same session later. When you need to examine the results of an earlier timing analysis, using the save and restore feature avoids repeating the costly time-consuming `update_timing` portion of the analysis. Additionally, you can save a session before it is updated so that you can also save and restore the time-consuming pre-update actions. The `restore_session` command takes you to the same point in the analysis using only a small fraction of the original runtime. This command clears out any existing design data and library data in PrimeTime memory before it restores the saved session.

Note:

PrimeTime uses the /tmp directory to store temporary files. You can use the `pt_tmp_dir` variable to change the location of this temporary storage directory. When you restore the session, the value explicitly set before using the `restore_session` command remains unchanged. If you have not set a value before restoring the session, the value saved with the `save_session` command is used.

Ensure the version of PrimeTime used to restore a session with the `restore_session` command is the same version used when saving the session with the `save_session` command. Locate the version used to save the session from the README file located in the session directory.

A saved and restored PrimeTime session retains the following types of information:

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

Here are some situations where you might find the save and restore feature useful:

- You use a script to run a PrimeTime session overnight. The script uses `save_session` after the final `update_timing` command. At any time later on, you and other users can restore the session and examine the results using `gui_start`, `report_delay_calculation`, `report_timing`, and so on.

- You save the current state of the analysis as a checkpoint, which you can restore later in case of an error or unexpected change in the analysis environment.
- You save the current session and then restore it multiple times to apply different chip operating modes. This is an alternative to saving and applying the timing data in SDF format.

The platforms used for saving and restoring the PrimeTime session can be different (Solaris, HP, Linux, 32-bit, 64-bit). However, the version of PrimeTime must be exactly the same, including any service packs.

You can use the `save_session` command to save sessions before or after using the `update_timing` command. If the timing of the design is already updated, the saved session includes the timing. If only incremental timing updates are pending, the incremental update is performed and the resulting timing is saved with the session. If a full timing update is pending or the timing has not yet been updated in the analysis, the `save_session` command does not save any timing information. Saving such sessions without timing allows designs and parasitics to be loaded once, saved, and reused, improving throughput for multiple design runs that share a netlist and parasitics. In all cases, the `save_session` command requires a linked design and causes an implicit link if the design has not yet been linked.

A restored session gives results that are functionally equivalent to the saved session. However, the ordering of items in reports can different depending on the command, such as the `list_libs`, `get_clocks`, and `get_pins` commands.

If a design has multiple paths that have exactly the same slack, the ordering of those paths in timing reports is consistent within a working session, but could change when a session is saved and restored. As a result, it is possible for the reported “worst-slack” path to change. In that case, however, the slack is the same as that of the previously reported worst-slack path.

Note that the following types of information are not saved or restored:

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

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

You must specify a directory name when you save or restore a session, ensuring PrimeTime has read and write permissions. For example,

```
pt_shell> save_session session1A
...
pt_shell> restore_session session1A
```

The session is saved in a new subdirectory created in the current directory. If the directory that you specify already exists, PrimeTime overwrites everything in this directory before saving the session. For more information, see the man pages for the `save_session` and `restore_session` commands.

3

Distributed and Threaded Multicore Analysis

This chapter concisely explains how to use distributed and threaded multicore analysis in PrimeTime. It includes the following sections:

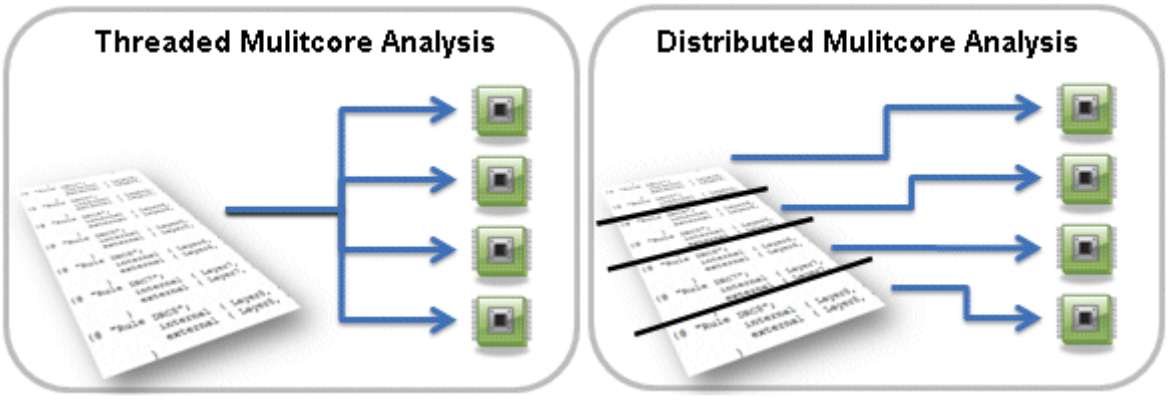
Overview of Multicore Analysis

PrimeTime multicore analysis targets the threaded and distributed multicore hardware resources generally available today. The multicore technology in PrimeTime and PrimeTime SI encompasses the threaded multicore analysis and distributed multicore analysis.

Threaded multicore analysis uses a shared memory approach and utilizes multiple cores on the same machine. Threaded multicore analysis is complementary to distributed multicore analysis and improves performance on a single server or on each of the distributed servers. Distributed multicore analysis partitions the design into several smaller pieces, each of which is assigned to a separate core for analysis.

Figure 3-1shows the two different approaches threaded multicore analysis and distributed multicore analysis use.

Figure 3-1 Threaded and Distributed Multicore Analysis Comparison



Threaded Multicore Analysis

Threaded multicore analysis enables PrimeTime and PrimeTime SI to improve performance on a shared memory server by running multiple threads on the cores available on that server. Threaded analysis is enabled by default in PrimeTime and PrimeTime SI. When using threaded multicore analysis, the tool executes many time-consuming algorithms in parallel to yield a faster run without requiring any script changes. The commands that employ parallelization include: `get_timing_paths`, `read_parasitics`, `report_analysis_coverage`, `report_constraint`, `report_timing`, `save_session`, `update_timing`, `update_noise`, and `write_sdf`.

You can control the number of cores that are used by setting the `-max_cores` option of the `set_host_options` command. You can set this option to `1` if only a single core is expected to be used.

In addition to the number of cores that are determined by the `set_host_options` command, PrimeTime gives a warning and limits the utilized cores to the physical number of cores available on the server. For example, in a dual core server, threads are launched to utilize only the two available cores.

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

You can use the `multi_core_allow_overthreading` variable to control over threading in multicore analysis. When the variable is set to its default of `true`, it is possible for simultaneously active threads to exceed the maximum limit set for the CPU cores. However, you can set this variable to `false` to guarantee that the process cores utilization does not exceed the maximum core limit. This could result in reduced multicore performance.

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

Executing Commands in Parallel

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

Note:

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

To use parallel command execution, you add either the `parallel_execute` command or the `redirect -bg` command to your scripts. The `parallel_execute` command contains the PrimeTime commands that you want to execute in parallel. It blocks the pt_shell until all of the embedded jobs are completed. You can add multiple PrimeTime commands into the `parallel_execute` loop. For example,

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

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

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

Note:

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

When using parallel command execution, the maximum number of parallel commands executed at one time is determined by the `-max_cores` option of the `set_host_options` command. If `-max_core` is set to `1` and PrimeTime runs in single-core analysis mode, all the commands specified with the `parallel_execute` command are executed serially in the order given. The `redirect -bg` command also runs the commands in the foreground, as if the `-bg` option was not specified.

Support for Path-Based Analysis

Threaded multicore analysis is implemented for path-based analysis when using the `-pba_mode path` and `-pba_mode exhaustive` options. This feature is supported for all command variations of path-specific and exhaustive recalculation. For example,

- `report_timing -pba_mode exhaustive ...`
- `report_timing -pba_mode path ...`

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

If multiple cores are unavailable, traditional single-core path-based analysis is used.

Note:
Threaded multicore analysis is not applied if you disable slew recomputation by setting the `pba_aocvm_only_mode` variable to `true`.

Distributed Multicore Analysis

Distributed multicore analysis enables you to use multiple cores to improve the performance of PrimeTime and PrimeTime SI by distributing the timing update and optimizing handling in other major flow areas. Distributed multicore analysis is explained in the following sections:

- Overview of Distributed Multicore Analysis
- Using Distributed Multicore Analysis
- Parasitics Handling in PrimeTime and PrimeTime SI
- Timing Update Using the Distributed Multicore Analysis Flow
- Distributed Multicore Analysis Reporting
- Two Distributed Multicore Analysis Syntax Examples
- Saving and Restoring Your Distributed Multicore Analysis Session
- Recommendations for Using Distributed Multicore Analysis
- Distributed Multicore Analysis User Messages
- Licensing Requirements for Distributed Multicore Analysis

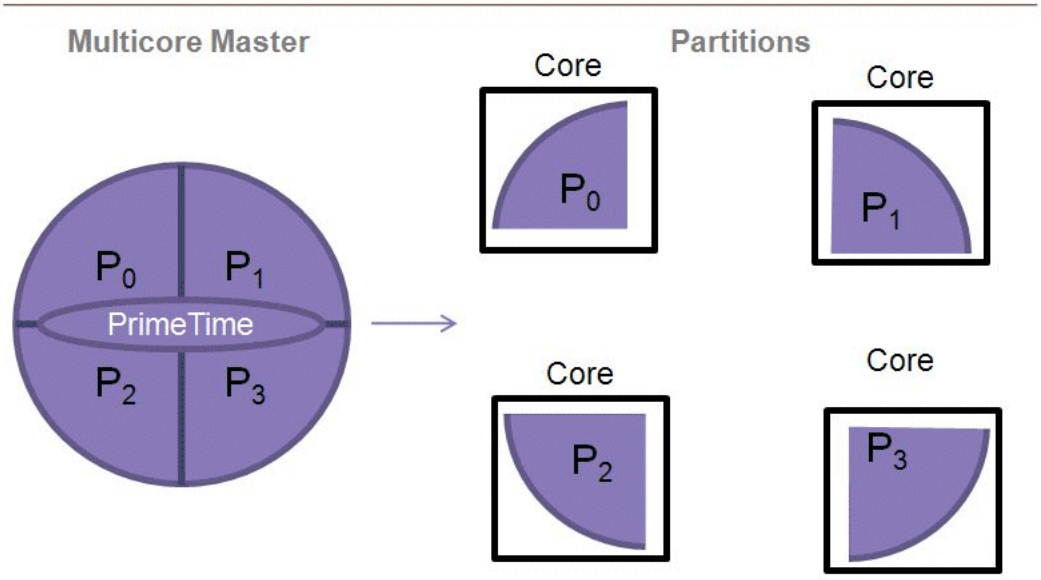
Overview of Distributed Multicore Analysis

The distributed multicore analysis capability builds upon the distributed parallelism infrastructure and other existing capabilities of PrimeTime. The mechanisms employed by PrimeTime distributed multicore analysis readily extend to other compute environment

configurations. It also supports upgrades to compute environments as newer hardware technologies become available. Distributed multicore analysis optimizes the flow turnaround time by running the timing update in parallel.

Figure 3-2 shows how the timing update phase is distributed when you execute it in the distributed multicore analysis flow. PrimeTime partitions the design and distributes it, either remotely across a network or locally within the same computer, to a number of cores. The pt_shell that you started in distributed multicore analysis flow is referred to as the multicore master. The multicore master manages all operations related to partitioning and distributing the design. After the design is distributed, the timing update proceeds normally for each partition. After each partition has finished, the partition communicates its completion status back to the multicore master.

Figure 3-2 Distributed Multicore Analysis Timing Update Distribution

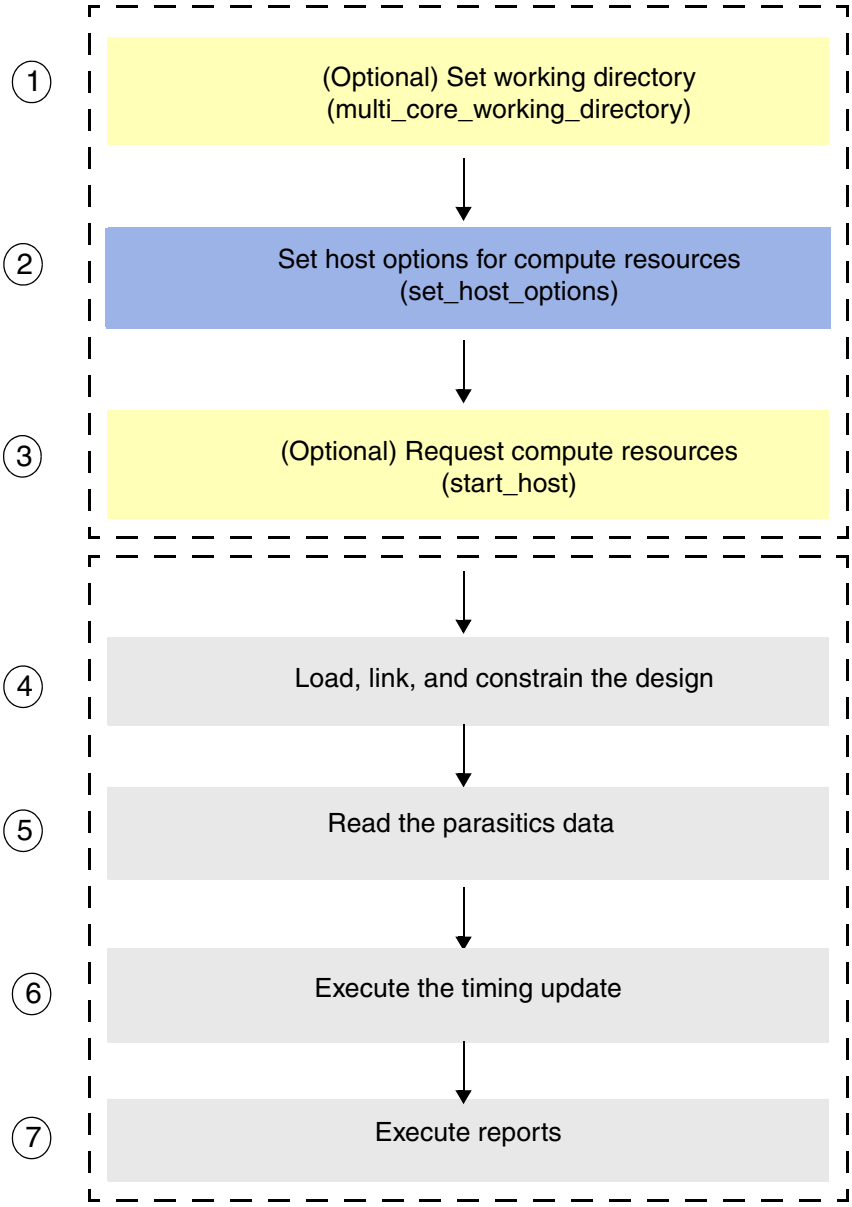


Using Distributed Multicore Analysis

To perform a distributed multicore analysis with the timing update enabled, you only need to add one command to your script. Figure 3-3 shows the distributed multicore analysis flow in PrimeTime. Step 2 illustrates the step that is specific to distributed multicore analysis while

steps 1 and 3 are optional distributed multicore analysis steps and are similar to the distributed multi-scenario analysis flow. Finally, steps 4 through 7 are common to the distributed multicore analysis and single-core analysis flow.

Figure 3-3 Distributed Multicore Analysis Usage Flow



Setting the Working Directory

Optionally, you can explicitly set the working directory where each PrimeTime process stores its output data by using the following syntax:

```
set multi_core_working_directory /directory_name
```

The `multi_core_working_directory` variable defines the root working directory for all distributed multicore analysis data, including log files. If you do not specify a working directory, the default is the directory from which the analysis was invoked.

You must set this variable to a directory that is write-accessible and network accessible by you and all PrimeTime processes. If you do not explicitly set this variable, the current working directory is used. After this directory is set in a distributed multicore analysis session and you have launched the partition processes, you cannot modify the `multi_core_working_directory` variable.

Setting Up Host Options for Compute Resources

The distributed multicore analysis flow uses the concept of a host option. A host option specifies unique processes associated with a given compute server or multiple computational slot on a managed farm of compute servers, for example, a Load Sharing Facility (LSF) or Grid Engine.

You add compute resources for each partition by using the `set_host_options` command. This command allows you to specify the compute resources you want to utilize during the timing update and reporting phases of distributed multicore analysis. This command does not start the host, but rather sets up the host options for that host.

You can explicitly specify that a 32-bit executable should be used when launching the partition processes by setting the `-32bit` option of the `set_host_options` command. If you do not specify this option, PrimeTime automatically sets the executable by default so that the partition processes match the multicore master. When the `multi_core_use_32bit_slaves` variable is set to its default of `true` and the partition processes are able to fit into the available 32-bit memory space, they are launched using a 32-bit executable regardless of whether the multicore master is on a 32-bit or 64-bit system. When you set this variable to `false`, the partition processes are launched with the same executable as the multicore master.

Note:

Additionally, you can configure the distributed environment by using the `set_distributed_parameters` command. For user configurations to take effect, you must issue the `set_distributed_parameters` command before the `start_hosts` command.

To remove all of the host options and terminate the processes that were launched based on these options, use the `remove_host_options` command. You can also remove a list of predefined host options by specifying the host option names. For example, if there were host options called host1, host2, and host3 and you wanted to remove host1 and host3, you would use the following command:

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

Note:

By default, all processes are stopped when you use the `remove_host_options` command.

For more information about these commands, see the man pages.

Requesting Compute Resources

You can use the `start_hosts` command to request compute resources, as the specifications permit, using the `set_host_options` command. The `start_hosts` command is invoked automatically by the `update_timing` command. Explicitly calling this command after the `set_host_options` command has the advantage of making compute resources available at the start of the timing update and optimizes the flow runtime. Regardless of whether the command is executed implicitly or explicitly, the amount of compute resources received might differ from the amount of resources requested. This is due to the latency involved in starting the partition processes.

You can create a detailed report of requested user-defined distributed host options specified by using the `report_host_usage` command. This command generates a report that describes all of the host options that you created in the distributed multicore analysis and distributed multi-scenario analysis (DMSA) flows.

To stop all hosts that you have started, use the `stop_hosts` command.

Note:

If you are switching from a distributed multicore analysis to a single-core analysis, stop all hosts, remove all host options, and terminate all processes by using the `remove_host_options` command with no arguments.

For more information about these commands, see the man pages. For more information about DMSA, see the *PrimeTime Distributed Multi-Scenario Analysis User Guide*.

Parasitics Handling in PrimeTime and PrimeTime SI

During the pre-update phase, you can instruct PrimeTime to read parasitics from the parasitics file using the `read_parasitics` command. PrimeTime distributed multicore analysis has optimized parasitics handling when you use a single parasitics file in the SBPF

format, allowing much faster reading and annotating of the parasitics relative to the other format types. To achieve this gain, you can create an SBPF file from other annotated parasitics formats by using the following syntax:

```
write_parastics -format SBPF file_name
```

In PrimeTime SI, the multicore master must read the parasitics file directly, storing the data in memory. This behavior occurs when you use the `-keep_capacitive_coupling` option of the `read_parasitics` command. The multicore master reads the parasitics file into memory and then dispatches the parasitic data to each of the partitions.

In PrimeTime, if a flat parasitics file is specified during the pre-update phase, the multicore master provides the partitions with the file name of the parasitics file. The partitions can access and read the parasitics file directly, improving runtime and reducing memory at the multicore master. Parasitics are loaded on an as-needed basis. For example, parasitics are loaded if an executed command requires them. In some situations in PrimeTime, the multicore master must read the parasitics file, just as it does in PrimeTime SI. This occurs if

- There are multiple parastitics files specified during the pre-update phase.
- A single command is used to specify multiple parasitics files.
- The parasitics file contains hierarchical parasitics.
- The parasitics file is from the Milkyway database.
- The format of the parasitics file is either Standard Parasitic Exchange Format, Synopsys Binary Parasitics Format (SBPF), Reduced Standard Parasitic Format (RSPF) or PARA (Milkyway).
- The initial reading of the parasitics file occurs after the pre-update phase.
- A full update is triggered after the initial update.

For more information about the `read_parasitics` and `write_parasitics` commands, see the man pages.

Timing Update Using the Distributed Multicore Analysis Flow

When the distributed multicore analysis flow is enabled, the `update_timing` command invokes partition creation and execution of the distributed timing update. When using the distributed multicore analysis flow in PrimeTime, the following factors apply:

- A limited number of commands are not supported in the distributed multicore analysis flow. PrimeTime handles the command in one of two ways:
  - When the `multi_core_skip_unsupported` variable is set to `true` (the default), if you try to execute a command that is unsupported, PrimeTime skips the command, displays a warning message (MC-017), and continues to operate in the distributed multicore analysis flow. Upon exiting the session, PrimeTime produces a summary of the commands that were skipped.
  - Skipping unsupported commands ensures optimal performance. However, there might be situations where you want to disable skipping unsupported commands. In this case, you can explicitly set the `multi_core_skip_unsupported` variable to `false`. When you execute an unsupported command, PrimeTime automatically terminates any partition processes, returns to the single-core analysis flow with a warning message (MC-007), and executes the command with the single-core analysis flow.

  For more information about unsupported commands, see the `multi_core_skip_unsupported` man page.
- The `timing_crpr_enable_adaptive_engine` variable is unsupported in the distributed multicore analysis flow. If you set either of these variables to `true` (the default is `false`), the setting of the `multi_core_skip_unsupported` variable controls their behavior in the following way:
  - If the `multi_core_skip_unsupported` variable is `true` (the default), PrimeTime automatically reverts these unsupported variables to their default values with a warning message (MC-019) and continues operating in the distributed multicore analysis flow.
  - If the `multi_core_skip_unsupported` variable is `false`, PrimeTime handles these variables just like other unsupported variables. The tool displays a warning message (MC-007) and returns to the single-core analysis flow.

  For more information about unsupported variables, see the `multi_core_skip_unsupported` man page.
- Reverting from the distributed multicore analysis to the single-core analysis flow might take some time as PrimeTime needs to perform certain aspects of its timing analysis to fully return to the single-core analysis flow.

You can alter the setting of the `multi_core_skip_unsupported` variable at any stage in the distributed multicore analysis flow. For more information about any variable, see the man page.

Distributed Multicore Analysis Reporting

When using PrimeTime and PrimeTime SI distributed multicore analysis, you can also use the multicore reporting capabilities of PrimeTime. This capability leverages the distributed multicore analysis infrastructure to generate reports more efficiently.

You can use reporting commands while in the distributed multicore analysis flow. These reports are generated by collecting the information at the partition and transferring the data to the multicore master.

When using distributed reports for optimal performance, the order in which the report paths appear might be different between a single-core analysis and a distributed multicore analysis run in PrimeTime. For example, there might be 10 paths, each with identical slack, that are the worst 10 paths in the design. The order that these paths are printed might vary slightly between the single-core analysis and distributed multicore analysis flows.

Two Distributed Multicore Analysis Syntax Examples

When using the distributed multicore analysis flow, you specify the compute resources you want to utilize during the timing update and reporting phases of distributed multicore analysis. You must launch the multicore master on an NFS mounted directory.

The first example shows the set of commands that you might use in the distributed multicore analysis flow when the maximum number of identical processes that PrimeTime should launch is four and the maximum time the command should wait for remote hosts to come online is 10,800 seconds.

```
pt_shell> set multi_core_working_directory ./pt_work_dir
pt_shell> set_host_options -name my_host_opts_3 -num_processes 4
pt_shell> start_hosts -timeout 10800
```

The second example shows the set of commands that you might use in the distributed multicore analysis flow when you are using a Load Sharing Facility (LSF) farm. The maximum number of processes that PrimeTime should launch is 8 and the minimum number of hosts the command waits for to come online is 6.

```
pt_shell> set multi_core_working_directory ./pt_work_dir
pt_shell> set_host_options -name lsf_hosts -num_processes 8 \
          -submit_command "/bin/lsf/bsub"  \
pt_shell> start_hosts -timeout 10800 -min_hosts 6
```

Saving and Restoring Your Distributed Multicore Analysis Session

In distributed multicore analysis, the save and restore usage model is just as in the PrimeTime single-core analysis flow. After launching PrimeTime and performing a full update timing using the `update_timing` command, you can use the `save_session` or `restore_session` commands at the multicore master to perform a save or restore session of your analysis. PrimeTime maintains the same number of slaves and similar configurations. A Tcl script called multicore_compute_resources.tcl is written out in the directory that you specified when using the `save_session` command. This Tcl script shows the computer resource configuration that you used for the session. Attempting to restore a distributed multicore analysis session in a single-core environment causes PrimeTime to automatically set up the distributed environment. Conversely, restoring a single-core analysis session in distributed multicore analysis flow automatically terminates the partitions.

Recommendations for Using Distributed Multicore Analysis

Designs that obtain the most impact from distributed multicore analysis have the following characteristics:

- Small pre-update and reporting phases
  Distributed multicore analysis benefits are predominantly applied to the timing update phase. Therefore, if the pre-update and reporting phases are large in comparison to the timing update flow, the benefits of distributed multicore analysis are reduced.
- Large hierarchical designs

Other guidelines that you should consider for obtaining the best results in the distributed multicore analysis flow are as follows:

- Make sure that the executing hardware has sufficient memory to avoid memory paging and swapping
- Operate over a fast network

  The distributed multicore analysis capability of PrimeTime transports data between partitions. This occurs during the partitioning, creating, and distributing of the design and also during distributed reporting. It also occurs in the PrimeTime SI iteration case. Typically, you can obtain the best performance over a Gigabit Ethernet or InfiniBand network in a managed compute farm environment or over the communications backplane of a single distributed multicore analysis compute resource.
- Use the binary parasitics flow

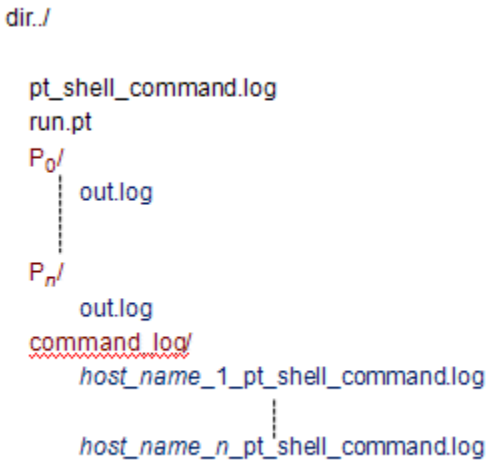
Use a single parasitics file that is in the SBPF format to optimize parasitics handling. For more information, see "Parasitics Handling in PrimeTime and PrimeTime SI" on page 3-9.

- Minimize the pre-update sections of the scripts to increase optimal performance and capacity by following established best practices. For example,
  - Avoid using the asterisk (*) wildcard for command specifications.
  - Use the `set_clock_groups` command in constraint setting.
  - Avoid creating large collections repeatedly. Instead, create what you need once and reference it using collection handles.
- Avoid executing large `report_timing` commands by using the `-slack_lesser_than` or `-slack_greater_than` options to reduce the search space of the report.

Distributed Multicore Analysis User Messages

Just as in the single-core analysis, errors, warnings, and informational messages can occur during distributed multicore analysis. In the distributed multicore analysis flow, if an error occurs, it is displayed at the multicore master terminal. Warning and informational messages that occur at the multicore master are displayed at the multicore master terminal. However, warning and informational messages that occur at the partition are added to out.log, the partition log file. This file is available inside each partition's directory structure. If you have specified the multicore working directory using the `multi_core_working_directory` variable, the P0 to Pn subdirectories are relative to this directory. By default, this variable is set to the current working directory. Figure 3-4 shows the multicore session output structure.

Figure 3-4 Logging and User Output Structure



Licensing Requirements for Distributed Multicore Analysis

After launching PrimeTime, the multicore master acquires any needed feature licenses from the license pool for itself and any partition processes. The PrimeTime multicore licensing model permits the initial license utilized by the multicore master to be used for up to four partition processes. For example, one PrimeTime license allows you to run a single job on up to four partitions. For each additional group of four partitions or part thereof, you check out an additional license.

The multicore master dynamically reallocates licenses between the multicore master and partition processes. After the multicore master has completed acquiring licenses, depending on the availability of licenses, only those partition processes that can be licensed remain active. All other partition processes are terminated. During the timing update, the licenses for the multicore master are reallocated to the partition processes and an informational message is issued. Upon exiting distributed multicore analysis, the multicore master reclaims its licenses from the partition processes and an informational message is issued.

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

During the distributed update, if PrimeTime cannot check out sufficient licenses for any feature, you receive the following error message:

```
SEC-21 (error) Failed to checkout license for feature(s)
```

PrimeTime then reverts to the single-core analysis flow. For more information about the single-core analysis flow, see “Analysis Flow in PrimeTime” on page 2-20.

4

Graphical User Interface

The PrimeTime graphical user interface (GUI) provides a way to view design data and analysis results in graphical form, including schematics, histograms, waveform plots, and data tables.

The PrimeTime GUI is described in the following sections:

- View Settings Window
- Menu Commands

Analysis Flow With the GUI

The GUI can help you visualize and understand the nature of timing problems in the design, including the type, number, magnitude, and locations of the problems. You can use the visual analysis tools after you read, link, constrain, check, and update the timing of the design.

An analysis session might consist of the following tasks:

1. Start the GUI.
2. Read, constrain, and analyze the design.
3. Generate a slack histogram to get a high-level overview of the timing quality of the design and to identify specific items that contribute to the worst slacks, such as paths, path endpoints, cells, and nets.
4. Generate a bottleneck histogram to find points in the design that contribute to the largest number of multiple timing violations. The block at the timing bottleneck might benefit from context characterization followed by optimization in Design Compiler.
5. Perform in-depth (path-level) analysis, examining specific paths in detail. A path-level analysis can include the following:
   - A bottleneck histogram generated with `-from`, `-through`, and `-to` options to restrict the scope of the paths analyzed.
   - Examining the path in the Path Inspector window, which includes a schematic, path element reports, graphical path delay profiles, and timing waveform diagrams.

Starting and Stopping a GUI Session

Before you start the GUI, make sure that the DISPLAY environment variable is set to your UNIX display name. To start a PrimeTime session that includes the GUI, at the system prompt, enter the following command:

```
pt_shell> pt_shell -gui
```

You can optionally set the DISPLAY environment variable at the same time that you invoke PrimeTime GUI. For example,

```
pt_shell> pt_shell -gui -display 192.180.50.155:0.0
```

You can also start the GUI from a pt_shell session. To start the GUI, use the following command at the pt_shell prompt:

```
pt_shell> gui_start
```

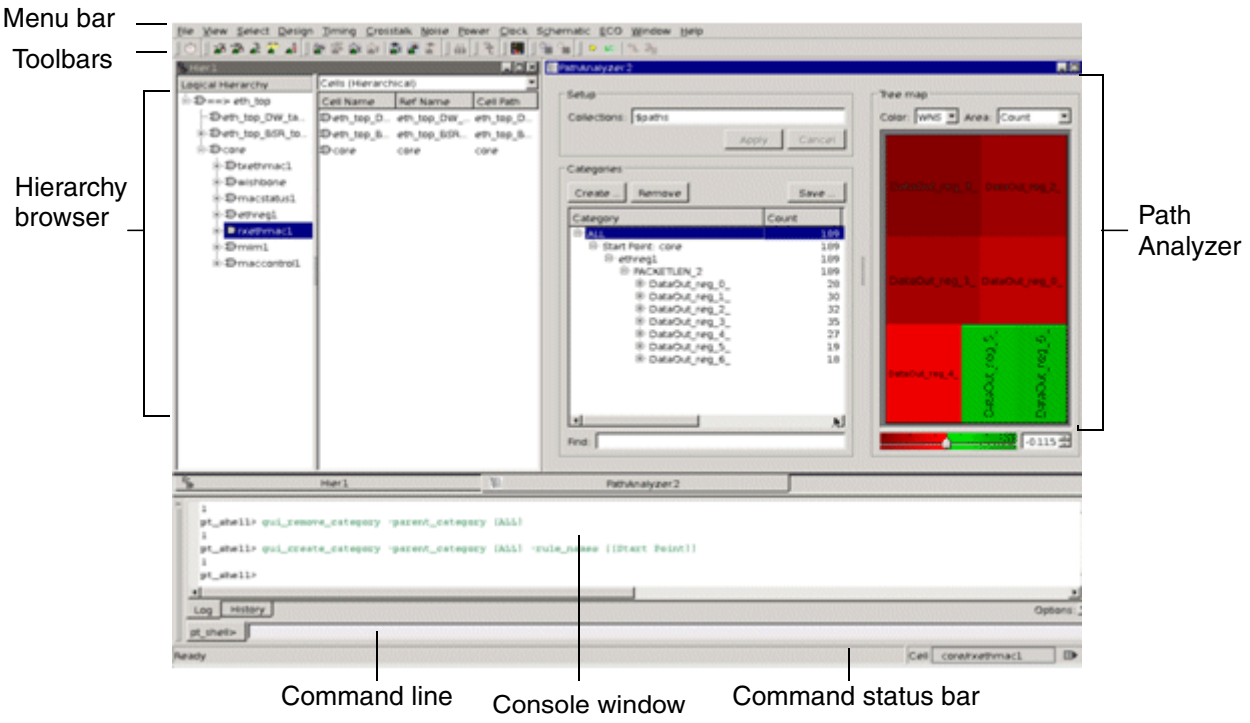
The `gui_start` command does not work when used in a script.

To stop the GUI while still keeping the original pt_shell session going in the terminal window, use the `gui_stop` command or choose File > Close GUI from the menu. To exit from PrimeTime entirely, use `exit` command or choose File > Exit from the menu.

GUI Windows

Starting the GUI opens the top-level GUI window. This is the main PrimeTime window in which you can display many types of analysis windows. You can open, close, move, resize, maximize, and minimize various types of windows within the top-level window. Figure 4-1 shows a newly opened top-level window containing the command menu, toolbars, hierarchy browser, path analyzer, and console. You can change the appearance by adding, resizing, and removing windows within the top-level window. You can also open multiple top-level windows to create different analysis views.

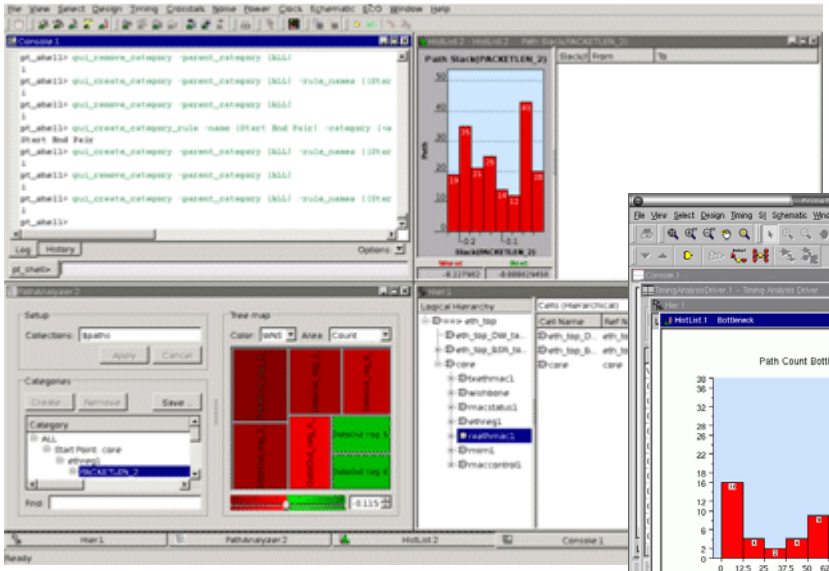
Figure 4-1 Top-Level GUI Window



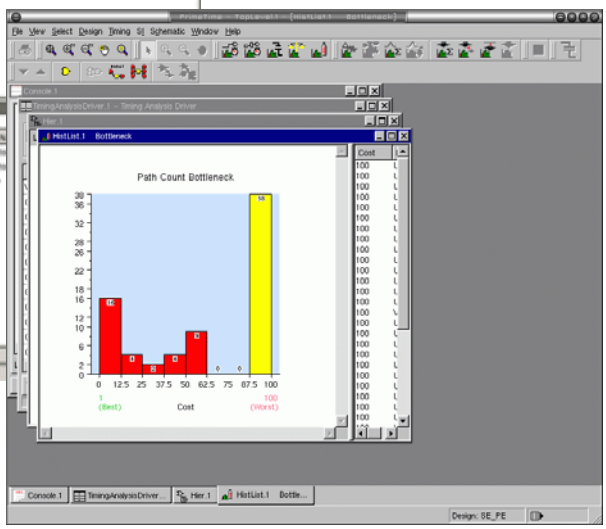
Manipulating Windows

You can open and view several top-level windows. You can use the toolbar to hide or display different windows, such as the console, histogram, or clock window. You can move, resize, and minimize windows at any time. To move a window, point to the control bar or title bar and drag it to the desired position. To lay out all windows evenly like tiles, choose Window > Tile from the menu; or to fan them out like a deck of cards, choose Window > Cascade. See Figure 4-2.

Figure 4-2 Tiled and Cascaded Windows



Window > Tile
to lay out windows like tiles



Window > Cascade
to overlap windows

To perform an operation on a window, you need to select it and make it the active window. The control bar or title bar of the currently selected window is highlighted in a distinctive color. Where windows overlap, the active window appears in front of all others.

To select a window and make it current, point to the window and click. You can also choose the menu commands Window > Next Window, Window > Previous Window, and Window > *window name*. You can choose Window > New Main Window to open a new window.

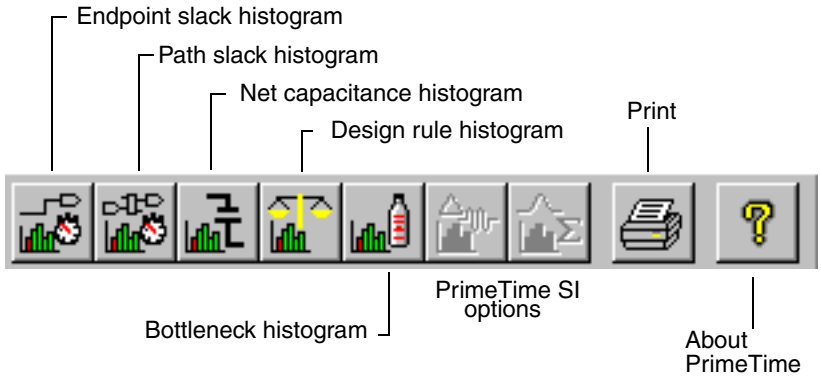
You can minimize any window, which means to temporarily turn it into a small icon and otherwise hide it from view. To do so, click the minimize [–] icon in the corner of the window (if it is an undocked window), or click the tab at the bottom corresponding to that window. To restore a minimized window, click the corresponding tab at the bottom.

To maximize a window (make it fill the whole top-level window), first undock it if it is not already undocked, and then click the maximize icon in the corner of the window. To close a window, make the window active and then choose Window > Close Window or click the Close [X] button in the corner of the window.

Toolbars

The toolbar provides quick access to often-used menu selections. Figure 4-3 shows an example of a toolbar.

Figure 4-3 Toolbar Options



Different toolbars are associated with different windows: one for schematics, one for histograms, and so on. You can show or hide toolbars by using the View > Toolbars > ... menu commands.

Options that cannot be used in the current context are disabled and displayed in light gray. The current context changes when you make a new selection. For example, after you select a path, the toolbar options that operate on paths become enabled, such as the Create Path Inspector and Create Path Schematic icons. When no paths are selected, those icons are disabled.

Selecting Objects

You can select objects for subsequent actions by clicking on them or by using the Select menu commands. For example, to display a path profile, you first need to select the path. There are different ways to make this selection. Usually the simplest method is to go to the timing path table and click on the path of interest.

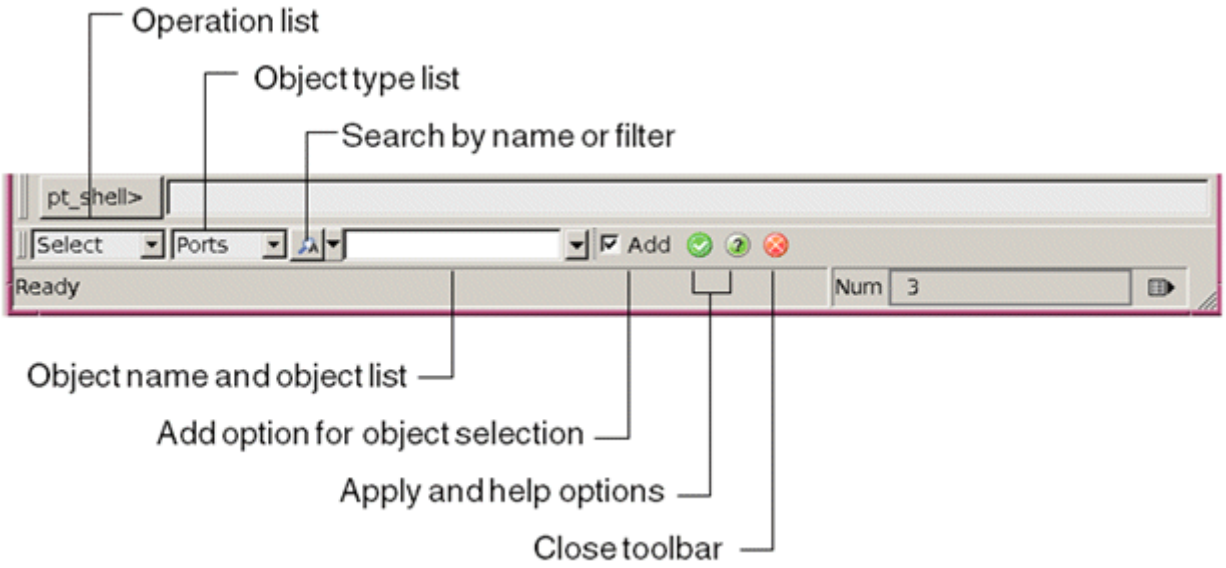
Another method is to use the Select Paths dialog box. From the menus, choose Select > Paths From/Through/To. In the Select Paths dialog box, specify the paths using the From, Through, and To fields or by setting the criteria, such as the amount of slack. Next, click OK to make the selection.

In the Clock Analyzer window, you can select or highlight a cell, net, pin, port, or clock object by name in the current design. To select or highlight an object

1. Choose Select > By Name Toolbar.

   Figure 4-4 shows the toolbar that appears at the bottom of the screen.

Figure 4-4 By Name Toolbar



2. Choose Select or Highlight from the operation list.
3. Optionally, select Cells, Nets, Pins, Ports, or Clocks to search for the specified type of objects.

   PrimeTime locates the objects that match the criteria.
4. Optionally, select one of the following search criteria:

   - Search by name

     Use to search the design for an object that matches the specified name or name pattern. Type part of a name and press Tab to enable PrimeTime to complete the name. If multiple names match the text, a list appears. Use the Up and Down arrow keys to scroll through the list. Click Enter to select an item.

     Type multiple object names separated by a blank space. If an object name contains a space, enclose the name in braces { } to treat it as a single name.

- Search by filter

  Use to search the design for all objects where the specified filter expression is located. Type part of the filter expression and press Tab to enable PrimeTime to complete the matching value. If multiple values match the text, a list appears.

Note:
Invalid name or filter search patterns are identified by a light red background.

5. Select an object name or list of objects.

   If PrimeTime locates multiple objects that match the text, the name completion list shows the first fifteen names in a list that you can scroll through.

6. To add to the current selection, click Add. Otherwise, deselect the option to replace the current selection.

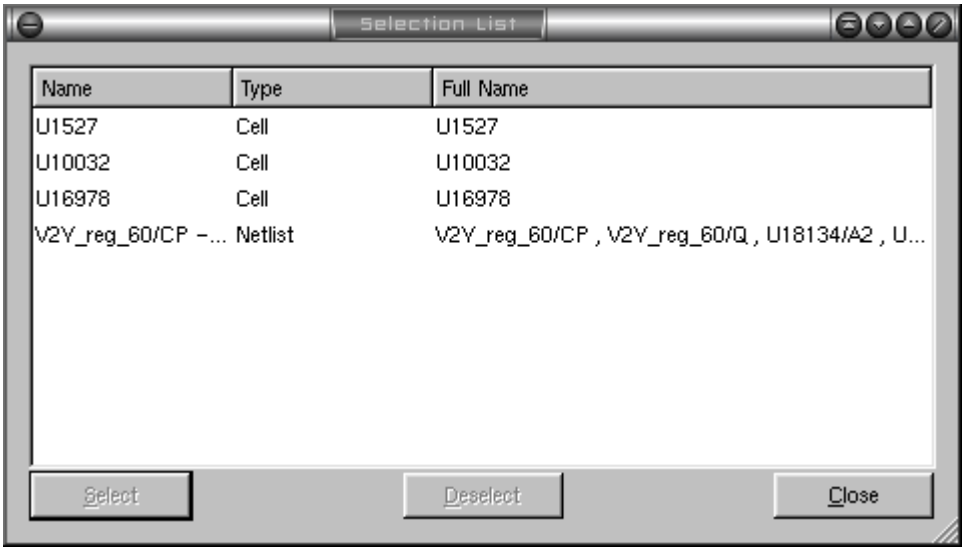
7. Click the apply icon to search for the specified criteria.

To select objects in the fanin or fanout of specified objects, choose Select > Fanin/Fanout.

To get a summary report on the current selection, choose Select > Query Selection. The report appears in the console window and in the pt_shell terminal window.

To see a detailed list of the currently selected objects, choose Select > Selection List. This opens a Selection List box like the one shown in Figure 4-5.

Figure 4-5 Selection List Box



To select multiple items, hold down the Ctrl key while you select. Otherwise, each new selection cancels the previous selection.

Selected objects are highlighted in all windows, not just the window in which you are making a selection. For example, if a cell appears in two different schematic windows and is also listed in the hierarchy browser, selecting that cell anywhere (for example, in one schematic) causes all three occurrences to be highlighted.

Window Types

Table 4-1 lists and briefly describes the types of PrimeTime windows that can be displayed within the top-level window. You can find additional information about each type of window in the remaining sections of this chapter.

Table 4-1 Types of Windows Within the Top-Level Window

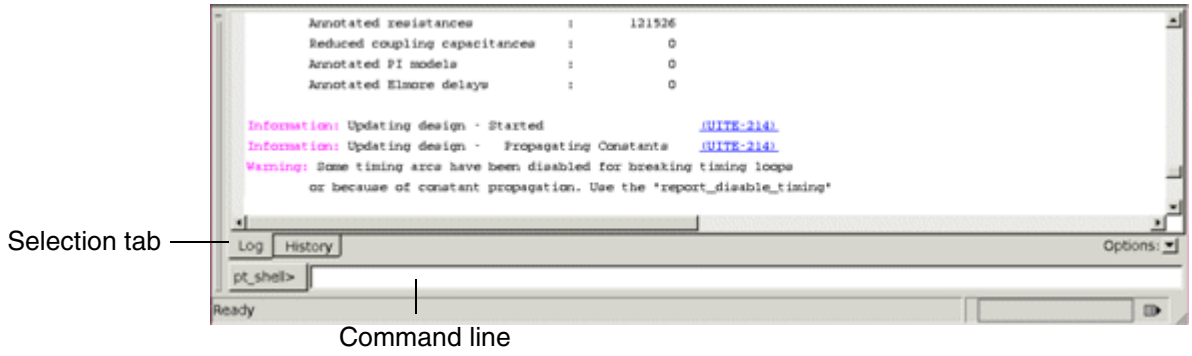
Window type	Description
Console	Lets you run pt_shell commands and view the pt_shell log, command history, or error/warning messages.
Hierarchy browser	Lets you browse through the design hierarchy and select a cell, net, port, or pin in the design.
Histogram	Displays the distribution of design or analysis data in bar graph form, such as endpoint slack, path slack, net capacitance, design rules, timing bottleneck, or crosstalk data.
Schematic	Displays a circuit schematic of a hierarchical cell or a path in the design.

By default, the Console window is displayed at the bottom. You can open or remove the default window and open any desired additional windows, such as the schematics and histograms windows. If a hierarchy browser window is not already showing, you can open one by choosing Design > New Hierarchy Browser View.

Console

You can use the Console window to execute pt_shell commands and view the text-format response from PrimeTime or the command history. A typical console window is shown in Figure 4-6. The command line is the long box at the bottom, to the right of the pt_shell prompt. You can enter commands on the command line, just as you would at the pt_shell prompt in a terminal window.

Figure 4-6 Console Window



Operating the console mirrors what is entered and displayed in the pt_shell terminal window from which you invoked the GUI. You can enter commands and view the response in either the pt_shell terminal window or in the console. The commands entered and the PrimeTime response are displayed in both places. If you do not need the console, you can close it and just use the terminal window.

The console offers some views that are not available in the terminal window. The tabs at the bottom let you select the type of text to display:

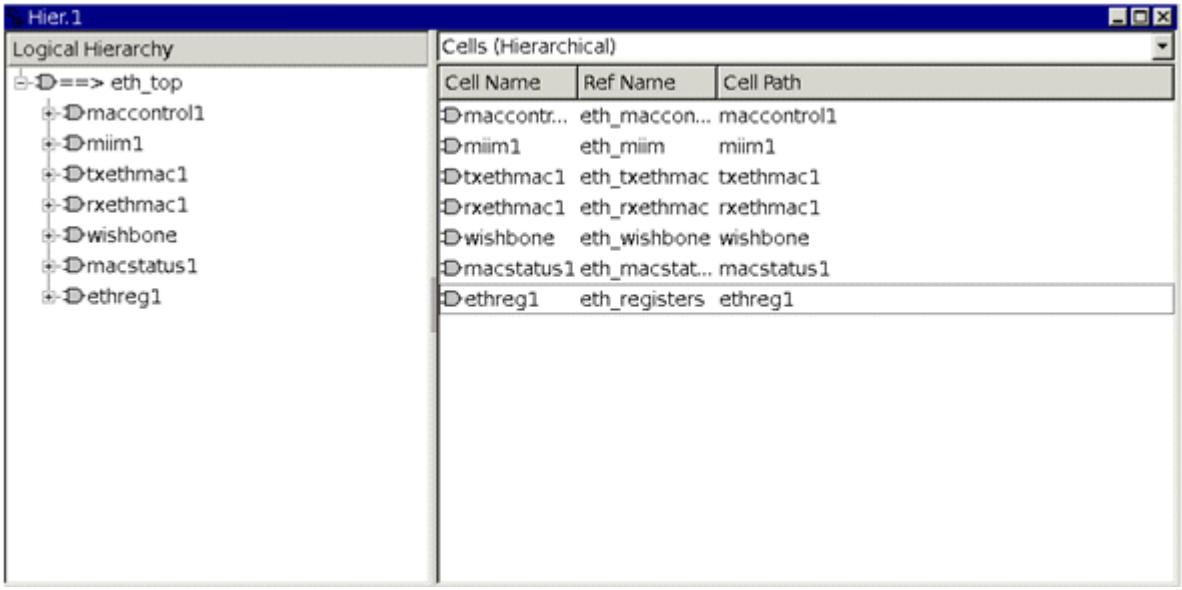
- Select the Log tab to display the pt_shell text-format response from PrimeTime. This is the same as what you see in a pt_shell terminal window.
- Select the History tab to view a history of recently executed commands. You can select commands to execute again or copy and paste the commands to create a script.

Hierarchy Browser

A Hierarchy Browser window lets you traverse the hierarchy of the design and select parts of the design for subsequent analysis. If a hierarchy browser window is not already present, you can open one by choosing Design > New Hierarchy Browser View.

Figure 4-7 shows a typical hierarchy browser window.

Figure 4-7 Hierarchy Browser Window



The pane on the left lets you browse the cell hierarchy. Click the [+] icon to view the lower-level cells beneath a cell, or [–] to collapse the view of lower-level cells.

When you select a cell in the left pane, the right pane shows the contents of that cell. Click the down arrow to select the types of objects to be listed: hierarchical cells, all cells, pins/ports, pins of child cells, nets, or clocks. When an object you select in the hierarchy browser is also displayed in a Schematic window, that object is also highlighted in the schematic.

Histogram

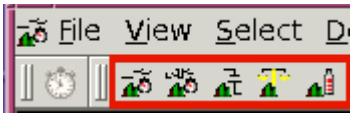
A histogram shows the distribution of design or analysis values in bar-graph form. It can show various types of information:

- Timing slack for all endpoints
- Timing slack for specified paths
- Total net capacitance for all nets
- Design rule check values on all pins (maximum or minimum capacitance, fanout, or transition time)
- Bottleneck cost for all cells

- Crosstalk analysis results (for PrimeTime SI users)
- Any attribute-specified data that you can extract from the design database with Tcl commands

You can open a histogram window by choosing Timing > Histogram > *Histogram Type* or by clicking the corresponding icon in the histogram toolbar. Figure 4-8 shows the toolbar icons that correspond with the Timing > Histogram > *Histogram Type* menu options.

Figure 4-8 Histogram Options From the Toolbar

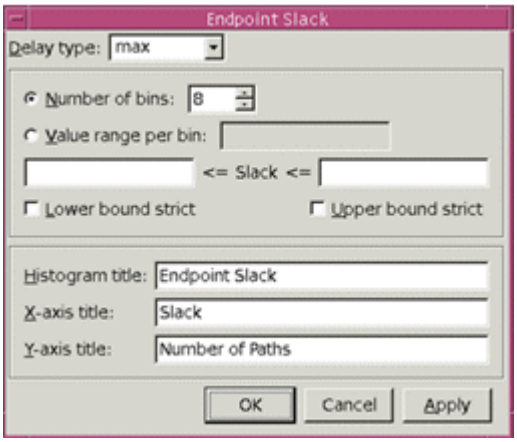


Upon selecting the histogram type, a dialog box asks you for the histogram display parameters. After you specify the parameters, click OK to generate the histogram. You can resize a Histogram window or adjust the border between the histogram plot on the left and the data fields on the right. Point to the border and drag.

Endpoint Slack Histogram

An endpoint slack histogram shows the distribution of worst slack values for all timing endpoints in the design. To generate the histogram, choose Timing > Histogram > Endpoint Slack from the menu, or click the equivalent icon in the histogram toolbar. This opens the Endpoint Slack dialog box as shown in Figure 4-9.

Figure 4-9 Endpoint Slack Dialog Box



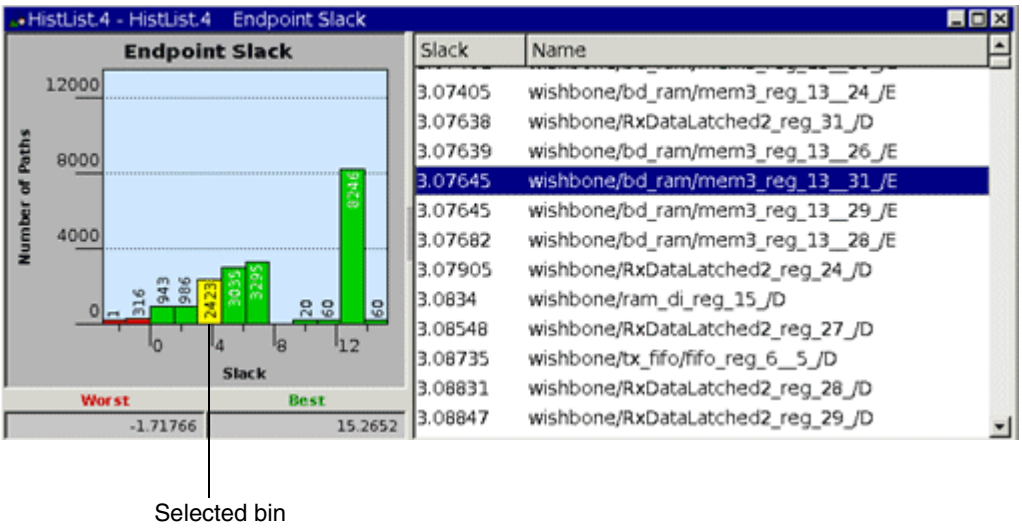
The dialog box offers several options: the delay type, max (setup) or min (hold); the number of histogram bins, the range of slack values plotted in the histogram, the vertical scale, and the text labels displayed in the histogram. The default horizontal range is from the minimum slack value to the maximum slack value found in the design.

When you click OK, PrimeTime uniformly divides the specified range, from minimum to maximum, into the specified number of bins. Then it plots the number of slack values falling into each bin, as shown in Figure 4-10.

Each bar indicates the number of slack values falling into a bin. Red bars indicate negative slacks, which are timing violations. Green bars indicate positive slacks.

You can select a bin by clicking it. The selected bin is highlighted in yellow. The pane on the right shows the path endpoints for the selected bin, together with their corresponding worst-case slack values.

Figure 4-10 Endpoint Slack Histogram



You can select one or more entries in the list to obtain more information about those entries. For example, in Figure 4-10, the second-worst-slack bin has been selected. The four endpoints in that bin are listed on the right. You can select entries in that list for subsequent operations.

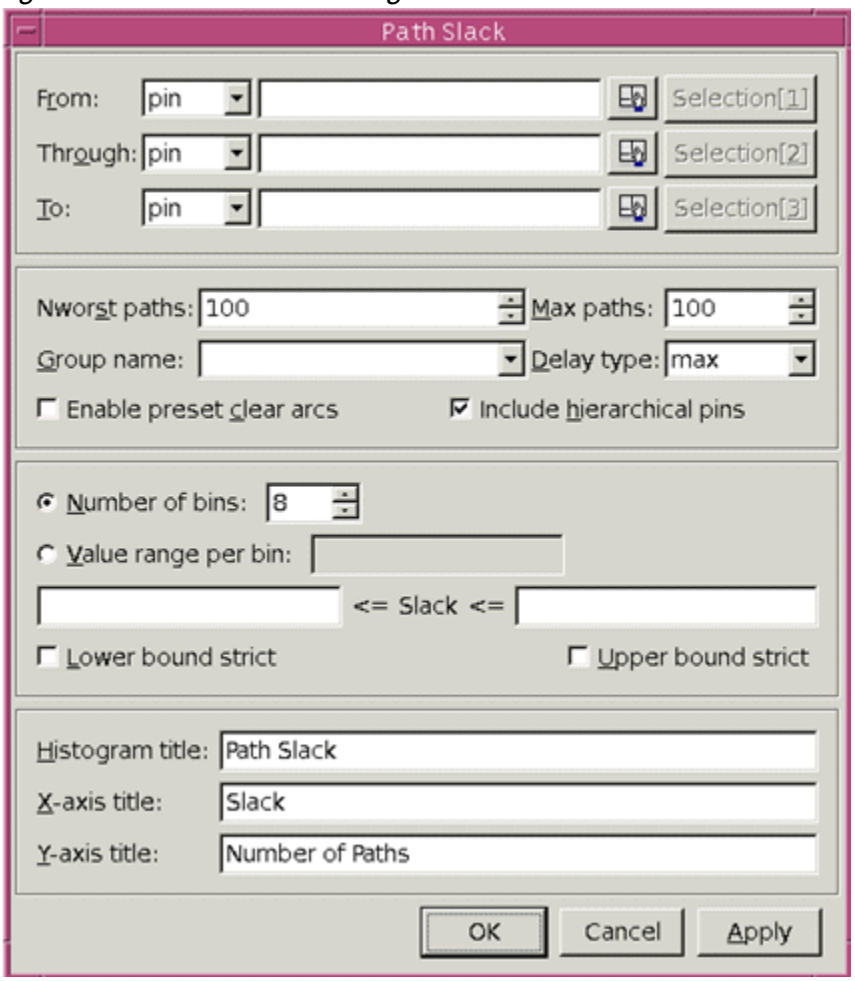
Path Slack Histogram

Like an endpoint slack histogram, a path slack histogram shows the distribution of slack values found in the design. However, instead of listing only the worst slack for each endpoint in the design, the plot shows the slack values for all paths that meet the criteria you specify. The plot can include the slack values of multiple paths that share a common endpoint.

To generate a path slack histogram, choose Timing > Histogram > Path Slack or click the equivalent icon in the histogram toolbar. This opens the Path Slack dialog box.

The upper section of the dialog box lets you specify the scope of the analysis by entering a combination of one or more from, through, or to points in the design, as shown in Figure 4-11. All paths that begin, pass through, or end on the respective points are analyzed and plotted in the histogram. Paths not within the specified scope are ignored for the analysis.

Figure 4-11 Path Slack Dialog Box

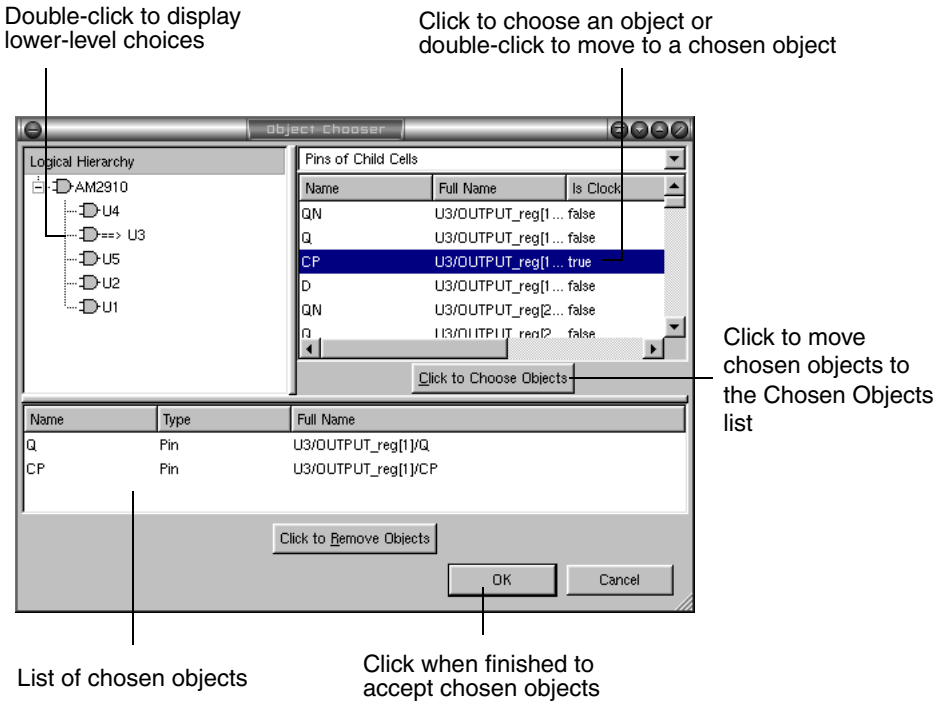


For example, if you fill in the From field with a single pin name (and leave the Through and To fields blank), PrimeTime finds all the paths that originate at the specified pin and plots the resulting slack values in the histogram.

You can fill in the From, Through, and To fields with the currently selected pins, ports, nets, or clocks by clicking Selection, which is next to the fields.

If you want to browse the design hierarchy in order to specify the scope of the design to analyze, click the small browser icon next to the From, To, or Through field. This opens the Object Chooser window (Figure 4-12), which can help you find and select the cells to analyze.

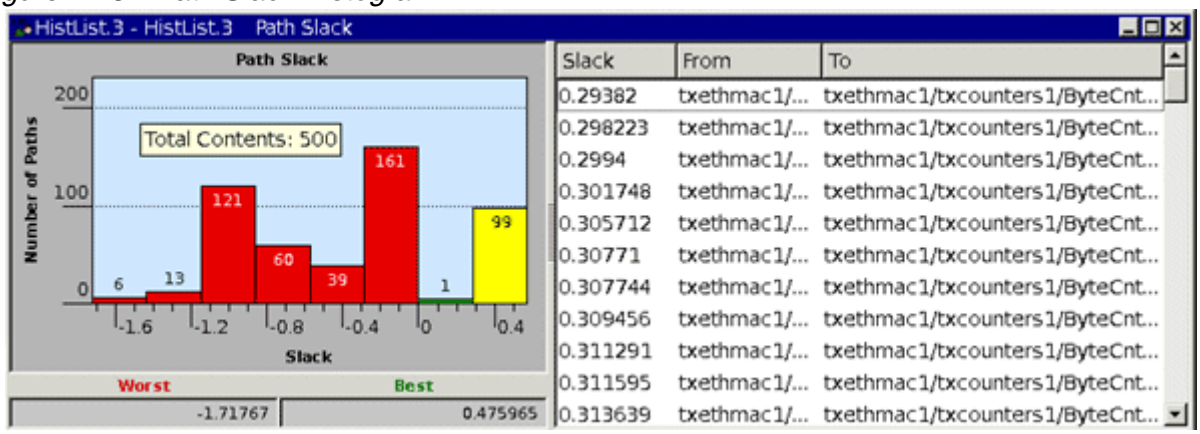
Figure 4-12 Object Chooser Window



You can specify the number of worst paths per endpoint to be reported (Nworst paths field) and the total maximum number of path slacks to be reported (Max paths field), and thereby restrict the number of data points included in the histogram plot.

Figure 4-13 shows an example of a path slack histogram.

Figure 4-13 Path Slack Histogram



The histogram plot shows the distribution of slack values for the paths included in the scope of the report. You can select a bin by clicking it, which highlights the bar in yellow. The paths and corresponding slack values for that bin are listed in the box on the right.

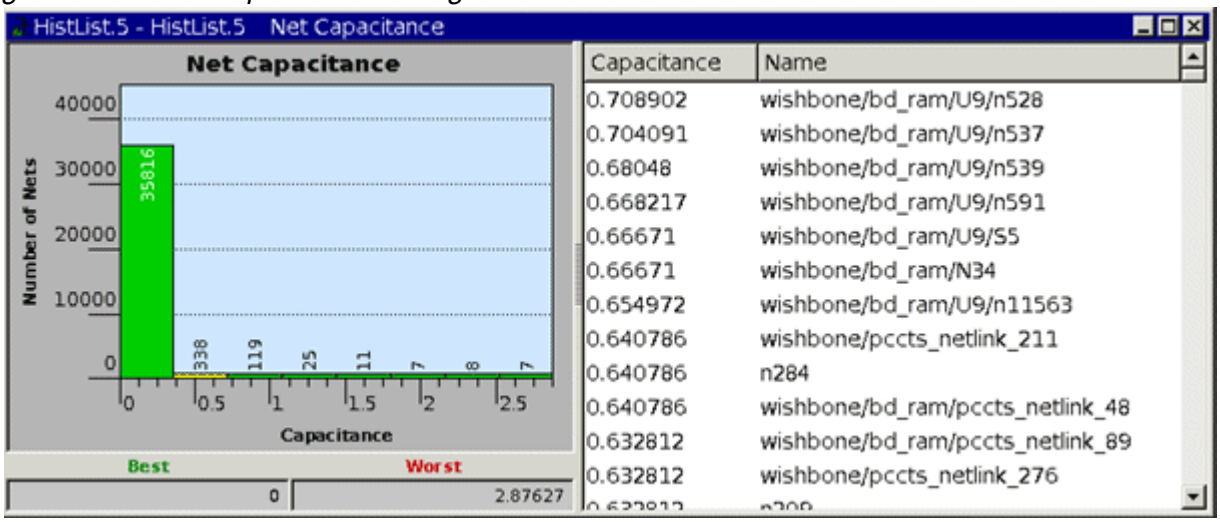
The path slack histogram options such as From, Through, To, and Max Paths are the same as the corresponding options of the `report_timing` command. For more information about using these options, see the man page for the `report_timing` command.

Net Capacitance Histogram

A net capacitance histogram shows the distribution of net capacitance values for all nets in the design. To generate the histogram, choose Timing > Histogram > Net Capacitance or click the equivalent icon in the histogram toolbar. In the Net Capacitance dialog box, specify the number of histogram bins and the range of capacitance values to plot. The default range is the full range of net capacitance values found in the design. Next, click OK.

Figure 4-14 shows an example of a net capacitance histogram. You can click any histogram bar to select its bin and view a list of the nets in that bin, together with their corresponding capacitance values. The capacitance units (such as pF or nF) are determined by the technology library.

Figure 4-14 Net Capacitance Histogram



To display a bar graph showing the relative amounts of pin and wire capacitance for a net, first select the net (for example, select a net in the bin list of the net capacitance histogram). Next, choose Timing > Net Capacitance Profile. The Net Capacitance Profiler dialog box appears showing the information about the selected net.

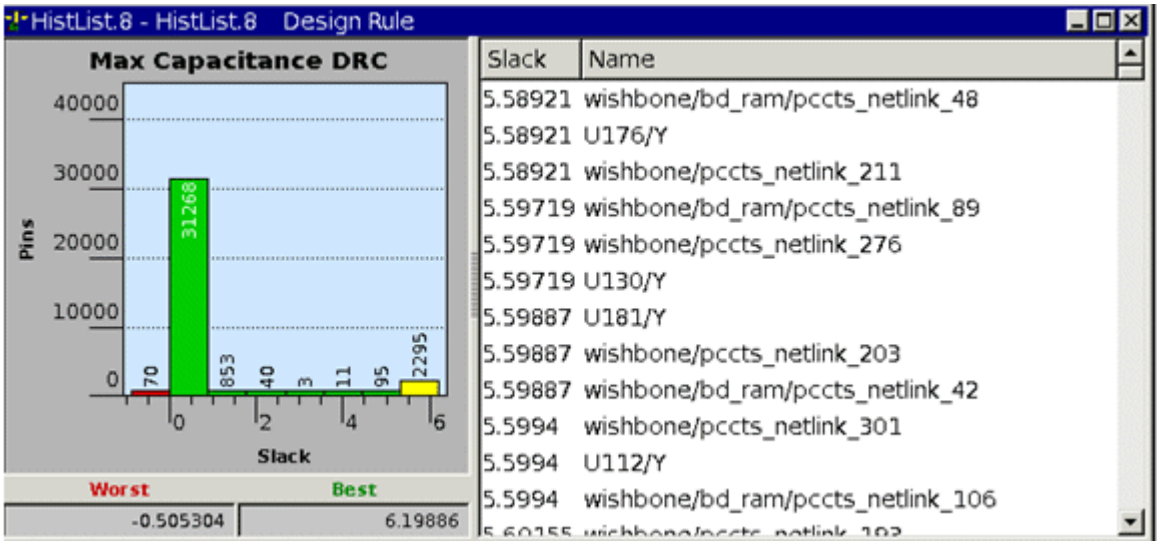
Design Rule Check Histogram

PrimeTime checks nets, ports, and pins for violations of design rules specified in the technology library or by design rule commands, such as the `set_max_capacitance` command. For more information about design rule checking, see “Design Rules” on page 8-16.

To generate a histogram plot of design rule slack values, choose Timing > Histogram > Design Rule Histogram, or click the equivalent icon in the histogram toolbar. From the DRC dialog box, select the type of design rule (minimum or maximum capacitance, minimum or maximum transition time, or minimum or maximum fanout). Click OK to generate the plot. Figure 4-15 shows an example of the design rule histogram.

Each design rule histogram plot shows the slack values for the design rule check. For example, a design rule histogram plot for the maximum capacitance rule shows the amount of capacitance slack of each net (the maximum capacitance limit of the design rule minus the actual capacitance of the net). Bins that meet the design rule are shown in green, while those that violate the design rule are shown in red. The selected bin is highlighted in yellow.

Figure 4-15 Design Rule Histogram (Maximum Capacitance)



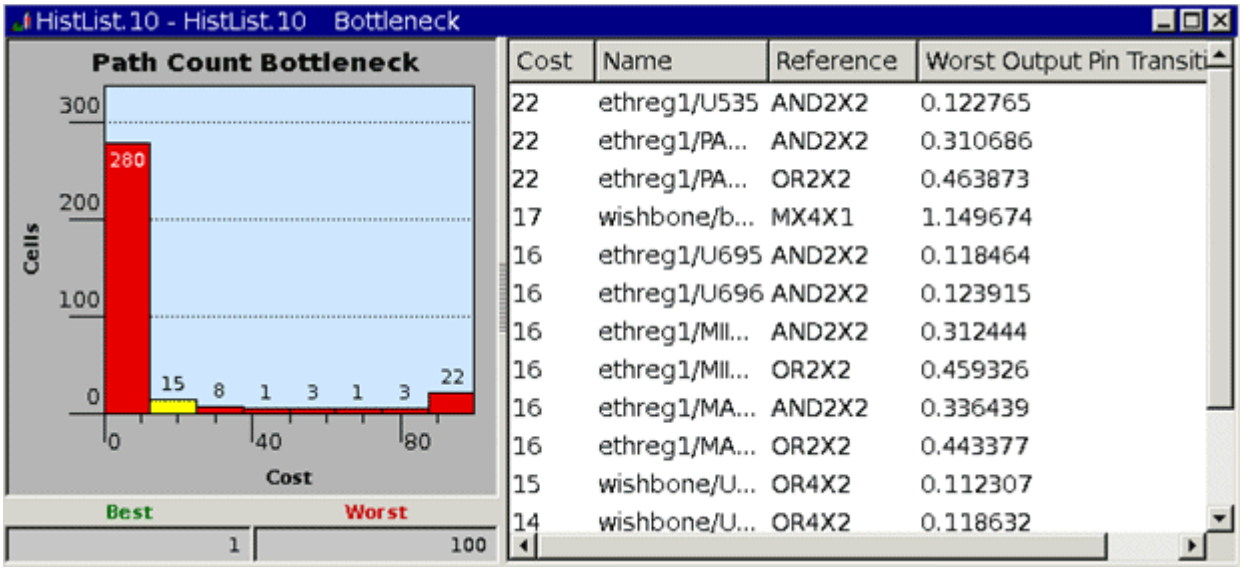
Timing Bottleneck Histogram

A timing bottleneck is a point in the design that contributes to multiple timing violations. A timing bottleneck histogram can help you find the cells the design that are causing the greatest number of timing violations. For more information about bottleneck analysis, see “Bottleneck Report” on page 13-44.

To generate a timing bottleneck histogram plot, choose Timing > Histogram > Timing Bottleneck or click the equivalent icon in the histogram toolbar. From the Timing Bottleneck dialog box, select the bottleneck cost type: Path Count, Path Cost, or Fanout Endpoint Cost. You can optionally restrict the scope of the analysis by setting the From, Through, or To fields; the Object Chooser and Selection options can help you fill in these

fields. You can also specify the bottleneck options, such as Nworst paths. For more information about these options, see the `report_bottleneck` man page. Click OK to generate the plot. Figure 4-16 shows an example of the bottleneck histogram.

Figure 4-16 Bottleneck Histogram



To see a list of cells contained in a bin, click the histogram bar. The selected bin is highlighted in yellow. The list on the right shows the names of the cells in the bin and the corresponding cost values.

PrimeTime SI Histograms

If you are a licensed user of PrimeTime SI, and if crosstalk analysis is currently enabled, you can display histograms of delta delay, bump voltages, and noise analysis results. For more information, see the *PrimeTime SI User Guide*.

Schematics

There are three types of schematics in PrimeTime: hierarchical, path, and clock. A hierarchical schematic shows the contents of a hierarchical cell and lets you traverse the hierarchy of the design. A path schematic shows the cells and nets in a particular path. A clock schematic shows the clocks in a design and lets you traverse the clock hierarchy. A Schematic window appears in its own separate top-level window, and it contains its own toolbar and menus.

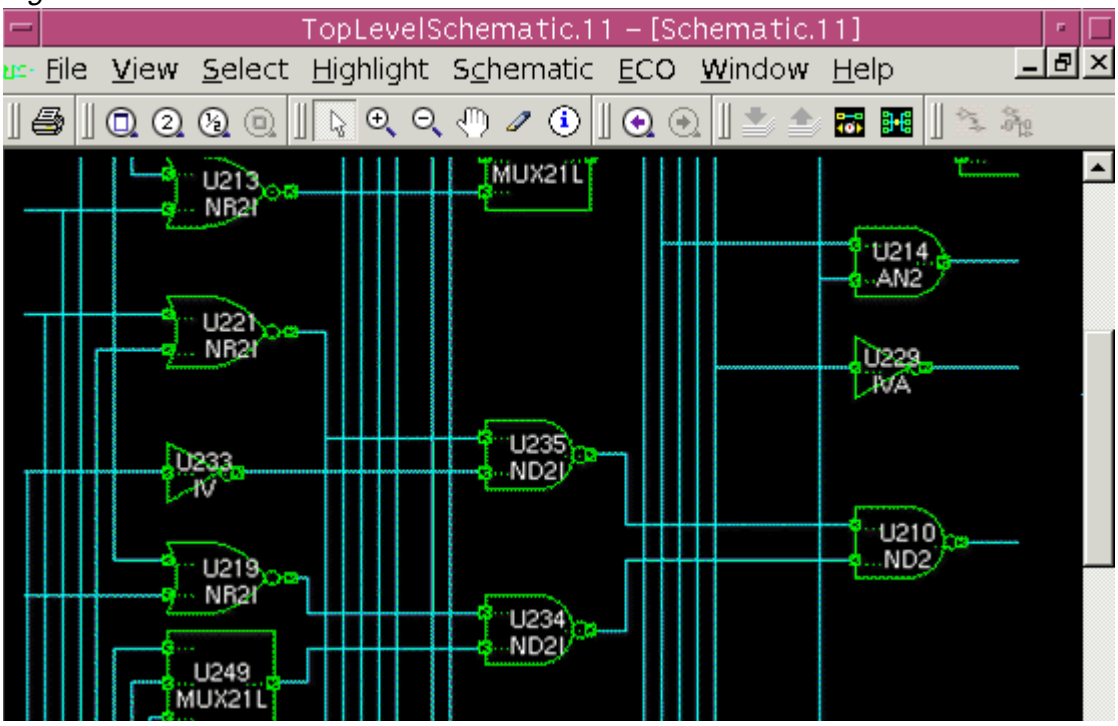
Hierarchical Schematic Window

To open a Hierarchical Schematic window, first select the hierarchical cell that you want to display, using either the hierarchy browser or the Select > By Name command; or click on a hierarchical cell in an existing schematic window. After you select the cell, choose Schematic > New Design Schematic View to open the schematic. Figure 4-17 shows an example of a hierarchical schematic. After you have opened the Schematic window, you can modify the viewing range and scale by scrolling and resizing the window and by choosing the zoom and pan commands in the View menu. Equivalent command icons are available in the View Zoom/Pan and View Tools toolbars.

You can select objects in a schematic by clicking on them. Selected objects are highlighted in white. If an object you select is listed in the Hierarchy Browser window, it is highlighted there as well.

To highlight specific objects in a schematic, use the commands in the View > Mouse Tools > Highlight Tool or choose an option from the Highlight menu. For example, to highlight the critical path (the default path reported by the `report_timing` command), choose Highlight > Critical Path.

Figure 4-17 Hierarchical Schematic Window



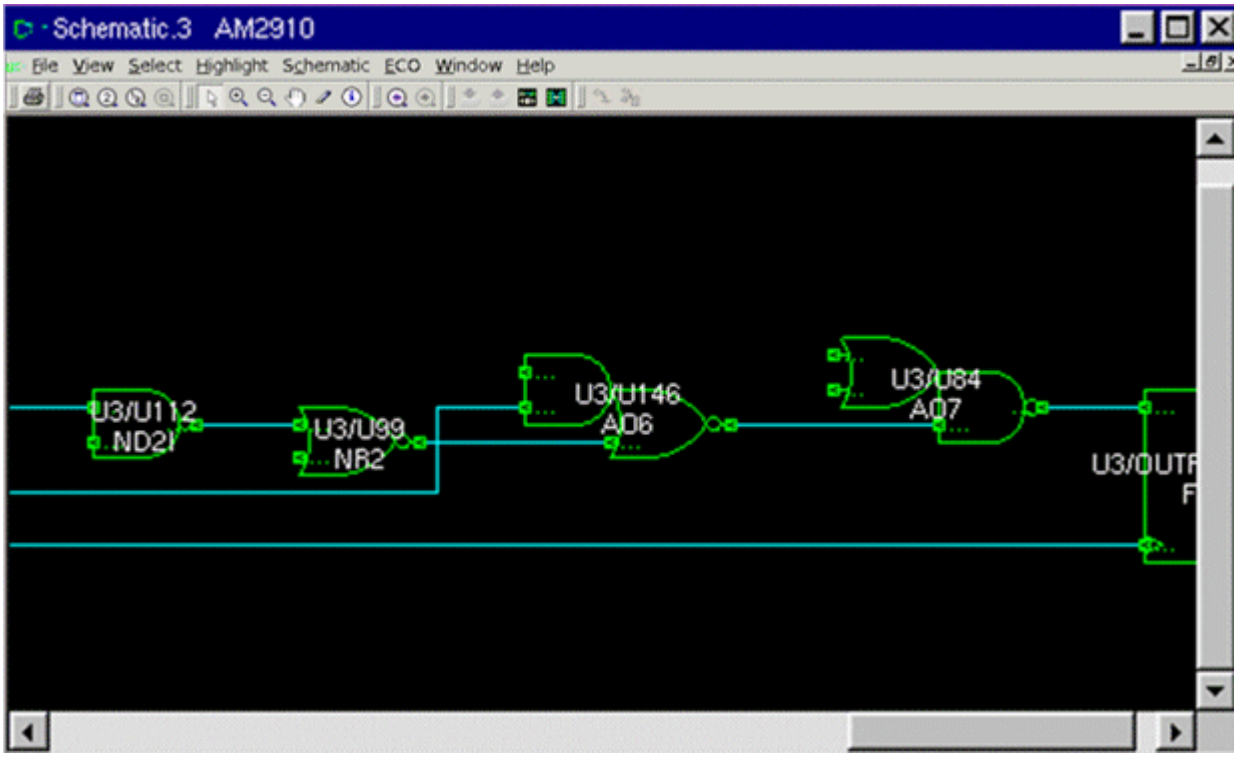
When you rest the mouse pointer on a an object, such as a cell, pin, or net, information about the object appears in an InfoTip. An InfoTip is a small, temporary box that displays information about the design at the mouse pointer location.

You can traverse the hierarchy of the design by using the Schematic > Move Down and Schematic > Move Up commands, or the equivalent icons in the Schematics toolbar. To move down the hierarchy, select the hierarchical cell of interest in the schematic, then choose Schematic > Move Down or click the down-arrow toolbar icon. This displays the lower-level schematic for the cell. To move back up to the parent schematic, choose Schematic > Move Up or click the up-arrow toolbar icon.

Path Schematic Window

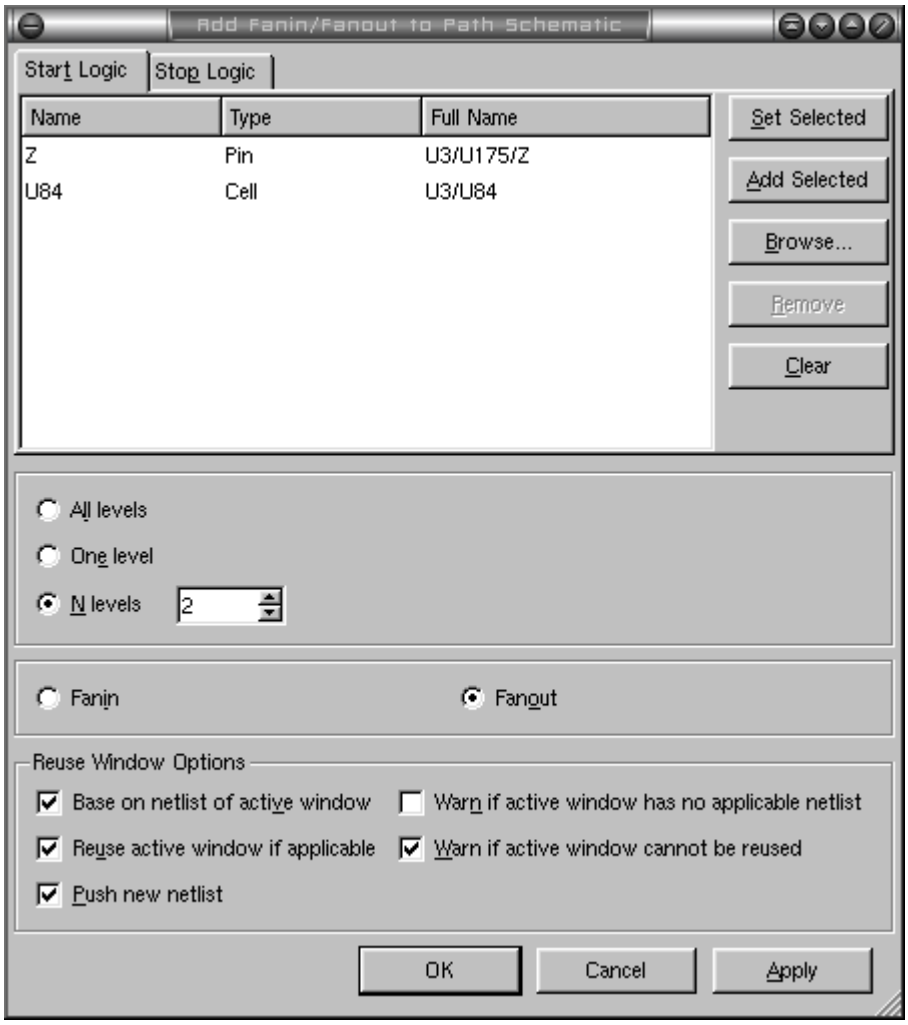
To open a Path Schematic window, first select the timing path that you want to display, using either Select > Paths From/To/Through or by selecting a path listed in the timing path table or Path Slack Histogram window. After you select the path, choose Schematic > New Path Schematic View to open the schematic. Figure 4-18 shows an example of a path schematic.

Figure 4-18 Path Schematic Window



You can increase the scope of the design that is displayed by adding the logic fanin to, or login fanout from, specified objects in the schematic. To do so, first select the object or objects of interest in the schematic. Next, choose Schematic > Add Fanin/Fanout. This opens a dialog box like the one shown in Figure 4-19.

Figure 4-19 Add Fanin/Fanout to Path Schematic



Click the Start Logic or Stop Logic tab and toggle on the Fanin or Fanout option. Click the Set Selected option, which adds the selected objects to the list box. You can optionally select more objects and use the Add Selected option to add them to the list, or click Browse to browse for objects to add to the list. Set the other options, such as the number of logic levels to be added. Click OK to update the schematic with the additional fanin or fanout logic.

You can similarly add more paths to the path schematic by choosing Schematic > Add Paths From/Through/to. This opens the Add Paths From/Through/To to Path Schematic dialog box. Specify the paths you would like to add and then click OK to update the schematic with additional paths.

For more information about the path schematic, see “Path Analyzer Window” on page 4-31.

Schematic Annotation

By default, schematic annotation on pins and ports uses the `case_value` attribute. The schematic annotation feature allows you to annotate pin attributes onto the path schematic. PrimeTime provides the following pin annotation attribute group name that you can update with other valid attributes. For example, use both the name and direction as annotation text for a pin by including the following attribute:

```
-gui_update_attrgroup -class Pin -name SchemAnnotAttr -attr_list {name direction}
```

The attributes supported are netlist object attribute within the design context. Timing Path related context is unsupported. If multiple valid values are specified in the attribute list, ensure each annotation string is delimited by a comma.

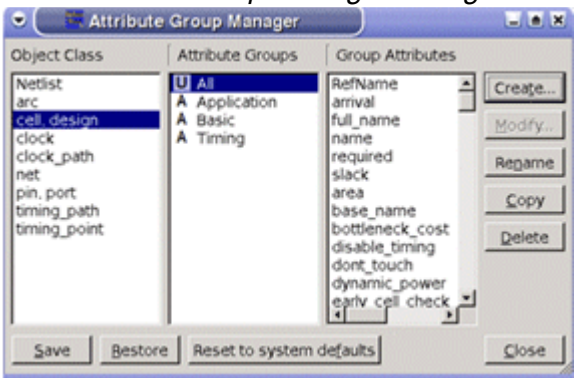
Another way to update the schematic annotation with attributes is to do the following steps in the GUI.

1. Select a pin object.
2. Choosing View > Property.

   The Properties dialog box appears.
3. From the "Attribute group" drop down, select the SchemAnnotAttr option.
4. Click the Activate Attribute Group Manager icon () that appears next to the Attribute group list.

The Attribute Group Manager dialog box appears as shown in Figure 4-20.

Figure 4-20 Attribute Group Manager Dialog Box



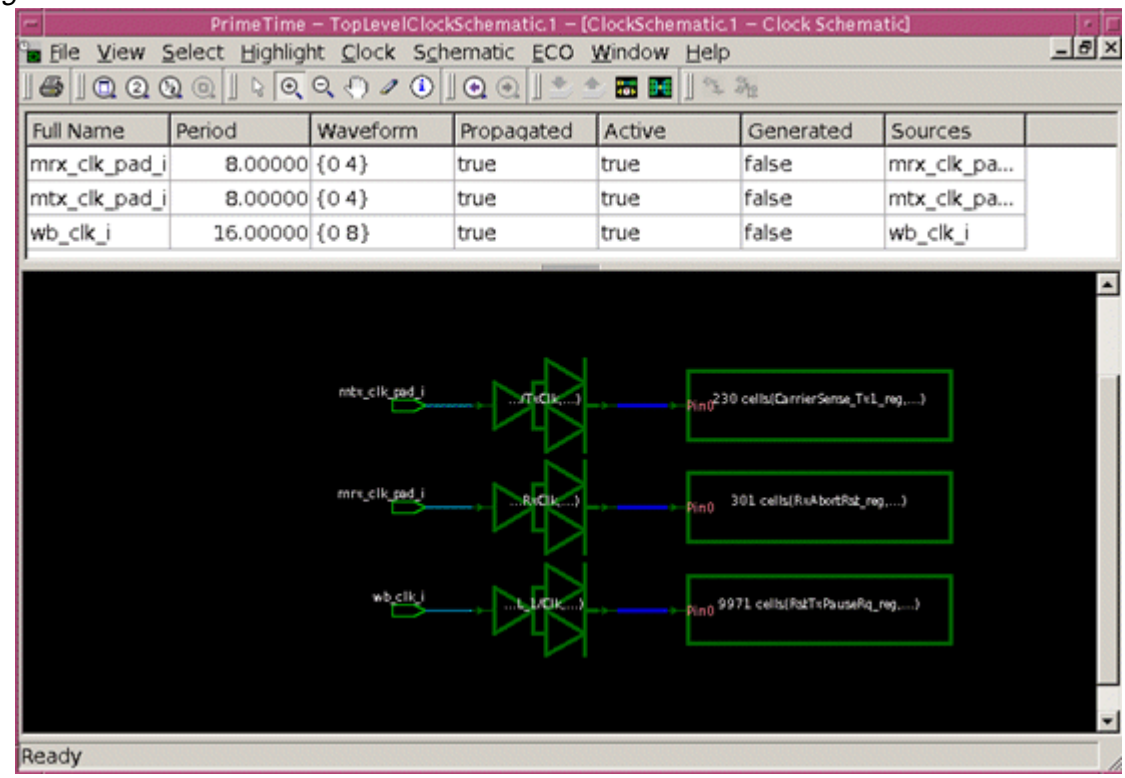
5. Click Modify to add or remove attributes from the "Group Attributes" list.
6. Click Save and then click Close.

Clock Schematic Window

There are several ways to access the Clock Schematic window. To access the Clock Schematic window, from the GUI choose Clock > Clock Schematic for Selected Clocks or choose Clock > Clock Schematic for Unclocked Pins. You can also access the Clock Schematic window from the Clock Analyzer window by right-clicking and choosing

Schematic of Selected Clocks. Another way to access the clock schematic is to use the Schematic menu from the Abstract Clock Graph window. Figure 4-21 shows an example of the Clock Schematic window.

Figure 4-21 Clock Schematic Window



For more information about the clock schematic, see “Clock Analyzer Window” on page 4-47 and “Abstract Clock Graph Window” on page 4-59.

Interactive Multi-Scenario Analysis Flow

The GUI in PrimeTime SI includes an interactive multi-scenario analysis flow that allows you to restore timing path collections simultaneously. This flow allows you to analyze and debug timing paths from multiple scenarios, and it is intended to be used as a quick analysis tool to help debug and track designs that are improperly constrained. You can also load path collections and categorize the timing paths by the available attributes.

With the interactive multi-scenario analysis flow, you can

- Load and debug multiple collections of timing paths

  The interactive multi-scenario analysis tool is designed to postprocess the timing paths across multiple scenarios within the same design.

- Categorize timing path collections

  Load timing path collections from the same design and categorize them. You can use existing rules or create new category rules. For more information, see "Path Analyzer Window" on page 4-31.

- Shift slacks of a category by a user-defined value

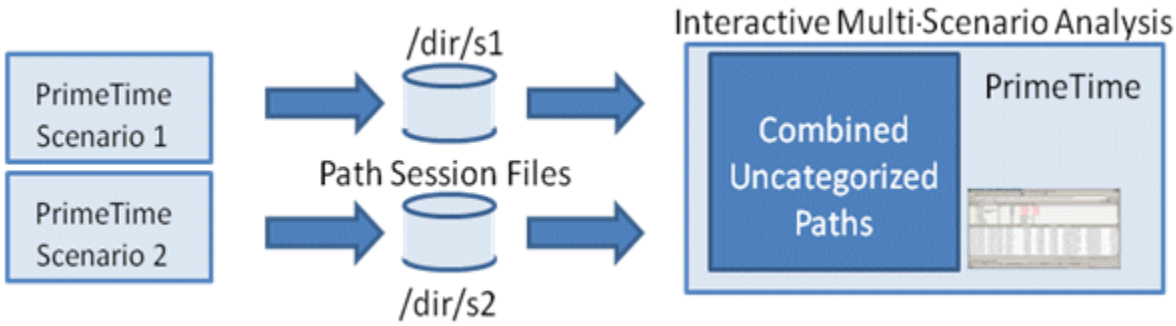
  For more information, see "Specifying a User-Defined Value for the Slack" on page 4-30.

- Reduce the number of times you run timing reports

  In the Path Inspector window, you can configure the information presented in the timing report without having to rerun it each time. Change the setting to add or remove a column of information. For more information, see "Timing Report" on page 4-46.

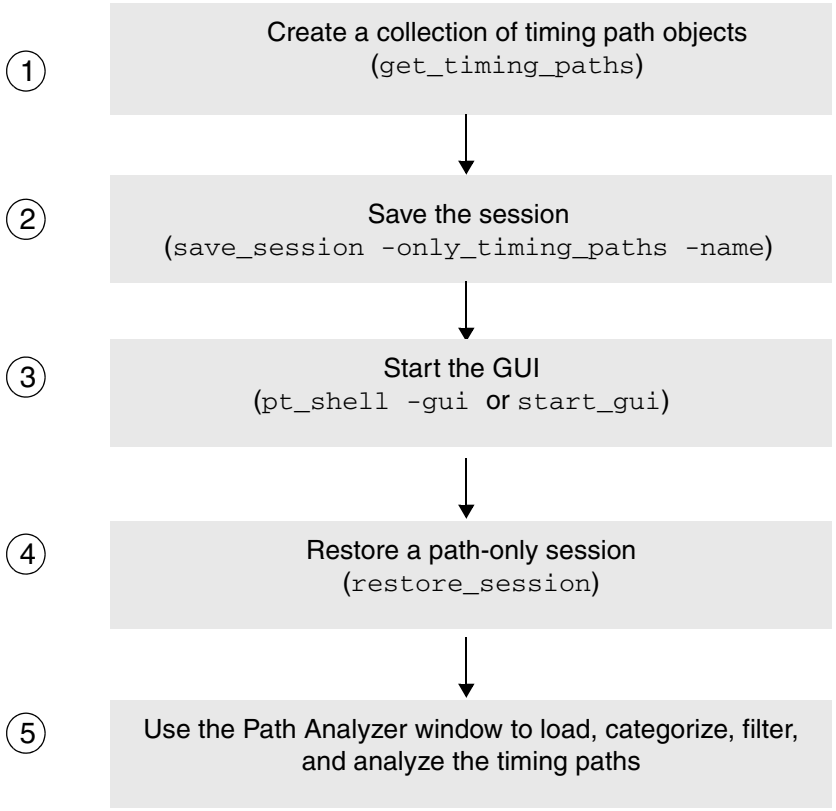
Figure 4-22 shows the high-level flow for the interactive multi-scenario analysis tool. First, save only the timing paths from one or more PrimeTime SI sessions. Next, restore multiple timing paths from the same design into a single session, which automatically changes PrimeTime SI to the interactive multi-scenario analysis mode. You can now analyze timing paths from multiple scenarios in the same session. You can then categorize the timing paths from multiple sessions using the Path Analyzer window. Use the Path Inspector, Schematics, and Histograms windows to further analyze the timing paths.

Figure 4-22 Interactive Multi-Scenario Analysis in PrimeTime SI



The interactive multi-scenario analysis flow described in Figure 4-23 shows the steps to create and save a collection of timing path objects. It also describes how to restore an analysis session and use the Path Analyzer window to categorize and analyze the timing paths.

Figure 4-23 Interactive Multi-Scenario Analysis Flow in PrimeTime SI



The interactive multi-scenario analysis flow described in Figure 4-23 is explained in more detail in the following steps:

1. Use the `get_timing_paths` command to create a collection of timing paths objects. For example,

```
pt_shell> set paths [get_timing_paths -start_end_pair \
          -slack_lesser_than 0.0]
```

2. Save the timing path collection sessions using the `save_session` command with the `-only_timing_paths` option. The saved session is a quick representation of the PrimeTime SI session; any session data that is unrelated to the timing paths is omitted.

The `-only_timing_paths` option saves the timing path collection in a representation that is independent of the timing data. The `-name` option assigns a name for the saved path session. You cannot use the `-include` or `-only_used_libraries` option with either the `-only_timing_paths` or `-name` option.

The saved timing collection sessions are smaller than regular saved sessions, and the session size is proportional to the size of the timing path collection. Paths from a distributed multi-scenario analysis (DMSA) mode can be saved using the remote execute feature.

The following example shows the `save_session` command:

```
pt_shell> save_session -only_timing_paths $paths /dir/s1
```

By default, the name assigned to the path session is the directory name where the session is saved. You can use the `-name` *`session_name`* option to assign a different session name.

3. To start a PrimeTime SI session that includes the GUI, use the `pt_shell -gui` or the `start_gui` command.

   Ensure you have a PrimeTime SI license.

4. Use the `restore_session` command to restore a session. You can restore multiple sessions of the same design in a single interactive multi-scenario analysis GUI session.

   You can incrementally load timing paths by using several instances of the `restore_session` command. For each restored session, the previously loaded timing path collections are maintained; they are not removed or overwritten. Path collections that are already loaded are kept, and another timing path collection is created. If you attempt to restore a collection containing the same name, the SR-023 error message appears. You can resolve this conflict by using the `-name` *`session_name`* option of the `restore_session` command.

   By default, the name assigned to the paths is the one used when saving the session. If no name was assigned when saving the session, the directory name where the session was restored is used. Alternatively, you can specify the name by using the `-name` *`session_name`* option. The following is an example of the `restore_session` command:

   ```
   pt_shell> restore_session /dir/s1 [-name paths_s1]
   pt_shell> restore_session /dir/s2
   ```

5. Use the Path Analyzer window to categorize and analyze the timing paths.

   In the Path Analyzer window, only attributes that are associated with the timing paths are available in the interactive multi-scenario analysis.

   The Path Analyzer window includes timing overviews of the timing paths by their groups, startpoint cell, or endpoint cell. The Path Analyzer window allows you to load multiple collections of timing paths within the same design using the `restore_session` command. This feature is supported only in the interactive multi-scenario analysis mode.

In addition, you can customize new category definitions for timing path collections and then populate the Path Analyzer window with the new grouping criteria. After defining the attribute of the path to be categorized, access the user-defined category from the “View paths by” list.

For more information about the Path Analyzer window, see Viewing the Timing Paths in the Path Analyzer Window.

6. Shift the slack of a category by a user-defined value.

   For more information about shifting slack, see “Specifying a User-Defined Value for the Slack” on page 4-30.

7. Use the Schematics, Path Inspector, and Histograms of Paths windows to further analyze the timing paths.

   If a GUI view or menu item is unsupported, it is either unavailable or partially grayed out. For example, the Abstract Clock Graph and Clock Analyzer windows are unavailable.

8. To return to a regular PrimeTime SI session, use the `remove_design -all` command to remove the path collections and exit the interactive multi-scenario analysis mode. An informational message is issued to let you know that the interactive multi-scenario analysis is no longer available.

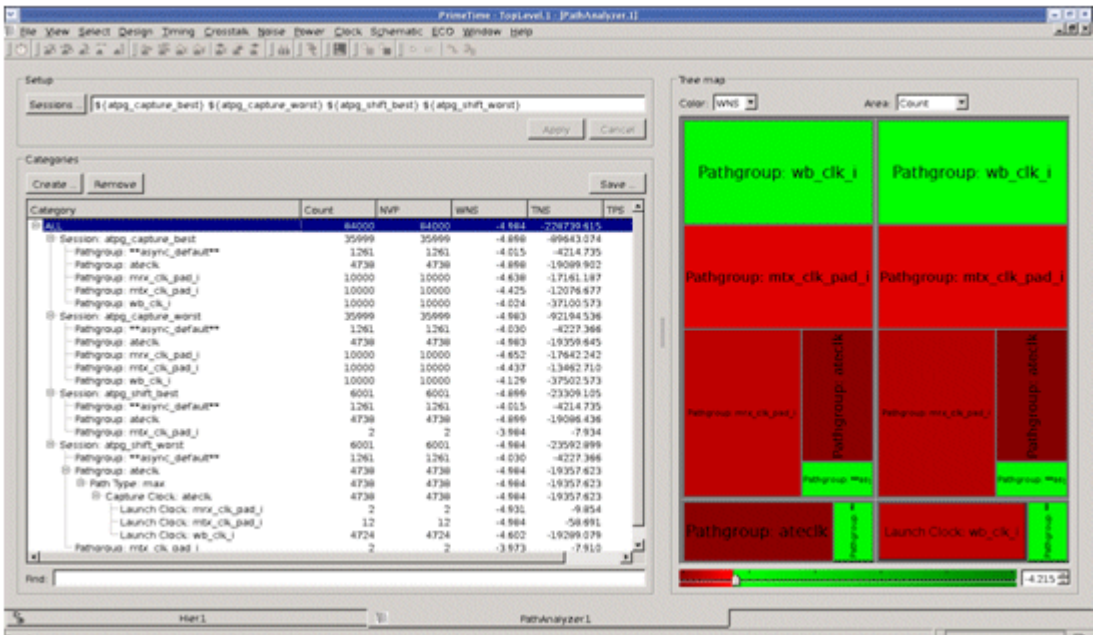
Viewing the Timing Paths in the Path Analyzer Window

To view the timing paths in the Path Analyzer window,

1. From the GUI, open the Path Analyzer window by choosing Timing > New Path Analyzer

Figure 4-24 shows the Path Analyzer window when you are using the interactive multi-scenario analysis flow.

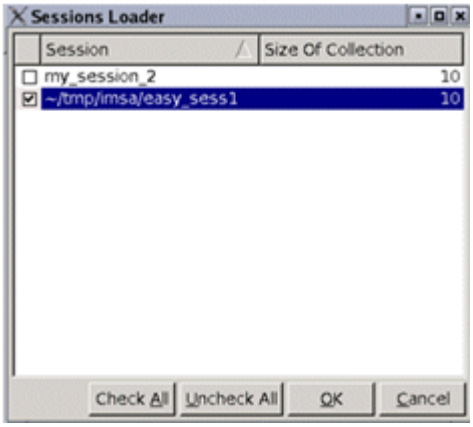
Figure 4-24 Path Analyzer Window - Interactive Multi-Scenario Analysis Flow



2. Select the session paths that you want to analyze by clicking Sessions.

   The Sessions Loader window, shown in Figure 4-25, appears showing all of the available sessions.

Figure 4-25 Sessions Loader Window



3. Select all the sessions you want to analyze and click OK to load the information.

   Use Check All to select all of the available sessions and Uncheck All to deselect all of the sessions.

4. From the Path Analyzer window, click Apply.

   Although there are separate sessions loaded and the paths have different session attributes, the paths appear in a single category called All. You can see the paths categorized by different sessions nested under the All category.

5. Categorize the timing paths using existing rules, or create new rules.

   You can create user-defined categories to categorize the timing paths. For more information, see the next section.

6. Use the tree map in the Path Analyzer window to evaluate the path groups.

   The Tree map displays hierarchical data in a tree structure as a set of nested rectangles. Each branch of the tree is displayed in a rectangle, which is then tiled with smaller rectangles representing subbranches. A rectangle representing a leaf node has an area proportional to a specified dimension on the data. The leaf nodes are colored to show a separate dimension of the data.

   For more information about the Path Analyzer window, see “Path Analyzer Window” on page 4-31.

Specifying a User-Defined Value for the Slack

In the interactive multi-scenario analysis mode, you can shift the slack of a category by a user-defined value in the Path Analyzer window. To shift the slack of a category:

1. From the Path Analyzer window, select a category.
2. Right-click to get a context menu, and choose “Shift Category”.

   The Shift Histogram dialog box appears as shown in Figure 4-26.

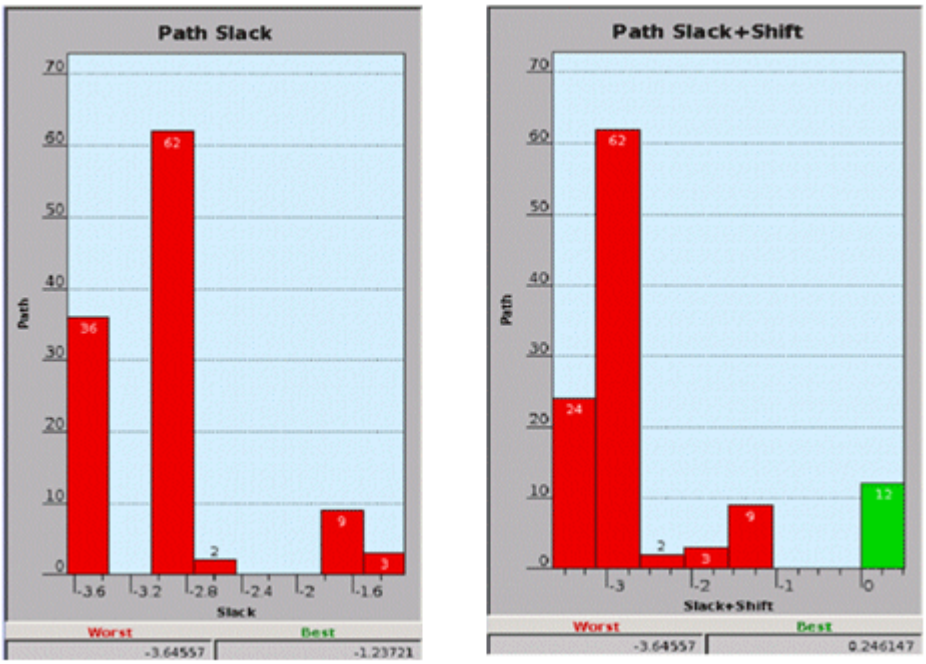
Figure 4-26 Shift Histogram Dialog Box

3. Select the slack shift value and parent category of the histogram.
4. Click OK.

The Table Histogram dialog box appears, which enables you to select the type of histogram to view.

5. In the Column field, select either the Slack+Shift or Slack option, select the appropriate options, and click OK. Figure 4-27 shows what you might see if you select either the Slack or Slack+Shift option.

Figure 4-27 Path Slack and Path Slack+Shift



6. View the histogram of the paths or parent category.

Path Analyzer Window

In the Path Analyzer window, you can load a collection of timing paths from the same design and categorize these timing paths by available attributes. You can also mark specific blocks, such as physical partitions, and categorize paths with these marked blocks. You can create custom categories based on available path attributes. You can also create category rules and display only the timing paths that you want to analyze.

To display timing paths by categories and create new category rules in PrimeTime, see the following sections:

- Loading the Timing Paths
- Categorizing the Timing Paths Using Existing Rules

- Creating User-Defined Category Rules
- Returning the Contents of a Category as a Collection
- Using Block Marks
- Analyzing the Timing Paths
- Recalculating Paths

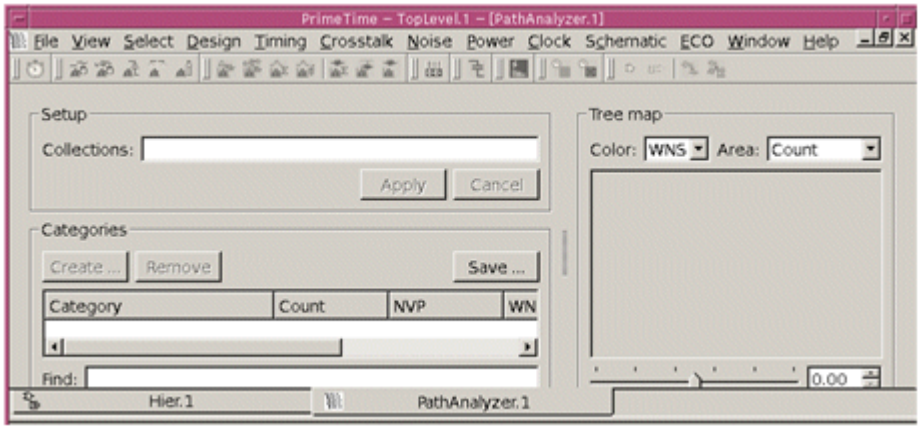
Loading the Timing Paths

The way you load timing paths in the Path Analyzer window depends on whether you are in the interactive multi-scenario anlysis flow or the normal flow. For more information about the interactive multi-scenario anlysis flow, see Interactive Multi-Scenario Analysis Flow. This section explains how to load timing paths in the normal flow.

To load timing paths,

1. From the GUI, open the Path Analyzer window, shown in Figure 4-28, by choosing Timing > New Path Analyzer.

Figure 4-28 Path Analyzer Window - Normal Flow



2. In the Collections field, type one of the following:
   - One or more timing path collections, separated by a space. For example, type $pathClct1 $pathClct2.
   - A command that provides a timing path collection.

     The command must be enclosed in Tcl style brackets [] to indicate that command needs to be evaluated to obtain the collection. For example, type `$pathClct1 [get_timing_paths ….]`. You can specify multiple collections in this way.

Specify only timing path collections. Those collections that do not contain timing paths are ignored. Objects, other than timing paths within a hybrid collection, are also ignored.

3. Click Apply.
4. Categorize the timing paths using existing rules, or create new rules.

   You can create user-defined categories to categorize the timing paths. For more information, see the next section.

5. Use the tree map in the Path Analyzer window to evaluate the path groups.

   The Tree map displays hierarchical data in a tree structure as a set of nested rectangles. Each branch of the tree is displayed in a rectangle, which is then tiled with smaller rectangles representing subbranches. A rectangle representing a leaf node has an area proportional to a specified dimension on the data. The leaf nodes are colored to show a separate dimension of the data.

Categorizing the Timing Paths Using Existing Rules

You can use the existing rules or create new category rules to categorize the timing paths. To categorize the timing paths,

1. From the Path Analyzer window in the GUI, click Create.

   The Create Category dialog box appears listing all of the existing category rules. You can create a filter or user-defined category. For more information about creating new category rules, see “Creating User-Defined Category Rules” on page 4-33.

   The paths are categorized by the rules that you select. For each category, the number of violating paths (NVP), worst negative slack (WNS), total negative slack (TNS), and total positive slack (TPS) are displayed in separate columns.

2. Select the rules that you want to apply to the timing paths, and click OK.

Creating User-Defined Category Rules

PrimeTime enables you to create user-defined categories based on available path attributes.

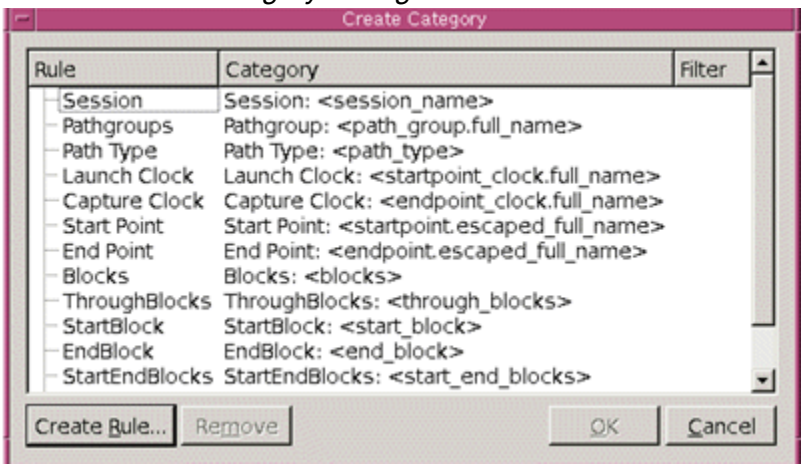
The priority of the user-defined categories is stored in the Category file. If a path does not satisfy any of the available grouping criteria, it is placed in the Uncategorized group for each level that has at least one subcategory.

To create a user-defined category rule,

1. From the Path Analyzer window in the GUI, click Create.

The following Create Category dialog box appears listing all of the existing category rules.

Figure 4-29 Create Category Dialog Box



The built-in rules make it easy for you to use the block-aware path attributes as dynamic category rules. The following built-in block-aware attributes of the timing paths are available:

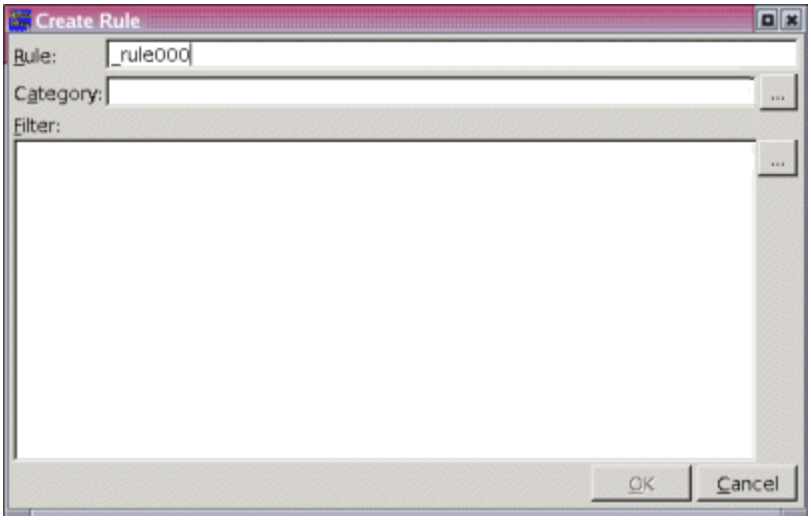
Rule name	Attribute	Definition
Blocks	`blocks`	An ordered block level path; includes start, through, and end blocks
ThroughBlocks	`through_blocks`	Includes through blocks, but does not include the start or end blocks
StartBlock	`start_blocks`	Includes the start block
EndBlock	`end_block`	Includes the end block
StartEndBlocks	`start_end_blocks`	A pair of blocks consisting of the start and end blocks
StartEndBlocksSorted	`start_end_blocks_sorted`	A pair of blocks consisting of the start and end blocks sorted alphabetically

For information about marking blocks, see “Using Block Marks” on page 4-38.

2. Click Create Rule.

The following Create Rule dialog box appears.

Figure 4-30 Create Rule Dialog Box



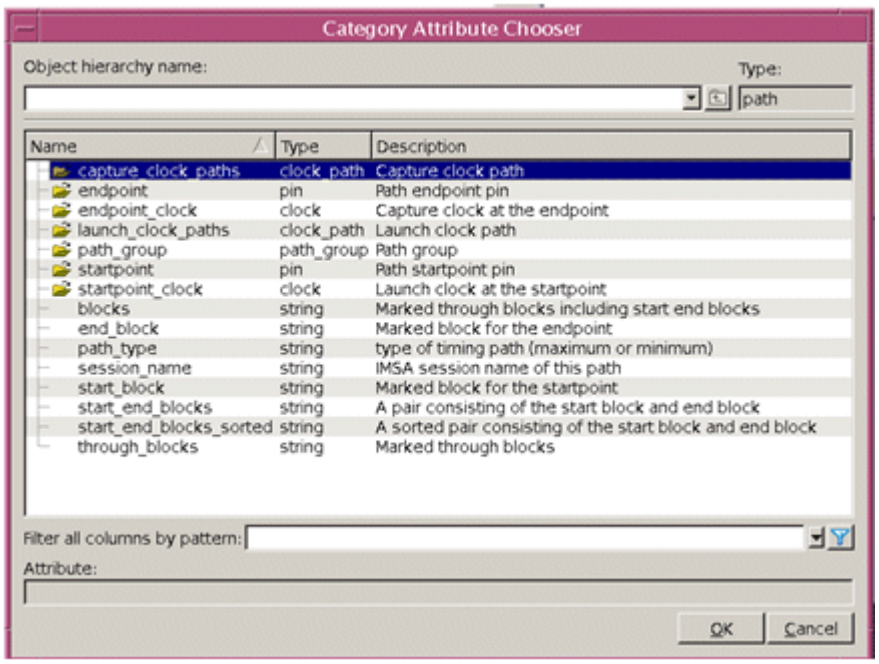
3. In the Rule field, type the name of the category rule that you would like to create.

   The name specified is only for reference, and you can change it.

4. In the Category field, type the category or click the ... option to select the appropriate path-related attributes.

When you click the ... option, the Category Attribute Chooser dialog box, shown in Figure 4-31, appears, which allows you to select an attribute and operator to construct a rule. After selecting an attribute and operator, click OK to return to the Create Rule dialog box.

Figure 4-31 Category Attribute Chooser Dialog Box



You can also use the Category field to specify a dynamic category. When creating a dynamic category, specify the attribute in the Category field and leave the filter field blank. For example, in the Category field you could type `EPCP: <endpoint_clock_pin.full_name>`, using the brackets to indicate a dynamic category. You can also concatenate multiple attributes using a space or other separator characters to create more complex dynamic categories. For example, you could type `EPC_SP: <endpoint_clock_pin.full_name> <startpoint.full_name>` to create a dynamic category based on the endpoint clock name and the startpoint name.

Note:

Using a slash (/) to separate the attribute values causes hierarchical categorization.

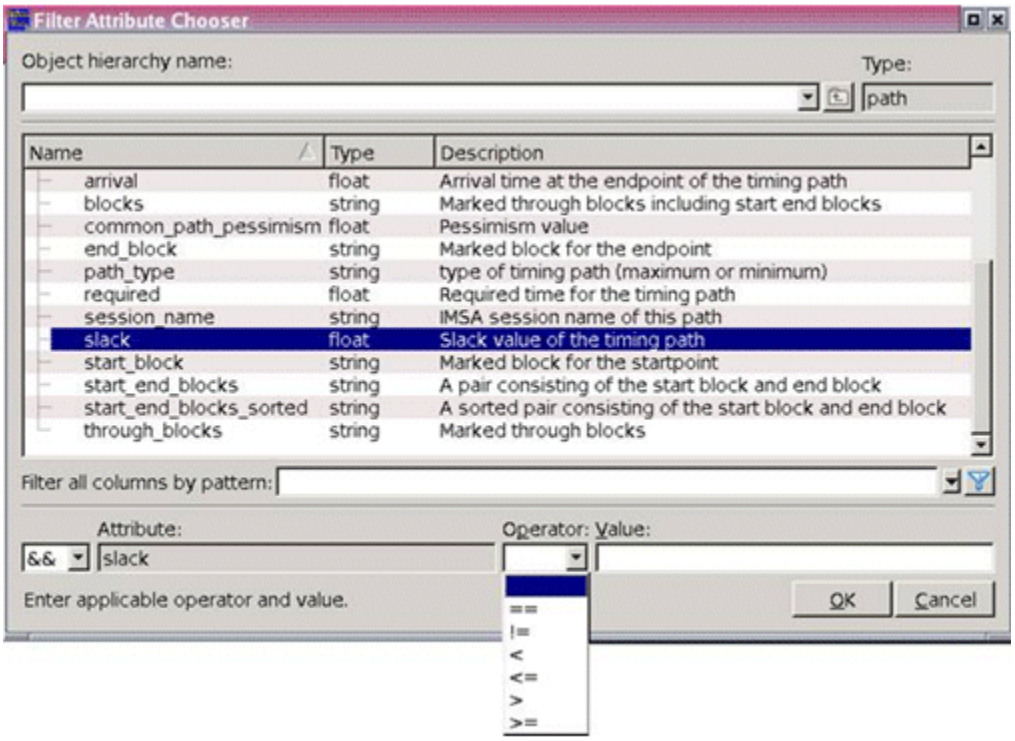
The information in the Category field is displayed in the Category section of the Path Analyzer window.

5. In the Filter field, specify a filter using the available attributes or click the ... option to select the appropriate attributes.

   When you click the ... option, the Filter Attribute Chooser dialog box, shown in Figure 4-31, appears. This dialog box allows you to construct full expressions with the `&&` and `||` combination box, which is updated based on the attribute that is selected. For

example, select slack < 0 to identify violating paths of negative slack. You can concatenate multiple filters just as you would concatenate logical expressions. Use && to specify an AND relationship between filter expressions, use || to create an OR relationship, and use parenthesis ( ) to define the evaluation order for the expressions.

Figure 4-32 Filter Attribute Chooser Dialog Box



6. Click OK.

   The new category rule is displayed in the Create Category dialog box. You can use this rule to filter the timing paths.

7. Select the new category rule, and click OK to filter the timing paths based on the rule.

Saving Categories

You can save the categories at any time. To save all of the categories, click Save from the Path Analyzer window. Specify a Tcl file name and location and click OK. You can then source the Tcl file to retrieve the saved categories. You should save the categories before removing any category rules to avoid losing work. Rules and filters that are used for categorization are also saved in the Tcl file.

Removing Categories

You can remove a subcategory or all of the categories. To remove a subcategory, highlight the category and click Remove. To remove all of the categories, highlight All and click Remove. All of the category rules that filtered the timing paths are removed.

Returning the Contents of a Category as a Collection

To return the contents of a specified category as a collection, including all direct and indirect offspring categories, use the `gui_get_category` command. For example, to obtain the contents of the ALL root category as a collection, use the following command:

```
pt_shell> set category_collection [gui_get_category -category {ALL}]
```

Use the `gui_get_category` command only if you have opened at least one Path Analyzer window. The contents of the category that resides in the specified Path Analyzer window are used to return the collection. For more information about this command, see the man page.

Using Block Marks

A block mark can appear in the GUI as a text label for a timing path category generated from a dynamic category rule. A common application is to mark interesting IP blocks within a design. You should keep these block marks short and meaningful. For example, you could use a team tag to mark blocks that belong to a specific team. You can use these block marks when categorizing timing paths and during block abstraction from the Advanced Clock Graph window.

- Use the `gui_set_cell_block_marks` command to set a block mark on one or more hierarchical or leaf-level cell instances.
- Use the `gui_get_cell_block_marks` command to get a list of the block mark string values on one or more hierarchical or leaf-level cell instances.

  For example, to set the block mark for a hierarchical cell instance identified by a cell name and then get the block mark of that cell instance, use the following commands:

  ```
  pt_shell> gui_set_cell_block_marks I_TOP/I_ALU  ALU
  pt_shell> gui_get_cell_block_marks I_TOP/I_ALU
  ALU
  ```

- Use the `gui_list_cell_block_marks` command to list the cell names and block mark values for all cells of the design that are marked.

  For example, to set block marks on hierarchical cell instances and then list all of the marked cell names and their block marks, use the following commands:

  ```
  pt_shell> gui_remove_cell_block_marks -all
  pt_shell> gui_set_cell_block_marks I_TOP/I_ALU  ALU
  ```

```
pt_shell> gui_set_cell_block_marks I_TOP/I_REG_FILE  REG_FILE
pt_shell> gui_list_cell_block_marks
I_TOP/I_ALU    ALU
I_TOP/I_REG_FILE    REG_FILE
```

To remove the block mark string value for one or more hierarchical or leaf-level cell instances, use the `gui_remove_cell_block_marks` command. To remove all of the block marks on any marked cell, use the `gui_remove_cell_block_marks -all` command.

Analyzing the Timing Paths

You can further analyze the timing paths in the GUI. By right-clicking in the Path Analyzer window, you can select category paths and create data tables. From the data table, you have access to the Path Inspector, Schematics, and Histograms windows to further analyze the paths.

Recalculating Paths

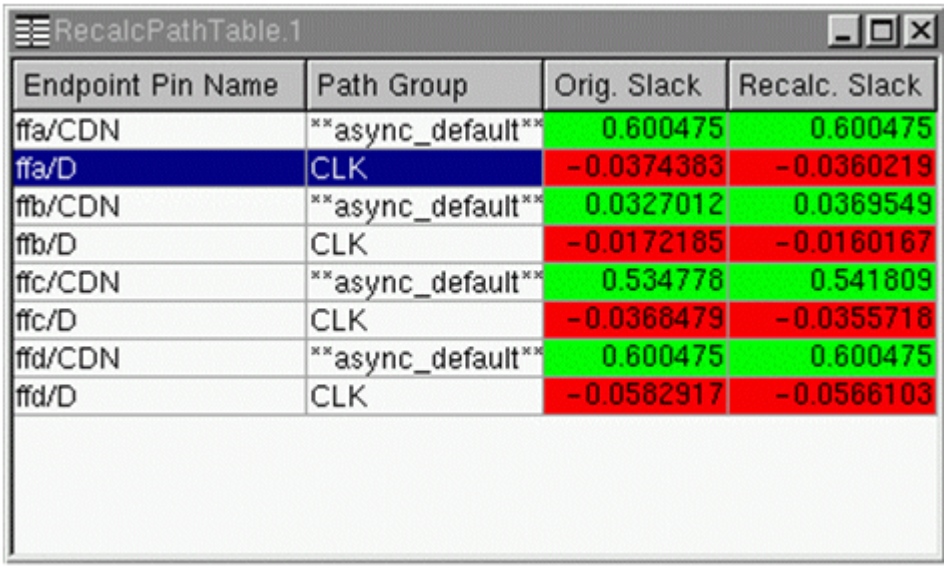
Using the recalculated path table, you can easily access and compare a set of normal paths to their recalculated path counterparts. From this window, you can also select a specific path and open a Path Pin Comparison that shows the timing path objects side by side, with the regular and recalculated values as well as the differences between them.

Recalculated Path Table

To generate a recalculated path table from the PrimeTime GUI, first select one or more of the paths that you want to recalculate from the Path Analyzer window. You can also use the `change_selection` command to select a set of paths from the command line. Next, choose Timing > New Recalculate Path Comparison Table. This creates a recalculated path table showing a summary of the normal and recalculated paths.

The information in the table includes the endpoint pin name, path group name, normal (path) slack, and recalculated (path) slack. If the slack is negative (violated), the cell appears red; otherwise, it is green. Figure 4-33 shows an example of a recalculated path table.

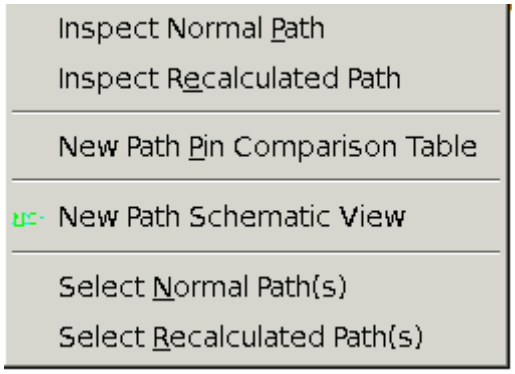
Figure 4-33 Recalculated Path Table



Endpoint Pin Name	Path Group	Orig. Slack	Recalc. Slack
ffa/CDN	**async_default**	0.600475	0.600475
ffa/D	CLK	-0.0374383	-0.0360219
ffb/CDN	**async_default**	0.0327012	0.0369549
ffb/D	CLK	-0.0172185	-0.0160167
ffc/CDN	**async_default**	0.534778	0.541809
ffc/D	CLK	-0.0368479	-0.0355718
ffd/CDN	**async_default**	0.600475	0.600475
ffd/D	CLK	-0.0582917	-0.0566103

You can view more information about the exact path differences between the normal and recalculated path. From the path comparison table, highlight the path, and right-click to obtain more options, such as the ability to launch the Path Inspector window to inspect the normal and recalculated path or to generate a path pin comparison table. Figure 4-34 shows the available options.

Figure 4-34 Recalculated-Path Context Menu

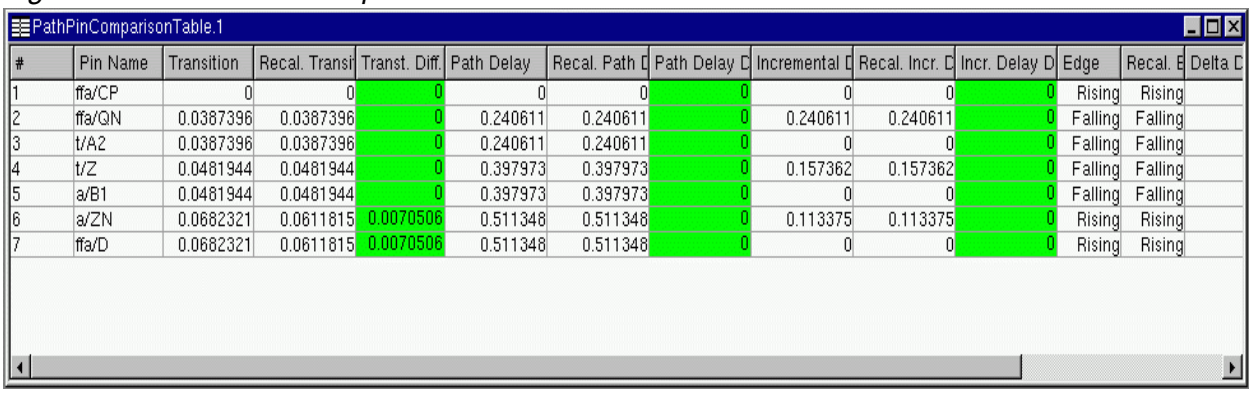


Note that the path pin comparison table is only enabled when the normal and recalculated path have the same number of through pins. The following section provides additional information about generating a path pin comparison table.

Path Pin Comparison Table

You can generate a path pin comparison table to see detailed information for normal and recalculated paths side by side. To generate a path pin comparison table from the PrimeTime GUI, you first select a row from the recalculated path table and then choose Timing > New Recalculated Path Pin Comparison Table. Alternatively, after selecting the path, you can right-click and select New Path Pin Comparison Table to generate this table. Figure 4-35 shows an example of a path pin comparison table.

Figure 4-35 Path Pin Comparison Table



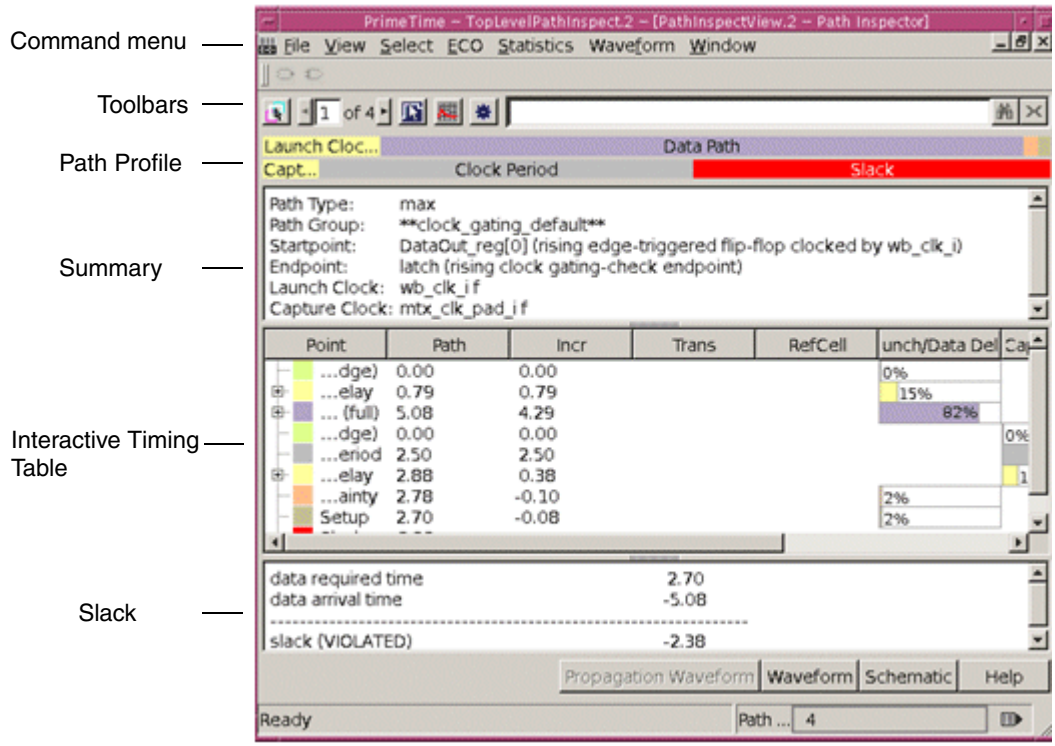
#	Pin Name	Transition	Recal. Transi	Transt. Diff.	Path Delay	Recal. Path D	Path Delay D	Incremental D	Recal. Incr. D	Incr. Delay D	Edge	Recal. E	Delta D
1	ffa/CP	0	0	0	0	0	0	0	0	0	Rising	Rising	
2	ffa/QN	0.0387396	0.0387396	0	0.240611	0.240611	0	0.240611	0.240611	0	Falling	Falling	
3	t/A2	0.0387396	0.0387396	0	0.240611	0.240611	0	0	0	0	Falling	Falling	
4	t/Z	0.0481944	0.0481944	0	0.397973	0.397973	0	0.157362	0.157362	0	Falling	Falling	
5	a/B1	0.0481944	0.0481944	0	0.397973	0.397973	0	0	0	0	Falling	Falling	
6	a/ZN	0.0682321	0.0611815	0.0070506	0.511348	0.511348	0	0.113375	0.113375	0	Rising	Rising	
7	ffa/D	0.0682321	0.0611815	0.0070506	0.511348	0.511348	0	0	0	0	Rising	Rising	

The path pin comparison table shows the pin names, pin transitions, path delays, incremental delays, edge, and delta delays along the pins in the normal and recalculated path, side by side. A positive change appears in green, and a negative charge appears in red.

Path Inspector Window

The Path Inspector window is a free-standing GUI window that enables you to analyze the details of a particular timing path. Figure 4-36 shows the Path Inspector window that is available by choosing Timing > Inspect Normal Path or Timing > Inspect Recalculated Path.

Figure 4-36 Path Inspector Window



The Path Inspector window is explained in the following sections:

- Toolbar Options
- Path Profile
- Configuring the Path Inspector
- Timing Report
- Accessing Additional Windows

Toolbar Options

Typically, the Path Inspector window is invoked with a set of timing paths already loaded. To access the Path Inspector window, load the design, select the timing paths from the path table, and choose Timing > Inspect Normal Path or Timing > Inspect Recalculated Path. In the Path Inspector window, you can:

- Load selected timing paths by clicking the “Load selected paths” icon (). Each timing path that is loaded appears on a separate page.
- View the individual timing paths by using the left and right arrow of the page icon () to scroll through the pages.

  This option becomes active only when there are multiple paths loaded in the Path Inspector window. One timing path is displayed per page. Clicking the next and previous arrows moves the current page forward or backward.
- Clear the global selection bus of the previously selected objects and select the currently displayed timing path instead by clicking the “Select path” icon ().
- Highlight the displayed timing path by clicking the “Highlight path” icon ().
- Configure the Path Inspector window by clicking the “Show path inspector settings dialog” icon (). For more information about this window, see “Configuring the Path Inspector” on page 4-44.
- Search the Interactive Timing Table section by typing a pattern in the text field and then clicking the "Find next" icon ().

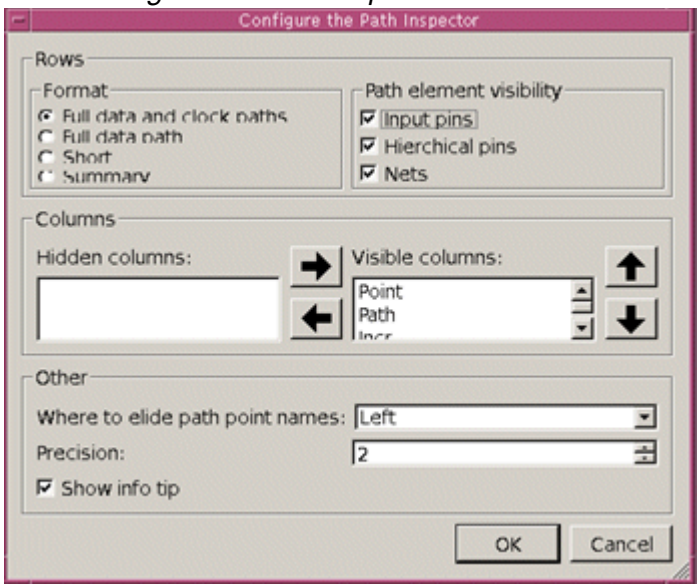
Path Profile

The Path Profile section of the Path Inspector window enables you to obtain an overview of a single path, which can help debug issues. The Path Profile section displays a comparison between the launch path and the capture path of a single timing path. Each bar width in the path profile represents the bar’s relative delay contribution. The launch and capture section are divided into subsections that help locate the reason the timing path failed.

Configuring the Path Inspector

You can use the Configure the Path Inspector dialog box, shown in Figure 4-37, to configure the contents of the timing path report. You can configure both report rows and columns.

Figure 4-37 Configure the Path Inspector



To configure the report rows, choose one of the following format options:

- Full data and clock paths

  This option corresponds to the `report_timing -path_type full_clock` or `report_timing -path_type full_clock_expanded` command. This format displays full data path and the full clock paths associated with the timing path, if any, in the path elements section. Whether the clock path is fully expanded with the clock path between a primary clock and a related generated clock depends on how the timing path being inspected was queried. The Path Inspector window shows the clock paths only if you queried the loaded timing path by choosing the “Full data and clock paths” option.

- Full data path

  This option corresponds with the `report_timing -path_type full` command. This format displays the full data path in the path elements section

- Short

  This option corresponds with the `report_timing -path_type short` command. This format displays only the startpoint and endpoint under the data path node that is shown.

- Summary

  This option is similar to the `report_timing -path_type summary` command. However, the Path Inspector displays the textual header and the end summary sections with no path elements section. You might find this useful to quickly iterate over a fairly large number of loaded timing paths to locate a specific path to inspect.

When a full path is displayed, the timing path is shown with a collapsed parent node indicating data, launch clock, or capture clock that can be expanded to show the full timing path elements. In the "Path element visibility" section select the options to include rows for that particular data in the report. To exclude the date, deselect the options.

Use the left and right arrows in the Columns section to show or hide specific columns within the report. Use the up and down arrows to change the order of the attribute columns.

In the Other group of the Path Inspector window, you have the following options:

- Use the “Where to elide path point names” option to specify how the name should be shortened if there not enough space in the name field.

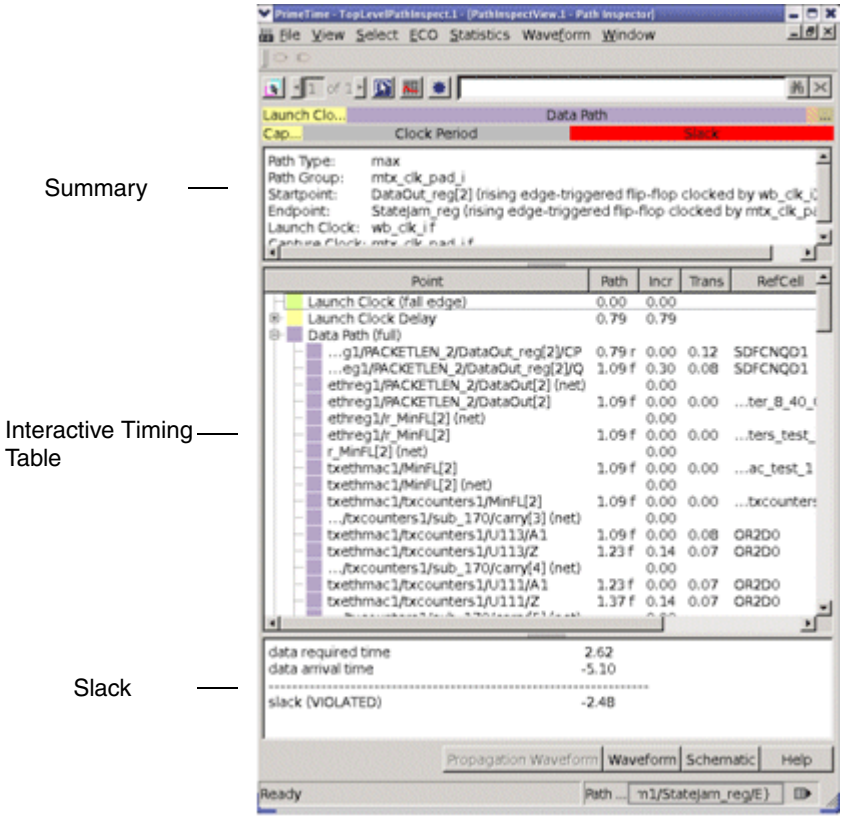
  The available options are Left, Right, and Middle. For example, if a pin is called "abcde" and there is only room for three characters, by choosing the Right option, the name would be displayed as "abc…". If you chose the Left option, the name would be displayed as "…cde", and if you chose the Middle option, the name would be displayed as ab...e.

- Use the Precision option to determine the precision for float numbers in the table.
- Use the Show info tip option to enable the informational tips to be displayed when you hover over them with the mouse.

Timing Report

The timing report is displayed in the Summary, Interactive Timing Table, and Slack sections of the Path Inspector window. These reports can be helpful for analyzing the timing path.

Figure 4-38 Path Inspector Window



To navigate the report content, use the information available in the following sections of the Path Inspector window:

- Summary

  Shows the summary information for the timing path.

- Interactive Timing Table

  Displays the timing path elements in sections such as data path, clock period, and clock uncertainty. You can expand or collapse each section. Use the following guidelines to navigate:

  - View a clock, pin, or net design object associated with a path element by selecting the path element row in the report.
  - View the launch clock path by selecting the launch clock delay node.

- View the capture clock path by selecting the capture clock delay node.
- View the timing path by selecting the data path row.

- Slack

  Displays a textual report similar to the `report_timing` command.

Accessing Additional Windows

You can select the following options from the Path Inspector window:

- Schematic

  By default, when you click the Schematic option, the Schematic window opens with the full data path, launch clock path, and capture clock path. Launch and capture clock paths often have a common section, and they diverge at a certain point called the clock reconvergence pessimism removal (CRPR) point. In the F-2011.06 Beta release, you can set up the schematic preferences to display the full data path but filter the common parts so that the CRPR point onwards is displayed.

  To filter the launch and capture clock paths that are common, use the following command:

  ```
  gui_set_pref_value -category pathInspect \
       -key mask_clock_paths_common_path -value true
  ```

- Waveform

  Displays timing waveform diagrams in the Waveform window.

- Propagation Waveform

  Displays propagation waveforms, which are available only for recalculated timing paths, in the Propagation Waveform window.

- Help

  Displays help for the Path Inspector window.

Clock Analyzer Window

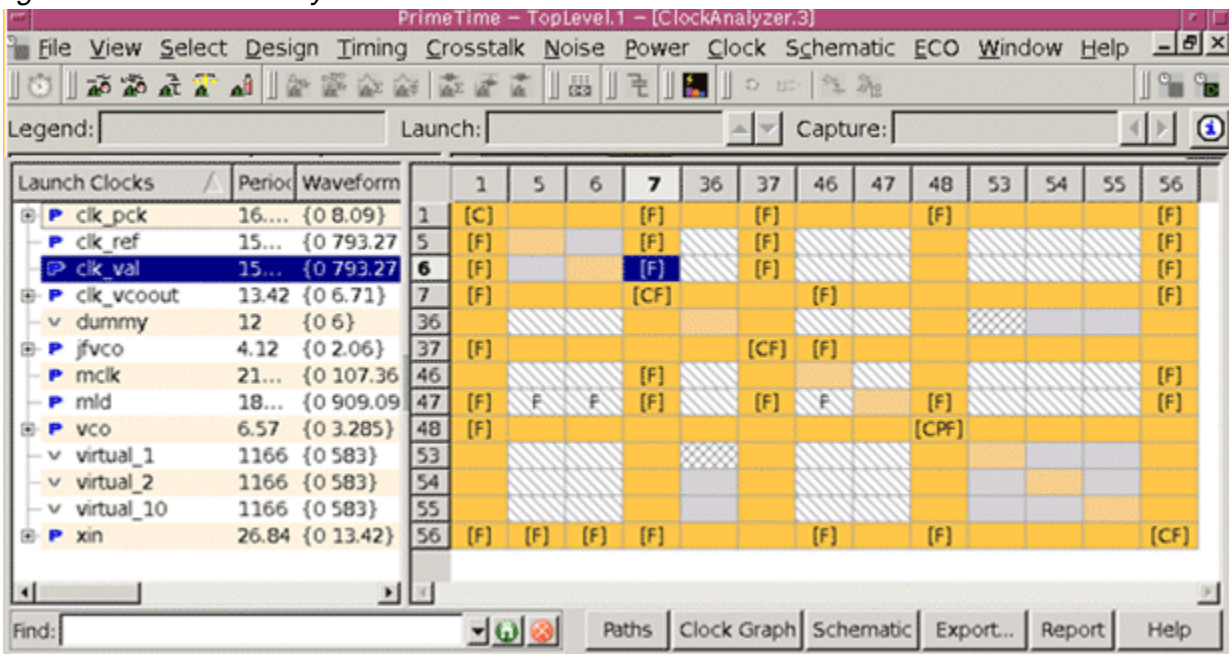
Use the Clock Analyzer window to quickly identify clock-to-clock relationships. The window helps you determine which clock domains communicate with other clock domains and also identify the clock domain crossings. You can also use the Clock Analyzer window to expand and collapse clocks based on the clock domains you are analyzing. The Clock Analyzer window consists of the clock tree and clock matrix, which shows the top-level primary clock and the entire set of dependent generated clocks beneath it.

A clock domain consists of a primary or generated clock and all the clocks derived from it. The generated clock has its own domain, which is a subset of the master clock domain. You can select one or more clocks. By selecting a parent, you are also selecting its children. Therefore, if you select the master clock, you are selecting the entire clock domain.

Each row of the matrix represents a clock in the design and has a corresponding number that is used to label the columns of the matrix. The resulting cells in the matrix identify the clock-to-clock relationships. As you expand or collapse the tree view, the corresponding rows and columns of the matrix expand and collapse appropriately. When you select a cell in the clock matrix, the Launch Clock and Capture Clock fields are populated with the current clock information. You can access the context-sensitive menu by highlighting and right-clicking one or more clocks in the tree view.

To graphically analyze the clock relationships in your design and drive the clock analysis, choose Clock > Clock Analyzer. Figure 4-39 shows the Clock Analyzer window.

Figure 4-39 Clock Analyzer Window



You can select the following options from the Clock Analyzer window:

- Paths

  Loads the paths for a specific launch-capture clock pair to the Path Analyzer window

- Clock Graph

  Accesses an advanced clock graph visualization of the selected clocks

- Schematic

Displays an abstract schematic of the selected clocks in a separate top-level window

- Export

  Saves clock information to a file in a comma separated value (CSV) format

  Select the All Clock Attributes option to export all clock attributes. If this option is not selected, only the attributes displayed in the tree view are saved.

- Report

  Generates a report for the selected clocks by running the `report_clock` command

- Help

  Displays the Clock Analyzer help information.

Each matrix cell indicates the clock domain or specific clock-to-clock relationships. The path constraints and synchronous or asynchronous relationship are identified using letters and color associations. For explicit paths, the Legend field shows the letter and description that corresponds to the selected cell. When a clock domain is collapsed, a summary of the path constraints is shown in the matrix cell, which means that a collapsed cell can have several letters associated with it. The collapsed clock domain is indicated with a light brown cell to differentiate it from other cells.

PrimeTime identifies the following types of clock crossings for paths with a launch and capture clock pair:

- Empty cell - No paths

  No paths exist between the corresponding launch and capture clock except when the launch and capture clock are the same

- C - Constrained paths

  Fully constrained paths exist between the launch and capture clocks of the selected matrix cell

- F - False paths

  All paths between the selected launch and capture clock pair have been designated false paths

- P - Partially constrained paths

  Only some paths between the selected launch and capture clock pair are designated false paths

Each cell represents a launch and capture clock pair. The cell is colored to visually show the relationship between the clocks. Table 4-2 shows the background cell pattern and color that indicates the synchronous or asynchronous relationships between the launch and capture locks.

Table 4-2 Background Cell Pattern

Pattern	Description
	Diagonal orange lines indicates that the launch and capture clocks are from the same clock domains and asynchronous with respect to each other.
	Diagonal gray lines indicates that the launch and capture clocks are from different clock domains and asynchronous with respect to each other.
	Solid gray indicates that the launch and capture clocks are from different clock domains but synchronous with respect to each other.
	Solid light orange indicates that the launch and capture clocks are from the same clock domain and synchronous with respect to each other.
	Solid dark orange indicates that either the launch and capture clocks corresponding to a particular cell are the same clock.
	Crisscross pattern indicates that the launch and capture clocks are logically or physically exclusive.

From the tree view of the Clock Analyzer window, you can expand or collapse fanout levels for the clock tree displayed in the active clock tree. You can expand or collapse all levels, or expand a specified number of levels. To help you locate clock objects and focus on a set of clocks of interest, you can sort by column, search for a clock within a clock domain, and apply filtering criteria.

You can also select the clock matrix and right-click the “Analyze Paths from *clock_name*” option. The Path Analyzer window appears, which allows you to evaluate timing overviews of the timing paths by their groups, startpoint cell, or endpoint cell. For more information about the Path Analyzer window, see the “Path Analyzer Window” on page 4-31.

Clock Schematics

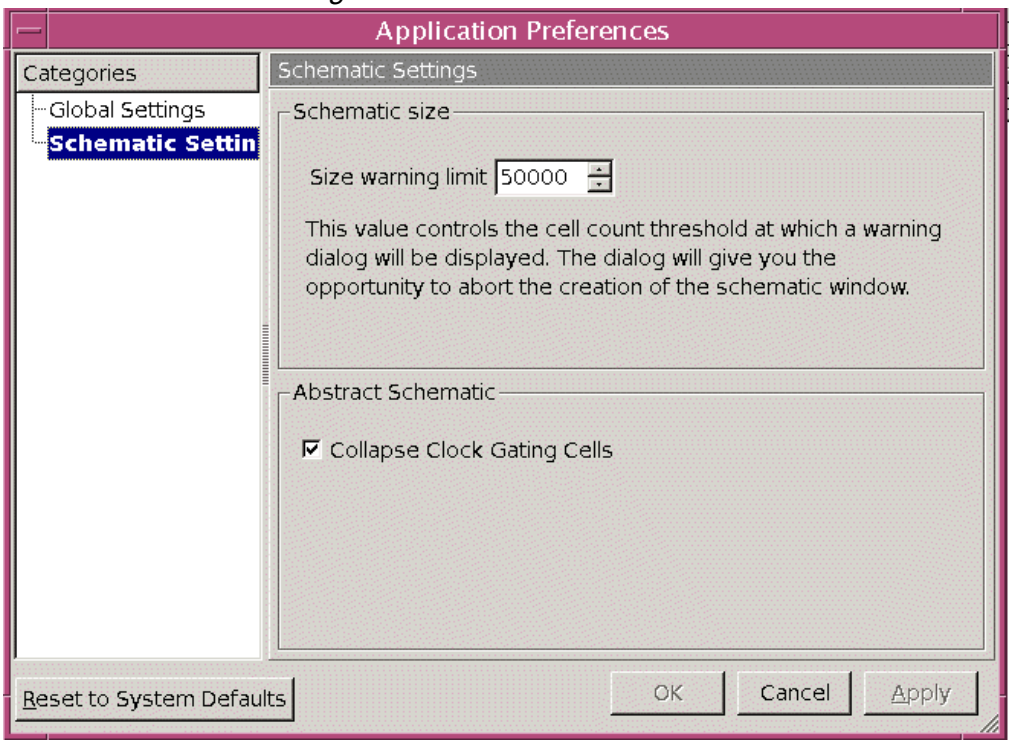
For ease of use, clock schematics collapse buffer and inverter clouds in the clock tree to reduce the size of the schematic. Sequential load cells that share the same functional clock signals are also collapsed together. To help with the analysis, the propagation paths of one or more clocks can be highlighted by color.

Metaobjects are representations of multiple objects of the same type. The clock tree schematics introduce the following terms and symbols:

- Metabuffer - A cluster of buffers and inverters collapsed together. This is a single symbol with one or two outputs: inverted with a circle at the output pin or noninverted without a circle at the output pin, or both.
- Metanet - A cluster of nets collapsed together because the connected objects were collapsed.
- Metahierarchy - A cluster of multiple hierarchy symbols collapsed together.
- Metaflop - A cluster of sequential load cells collapsed together.

The clock tree schematic feature allows collapsing of clock-gating cells into a metabuffer. This feature allows PrimeTime to consider clock-gating cells to be extended buffer cells for the purpose of buffer-tree collapsing. To enable the collapsing of clock-gating cells, choose View > Preferences in the user interface. Next, from the Schematic Settings window, select the Collapse Clock Gating Cells option as shown in Figure 4-40.

Figure 4-40 Schematic Settings Window

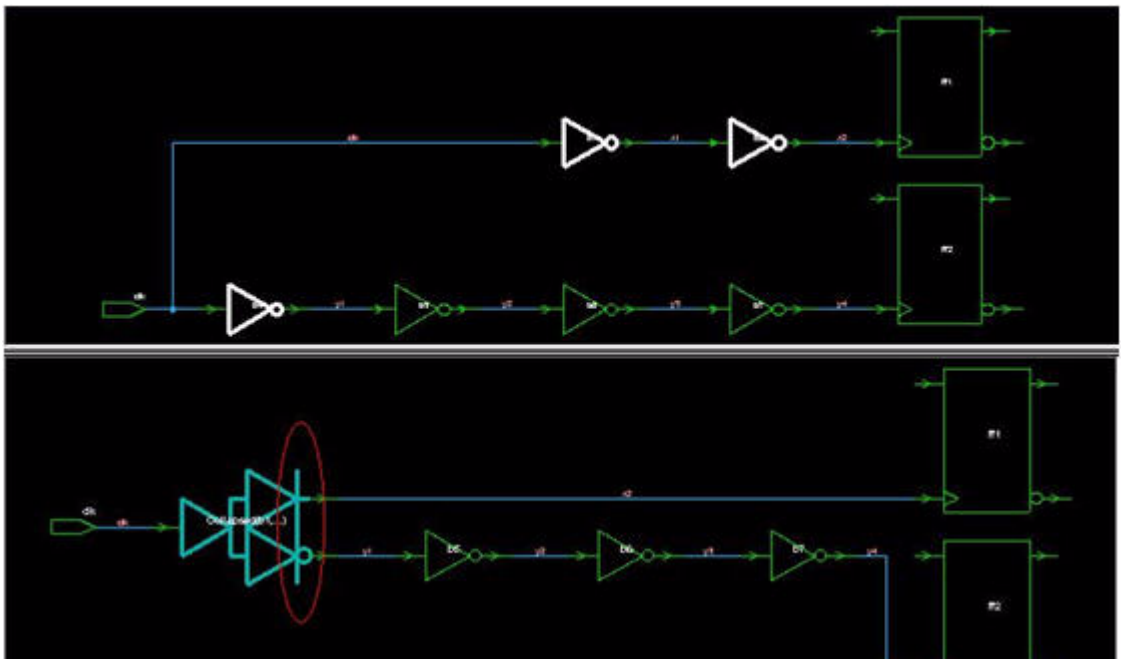


Expanding and Collapsing Operations

Expand and collapse operations are available in both the right-click context menu, and by choosing Clock > Schematic for Selected Clocks. To expand selected metaobjects, choose Expand Selected. Alternatively, you can choose Expand Unselected to expand metaobjects that are not selected. For example, you can obtain a fully unexpanded schematic by choosing Clock > Expand > Expand Unselected on the clock schematic with nothing selected. To collapse some objects, select them in the expanded schematic and choose Clock > Collapse > Collapse Selected.

In addition, you can use Expand First Level and Expand Last Level operations to expand selectively a logic level (fanin or fanout) out of the metaflops. Figure 4-41 shows how the schematic changes with collapse and expand operations. The top pane shows the expanded schematic, and the bottom pane shows the schematic after only the selected items are collapsed.

Figure 4-41 Collapse of Selected Buffer or Inverter Cells



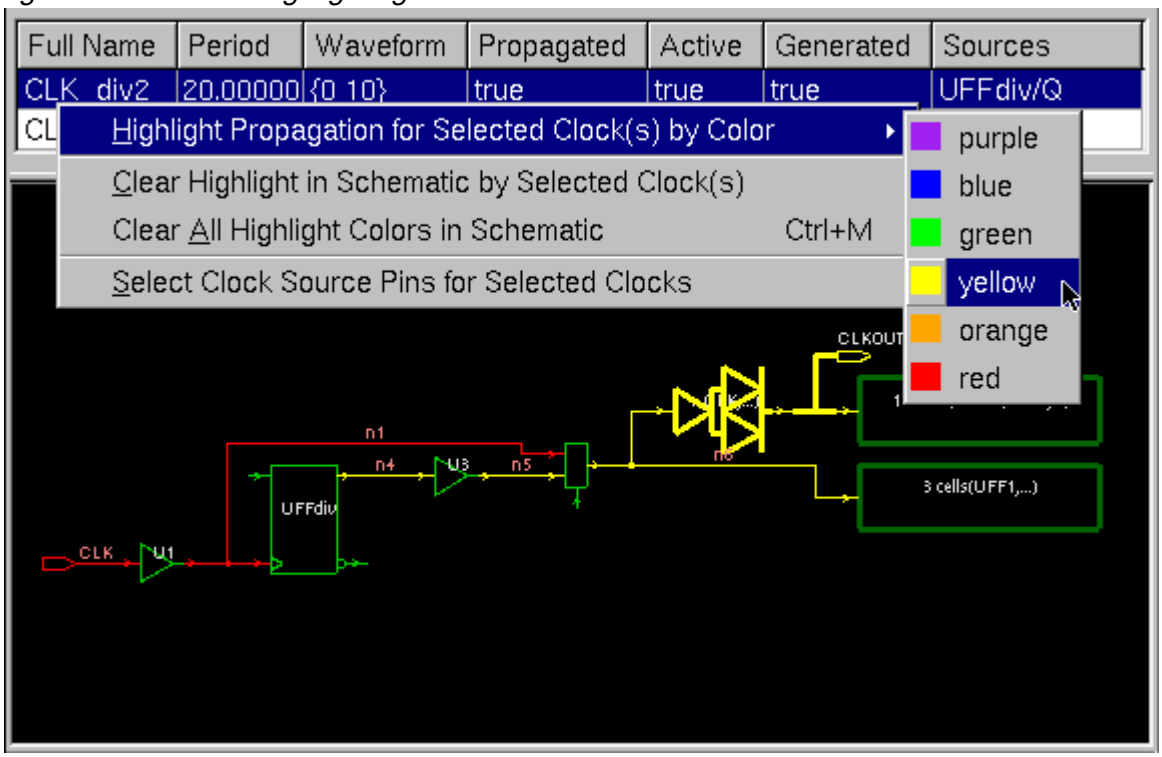
You can add logic to the schematic by using the Add Fanin/Fanout tool. This method can be useful for exploring the side inputs of control logic cells.

Clock Highlight and Select Operations

When multiple clocks are shown in the clock schematic, you can highlight the propagation path of selected clocks in different colors. The right-click context menu of the clock table above the schematic area has an entry "Highlight Propagation for Selected Clock(s) by

Color > (purple | blue | green | yellow | orange | red).” You can use this feature to highlight selected clocks in the color of your choice. For an example of clock highlighting, see Figure 4-42.

Figure 4-42 Clock Highlighting



When a net is in the propagation of multiple highlighted clocks, the last color applied is kept. You can also clear clock highlighting by clock using “Clear Highlight in Schematic by Selected Clock(s)” or clear all highlighted colors by using “Clear All Highlight Colors in Schematic.”

Normal schematic tools can also be used in the clock schematic. You can select the load or driver of selected objects by choosing the Select > Driver of Selected or Select > Load of Selected option. You can also select the entire fanin or fanout of selected objects by right-clicking and choosing Select Fanin Cone or Select Fanout Cone.

ToolTips and Information

You can get metaobject information through ToolTips as shown in Figure 4-43 and Figure 4-44.

Figure 4-43 Collapsed Cells and ToolTip

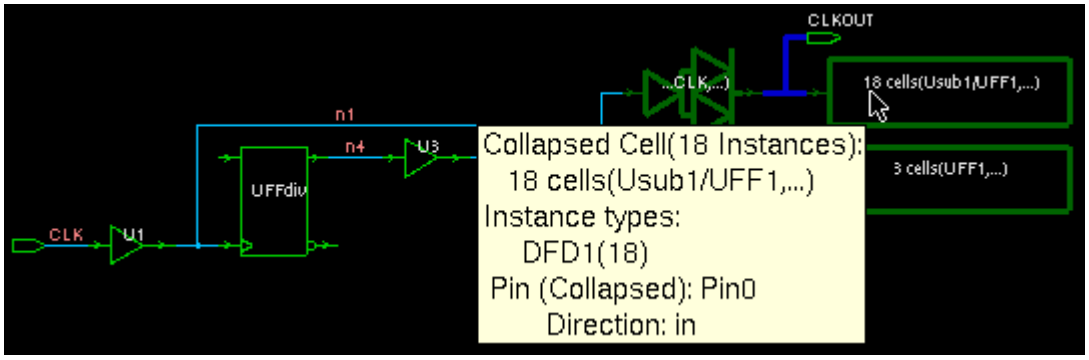
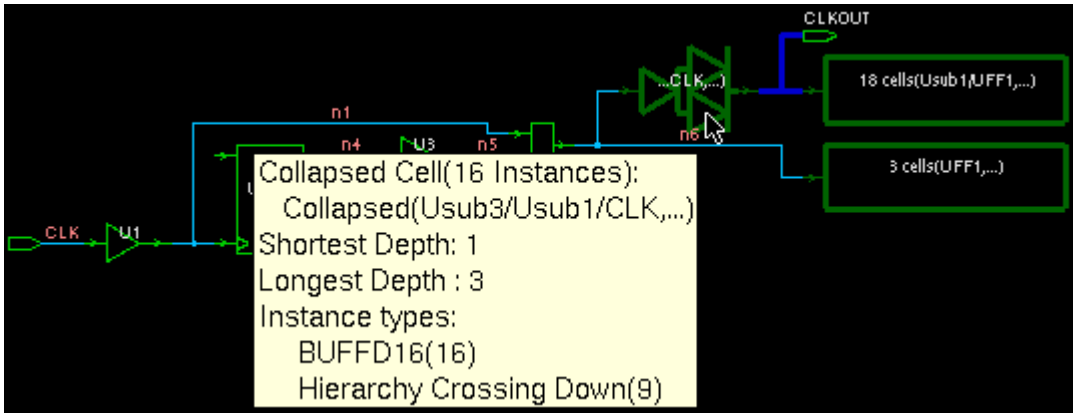
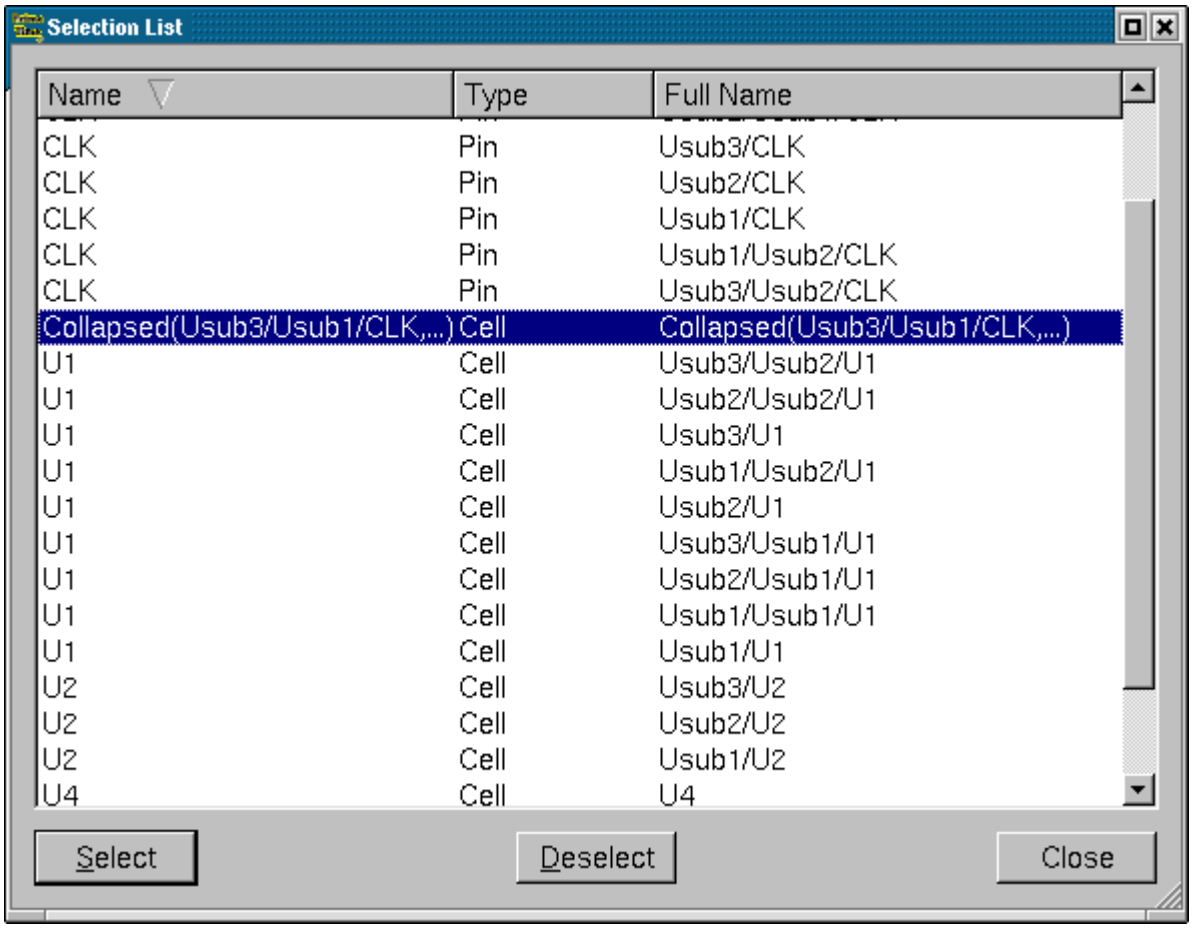


Figure 4-44 Collapsed Buffers With ToolTip



Selected metaobjects are visible in the selection list (Select > Selection List). When a metaobject is selected, all of the items it contains are also implicitly selected and appear in the selection list. Metaobject names start with “Collapsed” and contain their children. For an example, see Figure 4-45.

Figure 4-45 Selection List with Three Metaobjects (Names Starting With “Collapsed”)



Note that simplified clock tree schematic functions are enabled only when a clock is selected in a clock table. These functions are not available if a clock is selected in a separate Schematic or Hierarchy Browser window.

Sorting by Attributes

A design can have hundreds of clocks with some clock embedded deep into the tree. To assist in locating clock objects so that you can focus on a set of clocks, the Clock Analyzer window provides sorting by attributes. By default, the clocks are sorted alphabetically in the clock tree; however, you can sort the clocks by any available column. Clicking any column in the clock tree header sorts the clocks by the corresponding attribute in descending order. Clicking athe same column again sorts the clocks in ascending order. Regardless of the sorting order, the clock hierarchies are retained.

Find Tool

The Clock Analyzer window has a Find capability that can help you locate a specific clock. You can perform multiple selections and select a range of clocks, and you can locate launch clocks based on the functional definition such as pins along the clock path.

To locate a specific clock,

1. Display the Find tool by selecting a launch clock, right-clicking, and selecting “Find Clock(s)”.
2. In the “Find by” list, select Name, Cells, Pins, or Lib Cells to search for clocks by name, cells, pins, or library cells.

   When you select Name from the “Find by” list, the original clock name is located. PrimeTime locates the clocks that match the search criteria, and you can use wildcards.
3. In the text box, type the clock name or a portion of the name and then press enter or click the “Apply find-by-name” icon ().

   All matching clocks are displayed in the Clock Analyzer window.
4. Use the launch or capture feature available in the interactive Clock Analyzer window.

   The Clock Analyzer window is interactive. You can locate the next capture or launch clock in the clock matrix by clicking the arrows in either the Launch or Capture box.

   Navigate from one capture clock to the next by using the left and right arrow keys. Navigate to the next or previous launch clock by using the up and down arrow keys. if you select a matrix cell in the clock matrix, the arrow keys navigate to the corresponding cell that contains a relationship and skips the matrix cells with no relationship.

   Note:

   If a clock is within a collapsed tree, it is not located because PrimeTime performs the search only on visible clocks and does not automatically expand the tree.

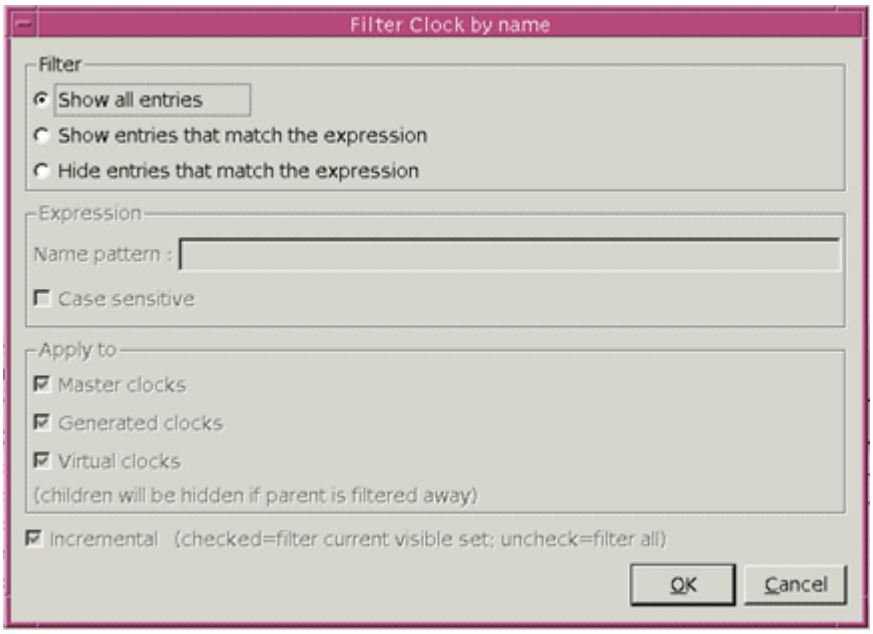
You can perform multiple selections using the mouse and Ctrl+Click. You can select Shift+Click to select a range of clocks. You can also type multiple clock patterns using a space as the delimiter. In this situation, the results of the searches for each different pattern are added together. You can begin typing a clock name and then hit the tab key to auto-complete the clock name if it is unique. If there are multiple names, a list with possible matches is displayed.

To select all of the generated clocks for a given master clock, highlight a launch clock, right-click, and select “Deep Select all Generated Clocks”. Selected matrix cells are shown by only highlighting the cell border rather than the entire cell so that the contents of the matrix cells are visible appropriately.

Filtering in the Clock Analyzer

You can filter the clock domains to reduce the number of visible clocks by including or excluding clocks whose names match a certain pattern or type. Highlight the clock domain, right-click, and choose Filter. The Filter Clock by Name dialog box appears, which is shown in Figure 4-46.

Figure 4-46 Filter Clock by Name Dialog Box



The available choices are

- Show all entries
- Turns filtering off and displays all of the entries.

  Show entries that match the pattern string in the Expression box. If a parent clock does not match the pattern, the associated children are not displayed.

- Hide entries that match the expression

  Turn on inverted filtering. Clocks that match the pattern string in the Expression box are hidden. If a parent clock matches the pattern, the associated children are not displayed.

In the Expression section, you can enter a string pattern to use when searching. You can use the asterisk (*) to match any string and the question mark (?) to match a single character. Patterns are matched using substring matching within the clock name.

In the “Apply to” section, specify whether the filtering should apply to master clocks, generated clocks, virtual clocks, or a combination of these clocks and their associated children.

Abstract Clock Graph Window

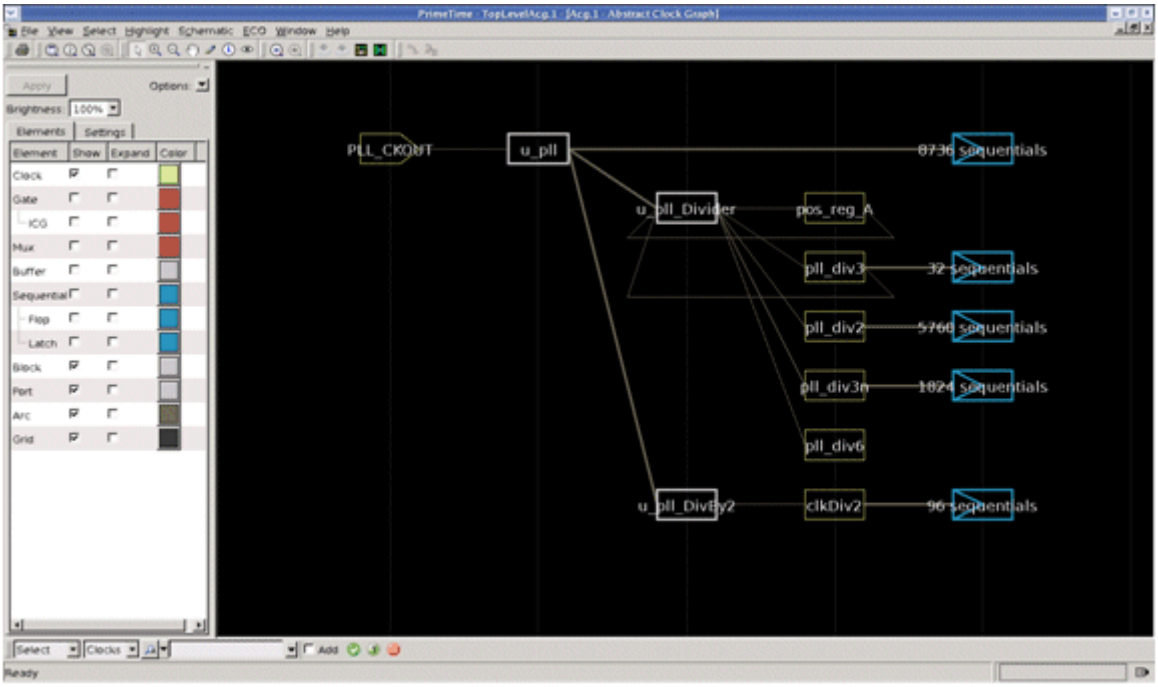
The Abstract Clock Graph window provides fine-grained control over abstraction, multilevel undo and redo capability, and detailed query reports. The abstract clock graph provides you with the ability to analyze and debug clocks at a higher level. The clock graph component improves visualization of the gates, registers, and buffers associated with a single clock by using a placement algorithm that positions the gates relative to the arrival times of their clock inputs. This abstract clock graph helps you identify the relationships between different clocks, such as primary and generated, clock convergence points, classes of clocked elements, and logic levels. The clock graph view displays time-based representations of clock tree fanout networks. The view is a symbolic representation of high-level analysis tasks and conveys the network clock structure within a tree view. You can use the clock graph view to examine clock tree fanin and fanout in a time-based display that facilitates clock tree latency and skew analysis.

You can select a timing arc between the master clock to a related generated-clock source pin, and the abstract clock graph shows the schematic of that path. You can select a clock or clock node to access the clock schematic of the selected clock. If you select a node that is not a clock, you can obtain the path schematic of the elements that the node represents by selecting the node and then right-clicking for menu options. The menu options change depending on the relevant schematic.

The clock graph view displays a symbolic representation of high-level analysis tasks and conveys the network structure within a tree view that is similar to a schematic display. However, the abstract clock graph view is a more intuitive and effective way to view the network structure than a general schematic view.

The Abstract Clock Graph window, shown in Figure 4-47, enables you to analyze clocks at a higher level. It helps identify relationships between different clocks, such as primary and generated, clock convergence points, classes of clocked elements, and levels of logic. The Abstract Clock Graph window is accessible in several ways through the GUI. To access clock graphs for all the clocks in the design, choose Clock > “Clock Graph for All Clocks”. To access selected clocks, choose “Clock Graph of Selected Clocks” from the Clock Analyzer window. To display the clock domain of a defined clock as a clock graph so that you can see which clocks are related, highlight a clock or set of clocks, right-click, and select “Clock Graph for Clock Domain”. PrimeTime displays an advanced clock graph visualization of the selected clocks.

Figure 4-47 Abstract Clock Graph Window



The Abstract Clock Graph window enables you to focus on the details of clock networks and constraints while abstracting out details that you are not interested in viewing. This is accomplished with a set of abstraction controls based on the marked blocks.

The Abstract Clock Graph window is block aware. You can abstract clocks based on the marked blocks, which provides you with more control. First, mark the blocks of interest to abstract. After marking certain blocks or hierarchy elements, abstract the clock graph by using these marked blocks. If no blocks are marked, the Abstract Clock Graph window

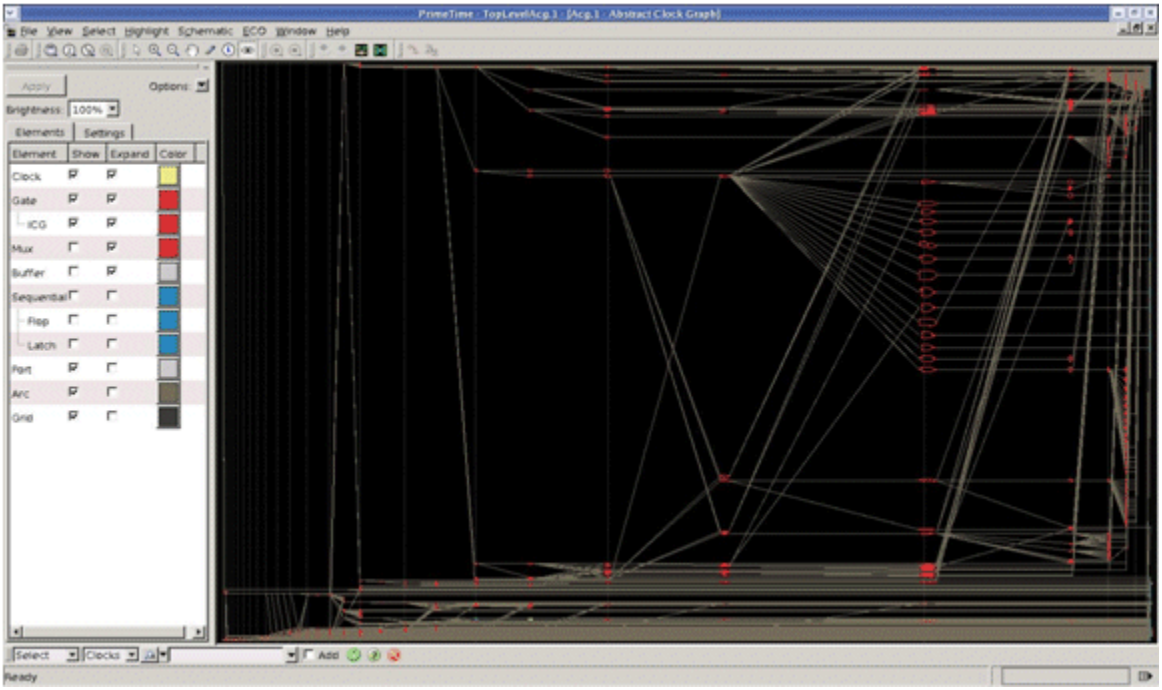
automatically assumes that all blocks are blocks of interest, and PrimeTime tries to abstract the clock graph with the highest level of hierarchy abstraction possible. You can focus on understanding the clock relationships and constraints between IP blocks during early assembly of a design. For more information about marking blocks and categorizing timing paths, see “Path Analyzer Window” on page 31.

Magnification Capabilities

PrimeTime enables magnification of the clock graph information using a fisheye tool. The fisheye tool of the clock graph enables you to view both local detail and global context simultaneously. It expands the area of interest and displays it with detail while also showing the remainder of the graph, but in less detail. From the Abstract Clock Graph, choose View > Mouse Tools > Fisheye Tool to access the fisheye tool. You can also click the eye () icon available from the toolbar.

Figure 4-48 shows the fisheye tool. As you move the mouse over the clock graph, the parts of the graph under the mouse increase in size. This enables you to inspect a large clock graph or path within context.

Figure 4-48 Fisheye Tool



Strokes Menu Available Through the Mouse

Access the strokes menu by pressing the middle mouse button until the menu appears. Drag the mouse in the direction of the action you want to perform. Table 4-3 shows the available options.

Table 4-3 Strokes Menu Options

Option	Description
Zoom_Fit_All	Fits the clock graph in the window.
Zoom_Out	Zooms out the clock graph out.
Zoom_In	Zooms in to obtain more detailed information.
Back	Undoes an operation. Equivalent to choosing Schematics > Back.
Forward	Redoes an operation. Equivalent to choosing Schematics > Forward.
Prim_Clks	Views the primary clocks.
All_Clks	Views all clocks.
Gen_Clks	Views the generated clocks.

Query Tool

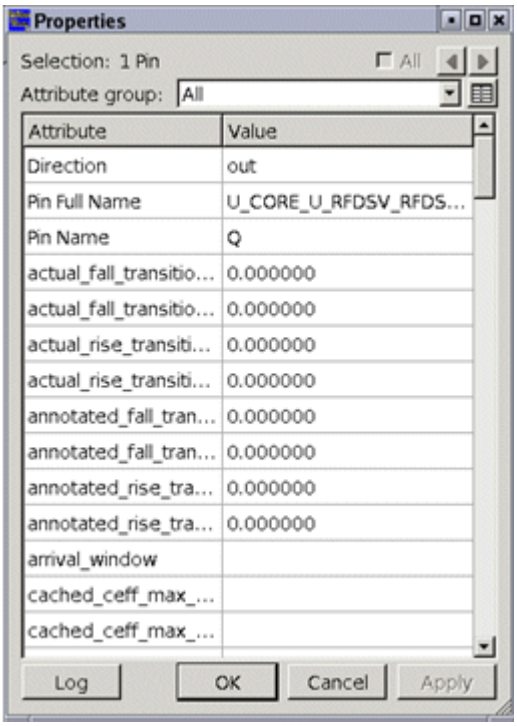
The Abstract Clock Graph window includes all object types available through the schematics such as buffers, inverters, integrated clock gating cells, different types of gates, as well as user-defined name and pattern matched types. You can access the query tool by choosing View > Mouse Tools > Query Tool or selecting the Query Tool icon. The query tool provides library cell names, MUX control data, all of the pin information, abstract object names, and object types for an Abstract Clock Graph object. Since this is an abstract view, the size of the data is intentionally limited in cases where there is an overwhelmingly large amount of data.

Creating New Attribute Groups

The Properties dialog box allows you to view the properties of the selected objects. In addition, you can use the Properties pane to view and edit the attribute group defined in PrimeTime. The attribute group allows you to specify a subset of the available attributes for

a group of objects. By default, PrimeTime provides the All, Timing, and Basic attribute groups. You can create custom attribute groups or redefine existing attribute groups. The Properties pane, shown in Figure 4-49, is accessible by choosing View > Property.

Figure 4-49 Properties Dialog Box

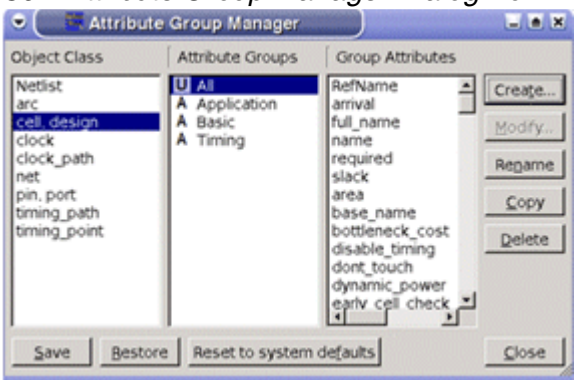


To create a new attribute group based on an object type,

1. Click the Activate Attribute Group Manager icon ( ) that appears next to the Attribute group list.

   The Attribute Group Manager dialog box appears as shown in Figure 4-50.

Figure 4-50 Attribute Group Manager Dialog Box



2. Select the object class that you want to create a new attribute group for and then click Create.
3. Type the name of the attribute group and then click OK.

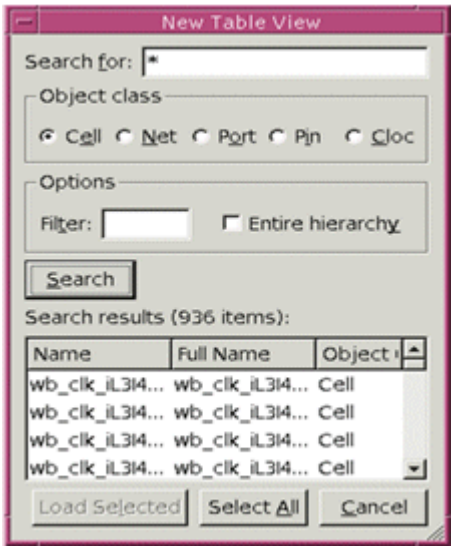
   The Attribute Group dialog box appears. The dialog box that appears depends on the object class that you selected.
4. Use the arrows to select the attributes for the new object class and then click OK.
5. Select the object class, attribute groups, and group attributes from the available list.
6. Click Save.
7. Click Close to exit the Attribute Group Manager dialog box.

   The new attribute group appears in the Attribute group list in the Properties pane.

Attribute Data Tables

The GUI makes it easy to extract attribute data from the design and to display the objects and attributes in a data table. In the main GUI window, choose Design > New Table View. This opens the New Table View dialog box. See Figure 4-51.

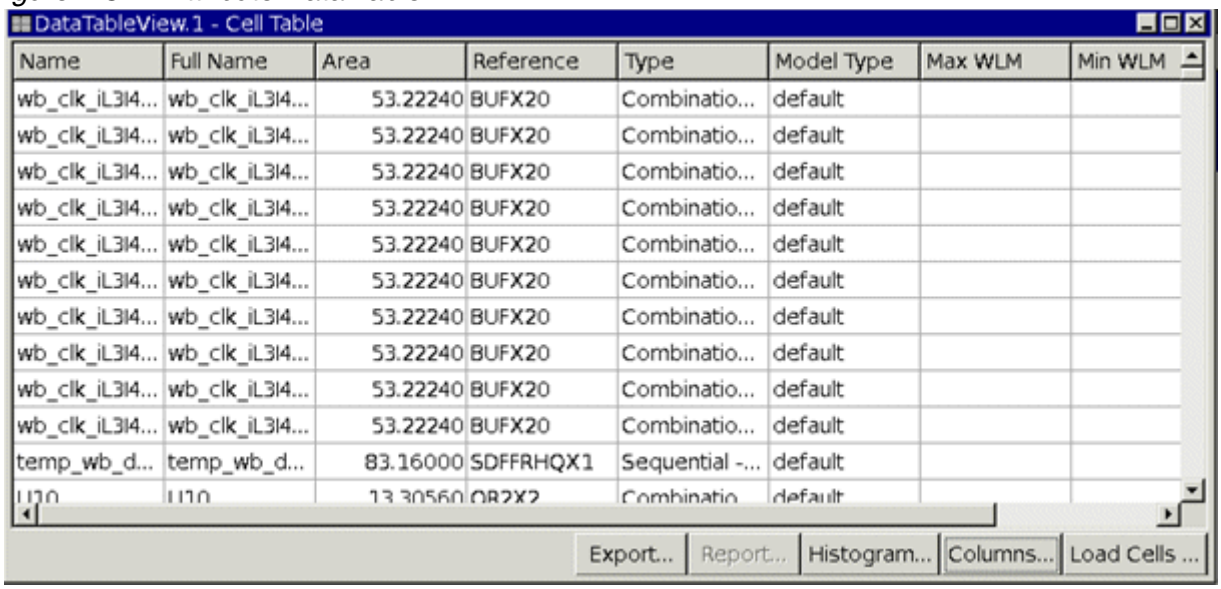
Figure 4-51 New Table View Dialog Box



In the dialog box, choose the attribute class of the objects you want to search for: Cell, Net, Port, Pin, or Clock. You can use the Search For and Filter fields to restrict the scope of the search by name and then click Search. This fills the list box with all the objects in the design that match the search criteria.

From this intermediate list, select the objects that you want to appear in the final data table. To select all of the objects in the list, click Select All. After you make your selection, click Load Selected. This loads the attribute data table with the selected objects. See Figure 4-52.

Figure 4-52 Attribute Data Table



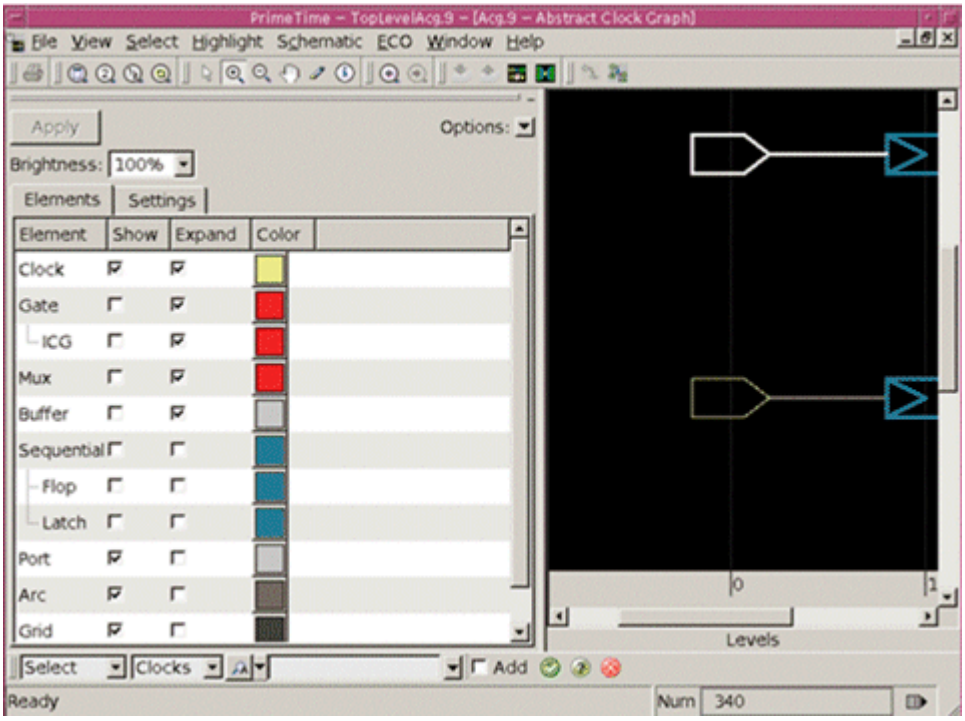
The attribute table has many of the same features as the timing path table in the main GUI window. For example, to sort the table according to an attribute, click on the header of the attribute column. To load a new set of objects into the table, click the Load Nets button (or Load Cells, Load Ports, and so on). To add or remove columns, click Columns. From the Show and Order columns dialog box, move the columns that you want to display from the Hidden columns area to the Visible columns area. You can also control the order the columns appear in the Data Tables view by using the up and down arrows to rearrange the order, from top to bottom in descending order.

View Settings Window

You can view and modify settings for the Clock Analyzer, Abstract Clock Graph, and Schematic windows. To display the View Settings window as shown in Figure 4-48, choose View > Toolbars > View Settings. Select an element to show or expand and then click Apply.

The view settings that are available depend on whether you are in the Clock Analyzer, Abstract Clock Graph, or Schematic window. Select an element to show or expand and then click Apply.

Figure 4-53 View Settings Window



Although the settings might be different, the features available in the View Settings window are the same across different views, such as the schematics view or the Abstract Clock Graph window. The following choices are available from the Options box:

- Preferences > Save to preferences

  Saves the view settings across sessions.

- Preferences > Set from preferences

  Applies previously saved preference settings to an existing view.

- Write settings script

  Writes a Tcl file, other than a preferences file, that you can source to get back the settings. This is helpful if you plan to modify settings to see what are the results or if you plan to send the settings to another user.

- Auto apply

  The changes are implemented and takes effect immediately. Otherwise, the settings are implemented after clicking Apply.

- Cancel changes

  Undoes any changes that are made before clicking Apply.

- Defaults

  Resets the view settings to the options available before you customized them. If this is the first time you are running the tool, the default settings are displayed.

Abstraction Control

The Abstract Clock Graph window provides you with local, fine-grained control over abstraction. For selected edges and nodes, you can abstract or expand objects within the same clock graph that are not currently in view by using the Expand column of the View Settings. Choose View > Toolbars > View Settings and then use the Expand column to select the elements you want to expand. You can also expand an object by right-clicking and selecting “Expand selected.” To collapse an object, deselect it from the Expand column or right-click and select “Collapse selected.”

You can incrementally view objects, obtaining more detailed information as the traversal occurs by double-clicking an object. Selective abstraction is also useful when attempting to locate objects by name or pattern. This feature helps with reconvergence analysis by providing reconvergence points and the surrounding details. From the View Settings window, select the objects that you want to expand in the Expand column, and click Apply.

Menu Commands

Table 4-4 lists and briefly describes the commands available in the menu bar of the top-level window.

Table 4-4 Menu Commands

Menu command	Action taken
File > Restore Session	Retrieves a previously saved session.
File > Save Session	Saves an existing session, including existing contents, used libraries, and noise data.
File > Execute Script	Executes a script just as when you execute a script from the source command.

Table 4-4 Menu Commands (Continued)

Menu command	Action taken
File > Save Screenshot As	Enables you to select the format and specify the file to save. A check box is available to enable you to specify whether you want a snapshot of the current view or the top-level window. The default is to take a snapshot of the current view, and save the file in the .png format.
File > Close GUI	Closes all PrimeTime GUI windows, leaving pt_shell active.
File > Exit	Exits completely from PrimeTime.
View > Preferences	Opens the Preferences dialog box.
View > Property	Opens the Properties dialog box.
View > Toolbars >	Controls the display of the available toolbars.
Select > Clear	Clears (deselects) the current selection.
Select > By Name	Selects cells, nets, ports, or pin by name, based on matching a specified pattern.
Select > Highlighted	Selects the highlighted object.
Select > Paths From/Through/ To	Selects paths based on from-to-through criteria or slack criteria.
Select > Fanin/Fanout	Selects objects in the fanin to, or fanout from, specified objects.
Select > Driver of Selected	Selects the driver of the selected objects.
Select > Load of Selected	Selects the load of the selected objects.
Select > Cells >	Selects cells based on their location in the hierarchy or pin/net connections.
Select > Ports/Pins >	Selects ports and pins based on their type or connections.
Select > Nets >	Selects nets based on their connections to ports/pins or cells.
Select > Clocks >	Selects clocks based on those that reach selected ports, pins, or cells.
Select > Selection List Ctrl + L	Displays a list box that describes the currently selected objects. The list box is updated automatically as you select new objects.

Table 4-4 Menu Commands (Continued)

Menu command	Action taken
Select > Query Selection	Generates a summary report on the selected objects.
Design > New Hierarchy Browser View	Displays a Hierarchy Browser window, from which you can select objects by traversing the hierarchy.
Design > New Table View for Selection	Displays a data table showing the characteristics of the currently selected objects. The selected objects must all be the same type of object (for example, all cells).
Design > New Table View	Displays a data table for a list of cells, nets, ports, pins, or clocks. You specify the object class and name-filtering criteria in a dialog box.
Design > Report>	Generates a text-format report on constraints, designs, cells, ports, nets, clocks, or wire load models in the design.
Timing > New Path Analyzer	Displays the Path Analyzer window if it is not already being displayed. This window consists of the design status view and the timing path table.
Timing > New Timing Path Table View for Selected	Displays a new timing path table. You specify the path criteria (`report_timing` options) in a dialog box.
Timing > New Timing Path Table View	Displays a new timing path table. You specify the path criteria (`report_timing` options) in a dialog box.
Timing > Inspector Normal Path	Displays the Path Inspector window for the currently selected path. The path inspector shows a wide range of information about the path, including a schematic, a path element table, a delay profile, clocking information, and timing waveforms.
Timing > Inspector Recalculated Path	Displays the Path Inspector window for the recalculated path. The path inspector shows a wide range of information about the path, including a schematic, a path element table, a delay profile, clocking information, and timing waveforms.
Timing > Histogram	Displays a histogram of design data: endpoint slack, path slack, net capacitance, design rules, or timing bottleneck.
Timing > New Recalculated Path Comparison Table	Displays a table comparing original and recalculated paths.
Timing > New Recalculated Path Pin Comparison Table	Displays a table showing path through pin value differences between the original and recalculated path.

Table 4-4 Menu Commands (Continued)

Menu command	Action taken
Timing > Net Capacitance Profile	Displays the relative amounts of net capacitance and pin capacitance for the currently selected net.
Timing > Check Timing	Displays a text-format report on the timing constraints, like the `check_timing` command.
Timing > Analysis Coverage	Displays a text-format report on the timing check coverage, like the `report_analysis_coverage` command.
Timing > Report Timing	Displays a text-format timing report, like the `report_timing` command.
Timing > Report Cell Timing	Displays a text-format cell timing report.
Timing > Clock Timing	Displays a text-format clock timing report, like the `report_clock_timing` command.
Timing > Update Timing	Updates the timing for the design, like the `update_timing` command.
Crosstalk > Delta Delay Histogram	Displays a histogram of crosstalk delay data.
Crosstalk > Path Delta Delay Histogram	Displays a histogram of crosstalkpath delay data.
Crosstalk > Bump Voltage Histogram	Displays a histogram of crosstalkbump voltage.
Crosstalk > Accumulated Bump Voltage Histogram	Displays a histogram of crosstalk accumulated bump voltage.
Crosstalk > Coupling Analysis For Selected Paths	Populates the coupling analysis table with the information for the “n” worst path, based on all the SI delays in the selected paths.
Crosstalk > Coupling Analysis For Selected Nets	Populates the coupling analysis table with the information for the selected nets.
Crosstalk> Coupling Analysis For SI Bottleneck Nets	Populates the SI victim analysis table with the information for the bottleneck nets (this is analogous to running the `report_si_bottleneck` command).
Noise > Noise Slack Histogram	Displays a histogram of noise slack.

Table 4-4 Menu Commands (Continued)

Menu command	Action taken
Noise > Noise Bump Histogram	Displays a histogram of noise bump.
Noise > Accumulated Noise Bump Histogram	Displays a histogram of accumulated noise bump.
Noise > Noise Immunity Curve	Displays a plot of the noise immunity characteristics of the cell at the currently selected input pin, as specified by the noise immunity table in the library definition of the cell.
Power > Show Power Analysis Driver	Displays the analysis driver so that you can analyze dynamic power and static power. You view total power consumption for the current design with its distribution between dynamic, switching, internal, and leakage consumption.
Power > Show Leakage Variation Data	Displays the Leakage Variation Analysis dialog box so that you can analyze the data.
Power > Net Power Design Map >	Displays the total power density as a map of the design. The power density displays as a tree map that is a visualization tool. You can use this tool to help you identify anomalies in large hierarchical data sets.
Power > Histogram of selected >	Use to browse the design hierarchy. You can view the cell hierarchical, pin, port, pin of child cell, net, and clock attributes.
Power > View Waveforms	Displays your switching activity in a waveform format.
Clock > Analyzer	Opens the Clock Analyzer window that provides a central place to run various clock analysis tasks.
Clock > Clock Graph for All Clocks	Opens the Abstract Clock Graph window that helps you identify the relationships between different clocks, such as primary and generated, clock convergence points, classes of clocked elements, and logic levels.
Clock > Schematic for Selected Clocks	Opens a new Schematic window showing the selected clocks.
Clock > Schematic for Unclocked Pins	Opens a new Schematic window showing all unclocked pins in the design.
Schematic > New Design Schematic View	Opens a new Schematic window showing the contents of the currently selected cell.

Table 4-4 Menu Commands (Continued)

Menu command	Action taken
Schematic > New Path Schematic View	Opens a new schematic showing the currently selected path.
Schematic > New Abstract Path Schematic View	Opens a new schematic showing the currently selected abstract path.
ECO > Insert Buffer	Inserts a buffer at the selected port or pin.
ECO > Insert Buffer at Net Driver	Inserts a buffer at the drivers for the selected nets.
ECO > Size Cell	Resizes the selected cell.
ECO > Size Driver	Resizes the drivers of selected nets.
ECO > Upsize Driver	Upsizes the drivers of the selected nets.
ECO > Downsize Driver	Downsizes the drivers of the selected nets.
ECO > Set Coupling Separation	Sets global coupling separation for selected nets.
ECO > Set Pairwise Coupling Separation	Sets pairwise coupling separation for the selected nets. This is only available for PrimeTime SI flows.
Window > New Main Window	Opens a new (additional) top-level main window.
Window > New Console	Opens a new console window for running pt_shell commands, viewing the command history, and viewing errors and warnings.
Window > Close Window	Closes the currently active window within the top-level window. If this is the last window, it closes the GUI and exits to the pt_shell prompt.
Window > Close All Windows (Close GUI)	Closes all GUI windows, and exit to the pt_shell prompt.
Window > Next Window	Cycles through multiple top-level windows to the next active window and brings it to the front.
Window > Previous Window	Cycles to the previous active window and brings it to the front.
Window > Close View	Closes the active view within the current window.
Window > Close All View	Closes all of the views within the current window.

Table 4-4 Menu Commands (Continued)

Menu command	Action taken
Window > Tile	Arranges all windows side by side within the top-level window.
Window > Cascade	Cascades all windows into a stack of windows within the top-level window.
Window > Next View	Cycles to the next active view within the top-level window and brings it to the front.
Window > Previous View	Cycles to the previous active view within the top-level window and brings it to the front.
Window > *window name*	Makes the named window active and brings it to the front.
Help > Man Pages	Opens a window for displaying man pages.
Help > Report Hotkey Bindings	Opens the Hotkeys dialog box, which provides a list of key shortcuts that are available.
Help > Getting Help	Provides an overview of the different resources available, such as user guides, release notes, and SolvNet articles.
Help > About	Shows the PrimeTime version number and copyright notice.

5

Design Data

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

Reading, reporting, and modifying design data are described in the following sections:

Search Path and Link Path

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

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

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

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

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

1. Set the search_path variable with the set command. The value should be a list of paths to the directories containing the design files, timing models files, and library files needed for the analysis. Delimit each path with a space and enclose the list in double quotation marks (" "). For example,

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

2. Set the link_path variable with the set command. The value should be a list of the library names. Use an asterisk (*) to make PrimeTime search for designs in memory. Delimit each element with a space and enclose the list in double quotation marks (" ").

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

Reading Design and Library Data

To analyze a design, you need to read the design data and associated technology libraries into PrimeTime. You can use the `read_ddc` command to read design information, including netlists and constraints for a design, from .ddc databases produced by other tools, including Design Compiler and IC Compiler. Otherwise, you can use the `read_db` command to read the technology libraries and design data in .db format. For information about reading design data in other formats, see "Reading Verilog and VHDL Design Data" on page 5-5.

Reading Design Data in .ddc Format

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

1. Set the search path. For example,

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

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

2. Set the link path (optional). For example,

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

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

3. Read the technology library into memory in the .db format. For example, enter

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

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

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

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

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

```
pt_shell> link_design TOP
```

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

Reading Design Data in .db Format

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

1. Set the search path. For example, enter

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

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

2. Set the link path. For example, enter

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

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

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

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

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

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

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

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

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

   ```
   pt_shell> link_design TOP
   ```

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

case sensitivity when reading in source files from case-sensitive formats might cause inconsistent and unexpected results. For more information about how to use the `link_force_case` variable, see the man page.

Reading Verilog and VHDL Design Data

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

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

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

Using the PrimeTime Verilog Reader

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

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

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

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

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

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

You turn translation back on using one of these directives:

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

Optional Preprocessor

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

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

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

Assign Statements and Synonyms

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

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

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

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

You cannot use wildcards with synonyms:

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

Physical Verilog Power Rail Connections

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

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

PrimeTime Verilog Reader Limitations

The following limitations apply to the native Verilog reader:

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

Using the HDL Compiler Verilog Reader

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

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

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

Note:

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

Setup Files for the HDL Compiler Verilog Reader

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

You can define the `ptxr_setup_file` variable in PrimeTime to reference a ptxr-specific setup file. By default, this variable does not exist. The ptxr setup file is written in Tcl. This file can contain a limited set of commands:

- Comments
- Blank lines
- Variable assignments

For example, you could create `my_ptxr.setup`:

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

To use this file:

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

To discontinue using the ptxr setup file:

```
pt_shell> unset ptxr_setup_file
```

Note:

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

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

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

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

Reading VHDL Design Files

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

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

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

Setup Files for the VHDL Compiler

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

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

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

To use the ptxr setup file:

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

To discontinue using the ptxr setup file:

```
pt_shell> unset ptxr_setup_file
```

Using a Milkyway Database

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

Reading a Milkyway Database

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

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

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

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

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

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

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

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

Writing a Milkyway Database

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

Limitations When Reading Milkyway Format

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

Note:

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

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

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

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

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

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

Removing Designs and Libraries

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

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

Setting the Current Design and Current Instance

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

If you specify a design, PrimeTime sets the current design to that design. This argument can be the name of a design, or a collection containing one design. If you do not specify a design, PrimeTime returns a collection containing the current design. You can use the `current_design` command as a parameter to other PrimeTime commands. To display designs in PrimeTime memory, use the `list_designs` command.

To see the current context and to change the context from one design to another, use a command sequence like the following:

```
pt_shell> current_design
{"TOP"}

pt_shell> list_designs

Design Registry:
      ADDER             /designs/dbs/my_design.ddc:ADDER
      FULL_ADDER        /designs/dbs/my_design.ddc:FULL_ADDER
      FULL_SUBTRACTOR   /designs/dbs/
my_design.ddc:FULL_SUBTRACTOR
      HALF_ADDER        /designs/dbs/my_design.ddc:HALF_ADDER
      HALF_SUBTRACTOR   /designs/dbs/
my_design.ddc:HALF_SUBTRACTOR
      SUBTRACTOR        /designs/dbs/my_design.ddc:SUBTRACTOR
*     TOP               /designs/dbs/my_design.ddc:TOP

pt_shell> current_design ADDER
{"ADDER"}
```

The `current_instance` command sets the working instance object (cell in the design hierarchy) and enables other commands to be used relative to the instance. This command differs from the `current_design` command, which changes the working design, then sets the current instance to the top level of the new current design. Combining the current design and current instance defines the context for many PrimeTime commands.

The `current_instance` command traverses the design hierarchy similarly to the way the UNIX cd command traverses the file hierarchy. This command operates with a variety of *`instance_name`* arguments:

- If you do not specify an *`instance_name`* argument, the focus of PrimeTime returns to the top level of the hierarchy.
- If you specify dot (`.`*`instance_name`*) as the argument, PrimeTime returns the current instance and makes no change.
- If you specify dot dot (`..`) as the *`instance_name`* argument, PrimeTime moves the current instance up one level in the design hierarchy.
- If you specify an *instance_name* argument that begins with slash (`/`), PrimeTime sets both the current design and the current instance.
- If you specify a valid cell at the current level of hierarchy as the *instance_name* argument, PrimeTime moves the current instance down to that level of the design hierarchy.

- If you specify multiple cell names separated by slashes, PrimeTime traverses multiple levels of hierarchy in a single call to the current instance. For example, `current_instance U1/U2` sets the current instance down two levels of hierarchy.
- You can nest the dot dot (. .) directive in complex `instance_name` arguments. For example, the command

  ```
  current_instance ../../MY_INST
  ```

  attempts to move the context up two levels of hierarchy, then down one level to the `MY_INST` cell.

Examples

To use the `current_instance` command to move up and down the design hierarchy, use a command sequence like the following. The `all_instances` command lists instances of a specified design relative to the current instance.

```
pt_shell> current_design TOP
{"TOP"}

pt_shell> current_instance U1
U1

pt_shell> current_instance "."
U1

pt_shell> query_objects [all_instances ADDER]
{"U1", "U2", "U3", "U4"}

pt_shell> current_instance U3
U1/U3

pt_shell> current_instance "../U4"
U1/U4

pt_shell> current_instance
Current instance is the top-level of design 'TOP'.
```

To use `current_design` to reset the current instance to the top level of the new design hierarchy, use a command sequence like the following:

```
pt_shell> current_design ADDER
{"ADDER"}

pt_shell> current_instance .
Current instance is the top-level of design 'ADDER'.
```

To use the `current_instance` command to go to an instance of another design, use a command sequence like the following. The new design whose name is the name after the first slash of the given instance name sets the current design.

```
pt_shell> current_instance "/ALARM_BLOCK/U6"
U6
pt_shell> current_design
{"ALARM_BLOCK"}
```

Listing Design and Library Information

PrimeTime provides commands for listing designs and libraries that are loaded. The `list_designs` command lists the designs that are in PrimeTime memory, including the current design, linked or partially linked designs, and designs instantiated in a linked design.

The notation for the status of the design is as follows:

- * – Design is the current design.
- `L` – Design is linked.
- `N` – Design is not in memory.
- `l` – Design is partially linked.

Examples

To list the designs in memory, enter

```
pt_shell> list_designs
Design Registry:
      AD4FULA      /u/designs/top.db:AD4FULA
      AD4PG        /u/designs/top.db:AD4PG
      ADD5A        /u/designs/add2.db:ADD5A
      MULTI        /u/designs/top.db:MULTI
```

To list the designs in use and in memory, enter

```
pt_shell> list_designs -only_used
Design Registry:
 *L AD4FULA        /u/designs/top.db:AD4FULA
    ADD5A          /u/designs/add2.db:ADD5A
    MULT1          /u/designs/top.db:MULT1
```

After removing the `MULTI` design, you can display it using the `-all` option. For example, enter

```
pt_shell> remove_design MULT1
Removing design 'MULT1'...
1
pt_shell> list_designs -all -only_used
Design Registry:
 *L AD4FULA        /u/designs/top.db:AD4FULA
    ADD5A          /u/designs/add2.db:ADD5A
  N MULT1          /u/designs/top.db:MULT1
```

The `list_libs` command lists the libraries in memory.

You can use the `-only_used` option to filter unused libraries out of the display. A library is in use if a linked design links to library cells from the library.

The status of the library is reported as a single character:

- "*" (asterisk) indicates the main library of the current design; the first library in the link path that has a time unit.
- "M" indicates the maximum library of a max-min pair (see the man page for the `set_min_library` command).
- "m" indicates the minimum library of a max-min pair (see the man page for the `set_min_library` command).

Examples

To list all the libraries in memory, enter

```
pt_shell> list_libs
Library Registry:
  my_lib        /u/lib/my_lib.db:my_lib
  tech1         /u/lib/tech1.db:tech1
```

After you link a design, you can identify the main library using the `-only_used` option. This option limits the display to the libraries used to link the current design. For example, enter

```
pt_shell> link_design top_flat
Linking design top_flat...
Design 'top_flat' was successfully linked.
1

pt_shell> list_libs -only_used

Library Registry:
 *  tech1         /u/lib/tech1.db:tech1
```

Linking the Design

To produce a design that is ready for timing analysis, link your design using the `link_design` command. Linking the design resolves references to library cells and subdesigns.

To link a design called `example`, enter

```
pt_shell> link_design example
Loading db file '/unit/libraries/library.db'
Loading db file '/unit/designs/A.db'
Loading db file '/unit/designs/B.db'

Design 'example' was successfully linked.
1
```

If you have not read all the subdesign or library files into memory, PrimeTime attempts to load them automatically during linking. This process is known as an autoload, which is dependent on the values of the `search_path` and `link_path` variables.

By default, the linker automatically creates black boxes for unresolved references. A black box is essentially an empty cell with no timing arcs. The result is a completely linked design on which analysis can be performed. You can disable automatic black box creation by setting the `link_create_black_boxes` variable to `false`.

Many commands, such as the `report_timing` command, require a linked design and attempt to automatically link the design if you have not done an explicit link. This process is known as an autolink.

Per-Instance Link Paths

You can have PrimeTime use different library cells for different instances of the same cell in the design. To do so, set the `link_path_per_instance` variable to a list, with each list element consisting of a list of instances and the corresponding link path to use for those instances. For example,

```
set link_path {* lib1.db}
set link_path_per_instance [list
    [list {ucore} {* lib2.db}]
    [list {ucore/usubblk} {* lib3.db}]]
```

Specified entries are used to link the specified level and below. If a given block matches multiple entries in the per-instance list, the more specific entry overrides the more general entry. In the example above:

- lib3.db is used to link blocks “ucore/usubblk” and below.
- lib2.db is used to link “ucore” and below (except within “ucore/subblk”).
- lib1.db is used for the remainder of the design (everything except within “ucore”).

The default value of `link_path_per_instance` is an empty list, meaning that the feature is disabled. To determine the current value of this variable, use `set link_path` or `echo $link_path`.

For more information, see the variable man page.

Handling Incomplete Data

During early design development, design data needs to be updated regularly. For example, a block recently updated by a block-level designer might have fewer pins than the same block that was used by a top-level designer.

You can continue working on the design and gathering useful information even in the presence of incomplete or “dirty” data. The `link_allow_design_mismatch` variable controls whether a design is successfully linked in the presence of dirty data. By default, the variable is set to `false`, and the design fails to link when mismatches exist. When you set this variable to `true`, mismatches between an instance and reference are ignored, the design links successfully, and you can continue working on the design.

The `is_design_mismatch` attribute is available on each cell, pin, and net. If an object is dirty, that is, the object is directly impacted by a mismatch between the block and top-level design, the `is_design_mismatch` attribute of the object is `true`. Use the `list_attributes` command to list the currently defined attributes.

The following information explains how PrimeTime handles dirty data and also explains what actions are required, if any:

- Top-level design has pins that are not available in subdesigns

  If the higher-level hierarchies have pins that do not exist at the lower level, these extra pins are ignored and do not exist in the linked netlist. Nets connected to missing pins are disconnected and left dangling. Annotations, such as constraints and parasitics, on these pins are not applied. The LNK-038 warning message is issued for each pin that is ignored.

- Library and netlist cell contains different power and ground (PG) information

  If the library and netlist cell have different PG information, the information in the library cell is assumed to be accurate. If the library cell does not have PG information, neither does the linked netlist.

- Direction of pins differ depending on the hierarchical level

  If pins at different hierarchy levels have different directions, the direction of the pins of the lower-level cells in the design is used.

- Pin names are case-sensitive

  The case of pin names at different hierarchical levels might not match. For example, a pin might be named UBLK/CLK at the top level and UBLK/clk at the block level. If the existing `link_force_case` variable is set to `case_insensitive`, the case of the pin is ignored and is not considered dirty. The default of the `link_force_case` variable is `check_reference`, which means the case sensitivity of the link is determined by the input format that was used to create the reference. For more information about how to use this variable, see the man page.

- Bus width causes mismatch

  If the width of a bus varies for different hierarchical blocks, the bits of the bus are connected beginning with the least significant bits. If the lower-level block has additional bus bits not present in the next higher level, the bits are unconnected. If the higher-level block has additional bits that are not present in the lower-level block, the pins are ignored, and the LNK-038 warning message is issued for each ignored pin.

To view the mismatches located while linking a design, use the `report_design_mismatch` command. For example,

```
pt_shell> report_design_mismatch
report_design_mismatch
****************************************
Report : report_design_mismatch
Design : top
Version: E-2010.12
Date   : Mon Sep  30 11:08:33 2010
****************************************

pin                          mismatch type
--------------------------------------------------------------------------------
u1/Z                         Pin direction mismatch
u2/Z                         Pin direction mismatch
--------------------------------------------------------------------------------
cell                         mismatch type
--------------------------------------------------------------------------------
u1                           Pin direction mismatch
u2                           Pin direction mismatch
--------------------------------------------------------------------------------
```

```
net                           mismatch type
-------------------------------------------------------------------------
b                             Pin direction mismatch
```

Link Errors

Certain link errors can cause the linking of the design to fail. When the `link_design` command fails, the output is a design with unresolved references.

There are two ways for the linker to fail to resolve a reference:

- The reference cannot be found. In this case, a black box is created when `link_create_black_boxes` is `true`.
- The reference was found, but the linker detected errors in trying to resolve the reference. In this case, a black box is not created, and the link fails.

There are various errors that can cause the link to fail to resolve a reference. The most typical errors are

- The instance has too many ports. An example is an instance of BOX that has 6 ports, but the reference has only 5.
- The instance has a pin that does not exist on the reference (library cell or design). A example would be an instance of a BOX with pins A, B, and X, where the reference has ports A, B, C, and Z. Generally, the reference can have more ports than the instance. This can be dependent on the source of the netlist. For example, in Verilog, the following instance of the previously described BOX would be fine:

  ```
  BOX u1 (.A(n1), .Z(out));
  ```

- The instance has scalar pins like `A[0]` and `A[1]`, but the reference has a bus A. Sometimes this indicates a library or subdesign that is out of sync with the top design, which might have been flattened.

Using information from the error messages, you can construct Tcl procedures to display the name and direction of each port on a design or library cell, and each pin on an instance. With these displays, you should be able to detect the problem and determine if the instance or the reference needs to be corrected.

Using the set_units Command for Scaling Units

The `set_units` command explicitly checks that the units specified are the same as the main library unit values. It can appear in Synopsys Design Constraints (SDC) files, on the command line, or through script files; its use is optional. By checking for consistency of the specified units against the current main library units, this command helps to identify the inadvertent use of inconsistent units from different constraint files.

The specified units are checked and if they are different from the main library units, a warning message is generated. If the units are the same (that is, acceptable), no warning message is generated.

The `write_sdc` command writes out the `set_units` command as the first command. The output of reporting commands is unaffected by the units specified with the `set_units` command. Reports continue to use the units defined in the main library.

An example of using the `set_units` command is as follows:

```
set_units -time 1ns -capacitance 1nF -current 1mA -voltage 1V -resistance
1kOhm -power 1W
set_units -time 1ns -capacitance 0.5pF
```

Checking the Syntax of the SDC File

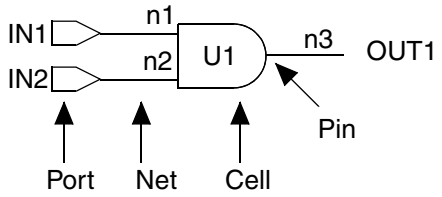
You can check the SDC file and verify that it is syntactically and semantically correct by using the `read_sdc -syntax_only` command. This validates the script without having any effect on the design. For more information about SDC, see the *Using the Synopsys Design Constraints Format Application Note*. To locate this application note, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

Design Objects

ASIC designs are hierarchical entities composed of objects such as cells, ports, and nets (see Figure 5-1). To perform detailed timing analysis and locate the source of timing problems, you need access to these objects. PrimeTime lets you create a collection of one or more objects for querying, reporting, or creating timing assertions.

Figure 5-1 Typical ASIC Design Objects



Most PrimeTime commands operate on a collection of objects. To create a collection of objects, use the appropriate “get” command: `get_ports`, `get_nets`, `get_clocks`, and so on. The result of a “get” command can be nested within another command that operates on the objects. For example,

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

State Objects

PrimeTime uses these state objects:

Current design

The design that is the current top of the hierarchy. Most objects are referenced relative to the current design. The `current_design` command sets the working design for many PrimeTime commands.

Current instance

The instance (or hierarchical cell) that is the current scope within the current design. Traverse the hierarchy by changing the current instance. The `current_instance` command sets the working instance object (cell) and enables other commands to be used relative to a specific instance in the design hierarchy.

Netlist Objects

PrimeTime commands, attributes, and assertions are directed toward a netlist object. You can create collections of netlist objects. PrimeTime uses these netlist objects:

`cell`

The instances in the design, including those that reference hierarchical blocks and primitive library cells.

`clock`

Clock objects do not exist in a design by default. They are created when you define the clocks for PrimeTime. Clocks are also imported from .ddc or .db format files.

`design`

A design.

`lib`

A library.

`lib_cell`

The cells in a technology library.

`lib_pin`

The pins on library cells.

`net`

The nets in the current design.

`path group`

Collections of paths considered as a group in the cost-function calculations that drive logic synthesis in Design Compiler. A group has information, such as a set of paths, a name, and weight. The weight allows you to control the way the maximum delay cost is computed when you optimize designs in Design Compiler. Timing reports are also organized by path group.

`pin`

The pins of lower-level cells in the design. Pins can be input, output, or inout (bidirectional).

`port`

The ports of the current design. Ports can be input, output, or inout (bidirectional).

`register`

The primitive instances of a design that have clocks defined on them. These instances include flip-flops and latches.

6

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 “Static Timing Analysis Overview” on page 1-9, which includes basic information about path startpoints and endpoints, delay calculation, setup and hold constraints, time borrowing, and timing exceptions.

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

Path Groups

PrimeTime organizes paths in the design into groups. This path grouping affects the generation of timing analysis reports. For example, by default the `report_timing` command reports the single path with the worst slack from each path group.

In Design Compiler, path grouping also affects design optimization. Each path group can be assigned a weight (also called cost function). The higher the weight, the more effort Design Compiler uses to optimize the paths in that group. You can assign weights to path groups in PrimeTime, but this weight information is not used in PrimeTime.

PrimeTime implicitly creates a path group each time you use the `create_clock` command to create a new clock. The name of the path group is the same as the clock name. PrimeTime assigns a path to that path group if the endpoint of the path is a flip-flop clocked by that clock. PrimeTime also creates the following path groups implicitly:

- **clock_gating_default**: paths that end on combinational elements used for clock gating
- **async_default**: paths that end on asynchronous preset/clear inputs of flip-flops
- **default**: constrained paths that do not fall into any of the other implicit categories. For example, a path that ends on an output port
- none: unconstrained paths

In addition to these implicit path groups, you can create your own user-defined path groups by using the `group_path` command. This command also lets you assign any particular path to a specific path group. To get information about the current set of path groups, use the `report_path_group` command. To remove a path group, use the `remove_path_group` command. Paths in that group are implicitly assigned to the default path group. For example, to place all paths to ports with names matching OUT_1* into their own group called out1bus, enter

```
pt_shell> group_path -name out1bus -to [get_ports OUT_1*]
```

Path Timing Reports

You can use path timing reports to focus on particular timing violations and to determine the cause of a violation. By default, the `report_timing` command reports the path with the worst slack for each path group based on maximum delay.

A path timing report provides detailed timing information about any number of requested paths. The level of detail you want to see in the output can vary. For example, you can view this information:

- Gate levels in the logic path
- Incremental delay value of each gate level
- Sum of the total path delays
- Amount of slack in the path
- Source and destination clock name, latency, and uncertainty
- OCV with clock reconvergence pessimism removed

The `report_timing` command provides options to control reporting of the following types of information:

- Number of paths
- Types of paths
- Amount of detail
- Startpoints, endpoints, and intermediate points along the path

The `-significant_digits` option of the `report_timing` command lets you specify the number of digits after the decimal point displayed for time values in the generated report. This option only controls the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

The `report_timing` command by itself, without any options, is the same as the following command:

```
pt_shell> report_timing -path_type full -delay_type max \
          -max_paths 1 -nworst 1
```

PrimeTime generates a maximum-delay (setup constraint) report on the full path, showing the single worst path per path group, and showing no more than one path per timing endpoint. For example,

```
pt_shell> report_timing

****************************************
Report : timing
Design : FP_SHR
****************************************

Operating Conditions:
Wire Loading Model Mode: top

  Startpoint: a (input port)
  Endpoint: c_d (output port)
  Path Group: default
  Path Type: max

  Point                              Incr       Path
-----------------------------------------------------
  input external delay              10.00      10.00 r
  a (in)                             0.00      10.00 r
  m1/Z (MUX21H)                      1.00      11.00 r
  u1/S (FA1)                         1.00      12.00 r
  c_d/Z (AN2)                        1.00      13.00 r
  c_d (out)                          0.00      13.00 r
  data arrival time                            13.00

  max_delay                         15.00      15.00
  output external delay            -10.00       5.00
  data required time                            5.00
-----------------------------------------------------
  data required time                            5.00
  data arrival time                           -13.00
-----------------------------------------------------
  slack (VIOLATED)                  -8.00
```

To show information about the four worst paths in each path group (rather than just one), enter

```
pt_shell> report_timing -max_paths 4
```

To show the transition time and capacitance, enter

```
pt_shell> report_timing -transition_time -capacitance
```

PrimeTime displays a report similar to the following one:

```
    ****************************************
    Report : timing
             -path full
             -delay max
             -max_paths 1
             -transition_time
             -capacitance
    Design : counter
    Version: 2000.11
    Date   : Wed Nov 15 09:51:19 2000
    ****************************************

  Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                                   Cap      Trans       Incr       Path
  ---------------------------------------------------------------------------
  clock CLK (rise edge)                                       0.00       0.00
  clock network delay (ideal)                                 0.00       0.00
  ffa/CLK (DTC10)                                 0.00        0.00       0.00 r
  ffa/Q (DTC10)                           3.85    0.57        1.70       1.70 f
  U7/Y (IV110)                            6.59    1.32        0.84       2.55 r
  U12/Y (NA310)                           8.87    2.47        2.04       4.58 f
  U17/Y (NA211)                           4.87    1.01        1.35       5.94 f
  U23/Y (IV120)                           2.59    0.51        0.37       6.30 r
  U15/Y (BF003)                           2.61    0.88        0.82       7.12 f
  U16/Y (BF003)                           2.61    1.46        0.99       8.11 r
  U10/Y (AN220)                           2.63    0.46        1.04       9.15 r
  ffd/D (DTN10)                                   0.46        0.00       9.15 r
  data arrival time                                                      9.15

 clock CLK (rise edge)                                       10.00      10.00
  clock network delay (ideal)                                 0.00      10.00
  ffd/CLK (DTN10)                                                       10.00 r
  library setup time                                         -1.33       8.67
  data required time                                                     8.67
  ---------------------------------------------------------------------------
  data required time                                                     8.67
  data arrival time                                                     -9.15
  ---------------------------------------------------------------------------
  slack (VIOLATED)                                                      -0.48
```

Reporting Timing Exceptions

The `report_timing` command has an `-exceptions` option that allows you to report the timing exceptions that apply to an individual path. You can choose to report the following:

- Exceptions that apply to a path
- Exceptions that were overridden by higher-priority exceptions
- Unconstrained path debugging that includes unconstrained startpoint and unconstrained endpoint
- Timing path attributes that show the unconstrained reasons

When using either the `report_timing -exceptions all` or `get_timing_paths` command to report why timing paths are unconstrained, you must first set the `timing_report_unconstrained_paths` variable to `true`. As shown below, to report the unconstrained reasons, you can use the `report_timing` command with the three options:

```
pt_shell> report_timing -exceptions dominant
pt_shell> report_timing -exceptions overridden
pt_shell> report_timing -exceptions all
```

In the following example where you specify conflicting exceptions, the maximum-delay exception has higher priority:

```
pt_shell> set_multicycle_path -through buf1/Z -setup 2
pt_shell> set_max_delay -through buf1/Z 1
```

If you use `report_timing -exceptions dominant` to generate a timing report on a path containing buf1/Z, the report includes a section showing the dominant timing exception that affected the path:

```
The dominant exceptions are:
From        Through        To     Setup          Hold
-------------------------------------------------------
*           buf1/Z         *      max=1
```

If you use `report_timing -exceptions overridden`, the report includes a section showing the overridden timing exception that had no effect on the timing calculation:

```
The overridden exceptions are:
From        Through        To     Setup          Hold
-------------------------------------------------------
*           buf1/Z         *      cycles=2       *
```

If you use `report_timing -exceptions all`, the timing report includes a section showing both the dominant and overridden timing exceptions. Alternatively, you can use the `get_timing_paths` command to create a collection of timing paths for custom reporting or

for other processing purposes. You can then pass this timing path collection to the `report_timing` command. By adding the `-exceptions all` argument, you obtain the additional debugging information. For example,

```
report_timing -from A -through B -to C -exceptions all
set x [get_timing_paths -from A -through B -to C]
report_timing $x -exceptions all
```

With the `get_timing_paths` command, you can access the path attributes that show the reasons why the path is unconstrained. Table 6-1 shows the timing path attributes along with their possible values, in relation to unconstrained paths:

Table 6-1 Timing Path Attributes and Values

Timing path attribute	Reason code	Reason code description
`dominant_exception`	false_path	False paths that are not considered.
	min_delay max_delay	Timing path is a minimum or maximum delay.
	multicycle_path	Multicycle path
`endpoint_unconstrained_reason`	no_capture_clock	No clock is capturing the data at the endpoint.
	dangling_end_point	Timing path is broken by a disabled timing component because it ends at a dangling (floating) point that has no timing constraints information.
	fanin_of_disabled	Path ending at a fanin of a disabled timing arc or component. The internal pin is either without constrains, pin connected to a black box, or there are unconnected pins. The endpoint of the timing path is part of the fanin of a disabled timing arc or component
`startpoint_unconstrained_reason`	no_launch_clock	No clock is launching the data from the startpoint.

Table 6-1 Timing Path Attributes and Values (Continued)

Timing path attribute	Reason code	Reason code description
	dangling_start_point	Path starting from a dangling pin. The timing path is broken by a disabled timing component because it starts from a dangling (floating) point that has no timing constraints information.
	fanout_of_disabled	Path starting from a fanout of a disabled timing arc or component. The internal pin is without constrains, pin connected to a black box, or there are unconnected pins. The startpoint of the timing path is part of the fanout of a disabled timing arc or component

The `report_timing -exceptions` command always shows three categories: the dominant exception, startpoint unconstrained reason, and endpoint unconstrained reason. If no exception information is present in a category, that section is empty.

The exception attributes on the `timing_path` objects (`dominant_exception`, `startpoint_unconstrained_reason`, and `endpoint_unconstrained_reason`) are only present on a path if information is present in that category. This makes it easy to filter for the presence or absence of exception information affecting a path. For example, to filter all paths from a collection that are affected by exceptions:

```
filter_collection $paths {defined(dominant_exception)}
filter_collection $paths {undefined(dominant_exception)}
```

The following report shows the output of `report_timing -exceptions all`:

```
pt_shell> report_timing -exceptions all
****************************************
Report : timing
        -path_type full_clock_expanded
.....
  (Path is unconstrained)

The dominant exceptions are:
From          To           Setup         Hold
-------------------------------------------------------------------
lr6_ff/CP     cr6_ff/D     FALSE         FALSE

The overridden exceptions are:
From          To           Setup         Hold
-------------------------------------------------------------------
F1/CP         F2/D         cycles=3      cycles=0
```

```
The unconstrained reasons (except for false path) are:
Reason                          Startpoint                Endpoint
-----------------------------------------------------------------
no_launch_clock                 F1/CP                       -
```

Reporting Exceptions Source File and Line Number Information

For some constraints, PrimeTime can track and report the source file and line number information for the constraints. Source files are read into the PrimeTime shell using the source or `read_sdc` commands or the `-f` command line option.

To enable the source file name and line number information for the current design constraints, you should set the `sdc_save_source_file_information` variable to `true`. By default, this variable is set to `false`.

Note:

You can modify the value of this variable only if you have not yet input exceptions.

This implies that you can set the variable to `true` in the setup file or inside pt_shell before applying a timing exception. If at least one exception command has already been successfully input when you try to set this variable, you receive an error and the value remains unchanged. For example,

```
pt_shell> echo $sdc_save_source_file_information
false
pt_shell> set_max_delay -to port1 0.7
pt_shell> set sdc_save_source_file_information true
Error: can't set "sdc_save_source_file_information":
        Use error_info for more info. (CMD-013)
pt_shell> echo $sdc_save_source_file_information
false
```

Currently, the scope of source-location tracking applies to the following timing exceptions commands:

- `set_false_path`
- `set_multicycle_path`
- `set_max_delay`
- `set_min_delay`

To report the source of the constraints, use the `report_exceptions` or `report_timing -exceptions` commands. Consider the following:

- Commands entered interactively do not have source location information.
- For commands that are input inside control structures, such as `if` statements and `foreach` loops, the line number of the closing bracket is reported.
- For commands that are input inside procedure calls, the line number invoking the procedure is reported.

Example 6-1 and Example 6-2 show examples of exception reports containing the source file and line number information.

Example 6-1 Exceptions Report

```
pt_shell> report_exceptions

****************************************
Report : exceptions
Design : design1
Version: A-2007.12
Date   : Thu Apr  5 19:42:40 2007
****************************************

Reasons :   f  invalid start points
            t  invalid end points
            p  non-existent paths
            o  overridden paths
From                To                   Setup          Hold           Ignored
------------------------------------------------------------------------------
a[1]                lar5_ff1/D           cycles=3       *
         [ location = multi.tcl:17 ]
a[2]                lar5_ff2/D           cycles=4       *
         [ location = multi.tcl:18 ]
data[16]            lr16_ff/D            FALSE          FALSE
         [ location = scripts/false_path.tcl:11 ]
```

Example 6-2 Timing Report With the -exceptions Option

```
pt_shell> report_timing -exceptions all -from a[1] -to lar5_ff1/D

****************************************
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
        -exceptions all
Design : design1
Version: A-2007.12
Date   : Thu Apr  5 19:46:44 2007
****************************************

[ Timing report omitted…]

The dominant exceptions are:
From          To             Setup                    Hold
-------------------------------------------------------------------------------
-----
a[1]          lar5_ff1/D     cycles=3                 cycles=0
         [ location = multi.tcl:17 ]

The overridden exceptions are:
        None
```

Path Specification Methods

The `report_timing` command, the timing exception commands (such as `set_false_path`), and several other commands allow a variety of methods to specify a single path or multiple paths for a timing report or for applying timing exceptions. One way is to explicitly specify the `-from $startpoint` and `-to $endpoint` options in the `report_timing` command for the path, as in the following examples:

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

Each specified startpoint can be a clock, a primary input port, a sequential cell, a clock input pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has an input delay specified. If you specify a clock, all registers and primary inputs related to that clock are used as path startpoints. If you specify a cell, the command applies to one path startpoint on that cell.

Each specified endpoint can be a clock, a primary output port, a sequential cell, a data input pin of a sequential cell, or a pin that has an output delay specified. If you specify a clock, all registers and primary outputs related to that clock are used as path endpoints. If you specify a cell, the command applies to one path endpoint on that cell.

PrimeTime also supports a special form where the startpoint or endpoint becomes a `-through` pin, and a clock object becomes the `-from` or `-to` object. You can use this method with all valid startpoint and endpoint types, such as input ports, output ports, clock flip-flop pins, data flip-flop pins, or clock gating check pins. For example,

```
pt_shell> report_timing -from [get_clocks ...] -through $startpoint

pt_shell> report_timing -through $endpoint -to [get_clocks ...]
```

Note:

This method is more computationally intensive, especially in larger designs. Whenever possible, you should use `-from` and `-to` to specify the paths.

You can restrict the `report_timing` command to only rising edges or only falling edges by using the command arguments `-rise_from`, `-fall_from`, `-rise_to`, `-fall_through`, and so on.

Multiple Through Arguments

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

```
pt_shell> report_timing -from A1 -through B1 -through C1 -to D1
```

This means any path that starts at A1, passes through B1 and C1 in that order, and ends at D1. For example,

```
pt_shell> report_timing -from A1 -through {B1 B2} -through {C1 C2} -to D1
```

This means any path that starts at A1, passes through either B1 or B2, then passes through either C1 or C2, and ends at D1.

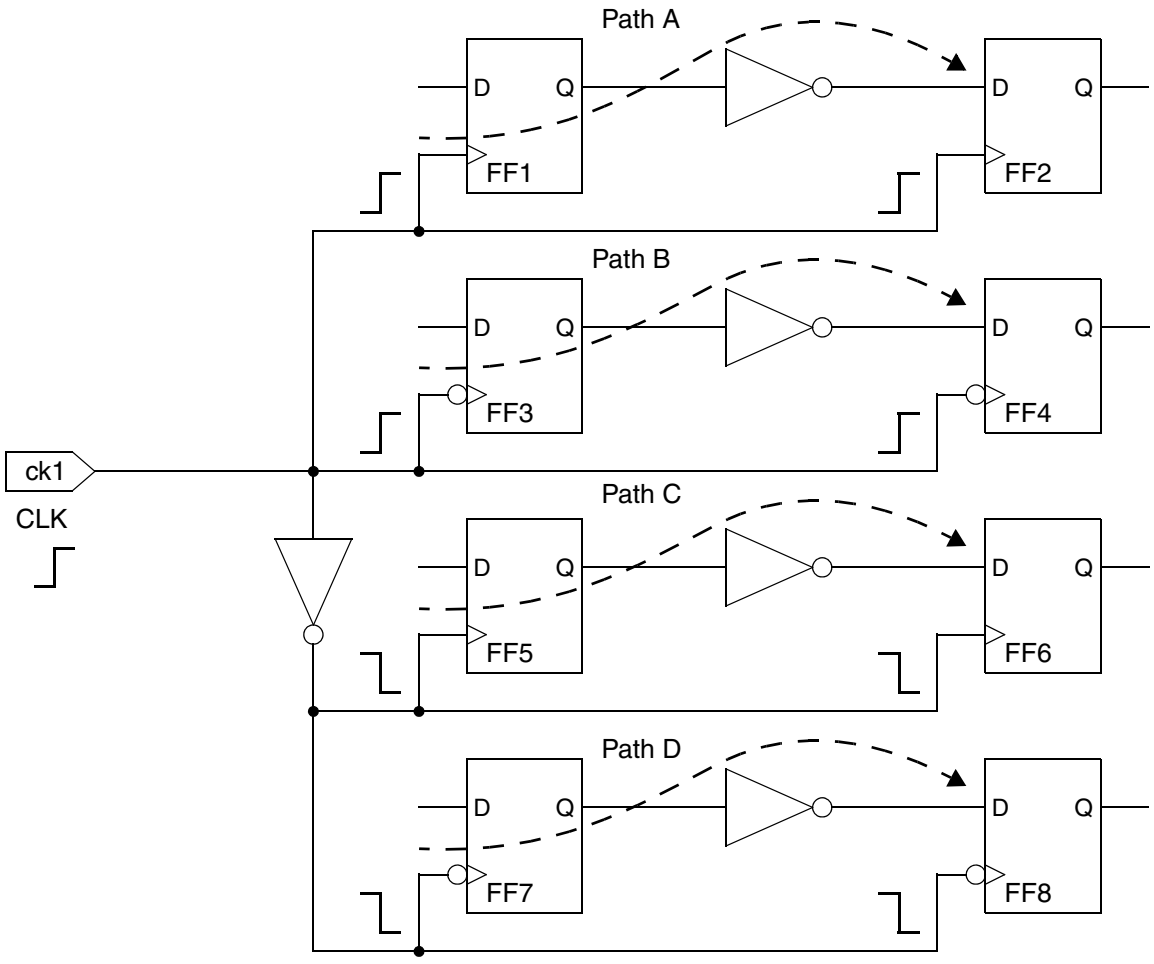
Rise/Fall From/To Clock

You can restrict the command to only to rising edges or only falling edges by using the command arguments `-rise_from`, `-fall_to`, and so on. When the “from” or “to” object is a clock, the command specifies paths based on the launch of data at startpoints or the capture of data at endpoints for a specific edge of the source clock. The path selection considers any logical inversions along the clock path.

For example, the `-rise_to` *`clock`* option specifies each path clocked by the specified clock at the endpoint, where a rising edge of the source clock captures data at the path endpoint. This can be a rising-edge-sensitive flip-flop without an inversion along the clock path, or a falling-edge-sensitive flip-flop with an inversion along the clock path. The data being captured at the path endpoint does not matter.

The following examples demonstrate the behavior with the `set_false_path` command. Consider the circuit shown in Figure 6-1. FF1, FF2, FF5, and FF6 are rising-edge-triggered flip-flops; and FF3, FF4, FF7, and FF8 are falling-edge-triggered flip-flops. FF1 through FF4 are clocked directly, whereas the FF5 through FF8 are clocked by an inverted clock signal.

Figure 6-1 Circuit for Path Specification Examples 1 Through 3



Example 1

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

In Figure 6-1, all paths clocked by CLK at the path endpoint (all four paths) are declared to be false.

Example 2

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

In Figure 6-1, all paths clocked by CLK at the startpoint that have data launched by a rising edge of the source clock are declared to be false, so Path A and Path D are false.

Example 3

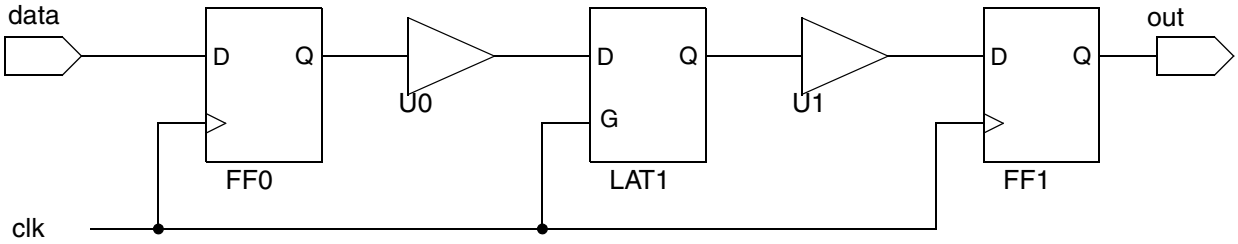
```
pt_shell> set_false_path -fall_to [get_clocks CLK]
```

In Figure 6-1, all paths clocked by CLK at the endpoint that capture data on a falling edge of the source clock are declared to be false, so Path B and Path C are false.

Example 4

Consider the circuit shown in Figure 6-2. The two flip-flops latch data on the positive-going edge of the clock. The level-sensitive latch opens on the rising edge and closes on the falling edge of the clock.

Figure 6-2 Circuit for Path Specification Example 4



Using `-rise_from` *`clock`* option of the `report_timing` command reports the paths with data launched by a rising edge of the clock, from `data` to `FF0`, from `FF0` to `LAT1`, from `LAT1/D` to `FF1`, from `LAT1/G` to `FF1`, and from `FF1` to `out`:

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

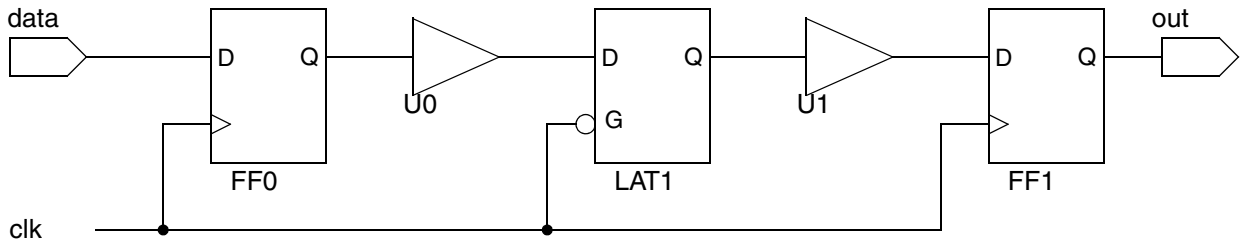
Using the `-fall_to` option in the command instead reports the path with data captured by a falling edge of the clock, which is from `FF0` to `LAT1`:

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

Example 5

The circuit shown in Figure 6-3 is the same as in the previous example, except that the level-sensitive latch opens on the falling edge and closes on the rising edge of the clock.

Figure 6-3 Circuit for Path Specification Example 5



Using the `-fall_from` *`clock`* option of the `report_timing` command reports the paths from `LAT1/D` to `FF1` and from `LAT1/G` to `FF1`:

```
pt_shell> report_timing -fall_from [get_clock clk] -nworst 100
```

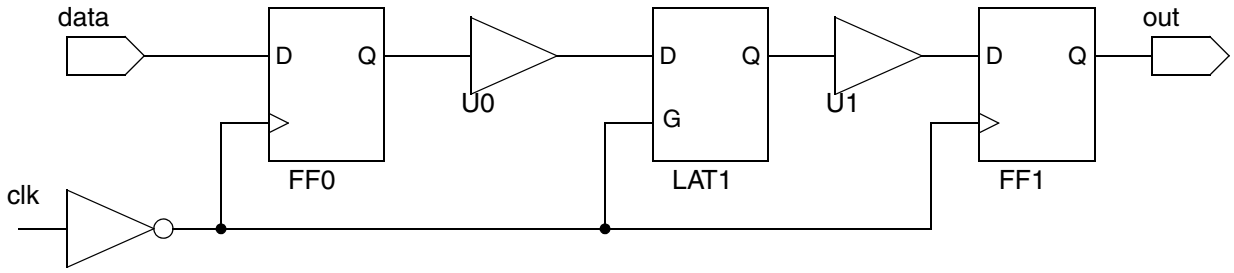
Using the `-rise_to` option in the command reports the paths from `data` to `FF0`, from `FF0` to `LAT1`, from `LAT1` to `FF1`, and from `FF1` to `out`:

```
pt_shell> report_timing -rise_to [get_clock clk] -nworst 100
```

Example 6

The circuit shown in Figure 6-4 is the same as in Example 4 (Figure 6-2 on page 6-15), except that the clock signal is inverted.

Figure 6-4 Circuit for Path Specification Example 6



Using the `-fall_from` *`clock`* option of the `report_timing` command reports the paths with data launched by a falling edge of the clock: from `FF0` to `LAT1`, from `LAT1/D` to `FF1`, from `LAT1/G` to `FF1`, and from `FF1` to `out`:

```
pt_shell> report_timing -fall_from [get_clock clk] -nworst 100
```

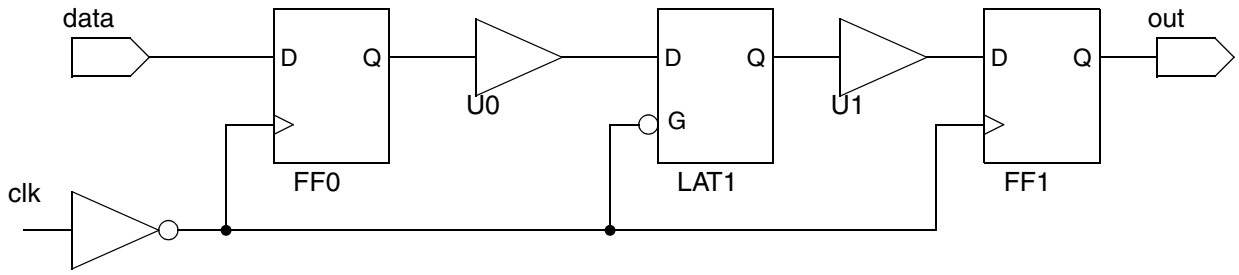
Using the `-rise_to` option of the command reports the path with data captured by a rising edge of the clock, which is from `FF0` to `LAT1` and from `FF1` to `out`:

```
pt_shell> report_timing -rise_to [get_clock clk] -nworst 100
```

Example 7

The circuit shown in Figure 6-5 is the same as in Example 5 (Figure 6-3 on page 6-16), except that the clock signal is inverted.

Figure 6-5 Circuit for Path Specification Example 7



Using the `-rise_from` *`clock`* option in the `report_timing` command reports the paths with data launched by a rising edge of the clock: from `data` to `FF0` and from `LAT1/G` to `FF1`:

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

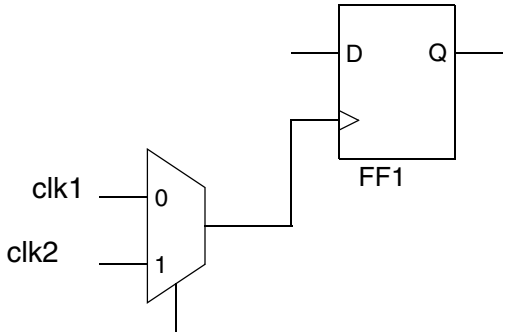
Using the `-fall_to` option in the command reports the paths from `data` to `FF0`, from `FF0` to `LAT1`, from `LAT1/D` to `FF1`, and from `LAT1/G` to `FF1`:

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

Example 8

In the circuit shown in Figure 6-6, multiple clocks reach the flip-flop.

Figure 6-6 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. You can do this by using the `-through` and `-to` options:

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

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

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

Reporting of Invalid Startpoints or Endpoints

The `timing_report_always_use_valid_start_end_points` variable specifies how PrimeTime reports invalid startpoints and endpoints. This variable specifies how `report_timing`, `report_bottleneck`, and `get_timing_paths` commands handle invalid startpoints when you use the `-from`, `-rise_from`, and `-fall_from` options. It also specifies how they handle invalid endpoints when you use the `-to`, `-rise_to`, or `-fall_to` options.

When the `timing_report_always_use_valid_start_end_points` variable is set to `false` (the default), the *from_list* is interpreted as all pins within the specified scope of the listed objects. Objects specified in a general way can include many invalid startpoints or endpoints. For example, `-from [get_pins FF/*]` includes input, output, and asynchronous pins. By default, PrimeTime considers these invalid startpoints and endpoints as through points, and continues path searching. It is easy to specify objects this way, but it wastes runtime on useless searching. When this variable is set to `true`, each reporting command ignores invalid startpoints and invalid endpoints.

Irrespective of this variable setting, it is recommended that you always specify valid startpoints (input ports and register clock pins) in the *from_list* and valid endpoints (output ports and register data pins) in the *to_list*.

Detecting max_paths or nworst Pruning

The `timing_report_maxpaths_nworst_reached` variable (default value is `false`) controls the issuing of an information message when the `report_timing -maxpaths -nworst` command reaches the number of paths specified by the`-max_paths` and `-nworst` options, but there are still qualifying paths left unreported. For example, if you had four endpoints: A (with five paths), B (with one path), C (with three paths), and D (with two paths), and you used the following command:

```
pt_shell> report_timing -max_paths 7 -nworst 3
```

If the `report_timing` command reported three paths from A, one path from B, and three paths from C, the two paths from D are dropped entirely. However, with the `timing_report_maxpaths_nworst_reached` variable set to `true` PrimeTime issues an information message similar to the following:

```
The group clk has -max_paths reached.
The following endpoints have nworst reached:
   A
```

You could then elect to increase the value of the `-max_paths` option, the `-nworst` option, or both.

Path Timing Calculation

The `report_delay_calculation` command reports the detailed calculation of delay from the input to the output of a cell or from the driver to a load on a net. The type of information you see depends on the delay model you are using.

Many ASIC vendors consider detailed information about cell delay calculation to be proprietary. To protect this information, the `report_delay_calculation` command shows cell delay details only if the library vendor has enabled this feature with the `library_features` attribute in Library Compiler. For more information, see “report_delay_calculation” on page 2-36 or the specific man page.

7

Clocks

An essential part of timing analysis is accurately specifying clocks and clock effects, such as latency (delay from the clock source) and uncertainty (amount of skew or variation in the arrival of clock edges). You can specify, report, and analyze clocks are described in the following sections:

Clock Overview

PrimeTime supports the following types of clock information:

Multiple clocks

You can define multiple clocks that have different waveforms and frequencies. Clocks can have real sources in the design (ports and pins) or can be virtual. A virtual clock has no real source in the design itself.

Clock network delay and skew

You specify the delay of the clock network relative to the source (clock latency) and the variation of arrival times of the clock at the destination points in the clock network (clock skew). For multiclock designs, you can specify interclock skew. You can specify an ideal delay of the clock network for analysis before clock tree generation, or you can specify that the delay needs to be computed by PrimeTime for analysis after clock tree generation. PrimeTime also supports checking the minimum pulse width along a clock network.

Gated clocks

You can analyze a design that has gated clocks. A gated clock is a clock signal under the control of gating logic (other than simple buffering or inverting a clock signal). PrimeTime performs both setup and hold checks on the gating signal.

Generated clocks

You can analyze a design that has generated clocks. A generated clock is a clock signal generated from another clock signal by a circuit within the design itself, such as a clock divider.

Clock transition times

You can specify the transition times of clock signals at register clock pins. The transition time is the amount of time it takes for the signal to change from one logic state to another.

Specifying Clocks

You need to specify all clocks used in the design. The clock information includes:

- Period and waveform
- Latency (insertion delay)
- Uncertainty (skew)
- Divided and internally generated clocks

- Clock-gating checks
- Fixed transition time for incomplete clock networks

PrimeTime supports analysis of the synchronous portion of a design. PrimeTime analyzes paths between registers or ports. For a design with multiple interacting clocks, PrimeTime determines phase relationship between the startpoint and endpoint clocks. The clocks can be single phase, multiple phase, or multiple frequency clocks.

Creating Clocks

You must specify all of the clocks in the design by using the `create_clock` command. This command creates a clock at the specified source. A source can be defined at an input port of the design or an internal pin. PrimeTime traces the clock network automatically so the clock reaches all registers in the transitive fanout of its source.

A clock you create with the `create_clock` command has an ideal waveform that ignores the delay effects of the clock network. After you create the clock, you must describe the clock network to perform accurate timing analysis (described in "Specifying Clock Characteristics" on page 7-5).

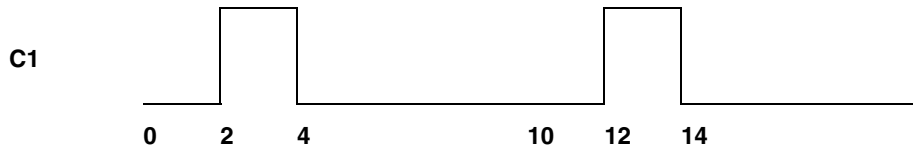
The `create_clock` command creates a path group having the same name as the clock. This group contains all paths ending at points clocked by this clock. For information about path groups, see "Path Groups" on page 6-2.

To create a clock on ports C1 and CK2 with a period of 10, a rising edge at 2, and a falling edge at 4, enter

```
pt_shell> create_clock -period 10 -waveform {2 4} {C1 CK2}
```

The resulting clock is named C1 to correspond with the first source listed. C1 produces the waveform shown in Figure 7-1.

Figure 7-1 C1 Clock Waveform



PrimeTime supports analyzing multiple clocks propagated to a single register. You can define multiple clocks on the same port or pin by using the `-add` option of the `create_clock` command. For more information, see the `create_clock` man page.

Creating a Virtual Clock

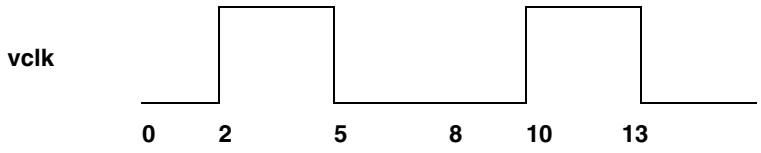
You can use the `create_clock` command to define virtual clocks for signals that interface to external (off-chip) clocked devices. A virtual clock has no actual source in the current design, but you can use it for setting input or output delays.

To create a virtual clock named vclk, enter

```
pt_shell> create_clock -period 8 -name vclk -waveform {2 5}
```

The vclk clock has the waveform shown in Figure 7-2.

Figure 7-2 vclk Clock Waveform



Selecting Clock Objects

The `get_clocks` command selects clocks for a command to use, for example, to ensure that a command works on the CLK clock and not on the CLK port. For more information, see the `get_clocks` man page.

To report the attributes of clocks having names starting with PHI1 and a period less than or equal to 5.0, enter

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

Applying Commands to All Clocks

The `all_clocks` command is equivalent to the PrimeTime `get_clocks *` command. The `all_clocks` command returns a token representing a collection of clock objects. The actual clock names are not printed.

To disable time borrowing for all clocks, enter

```
pt_shell> set_max_time_borrow 0 [all_clocks]
```

Removing Clock Objects

The `remove_clock` command removes a list of clock objects or clocks in the design. For more information, see the `remove_clock` man page.

Note:
The `reset_design` command removes clocks as well as other information.

To remove all clocks whose names start with CLKB, enter

```
pt_shell> remove_clock [get_clocks CLKB*]
```

To remove all clocks in the design, enter

```
pt_shell> remove_clock -all
```

Specifying Clock Characteristics

Clocks you create with the `create_clock` command have perfect waveforms that ignore the delay effects of the clock network. For accurate timing analysis, you must describe the clock network. The main characteristics of a clock network are latency and uncertainty.

Latency consists of clock source latency and clock network latency. Clock source latency is the time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design. Clock network latency is the time a clock signal (rise or fall) takes to propagate from the clock definition point in the design to a register clock pin.

Clock uncertainty is the maximum difference between the arrival of clock signals at registers in one clock domain or between domains. This is also called skew. Because clock uncertainty can have an additional margin built in to tighten setup or hold checks, you can specify different uncertainties for setup and hold checks on the same path.

Setting Clock Latency

PrimeTime provides two methods for representing clock latency. You can either

- Allow PrimeTime to compute latency by propagating the delays along the clock network. This method is very accurate, but it can be used only after clock tree synthesis has been completed.
- Estimate and specify explicitly the latency of each clock. You can specify this latency on individual ports or pins. Any register clock pins in the transitive fanout of these objects are affected and override any value set on the clock object. This method is typically used before clock tree synthesis.

Setting Propagated Latency

You can have PrimeTime automatically determine clock latency by propagating delays along the clock network. This process produces highly accurate results after clock tree synthesis and layout, when the cell and net delays along the clock network are all back-annotated or net parasitics have been calculated. The edge times of registers clocked by a propagated clock are skewed by the path delay from the clock source to the register clock pin. Using propagated latency is appropriate when your design has actual clock trees and annotated delay or parasitics.

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

```
pt_shell> set_propagated_clock [get_clocks CLK]
```

You can set the propagated clock attribute on clocks, ports, or pins. When set on a port or pin, it affects all register clock pins in the transitive fanout of the object.

To remove a propagated clock specification on clocks, ports, pins, or cells in the current design, use the `remove_propagated_clock` command.

Specifying Clock Source Latency

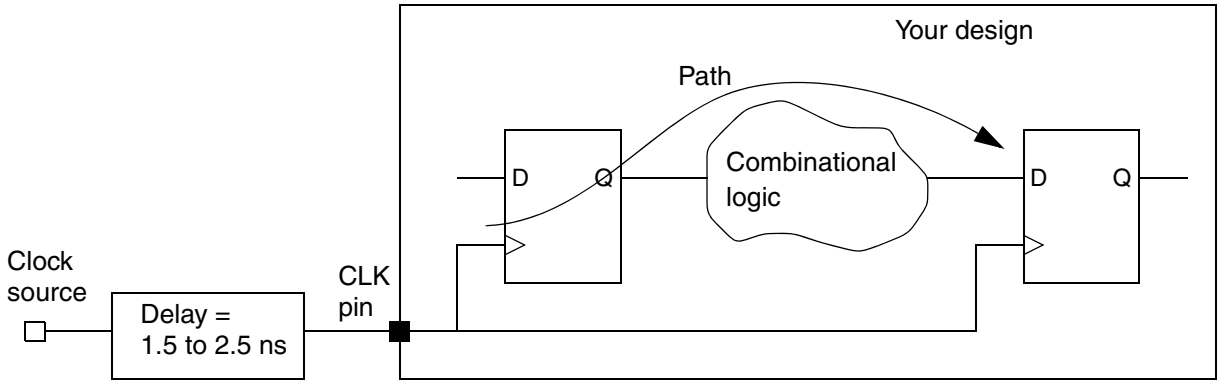
You can specify clock source latency for ideal or propagated clocks. Source latency is the latency from the ideal waveform to the source pin or port. The latency at a register clock pin is the sum of clock source latency and clock network latency.

For internally generated clocks, PrimeTime can automatically compute the clock source latency if the master clock of the generated clock has propagated latency and there is no user-specified value for generated clock source latency.

Propagated latency calculation is usually inaccurate for prelayout design because the parasitics are unknown. For prelayout designs, you can estimate the latency of each clock and directly set that estimation with the `set_clock_latency` command. This method, known as ideal clocking, is the default method for representing clock latency in PrimeTime. The `set_clock_latency` command sets the latency for one or more clocks, ports, or pins.

To specify an external uncertainty for source latency, use the `-early` and `-late` options of the `set_clock_latency` command. For example, consider a source latency that can vary from 1.5 to 2.5 ns, as illustrated in Figure 7-3.

Figure 7-3 External Source Latency



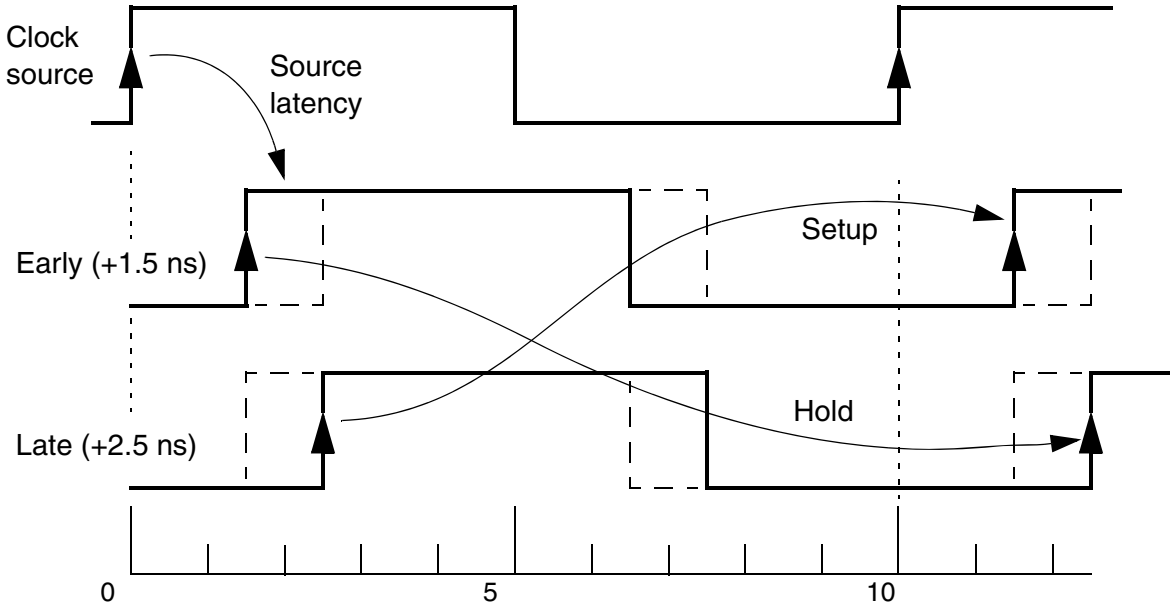
To specify this type of source latency, you can use commands such as the following:

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

PrimeTime uses the more conservative source latency value (either early or late) for each startpoint and endpoint clocked by that clock. For setup analysis, it uses the late value for each startpoint and the early value for each endpoint. For hold analysis, it uses the early

value for each startpoint and the late value for each endpoint. Figure 7-4 shows the early and late timing waveforms and the clock edges used for setup and hold analysis in the case where the startpoint and endpoint are clocked by the same clock.

Figure 7-4 Early/Late Source Latency Waveforms



The following examples demonstrate how to set different source latency values for rising and falling edges.

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

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

To specify an early rise and fall source latency of 0.8 and a late rise and fall source latency of 0.9 for CLK1, enter

```
pt_shell> set_clock_latency 0.8 -source -early [get_clocks CLK1]
pt_shell> set_clock_latency 0.9 -source -late [get_clocks CLK1]
```

The `remove_clock_latency` command removes user-specified clock network or source clock latency information from specified objects. For more information, see the man page.

Setting Clock Uncertainty

You can model the expected uncertainty (skew) for a prelayout design with setup or hold and rise or fall uncertainty values. PrimeTime subtracts a setup uncertainty value from the data required time when it checks setup time (maximum paths). PrimeTime adds a hold

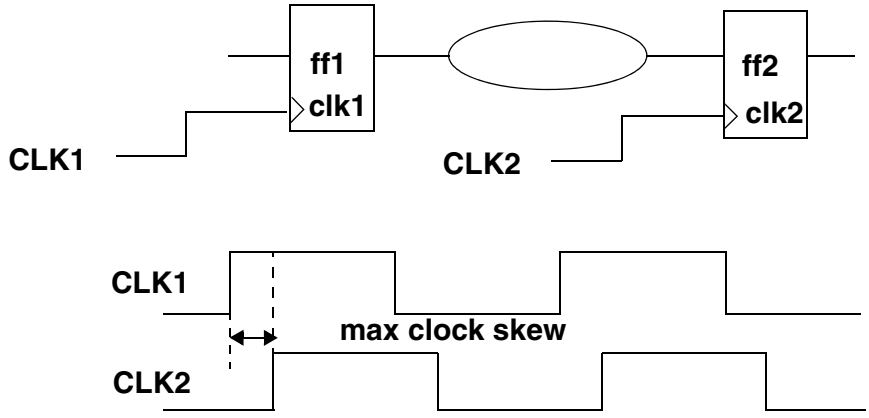
uncertainty value to the data required time when it checks the hold time (minimum paths). If you specify a single uncertainty value, PrimeTime uses it for both setup checks and hold checks.

You can specify the uncertainty or skew characteristics of clocks by using the `set_clock_uncertainty` command. The command specifies the amount of time variation in successive edges of a clock or between edges of different clocks. It captures the actual or predicted clock uncertainty.

You can specify simple clock uncertainty or interclock uncertainty. Simple uncertainty is the variation in the generation of successive edges of a clock with respect to the exact, nominal times. You specify one or more objects, which can be clocks, ports, or pins. The uncertainty value applies to all capturing latches clocked by the specified clock or whose clock pins are in the fanout of the specified ports or pins.

Interclock uncertainty is more specific and flexible, supporting different uncertainties between clock domains. It is the variation in skew between edges of different clocks. You specify a “from” clock using the `-from`, `-rise_from`, or `-fall_from` option and a “to” clock the `-to`, `-rise_to`, or `-fall_to` option. The interclock uncertainty value applies to paths that start at the “from” clock and end at the “to” clock. Interclock uncertainty is relevant when the source and destination registers are clocked by different clocks. You can define uncertainty similarly between two clock pins driven from the same clock, or you can define it as an interclock uncertainty between two registers with different clocks, as shown in Figure 7-5.

Figure 7-5 Example of Interclock Uncertainty



When performing a setup or hold check, PrimeTime adjusts the timing check according to the worst possible difference in clock edge times. For example, for a setup check, it subtracts the uncertainty value from the data required time, thus requiring the data to arrive sooner by that amount, to account for a late launch and an early capture with the worst clock skew.

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

```
pt_shell> set_clock_uncertainty 5 [get_clocks CLKA]
pt_shell> set_clock_uncertainty 2 -from [get_clocks CLKB] \
          -to [get_clocks CLKA]
```

When the path is from CLKB to CLKA, the interclock uncertainty value 2 is used.

The following commands specify interclock uncertainty for all possible interactions of clock domains. If you have paths from CLKA to CLKB, and CLKB to CLKA, you must specify the uncertainty for both directions, even if the value is the same. For example,

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

To set simple clock uncertainty (setup and hold) for all paths leading to endpoints clocked by U1/FF*/CP, enter

```
pt_shell> set_clock_uncertainty 0.45 [get_pins U1/FF*/CP
```

To set a simple setup uncertainty of 0.21 and a hold uncertainty of 0.33 for all paths leading to endpoints clocked by CLK1, enter

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

To remove clock uncertainty information from clocks, ports, pins, or cells, or between specified clocks, use the `remove_clock_uncertainty` command removes uncertainty.

Dynamic Effects of Clock Latency

Dynamic effects on the clock source latency, such as phase-locked loop (PLL) clock jitter can be modeled using the `-dynamic` option of the `set_clock_latency` command. This option allows you to specify a dynamic component of the clock source latency. Clock reconvergence pessimism removal (CRPR) handles the dynamic component of clock latency in the same way as it handles the PrimeTime SI delta delays. For more information about CRPR, see “CRPR With Dynamic Clock Arrivals” on page 12-28.

You can model clock jitter using the `set_clock_uncertainty` command. However, the clock uncertainty settings do not affect the calculation of crosstalk arrival windows and are not considered by CRPR. The `set_clock_latency` command allows you to specify clock jitter as dynamic source clock latency. The clock jitter specified in this manner properly affects the calculation of arrival windows. The static and dynamic portions are also correctly handled by CRPR.

For example, to specify an early source latency of 3.0 and a late source latency of 5.0 for clock CLK:

```
pt_shell> set_clock_latency -early -source 3.0 [get_port CLK]
pt_shell> set_clock_latency -late -source 5.0 [get+port CLK]
```

To model external dynamic effects, such as jitter in the clock source, you can adjust the early and late source latency values. For example, to add plus or minus 0.5 more time units of dynamic latency to the static latency:

```
pt_shell> set_clock_latency -early -source 2.5  \
          -dynamic -0.5 [get_clocks CLK]

pt_shell> set_clock_latency -source -late 5.5  \
          -dynamic 0.5 [get_clocks CLK]
```

The first of these two commands specifies a total early source latency of 2.5, consisting of static source latency of 3.0 and dynamic source latency of –0.5. The second command specifies a total late source latency of 5.5, consisting of static source latency of 5.0 and dynamic source latency of 0.5.

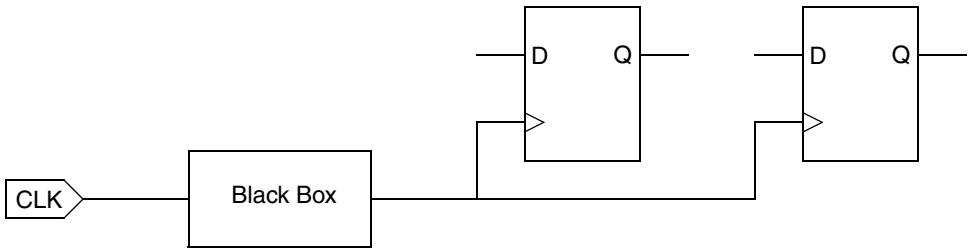
If dynamic latency has been specified as in the foregoing example and the clock named CLK has been set to use propagated latency, the total latency (static and dynamic) is used for delay calculations. However, for a timing path that uses different clock edges for launch and capture, CRPR uses only the static latency, not the dynamic latency, leading up to the common point in the clock path, and removes the resulting static pessimism from the timing results.

The `report_timing` command reports the static and dynamic component of clock latency separately. The `report_crpr` command reports the clock reconvergence pessimism (CRP) time values calculated for both static and dynamic conditions, and the choice from among those values actually used for pessimism removal. For more information about the `report_crpr` command, see “Reporting CRPR Calculations” on page 12-29.

Dynamic latency is supported only as part of a source latency specification. It is not supported as part of a network latency specification within a design or block. Therefore, you can specify dynamic latency only for the source latency of a clock object. You cannot set it on the source latency of port or pin objects, although support for this capability is planned

for a future release. If you want to model dynamic effects of an internal black box like the one shown in Figure 7-6, you can create a clock at the output of the black box and specify the dynamic latency of that clock source.

Figure 7-6 Internal Black Box in Clock Network



Estimating Clock Pin Transition Time

Transition times are typically computed for the ports, cells, and nets in the clock network. If the clock network is not complete, the result can be inaccurate transition times on register clock pins. For example, a single buffer might be driving 10,000 register clock pins because the clock tree has not been constructed. This can cause a long transition time on the register clock pins affecting clock-to-output delays, as well as setup and hold delay calculation.

You can estimate the clock transition time for an entire clock network using the `set_clock_transition` command. To specify a nonzero transition time for an ideal clock, use the `set_clock_transition` command. For example,

```
pt_shell> set_clock_transition 0.64 -fall [get_clocks CLK1]
```

The transition time value applies to all nets directly feeding sequential elements clocked by the specified clock.

Use the `-rise` or `-fall` option of the command to specify a separate transition time for only rising or only falling edges of the clock. Use the `-min` or `-max` option to specify the transition time for minimum operating conditions or maximum operating conditions.

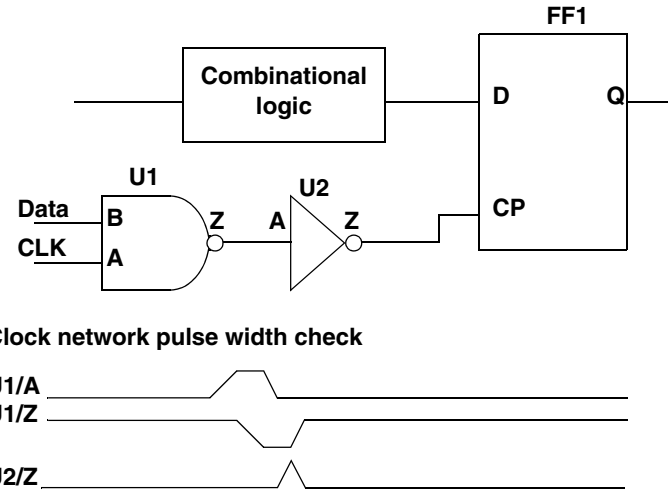
To remove the estimated clock pin transition time, use the `remove_clock_transition` command.

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 because of gating logic or delay characteristics of the clock network. Two problems can result:

- If the clock pulse is too small at a register clock pin, the device might not capture data properly. Library cell pins might have a minimum pulse width limit, which is checked by the `report_constraint` command.
- The pulse width might shrink so much at some point in the clock network that it is not propagated further, as shown in Figure 7-7. PrimeTime can check the entire clock network to ensure that the pulse does not shrink below a certain threshold.

Figure 7-7 Clock Network Pulse Width Check



No default value is assumed for clock tree pulse width checks. To specify the constraint for normal non-pulse generator for clocks, cells, pins, ports, and the current design, use the `set_min_pulse_width` command. To constrain the pulse generator networks, use the `set_pulse_clock_min_width` and `set_pulse_clock_max_width` commands.

Note:

The `report_constraint` command checks minimum pulse width for register clock pins and for clock networks.

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

Note:

You can also specify minimum pulse width on clock pins in the library cell description.

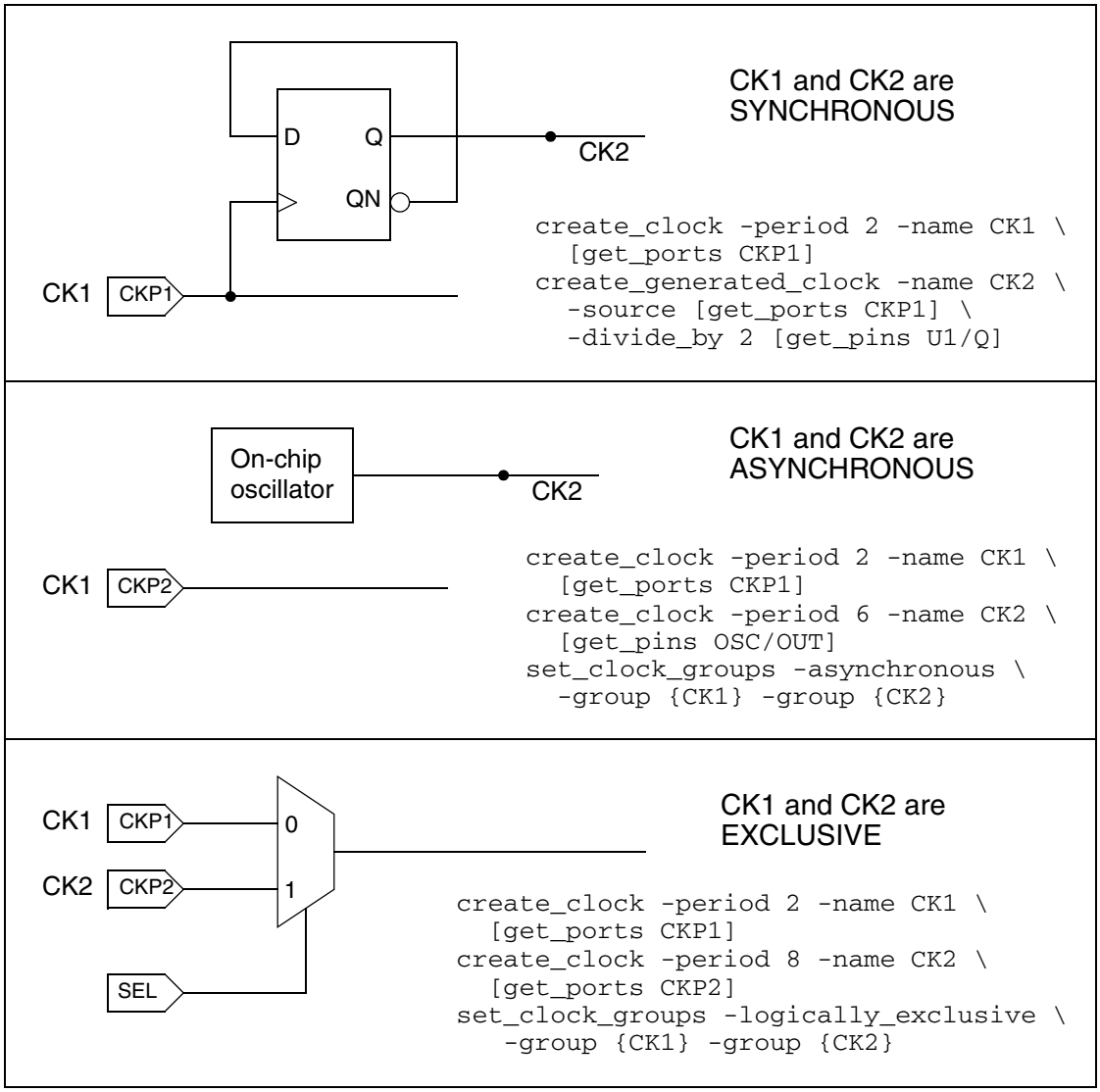
To remove minimum pulse width checks for clock signals in a clock tree or at sequential devices, use the `remove_min_pulse_width` command.

Using Multiple Clocks

When multiple clocks are defined for a design, the relationships between the clock domains depend on how the clocks are generated and how they are used in the design. The relationship between two clocks can be synchronous, asynchronous, or exclusive, as shown by the examples in Figure 7-8.

For PrimeTime to analyze paths between different clock domains correctly, you might need to specify false paths between clocks, exclude one or more clocks from consideration during the analysis, or specify the nature of the relationships between different clocks.

Figure 7-8 Synchronous, Asynchronous, and Exclusive Clocks



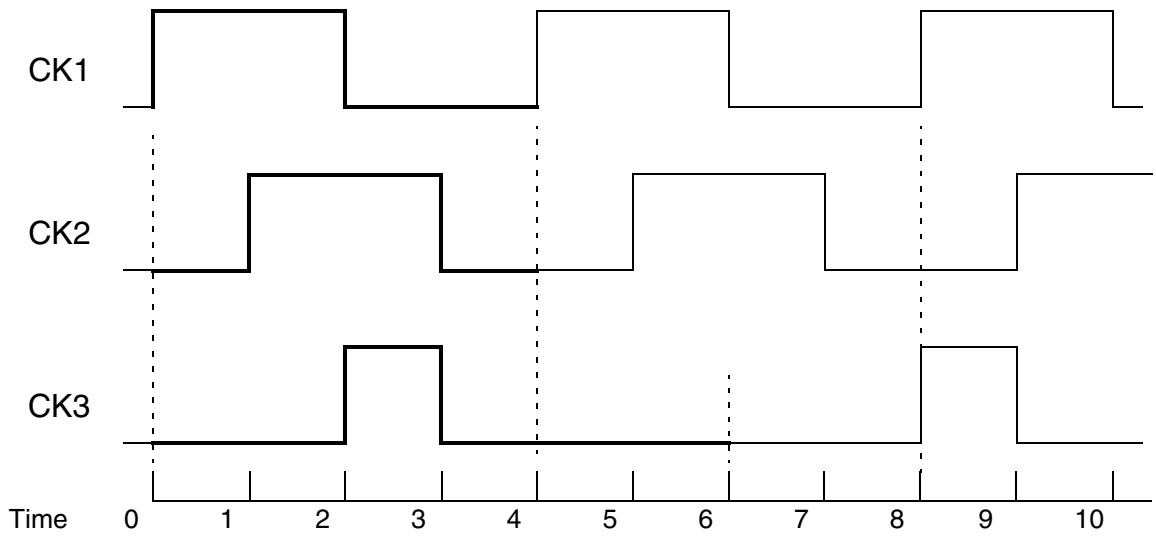
Synchronous Clocks

Two clocks are synchronous with respect to each other if they share a common source and have a fixed phase relationship. Unless you specify otherwise, PrimeTime assumes that two clocks are synchronous if there is any path with data launched by one clock and captured by the other clock. The clock waveforms are synchronized at time zero, as defined by the `create_clock` command. For example, consider the following `create_clock` commands:

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

PrimeTime creates the clocks as specified in the commands, with the waveforms synchronized as shown in Figure 7-9. PrimeTime adjusts the timing relationships further for any specified or calculated latency or uncertainty.

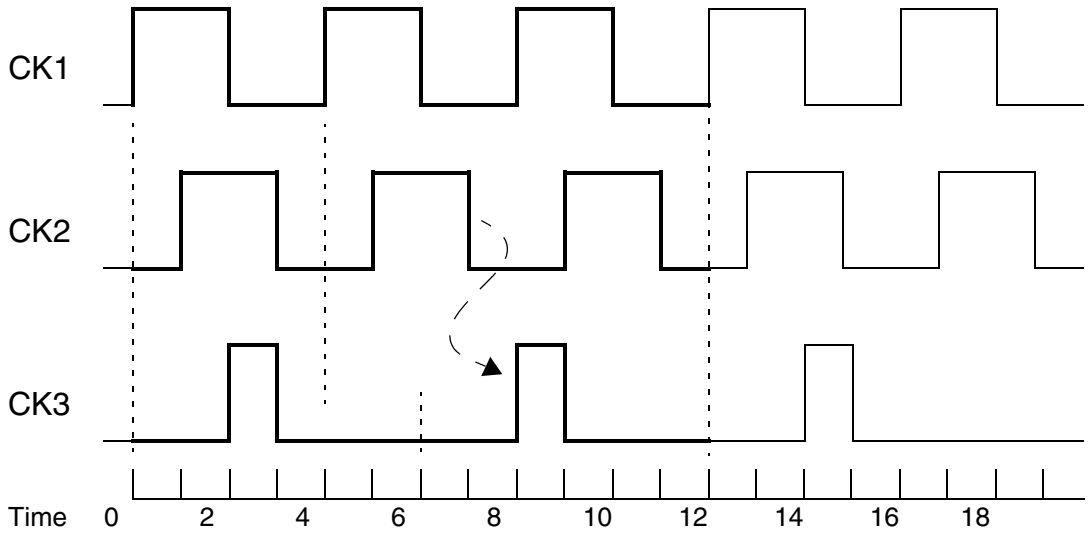
Figure 7-9 Synchronous Clock Waveforms



In a design that uses these three clocks, there might be paths launched by one clock and captured by another clock. When such paths exist, to test all possible timing relationships between different clock edges, PrimeTime internally “expands” the clocks to the least common multiple of all synchronous clock periods, thus creating longer-period clocks with multiple rising and falling edges For example, the three clocks in the foregoing example have periods of 4, 4, and 6. The least common multiple of these periods, called the base period,

is 12. To analyze the paths that cross the clock domains, PrimeTime internally expands the clocks by repeating them over the base period. The resulting clock waveforms are shown in Figure 7-10. Each expanded clock has a period of 12.

Figure 7-10 Expanded Clock Waveforms



PrimeTime checks timing paths between all edges in the expanded clocks. For example, the most restrictive setup check between a falling edge of CK2 and a rising edge of CK3 is from time=7 to time=8, as shown by the dashed arrow in Figure 7-10.

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 the `set_clock_groups -logically_exclusive` or `set_false_path` command.

Asynchronous Clocks

Two clocks are asynchronous if they do not communicate with each other in the design. For example, a free-running, on-chip oscillator is asynchronous with respect to a system clock signal coming into the chip from the outside. Clock edges in the two clock domains can occur at any time with respect to each other.

You can declare a relationship between two clocks to be asynchronous. In that case, PrimeTime does not check the timing paths launched by one clock and captured by the other clock, which is like declaring a false path between the two clocks. In addition, if you are doing crosstalk analysis, PrimeTime SI assigns infinite arrival windows to the nets in aggressor-victim relationships between the two clock domains.

To declare an asynchronous relationship between two clocks, use the `set_clock_groups -asynchronous` command. For more information, see the man page for the command or the *PrimeTime SI User Guide*.

Exclusive Clocks

Two clocks are exclusive if they do not interact with each other. For example, a circuit might multiplex two different clock signals onto a clock line, one a fast clock for normal operation and the other a slow clock for low-power operation. Only one of the two clocks is enabled at any given time, so there is no interaction between the two clocks.

To prevent PrimeTime from spending time checking the interaction between exclusive clocks, you can declare a false path between the clocks or use the `set_clock_groups -logically_exclusive` command to declare the clocks to be exclusive. Otherwise, you can use case analysis to disable the clock that you do not want to include in the current analysis.

To declare clocks CK1 and CK2 to be exclusive:

```
pt_shell> set_clock_groups -logically_exclusive \
          -group {CK1} -group {CK2}
```

This causes PrimeTime to ignore any timing path that starts from the CK1 domain and ends at the CK2 domain, or from the CK2 to the CK1 domain. This is like setting a false path from CK1 to CK2 and from CK2 to CK1. However, this setting is not reported by the `report_exceptions` command. To find out about clock groups that have been set, use the `report_clock -groups` command. If desired, you can also use the `report_clock -groups` *`clock_list`* command to restrict the clock groups report to specific clocks of interest.

Avoid setting false paths between clock domains that have been declared to be exclusive because doing so is redundant. The `set_clock_groups -logically_exclusive` command invalidates all false paths set between the exclusive clock domains. This is also true for the `set_clock_groups -asynchronous` command.

You can specify multiple clocks in each group. For example, to declare clocks CK1 and CK2 to be exclusive with respect to CK3 and CK4:

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

This causes PrimeTime to ignore any path that starts in one group and ends in the other group.

If you specify more than two groups, each group is exclusive with respect to the other specified groups. For example,

```
pt_shell> set_clock_groups -logically_exclusive \
          -group {CK1 CK2} -group {CK3 CK4} -group {CK5}
```

If you specify just one group, that group is exclusive with respect to all other clocks in the design. For example,

```
pt_shell> set_clock_groups -logically_exclusive \
          -group {CK1 CK2}
```

You can optionally assign a name to a clock group declaration, which makes it easier to later remove that particular declaration:

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

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

```
pt_shell> remove_clock_groups -logically_exclusive -name EX1
```

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

```
pt_shell> remove_clock_groups -logically_exclusive -all
```

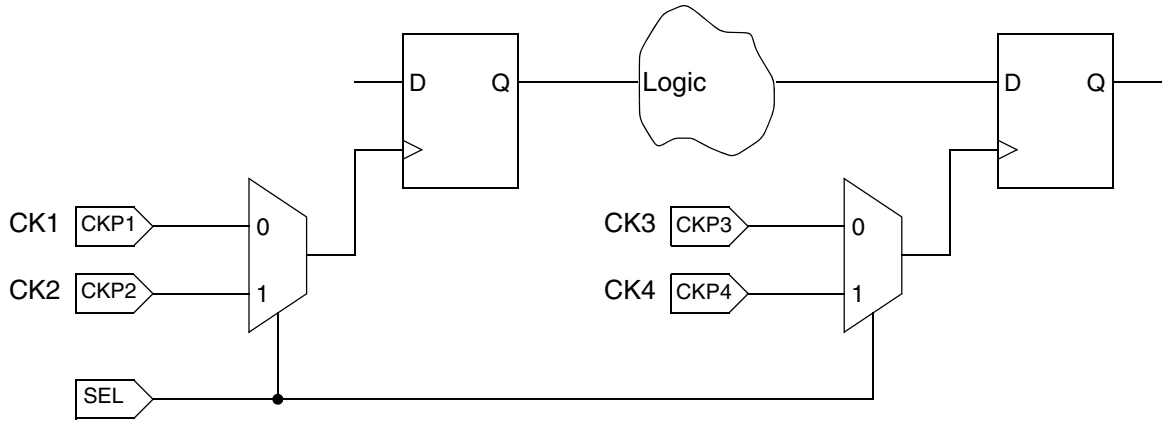
The `set_clock_groups -asynchronous` command defines groups of clocks that are asynchronous with respect to each other. Asynchronous clock group assignments are separate from exclusive clock group assignments, even though both types of clock groups are defined with the `set_clock_groups` command. Clock groups can be physically exclusive as well as logically exclusive due to multiplexing of the clock signals or physical separation. There can be no crosstalk between physically exclusive clocks, as well as no logical interaction. In that situation, use the `-physically_exclusive` option rather than the `-logically_exclusive` option. This prevents the tool from attempting to perform crosstalk analysis between the clock nets. For information about the handling of asynchronous clocks in crosstalk analysis, see the *PrimeTime SI User Guide*.

The following examples demonstrate some of the ways to specify exclusive clocks.

Example: Four Clocks and One Selection Signal

Consider the circuit shown in Figure 7-11. There are four clocks, CK1 through CK4. By default, PrimeTime analyzes the interactions between all combinations of clocks. However, the logic enables only two clocks at a time, either CK1 and CK3 or CK2 and CK4.

Figure 7-11 Four Clocks and One Selection Signal



One way to prevent checking between unrelated clocks is to set a false path between the clocks. For example,

```
pt_shell> set_false_path -from CK1 -to CK2
pt_shell> set_false_path -from CK2 -to CK1
pt_shell> set_false_path -from CK3 -to CK4
pt_shell> set_false_path -from CK4 -to CK3
pt_shell> set_false_path -from CK1 -to CK4
pt_shell> set_false_path -from CK4 -to CK1
pt_shell> set_false_path -from CK2 -to CK3
pt_shell> set_false_path -from CK3 -to CK2
```

In that case, PrimeTime tests all of the valid combinations of enabled clocks in a single run, while ignoring the invalid combinations.

Another way is to use case analysis and set a logic value, either 0 or 1, on the SEL input, which checks the timing for a particular case of SEL=0 or SEL=1. For example,

```
pt_shell> set_case_analysis 0 [get_ports SEL]
```

With SEL=0, only CK1 and CK3 are active; CK2 and CK4 are ignored. If you want to analyze both cases, two analysis runs are necessary: one with SEL=0 and another with SEL=1.

Another method to accomplish the same effect is to use the `set_disable_timing` command. For example, to disable checking of all paths leading from the CKP2 and CKP4 clock input pins of the design:

```
pt_shell> set_disable_timing [get_ports {CKP2 CKP4}]
```

Still another way is to specify which clocks can be active together at the same time and which clocks are currently active. For example,

```
pt_shell> set_clock_groups -logically_exclusive -name E1 \
            -group {CK1 CK3} -group {CK2 CK4}
pt_shell> set_active_clocks [all_clocks]
```

The `set_clock_groups` command defines groups of clocks that are exclusive with respect to each other. PrimeTime does not check paths that start from a clock in one group and end at a clock in another group. If you specify just one group, that group is considered exclusive with respect to all other clocks in the design.

In the preceding example, the `set_active_clocks` command makes all four clocks active, so that PrimeTime analyzes all valid paths while avoiding the invalid clock combinations.

If you want to consider only the case where SEL=0, you can do it easily by using a different `set_active_clocks` command:

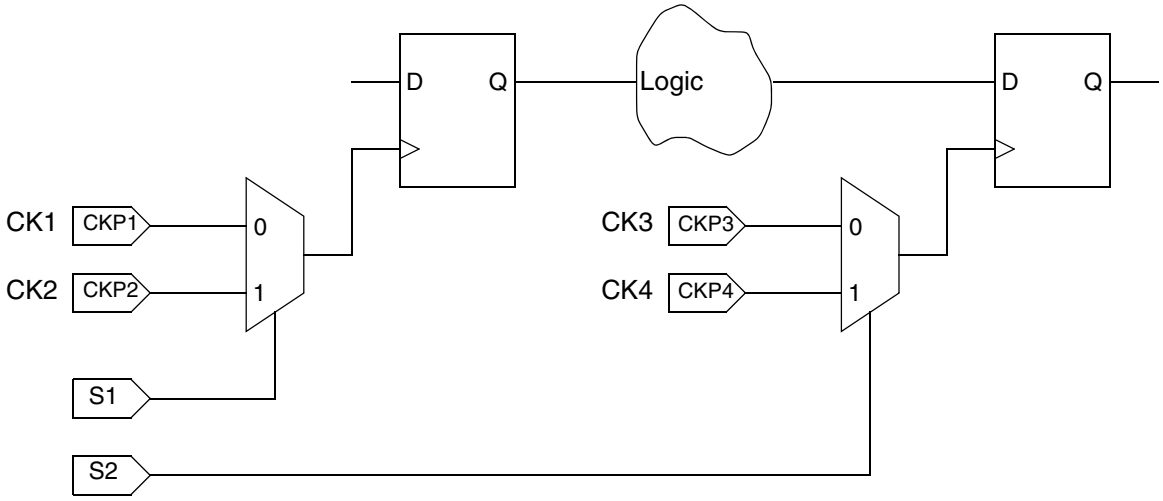
```
pt_shell> set_clock_groups -logically_exclusive -name E1 \
           -group {CK1 CK3} -group {CK2 CK4}
pt_shell> set_active_clocks {CK1,CK3}
```

Setting clocks CK1 and CK3 active means that CK2 and CK4 are inactive, which is just like using case analysis and setting SEL=0 or setting false paths between all combinations of clocks not using CK1 and CK3.

Example: Four Clocks and Two Selection Signals

Consider the circuit shown in Figure 7-12, which is similar to the previous example. There are two separate clock selection inputs, S1 and S2, for the two multiplexers.

Figure 7-12 Four Clocks and Two Selection Signals



Paths between CK1 and CK2 and between CK3 and CK4 are not valid, but all other combinations are possible.

To check all valid paths while avoiding invalid ones, you can declare false paths between the clocks:

```
pt_shell> set_false_path -from CK1 -to CK2
pt_shell> set_false_path -from CK2 -to CK1
pt_shell> set_false_path -from CK3 -to CK4
pt_shell> set_false_path -from CK4 -to CK3
```

Another way is to use case analysis and set logic values on S1 and S2 to check a particular case. For example,

```
pt_shell> set_case_analysis 0 [get_ports S1]
pt_shell> set_case_analysis 0 [get_ports S2]
```

If you want to analyze all four cases using case analysis, four analysis runs are necessary, with S1-S2 = 00, 01, 10, and 11. Still another way is to use the `set_clock_groups` and `set_active_clocks` commands. For example,

```
pt_shell> set_clock_groups -logically_exclusive -name mux1 \
          -group {CK1} -group {CK2}
pt_shell> set_clock_groups -logically_exclusive -name mux2 \
          -group {CK3} -group {CK4}
pt_shell> set_active_clocks {CK1,CK2,CK3,CK4}
```

PrimeTime analyzes all valid paths from CK1 to CK3 and CK4, and from CK2 to CK3 and CK4 (and in the opposite direction if there are any such paths), but not between CK1 and CK2 or between CK3 and CK4.

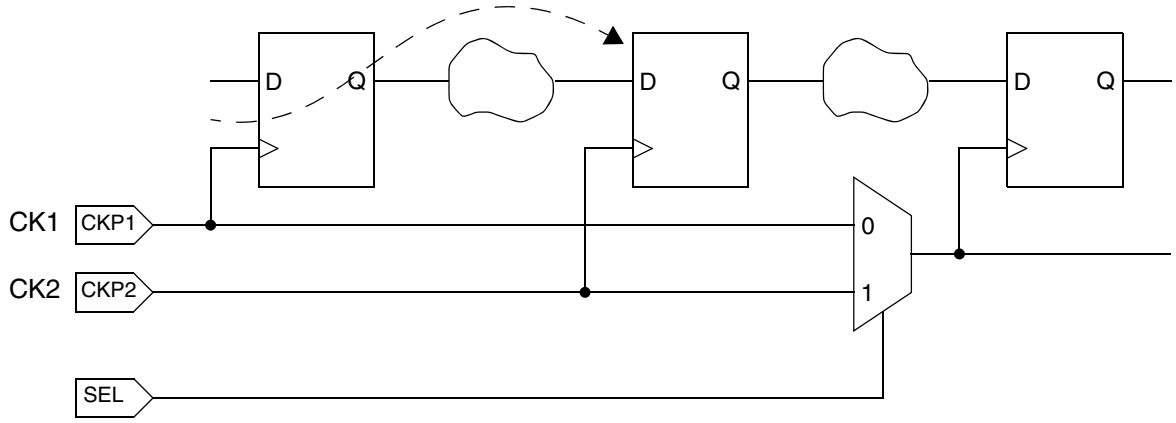
If you only want to consider the case where S1-S2=00, you can do it easily by using a different `set_active_clocks` command:

```
pt_shell> set_clock_groups -logically_exclusive -name mux1 \
          -group {CK1} -group {CK2}
pt_shell> set_clock_groups -logically_exclusive -name mux2 \
          -group {CK3} -group {CK4}
pt_shell> set_active_clocks {CK1,CK3}
```

Example: Multiplexed Clocks and Case Analysis

Consider the circuit shown in Figure 7-13. The clocks CK1 and CK2 are at the startpoint and endpoint of one path. These clocks are also multiplexed onto a shared clock line.

Figure 7-13 Multiplexed Clocks Specified With Case Analysis



If you want PrimeTime to check the valid path between CK1 and CK2, but avoid checking invalid paths between the two clock domains downstream from the multiplexer, the best way to do it is with case analysis.

Removing Clocks From Analysis

By default, PrimeTime analyzes all clocks specified for a design and all interactions between different clock domains. In a design with multiple clocks, it is often desirable to do timing analysis with only certain clocks enabled. For example, a device might use a fast clock for normal operation and a slow clock in power-down mode; you might be interested in the timing behavior for just one operating mode.

One way to specify the active clocks is to use case analysis. You specify a logic value, either 0 or 1, for a port or pin in the design that controls clock selection. For example, if a port called SEL selects the active clock in the device, you could use the following command:

```
pt_shell> set_case_analysis 0 [get_ports SEL]
```

In that case, timing analysis is restricted to the case where SEL=0.

Another method is to use the `set_active_clocks` command, which specifies the list of clocks that are active for the analysis. Clocks not listed in the command are inactive and not considered during the analysis. For example,

```
pt_shell> set_active_clocks {CK1,CK2}
```

Clocks CK1 and CK2 are considered for analysis and all others are ignored. If a generated clock is based on an inactive clock, the generated clock is also made inactive. Using the `set_active_clocks` command triggers a full timing update for the design.

To make all clocks active (the default behavior), use the `set_active_clocks [all_clocks]` command. To choose an entirely new set of active clocks, use the `set_active_clocks` command again; each use of the command overrides the previous settings. The `set_active_clocks` command works for standard timing analysis, but it does not work for context characterization, model extraction, or the `write_sdc` command.

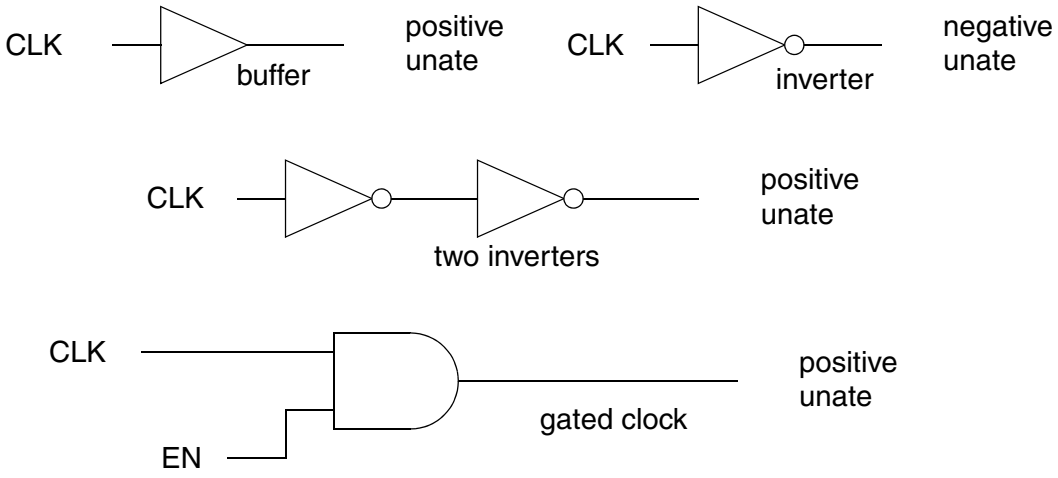
Clock Sense

PrimeTime keeps track of inverters and buffers in clock trees. It recognizes the positive or negative sense of the clock signal arriving at each register clock pin. No specific action is necessary to tell PrimeTime the sense of a clock tree that has only buffers and inverters. In this case, the clock signal arriving at the register clock pin is said to be “unate.”

A clock signal is “positive unate” if a rising edge at the clock source can only cause a rising edge at the register clock pin, and a falling edge at the clock source can only cause a falling edge at the register clock pin.

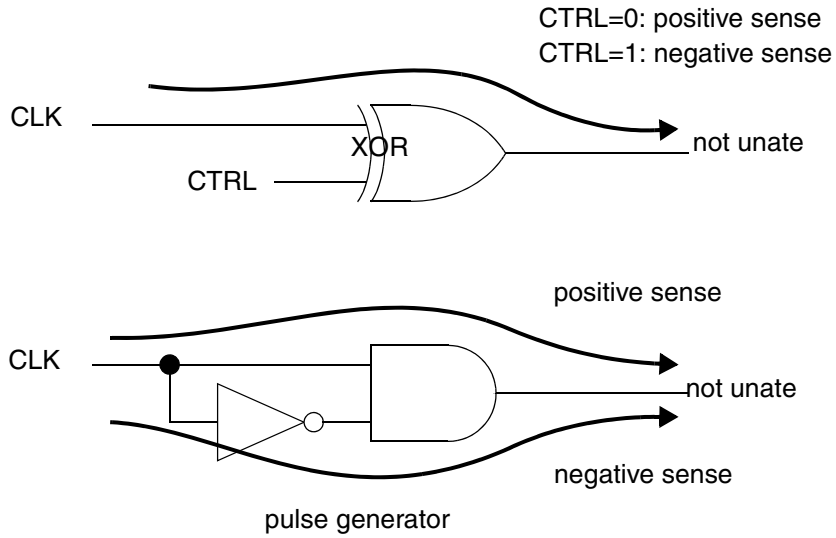
Similarly, a clock signal is “negative unate” if a rising edge at the clock source can only cause a falling edge at the register clock pin, and a falling edge at the clock source can only cause a rising edge at the register clock pin. In other words, the clock signal is inverted. See Figure 7-14.

Figure 7-14 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 7-15.

Figure 7-15 Non-Unate Clock Signals



PrimeTime considers the output of the pulse generator at the bottom of Figure 7-15 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.

You can use the `set_clock_sense` command to specify either a positive or negative clock sense to be propagated forward from a pin within a non-unate part of the clock network. The command 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.

To resolve this ambiguity for PrimeTime, you can specify the sense of a clock signal at a point in the clock path using the `set_clock_sense` command. For example,

```
pt_shell> set_clock_sense -positive [get_pins xor1.z]
```

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

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

```
pt_shell> set_clock_sense -positive \
          -clock [get_clocks CLK] [get_pins mux1.z]
```

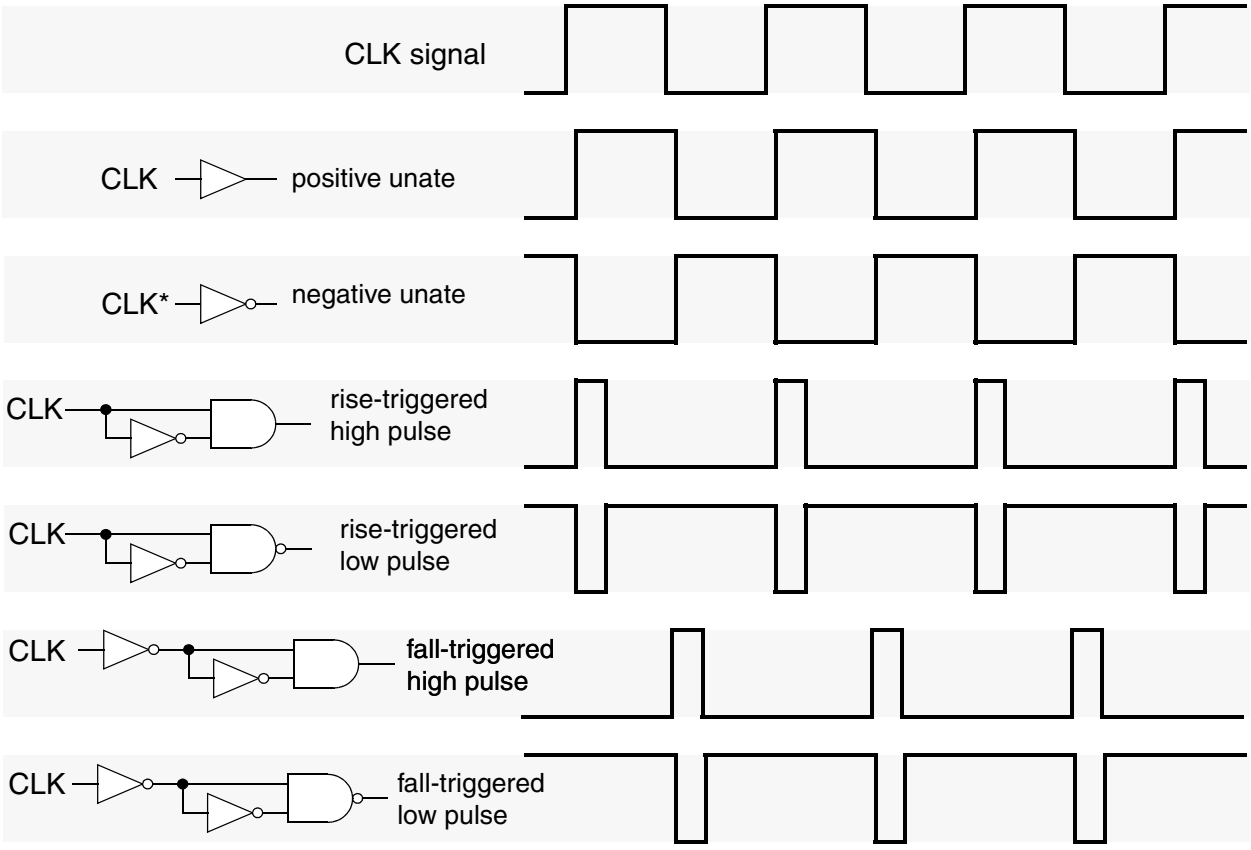
The `set_clock_sense` command has some additional sense types to support pulse clocks. Specifying the clock sense at a point in the design uses one of the following forms of syntax:

```
set_clock_sense -positive object_list
set_clock_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_clock_sense`, use the `remove_clock_sense` command.

Figure 7-16 shows some examples of clock-modifying circuits and the corresponding clock senses.

Figure 7-16 Clock Sense Examples



To stop clock propagation forward from a specified pin or cell timing arc, use one of the following commands:

- `set_clock_sense -stop_propagation`

  You could use the `-stop_propagation` option to stop propagation of specified clocks in the clock list from the specified pins or cell timing arcs in the object list. This is appropriate in cases where the clock physically does not propagate past a certain pin as both clock and clock as data propagation are stopped at the pin in any form.

- `set_clock_sense -logical_stop_propagation`

  You use the `-logical_stop_propagation` option to stop only the clock acting as a clock, but not the clock acting as data propagation. You might want the clock to continue to propagate forward, but in a way that PrimeTime does not analyze it as clock and only treats the clock as data.

Note:
The `timing_arc` object is supported only with the `-stop_propagation` option.

The difference between these two options is in whether the clock is physically stopped, that is the clock does not propagate beyond the stop pin at all; or the clock is logically stopped, that is the clock edges propagate beyond the stop pin, but only as data edges.

Figure 7-17 shows an example of physical clock stopping. In this example, the control logic is such that UMUX2 is allowed to select CLK2, but it never selects TESTCLK. In this case, TESTCLK never physically exists beyond UMUX2. To model this in PrimeTime, you should use the following command:

```
set_clock_sense -stop_propagation -clocks TESTCLK UMUX2/Z
```

Figure 7-17 Physical Clock Stopping

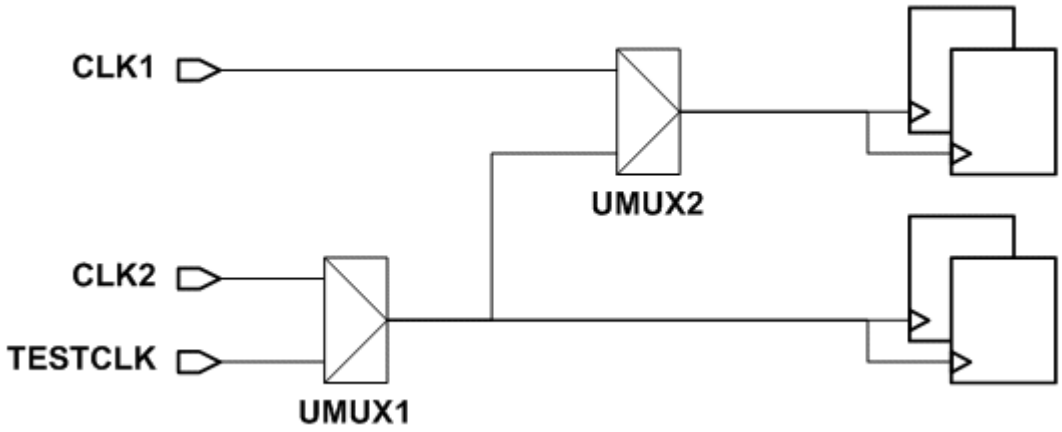
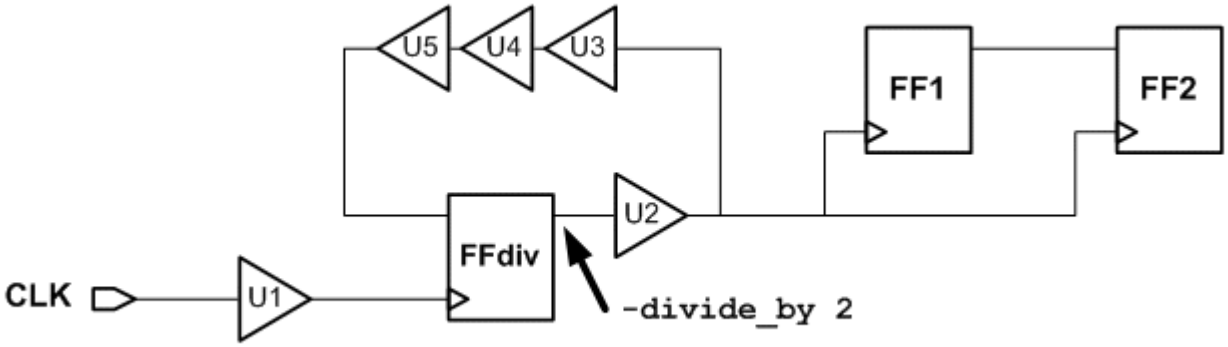


Figure 7-18 shows an example of logical clock stopping whereby the clock loops back around to the FFdiv divider input, and it is captured as data. It might be desirable to analyze the U3 through U5 buffers only as a data path. In this case, the clock could be logically stopped at the input of U3 by using the following command:

```
set_clock_sense -logical_stop_propagation -clocks CLK U3/A
```

Figure 7-18 Logical Clock Stopping



The clock signal continues to propagate to the FFdiv input, but only in a data capacity. Buffers U3 through U5 are not returned by the `get_clock_network_objects` command, and the clocks attribute of their pins does not show any active clock signals.

Using Pulse Clocks

A pulse clock consists of a sequence of short pulses whose rising and falling edges are both triggered by the same edge of another clock. Pulse clocks are often used to improve performance and reduce power consumption.

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

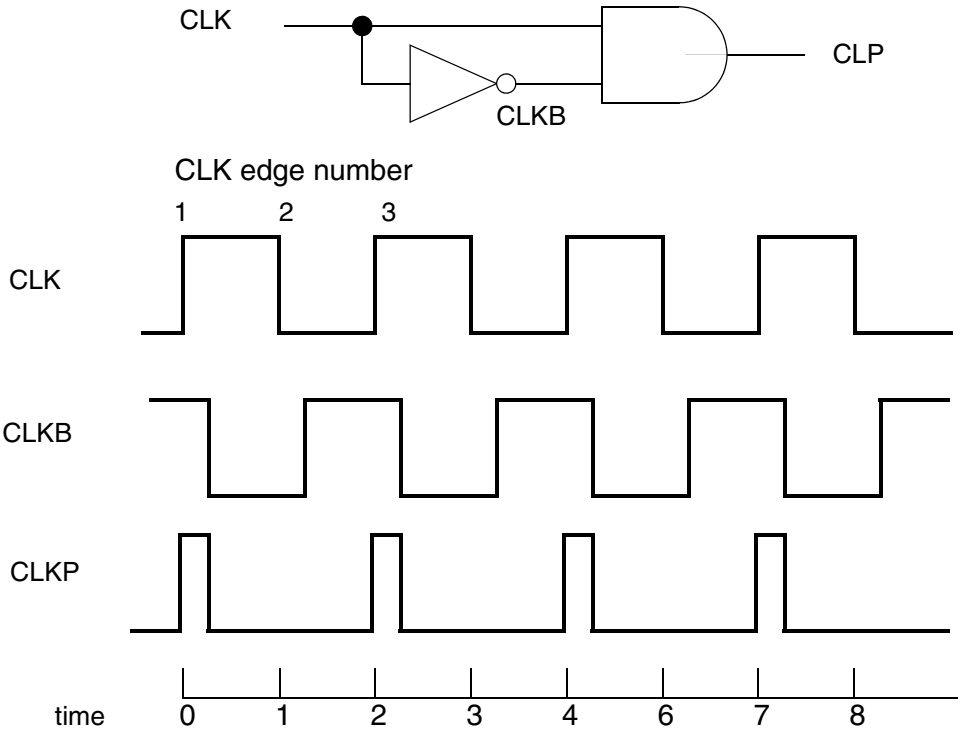
- Use a pulse generator cell that has been characterized with pulse generator attributes in the .lib description.
- Use the `create_generated_clock` command to describe the pulse timing with respect to the source clock.
- Use the `set_clock_sense` command to specify the sense of the generated pulses 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 in PrimeTime to specify the pulse clock characteristics. For information about specifying the pulse generator characteristics of a library cell, see the Library Compiler documentation.

If characterized pulse generator cells are not available in the library, you must specify the pulse clock characteristics at each pulse generation point in the design, using either the `create_generated_clock` or `set_clock_sense` command. Using the

`create_generated_clock` command creates a new clock domain at a pulse generation point. Using `set_clock_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 7-19. Each rising edge of CLK generates a pulse on CLKP.

Figure 7-19 Pulse Clock Specified as a Generated Clock



The same edge (edge number 1) of the source triggers both the rising and falling edges of the pulse clock. The pulse width is determined by the delay of the inverter.

To specify the generated pulse clock CLKP as a generated clock:

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

Specifying the generated clock as a pulse clock using repeated edge digits (rather than specifying the pulse clock edge times) ensures correct checking of delays between the source clock and the pulse clock.

In general, the position of the repeated digit determines whether an active-high or active-low pulse is generated, and the edge number that is repeated determines the type of edge in the master clock used to trigger the pulse:

- `-edges {1 1 3}` -- rising edge of source triggers high pulse
- `-edges {2 2 4}` -- falling edge of source triggers high pulse
- `-edges {1 3 3}` -- rising edge of source triggers low pulse
- `-edges {2 4 4}` -- falling edge of source triggers low pulse

Instead of using the `create_generated_clock` command to define a new clock, you can use the `set_clock_sense` command to specify the sense of the existing clock:

```
pt_shell> set_clock_sense -pulse rise_triggered_high_pulse \
           get+pins and2/z
```

This command tells PrimeTime that the clock at the output of the AND gate is a pulse that rises and falls on the rising edge of the source clock. The pulse clock is not defined as a separate clock domain. Instead, it is just a different sense (rise-triggered high pulse sense) of the source clock downstream from the specified point in the clock network.

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

```
set_clock_sense -pulse rise_triggered_high_pulse object_list
set_clock_sense -pulse rise_triggered_low_pulse object_list
set_clock_sense -pulse fall_triggered_high_pulse object_list
set_clock_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_clock_sense` command. To determine the actual pulse width, PrimeTime considers the different rise and fall latency values at the pulse generator output pin.

(high pulse width) = (fall network latency) – (rise network latency)

(low pulse width) = (rise network latency) – (fall network latency)

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

```
pt_shell> set_clock_latency -rise 0.6 and2/z
pt_shell> set_clock_latency -fall 1.1 and2/z
```

Constrain Minimum and Maximum Pulse Width in the Fanout of Pulse Generator Cells

PrimeTime provides the ability to constrain the pulse clock network. The transitive fanout of pulse generator is referred to as pulse clock network. The clock propagating through the pulse generator in the pulse clock network is referred to as the pulse clock. However, if the pulse generator has sequential arcs, its output is not a clock signal unless a generated clock is defined at the output. Use the `set_pulse_clock_min_width` and `set_pulse_clock_max_width` commands to constrain the pulse generator networks.

Use the `set_pulse_clock_min_width` command to constrain the minimum pulse width in the fanout of pulse generator instances by pulse generator instance name or pulse generator library cell. The minimum pulse width constraint from the library also applies to the pulse clock network. You can set minimum pulse width by using the following syntax:

```
set_pulse_clock_min_width -transitive_fanout value object_list
```

Note that if the CRPR is enabled, the clock reconvergence pessimism (CRP) value is applied to the pulse width as a credit. The CRP is subtracted from the pulse width for maximum pulse width calculation and added to the pulse width for minimum pulse width calculation. For more information, see the CRPR documentation.

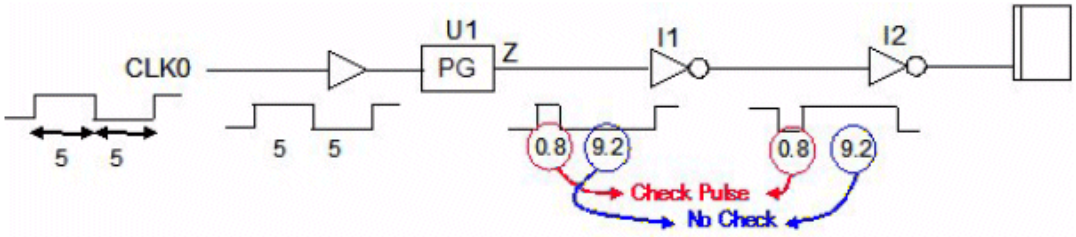
`set_pulse_clock_max_width`Use the command to constrain the maximum pulse width in the fanout of pulse generator instances by pulse generator instance name or pulse generator library cell. You can set maximum pulse width by using the following syntax:

```
set_pulse_clock_max_width -transitive_fanout value object_list
```

For either the maximum or minimum version of the command, if you constrain the pulse width by a clock, the constraint applies to the fanout of all pulse generators driven by that clock. If you constrain it by design, the constraint applies to all the pulse clock networks in the design. If there are conflicting constraints, the most restrictive constraint is used.

The pulse width constraint of high or low is inferred based on sense propagation. Figure 7-20 illustrates the sense propagation. For example, before the inverter, the minimum pulse width of the high pulse is constrained while after the inverter the minimum pulse width of the low pulse is constrained.

Figure 7-20 Sense Propagation



When removing either the maximum or minimum pulse width constraint you must explicitly specify all of the sources of the constraint. To remove the minimum pulse width constraint from the pulse generator network, you can use the `remove_pulse_clock_min_width` command. To remove the maximum pulse width constraint from the pulse generator network, you can use the `remove_pulse_clock_max_width` command.

You can use the `report_pulse_clock_min_width`, `report_pulse_clock_max_width`, `report_constraint -pulse_clock_min_width`, and `report_constraint -pulse_clock_max_width` commands, as appropriate, to report the pulse width of all pulse generator networks.

Pulse clock constraint checking is controlled by the `timing_enable_pulse_clock_constraints` variable. When this variable is set to `true` (the default), the general minimum pulse width constraints are checked everywhere, except the pulse clock network. More specific pulse clock constraints are used in the pulse clock network. If you set the `timing_enable_pulse_clock_constraints` variable to `false`, the more specific pulse clock constrains are ignored and the general minimum pulse width constraints are used for the design, including pulse clock networks. Note that during the `report_analysis_coverage` command, this variable is temporarily changed back to `false` to properly obtain design information. An informational message is also printed. For more information about these commands, see the specific man page.

Constrain Minimum and Maximum Transition at the Input Pin of Pulse Generator Cells and Maximum Transition at the Fanout of Pulse Generator Cells

PrimeTime provides support for constraining the minimum transition value at the input of a pulse generator instance. To constrain the minimum transition, use the following syntax:

```
set_pulse_clock_min_transition [-rise|-fall] value object_list
```

You can constrain the minimum transition by pulse generator instance name, pulse generator library cell, clock, or design. If the constraint is applied on a clock, the input of all the pulse generators in that clock network is constrained. If the constraint is applied to a design, all the pulse generator inputs are constrained. You can constrain minimum transition only at the input of pulse generators.

To constrain the maximum transition in the fanout of pulse generator instances, use the `set_pulse_clock_max_transition -transitive_fanout` command.

You can constrain this maximum transition by pulse generator instance name, pulse generator library cell, clock, or design. When the constraints are set on a clock, the fanout of all pulse generators driven by this clock are constrained. When the pulse clock maximum transition constraint is set on the design, all the pulse networks are constrained.

Note:

The maximum transition constraint set on design by using the `set_max_transition` command applies to the pulse network as does the maximum transition constraint specified on the library pins.

For both the maximum and minimum case, if the constraints are conflicting the most restrictive constraint are valid. You can separately constrain rise and fall transitions by using the `-rise` and `-fall` options.

The `set_pulse_clock_max_transition` command also allows you to specify the constraint that you want applied only to the input of pulse generators. The `-transitive_fanout` option specifies the constraint set at the transitive fanout of pulse generator. If this option is not set, only the input of the pulse generators are constrained.

You can remove the constraint from the input of pulse generator cells or from the pulse generator, using, respectively, the `remove_pulse_clock_min_transition` or `remove_pulse_clock_max_transition -transitive_fanout` commands.

Note:

To remove the constraint from the input of pulse generators, do not use the `-transitive_fanout` option.

To report the maximum transition computation, you can use the `report_pulse_clock_max_transition` and `report_constraint -pulse_clock_max_transition` commands. You can report the minimum transition computation at the input of all pulse generator networks by using the `report_pulse_clock_min_transition` and the `report_constraint -pulse_clock_min_transition` commands.

Note:

To report the maximum transition computation at the input of pulse generator cells, do not use the `-transitive_fanout` option.

To enable pulse clock constraint checking, set the `timing_enable_pulse_clock_constraints` variable to `true`. For more information about these commands, see the man page.

Timing PLL-Based Designs

Phase-locked loops (PLL) are common in high-speed designs. Their ability to nearly zero out the delay of a large clock tree allows for much higher speed interchip communication. Certain effects, such as OCV and signal integrity analysis,requires a complete and accurate static timing analysis of the design, including the PLL. The PLL reduces the clock skew at launch and capture flip-flops by making the phase of the clock at the feedback pin the same as the phase at the reference clock.

PrimeTime supports the analysis of PLL cells. The PLL library model contains the information regarding the reference clock pin, output pin, and feedback pin in the form of the attributes on the PLL cell pins. This information is used to perform additional error checking during the definition of the generated clock at the outputs of the PLL. For each PLL, you provide all of the relevant information regarding the reference, feedback, and output pins so that each PLL cell is identified. During the timing update, it automatically computes the timing of the feedback path and applies it as a phase correction on the PLL cell. This approach improves runtime and also simplifies the analysis script. This approach supports:

- Multiple PLLs
- PLLs with multiple output
- PLL jitter and long-term drift
- CRPR calculations for PLL paths
- PrimeTime SI analysis
- Sequential cells in the feedback path
- PLL adjustment during path-based analysis

Usage for PLL Timing

You create each PLL-generated clock by using the `create_generated_clock` command with the `-pll_feedback` and `-pll_output` options. Both these options are required when defining a generated clock at the output of a PLL. The `-pll_feedback` option indicates which PLL feedback pin is used for the clock's phase shift correction. The `-pll_output` option specifies the PLL clock output pin that propagates to the feedback pin. For single-output PLLs, the `-pll_output` pin is the same as the generated clock source pin. For multiple-output PLLs with a single shared feedback path, the `-pll_output` pin can differ from the source pin. When using these options to define a PLL, only certain options are available with the following restrictions applying:

- The `source_objects` option specifies a single pin object that corresponds to the clock output pin of the PLL being described.
- The `master_pin` option corresponds to the reference clock input pin of the PLL.
- The `feedback_pin` option corresponds to the feedback input pin of the PLL.
- The `output_pin` option corresponds to the output pin from which the feedback path starts. The `output_pin` option can be different from the pin in the `source_objects` option if the source pin's phase correction is determined by another clock output pin of the PLL.
- All four pins must belong to the same cell.

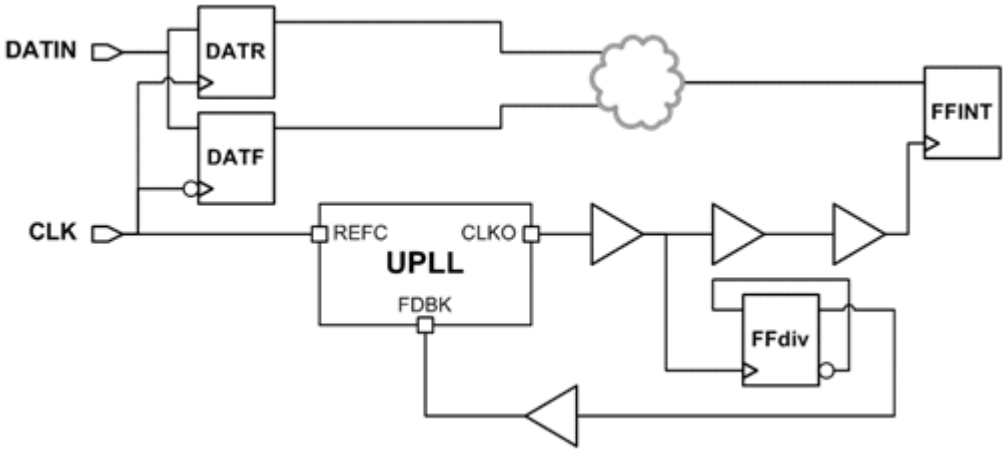
For the PLL adjustment to occur correctly, you should set both the PLL output clock and reference clock arriving at the reference clock pin of the PLL as propagated clocks. For more information about the `create_generated_clock` command, see the man page.

Sequential Cells on a Feedback Path

If the feedback path has any sequential elements on it, you need to define a generated clock on the output of the last sequential element. The master pin of this generated clock can be any pin that satisfies the following two conditions:

- The fanout of the output pin is connected to the feedback pin
- It lies on the feedback path

Figure 7-21 Example Circuit



Considering the circuit in Figure 7-21, the PLL output clock at pin CLKO is twice the frequency of the PLL reference clock at pin REFC. The sequential cell FFdiv acts as a clock divider and converts the frequency of the PLL output clock to that of the reference clock. To correctly define the PLL configuration, you need to define a generated clock at the output clk_out1 of the UPLL, using the following command:

```
create_generated_clock \
-name CLK_pll \
-source [get_pins UPLL/REFC] \
-pll_output [get_pins UPLL/CLKO] \
-pll_feedback [get_pins UPLL/FDBK] \
-multiply_by 2 \
[get_pins UPLL/CLKO]
```

In addition to the PLL output clock, you also need to define a clock divider at the output of the flop FFdiv, using the following command:

```
create_generated_clock \
-name pll_FDBK_clk \
-source [get_pins UPLL/CLKO] \
-divide_by 2 \
[get_pins FFdiv/Q]
```

This setup would allow PrimeTime to perform correct PLL adjustment.

PLL Drift and Jitter

The PLL drift and PLL jitter characteristics of the phase-corrected output clock are defined using the `set_clock_latency` command for the PLL output clock with the `-pll_shift` option. The presence of the `-pll_shift` option implies that the delay that is specified would be added to the base early or late latency of the PLL generated clock. You should note that this is different than the usual behavior of the `set_clock_latency` command, where

PrimeTime overrides the clock latency values with the specified values. When you specify this option, the delay value corresponds to the PLL drift. PLL jitter is specified using both the `-pll_shift` and `-dynamic` options. The `-pll_shift` option should be specified for the generated clock that is defined at the PLL output connected to the PLL feedback pin. PrimeTime applies the same shift to every output of the PLL. The following example shows the command syntax for PLL jitter:

```
set drift 0.100
set jitter 0.020

create_clock -period 5 [get_ports CLK]
create_generated_clock -name CLK_pll -divide_by 1 \
   -source UPLL/CLKIN \
   -pll_feedback UPLL/FDBK \
   -pll_output UPLL/CLKOUT
set_clock_latency -source -pll_shift [get_clocks CLK_pll] \
   -early -${drift} -dynamic -${jitter}
set_clock_latency -source -pll_shift [get_clocks CLK_pll] \
   -late +${drift} -dynamic +${jitter}
set_propagated_clock {CLK CLK_pll}
```

For more information, see the `set_clock_latency` command.

CRPR Calculations for PLL Paths

For PLLs with more than one output, PrimeTime considers all the output clocks to have the same phase. Thus, PrimeTime considers the outputs to be indistinguishable from each other with respect to clock reconvergence pessimism calculations. For example, for a PLL with two outputs, OUTCLK1 and OUTCLK2, and a path launched by a clock at OUTCLK1 and captured by a clock at OUTCLK2, PrimeTime removes the clock reconvergence pessimism up to the outputs of the PLL. PrimeTime does this, even though the last physical common pin on the launch and capture clock paths is the reference pin of the PLL. Removing the clock reconvergence pessimism is essential as the output clocks of the PLL have to be in phase with each other. The `report_crpr` command demonstrates this behavior by showing one of the PLL outputs as the common pin for paths launched and captured by different outputs of the PLL.

Reporting and Timing Checks

The `check_timing` command validates that a PLL clock reaches each feedback pin. This check is also performed when using the `update_timing` command.

The `report_timing` command includes the PLL adjustment next to the output of the PLL as follows:

```
...
clock CLK (rise edge)                              0.00          0.00
clock source latency                               0.00          0.00
clk_in (in)                                        0.00          0.00 r
inst_my_pll/REFCLK (my_pll)                        0.00          0.00 r
inst_my_pll/OUTCLK (my_pll) (gclock source)  -798.03 *      -798.03 r
b1/A (buf1a1)                                      0.00       -798.03 r
...
```

Requirements for PLL Library Cells

The PLL library cell should have a single positive unate timing arc from the reference clock pin to each of the outputs of the PLL. The reference pin, output clock pin, and feedback pin of the PLL are identified by `is_pll_reference_pin`, `is_pll_output_pin`, and `is_pll_feedback_pin` attributes, respectively. You can use the `is_pll_cell` cell attribute to identify a particular cell as a PLL cell. An example of a PLL library cell is as follows:

```
cell(my_pll) {
  is_pll_cell : true;

  pin( REFCLK ) {
direction : input;
is_pll_reference_pin : true;
  }

  pin( FBKCLK ) {
direction : input;
is_pll_feedback_pin : true;
  }

  pin (OUTCLK1) {
direction : output;
is_pll_output_pin : true;
    timing() {// Timing Arc
      related_pin: "REFCLK";
      timing_sense: positive_unate;
      cell_fall(scalar) {
        values("0.0")
      }
    }
  }
```

```
  pin (OUTCLK2) {
direction : output;
is_pll_output_pin : true;
    timing() {// Timing Arc
      related_pin: "REFCLK";
      timing_sense: positive_unate;
      cell_fall(scalar) {
        values("0.0")
      }
    }
  }
}
```

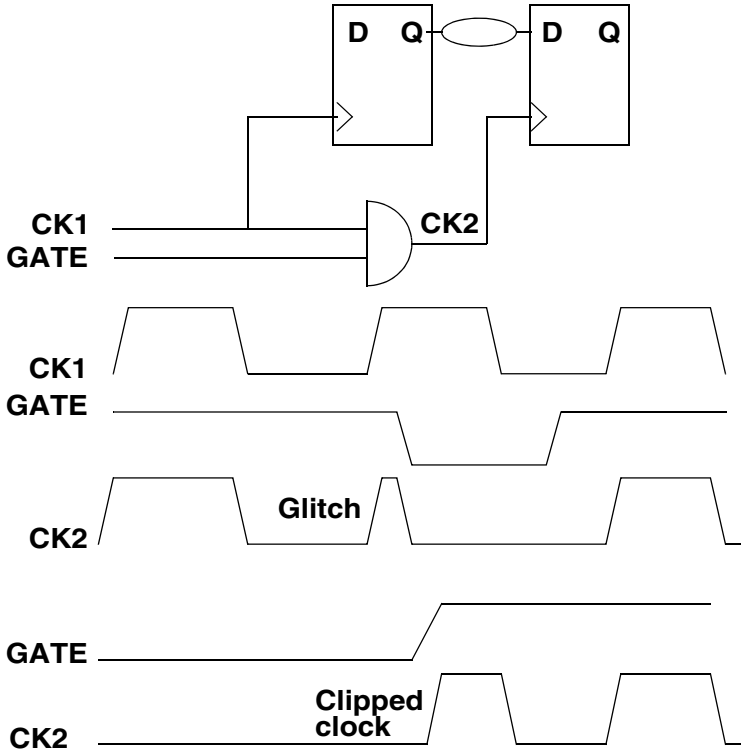
Specifying Clock-Gating Setup and Hold Checks

A gated clock signal occurs when the clock network contains logic other than inverters or buffers. For example, if a clock signal acts as one input to a logical AND function and a control signal acts as the other input, the output is a gated clock.

PrimeTime automatically checks for setup and hold violations on gating inputs to ensure that the clock signal is not interrupted or clipped by the gate. This check is performed only for combinational gates where one signal is a clock that can be propagated through the gate, and the gating signal is not a clock.

The clock-gating setup check ensures that the control data signal enables the gate before the clock becomes active. The arrival time of the leading edge of the clock pin is checked against both edges of any data signal feeding the data pins to prevent a glitch at the leading edge of the clock pulse or clipped clock pulse. The clock-gating setup violations are shown in Figure 7-22.

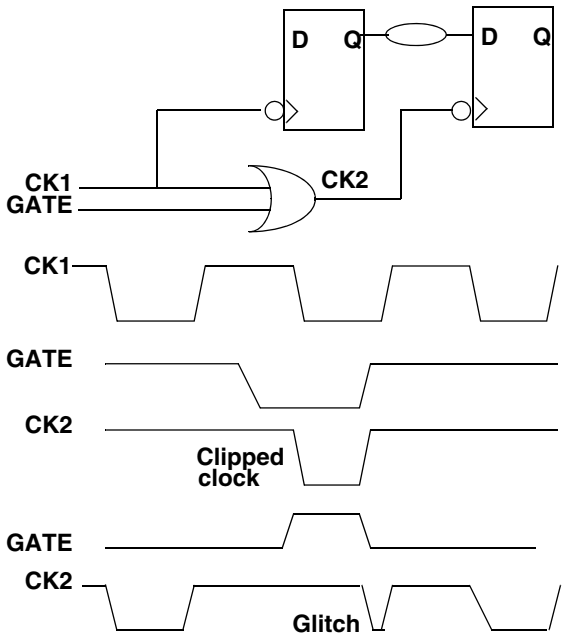
Figure 7-22 Clock-Gating Setup Violations



The clock-gating hold check ensures that the controll data signal remains stable while the clock is active. The arrival time of the trailing edge of the clock pin is checked against both edges of any data signal feeding the data pins using the hold equation. A clock-gating hold violation causes either a glitch at the trailing edge of the clock pulse or a clipped clock pulse. Note that for clock gating checks, it is required that the gating check must reach a clock pin

to succeed. These clock pins include reference pins of sequential timing constraints, from-pins of sequential arcs, and so on. The clock-gating hold violations are shown in Figure 7-23.

Figure 7-23 Clock-Gating Hold Violations



By default, PrimeTime checks setup and hold times for gated clocks. A value of 0.0 is set as the setup and hold time (unless the library cell for the gate has gating setup or hold timing arcs). You can specify a nonzero setup or hold value with the `set_clock_gating_check` command.

The `set_clock_gating_check` command affects only clock gating checks that exist at the specified cell or pin. To apply the clock gating parameters for an entire clock domain's network, specify a clock object by using the `set_clock_gating_check ... [get_clocks CLK]` command. For example, to specify a setup requirement of 0.2 and a hold requirement of 0.4 on all gates in the clock network of CLK1, enter

```
pt_shell> set_clock_gating_check -setup 0.2 -hold 0.4 [get_clocks CLK1]
```

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

```
pt_shell> set_clock_gating_check -setup 0.5 [get_cells and1]
```

The `report_clock_gating_check` command performs clock-gating setup and hold checks. Use it to identify any clock-gating violations.

```
pt_shell> report_clock_gating_check objects
```

To remove clock-gating checks set with the `set_clock_gating_check` command, use the `remove_clock_gating_check` command. For more information, see the specific man page.

Disabling or Restoring Clock-Gating Checks

You can specify any pin or cell in the current design or subdesigns to disable or restore clock-gating checks.

To disable clock-gating checks on cells or pins, use this command:

```
pt_shell> set_disable_clock_gating_check objects
```

To restore clock-gating checks on cells or pins, use this command:

```
pt_shell> remove_disable_clock_gating_check objects
```

If objects are not specified, all clock-gating checks are disabled. It is equivalent to setting the `timing_disable_clock_gating_checks` variable to `true`.

To restore disabled clock-gating checks for the object *U44*, enter

```
pt_shell> remove_disable_clock_gating_check U44
```

To disable clock-gating checks on the specified cell or pin, enter

```
pt_shell> set_disable_clock_gating_check U44/Z
```

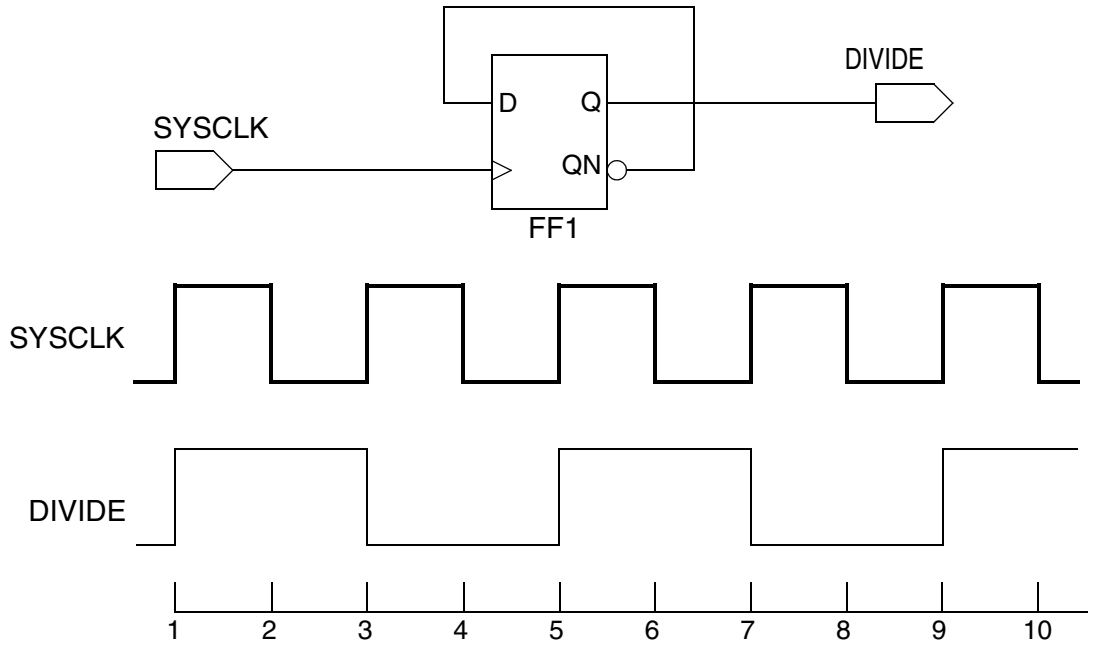
Specifying Internally Generated Clocks

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

Figure 7-24 shows the waveforms of the master and generated clock for a divide-by-2 clock generator. The clock waveform is ideal, with no clock-to-Q delay. PrimeTime does not consider the circuit logic of a clock generator, so you must specify the behavior of a

generated clock as a separate clock. However, you can define the relationship between the master clock and the generated clock, so that the generated clock characteristics change automatically when the master clock is changed.

Figure 7-24 Divide-by-2 Clock Generator



The `create_generated_clock` command specifies the characteristics of an internally generated clock. By default, using `create_generated_clock` on an existing generated clock object overwrites that clock. Generated clock objects are expanded to real clocks at the time of analysis. The clock expansion happens automatically within the `report_timing` command or explicitly with the `update_timing` command.

You can create the generated clock as a frequency-divided clock (`-divide_by` option), frequency-multiplied clock (option), or edge-derived clock (`-multiply_by-edges` option). You can modify the generated clock waveform by specifying either `-multiply_by`, `-divide_by`, or `-edges` with the `-combinational` option. When you create the generated clock using the `-combinational` option, there must be a valid path for propagating the rise and fall edges of master clock to the generated clock source pin and the source latency paths for this type of generated clock only includes the logic where the master clock propagates. For more information, see the `create_generated_clock` man page.

Specifying a Divide-by-2 Generated Clock

To specify a divide-by clock, use the `-divide_by` option of the `create_generated_clock` command and specify the frequency division factor. For example, to create the divide-by-2 generated clock in Figure 7-24, shown previously, specify 2 as the frequency division factor:

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

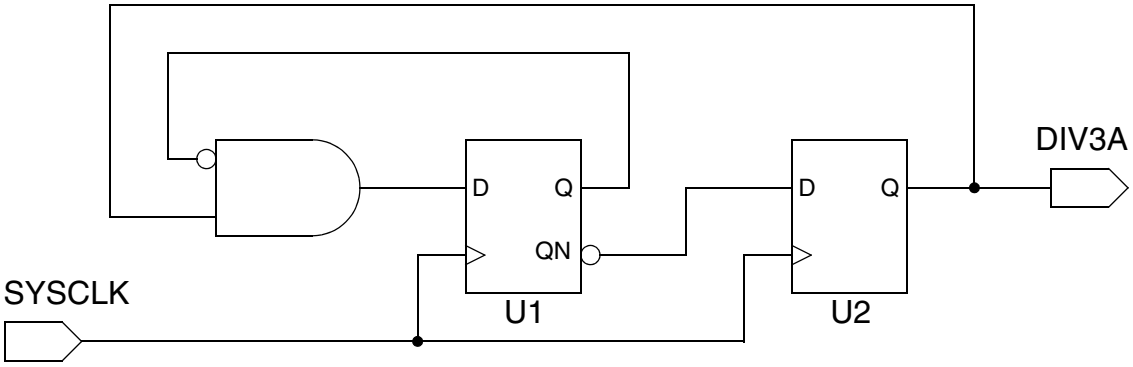
Specify a port or a pin (not a clock name) as the master source from which the new clock is generated. Specify a pin as the creation point for the new generated clock.

Note that generated clock edges are based on occurrences of rising edges at the master clock source pin (specified with the `-source` option). If you need to create a generated clock based on falling edges at the master clock source pin, use the `-edges` option rather than the `-divide_by` option (see “Creating a Divide-by Clock Based on Falling Edges” on page 7-49).

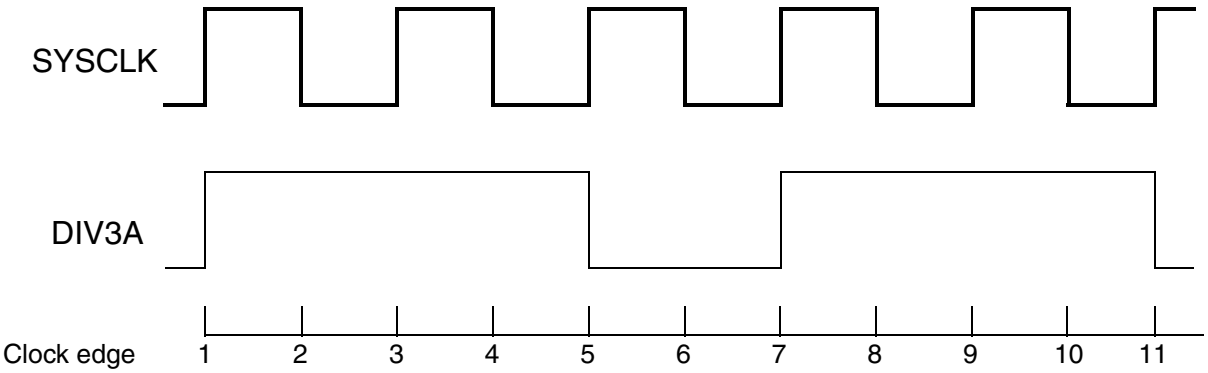
Creating a Generated Clock Based on Edges

You can use the `create_generated_clock -edges` command to specify the generated clock in terms of edges of the master clock waveform on the master pin. For example, consider the clock generator circuit shown in Figure 7-25.

Figure 7-25 Divide-by-3 Clock Generator




```
create_clock -period 2 -name SYSCLK [get_ports SYSCLK]
```



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

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

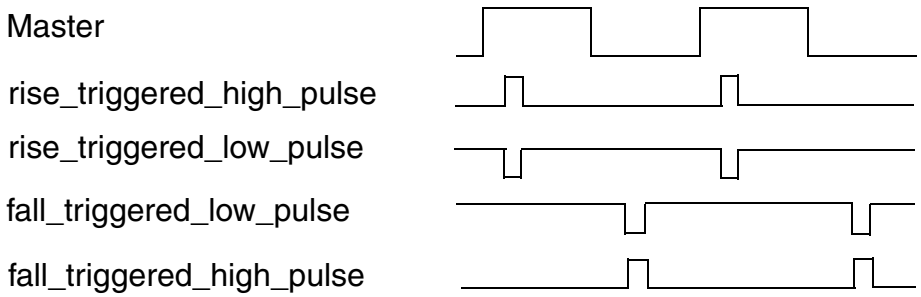
Creating Clock Senses for Pulse Generators

Pulse generators can improve circuit performance and save power consumption. Because of clock gating, skew management, and preserving generated pulse integrity, it might be necessary to insert thousands of pulse generators in a design to control the pulse. Although you can specify `create_generated_clock` with nondecreasing edges at the output of each pulse generator, it does not scale well. Not only do you have to find and specify each generated clock, the proliferation of clock groups makes it hard to use. Therefore, for pulse generators that do not change the frequency of the incoming clock, you can set the following clock senses with the `pulse_clock` attribute to support the pulse generators:

- `rise_triggered_high_pulse`
- `rise_triggered_low_pulse`
- `fall_triggered_high_pulse`
- `fall_triggered_low_pulse`

Setting these clock senses supports the four types without the need to add a generated clock.

Figure 7-26 Pulse Clock Types That Do Not Change Frequency



This pin is allowed on internal, bidirectional, or output pins. If you set it on a bidirectional pin, it affects only those clock paths that follow through the output side of that pin. The ideal clock width for clocks with any of the pulse clock senses is 0. The pulse width is computed as:

```
high pulse = fall_network_latency - rise_network_latency
low pulse = rise_network_latency - fall_network_latency
```

Use the `set_clock_latency` command to set the ideal clock pulse width. You can add the command to any pin in the clock network that affects all registers in the fanout of the command. For example, to set an ideal pulse high width of 0.5 for all registers downstream from pin PG/Z and with an overall latency of 0.6, you would use the following syntax:

```
set_clock_latency -rise 0.6 PG/Z
set_clock_latency -fall 1.1 PG/Z
```

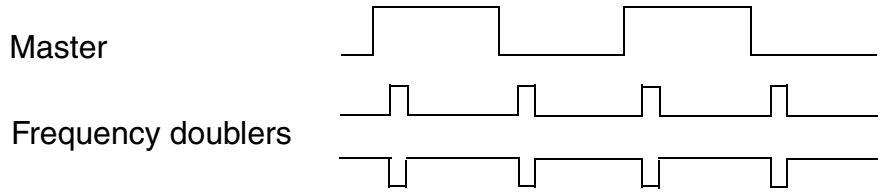
If your waveform is non-monotonic, yet has increasing values, use the `create_clock -waveform` command. This instructs PrimeTime to create the correct clock for those clocks driven by a pulse generator. For example,

```
create_clock -waveform {0.0 0.0} -period 10 [get_port BCK]
set_clock_latency -source -rise 0.23 -fall 0.65 [get_port BCK]
```

PrimeTime generates an error for any clock specified as a pulse generator if it combines with another sense of the same clock. To specify the clock sense in those locations where multiple senses of the same clock have merged together and it is ambiguous which sense PrimeTime should use, use the `set_clock_sense` command. Specify the type of pulse clock sense you want using the `set_clock_sense -pulse` command, specifying the value as `rise_triggered_high_pulse`, `rise_triggered_low_pulse`, `fall_triggered_high_pulse`, or `rise_triggered_low_pulse`.

As previously mentioned, you can handle pulse generators that alter frequency by using the `create_generated_clock` command. For example, this approach works well in the case of frequency doublers, as shown in Figure 7-27.

Figure 7-27 Pulse Clock Types That Change Frequency

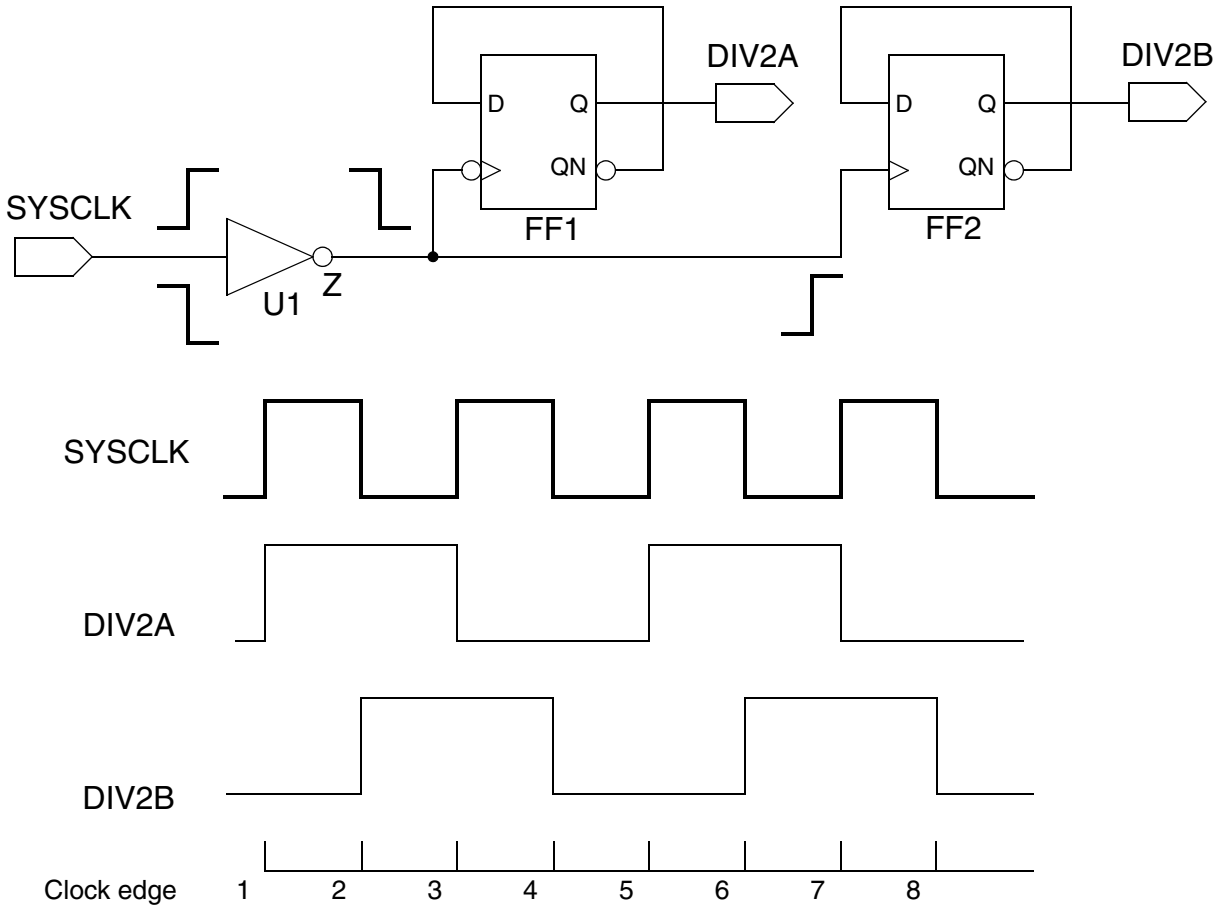


If the requested sense of clock does not exist, PrimeTime issues a warning. For example, if there was only a positive unate path from the clock source to a pin with `set_clock_sense -pulse rise_triggered_high_pulse`, PrimeTime would not run the command as no rising at the clock source to failing at the clock sense pin path is found. However, if there are both a positive and negative unate path from the clock source to a pin, PrimeTime would run this same command with the rise latency coming from the positive unate path and the fall latency coming from the negative unate path.

Creating a Divide-by Clock Based on Falling Edges

If you need to create a generated clock based on falling edges at the master clock pin, use the `create_generated_clock` command with the `-edges` option. The generated clock examples in Figure 7-28 demonstrate how to do this.

Figure 7-28 Generated Divide-by-2 Clocks Based on Different Edges



The generated clock DIV2A is based on the rising edge of the master clock at the SYSCLK port, so you can specify the generated clock using either the `-divide_by` option or the `-edges` option:

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

pt_shell> create_generated_clock -name DIV2A \
             -source [get_ports SYSCLK] -edges { 1 3 5 } [get_pins FF1/Q]
```

The generated clock DIV2B is based on the falling edge of the master clock at the SYSCLK port, so you cannot use the `-divide_by` option. However, you can still use the `-edges` option:

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

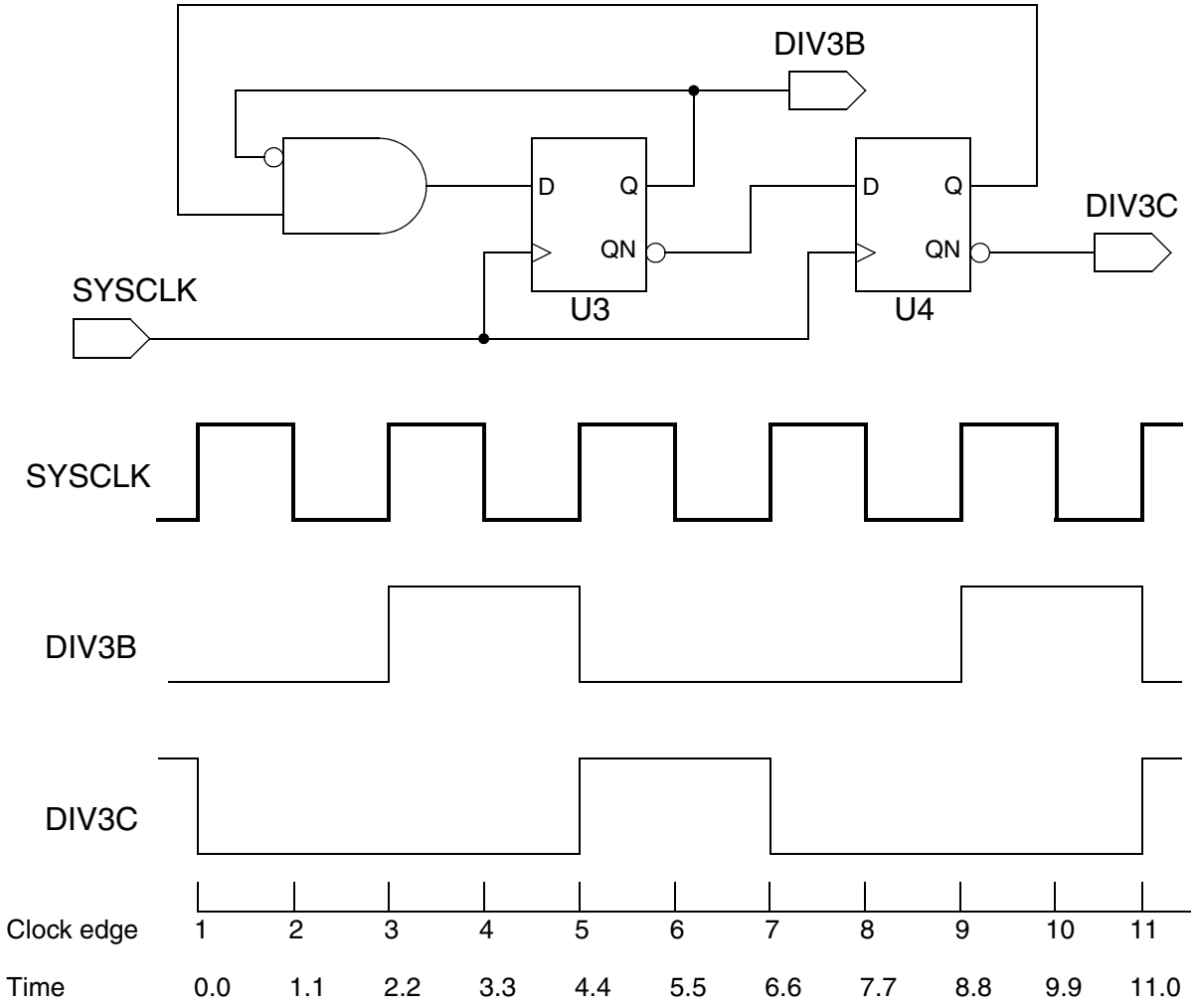
Another way to specify DIV2B is to use a different source pin:

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

Shifting the Edges of a Generated Clock

You can shift the edges of a generated clock by a specified amount of time. This shift is not considered clock latency. For example, consider the clock generator circuit shown in Figure 7-29.

Figure 7-29 Generated Divide-by-3 Clock With Shifted Edges



To specify the master source clock and the two generated clocks, you could use commands such as the following:

```
pt_shell> create_clock -period 2.2 -name CLK [get_ports SYSCLK]

pt_shell> create_generated_clock -edges { 3 5 9 } \
           -name DIV3B -source [get_ports SYSCLK] [get_pins U3/Q]
```

```
pt_shell> create_generated_clock -edges { 3 5 9 } \
          -edge_shift { 2.2 2.2 2.2 }  \
          -name DIV3C -source [get_ports SYSCLK] [get_pins U4/QN]
```

Note:

You can use the `-edge_shift` option only in conjunction with the `-edges` option.

The `report_clock` command reports the specified clock as follows,

```
    p - propagated_clock
    G - Generated clock

Clock           Period   Waveform        Attrs     Sources
-------------------------------------------------------
-
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     Master       Generated    Waveform
Clock         Source       Source       Modification
-----------------------------------------------------------
-
DIV3B         MYCLK        U3/Q         edges( 3 5 9 )
DIV3C         MYCLK        U4/Q         edges( 3 5 9 )
                                        shifts( 2.2 2.2 2.2 )
```

Multiple Clocks at the Source Pin

The `create_generated_clock` command defines a clock that depends on the characteristics of a clock signal reaching a pin in the design, as specified by the `-source` argument. Because it is possible for the source pin to receive multiple clocks, you might need to specify which clock is used to create the generated clock. To specify a clock, use the `-master_clock` option together with the `-source` option in the `create_generated_clock` command.

To get information about a generated clock, use the `report_clock` command. The report tells you the name of the master clock, the name of the master clock source pin, and the name of the generated clock pin.

Selecting Generated Clock Objects

The `get_clocks` command selects a clock object. You can restrict the clock selection to certain clocks according to naming conventions or by filtering. This command returns a token that represents a collection of clocks whose names match the pattern and whose attributes pass a filter expression. For example, if all generated clock names match the pattern CLK_DIV*, you can do the following:

```
pt_shell> set_false_path \
          -from [get_clocks CLK_DIV*] \
          -to [get_clocks CLKB]
```

If the generated clocks do not follow a naming pattern, you can use filtering based on attributes instead. For example,

```
pt_shell> set_false_path \
          -from [get_clocks CLK* \
          -filter "is_generated==TRUE"] \
          -to [get_clocks CLKB]
```

Reporting Clock Information

The `report_clock` command shows information about clocks and generated clocks in the current design. For a detailed report about a clock network, use the `report_clock_timing` command as described in “Clock Network Timing Report” on page 13-34.

The `report_transitive_fanout -clock_tree` command shows the clock networks in your design.

Removing Generated Clock Objects

The `remove_generated_clock` command removes generated clocks. For more information, see the `remove_generated_clock` man page. To remove the generated clock named CLK_DIV2 (and its corresponding clock object), enter

```
pt_shell> remove_generated_clock CLK_DIV2
```

Generated Clock Edge Specific Source Latency Propagation

PrimeTime calculates the clock source latency for a generated clock taking into account the edge relationship between master clock and generated clock. It finds the worst-case path that propagates forward to produce the specified edge type defined by the generated clock with respect to the master clock source. This produces a more accurate, less pessimistic value for the generated clock source latency.

For increased accuracy of this analysis, the `timing_edge_specific_source_latency` variable is set to `true` by default.

If paths exist, but they do not satisfy the needed edge relationships, PrimeTime returns a error message (UITE-461) stating that no path is reported for generated clock source latency (zero clock source latency is used). The following is an example of an unsatisfied generated clock returning a UITE-461 error message:

```
Error: Generated clock 'clk_div2' 'rise_edge' is not
satisfiable; zero source latency will be used. (UITE-461)
Error: Generated clock 'clk_div2' 'fall_edge' is not
satisfiable; zero source latency will be used. (UITE-461)

Path Type: min

Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk_div2 (rise edge)              0.000      0.000
  clock source latency                    0.000      0.000
  Udiv/Q (FD1)                            0.000      0.000 r

  clock clk_div2 (rise edge)              0.000      0.000
  clock source latency                    0.000      0.000
  Udiv/Q (FD1)                            0.000      0.000 r
  ….
```

8

Timing Analysis Conditions

A timing analysis by PrimeTime depends on the conditions that you specify. These can include such conditions as input delays, output delays, port capacitance, wire load models, and operating conditions.

Specifying and using these conditions are described in the following sections:

- Input Delays
- Output Delays
- Drive Characteristics at Input Ports
- Port Capacitance
- Wire Load Models
- Operating Conditions
- Slew Propagation
- Design Rules

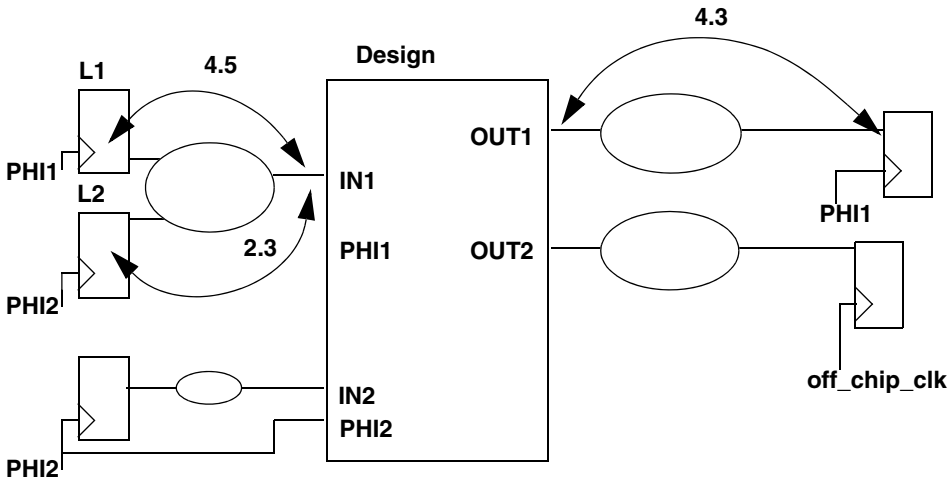
Input Delays

The `set_input_delay` command specifies the timing of external paths leading to an input port. An input delay is specifying an arrival time at an input port relative to a clock edge.

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 8-1.

Figure 8-1 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:

```
pt_shell> set_input_delay 4.5 -clock PHI1 {IN1}
```

Example 2

If paths from multiple clocks or edges reach the same port, specify each one using the `-add_delay` option. If you omit the `-add_delay` option, existing data is removed. For example,

```
pt_shell> set_input_delay 2.3 -clock PHI2 -add_delay {IN1}
```

If the source of the delay is a level-sensitive latch, use the `-level_sensitive` option. This allows PrimeTime to determine the correct single-cycle timing constraint for paths from this port. Use the `-clock_fall` option to denote a negative level-sensitive latch; otherwise, the `-level_sensitive` option implies a positive level-sensitive latch.

To see input delays on ports, use the `report_port -input_delay` command.

To remove input delay information from ports or pins in the current design set using the `set_input_delay` command, use the `remove_input_delay` command. The default is to remove all input delay information in the `port_pin_list` option.

Using Input Ports Simultaneously for Clock and Data

PrimeTime allows an input port to behave simultaneously as a clock and data port. You can use the `timing_simultaneous_clock_data_port_compatibility` variable to enable or disable the simultaneous behavior of the input port as a clock and data port. When this variable is `false`, the default, simultaneous behavior is enabled and you can use the `set_input_delay` command to define the timing requirements for input ports relative to a clock. In this situation, the following applies:

- If you specify the `set_input_delay` command relative to a clock defined at the same port and the port has data sinks, the command is ignored and an error message is issued. There is only one signal coming to port, and it cannot be at the same time data relative to a clock and the clock signal itself.
- If you specify the `set_input_delay` command relative to a clock defined at a different port and the port has data sinks, the input delay is set and controls data edges launched from the port relative to the clock.
- Regardless of the location of the data port, if the clock port does not fanout to data sinks, the input delay on the clock port is ignored and you receive an error message.

When you set the `timing_simultaneous_clock_data_port_compatibility` variable to `true`, the simultaneous behavior is disabled and the `set_input_delay` command defines the arrival time relative to a clock. In this situation, when an input port has a clock defined on it, PrimeTime considers the port exclusively as a clock port and imposes restriction on the data edges that are launched. PrimeTime also prevents setting input delays relative to another clock.

To control the clock source latency for any clocks defined on an input port, you must use the `set_clock_latency` command.

Output Delays

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.

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

To remove output delay from output ports or pins set through the `set_output_delay` command, use the `remove_output_delay` command. By default, all output delays on each object in the port or pin list are removed. To restrict the removed output delay values, use the `-clock`, `-clock_fall`, `-min`, `-max`, `-rise`, or `-fall` option.

Example

To set an output delay of 4.3 relative to the rising edge of clock PHI1 on port OUT1 (see Figure 8-1 on page 8-2):

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

Drive Characteristics at Input Ports

You need to define the drive capability of the external cell driving each input port. PrimeTime uses this information to calculate the load-dependent cell delay for the port and to produce an accurate transition time for calculating cell delays and transition times for the following logic stages.

The `set_driving_cell` command can specify a library cell arc for the driving cell so that timing calculations are accurate even if the capacitance changes. This command causes the port to have the transition time calculated as if the given library cell were driving the port.

For less precise calculations, you can use the `set_drive` or `set_input_transition` command. The most recent drive command has precedence. If you issue the `set_drive` command on a port and then use the `set_driving_cell` command on the same port, information from the `set_drive` command is removed.

Setting the Port Driving Cell

The `set_driving_cell` command directs PrimeTime to calculate delays as though the port were an instance of a specified library cell. The port delay is calculated as a cell delay that consists of only the load-dependent portion of the port.

The transition time for the port is also calculated as though an instance of that library cell were driving the net. The delay calculated for a port with information from the `set_driving_cell` command takes advantage of the actual delay model for that library cell, whether it is nonlinear or linear. The input delay specified for a port with a driving cell or drive resistance should not include the load-dependent delay of the port.

To display port transition or drive capability information, use the `report_port` command with the `-drive` option.

With the `set_driving_cell` command you can specify the input rise and fall transition times for the input of a driving cell using the `-input_transition_rise` or the `-input_transition_fall` option. If no input transition is specified, the default is 0. For more information, see the man page.

Setting the Port Drive Resistance

The `set_drive` command defines the external drive strength or resistance for input and inout ports in the current design. In the presence of wire load models, the transition time and delay reported for the port are equal to Rdriver * Ctotal. PrimeTime uses this transition time in calculating the delays of subsequent logic stages.

You can use the `set_drive` command to set the drive resistance on the top-level ports of the design when the input port drive capability cannot be characterized with a cell in the technology library. However, this command is not as accurate for nonlinear delay models compared to the `set_driving_cell` command. The `set_drive` command is useful when you cannot specify a library cell (for example, when the driver is a custom block not modeled as a Synopsys library cell). For more information, see the man page.

Setting a Fixed Port Transition Time

The `set_input_transition` command defines a fixed transition time for input ports. The port has zero cell delay. PrimeTime uses the specified transition time only in calculating the delays of logic driven by the port.

A fixed transition time setting is useful for

- Comparing the timing of a design to another tool that supports only input transition time.
- Defining transition for ports at the top level of a chip, where a large external driver and a large external capacitance exist. In this case, the transition time is relatively independent of capacitance in the current design.

For more information, see the man page.

Displaying Drive Information

To display drive information, you can use the `report_port` command with the `-drive` option. For more information, see the man page.

Removing Drive Information From Ports

The commands shown in Table 8-1 remove drive information from ports.

Table 8-1 Commands to Remove Drive Information

To remove this	Use this
Driving cell information from a list of ports	`remove_driving_cell`
Drive resistance	`set_drive 0.0`
Input transition	`set_input_transition 0.0`
Drive data and all user-specified data, such as clocks, input and output delays	`reset_design`

Port Capacitance

The `set_load` command sets load capacitance on ports. To accurately time a design, you need to describe the external load capacitance of nets connected to top-level ports, including pin capacitance and wire capacitance. For more information, see the `set_load` man page.

Example 1

To specify the external pin capacitance of ports, enter

```
pt_shell> set_load -pin_load 3.5 {IN1 OUT1 OUT2}
```

You also need to account for wire capacitance outside the port. For prelayout, specify the external wire load model and the number of external fanout points. This process is described in "Setting Wire Load Models Manually" on page 8-7.

Example 2

For post-layout, specify the external annotated wire capacitance as wire capacitance on the port. For example, enter

```
pt_shell> set_load -wire_load 5.0 {OUT3}
```

To remove port capacitance values, use the `remove_capacitance` command.

Wire Load Models

A wire load model attempts to predict the capacitance and resistance of nets in the absence of placement and routing information. The estimated net capacitance and resistance are used for delay calculation. Technology library vendors supply statistical wire load models to support estimating wire loads based on the number of fanout pins on a net. You can set wire load models manually or automatically.

Setting Wire Load Models Manually

The `set_wire_load_model` command manually sets a named wire load model on a design, instance, list of cells, or list of ports.

For example, consider this design hierarchy:

```
TOP
  MID (instance u1)
    BOTTOM (instance u5)
  MID (instance u2)
    BOTTOM (instance u5)
```

To set a model called 10x10 on instances of BOTTOM, a model called 20x20 on instances of MID, and a model called 30x30 on nets at the top level, use these commands:

```
pt_shell> set_wire_load_mode enclosed

pt_shell> set_wire_load_model -name 10x10 \
          [all_instances BOTTOM]

pt_shell> set_wire_load_model -name 20x20 \
          [all_instances MID]

pt_shell> set_wire_load_model -name 30x30
```

To capture the external part of a net that is connected to a port, you can set an external wire load model and a number of fanout points. For example, to do this for port Z of the current design:

```
pt_shell> set_wire_load_model -name 70x70 [get_ports Z]
pt_shell> set_port_fanout_number 3 Z
```

To calculate delays, PrimeTime assumes that port Z has a fanout of 3 and uses the 70x70 wire load model.

To see wire load model settings for the current design or instance, use the `report_wire_load` command. To see wire load information for ports, use the `report_port` command with `-wire_load` option. To remove user-specified wire load model information, use the `remove_wire_load_model` command.

Automatic Wire Load Model Selection

PrimeTime can set wire loads automatically when you update the timing information for your design. If you do not specify a wire load model for a design or block, PrimeTime automatically selects the models based on the wire load selection group, if specified.

If you do not apply a wire load model or selection group but the library defines a `default_wire_load` model, PrimeTime applies the library-defined model to the design. Otherwise, the values for wire resistance, capacitance, length, and area are all 0.

Automatic wire load selection is controlled by selection groups, which map the block sizes of the cells to wire load models. If you specify the `set_wire_load_selection_group` command on the top design, or if the main technology library defines a `default_wire_load_selection_group`, PrimeTime automatically enables wire load selection.

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

When wire load selection is enabled, the wire load is automatically selected for hierarchical blocks larger than the minimum cell area, based on the cell area of the block.

To set the minimum block size for automatic wire load selection, enter

```
pt_shell> set_wire_load_min_block_size size
```

In this command, *size* is the minimum block size for automatic wire load selection, in library cell area units. The specified size must be greater than or equal to 0.

The auto_wire_load_selection environment variable specifies automatic wire load selection. The default setting is true, enabling automatic wire load selection if a selection group is associated with the design. To disable automatic wire load selection, enter

```
pt_shell> set auto_wire_load_selection false
```

To remove the wire load selection group setting, use the remove_wire_load_selection_group command.

Setting the Wire Load Mode

The current wire load mode setting, which can be set with the set_wire_load_mode command, determines the wire load models used at different levels of the design hierarchy. There are three possible mode settings: top, enclosed, or segmented.

If the mode for the top-level design is top, the top-level wire load model is used to compute wire capacitance for all nets within the design, at all levels of hierarchy. If the mode for the top-level design is either enclosed or segmented, wire load models on hierarchical cells are used to calculate wire capacitance, resistance, and area for nets inside these blocks.

If the enclosed mode is set, PrimeTime determines net values using the wire load model of the hierarchical cell that fully encloses the net. If the segmented mode is set, PrimeTime separately determines net values for segments of the net in each level of hierarchy, and then obtains the total net value from the sum of all segments of the net.

If you do not specify a mode, the default_wire_load_mode setting of the main technology library is used. The enclosed mode is often the most accurate. However, your ASIC vendor might recommend a specific mode. For more information, see the set_wire_load_mode man page.

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

```
pt_shell> set_wire_load_mode enclosed
```

The design in Figure 8-2 has the following wire load models set:

```
set_wire_load_model -name Big (the current design)set_wire_load_model -
name Medium (instances U1 and U2)set_wire_load_model -name Small
(instance U2/U1)
```

Figure 8-2 Example Design for Wire Load Mode Settings

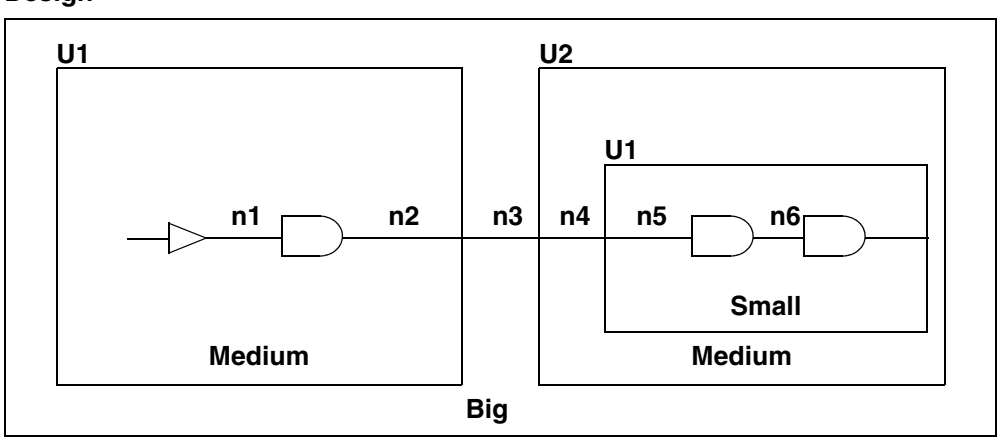


Table 8-2 lists the resulting wire load modes and models that apply to the nets in Figure 8-2.

Table 8-2 Wire Load Models

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

Reporting Wire Load Models

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

- `report_wire_load`
- `report_port -wire_load`

For more information, see the man pages for these commands.

Operating Conditions

Integrated circuits exhibit different performance characteristics for different operating conditions: fabrication process variations, power supply voltage, and temperature. The technology library defines nominal values for these parameters and specifies delay information under those conditions. A set of operating conditions contains the following values:

Operating Condition	Description
Process derating factor	This value is related to the scaling of device parameters resulting from variations in the fabrication process. A process number less than the nominal value usually results in smaller delays.
Ambient temperature	The chip temperature affects device delays. The temperature of the chip depends on several factors, including ambient air temperature, power consumption, package type, and cooling method.
Supply voltage	A higher supply voltage usually results in smaller delays.
Interconnect model type	This value defines an RC tree topology that PrimeTime uses to estimate net capacitance and resistance during prelayout analysis.

The delay information for each timing arc is specified at nominal process, temperature, and voltage conditions. If your operating conditions are different from this, PrimeTime applies scaling factors to account for the variations in these conditions. Many libraries use linear scaling for process, temperature, and voltage.

If the technology library contains scaled cell information, you can include the exact delay tables or coefficients for specific operating conditions. This method can be very accurate for library cells that do not scale linearly. For more information, see the Library Compiler and Design Compiler reference manuals.

You can use a single set of operating conditions to do analysis (for setup and hold) or you can specify minimum and maximum conditions. If you do not set operating conditions on your design, PrimeTime uses the default set of operating conditions if the main library contains them, or the nominal values of the main library. See Chapter 12, “Operating Conditions,” for more information.

Interconnect Model Types

PrimeTime uses interconnect model information when it calculates net delays for prelayout designs, when annotated net delays and parasitic information are not available. Two nets with the same total resistance and capacitance, but different RC tree topologies, can have different pin-to-pin delays.

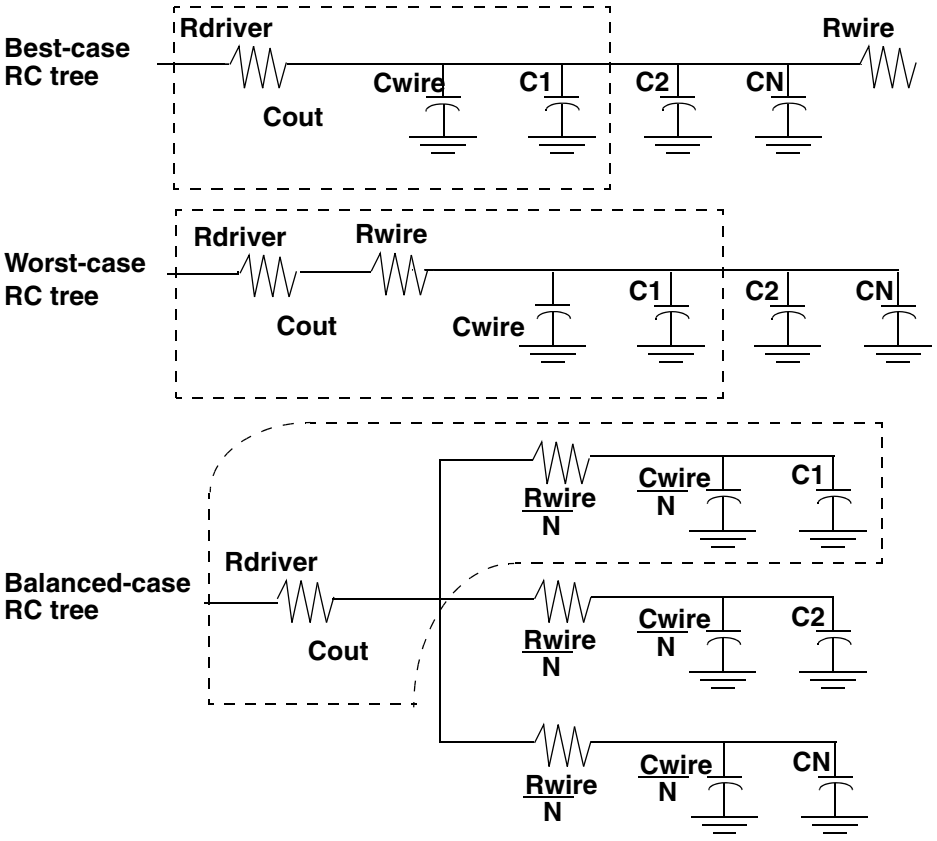
This section provides background information about interconnect model types. You cannot modify the types in PrimeTime. For more information, see the Library Compiler reference manuals.

The interconnect model is defined by the `tree_type` specification in each technology library's set of operating conditions. A `tree_type` specification indicates the type of wire resistance and capacitance topology: `best_case_tree`, `worst_case_tree`, or `balanced_tree`. For example,

```
operating_conditions(BEST) {
      process     : 1.1;
      temperature : 11.0;
      voltage     : 4.6;
      tree_type   : ”best_case_tree”;
 }
operating_conditions(TYPICAL) {
      process     : 1.3;
      temperature : 31.0;
      voltage     : 4.6;
      tree_type   : ”balanced_tree”;
 }
operating_conditions(WORST) {
     process      : 1.7;
     temperature  : 55.0;
     voltage      : 4.2;
     tree_type    : ”worst_case_tree”;
}
```

If the technology library does not define the tree type, PrimeTime uses the `balanced_tree` model. Figure 8-3 shows the tree type model networks.

Figure 8-3 RC Interconnect Topologies for Fanout of N



Setting Operating Conditions

The `set_operating_conditions` command sets process, temperature, and voltage conditions for timing analysis.

The operating conditions you specify must be defined in a specified library or a library in the link path. You can create custom operating conditions for a library with the `create_operating_conditions` command. Use the `report_lib` command to get a list of the available operating conditions in a technology library before you use the `set_operating_conditions` command. For more information, see the `set_operating_conditions` man page.

To set WCCOM from the tech_lib library as a single operating condition, enter

```
pt_shell> set_operating_conditions WCCOM -library tech_lib
```

To set WCCOM as the maximum condition and BCCOM as the minimum condition, enter

```
pt_shell> set_operating_conditions -analysis_type bc_wc \
            -min BCCOM -max WCCOM
```

Because you do not specify a library, PrimeTime searches all libraries in the link path. After you set the operating conditions, you can report or remove operating conditions.

Creating Operating Conditions

A technology library contains a fixed set of operating conditions. You can use the `create_operating_conditions` command to create new operating conditions in a library you specify and use these custom operating conditions to analyze your design during the current session. You cannot write these operating conditions to a library .db file.

To see the operating conditions defined for a library, use the `report_lib` command. To set operating conditions on the current design, use the `set_operating_conditions` command.

To create a new operating condition called WC_CUSTOM in the library `tech_lib`, enter

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

Operating Condition Information

These commands report, remove, or reset operating condition information.

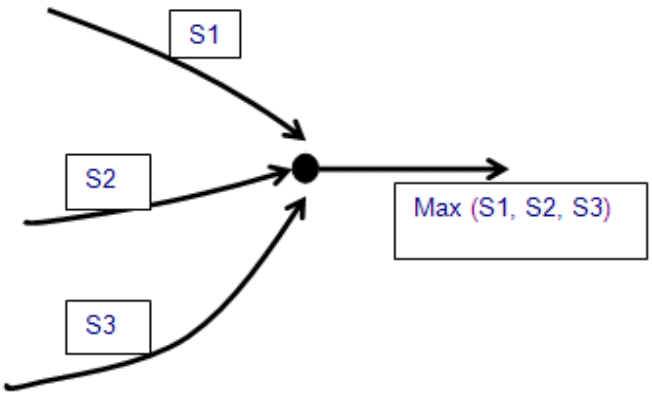
Action	Use this command
List the operating condition settings for the design.	`report_design`
Remove operating conditions from the current design.	`remove_operating_conditions`
Reset operating conditions to the default and remove all user-specified data, such as clocks, input and output delays.	`reset_design`

Slew Propagation

The `timing_slew_propagation_mode` variable allows you to specify how PrimeTime propagates slew through the circuit. You can set this variable to `worst_slew` (the default), which gives a possibly pessimistic but safe analysis; or `worst_arrival`, which gives a more accurate but, possibly, optimistic analysis. The method of slew propagation affects the delay of a cell arc or net.

In the `worst_slew` (default) mode, at a pin where multiple timing arcs meet (or merge), PrimeTime computes the slew per driving arc at the pin, then selects the worst slew value at the pin to propagate along. Note that the slew selected might not be from the input that contributes to the worst path, so the calculated delay from the merge pin could be pessimistic. Figure 8-4 shows an example of slew propagation.

Figure 8-4 Slew Propagation



In the `worst_arrival` mode, PrimeTime selects and propagates the slew of the input with the worst arrival time, selecting from multiple inputs propagated from the same clock domain. PrimeTime selects and propagates different slews from different clock domains.

Each slew propagation method, worst-arrival or worst-slew, has its own advantages. The worst-arrival method generally provides more accurate analysis for the worst path for each timing endpoint. However, it can potentially produce optimistic results that miss timing violations. The worst-slew method performs a pessimistic analysis that can be relied upon for timing signoff.

You can use both methods to benefit from their respective advantages. If the two results are nearly the same, it suggests that both reports are reasonably accurate. However, if the worst-arrival method shows significantly improved slack, more analysis is necessary to make sure that there is no optimism in the results.

Minimum slew propagation is similar to maximum slew propagation. In other words, PrimeTime selects minimum slew based on the input with the best delay at the merge point. With the worst-arrival method, if the operating condition was originally set to single (`set_operating_conditions -analysis_type single`), PrimeTime automatically switches the operating condition to `on_chip_variation` to ensure that both minimum and maximum slews are propagated, and also issues a message to indicate the new analysis mode.

Associated with the worst-arrival mode are two options to the `set_input_transition` and `set_driving_cell` commands, `-clock` *`clock_name`* and `-clock_fall`. To set the input transition time and delay relative to a clock, use the `-clock` option. This means the transition applies only to external paths driven by the clock. By default, the transition time and delay are set relative to the rising edge of the clock. To use the falling edge instead, use the `-clock_fall` option.

Using the `-clock` or `-clock_fall` option is meaningful only in the worst-arrival slew propagation mode. In the worst-slew (default) mode, PrimeTime takes the worst value over all clocks specified with the command.

Design Rules

PrimeTime checks for violations of electrical rules that are defined in the library or by PrimeTime commands. These rules include

- Maximum limit for transition time
- Maximum and minimum limits for capacitance
- Maximum limit for fanout

To get a report on design rule violations in a design, use the `report_constraint` command.

Maximum Transition Time

The maximum transition time in PrimeTime is treated in a similar fashion as the slew. In this section slew is first discussed in the context of thresholds and derate, then the discussion is extended to maximum transition time.

You can represent a SPICE waveform as a floating-point number in Liberty tables. Figure 8-5 shows an example where:

- SPICE waveform is measured 10-90
- SPICE waveform is showing in blue line

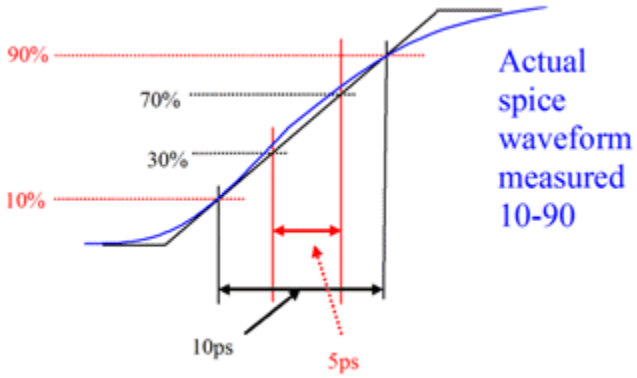
- SPICE measured transition time with slew threshold 10-90 is 10 ps
- Linearized waveform measuring transition time with slew threshold 0-100 is 12.5 ps = 10 * (100-0) / (90-10)
- Representation as single float with slew threshold 30-70 is 5 ps = 12.5 * (70-30) / (100-0)

The SPICE waveform can thus be represented as a floating-point number in Liberty tables or in PrimeTime report as follows:

- A: 10-ps slew threshold as 10-90 with slew derate 1.0
- B: 12.5-ps slew threshold as 10-90 with slew derate 0.8
- C: 5-ps slew threshold as 10-90 with derate 2.0

Note that slew threshold in library is always the threshold used for SPICE measurement. Case A is the native representation, measured in Liberty. Cases B and C are rescaled to the slew threshold 0-100 and 30-70, respectively.

Figure 8-5 SPICE Waveform



Handling Slew in Multiple Threshold and Derate Environment

Liberty allows an arbitrary slew threshold to minimize the error due to slew linearization for various process technologies. Therefore whenever slew information from library interacts with another library (or even a pin of the same library with different slew threshold), a conversion to a common base is required.

Converting Single-Float Slews Between Thresholds and Derates

Assume that you have a library L1 with thresholds TL1-TH1, derate SD1, and L2 with TL2-TH2, derate SD2. Note that L1, L2 are just entities that have their own local thresholds, such as library, design, and library pin.

Assume S1 - slew in local thresholds and a derate of L1, and S2 - slew in local thresholds and a derate of L2. The same conversion rule applies if maximum transition is considered instead of slews.

You can then obtain slews expressed in the local thresholds and derate of the other object as follows:

Equivalent slew S2_1, which is slew S2 expressed in the local derate /threshold of L1:

S2_1 = S2 * (SD2 / (TH2 - TL2)) * ((TH1 - TL1)/ SD1)

The meaning of S2_1 is a float number that represents a waveform of slew S2 but measured in the context of L1. The S1 and S2_1 can be directly compared; the S1 and S2 cannot.

Note that in the presence of detailed RC, slew is computed appropriately in the context of threshold and derate.

The next section discusses how maximum transition constraint is handled in the context of thresholds and derate.

Maximum Transition Constraint Storage

Maximum transition constraints can come from a user input, library, and library pin. User-specified maximum transition constraints are expressed with the main library derate and slew threshold of PrimeTime.

The `set_max_transition` command sets a maximum limit on the transition time for specified pins, ports, designs, or clocks. Setting limits on design constrains all pins in the design. When specified on clocks, pins in clock domain are constrained. Within a clock domain, you can optionally restrict the constraint further to only clock paths or data paths, and to only rising or falling transitions. During constraint checking on a pin or port, the most restrictive constraint specified on a design, pin, port, clock (if the pin or port is in that clock domain), or library is considered. This is also true where multiple clocks launch the same path. The `set_max_transition` command places the `max_transition` attribute, which is a design rule constraint, on the specified objects. In PrimeTime, the slews and maximum transition constraint attributes are reported in the local threshold and derate of each pin or library. For example, the maximum transition time set in the context of the design threshold and derate is scaled to that of the design pin's threshold and derate. The scaling of transition time for slew threshold is on by default.

Table 8-3 describes converting slews between different slew thresholds and derates.

Table 8-3 Converting Slews

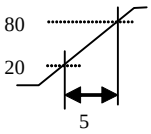
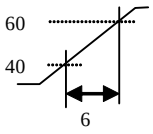
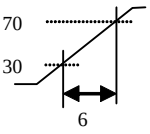
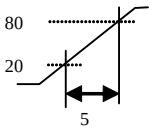
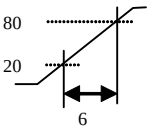
Object	Main library (L1)	Local library	Library pin	Design	Cell instance pin
Threshold and derate source	Yes	Yes	Yes for slew threshold. Inherits derate of the library	Uses main library derate and slew threshold	The more local trip points have higher precedence than the more general ones
Limit (maximum transition)	Yes	Yes	Yes	Yes - in main library threshold and derate	The most restrictive limit applies
Thresholds	20/80 (rise/ fall)	40/60 (rise/ fall)	30/70 (rise/ fall)	Uses main library threshold	30/70 (rise/ fall)
Derate	0.5 (defined)	0.6 (defined)	0.6 (inherited)	0.5 (inherited)	0.6 (inherited)
PrimeTime or Liberty value (slew or limit)	10 ps	10 ps	10 ps	10 ps	10 ps
Linearized SPICE waveform: (rise) Similar for fall					
Expressed in derate and threshold of the main library	10 ps	10 * (0.6 / (0.6 – 0.4)) * ((0.8 – 0.2)/0.5)	10 * (0.6 / (0.7 – 0.3)) * ((0.8 – 0.2)/ / 0.5)	10 * 0.5 / (0.8 – 0.2) * ((0.8 – – 0.2) / 0.5)	10 * 0.6 / (0.7 – 0.3) * ((0.8 – 0.2) / 0.5)

Table 8-3 Converting Slews (Continued)

Object	Main library (L1)	Local library	Library pin	Design	Cell instance pin
PrimeTime or Liberty value (slew or limit)	10 ps	-	-	-	-
Main library limit expressed in local threshold and derate	10 ps	10 * (0.5/(0.8 -.02)) * ((.6 -.4)/.6)	10 * (0.5/(0.8 – 0.2)) * ((0.7 – 0.3) / 0.6)	10*(0.5/((0.8 – 0.2)) * ((0.8 – 0.2)/0.5)	10 * (0.5/(0.8 – 0.2)) * ((0.7 – 0.3)/0.6)

Evaluating Maximum Transition Constraint

To view the maximum transition constraint evaluations, use the `report_constraint -max_transition` command. PrimeTime reports all constraints and slews in the threshold and derate of the pin of the cell instance, and the violations are sorted based on the absolute values (that is, they are not expressed in that of design threshold and derate). You can also use the `report_constraint` command to report constraint calculations only for maximum capacitance and maximum transition for a specified port or pin list. Use the `object_list` option to specify a list of pins or ports in the current design that you want to display constraint related information.

To see the port maximum transition limit, use the `report_port -design_rule` command. To see the default maximum transition setting for the current design, use the `report_design` command. To undo maximum transition limits previously set on ports, pins, designs, or clocks, use `remove_max_transition`.

Example 1 - Setting a Maximum Transition Limit

To set a maximum transition limit of 2.0 units on the ports of OUT*, enter

```
pt_shell> set_max_transition 2.0 [get_ports "OUT*"]
```

To set the default maximum transition limit of 5.0 units on the current design, enter

```
pt_shell> set_max_transition 5.0 [current_design]
```

To set the maximum transition limit of 4.0 units on all pins in the CLK1 clock domain, for rising transitions in datapaths only, enter

```
pt_shell> set_max_transition 4.0 [get_clocks CLK1] -type data_path -rise
```

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

Example 2 - Scaling the Maximum Transition With Different Library Slew Threshold

Consider library lib1 having a slew threshold of 10/90. The design has a maximum transition limit set by you at 0.3 ns. The main library slew threshold for rise and fall is 30/70. By using the `report_constraint -max_transition -all_violators -sig 4` command, you get:

```
                Required       Actual
Pin             Transition    Transition        Slack
 ------------------------------------------------------------
FF1/D           0.6000         0.4000           0.2000
```

Example 3 - Scaling the Maximum Transition With the Slew Derate Factor Specified in the Library

Consider Library lib1 having a slew derate factor of 0.5 and a slew threshold for rise and fall of 30/70. You set a maximum transition on the design at 0.3 ns. The main library slew threshold for rise and fall is 30/70 and slew derate is 1.0. By using the `report_constraint -max_transition -all_violators -sig 4` command, you see:

```
                Required       Actual
Pin             Transition    Transition        Slack
------------------------------------------------------------
FF1/D           0.6000         0.4000           0.2000
```

Minimum and Maximum Net Capacitance

The `set_min_capacitance` command sets a minimum capacitance limit for the specified ports or for the whole design. Similarly, the `set_max_capacitance` command sets a maximum limit on total capacitance for ports or the design. Setting a capacitance limit on a port applies to the net connected to that port. Setting a capacitance limit on a design sets the default capacitance limit for all nets in the design. You have the option to additionally specify max capacitance limit on pins or clocks. When specified on clocks, the pins in the clock domain are constrained. Within a clock domain, you can optionally restrict the constraint further to only clock paths or data paths and only to rising or falling capacitance. The `set_min_capacitance` and `set_max_capacitance` commands place the `min_capacitance` or `max_capacitance` attribute (a design rule constraint) on the specified objects. Capacitance constraint checks are only applicable for output pins. During constraint check, the most restrictive constraint is considered.

To see the capacitance constraint evaluations, use the `report_constraint -min_capacitance` or `-max_capacitance` command. To see port capacitance limits, use the `report_port -design_rule` command. To see the default capacitance settings for the current design, use the `report_design` command. To undo capacitance limits that you set from the UI, use the `remove_min_capacitance` and `remove_max_capacitance` commands.

To set a minimum capacitance limit of 0.2 units on the ports of OUT*, enter

```
pt_shell> set_min_capacitance 0.2 [get_ports "OUT*"]
```

To set the default minimum capacitance limit of 0.1 units on the current design, enter

```
pt_shell> set_min_capacitance 0.1 [current_design]
```

For more information, see the *Synopsys Timing Constraints and Optimization User Guide*.

Maximum Fanout Load

The `set_max_fanout` command sets a maximum fanout load for specified output ports or designs. This command sets the `max_fanout` attribute (a design rule constraint) on the specified objects.

Setting a maximum fanout load on a port applies to the net connected to that port. Setting a maximum fanout load on a design sets the default maximum for all nets in the design. In case of conflict between these two values, the more restrictive value applies.

Library cell pins can have a `max_fanout` value specified. PrimeTime uses the more restrictive of the limit you set or the limit specified in the library.

To see maximum fanout constraint evaluations, use the `report_constraint -max_fanout` command. To see port maximum fanout limits, use the `report_port -design_rule` command. To see the default maximum fanout setting for the current design, use the `report_design` command.

To undo maximum fanout limits you set on ports or designs, use the `remove_max_fanout` command.

To set a maximum fanout limit of 2.0 units on the ports IN*, enter the following syntax:

```
pt_shell> set_max_fanout 2.0 [get_ports "IN*"]
```

To set the default maximum fanout limit of 5.0 units on the current design, enter the following syntax:

```
pt_shell> set_max_fanout 5.0 [current_design]
```

Fanout Load Values for Output Ports

The fanout load for a net is the sum of `fanout_load` attributes for the input pins and output ports connected to the net. Output pins can have maximum fanout limits, specified in the library or set through the `set_max_fanout` command. By default, ports are considered to have a fanout load of 0.0. The `set_fanout_load` command specifies the expected fanout load for output ports in the current design.

To set the fanout load on ports matching OUT* to 3.0, enter the following syntax:

```
pt_shell> set_fanout_load 3.0 "OUT*"
```

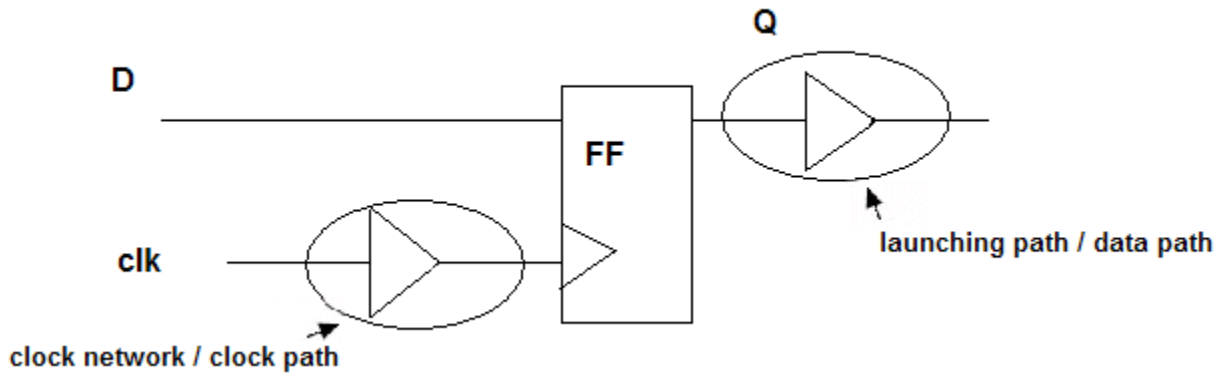
For more information, see the *Synopsys Timing Constraints and Optimization User Guide*.

Constraining Rise and Fall Maximum Capacitance

To constrain rise and fall maximum capacitance at the output of the driver pins in the clock path and data path for clocks in the design, use the `set_max_capacitance` command.

PrimeTime applies maximum capacitance constraints on a clock to all pins in the clock network (clock path) and the pins in the launching path of the clock (data path of the clock). Optionally, you can restrict constraints on a clock to a clock path, data path, rise, and fall capacitance as shown in Figure 8-6.

Figure 8-6 Restrict Constraints



If you specify the `clock_list` option without specifying the `-clock_path`, `-data_path`, `-rise`, or `-fall` options, both clock path and data path and both rise and fall are considered by default. You can use the `-clock_path`, `-data_path`, `-rise`, or `-fall` options only if the list of objects is a clock list, not a port list or design. If a pin (port) has multiple constraints from the design, library, clock (and port), the most restrictive constraint is considered. PrimeTime computes the slack at a pin as the difference between the most restrictive constraint and the effective capacitance at a pin (port).

This feature is applicable to generated clocks. To report the constraint violations, use the `report_constraint` command. For more information about the `set_max_capacitance` command, see the man page.

9

Timing Exceptions

By default, PrimeTime assumes that data launched at a path startpoint should be captured at the path endpoint by the very next occurrence of a clock edge at the endpoint. For paths that are not intended to operate in this manner, you need to specify a timing exception. Otherwise, the timing analysis does not match the behavior of the real circuit.

This chapter describes timing exception in the following sections:

Timing Exception Overview

PrimeTime supports the following ways to specify timing exceptions:

- Setting 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.
- Setting 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 time values.
- Setting 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.

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. For more information, see "Path Specification Methods" on page 6-11.

When you specify a path by its startpoint and endpoint, be sure to specify a timing path that is valid in PrimeTime. A path startpoint must be either a register clock pin or an input port. A path endpoint must be either a register data input pin or an output port.

To view a list of timing exceptions that have been applied to a design, use the `report_exceptions` command. You can also preserve source location information, namely the source file and line number when tracking and reporting information for constraints. For more information, see "Reporting Exceptions Source File and Line Number Information" on page 6-9.

To restore the default timing constraints on a path, use the `reset_path` command.

Single-Cycle (Default) Path Delay Constraints

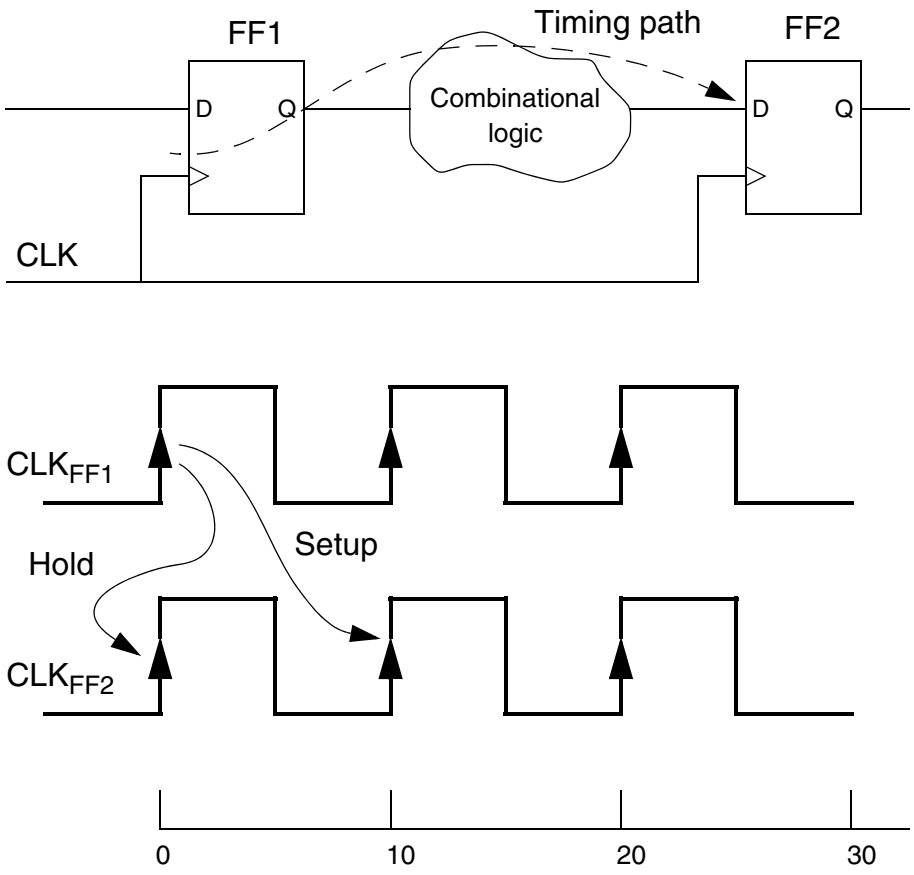
To determine whether you need to set a timing exception, you must understand how PrimeTime calculates maximum and minimum delay times for paths. You must also understand how this calculation is similar or dissimilar to the operation of the design that is being analyzed. For an introduction to static timing analysis principles, see "Static Timing Analysis Overview" on page 1-9.

Path Delay for Flip-Flops Using a Single Clock

Figure 9-1 shows how PrimeTime determines the setup and hold constraints for a path that begins and ends on rising-edge-triggered flip-flops. In this example, the two flip-flops are triggered by the same clock. The clock period is 10 ns.

PrimeTime performs a setup check to verify that the data launched from FF1 at time=0 arrives at the D input of FF2 in time for the capture edge at time=10. If the data takes too long to arrive, it is reported as a setup violation.

Figure 9-1 Single-Cycle Setup and Hold for Flip-Flops



Similarly, PrimeTime performs a hold check to verify that the data launched from FF1 at time 0 does not get propagated so soon that it gets captured at FF2 at the clock edge at time 0. If the data arrives too soon, it is reported as a hold violation.

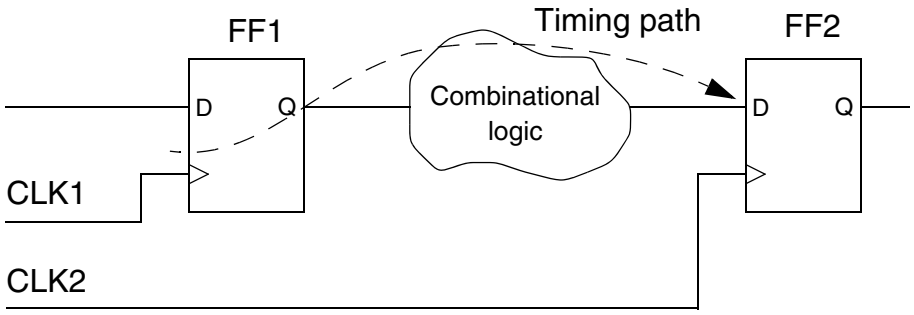
The setup and hold requirements determined by PrimeTime for sequential elements take into account all relevant parameters such as the delay of the combinational logic elements in the path, the setup and hold requirements of the flip-flop elements as defined in the technology library, and the net delays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

Path Delay for Flip-Flops Using Different Clocks

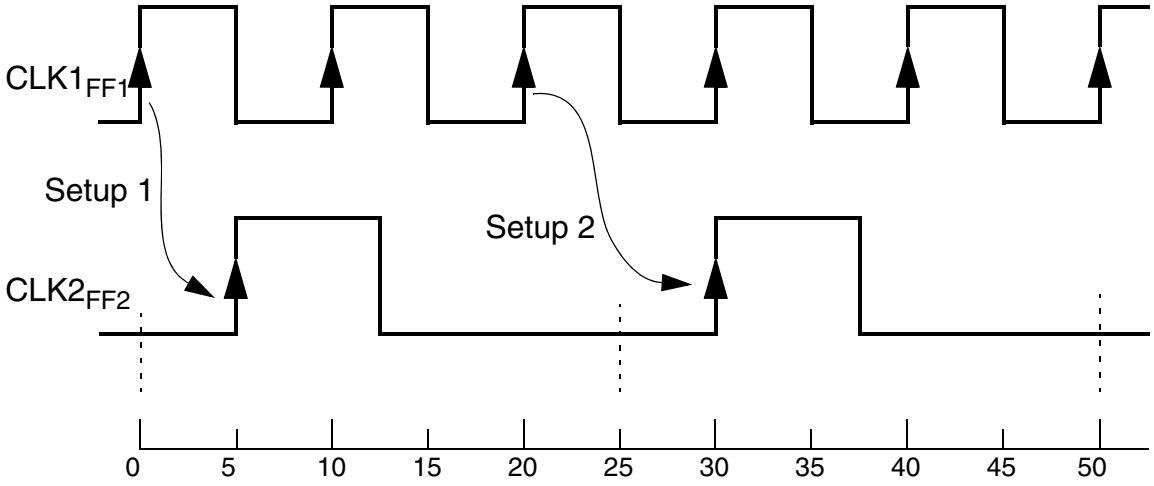
The algorithm for calculating path delays is more complicated when the launch and capture flip-flops belong to different clock domains.

Consider the circuit shown in Figure 9-2. The flip-flops at the beginning and end of the timing path are clocked by different clocks, CLK1 and CLK2. The two clocks are declared by the `create_clock` commands shown in the figure.

Figure 9-2 Setup Constraints for Flip-Flops With Different Clocks




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



By default, PrimeTime assumes that the two clocks are synchronous to each other, with a fixed phase relationship. If this is not the case for the real circuit (for example, because the two clocks are never active at the same time or because they operate asynchronously), then you need to declare this fact by using any of several methods. Otherwise, PrimeTime spends time checking constraints and reporting violations that do not exist in the actual circuit.

Setup Analysis

PrimeTime looks at the relationship between the active clock edges over a full repeating cycle, equal to the least common multiple of the two clock periods. For each capture (latch) edge at the destination flip-flop, PrimeTime assumes that the corresponding launch edge is the nearest source clock edge occurring before the capture edge.

In Figure 9-2, there are two capture edges in the period under consideration. For the capture edge at time=5, the nearest preceding launch edge is at time=0. The corresponding setup relationship is labeled Setup 1.

For the capture edge at time=30, the nearest preceding launch edge is at time=20. This setup relationship is labeled Setup 2. The source clock edge at time=30 occurs at the same time as the capture edge, not earlier, so it is not considered the corresponding launch edge.

Setup 1 allows less time between launch and capture, so it is the more restrictive constraint. It determines the maximum allowed delay for the path, which is 5 ns for this example.

Hold Analysis

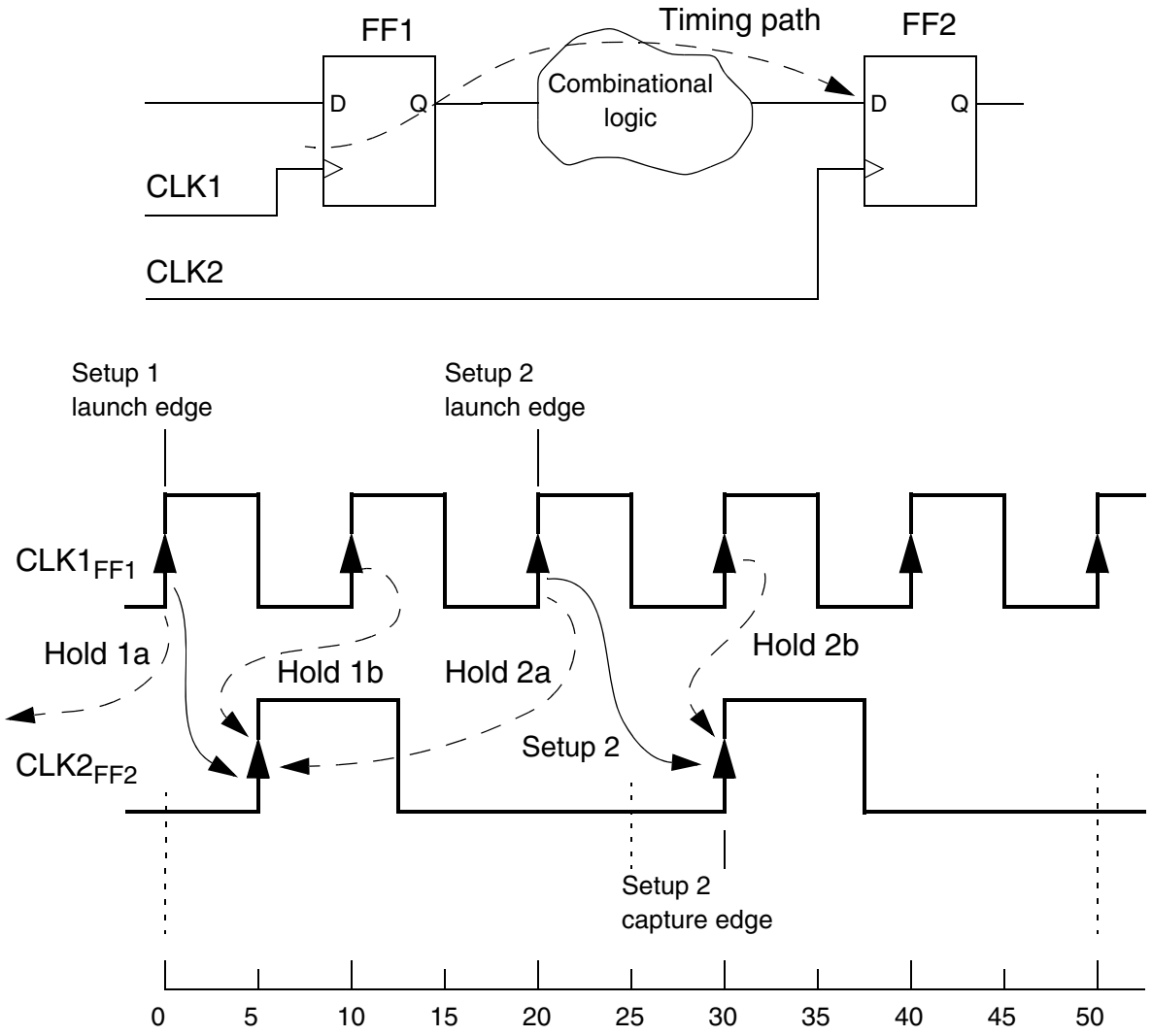
The hold relationships checked by PrimeTime are based on the clock edges adjacent to those used to determine the setup relationships. To determine the most restrictive hold relationship, PrimeTime considers all valid setup relationships, including both Setup 1 and Setup 2 in Figure 9-2.

For each setup relationship, PrimeTime performs two different hold checks:

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

Figure 9-3 shows the hold checks performed by PrimeTime for the current example. First consider the setup relationship labeled Setup 2. PrimeTime confirms that the data launched by the setup launch edge is not captured by the previous capture edge; this relationship is labeled Hold 2a. It also confirms that the data launched by the next clock edge at the source is not captured by the setup capture edge; this relationship is labeled Hold 2b.

Figure 9-3 Hold Constraints for Flip-Flops With Different Clocks



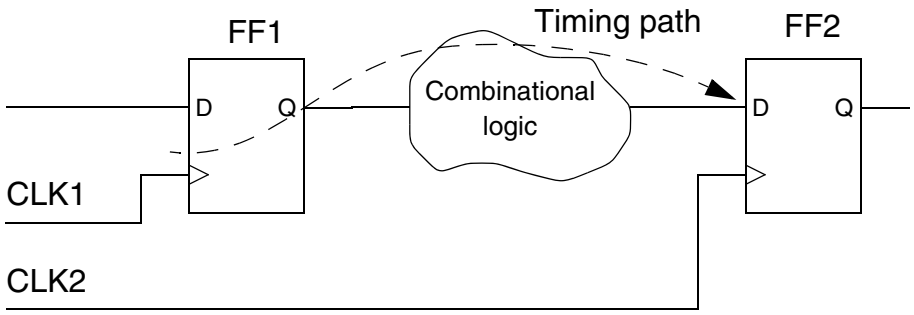
PrimeTime similarly checks the hold relationships relative to the clock edges of Setup 1, as indicated in the figure. The most restrictive hold check is the one where the capture edge occurs latest relative to the launch edge, which is Hold 2b in this example. Therefore, Hold 2b determines the minimum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

Single-Cycle Path Analysis Examples

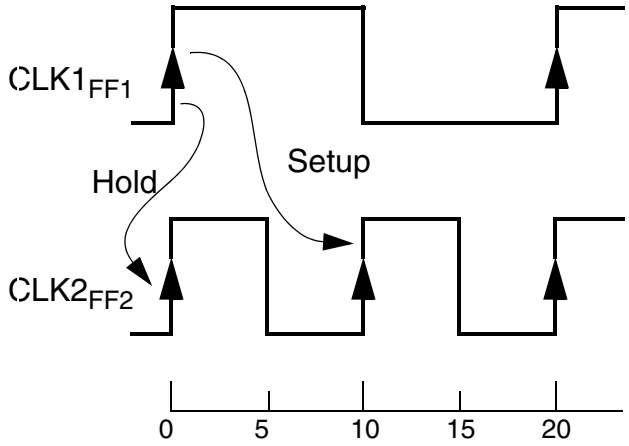
The following examples further illustrate how PrimeTime calculates the delay requirements for edge-triggered flip-flops in the absence of timing exceptions.

In Figure 9-4, 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 9-4 Delay Requirements With 20ns/10ns Clocks

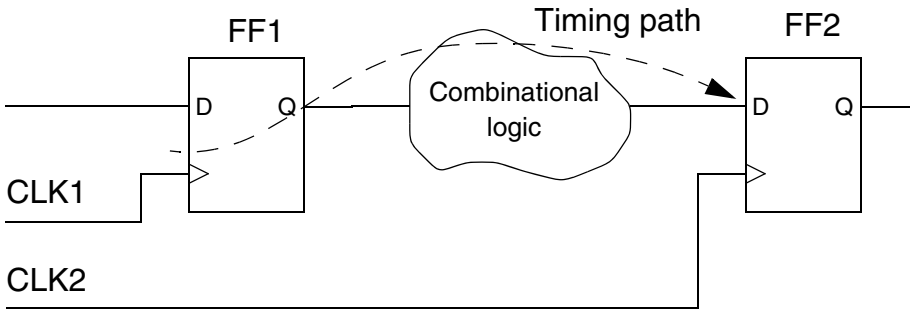



```
pt_shell> create_clock -period 20 -waveform {0 10} CLK1
pt_shell> create_clock -period 10 -waveform {0 5} CLK2
```

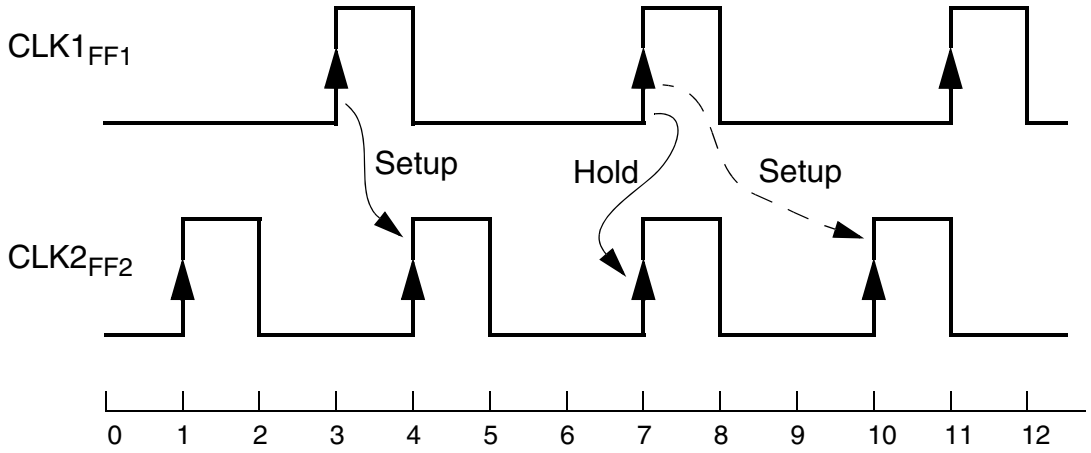


In Figure 9-5, CLK1 has a period of 4 ns and CLK2 has a period of 3 ns. The most restrictive setup relationship is the launch edge at time=3 to the capture edge at time=4. The most restrictive hold relationship is the launch edge at time=7 to the capture edge at time=7, based on the setup relationship shown by the dashed-line arrow in the timing diagram.

Figure 9-5 Delay Requirements With 4ns/3ns Clocks




```
pt_shell> create_clock -period 4 -waveform {3 4} CLK1
pt_shell> create_clock -period 3 -waveform {1 2} CLK2
```



Setting False Paths

A false path is a logic path 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:

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

Declaring a path to be false removes all timing constraints from the path. PrimeTime still calculates the path delay, but does not report it to be an error, no matter how long or short the delay.

Setting a false path is a point-to-point timing exception. This is different from using the `set_disable_timing` command, which disables timing analysis for a specified pin, cell, or port. Using the `set_disable_timing` command removes the affected objects from timing analysis, rather than removing the timing constraints from the paths. If all paths through a pin are false, using `set_disable_timing [get_pins pin_name]` is more efficient than using `set_false_path -through [get_pins pin_name]`.

Another example of a false path is a path between flip-flops belonging to two clock domains that are asynchronous with respect to each other.

To declare all paths between two clock domains to be false, you can use a set of two commands such as the following:

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

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

For efficiency, be sure to specify each clock by its clock name, not by the pin name (use `get_clocks`, not `get_pins`).

An alternative is to use the `set_clock_groups` command to exclude paths from consideration that are launched by one clock and captured by another. Although this has the same effect as declaring a false path between the two clocks, it is not considered a timing exception and is not reported by the `report_exceptions` command.

PrimeTime 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. For more information, see *PrimeTime Advanced Timing Analysis User Guide*.

Setting Maximum and Minimum Path Delays

By default, PrimeTime calculates the maximum and minimum path delays by considering the clock edge times. To override the default maximum or minimum time with your own specific time value, use the `set_max_delay` or `set_min_delay` command. For example, to set the maximum path delay between registers REGA and REGB to 12, use the following command:

```
pt_shell> set_max_delay 12  \
           -from [get_cells REGA] -to [get_cells REGB]
```

With this timing exception, PrimeTime ignores the clock relationships. A path delay between these registers that exceeds 12 time units minus the setup requirement of the endpoint register is reported as a timing violation. Similarly, to set the minimum path delay between registers REGA and REGB to 2, use the following command:

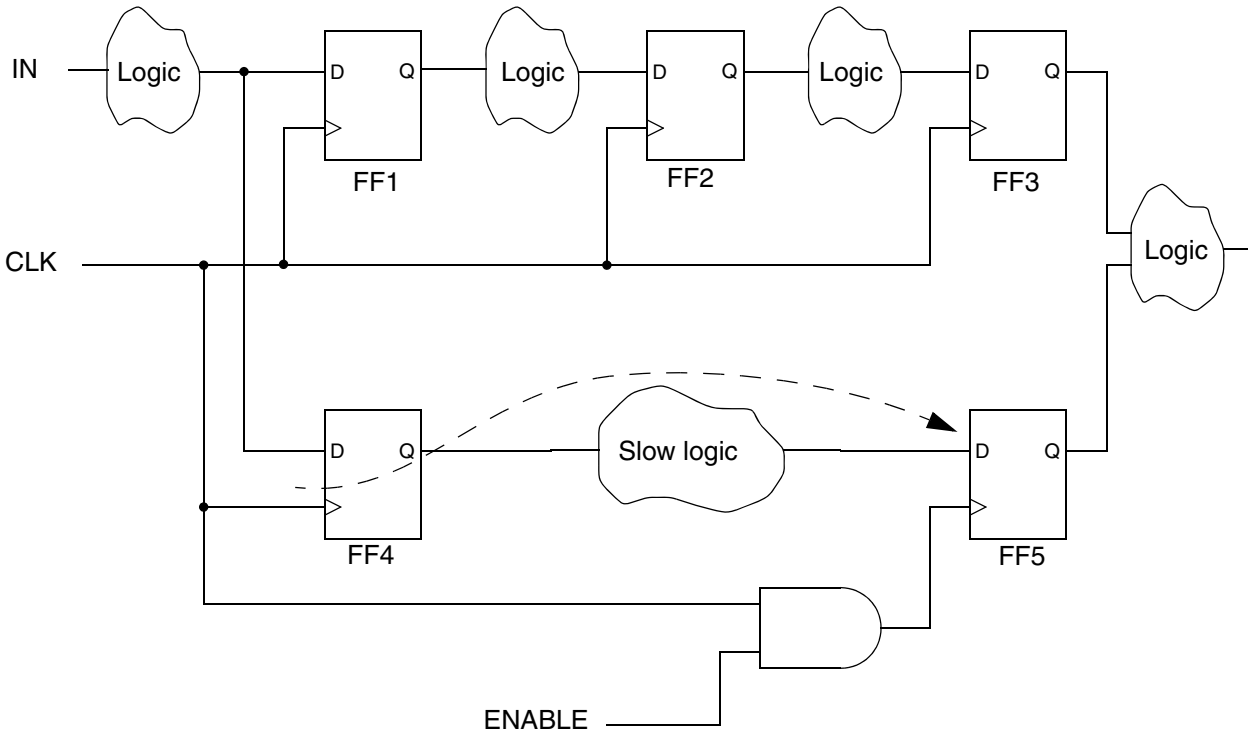
```
pt_shell> set_min_delay 2.0  \
          -from [get_cells REGA] -to [get_cells REGB]
```

Again, PrimeTime ignores the clock relationships. A path delay between these registers that is less than 2 time units plus the hold requirement of the endpoint register is reported as a timing violation. You can optionally specify that the delay value apply only to rising edges or only to falling edges at the endpoint. For more information, see the `set_max_delay` or `set_min_delay` command man page.

Setting Multicycle Paths

The `set_multicycle_path` command specifies the number of clock cycles required to propagate data from the start of a path to the end of the path. PrimeTime calculates the setup or hold constraint according to the specified number of cycles. For example, consider the circuit shown in Figure 9-6. The path from FF4 to FF5 is designed to take two clock cycles rather than one. However, by default, PrimeTime assumes single-cycle timing for all paths. Therefore, you need to specify a timing exception for this path.

Figure 9-6 Multicycle Path Example



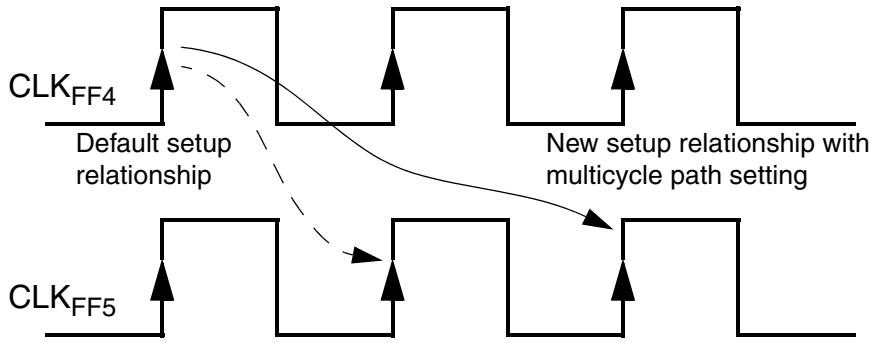
One timing exception method is to specify an explicit maximum delay value with the `set_max_delay` command. However, you might want to use the `set_multicycle_path` command instead because the maximum delay value is automatically adjusted when you change the clock period.

To set the multicycle path for the design shown in Figure 9-6, you can use the following command:

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

The first command tells PrimeTime that the path takes two clock cycles rather than one, establishing the new setup relationship shown in Figure 9-7. The second capture edge (rather than the first) following the launch edge becomes the applicable edge for the end of the path.

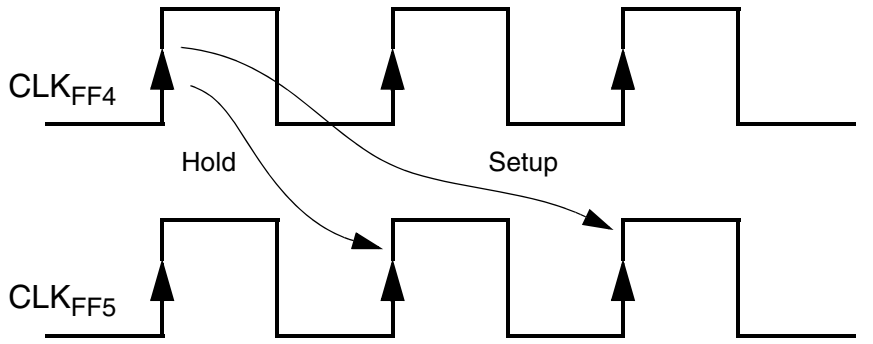
Figure 9-7 Multicycle Path Setup




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

Changing the setup relationship implicitly changes the hold relationship as well because all hold relationships are based on the valid setup relationships. PrimeTime verifies that the data launched by the setup launch edge is not captured by the previous capture edge. The new hold relationship is shown in Figure 9-8.

Figure 9-8 Multicycle Path Hold Based on New Setup




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

The hold relationship shown in Figure 9-8 is probably not the correct relationship for the design. If FF4 does not need to hold the data beyond the first clock edge, you need to specify another timing exception.

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

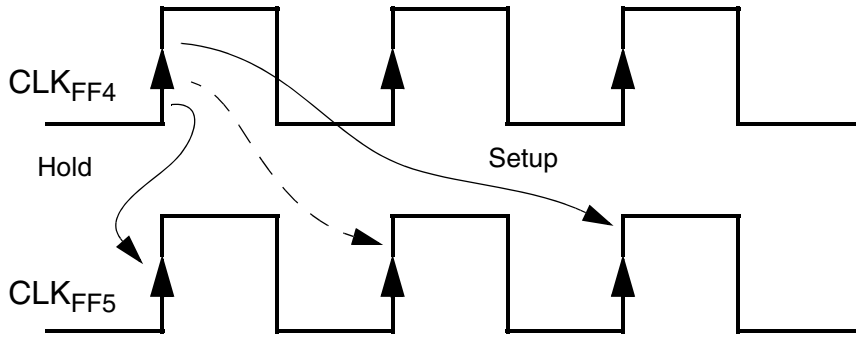
```
pt_shell> set_multicycle_path -setup 2 \
          -from [get_cells FF4] -to [get_cells FF5]

pt_shell> set_min_delay 0 -from [get_cells FF4] -to [get_cells FF5]
```

or preferably:

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

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

Figure 9-9 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 9-9 Multicycle Path Hold Set Correctly



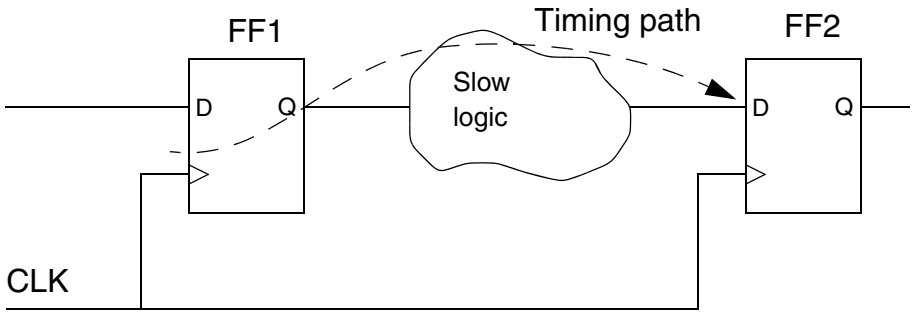

```
pt_shell> set_multicycle_path -setup 2 \
          -from [get_cells FF4] -to [get_cells FF5]


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

Note that PrimeTime interprets the `-setup` and `-hold` values in the `set_multicycle_path` command differently. The integer value for the `-setup` argument specifies the number of clock cycles for the multicycle path. In the absence of a timing

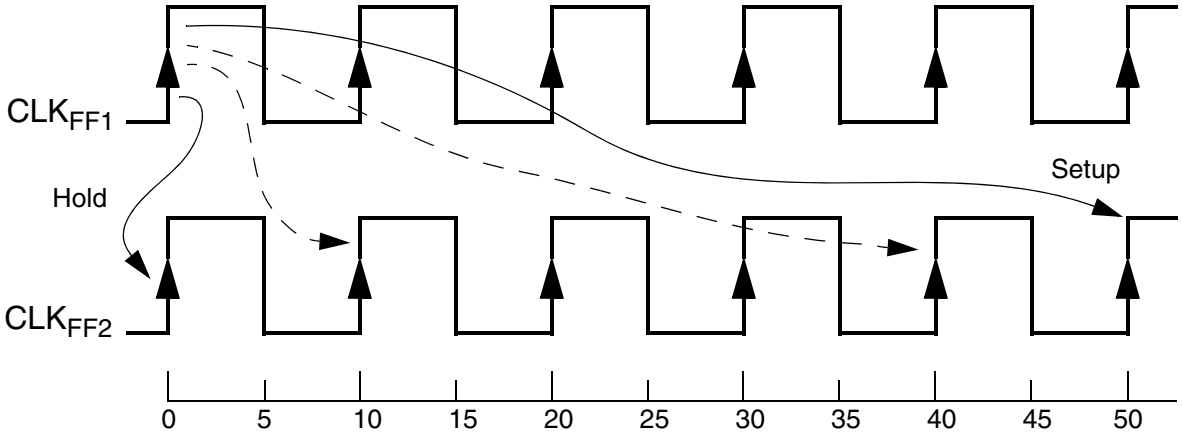
exception, the default is 1. The integer value for the `-hold` argument specifies the number of clock cycles to move the capture edge backward with respect to the default position (relative to the valid setup relationship); the default setting is 0.

The example shown in Figure 9-10 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 9-10 Multicycle Path Taking Five Clock Cycles




```
pt_shell> create_clock -period 10 -waveform {0 5} CLK1
```




```
pt_shell> set_multicycle_path -setup 5 \
          -from [get_cells FF1] -to [get_cells FF2]

pt_shell> set_multicycle_path -hold 4 \
          -from [get_cells FF1] -to [get_cells FF2]
```

With `set_multicycle_path -setup 5`, the setup relationship spans five clock cycles rather than one, from time=0 to time=50, as shown by the long solid-line arrow. This implicitly changes the hold relationship to the prior capture edge at time=40, as shown by the long dashed-line arrow.

To move the capture edge for the hold relationship back to time=0, you need to use `set_multicycle_path -hold 4` to move the capture edge back by four clock cycles.

To summarize, PrimeTime determines the number of hold cycles as follows:

(hold cycles) = (setup option value) – 1 – (hold option value)

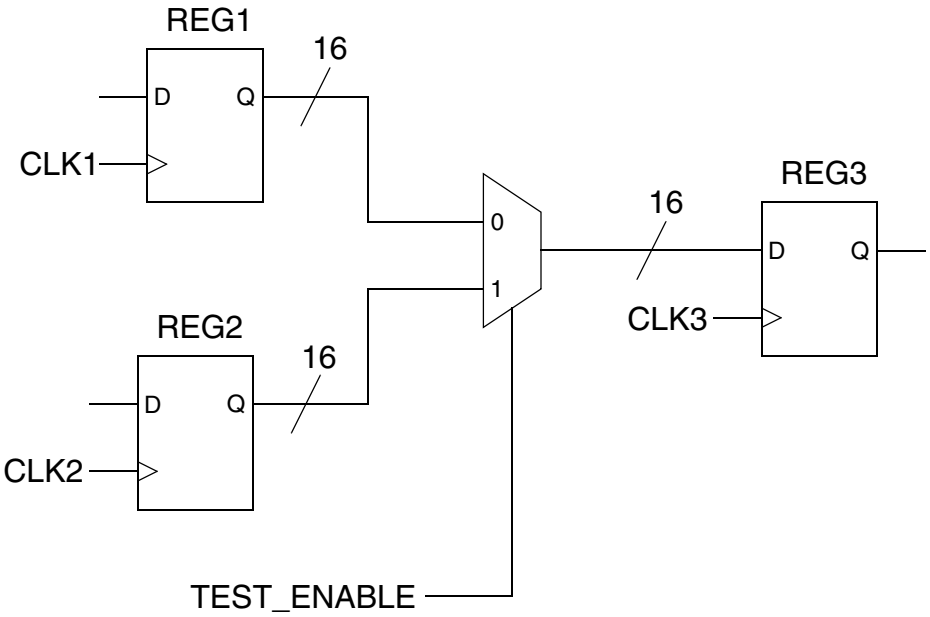
By default, hold cycles = 1 – 1 – 0 = 0. For Figure 9-9, hold cycles = 2 – 1 – 1 = 0. For Figure 9-10, hold cycles = 5 – 1 – 4 = 0.

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

Specifying Exceptions Efficiently

In many cases, you can specify the exception paths many different ways. Choosing an efficient method can reduce the analysis runtime. For example, consider the circuit shown in Figure 9-11. 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 9-11 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" in Chapter 10)
- Set an exclusive relationship between the CLK1 and CLK2 clock domains. See "Exclusive Clocks" on page 7-17.
- Declare the paths between clock domains CLK2 and CLK3 to be false.

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

  This method is an efficient way to specify the false paths because PrimeTime only needs to keep track of the specified clock domains. It does not have to keep track of exceptions on registers, pins, nets, and so on.

- Declare the 16 individual paths from REG2 to REG3 to be false.

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

pt_shell> set_false_path -from [get_pins REG2[1]/CP] \
          -to [get_pins REG3[1]/D]

pt_shell> ...
```

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

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

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

This method is even less efficient because PrimeTime must keep track of paths from each clock pin of REG2 to each data pin of REG3, a total of 256 paths.

- Declare all paths from REG2 to be false.

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

This method is similar to the previous one. PrimeTime must keep track of all paths originating from each clock pin of REG2, a total of 256 paths.

In summary, look at the root cause that is making the exceptions necessary and find the simplest way to control the timing analysis for the affected paths. Before using false paths, consider using case analysis (`set_case_analysis`), declaring an exclusive relationship between clocks (`set_clock_groups`), or disabling analysis of part of the design (`set_disable_timing`). These alternatives can be more efficient than using the `set_false_path` command.

If you must set false paths, avoid specifying a large number of paths using the `-through` argument, by using wildcards, or by listing the paths one at a time. After you set false paths and other timing exceptions, you might be able to simplify the set of exception-setting commands by using the `transform_exceptions` command. For more information, see "Transforming Exceptions" on page 9-23.

Exception Order of Precedence

If different timing exception commands are in conflict for a particular path, the exception PrimeTime uses for that path depends on the exception types or the path specification methods used in the conflicting commands. A set of rules establishes the order of priority for different exception-setting situations.

Note that PrimeTime applies the exception precedence rules independently on each path (not each command). For example, suppose that you use the following commands:

```
pt_shell> set_max_delay -from A 5.1
pt_shell> set_false_path -to B
```

The `set_false_path` command has priority over the `set_max_delay` command, so any paths that begin at A and end at B are false paths. However, the `set_max_delay` command still applies to paths that begin at A but do not end at B.

Exception Type Priority

The following pairs of timing exception types are not considered to be in conflict, so both settings can be valid for a path:

- Two `set_false_path` settings
- `set_min_delay` and `set_max_delay` settings
- `set_multicycle_path -setup` and `-hold` settings

In case of conflicting exceptions for a particular path, the timing exception types have the following order of priority, from highest to lowest:

1. `set_false_path`
2. `set_max_delay` and `set_min_delay`
3. `set_multicycle_path`

For example, if you declare a path to be false and also set its maximum delay to some value, the false path declaration has priority. The maximum delay setting is ignored. You can list ignored timing exceptions by using `report_exceptions -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,

```
pt_shell> set_max_delay 12 -from [get_clocks CLK1]

pt_shell> set_max_delay 15 -from [get_clocks CLK1] \
          -to [get_clocks CLK2]
```

The first command sets the maximum delay of all paths starting from CLK1. However, the second command is more specific, so it overrides the first command for paths starting at CLK1 and ending at CLK2. The remaining paths starting from CLK1 are still controlled by the first command.

The various `-from/-to` path specification methods have the following order of priority, from highest to lowest:

1. `-from` *pin*, `-rise_from` *pin*, `-fall_from` *pin*
2. `-to` *pin*, `-rise_to` *pin*, `-fall_to` *pin*

3. `-through`, `-rise_through`, `-fall_through`
4. `-from` *`clock`*, `-rise_from` *`clock`*, `-fall_from` *`clock`*
5. `-to` *`clock`*, `-rise_to` *`clock`*, `-fall_to` *`clock`*

Use the preceding list to determine which of two conflicting timing exception commands has priority (for example, two `set_max_delay` commands). Starting from the top of the list:

1. A command containing `-from` *`pin`*, `-rise_from` *`pin`*, or `-fall_from` *`pin`* has priority over a command that does not contain `-from` *`pin`*, `-rise_from` *`pin`*, or `-fall_from` *`pin`*.
2. A command containing `-to` *`pin`*, `-rise_to` *`pin`*, or `-fall_to` *`pin`* has priority over a command that does not contain `-to` *`pin`*, `-rise_to` *`pin`*, or `-fall_to` *`pin`*.

... and so on down the list until the priority is resolved.

Here are some possible path specification combinations, listed in order of priority from highest to lowest, according to the preceding priority rules:

1. `-from` *`pin`* `-to` *`pin`*
2. `-from` *`pin`* `-to` *`clock`*
3. `-from` *`pin`*
4. `-from` *`clock`* `-to` *`pin`*
5. `-to` *`pin`*
6. `-from` *`clock`* `-to` *`clock`*
7. `-from` *`clock`*
8. `-to` *`clock`*

Reporting Exceptions

To get a report on timing exceptions that have been set, use the `report_exceptions` command. You can reduce the scope of the report by using the path specification arguments `-from`, `-to`, `-through`, `-rise_from`, `-fall_to`, and so on, to match the path specifiers used when the original exceptions were created.

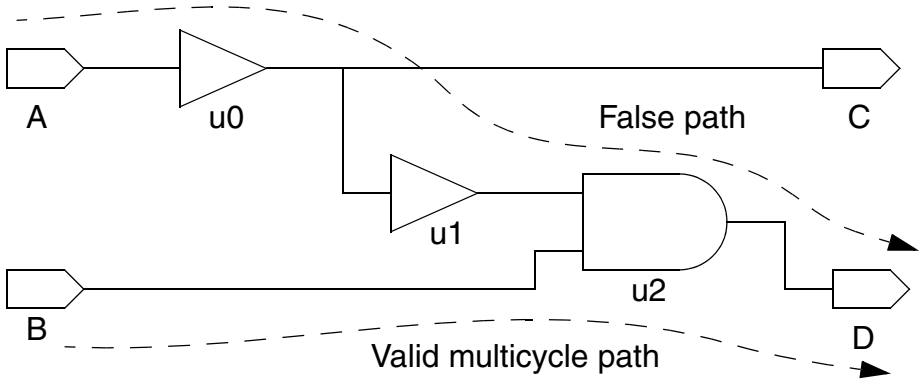
The `report_exceptions` command causes a complete timing update, so be sure to use it only after you have set up all timing assertions and you are ready to receive the report.

Exception-setting commands are sometimes partially or fully ignored for a number of reasons. For example, a command that specifies a broad range of paths can be partially overridden by another exception-setting command that specifies a subset of those paths. For a partially ignored command, the exception report shows the reason that some part of the command is being ignored.

By default, a command that is fully ignored is not reported by the `report_exceptions` command. To get a report on just the commands that are fully ignored, use the `-ignored` option with the `report_exceptions` command.

Consider the design shown in Figure 9-12 and the following exception-setting and exception-reporting commands.

Figure 9-12 Design to Demonstrate Ignored Exceptions




```
pt_shell> set_false_path -from A -to D
pt_shell> set_multicycle_path 2 -from A -to D
pt_shell> set_multicycle_path 2 -from {A B C} -to D
pt_shell> report_exceptions
...
Reasons:
f - invalid startpoint(s)
t - invalid endpoint(s)
p - non-existent paths
o - overridden paths

From          To      Setup      Hold       Ignored
-------------------------------------------------
A             D       FALSE      FALSE
{ A B C }     D       2          *          f,o
```

```
pt_shell> report_exceptions -ignored
...
From          To       Setup      Hold       Ignored
-----------------------------------------------
A             D        2          *          o
```

The first exception-setting command sets a false path from port A to port D. This is the highest-priority command, so it is fully enforced by PrimeTime.

The second exception-setting command attempts to set a multicycle path from port A to port D. It is fully ignored because it is overridden by the higher-priority false path command. Because this command is fully ignored, it is reported only when you use the `-ignored` option of the `report_exceptions` command. In the report, the letter code “o” in the “Ignored” column shows the reason the command is being ignored.

The third exception-setting command attempts to set multicycle paths from ports A to D, B to D, and C to D. It is partially valid (not fully ignored), so it is reported by the `report_exceptions` command without the `-ignored` option. The path from A to D is ignored because it is overridden by the false path command. The path from B to D is valid, so that multicycle path is enforced by PrimeTime. The path from C to D is invalid because port C is not a valid startpoint for a path. In the report, the letter codes in the “Ignored” column indicate the reasons that some of the paths specified in the command are being ignored.

Checking Ignored Exceptions

A timing exception that you entered, but is not accepted by PrimeTime is called an ignored exception. There are several reasons an exception is ignored:

- Specified startpoint or endpoint is not valid. The startpoint must be a register clock pin or input port. The endpoint must be a register data input pin or output port.
- Specified path is not constrained (for example, the startpoint is an input port with no input delay set, or the capture flip-flop at the endpoint is not clocked by a defined clock signal).
- Path is invalid because of `set_disable_timing`, constant propagation, loop breaking, or case analysis.
- Exception has a lower priority than another exception applied to the same path.

The `report_exceptions` command reports exceptions that are fully and partially valid. To get a report on all fully ignored exceptions, use `report_exceptions -ignored`. It is a good idea to examine these reports to confirm that you have specified all exceptions correctly.

If ignored exceptions are reported, you should determine the cause and correct them by changing the path specifications or removing the exception-setting commands. Large numbers of ignored exceptions can increase memory usage and analysis runtime.

If the reason that an exception is partially or fully ignored is not immediately apparent, check the reasons listed in the “Ignored” column of the exception report and consider the possible causes listed above. It might be helpful to get more information using the following commands:

```
transform_exceptions -dry_run
report_exceptions -ignored
```

To find out if there is a logical path between two points, use:

```
all_fanout -from point_a -endpoints_only
all_fanin -to point_b -startpoints_only
```

After a timing update, to examine the path timing, you can use:

```
report_timing -from point_a -to point_b
```

To convert the current set of exception-setting commands to a simpler, equivalent set of commands, use the `transform_exceptions` command. For an example of a report on ignored exceptions, see Figure 9-12 on page 9-20.

Removing Exceptions

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

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

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

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

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

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

```
pt_shell> set_false_path -through [get_pins d/Z] -reset_path
```

This example first resets all timing exceptions previously applied to the paths through the specified point, then applies the false path exception to these paths.

To remove all exceptions from the design, you can use the `reset_design` command, which removes all user-specified clocks, path groups, exceptions, and attributes (except those defined with the `set_user_attribute` command).

Transforming Exceptions

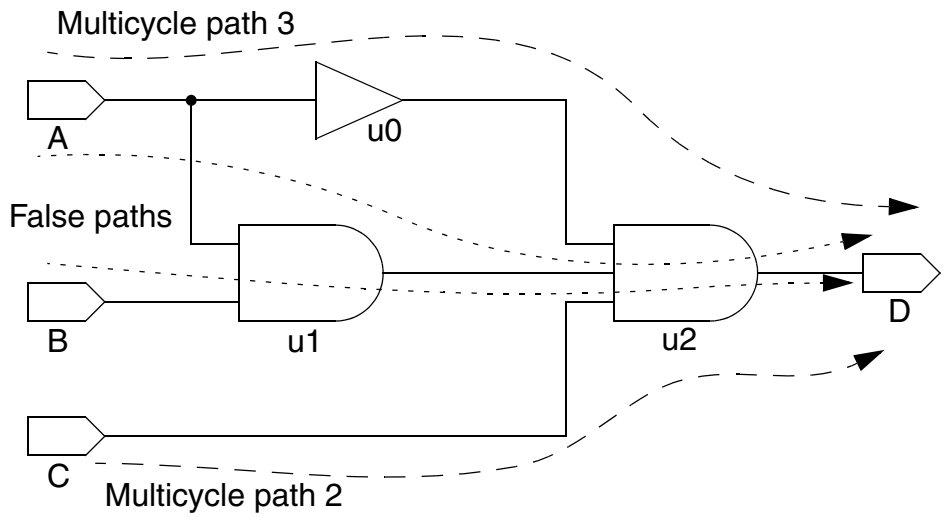
Exceptions that you set are sometimes partially or completely ignored for various reasons. Multiple exception-setting commands with overlapping path specifications can make it difficult to understand the scope of each command.

To gain a better understanding of these exceptions, you can use the `report_exceptions` command. The generated report indicates the reasons that paths were ignored for each exception-setting command, based on the exceptions originally entered by you. However, the report does not always make it clear what exceptions apply to a particular path.

The `transform_exceptions` command has the ability to transform the current set of exceptions into a new, simpler set of equivalent exception-setting commands. The command can be used to get information about the exceptions that apply to a particular path or group of related paths, or to transform a complex set of exception-setting commands into a simpler, equivalent set of commands.

The design and exception-setting commands shown in Figure 9-13 demonstrate some of the principles of exception transformation.

Figure 9-13 Design to Demonstrate Transformed Exceptions




```
pt_shell> set_false_path -through [get_pins u1/Z]

pt_shell> set_multicycle_path 2 \
          -from [get_ports A] -to [get_ports D]


pt_shell> set_multicycle_path 3 -from [get_ports {B C}]
          -through [get_pins u2/Z] -to [get_ports D]
```

The false path through u1/Z sets the two false paths shown in the figure, starting at ports A and B and both ending at port D.

The `set_multicycle_path 2` command specifies two paths that start at port A and end at port D: one through u0 and the other through u1. However, the path through u1 is overridden by the `set_false_path` command, which has a higher priority.

The `set_multicycle_path 3` command specifies two paths: one starting at port B and the other starting at port C, with both ending at port D. However, the path starting at port B is overridden by the `set_false_path` command. There are many ways you could specify this same set of exceptions. For example, you could simplify the last command as follows:

```
pt_shell> set_multicycle_path 3  \
          -from [get_ports C] -to [get_ports D]
```

After you set the exceptions on a design, you can use the `transform_exceptions` command to automatically eliminate invalid, redundant, and overridden exceptions and find a simpler, equivalent set of commands. You can then write out the new set of commands with `write_sdc` or `write_script`. A simpler set of commands is easier to understand and is easier for back-end tools to use. To write out just the exception-setting commands (without the other timing context commands), use `-include {exceptions}` in the `write_sdc` or

write_script command. By default, transform_exceptions transforms all exceptions throughout the design. For example, using the command on the design shown in Figure 9-13 gives the following results:

```
pt_shell> transform_exceptions
...
Transformations Summary   false  multicycle  delay  Total
-------------------------------------------------------------
  non-existent paths      0      1           0      1
-------------------------------------------------------------
Total transformations                                5

Exceptions Summary        false  multicycle  delay  Total
-------------------------------------------------------------
  Modified                0      1           0      1
  Analyzed                1      2           0      3
```

To restrict the scope of the design on which the command operates, you can use the options -from, -through, -to, -rise_from, and so on. In that case, PrimeTime only transforms exception commands that operate on at least one of the specified paths.

To generate a transformation report without actually performing the transformation, use the -dry_run option. The generated report can help you understand why certain paths are being ignored in the original exception-setting commands.

Exception transformation depends on the design context as well as the set of exception-setting commands that have been used. If you change the design, the transformed exceptions might not be equivalent to the original exceptions. To keep a record of the current set of exception-setting commands, use write_sdc or write_script before doing the transformation.

Exception Removal

A major effect of exception transformation is the removal of invalid, redundant, and overridden paths from the original exception-setting commands. To find out the reasons that paths are being eliminated, use the -verbose option in the transform_exceptions command. For example,

```
pt_shell> transform_exceptions -verbose
...
From           Through        To        Setup         Hold
-------------------------------------------------------------
*              u1/Z           *         FALSE         FALSE

A              *              D         cycles=2      cycles=0

{ B C }        u2/C           D         cycles=3      cycles=0
  NON_EXISTENT_PATH
```

```
        -from   B
        -through     u2/C
        -to     D
...
```

Each time PrimeTime eliminates paths from the exception-setting commands, it reports one of the following reasons:

- Non-existent path: The path does not exist in the design or has been disabled (for example, by `set_disable_timing`, `set_false_path`, or case analysis).
- Overridden: The exception applied to the path is overridden by another exception of higher priority.
- Invalid startpoint: The path startpoint is not a primary input or a clock pin of a sequential element.
- Invalid endpoint: The path endpoint is not a primary output or a data input pin of a sequential element.
- Unconstrained path: The path is not constrained (for example, a primary input with no input delay specified).
- Clock network path: The path is part of a clock network, which by default is unconstrained.
- Single-cycle MCP: The exception is a multicycle path with the number of cycles set to the default value (1 for a setup check or 0 for a hold check).
- Mode analysis: The path has been invalidated by mode analysis (for example, a path that involves writing to RAM with the RAM module set to read mode).
- Exclusive domains: The path crosses between clock domains that have been defined to be exclusive by the `set_clock_groups` command.

By default, `transform_exceptions` removes exceptions for all of the reasons listed above. To selectively restrict exception removal to certain reasons, use the `-remove_ignored` option and specify the types of ignored exceptions to remove. For example,

```
pt_shell> transform_exceptions -remove_ignored overridden
```

These are the possible settings for the `-remove_ignored` option:

- `invalid_specifiers`: paths specified with an invalid startpoint or endpoint, single-cycle multicycle paths
- `no_path`: paths that do not exist or are inactive due to case analysis, `set_disable_timing`, or other reasons
- `overridden`: paths overridden by other exceptions or mode analysis; and paths between exclusive clock domains

- clock_path: clock network paths
- unconstrained_path: paths that are not constrained

Exception Flattening

The transform_exceptions command performs flattening when you use the -flatten option. Flattening separates multiple -from, -through, or -to objects in a single command into multiple commands. For example,

```
pt_shell> set_false_path -through {u0/Z u1/Z}
pt_shell> transform_exceptions -flatten
...
pt_shell> write_script -include {exceptions}
```

The script written by the write_script command contains the result of flattening the original commands:

```
set_false_path -through [get_pins {u0/Z}]
set_false_path -through [get_pins {u1/Z}]
```

The single set_false_path command is flattened into two commands by separating the two -through objects.

10

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.

Case analysis concepts are described in the following sections:

Performing Case Analysis

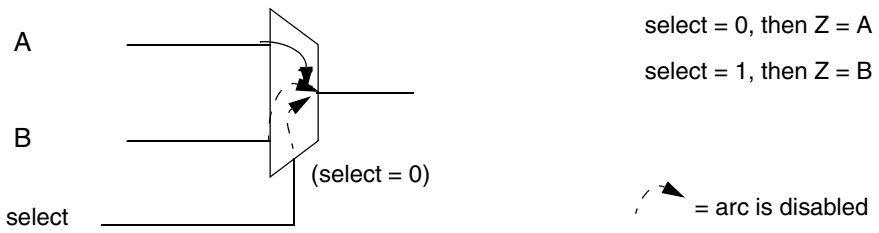
Case analysis lets you perform timing analysis using logic constants or logic transitions on ports or pins to limit the signals propagated through the design. Setting a case analysis with a logic constant propagates the constant forward through the design, then disables paths where the constant is propagated. Setting a case analysis with a logic transition (rising or falling) eliminates certain paths by limiting the transitions considered during analysis.

When you perform case analysis, keep the following points in mind:

- Case analysis propagates the constant values specified on nets forward (but not backward) through the design.
- Case analysis does not affect sequential cells if the library was compiled with Library Compiler v3.3 or earlier.
- Propagating constants across the sequential cells is disabled by default. You can enable it by setting the `case_analysis_sequential_propagation` variable to `always`.
- Case analysis, by default, propagates user set case analysis logic values as well as logic constants from the netlist. You can disable the propagation of logic constants from the netlist by setting the `disable_case_analysis_ti_hi_lo` variable to `true`. You can disable the propagation of all logic values (from the user and netlist) by setting the `disable_case_analysis` variable to `true` and overrides the value of the `disable_case_analysis_ti_hi_lo` variable.

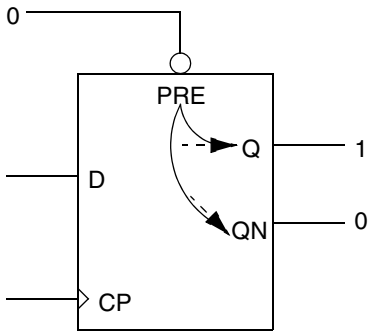
The multiplexer in Figure 10-1 has two signals (A and B) going in, one signal (Z) going out, and 0 is set on a select signal coming into the multiplexer. The select disables the arc from B to Z because the constant is blocking the data from B to Z.

Figure 10-1 Constant Blocking of Data



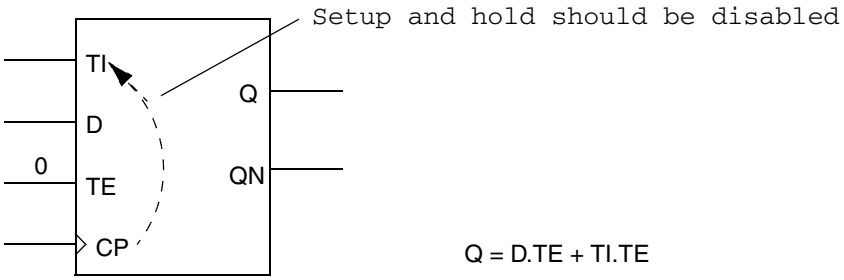
When case analysis propagates a constant to a sequential element's asynchronous preset or clear pin, the sequential cell outputs also propagate the constant 1 or 0, as shown in Figure 10-2.

Figure 10-2 Constant Propagation Through an Asynchronous Input



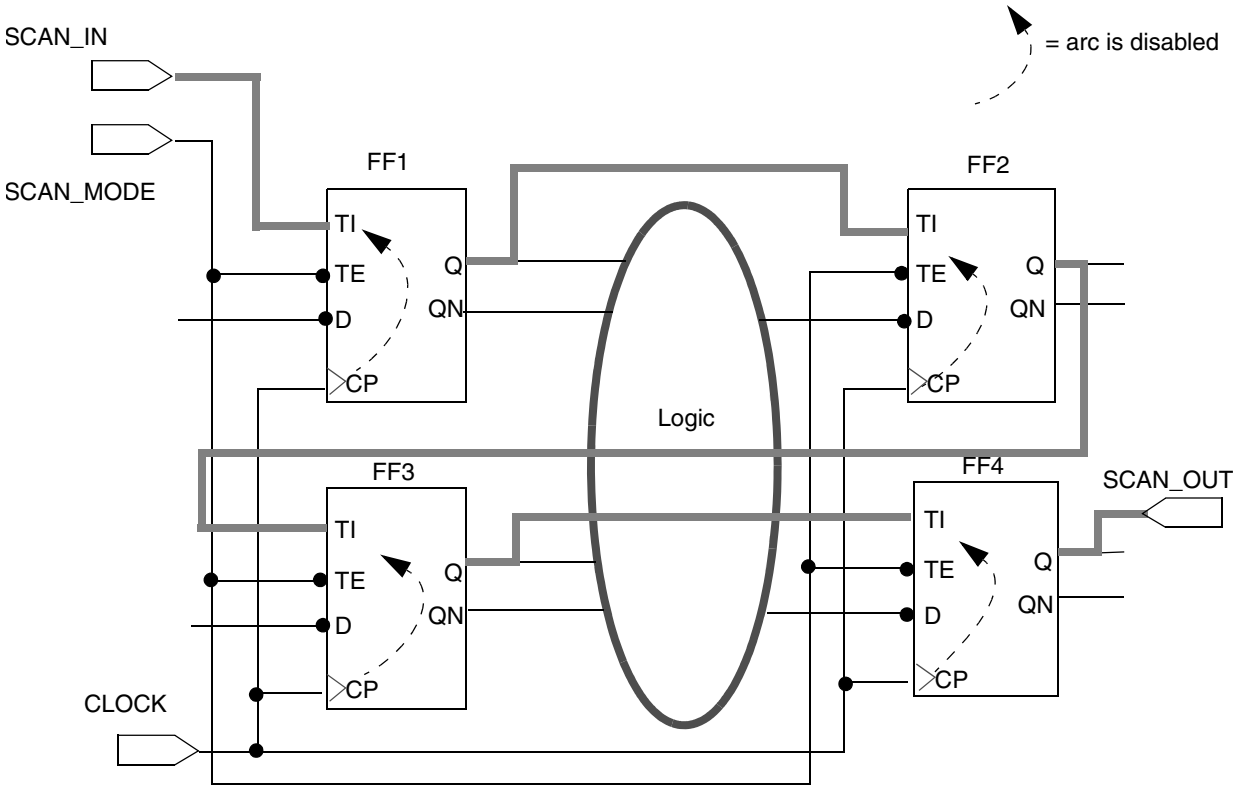
When your design uses scan flip-flops, case analysis is useful for disabling the scan chain. To do this, set the scan mode control pin to the constant value that disables the test mode. PrimeTime propagates the test-mode constant value to the scan-mode pin of each scan flip-flop. Case analysis can determine from the constant values which timing arcs to disable, as shown in Figure 10-3.

Figure 10-3 Case Analysis Disabling Timing Checks



The most common use of case analysis is to disable a scan chain, as shown in Figure 10-4. Setting the SCAN_MODE port to 0 disables the scan chain. As a result, the scan chain is not reported by the `report_timing` command.

Figure 10-4 Using Case Analysis to Disable a Scan Chain



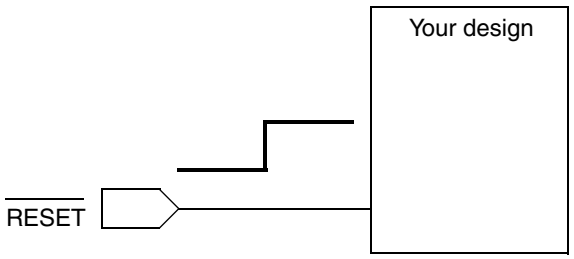
By setting a logic constant of 0 on the SCAN_MODE port, PrimeTime disables the scan chain that starts at SCAN_IN and ends at SCAN_OUT. PrimeTime disables the scan chain because the logic 0 set on the SCAN_MODE port propagates to the FF1/TE, FF2/TE, FF3/TE, and FF4/TE pins. A logic constant on the TE pin of each scan flip-flop disables the setup and hold arcs from CP to TI because of the sequential case analysis on the FF1, FF2, FF3, and FF4 cells.

To set case analysis with a 0 value, enter

```
pt_shell> set_case_analysis 0 [get_ports "SCAN_MODE"]
```

Setting a case analysis for a transition can be useful for analyzing the behavior of a circuit for a specific situation. For example, in the circuit shown in Figure , a rising-edge transition on the reset input brings the device out of the reset mode.

Figure 10-5 Case Analysis for a Rising Edge



To analyze the timing of the circuit coming out of reset mode, you are only concerned about rising edges on the reset input, not logic 0, logic 1, or falling edges. To set case analysis in this situation, enter the following syntax:

```
pt_shell> set_case_analysis rising [get_ports RESET]
```

Note:

The `-rising` and `-falling` options are ignored by PrimeTime PX during power calculation.

Constant Propagation Based on Cell Logic

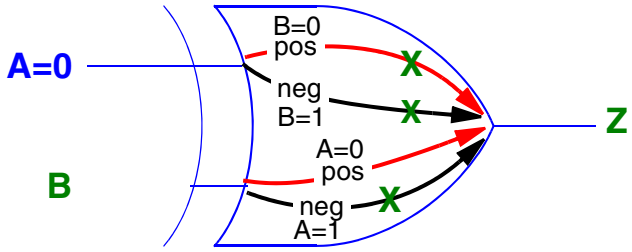
Case analysis can disable timing arcs and propagate case values for the following types of cells:

- Combinational cells with the `function` attribute specified in the library description
- Sequential cells with Boolean State Table (BST) specified in the library description
- Cells with timing arcs that have state-dependent conditions specified using the `when` statement in Library Compiler. (See the Library Compiler user guides for the correct usage of the `when` statement and state-dependent delays.)

  If a cell has any input at a logic constant case value, arcs that could never propagate a transition are disabled. If any other arcs have a `when` condition, those conditions are evaluated. The `when` arcs are disabled if the `when` expression evaluates to false. Note that the `when` statement cannot enable an arc that has been disabled by case analysis.

Figure 10-6 shows how arcs of an XOR gate are disabled when using the `when` statement in the library description. When A = 0, all arcs connected to A, and negative_unate arcs from B to Z are disabled. Note, there might be several arcs of the same sense between the two pins.

Figure 10-6 Disabled Arcs Using the when Statement



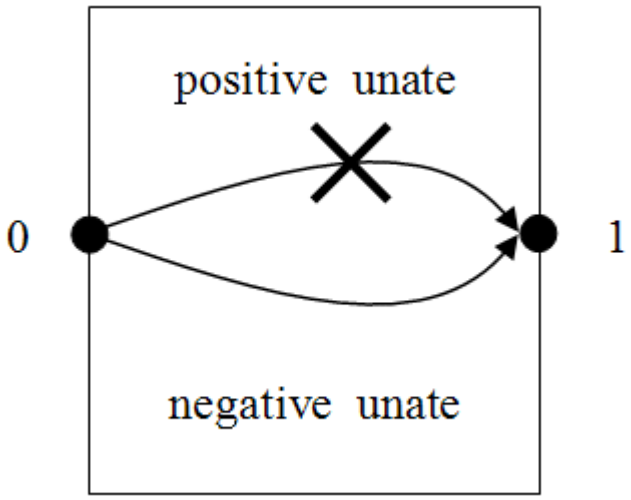
The only enabled arcs are positive_unate arcs from **B** to **Z**.
(**X** identifies disabled arcs)

- Case analysis on three-state cells allows for disabling timing arcs across a three-state cell. In this context, case analysis utilizes the associated three-state logic function in the cell library definition for justifying three-state arcs. The three-state logic function can be commonplace in pad cells, so it is desirable to be able to disable the data path (A->IO) arc when the three-state function turns this path off due to tristating the output.
- Physically invalid arcs

  When case analysis propagates logic values from the inputs of a cell to its outputs, it disables arcs that are physically invalid. Arcs are considered physically invalid if the logic value on the input and output do not align with the sense of that arc. For example, the cell

below has a logic value 0 on its input and a logic value 1 on its output. The sense of this combination of logic values is negative unate. Therefore, the positive unate arc between the pins is physically invalid and is disabled by case analysis as shown in Figure 10-7.

Figure 10-7 Physically Invalid Arc



Case Analysis of Integrated Clock-Gating Cells

As defined by the Library Compiler `clock_gating_integrated_cell` attribute, there are effectively three broad categories of integrated clock-gating cells, distinguished by the internal memory element they utilize. These integrated clock-gating cells are:

- Latch memory element
- Flip-flop memory element
- No memory element

PrimeTime currently supports latched integrated clock-gating cells. These cells are sequential in nature. To enable the propagation of logic values from the input pins of integrated clock-gating cells to their fanout, set the value of the `case_analysis_propagate_through_icg` variable to `true`. To disable the propagation logic values and disable simulation, set the variable back to `false` (the default). Integrated clock-gating cells that have no memory element are not sequential; therefore, they are considered combinational and always propagate.

Two possible situations can arise, regardless of whether the enable pin is active or not:

- If a logic value reaches the input clock pin of the integrated clock-gating cell, it does not propagate to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin does not propagate to the gated clock output pin and then into the fanout of the cell.
- If no logic value reaches the input clock pin of the integrated clock-gating cell, no logic value is propagated to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin is propagated to the gated clock output pin and then into the fanout of the cell.

  When the `case_analysis_propagate_through_icg` variable is set to `true`, integrated clock-gating cells are fully simulated.

  The following situations can arise:

  - When the enable pin is active (logic 1)

    1. If a logic value reaches the input clock pin of the integrated clock-gating cell, it propagates to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin does not propagate to the gated clock output pin and then into the fanout of the cell.
    2. If no logic value reaches the input clock pin of the integrated clock-gating cell, there is no logic value propagated to the gated clock output pin and then into the fanout of the cell. Any clock propagating up to the input clock pin is propagated to the gated clock output pin and then into the fanout of the cell.

  - When the enable pin is inactive (logic 0)

    1. If a logic value reaches the input clock pin of the integrated clock-gating cell, it does not propagate to the gated clock output pin and then into the fanout of the cell. The appropriate logic value is propagated from the gated clock output pin into the fanout of the cell. Any clock propagating up to the input clock pin is not propagated to the gated clock output pin and then into the fanout of the cell.
    2. If no logic value reaches the input clock pin of the integrated clock-gating cell, there is no logic value propagated to the gated clock output pin. The appropriate logic value is propagated from the gated clock output pin into the fanout of the cell. Any clock propagating up to the input clock pin is not propagated to the gated clock output pin and then into the fanout of the cell.

  - When the enable pin is inactive (logic 0) and the cell is disabled, the appropriate logic value propagated in the fanout of the cell depends on the integrated clock-gating cell type. For example, a `latch_posedge` integrated clock-gating cell propagates a logic 0 in its fanout, whereas a `latch_negedge` integrated clock-gating cell propagates a logic 1 in its fanout.

Setting Case Analysis Values

The `set_case_analysis` command sets the logic values in the pins. It specifies that a port or pin is at a constant logic value 1 or 0, or is considered with a rising or falling transition.

When case analysis is specified as a constant value, this value is propagated through the network as long as the constant value is a controlling value for the traversed logic. For example, if you specify that one of the inputs of a NAND gate is a constant value `0`, it is propagated to the NAND output, which is considered at a logic constant `1`. This propagated constant value is then propagated to all cells driven by this signal.

Note:

When a logic value is propagated onto a net, the associated design rule checks are disable; however, you can perform maximum capacitance design rule checks on driver pins for constant nets by setting the `timing_enable_max_capacitance_set_case_analysis` variable to `true`.

For more information about commands and variables, see the specific man page.

To perform timing analysis for system mode by setting the case analysis for the test port to constant logic 0, enter

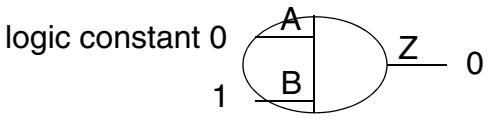
```
pt_shell> set_case_analysis 0 Test
```

You can use case analysis in combination with mode analysis (see Chapter 11, “Mode Analysis”). For example, use case analysis to specify that certain internal logic is a constant value because the RAM block is in read mode:

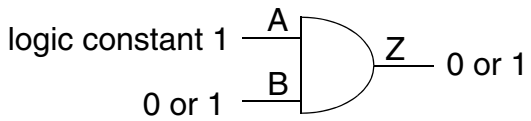
```
pt_shell> set_case_analysis 1 U1/U2/select_read
```

As shown in the following examples, if there are logic constants and case analysis on inputs of a cell, and the logic constants are set to any controlling logic value of that cell, then the logic constant is propagated from the output; otherwise case analysis is propagated.

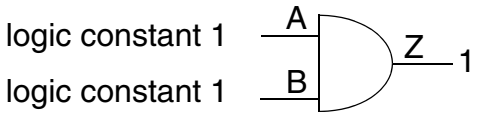
If you have an AND gate and input pin A has logic constant 0 (controlling the logic value of the AND gate), the logic constant 0 is propagated from the output pin, irrespective of the value on B. For example,



If you have the same AND gate, but pin A has logic constant 1, it is not a controlling logic value; the logic constant is not propagated to the output pin. For example,



If you have the same AND gate and both pins A and B have logic constant 1 (noncontrolling value), then a logic constant of 1 is propagated from the output of the gate. For example,

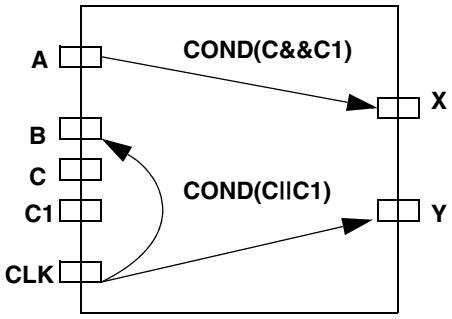


Evaluating Conditional Arcs Using Case Analysis

You can enable or disable conditional arcs from Liberty models by performing case analysis. Liberty is a modeling language that describes the static timing characteristics of large blocks for which there is no corresponding gate-level netlist. If an arc has a condition associated with it, PrimeTime evaluates the condition. If the design does not meet the condition for the arc, PrimeTime disables the conditional arc. Note that conditional testing cannot enable an arc that has been disabled by case analysis.

For example, in Figure 10-8, a delay arc exists from A to X with condition COND(C&&C1). PrimeTime disables the arc when the Boolean expression (C&&C1) evaluates to false. A setup arc also exists from CLK to B with condition COND(C||C1). PrimeTime disables the setup arc when the Boolean expression (C||C1) evaluates to false. The arc from CLK to Y is a nonconditional arc.

Figure 10-8 Conditional and Nonconditional Arcs



Examples

The following commands cause PrimeTime to disable the delay arc from A to X because it evaluates the condition to be false:

```
pt_shell> set_case_analysis 0 C
pt_shell> set_case_analysis 1 C1
```

The following command enables the delay arc from A to X and the setup arc from CLK to B:

```
pt_shell> set_case_analysis 1 {C C1}
```

For more information about specifying conditional arcs in the Liberty modeling language, see the *Library Compiler Technology and Symbol Libraries Reference Manual*.

Reporting Case Analysis Values

The `report_case_analysis` command reports the case analysis values.

Example

To report the case analysis values, enter

```
pt_shell> report_case_analysis
```

PrimeTime displays a report similar to this:

```
****************************************
Report : case_analysis
Design : false_path
****************************************

Pin name                          Case analysis value
-------------------------------------------------
```

```
test_port                          0
U1/U2/core/WR                      1
```

The `report_case_analysis -all` shows both the logic constants and case analysis set on the design. For example,

```
pt_shell> report_case_analysis -all
****************************************
Report : case_analysis
         -all
Design : TOP
Version: 2000.11
Date   : Wed Nov  15 11:23:37 2000
****************************************

Pin name                               Logic constant value
-------------------------------------------------------------

U3/Logic0/output                       0

Pin name                               User case analysis value
-------------------------------------------------------------
HRS                                    1
ALARM                                  1
MINS                                   1
```

Removing Case Analysis Values

The `remove_case_analysis` command removes case analysis values. For example, to remove case analysis values from the test_port design, enter the following syntax:

```
pt_shell> remove_case_analysis test_port
```

To suppress propagating logic constants (including those set with case analysis and pins tied to logic high or low), set the `disable_case_analysis` variable to `true`. You can use this feature when you want to write scripts for Design Compiler that can be synthesized correctly.

Constant Propagation Log File

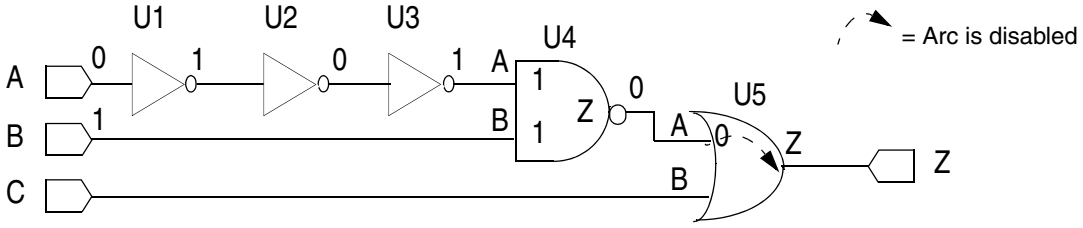
When PrimeTime performs constant propagation, it disables timing arcs at the boundary of the domain where a constant propagates. For this reason, when PrimeTime disables a timing arc, knowing the exact origin of the constant value propagated on a given pin of the design might be difficult.

To help you determine the cause of a timing arc disabled because of constant propagation, you can set the `case_analysis_log_file` variable to a file name. The constant propagation process is logged to the named file. You can also view the timing arcs disabled after constant propagation with the `report_disable_timing` command.

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.

```
pt_shell> set_case_analysis 0 {get_port "A"}
pt_shell> set_case_analysis 1 {get_port "B"}
pt_shell> set case_analysis_log_file my_design_cnst.log
```



In the example, the only arc that PrimeTime needs to disable is the arc in U5 from U5/A to U5/Z. PrimeTime needs to disable this arc because U5/A is at constant value 0 and no constant is propagated to U5/Z. The constant propagation log file my_design_cnst.log looks like this:

```
****************************************
Report : case_analysis propagation
Design : my_design
Version: 2000.11
****************************************

1.1 Forward propagation on NET pins (level 1)
-----------------------------------------------
  Propagating logic constant '0' from pin 'A' on net 'A':
    > pin 'U1/A' is at logic '0'
  Propagation of logic constant '1' from pin 'B' on net 'B':
    > pin 'U4/B' is at logic '1'

1.2 Forward propagation through CELLS (level 1)
-------------------------------------------------
  Cell 'U1' (libcell 'IV') :
    input pin U1/A is at logic '0'
    > output pin 'U1/Z' is at logic '1'

2.1 Forward propagation on NET pins (level 2)
```

```
-----------------------------------------------
  Propagating logic constant '1' from pin 'U1/Z' on net
'w1':
    > pin 'U2/A' is at logic '1'

2.2 Forward propagation through CELLS (level 2)
-----------------------------------------------
  Cell 'U2' (libcell 'IV') :
    input pin U2/A is at logic '1'
    > output pin 'U2/Z' is at logic '0'

3.1 Forward propagation on NET pins (level 3)
-----------------------------------------------
  Propagating logic constant '0' from pin 'U2/Z' on net
'w2':
    > pin 'U3/A' is at logic '0'

3.2 Forward propagation through CELLS (level 3)
-----------------------------------------------
  Cell 'U3' (libcell 'IV') :
    input pin U3/A is at logic '0'
    > output pin 'U3/Z' is at logic '1'

4.1 Forward propagation on NET pins (level 4)
-----------------------------------------------
  Propagating logic constant '1' from pin 'U3/Z' on net
'w3':
    > pin 'U4/A' is at logic '1'

4.2 Forward propagation through CELLS (level 4)
-----------------------------------------------
  Cell 'U4' (libcell 'ND2') :
    input pin U4/A is at logic '1'
    input pin U4/B is at logic '1'
    > output pin 'U4/Z' is at logic '0'

5.1 Forward propagation on NET pins (level 5)
-----------------------------------------------
  Propagating logic constant '0' from pin 'U4/Z' on net
'w4':
    > pin 'U5/A' is at logic '0'

5.2 Forward propagation through CELLS (level 5)
-----------------------------------------------

6. End of Logic Constant Propagation
------------------------------
```

The result of the `report_disable_timing` command looks similar to the following:

```
Cell or Port              From        To          Flag  Reason
-----------------------------------------------------------------
```

```
U5                          A           Z           c     A = 0
```

Interaction of Case Analysis With Design Rule Checking

To check the maximum capacitance on the constant propagation path, set the `timing_enable_max_capacitance_set_case_analysis` variable to `true`.

11

Mode Analysis

Timing models and library cells can have operating modes defined in them, such as read and write modes for a RAM cell. Each mode has an associated set of timing arcs that PrimeTime analyzes while that mode is active.

Mode analysis techniques are described in the following sections:

Mode Analysis Overview

Library cells and timing models can have operating modes defined in them, such as read and write modes for a RAM cell. Each mode has an associated set of timing arcs that PrimeTime analyzes while that mode is active. The timing arcs associated with inactive modes are not analyzed. In the absence of any mode settings, all modes are active and all timing arcs are analyzed.

There are two kinds of modes that can be set: cell modes and design modes. Cell modes are defined in a timing model or library cell, such as the read and write modes for a RAM cell. Design modes are user-defined modes that exist at the design level, such as normal and test modes.

Each design mode can be mapped to a set of cell modes in cell instances or to a set of paths. When a design mode is active, all cell modes mapped to that design mode are activated and all paths mapped to that design modes are enabled. When a design mode is inactive (due to selection of a different design mode) then all cell modes mapped to that design mode are made inactive and all paths mapped to that design mode are set to false paths.

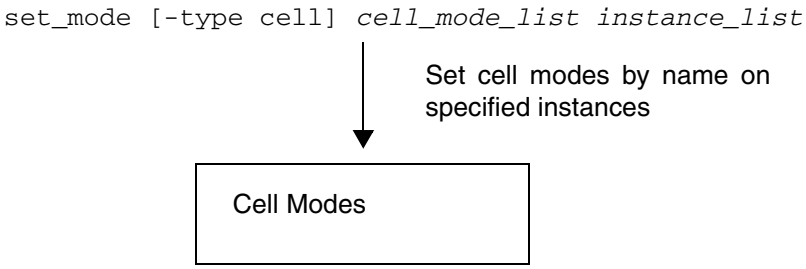
There are different methods for specifying cell operating modes in a design. In order of priority, from highest to lowest, these methods are:

1. Set cell modes directly on cell instances with the `set_mode -type cell` command. You can optionally omit `-type cell` from the command because it is the default mode type. For example, `set_mode READ {U1}` sets the READ mode on cell instance U1.
2. Set cell modes indirectly using design modes with the `set_mode -type design` command. To create design modes, use `define_design_mode_group`. Mapping can be done by specifying a list of instances or a list of paths. To map design modes into cell modes applied to instances or into paths that are enabled/disabled in the design, use `map_design_mode`.
3. Place cells into modes using case analysis. For example, if a cell U1 has a cell mode called READ which is defined to be active when input RW is 0, setting or propagating a case analysis value of 0 on the U1/RW input activates the READ mode and deactivates all other modes for that cell. The `set_case_analysis 0 [get_pins U1/RW]` command sets a logic 0 directly on the RW pin of U1.

The following figures show how to select modes:

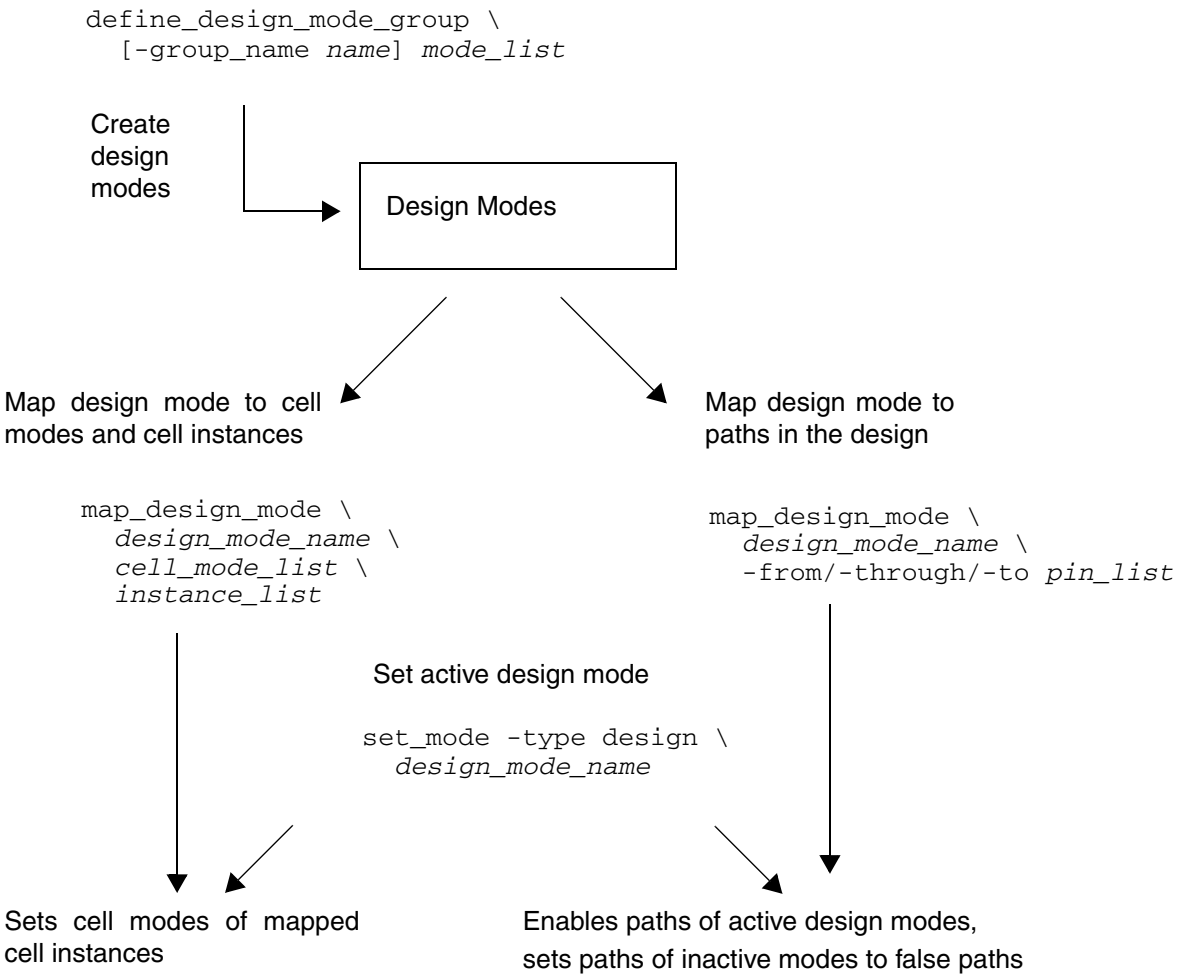
- Figure 11-1 shows how to select modes in specific cell instances by name. This mode selection method has the highest priority.

Figure 11-1 Selecting Modes Directly on Cell Instances



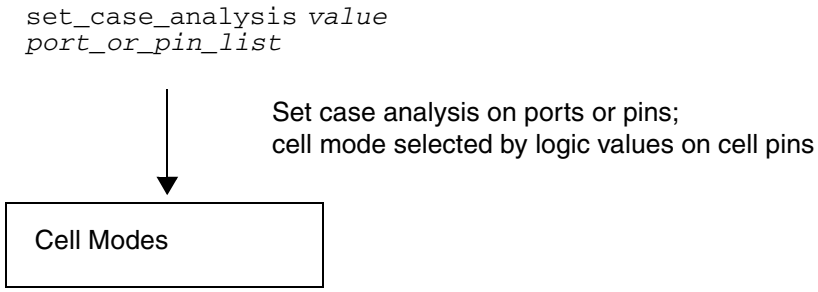
- Figure 11-2 shows how to create, map, and select design modes. This mode selection method has medium priority.

Figure 11-2 Selecting Modes Using Design Modes



- Figure 11-3 shows how to select modes by using case analysis, taking advantage of the conditional modes defined in the cell. This mode selection method has the lowest priority.

Figure 11-3 Selecting Modes Using Case Analysis



How Cell Modes Are Defined

Library cells and timing models created with the Liberty modeling language can contain operating modes. To define modes for library cells, use the mode specification features of Library Compiler. See the Library Compiler documentation for more information. To find out about the available cell modes and design modes in the current design, use the `report_mode -type cell` or `report_mode -type design command`.

Mode Groups

Modes are organized into groups. Within each group, there are two possible states: all mode enabled (the default), or one mode enabled and all other modes disabled. Often there is only one mode group.

A library cell with modes can have one or more mode groups defined. Each cell mode group has a number of cell modes. Each cell mode is mapped to a number of timing arcs in the library cell. Every instance of that library cell has these cell mode groups together with the cell modes. For example, a typical RAM block can have read, write, latched, and transparent modes. You can group the read and write modes, then group the latched and transparent modes in a different mode group. The advantage of grouping modes is that when you set a cell mode, PrimeTime makes the corresponding mutually exclusive modes inactive.

For example, specifying the RAM block for the read mode implicitly specifies that the write mode is inactive, irrespective of any setting for the transparent and latched modes. Similarly, specifying the RAM block for the latched mode implies that transparent mode is inactive, irrespective of the read and write mode setting. The two mode groups (read/write and transparent/latched) are independent.

By default, all cell modes are enabled in each cell instance. Using the `set_mode -type cell` command, you can set each cell mode group to have one mode enabled and all other modes disabled. When a mode is disabled, all timing arcs in that cell mapped to that mode are disabled (not analyzed for timing).

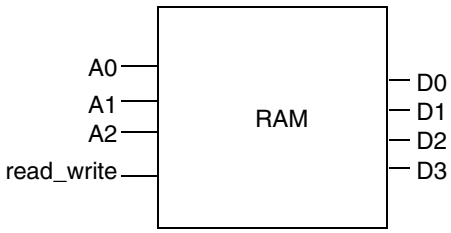
Setting Modes Using Case Analysis

Some library cells and timing models have conditional modes defined in them. This means that a mode selection is invoked by the logical conditions at the cell inputs. For example, the read mode of a RAM cell could be invoked by a logic 0 on the read/write input pin.

When you set the controlling logic value on the input pin using case analysis (or when case analysis values are propagated to that pin), the cell is implicitly placed into the appropriate analysis mode.

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 Liberty cell can be operated in read and write modes.



The `read_write` pin of the Liberty cell controls the read/write mode of the RAM. If the RAM is modeled in the Liberty modeling language with the following mode and condition association, you can perform the read and write mode analysis by setting logic values on the `read_write` input pin:

```
cell(RAM) {
  mode_definition(read_write) {
    mode_value(read) {
      when : "read_write";
      sdf_cond : "read_write == 0";
    }
    mode_value(write) {
      when : "read_write";
      sdf_cond : "read_write == 1";
    }
  }
...
}
```

To enable the read mode in PrimeTime, enter

```
pt_shell> set_case_analysis 0 [get_ports read_write]
```

To enable the write mode in PrimeTime, enter

```
pt_shell> set_case_analysis 1 [get_ports read_write]
```

Setting or propagating a logic value to the read_write pin implicitly selects the corresponding mode and disables the other mode. Only the timing arcs associated with the active mode are used in the timing analysis.

When no modes in a mode group are selected by case analysis or other mode selection methods, all of the modes are enabled. The default mode is enabled if no mode is selected.

Setting Modes Directly on Cells

To specify the active cell modes directly for cell instances in the design, use the following command:

```
pt_shell> set_mode -type cell cell_mode_list instance_list
```

cell_mode_list is a list of modes to be made active, no more than one mode per mode group. If `cell` has only one mode group, you can list only one mode to be made active. `instance_list` is a list of instances in the design to be placed into the specified mode. You can optionally omit `-type cell` because `cell` is the default mode type. (You must specify `-type design` to set a design mode). For example, to set the U1/U2/core cell to read mode, enter

```
pt_shell> set_mode read U1/U2/core
```

This makes the read mode active and all other modes in the mode group inactive for the U1/U2/core cell.

To cancel the mode selection and make all modes active for the U1/U2/core cell, enter

```
pt_shell> reset_mode U1/U2/core
```

Defining and Setting Design Modes

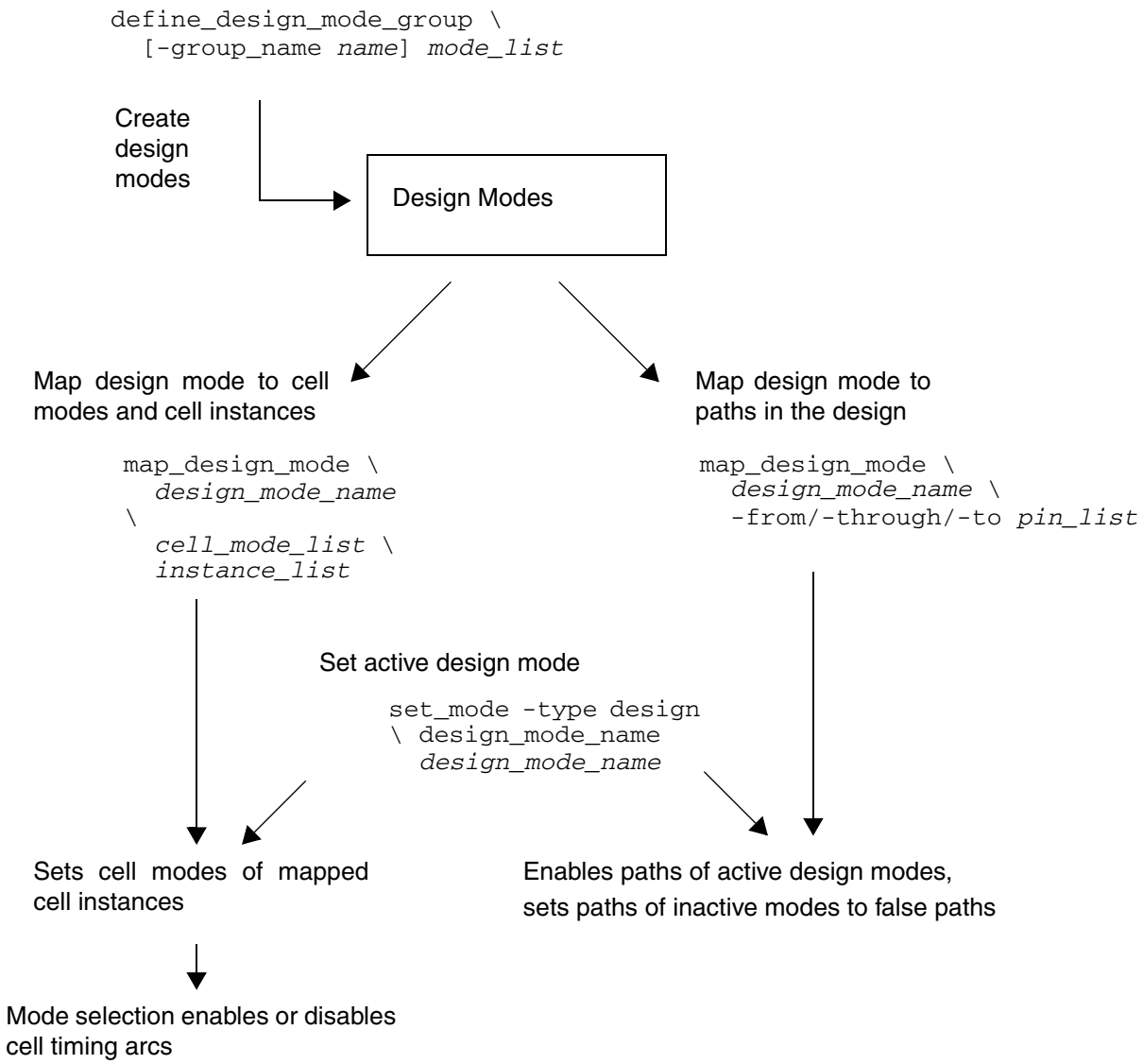
Design modes are user-defined modes that exist at the design level, such as normal and test modes for a chip design. You create design modes in PrimeTime at the design level and map them to cell instance design modes or to paths in the design. When you set a design mode, the associated cell instances are set to the mapped cell modes, or the associated paths are enabled and paths associated with other design modes are set to false paths.

To define, map, and set a design mode,

1. Specify a set of modes for the current design using the `define_design_mode_group` command.
2. For each design mode, specify the associated cell modes or paths with the `map_design_mode` command.
3. Set the current design mode using the `set_mode -type design` command.

This process is summarized in Figure 11-4.

Figure 11-4 Defining, Mapping, and Setting Design Modes



Defining Design Modes

To create a new set of design modes, use the `define_design_mode_group` command. In the command, specify the list of design modes. This is the command syntax:

```
pt_shell> define_design_mode_group [-group_name name] mode_list
```

If you are planning to have more than one design mode group, assign a unique group name for the set of design modes, and use another `define_design_mode_group` command for each design mode group. Otherwise, you can omit the `-group_name` *`name`* option and just specify the list of design modes in a single command. In that case, the design modes are assigned to a group called default. For example, to create design modes called read and write, use the following command:

```
pt_shell> define_design_mode_group {read write}
```

To remove one or more design modes defined previously, use the `remove_design_mode` *`mode_list`* command.

Mapping Design Modes

After you create design modes with the `define_design_mode_group` command, you need to specify what each design mode does. This process is called mapping. You can map a design mode either to a set of cell modes and cell instances or to a set of paths.

Mapping a Design Mode to Cell Modes and Instances

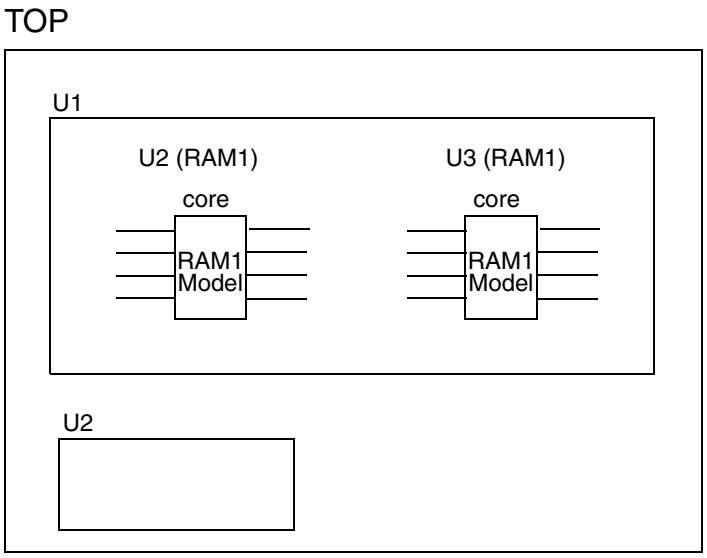
When you map a design mode to a cell mode and cell instances, activating that design mode applies the specified cell mode to the cell instances. To map a design mode in this manner, use the following command syntax:

```
map_design_mode design_mode_name \
   cell_mode_list instance_list
```

Specify the design mode name, the cell mode to be applied, and the list of instances on which to apply the cell mode. If the cell has more than one mode group, you can specify up to one cell mode per mode group in the `cell_mode_list`. Otherwise, specify just one cell mode to be applied to the cell instances. Use a separate `map_design_mode` command to map each design mode. For example, consider the design shown in Figure 11-5. To map the `execute` design mode to invoke the `read_ram` cell mode in the cell instance U1/U2/core, use the following command:

```
pt_shell> map_design_mode execute read_ram U1/U2/core
```

Figure 11-5 Mapping a Design Mode to a Cell Mode and Cell Instance



After this mapping, setting the design mode called “execute” makes that design mode active, causing the cell mode read to be applied to the U1/U2/core instance. This also deactivates the other design modes in the same design mode group as execute, causing those design modes to return to their default state.

Mapping a Design Mode to a Set of Paths

When you map a design mode to a set of paths, activating that design mode activates those paths. Paths that are mapped to the other design modes in the same design mode group are set to false paths. These false paths are, therefore, removed from the timing analysis.

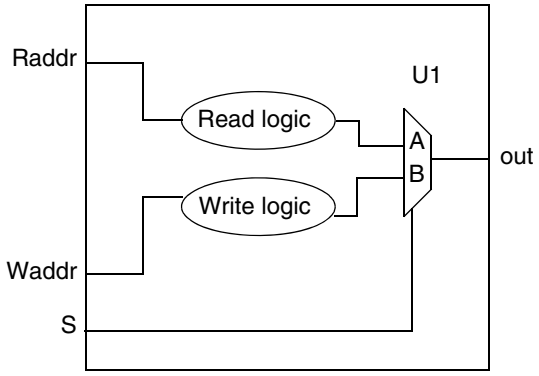
To map a design mode to a set of paths:

```
map_design_mode design_mode_name \
  [-from pin_list ] [-through pin_list ] [-to pin_list ]
```

Specify the design mode name and the set of paths associated with that design mode. Use a separate `map_design_mode` command to map each design mode. For example, consider the design shown in Figure 11-6. To map the design mode read to the set of paths through the read logic, and the design mode write to the set of paths through the write logic, enter,

```
pt_shell> map_design_mode read -from Raddr -through U1/A
pt_shell> map_design_mode write -from Waddr -through U1/B
```

Figure 11-6 Extracting Modes Using the -through Option



After this mapping, setting the design mode called read enables the paths through the read logic. At the same time, it deactivates the write design mode (because read and write belong to the same mode group), causing the associated paths through the write logic to become false paths. Setting the write design mode has the opposite effect.

Unmapping a Design Mode

If you want to cancel or change the mapping of a design mode, you can do so with the `remove_design_mode` command. To remove one or more cell instances previously mapped to a design mode, use the following syntax:

`remove_design_mode` *`design_mode_name`* `cell_mode_name` *`instance_list`*

To remove one or more paths previously mapped to a design mode, use the `remove_design_mode` command.

Removing the mapping of a design mode reverses the effects of any cell mode settings or false paths applied as a result of that mapping. To remove one or more design modes entirely, including all their mapping, use the following syntax:

`remove_design_mode` *`design_mode_list`*

Removing a design mode reverses the effects of any cell mode settings or false paths applied as a result of activating or deactivating that design mode.

Setting Design Modes

After you create design modes with the `define_design_mode_group` command and map all these modes with the `map_design_mode` command, you can set the active mode with the `set_mode -type design` command. Setting a design mode makes that mode active and makes the other design modes in the same design mode group inactive.

To set the current design mode, use the following syntax:

`set_mode -type design` ***`design_mode_name`***

If there are multiple design mode groups, you can specify a list of design mode names, one per mode group. Otherwise, just specify a single design mode.

For a design mode mapped to cell modes and cell instances, the cell modes are applied to the associated instances. Other design modes in the same design mode group are made inactive, causing the associated cell instances to be returned to their default state.

For a design mode mapped to paths, the associated paths are enabled. Other design modes in the same design mode group are made inactive, causing the associated paths to become false paths that are not analyzed for timing.

To cancel the effects of a `set_mode -type design` command, use the following command syntax:

`reset_mode -type design -group` ***`group_name`***

or

`reset_mode -type design` ***`design_mode_name`***

If there are multiple design mode groups, you can list multiple group names or multiple design mode names, one per group. Otherwise, specify just a single group name or design mode name.

Resetting the design mode to the default state causes no mode to be selected, which means that no cell modes are applied and no paths are set to be false paths.

Reporting Modes

The `report_mode` command reports on modes that have been defined or set in the design. To get a report on cell modes, use `report_mode -type cell` or just `report_mode` alone. To get a report on design modes, use `report_mode -type design`. For example, to get a cell mode report:

```
pt_shell> report_mode

...
Cell               Mode(Group)   Status     Condition   Reason
-------------------------------------------------------------------
Core (LIB_CORE)    read(rw)      disabled   (A==1)      cond
                   oen-on(rw)    ENABLED    (A==0)      cond
                   write(rw)     disabled   (B==1)      cond
Core2 (LIB_CORE)   read(rw)      disabled   (A==1)      cell
                   oen-on(rw)    ENABLED    (A==0)      cell
                   write(rw)     disabled   (B==1)      cell
Core3 (LIB_CORE)   read(rw)      disabled   (A==1)      design - dm1
                   oen-on(rw)    ENABLED    (A==0)      design - dm1
                   write(rw)     disabled   (B==1)      design - dm1
Core4 (LIB_CORE)   read(rw)      disabled   (A==1)      default
                   oen-on(rw)    ENABLED    (A==0)      default
                   write(rw)     disabled   (B==1)      default
-------------------------------------------------------------------
```

The report shows the name of each cell instance with a mode setting, the possible mode settings and group names for the cell, the current status of each mode (enabled or disabled), the condition causing the current status, and the reason for the mode setting. The possible reasons are:

- cond – The mode is set conditionally by case analysis.
- cell – The mode is set directly by `set_mode -type_cell`.
- design – The mode is set by the indicated design mode.
- default – The default mode setting is in effect.

Here is an example of a design mode report:

```
pt_shell> map_design_mode {read,latching} {U2/core,U1/core} read
pt_shell> map_design_mode -from {INFF1/CLK} -to {INFF3/D}
pt_shell> map_design_mode -from {INFF1/CLK} -to {INFF4/D}
pt_shell> map_design_mode {write,transparent} {U2/core,U1/core} write
pt_shell> report_mode -type design
...
-------------------------------------------------------------------
Design Mode Group : 'default'
      Design Mode : 'read' (is current design mode)
```

```
      ------------------------------------------------------------
         Cell                          Mode(Group)
         ------------------------------------------------------------
         U2/core                       latching(output_latch)
                                       read(rw)
         ------------------------------------------------------------
         U1/core                       latching(output_latch)
                                       read(rw)
         From Pin
         ------------------------------------------------------------
         INFF1/CLK
         INFF1/CLK

         To Pin
         ------------------------------------------------------------
         INFF3/D
         INFF4/D
---------------------------------------------------------------------

---------------------------------------------------------------------
Design Mode Group : 'default'
      Design Mode : 'write'
      ------------------------------------------------------------
         Cell                          Mode(Config)
         ------------------------------------------------------------
         U2/core                       transparent(output_latch)
                                       write(rw)
         ------------------------------------------------------------
         U1/core                       transparent(output_latch)
                                       write(rw)
---------------------------------------------------------------------
```

12

Operating Conditions

Minimum-maximum analysis is a technique that lets you perform both minimum-delay and maximum-delay checking in a single analysis run, taking into account the two extremes in operating conditions.

PrimeTime operating conditions are described in the following sections:

Operating Condition Analysis Modes

Semiconductor device parameters can vary with conditions such as fabrication process, operating temperature, and power supply voltage. In PrimeTime, the `set_operating_conditions` command specifies the operating conditions for analysis, so that PrimeTime can use the appropriate set of parameter values in the technology library.

PrimeTime offers three analysis modes with respect to operating conditions, called the single, best-case/worst-case, and on-chip variation (OCV) modes:

- In the single operating condition mode, PrimeTime 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, PrimeTime 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. This mode lets you check both extremes in a single analysis run, thereby reducing overall runtime for a full analysis.
- In the OCV mode, PrimeTime performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For a setup check, it uses maximum delays for the launch clock path and data path, and minimum delays for the capture clock path. For a hold check, it uses minimum delays for the launch clock path and data path, and maximum delays for the capture clock path.

Table 12-1 and Table 12-2 show the clock arrival times, delays, operating conditions, and delay derating used for setup checks and for hold checks under each of the operating condition analysis modes.

Table 12-1 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.

Table 12-1 Timing Parameters Used for Setup Checks (Continued)

Analysis mode	Launch clock path	Data path	Capture clock path
OCV 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 12-2 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.
OCV 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

The `set_operating_conditions` command defines the operating conditions for timing analysis and specifies the analysis type: single, best-case/worst-case, or OCV. The operating conditions must be defined in a specified library or pair of libraries.

By default, PrimeTime performs analysis under one set of operating conditions at a time (single operating condition mode). Using this mode, you need to perform multiple analysis runs to handle multiple operating conditions. Typically, you need to analyze at least two operating conditions to ensure that the design has no timing violations: best case (minimum path report) for hold checks and worst case (maximum path report) for setup checks.

PrimeTime can simultaneously perform timing analysis for the best-case and worst-case operating conditions. Compared to running single operating condition analysis, you can save time by using minimum-maximum analysis in a single timing analysis run to report timing paths at the two extremes of operating conditions. You can choose either best-case/worst-case analysis or OCV analysis.

In the best-case/worst-case mode, each setup check uses delays computed at the maximum operating condition and each hold check uses delays computed for at the minimum operating condition.

In the OCV mode, PrimeTime performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For setup checks, it uses maximum delays for the launch clock path and data path, and minimum delays for the capture clock path. For hold checks, it uses minimum delays for the launch clock path and data path, and maximum delays for the capture clock path.

In the OCV mode, when a path segment is shared between the clock paths that launch and capture data, the path segment might be treated as having two different delays at the same time. Not accounting for the shared path segment can result in a pessimistic analysis. For a more accurate analysis, this pessimism can be corrected. For more information, see “Clock Reconvergence Pessimism Removal” on page 12-19.

A minimum-maximum analysis considers the minimum and maximum values specified for the following design parameters:

- Input and output external delays
- Delays annotated from Standard Delay Format (SDF)
- Port wire load models
- Port fanout number
- Net capacitance
- Net resistance
- Net wire load model
- Clock latency
- Clock transition time
- Input port driving cell

For example, to calculate a maximum delay, PrimeTime uses the longest path, worst-case operating conditions, latest-arriving clock edge, maximum cell delays, longest transition times, and so on.

You can perform minimum-maximum analysis using a single library with minimum and maximum operating conditions specified, or two libraries, one for the best-case conditions sand one for the worst-case conditions. For more information, see “Using Two Libraries for Analysis” on page 12-14.

Enable minimum-maximum analysis by using one of these methods:

- Set the minimum and maximum operating conditions with the `set_operating_conditions` command:

```
pt_shell> set_operating_conditions -min BCCOM \
          -max WCCOM
```

- Read in an SDF file, then use the option of the SDF reader to read both the minimum and maximum delay from the SDF file:

```
pt_shell> read_sdf -analysis_type bc_wc my_design.sdf
```

Minimum-Maximum 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 either of the following ways:

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

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

- Minimum and maximum from the SDF triplet
- Two SDF files
- Either of the preceding choices, with additional multipliers for maximum and minimum values

To have PrimeTime calculate the delay, use one of the following:

- A single operating condition with timing derating factors to model variation
- Two operating conditions (best-case and worst-case) to model the possible OCV
- Either of the preceding choices with timing derating factors for the minimum and maximum value
- Separate derating factors for cells versus nets
- Separate derating factors for different timing checks (setup, hold, and so forth)

Table 12-3 summarizes the usage of minimum and maximum delays from SDF triplet data.

Table 12-3 Minimum-Maximum Delays From SDF Triplet Data

Analysis mode	Delay based on operating conditions	One SDF file	Two SDF files
Single operating condition	Setup - max data at operating condition - min capture clock at operating condition	Setup - (a:b:***c***) - (a:b:***c***)	
	Hold - min data at operating condition - max capture clock at operating condition	Hold - (a:b:***c***) - (a:b:***c***)	
Best case–worst case	Setup - max data at worst case - min capture clock at worst case	Setup - (a:b:***c***) - (a:b:***c***)	Setup SDF1 - (a:b:***c***) SDF2 - (a:b:***c***)
	Hold - min data at best case - max capture clock at best case	Hold - (***a***:b:c) - (***a***:b:c).	Hold SDF1 - (***a***:b:c) SDF2 - (***a***:b:c)
On-chip variation	Setup - max data at worst case - min capture clock at best case	Setup - (a:b:***c***) - (***a***:b:c)	Setup SDF1 - (a:b:***c***) SDF2 - (**a**:b:c)
	Hold - min data best case - max capture clock at worst case	Hold - (***a***:b:c) - (a:b:***c***)	Hold SDF1 - (***a***:b:c) SDF2 - (a:b:***c***)

The triplet displayed in ***bold italic*** is the triplet that PrimeTime uses.

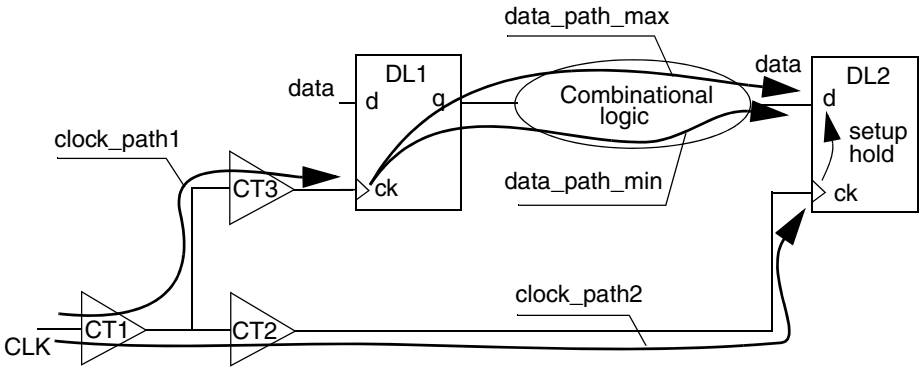
Setup and Hold Checks

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 OCV.

Path Delay Tracing for Setup and Hold Checks

Figure 12-1 shows how setup and hold checks are done in PrimeTime.

Figure 12-1 Design Example



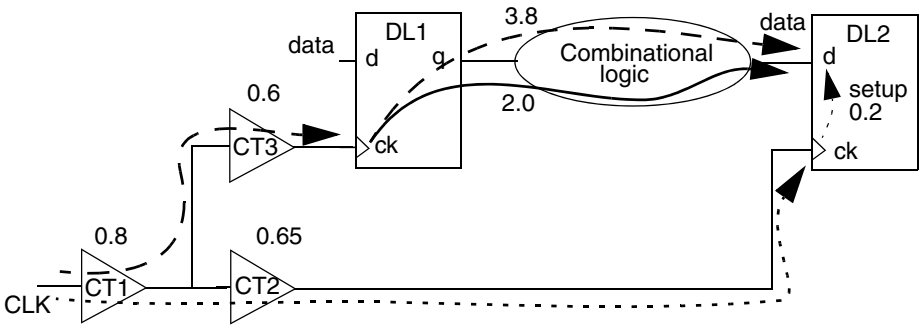
- The setup timing check from pin DL2/ck to DL2/d considers
  - Maximum delay for clock_path1
  - Maximum delay for data path (data_path_max)
  - Minimum delay for clock_path2
- The hold timing check from pin DL2/ck to DL2/d considers
  - Minimum delay for clock_path1
  - Minimum delay for data path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

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

Setup Timing Check for Worst-Case Conditions

Figure 12-2 shows how cell delays are computed for worst-case conditions. To simplify the example, the net delays are ignored.

Figure 12-2 Setup Check Using Worst-Case Conditions



PrimeTime checks for a setup violation as follows:

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

In the equation,

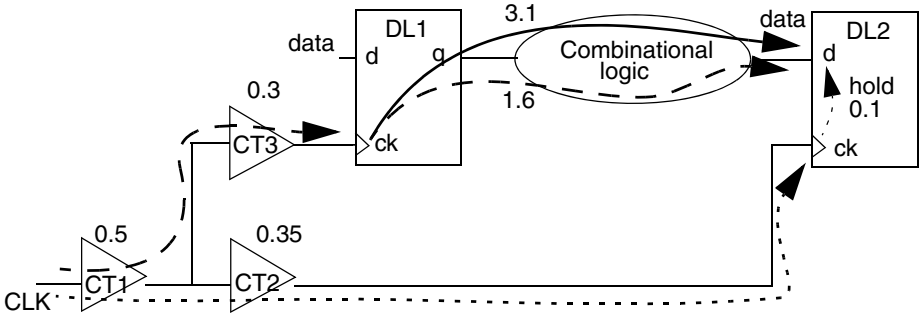
clockpath1 = 0.8 + 0.6 = 1.4
datapathmax = 3.8
clockpath2 = 0.8 + 0.65 = 1.45
setup = 0.2

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 12-3 shows how cell delays are computed for best-case conditions. Note that the cell delays are different from the delays in Figure 12-2.

Figure 12-3 Hold Check Using Best-Case Conditions



PrimeTime checks for a hold violation as follows:

$$\text{clockpath1} + \text{datapathmin} - \text{clockpath2} - \text{hold} \geq 0$$

In the equation,

clockpath1 = 0.5 + 0.3 = 0.8
datapathmin = 1.6
clockpath2 = 0.5 + 0.35 = 0.85
hold = 0.1

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

Simultaneous Best-Case/Worst-Case Conditions

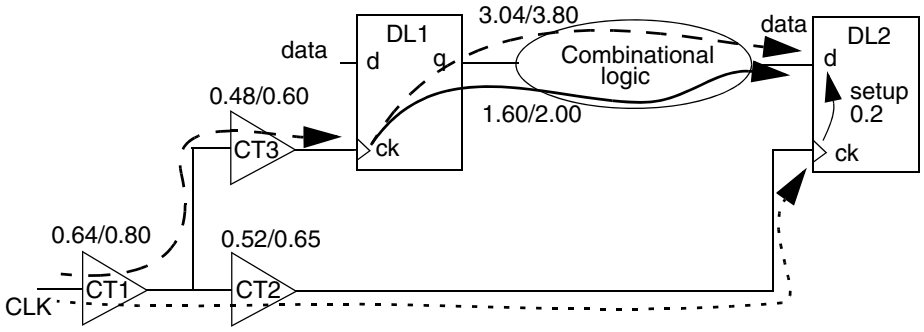
PrimeTime can operate in a mode that computes delays simultaneously for best-case and worst-case conditions for hold and setup checks, enabling you to check both conditions in one timing analysis run. In terms of path delay tracing, PrimeTime uses worst-case conditions for setup checks and best-case conditions for hold checks. The timing reports for setup and hold checks are the same as for Figure 12-2 and Figure 12-3.

In simultaneous mode, PrimeTime does not compare data arrival at worst-case conditions to clock arrival at best-case conditions. In this mode, the timing reports show delays computed in the same operating condition (worst case for setup or best case for hold).

Path Tracing in the Presence of Delay Variation

In Figure 12-4, each delay of a cell or a net has an uncertainty because of OCV. For example, you can specify that OCV can be between 80 percent and 100 percent of the nominal delays for worst-case conditions. Figure 12-4 shows the resulting delays.

Figure 12-4 OCV for Worst-Case Conditions



In this mode, for a given path, the maximum delay is computed at 100 percent of worst case, and the minimum delay is computed at 80 percent of worst case. PrimeTime checks for a setup violation as follows:

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

In the expression,

clockpath1 = 0.80 + 0.60 = 1.40 (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 PrimeTime consider OCV by increasing the clock uncertainty values with the `set_clock_uncertainty` command.

Specifying the Analysis Mode

The following examples demonstrate how to specify the operating condition analysis modes and how PrimeTime calculates the delays in each mode.

Single Operating Condition Analysis

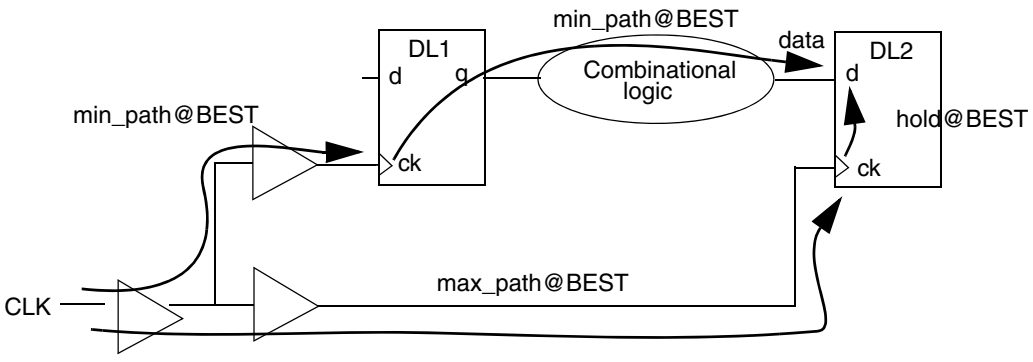
The commands for reporting one operating condition (best case or worst case) and the reported paths are shown in the following sections.

The commands for setting and reporting only the best-case operating conditions are

```
pt_shell> set_operating_conditions BEST
pt_shell> report_timing -delay_type min
```

Figure 12-5 shows the path reported.

Figure 12-5 Timing Path for One Operating Condition—Best Case

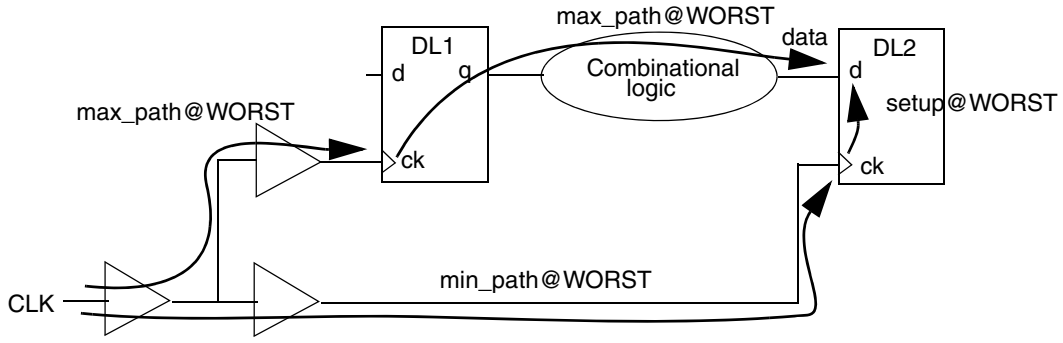


The commands for setting and reporting only the worst-case operating conditions are

```
pt_shell> set_operating_conditions WORST
pt_shell> report_timing -delay_type max
```

Figure 12-6 shows the path reported.

Figure 12-6 Timing Path for One Operating Condition—Worst Case



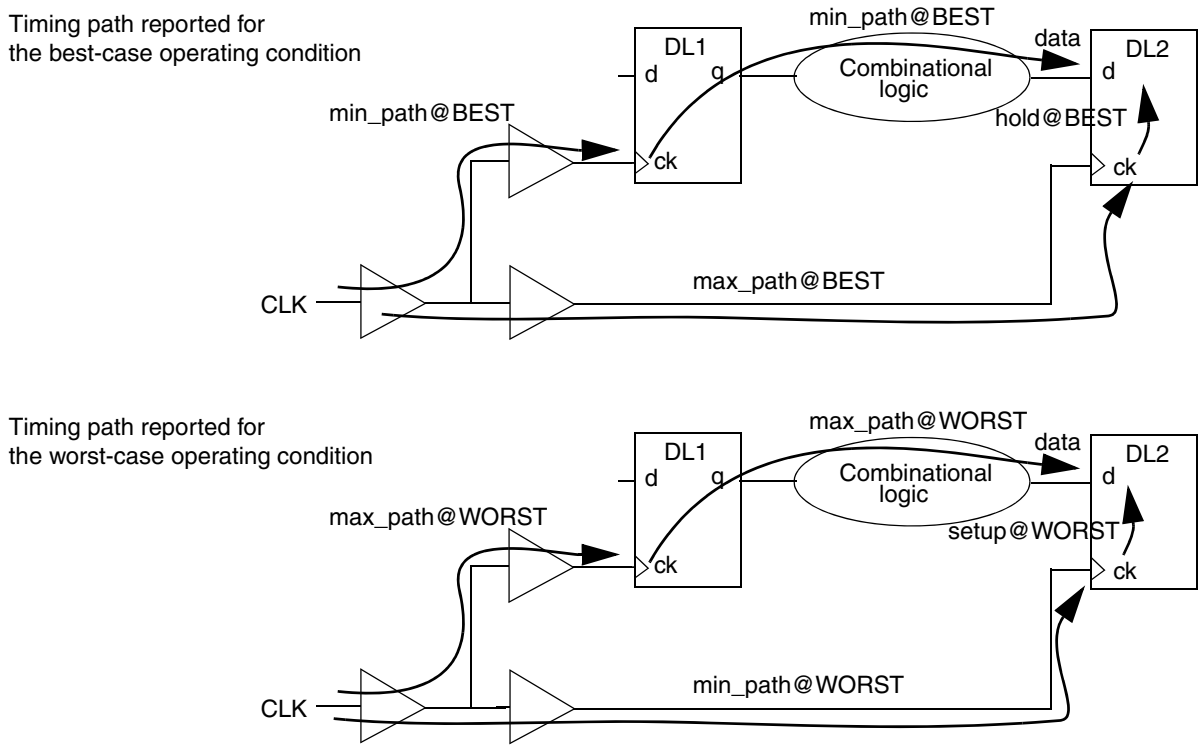
Best-Case/Worst-Case Analysis

The command sequence for setting and reporting simultaneous best-case and worst-case operating conditions is:

```
pt_shell> set_operating_conditions -min BEST -max WORST
pt_shell> report_timing -delay_type min
pt_shell> report_timing -delay_type max
```

Figure 12-7 shows the paths reported for the best case and the worst case when you set simultaneous best-case and worst-case operating conditions.

Figure 12-7 Timing Paths for Best-Case/Worst-Case Operating Conditions



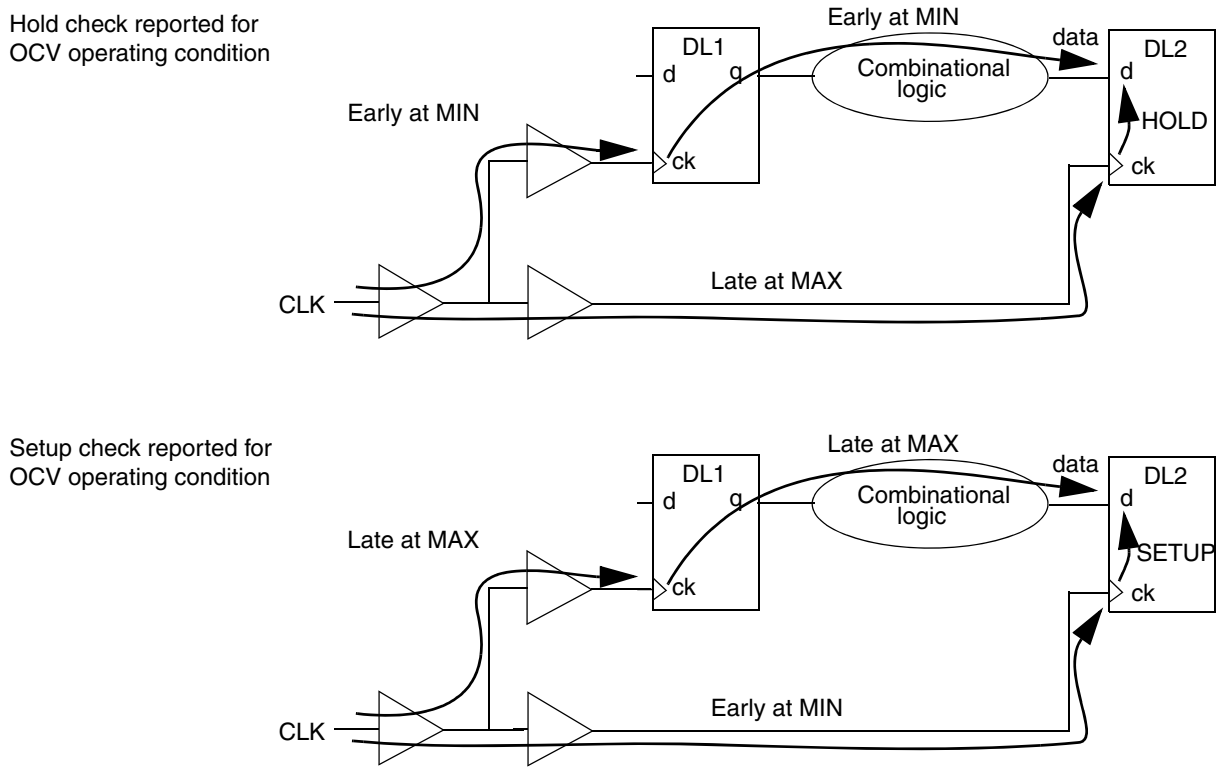
On-Chip Variation Analysis

The command sequence for running OCV analysis is

```
pt_shell> set_operating_conditions -analysis_type  \
          on_chip_variation -min MIN -max MAX

pt_shell> report_timing -delay_type min

pt_shell> report_timing -delay_type max
```

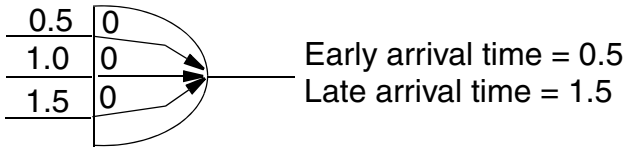
Figure 12-8 shows the paths reported when you run OCV analysis.

Figure 12-8 Timing Path Reported During OCV Analysis



As shown in Figure 12-9, the early arrival time at the AND gate is 0.5 ns and the late arrival time is 1.5 ns, assuming the gate delay is zero.

Figure 12-9 Early and Late Arrival Time



Here are some additional OCV examples.

Example 1

This command sequence performs timing analysis for OCV 20 percent below the worst-case commercial (WCCOM) operating condition.

```
pt_shell> set_operating_conditions -analysis_type \
           on_chip_variation WCCOM

pt_shell> set_timing_derate -early 0.8

pt_shell> report_timing
```

Example 2

This command sequence performs timing analysis for OCV between two predefined operating conditions: WCCOM_scaled and WCCOM.

```
pt_shell> set_operating_conditions -analysis_type \
          on_chip_variation -min WCCOM_scaled -max WCCOM

pt_shell> report_timing
```

Example 3

This command sequence performs timing analysis with OCV. For cell delays, the OCV is between 5 percent above and 10 percent below the SDF back-annotated values. For net delays, the OCV is between 2 percent above and 4 percent below the SDF back-annotated values. For cell timing checks, the OCV is 10 percent above the SDF values for setup checks and 20 percent below the SDF values for hold checks.

```
pt_shell> set_timing_derate -cell_delay -early 0.90
pt_shell> set_timing_derate -cell_delay -late 1.05
pt_shell> set_timing_derate -net -early 0.96
pt_shell> set_timing_derate -net -late 1.02
pt_shell> set_timing_derate -cell_check -early 0.80
pt_shell> set_timing_derate -cell_check -late 1.10
```

Using Two Libraries for Analysis

The `set_min_library` command directs PrimeTime to use two 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, PrimeTime first checks the library cell in the maximum library, and then looks in the minimum library to see if a match exists. If a library cell with the same name, the same pins, and the same timing arcs exists in the minimum library, PrimeTime uses that timing information for minimum analysis. Otherwise, it uses the information in the maximum library. For information about the `set_min_library` command, see the man page.

Setting Derating Factors

You can have PrimeTime adjust minimum delays and maximum delays by specified factors to model the effects of operating conditions. This adjustment of calculated delays is called derating. Derating affects the delay and slack values reported by the `report_timing` command and other reporting commands.

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

```
pt_shell> set_timing_derate -early -cell_delay 0.9

pt_shell> set_timing_derate -late -cell_delay 1.2
```

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

The `-early` or `-late` option specifies whether the shortest-path or longest-path delays are affected by the derating factor. Exactly one of these two options must be used in the command. To set both early and late derating, use two `set_timing_derate` commands. Early path delays include the capture clock path for a setup check, and the launch clock path and data path for a hold check. Late path delays include the launch clock path and data path for a setup check, and the capture clock path for a hold check.

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

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

Delays are adjusted according to the following formula:

delay_new = old_delay + ((derate_factor – 1.0) * abs(old_delay))

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

delay_new = old_delay * derate_factor

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

delay_new = old_delay * (2.0 – 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. If you are using PrimeTime SI, a similar situation can occur for a net due to a change in delay caused by crosstalk.

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

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

If you want only some cells or nets to be affected by derating, you can list them in the command. The list can include library cells, hierarchical cells, leaf-level cells, and nets. In the absence of a list, the whole design is affected.

To set a derate factor on all nets within a hierarchy, use the `-net_delay` option. For example,

```
pt_shell> set_timing_derate -net_delay -early 0.8  \
          [get_cells hier_cell]
```

If you do not specify the `-net_delay` option, only the cells have the derate factor set on them. This feature allows you to specify different derate factors for delta net delays and non-delta net delays. To set a derate factor for non-delta net delays only, use the `-static` option:

```
set_timing_derate -net_delay -static -early/late -data/clock $net $value1
```

To set a derate factor for delta net delays only, use the `-dynamic` option:

```
set_timing_derate -net_delay -dynamic -early/late
   -data/clock $net $value2
```

If both `-static` and `-dynamic` options are omitted, the derate factor is applied to both delta and non-delta net delays.

Different sets of derate factors can be specified for a deterministic PrimeTime analysis and a variational PrimeTime analysis. To set a derate factor for deterministic delays, use the `-scalar` option. To set a derate factor for delays that have been computed using variational information, use the `-variation` option. For more information about using derate factors in a variation analysis, see the *PrimeTime Advanced Timing Analysis User Guide*.

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

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

```
pt_shell> set_timing_derate -late -cell_delay 1.4 [get_cells inv*]
            # derates a collection of cells in the design
pt_shell> set_timing_derate -late -cell_delay 1.3  \
          [get_lib_cells class/ND*]
            # derates cells in a collection of cells in a library class
pt_shell> set_timing_derate -late -cell_delay 1.2
           # derates all cells
pt_shell> set_timing_derate -late -1.1
           # derates all nets and cells
```

In a hierarchical design, the `timing_derate_precedence_compatibility` variable controls the precedence behavior for derating. When this variable is set to its default value of `false`, derating is applied to cells in the following order of precedence, from highest to lowest priority:

1. Leaf-level cell
2. Library cell

3. Hierarchical cell (containing leaf-level cells)
4. Design

Using the default derate precedence behavior is consistent with the derate precedence rules used in Design Compiler and IC Compiler. It is also consistent with other precedence-based cell attributes in PrimeTime, such as the `dont_touch` attribute.

When the the `timing_derate_precedence_compatibility` variable is set to `true`, derating is applied to cells in a hierarchical design in the following order of precedence, from highest to lowest priority:

1. Leaf-level cell
2. Hierarchical cell (containing leaf-level cells)
3. Library cell
4. Design

PrimeTime stores and checks global timing derating factors against block scope. These stored values are checked to make sure that the top-level ranges are completely within the block-level range. For cases where a single value is used at the block level (that is, when the early derate is the same as the late derate), the top-level values must be the same as well. If derate values are not specified at either the block or top levels, the derate value used is 1.0 and the scope check is performed against this value.

Use the `timing_use_constraint_derates_for_pulse_checks` variable to enable or disable timing constraint derates for minimum pulse width and minimum period constraints. This variable is set to `false` by default. When you set this variable to `true`, PrimeTime uses factors defined by the `set_timing_derate` command to derate the minimum pulse width and minimum period constraints in the following way:

- Use the `-late -cell_check -rise` options to apply the minimum pulse width constraints for high pulses
- Use the `-late -cell_check -fall` options to apply the minimum pulse width constraints for low pulses

PrimeTime applies the greater of the above values to the minimum period constraints.

If you use the `set_timing_derate` command while the analysis is set to single operating condition mode, PrimeTime automatically changes to the OCV mode, like using this command:

```
pt_shell> set_operating_conditions -analysis_type on_chip_variation
```

From this mode, you can change to the best-case/worst-case mode, if desired:

```
pt_shell> set_operating_conditions -analysis_type bc_wc
```

If the analysis mode is already set to best-case/worst-case, it stays in that mode when you use `set_timing_derate`.

If you use the `-derate` option with the `report_timing` command, the global, net-specific, or instance-specific derate factor is applied to each cell or net is reported in a column next to the incremental delay for each item.

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

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

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

Clock Reconvergence Pessimism Removal

Clock reconvergence pessimism is an accuracy limitation that occurs when two different clock paths partially share a common physical path segment and the shared segment is assumed to have a minimum delay for one path and a maximum delay for the other path. This condition can occur any time that launch and capture clock paths use different delays, most commonly with OCV analysis (see Table 12-4 on page 12-26). Automated correction of this inaccuracy is called clock reconvergence pessimism removal (CRPR).

PrimeTime performs CRPR, when enabled, at the same time as regular timing analysis. You need to enable CRPR before you do timing analysis. For information about enabling CRPR, see “Using CRPR Commands” on page 12-24. The following examples demonstrate how the pessimism removal works.

On-Chip Variation Example

Consider the following command sequence for running OCV analysis and the corresponding design in Figure 12-10:

```
pt_shell> set_operating_conditions -analysis_type \
          on_chip_variation -min MIN -max MAX

pt_shell> set_timing_derate -net -early 0.80

pt_shell> report_timing -delay_type min

pt_shell> report_timing -delay_type max
```

Figure 12-10 Clock Reconvergence Pessimism Example

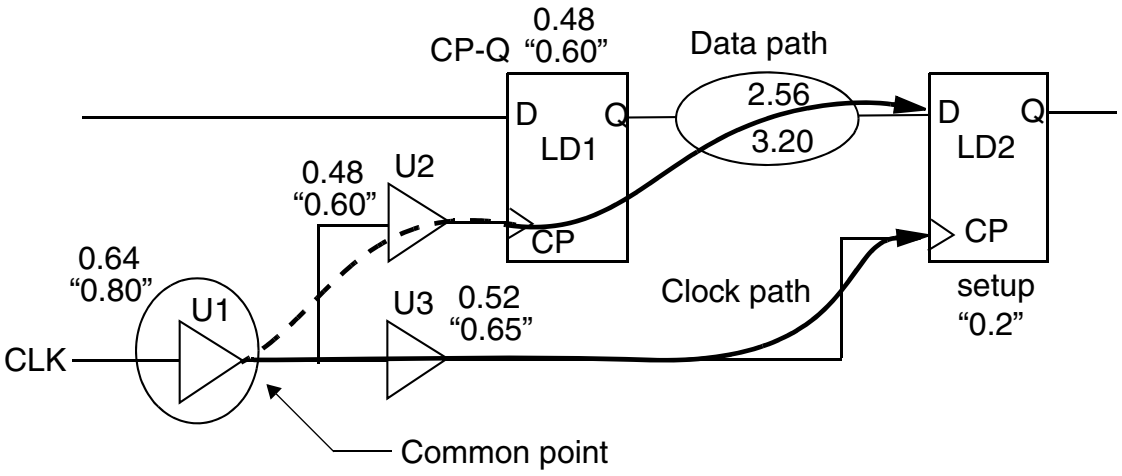


Figure 12-10 shows how the analysis is done. Each delay (considered equal for rising and falling transitions to simplify this example) has a minimum value and a maximum value computed for the minimum and maximum operating conditions.

The setup check at LD2/CP considers the clock path to the source latch (CLK to LD1/CP) at 100 percent worst case, and the clock path to the destination latch (CLK to LD2/CP) at 80 percent worst case.

Although this is a valid approach, the test is pessimistic because clock path1 (CLK to LD1/CP) and clock path2 (CLK to LD2/CP) share the clock tree until the output of U1. The shared segment is called the *common portion*, consisting of just cell U1 in this example. The last cell output in the shared clock segment is called the *common point*, which is the output of U1 in this case.

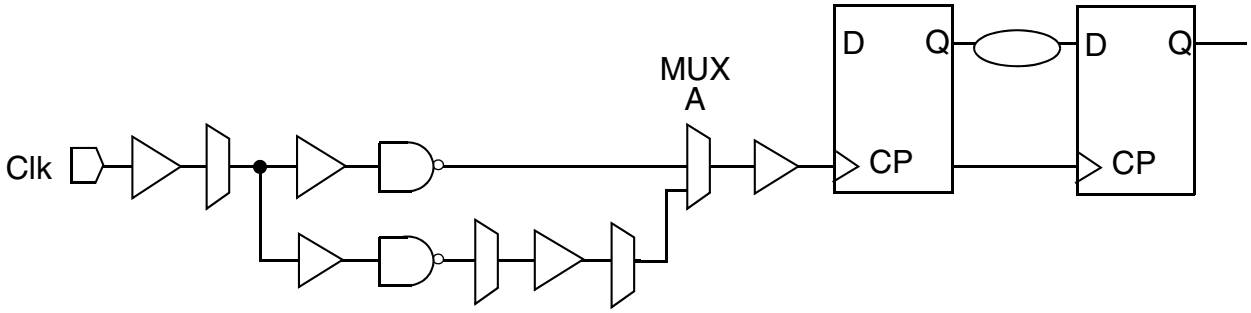
The setup check considers that cell U1 simultaneously has two different delays, 0.64 and 0.80, resulting in a pessimistic analysis in the amount of 0.16. This amount, obtained by subtracting the earliest arrival time from the latest arrival time at the common point, is called the *clock reconvergence pessimism*.

This inaccuracy also occurs in an analogous way for the hold test at the LD2 latch.

Reconvergent Logic Example

Figure 12-11 shows a situation where clock reconvergence can occur, even in the absence of OCV analysis. In this example, there is reconvergent logic in the clock network. The two clock paths that feed into the multiplexer cannot be active at the same time, but an analysis could consider both the shorter and longer paths for one setup or hold check.

Figure 12-11 Reconvergent Logic in a Clock Network



Minimum Pulse Width Checking Example

The `report_constraint` command checks for minimum pulse width violations in clock networks (as specified by the `set_min_pulse_width` command) and at cell inputs (as specified in the technology library). CRPR, when enabled, can increase the accuracy of minimum pulse width checking. For information about enabling CRPR, see “Using CRPR Commands” on page 12-24.

For example, consider the circuit shown in Figure 12-12. The external clock source has early and late source latency set on it. In addition, the two buffers in the path have minimum and maximum rise and fall delay values defined.

The `report_constraint` command checks the pulse width of the clock signal in the clock network and upon reaching the flip-flop. For level-high pulse width checking, PrimeTime considers maximum delay for the rising edge and minimum delay for the falling edge of the clock (and conversely for level-low pulse width checking).

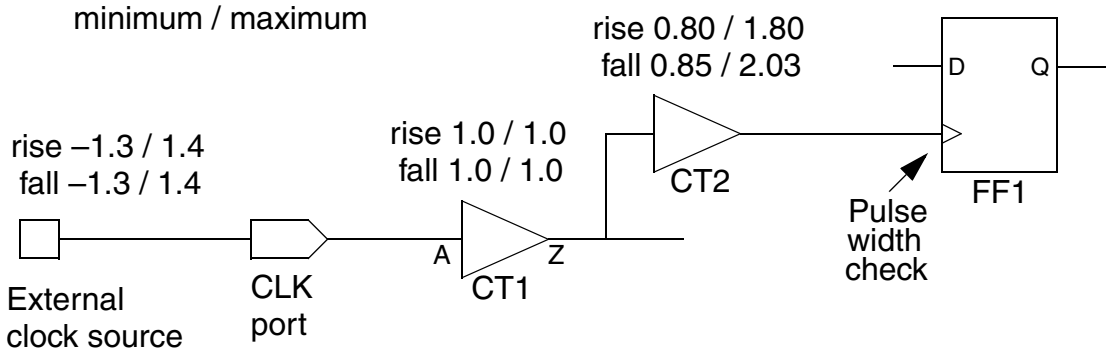
For the example shown in Figure 12-12, in the absence of CRPR, the worst-case pulse width is very small and violates the pulse width constraint of the flip-flop. However, this analysis is pessimistic because it assumes simultaneous worst-case delays for rising and falling edges. In a real circuit, rising-edge and falling-edge delays are at least somewhat correlated. For example, for the delay from the external clock source to the CLK input port, if the rising-edge delay is at the minimum, –1.3, the falling-edge delay is probably equal or close to –1.3, and not at the maximum of +1.4.

To account for correlation between rising-edge and falling-edge delays, enable CRPR. In that case, PrimeTime adds a certain amount of slack back into the minimum pulse width calculation. The amount added is equal to the range of minimum rise delay or the range maximum fall delay for the path, whatever is smaller:

$$\mathrm{crp} = \min((\mathrm{Mr} - \mathrm{mr}),\ (\mathrm{Mf} - \mathrm{mf}))$$

where *crp* = clock reconvergence pessimism, *Mr* = cumulative maximum rise delay, *mr* = cumulative minimum rise delay, *Mf* = cumulative maximum fall delay, and *mf* = cumulative minimum fall delay. For an example of this calculation applied to pulse width checking, see Figure 12-12.

Figure 12-12 Minimum Pulse Width Analysis

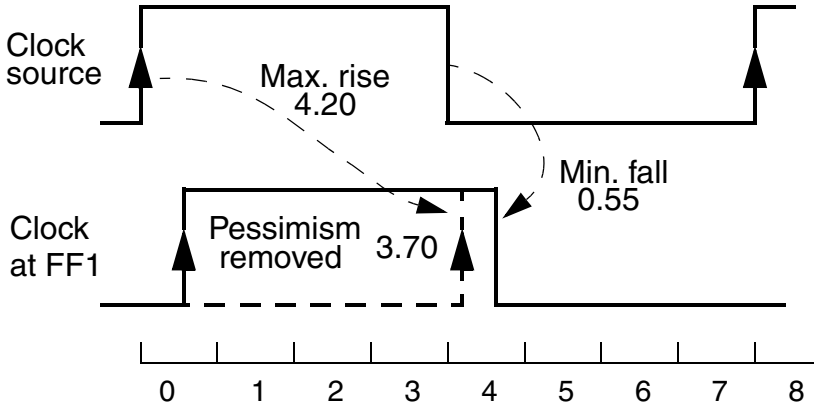


Minimum rise = –1.3 + 1.0 + 0.80 = 0.50
Maximum rise = 1.4 + 1.0 + 1.80 = 4.20
Rise range = 4.20 – 0.50 = 3.70

Minimum fall = –1.3 + 1.0 + 0.85 = 0.55
Maximum fall = 1.4 + 1.0 + 2.03 = 4.43
Fall range = 4.43 – 0.55 = 3.88

Clock reconvergence pessimism = smaller of rise range or fall range = 3.70

Minimum pulse width check: maximum rise = 4.20, minimum fall = 0.55



Using CRPR Commands

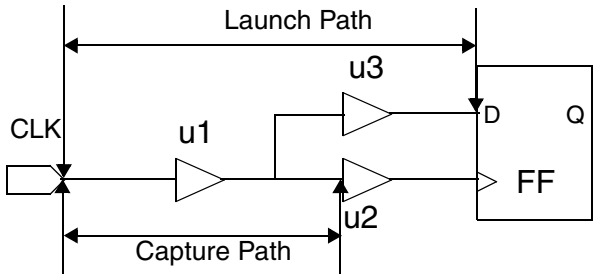
To enable CRPR, set the `timing_remove_clock_reconvergence_pessimism` variable to `true` before you begin timing analysis. By default, the setting is `false` and the algorithm is disabled. Using the algorithm results in a more accurate (less pessimistic) analysis, but causes an increase in runtime and memory usage. Any change in this variable setting causes a complete timing update, like the `update_timing -full` command.

When CRPR is enabled, PrimeTime uses the correction algorithm and reports the results in all the `report_timing`, `report_constraint`, `report_analysis_coverage`, and `report_bottleneck` major types of reports. For the OCV example shown in Figure 12-10, with CRPR enabled, you might produce something similiar to the following report:

```
pt_shell> report_timing -delay_type 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 12-13 (technically, this would not be considered clock reconvergence pessimism because the launching path would be considered to be a data path, not a clock path).

Figure 12-13 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 might cause a severe performance penalty during a timing update.

To specify whether to perform CRPR on clock paths that have opposite-sense transitions in a shared path segment, set the `timing_clock_reconvergence_pessimism` variable. You can set this variable to `normal`, which is the default, or `same_transition`. If this variable is set to the `normal` value, PrimeTime still performs CRPR with opposite-sense transitions in the shared path segment. In that case, if the two transition types produce different correction values, the smaller of the two values is used. If you set the `timing_clock_reconvergence_pessimism` variable to `same_transition`, the CRPR value is removed at the last common point where the launch and capture edge have the same sense. The common point is identified by tracing backwards from the last topological common point on a unique common clock path. In case PrimeTime reaches a common point at which multiple clock paths converge, that point is returned as the common point.

Table 12-4 summarizes the application of either rising or falling clock reconvergence pessimism based on the transition types at the common point in the clock path and `timing_clock_reconvergence_pessimism` variable setting.

Table 12-4 Application of Rise/Fall CRP Value Based on Variable Setting

Transition types at common point in clock paths	**timing_clock_reconvergence _pessimism = normal**	**timing_clock_reconvergence _pessimism = same_transition**
Both rising	crp_rise pessimism removed	crp_rise pessimism removed
Both falling	crp_fall pessimism removed	crp_fall pessimism removed
One rising, one falling	smaller of crp_rise or crp_fall removed	Pessimism at last common point with same sense removed
Transition types at common point in clock paths	`timing_clock_reconvergence _pessimism` = normal	`timing_clock_reconvergence _pessimism` = same_transition

For the minimum pulse width checking example shown in Figure 12-12, you could set up the analysis with a script similar to this:

```
set_operating_conditions -analysis_type on_chip_variation
create_clock -period 8 CLK
set_propagated_clock [all_clocks]
set_clock_latency -source -early -1.3 CLK
set_clock_latency -source -late  1.4  CLK
set_annotated_delay -cell -from CT1/A -to CT1/Z 1
set_annotated_delay -cell -from CT2/A -to CT2/Z -min -rise 0.8
set_annotated_delay -cell -from CT2/A -to CT2/Z -max -rise 1.8
set_annotated_delay -cell -from CT2/A -to CT2/Z -min -fall 0.85
set_annotated_delay -cell -from CT2/A -to CT2/Z -max -fall 2.03
```

With CRPR enabled, a `report_constraint` report on the path would look like this:

```
Point                                       Incr       Path
-----------------------------------------------------------
clock CLK (rise edge)                       0.00       0.00
clock network delay (propagated)            4.20       4.20 r
FF1/CP                                      0.00       4.20 r
open edge arrival time                                 4.20

clock CLK (fall edge)                       4.00       4.00
clock network delay (propagated)            0.55       4.55 f
FF1/CP                                      0.00       4.55 f
clock reconvergence pessimism               3.70       8.25
close edge arrival time                                8.25
-----------------------------------------------------------
```

```
required pulse width (high)                              3.50
actual pulse width                                       4.05
-----------------------------------------------------------
slack                                                    0.55
```

You can set a threshold that allows PrimeTime to ignore small amounts of clock reconvergence pessimism. For computational efficiency, PrimeTime merges multiple points for CRPR calculations when the CRP differences between adjacent points are smaller than the specified threshold. This merging of nodes can reduce CPU and memory usage, without a significant loss of accuracy when an appropriate threshold is set.

The `timing_crpr_threshold_ps` variable specifies the threshold. The units are picoseconds, irrespective of the time units of the technology library. By default, the variable is set to 20, which allows adjacent nodes to be merged when the difference in clock reconvergence pessimism is 20 ps or less.

For a good balance between performance and accuracy, try setting this variable to one-half the stage delay of a typical gate in the clock network. (The stage delay is gate delay plus net delay.) You might want to use a larger value during the design phase for faster analysis, and then a smaller value for sign-off accuracy.

CRPR and Crosstalk Analysis

When you perform crosstalk analysis using PrimeTime SI, a change in delay due to crosstalk along the common segment of a clock path can be pessimistic, but only for a zero-cycle check. A zero-cycle check occurs when the exact same clock edge drives both the launch and capture events for the path. For other types of paths, a change in delay due to crosstalk is not pessimistic because the change cannot be assumed to be identical for the launch and capture clock edges.

Accordingly, the CRPR algorithm removes crosstalk-induced delays in a common portion of the launch and capture clock paths only if the check is a zero-cycle check. In a zero-cycle check, aggressor switching affects both the launch and capture signals in the same way at the same time.

Here are some cases where the CRPR might apply to crosstalk-induced delays:

- Standard hold check
- Hold check on a register with the Q-bar output connected to the D input, as in a divide-by-2 clock circuit
- Hold check with crosstalk feedback due to parasitic capacitance between the Q-bar output and D input of a register

- Hold check on a multicycle path set to zero, such as circuit that uses a single clock edge for launch and capture, with designed-in skew between launch and capture
- Certain setup checks where transparent latches are involved

CRPR With Dynamic Clock Arrivals

A similar scenario to CRPR with crosstalk can occur if dynamic annotations have been set in the clock network. Dynamic annotations include dynamic clock latency and dynamic rail voltage, set by `set_clock_latency` and `set_voltage` commands respectively. These dynamic annotations can lead to dynamic clock arrivals. CRPR handles these dynamic clock arrivals in the same way as it handles delays due to crosstalk. The CRPR value is calculated using dynamic clock arrivals for zero cycle paths only. For all other paths, only the static component of the clock arrival is used thus producing more accurate results.

An example of such dynamic annotations is as follows:

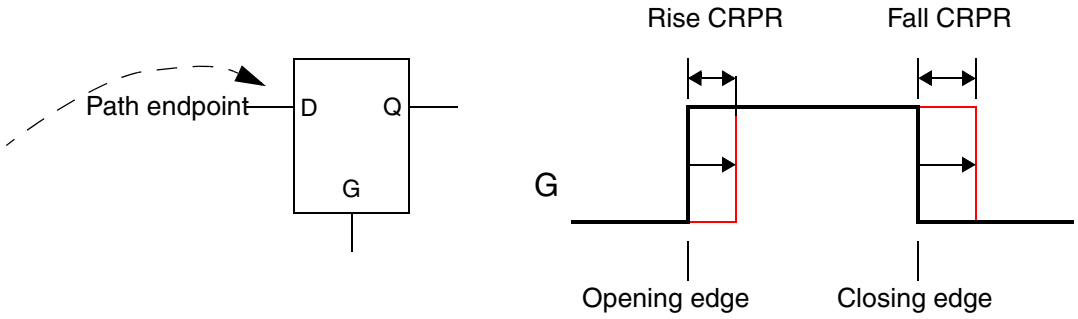
```
pt_shell> set_clock_latency -early -source 2.5  \
          -dynamic -0.5 [get_clocks CLK]

pt_shell> set_clock_latency -source -late 5.5  \
          -dynamic 0.5 [get_clocks CLK]
```

The first of these two commands specifies a total early source latency of 2.5, consisting of static source latency of 3.0 and dynamic source latency of –0.5. The second command specifies a total late source latency of 5.5, consisting of static source latency of 5.0 and dynamic source latency of 0.5. In this case, the static CRP is equal to 2 (5 minus 3). The dynamic CRP is equal to 3 (5.5 minus 2.5).

Transparent Latch Edge Considerations

For a path that ends at a transparent latch, PrimeTime calculates two clock reconvergence pessimism values: one for rising edges and one for falling edges at the common node. The opening and closing clock edges are effectively shifted, each by an amount equal to its corresponding pessimism value, as indicated in Figure 12-14.

Figure 12-14 CRPR for Latches



In practice, the opening-edge pessimism value affects the slack of nonborrowing paths and also reduces the amount of time borrowed for borrowing paths. Meanwhile, the closing-edge pessimism value increases the maximum amount of time borrowing allowed at a latch and reduces the amount of the violation for a path that fails to meet its setup constraint.

To get a report about the calculation of clock reconvergence pessimism values for level-sensitive latches, use the `report_crpr` command.

Reporting CRPR Calculations

The `report_crpr` command reports the calculation of clock reconvergence pessimism (CRP) between two register clock pins or ports. It reports the time values calculated for both static and dynamic conditions, and the choice from among those values actually used for pessimism removal. In the command, you specify the pins of the launch and capture registers, the clock, and type of check (setup or hold). For example,

```
pt_shell> report_crpr -from [get_pins ffa/CP] \
            -to [get_pins ffd/CP] \
            -clock CLK -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. This is because the `report_timing` command, for computational efficiency, “merges” multiple points for CRPR calculations when the CRP differences between adjacent points are too small to be significant. The `timing_crpr_threshold_ps` variable sets the time threshold for merging, which is 20 picoseconds by default. When the `report_crpr` and `report_timing` commands give

different CRP values, the value reported by `report_crpr` command is more accurate because it does not merge adjacent points. For more information about the `report_crpr` command, see the man page.

13

Generating Reports

PrimeTime can generate a wide range of reports that provide information about the design contents and analysis results. The most commonly used reports are described in the following sections:

- Memory and CPU Resources Reports
- Support for Profiling of Tcl Scripts
- Design Information Report
- Wire Load Report
- Disabled Timing Arcs
- Fanin and Fanout Logic
- Bus Report
- Design Constraint Checking
- Analysis Coverage
- Path Timing Report
- Timing Update Efficiency
- Clock Network Timing Report
- Bottleneck Report

- Global Slack Report
- Constraint Report
- Maximum Skew Checks
- No-Change Timing Checks
- Unit Reporting
- Clock-Gating and Recovery/Removal Checks

Memory and CPU Resources Reports

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

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

For processes that are configured to using multiple CPU cores on the same host machine, such as a single-core or threaded multicore analysis, the `report_host_usage` command provides the resource usage for the current PrimeTime session. Here is an example of the report output for a single-core or threaded multicore analysis flow.

```
pt_shell> report_host_usage
****************************************
Report : host_usage
Version: E-2010.12
Date   : Tue Dec 21 11:11:39 2010
****************************************
Usage limits (cores)

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

 Memory usage

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

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

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

Note:
The report output for the distributed multicore flow is slightly different from teh DMSA flow. For an example of the report, see the man page.

```
pt_shell> report_host_usage
****************************************
Report : host_usage
Version: E-2010.12
Date   : Tue Dec 21 11:11:39 2010
****************************************

Options Name       Host Name          32Bit     Num Processes
-------------------------------------------------------------
my_opts1          >>farm<<           N         2

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

Usage limits (cores)

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

Memory usage

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

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

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

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

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

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

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

Elapsed time for distributed processes:
my_opts1        1   ptopt018          1834 seconds
my_opts2        2   ptopt018          1833 seconds
                3   ptopt018          1830 seconds
my_opts3        4   >>farm<<          1750 seconds
                5   >>farm<<          1765 seconds

Maximum memory usage for this session: 4378.52 MB
CPU usage for this session:    1800 seconds
Elapsed time for this session: 1980 seconds
```

Support for Profiling of Tcl Scripts

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

To use the Tcl profiling capability,

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

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

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

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

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

   After you have issued either the `stop_profile` command or exited the pt_shell, PrimeTime writes out several files into the /profile subdirectory of the current working directory. Table 13-1 provides a description of the reports that are automatically generated.

Table 13-1 Tcl Profile and Stopwatch Reports

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

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

Example 13-1 shows an example of the Tcl script profile report that is automatically displayed. The example shows that the `get_timing_paths` command was called once from the gtp_only.tcl script. It took 8 seconds to execute in total, which is shown in the Elapsed column.

Example 13-1 Tcl Script Profile Report

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

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

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

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

```
###EVNT-12645:args= -nworst 2000 -max 500
STOPWATCH[start_get_timing_paths]: CUR_datetime = Wed Apr 13 03:11:54 2011
```

The `start_` text is appended to the command to indicate that the stopwatch entry was saved before the command was started. The current values of the performance and memory metrics and their differences from the previous stopwatch event are also provided. For example,

```
STOPWATCH[start_get_timing_paths]: CUR_elapsed   = 29.28 s
```

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

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

```
###EVNT-12646:args= -nworst 2000 -max 500
STOPWATCH[after_get_timing_paths]: CUR_datetime = Wed Apr 13 03:11:54 2011
STOPWATCH[after_get_timing_paths]: CUR_elapsed     = 36.91 s
STOPWATCH[after_get_timing_paths]: DELTA_elapsed   = 7.63 s
STOPWATCH[after_get_timing_paths]: CUR_cputime     = 21.63 s
STOPWATCH[after_get_timing_paths]: DELTA_cputime   = 7.48 s
STOPWATCH[after_get_timing_paths]: CUR_peakmem     = 276.54 MB
STOPWATCH[after_get_timing_paths]: DELTA_peakmem   = 143.94 MB
```

Example 13-2 shows portions of the Stopwatch report.

Example 13-2 Stopwatch Report

```
###EVNT-1:args=
STOPWATCH[START]: CUR_datetime    = Wed Apr 13 03:11:21 2011
STOPWATCH[START]: CUR_elapsed     = 2.81 s
STOPWATCH[START]: DELTA_elapsed   = 0.00 s
STOPWATCH[START]: CUR_cputime     = 2.54 s
STOPWATCH[START]: DELTA_cputime   = 0.00 s
STOPWATCH[START]: CUR_peakmem     = 26.57 MB
STOPWATCH[START]: DELTA_peakmem   = 0.00 MB
…
###EVNT-12642:args=
STOPWATCH[start_update_timing]: CUR_datetime    = Wed Apr 13 03:11:46 2011
STOPWATCH[start_update_timing]: CUR_elapsed     = 20.40 s
STOPWATCH[start_update_timing]: DELTA_elapsed   = 0.00 s
STOPWATCH[start_update_timing]: CUR_cputime     = 5.54 s
STOPWATCH[start_update_timing]: DELTA_cputime   = 0.00 s
STOPWATCH[start_update_timing]: CUR_peakmem     = 100.60 MB
STOPWATCH[start_update_timing]: DELTA_peakmem   = 0.00 MB
###EVNT-12643:args=
STOPWATCH[after_update_timing]: CUR_datetime    = Wed Apr 13 03:11:46 2011
STOPWATCH[after_update_timing]: CUR_elapsed     = 29.28 s
STOPWATCH[after_update_timing]: DELTA_elapsed   = 8.88 s
STOPWATCH[after_update_timing]: CUR_cputime     = 14.15 s
STOPWATCH[after_update_timing]: DELTA_cputime   = 8.61 s
STOPWATCH[after_update_timing]: CUR_peakmem     = 132.60 MB
STOPWATCH[after_update_timing]: DELTA_peakmem   = 32.00 MB
###EVNT-12644:args= /u/victorb/work_tests/tcl_prof/gtp_only.tcl
STOPWATCH[start_source]: CUR_datetime    = Wed Apr 13 03:11:54 2011
STOPWATCH[start_source]: CUR_elapsed     = 29.28 s
STOPWATCH[start_source]: DELTA_elapsed   = 0.00 s
STOPWATCH[start_source]: CUR_cputime     = 14.15 s
STOPWATCH[start_source]: DELTA_cputime   = 0.00 s
STOPWATCH[start_source]: CUR_peakmem     = 132.60 MB
STOPWATCH[start_source]: DELTA_peakmem   = 0.00 MB
###EVNT-12645:args= -nworst 2000 -max 500
STOPWATCH[start_get_timing_paths]: CUR_datetime    = Wed Apr 13 03:11:54 2011
STOPWATCH[start_get_timing_paths]: CUR_elapsed     = 29.28 s
```

```
STOPWATCH[start_get_timing_paths]: DELTA_elapsed   = 0.00 s
STOPWATCH[start_get_timing_paths]: CUR_cputime     = 14.15 s
STOPWATCH[start_get_timing_paths]: DELTA_cputime   = 0.00 s
STOPWATCH[start_get_timing_paths]: CUR_peakmem     = 132.60 MB
STOPWATCH[start_get_timing_paths]: DELTA_peakmem   = 0.00 MB
###EVNT-12646:args= -nworst 2000 -max 500
STOPWATCH[after_get_timing_paths]: CUR_datetime    = Wed Apr 13 03:11:54 2011
STOPWATCH[after_get_timing_paths]: CUR_elapsed     = 36.91 s
STOPWATCH[after_get_timing_paths]: DELTA_elapsed   = 7.63 s
STOPWATCH[after_get_timing_paths]: CUR_cputime     = 21.63 s
STOPWATCH[after_get_timing_paths]: DELTA_cputime   = 7.48 s
STOPWATCH[after_get_timing_paths]: CUR_peakmem     = 276.54 MB
STOPWATCH[after_get_timing_paths]: DELTA_peakmem   = 143.94 MB
###EVNT-12651:args= /u/victorb/work_tests/tcl_prof/gtp_only.tcl
STOPWATCH[after_source]: CUR_datetime    = Wed Apr 13 03:11:54 2011
STOPWATCH[after_source]: CUR_elapsed     = 36.91 s
STOPWATCH[after_source]: DELTA_elapsed   = 0.00 s
STOPWATCH[after_source]: CUR_cputime     = 21.63 s
STOPWATCH[after_source]: DELTA_cputime   = 0.00 s
STOPWATCH[after_source]: CUR_peakmem     = 276.54 MB
STOPWATCH[after_source]: DELTA_peakmem   = 0.00 MB
…
###EVNT-12664:args=
STOPWATCH[STOP]: CUR_datetime    = Wed Apr 13 03:11:59 2011
STOPWATCH[STOP]: CUR_elapsed     = 41.89 s
STOPWATCH[STOP]: DELTA_elapsed   = 0.00 s
STOPWATCH[STOP]: CUR_cputime     = 21.68 s
STOPWATCH[STOP]: DELTA_cputime   = 0.00 s
STOPWATCH[STOP]: CUR_peakmem     = 294.82 MB
STOPWATCH[STOP]: DELTA_peakmem   = 0.00 MB
```

Design Information Report

Design information reports show information about objects and assertions within a design. PrimeTime can display information about design attributes, clocks, cells, nets, path groups, and minimum pulse. The design information reports are explained in further detail in the following sections.

- Design Attributes
- Clock Information
- Cell Information
- Net Information
- Path Group Report
- Minimum Pulse Width Report

Design Attributes

The `report_design` command provides a summary of the current design attributes, including

- Analysis type
- Operating conditions
- Wire load models
- Design rules
- Derating factors

To display a report about the current design, use the `report_design` command. For example,

```
pt_shell> report_design

****************************************
Report : design
Design : counter
****************************************
 Design Attribute                    Value
---------------------------------------------------
Operating Conditions:
  operating_condition_max_name       WCCOM
  process_max                        1.5
  temperature_max                    70
  voltage_max                        4.75
  tree_type_max                      worst_case

Wire Load :               (use report_wire_load for more
information)
  wire_load_mode                  segmented
  wire_load_model_max             70x70
  wire_load_model_library_max     tech_lib
  wire_load_selection_type_max    user-specified
  wire_load_selection_group_max   --
  wire_load_min_block_size        0

Design Rules:
  max_capacitance                 15
  min_capacitance                 --
  max_fanout                      20
  min_fanout                      --
  max_transition                  3.45
  min_transition                  0.1
  max_area                        --

Timing Ranges:
```

```
  fastest_factor                --
  slowest_factor                --
```

Clock Information

The `report_clock` command displays information about all clocks and generated clocks in the current design. PrimeTime can also report on specific clocks.

If you issue a `report_clock` command directly after a `create_generated_clock` command, the newly generated clock period and waveform do not appear in the `report_clock` output until you issue an `update_timing` command.

To display information about the clocks in a design, use the `report_clock` command. For example,

```
pt_shell> report_clock

****************************************
Report : clock
Design : TOP
****************************************
Attributes:
    p - propagated_clock
    G - generated clock

Clock              Period      Waveform    Attrs    Sources
------------------------------------------------------------
CLK                20.00      {5 15}               {CLK}
CLKDIV             40.00      {5 25}       G       {INFF1/Q}

Generated   Master             Generated           Waveform
Clock       Source             Source              Modification
------------------------------------------------------------
CLKDIV      C1                 INFF1/Q             divide_by(2)
```

Cell Information

The `report_cell` command provides information about all cells in the current instance. PrimeTime can also display information about a specific cell or set of cells.

This report includes the

- Corresponding cell reference (the library cell name)
- Library name containing the cell
- Area of the cell

Default Report

To create a default report, use the `report_cell` command. For example,

```
pt_shell> report_cell

****************************************
Report : cell
Design : TOP
****************************************

Cell                Reference       Library     Area  Attributes
------------------------------------------------------------------
U1                  AN2             tech_lib               2.00
ff1                 FF              tech_lib               7.00
ff2                 FF              tech_lib               7.00
ff3                 FF              tech_lib               7.00
ff4                 FF              tech_lib               7.00
u4                  BOTTOM          TOP                   16.00
------------------------------------------------------------------
```

Connectivity Report

To report connectivity information for each cell in a design, use the following syntax:

```
pt_shell> report_cell -connections

Connections for cell 'ff1':
    Reference:          FF
    Library:            tech_lib

    Input Pins          Net
    ----------------    ------------
    D                   I1
    CP                  CP

    Output Pins         Net
    ----------------    ------------
    Q                   <no_net>
    QN                   net11
```

Verbose Report

The `-verbose` option and the `-connections` option display additional attributes and show pins on the other side of the net connected to cell pins. For example,

```
pt_shell> report_cell -connections -verbose

Connections for cell 'ff1':
    Reference:          FF
    Library:            tech_lib
    Area:               7
```

```
Input Pin      Net         Net Driver Pins       Driver Pin Type
-----------------------------------------------------------------
D              I1          I1                    Input Port
CP             CP          CP                    Input Port

Output Pins    Net         Net Load Pins         Load Pin Type
-----------------------------------------------------------------
Q              <no_net>    QN net11 U1/B         Input Pin (OR2)
```

Net Information

The `report_net` command shows information about nets in the current instance, including

- Fanout and fanin
- Capacitance and resistance
- Pins
- Net connections

PrimeTime can also display a specific net in the design.

By default, the `report_net` command displays brief information about all nets in the current instance. To create a default report, use the `report_net` command. For example,

```
pt_shell> report_net

****************************************
Report : net
Design : TOP/u4 (BOTTOM)
****************************************
Attributes:
    c - annotated capacitance
    r - annotated resistance

Net              Fanout     Fanin      Cap     Resistance     Pins   `   Attributes
-------------------------------------------------------------------------------
CP                4          1         4.00     0.00          5
I1                2          1         2.00     0.00          3
O1                2          1         2.00     0.00          3
net1              1          1         1.00     0.00          2
net12             1          1         1.00     0.00          2
-------------------------------------------------------------------
Total 5 nets     10          5        10.00     0.00         15
Maximum           4          1         4.00     0.00          5
Average           2.00       1.00      2.00     0.00          3.00
```

Connectivity Report

The `-connections` option creates a connectivity report for the nets in the design. For example,

```
pt_shell> report_net -connections

Connections for net 'u4/net11':

    Driver Pins             Type
    ------------            ----------------
    u4/ff1/QN               Output Pin (FF)

    Load Pins               Type
    ------------            ----------------
    u4/U1/B                 Input Pin (OR2)
```

Verbose Report

The `-verbose` option produces a verbose report that displays additional attributes. For example,

```
pt_shell> report_net -connections -verbose

Connections for net 'u4/net11':
    pin capacitance:    1
    wire capacitance:   0
    total capacitance:  1
    wire resistance:    0
    number of drivers:  1
    number of loads:    1
    number of pins:     2

    Driver Pins          Type              Pin Cap
    ------------     ----------------      --------
    u4/ff1/QN         Output Pin (FF)         0

    Load Pins            Type              Pin Cap
    ------------     ---------------       --------
    u4/U1/B           Input Pin (OR2)          1
```

Path Group Report

Paths are organized into groups based on the clocks used to capture data at the path endpoints. The `report_path_group` command displays information about the path groups in the current design.

To produce a path group report, use the `report_path_group` command. For example,

```
pt_shell> report_path_group

****************************************
Report : path_group
Design : Test
Version: A-2007.12-DEV
Date:    Sun Jul 29 12:03:16 2007
****************************************
Path_Group              Weight       From       Through          To
--------------------------------------------------------------------
**default**             1.00          -           -              -
C1                      1.00          *           *              C1
C2                      1.00          *           *              C2
CLK                     1.00          *           *              CLK
```

Minimum Pulse Width Report

The `report_min_pulse_width` command reports information about the minimum pulse width for specified pins or ports. The report includes the pin name, required pulse width, actual pulse width, whether or not the constraint is violated, amount by which the constraint value is violated, and the design object that is the worst violator. You can use this command to generate detailed reports on minimum pulse width by using the `-path_type` and `-input_pins` options. Using the `-path_type`, `full_clock` and `full_clock_expanded` options, you can report the detailed clock path for any intermediate pin in the clock network where you have specified a minimum pulse width check. You can also report derate factors in the minimum pulse width report by using the `-derate` option with the `-path_type`, `full_clock` and `full_clock_expanded` options. When you use the `-derate` option, it has the same look and feel as the `-derate` option of the `report_timing` command.

The default path type is `summary`, and it prints the object name, minimum pulse width constraint, the actual pulse width, and slack. To follow is an example of the report that is generated using the `report_min_pulse_width` command:

```
set_timing_derate -late 2.0  -cell_check
report_min_pulse_width -all_vio -path_type full_clock -derate

****************************************
Report : min pulse width
         -all_violators
         -path_type full_clock
         -derate
Design : TOP_9000155610
Version: E-2010.12
Date   : Tues Dec 21 13:53:11 2010
****************************************
```

```
  Pin: reg3g/CP
  Related clock: clk1
  Check: sequential_clock_pulse_width

  Point                                          Derate    Incr       Path
-----------------------------------------------------------------------------
  clock clk1 (rise edge)                                   0.00       0.00
  clock source latency                                     0.00       0.00
  d (in)                                                   0.00       0.00 r
  abufc/Z (BUFFHVTD1)                             1.00     0.05       0.05 r
  del3/Z (DELHVT4)                                1.00     4.30       4.35 r
  abufcd/Z (BUFFHVTD1)                            1.00     0.07       4.42 r
  reg3g/CP (DFCNHVTD1)                            1.00     0.00       4.42 r
  open edge clock latency                                             4.42

  clock clk1 (fall edge)                                   2.97       2.97
  clock source latency                                     0.00       2.97
  d (in)                                                   0.00       2.97 f
  abufc/Z (BUFFHVTD1)                             1.00     0.06       3.03 f
  del3/Z (DELHVT4)                                1.00     3.90       6.93 f
  abufcd/Z (BUFFHVTD1)                            1.00     0.10       7.04 f
  reg3g/CP (DFCNHVTD1)                            1.00     0.00       7.04 f
  close edge clock latency                                            7.04
 -----------------------------------------------------------------------------
  required pulse width (high)                     2.00               5.40
  actual pulse width                                                 2.61
-----------------------------------------------------------------------------
  slack (VIOLATED)                                                  -2.79
```

By default, all objects with minimum pulse width constraints that are annotated are reported. To display only the violating minimum pulse width checks use the `-all_violators` option.

Wire Load Report

Wire load reports provide information about the wire loads you have defined in your design, including

- Cell name and design name
- Wire load model for the cell
- Library that contains the wire load model
- Selection type

Wire Load Information for the Current Design

The `report_wire_load` command provides the wire load information for the current design.

Wire Load Information About Ports

The `report_port` command with its `-wire_load` option provides wire load information about ports. To see wire load information about the ports in a design, enter

```
pt_shell> report_port -wire_load

****************************************
Report : port
         -wire_load
Design : AN2_DESIGN
****************************************
                External Fanout
                Number   Wire Load
Port            Points   Model
----------------------------------------
A               0        --
B               0        --
Z               0        70x70: tech_lib
```

Disabled Timing Arcs

The `report_disable_timing` command reports the disabled timing arcs in the design. You can disable timing arcs in several ways: by using the `set_disable_timing` command, propagating constants, case analysis, loop breaking, conditional arcs (arcs that have a `when` statement defined in the library), and default arcs (when the `timing_disable_cond_default_arcs` variable is set to `true`).

To produce a report about disabled timing arcs, use the `report_disable_timing` command. For example,

```
pt_shell> report_disable_timing

****************************************
Report : disable_timing
Design : TOP
Version: ...
Date   : ...
****************************************
```

```
Flags :     c  case-analysis
            C  Conditional arc
            d  default conditional arc
            f  false net-arc
            l  loop breaking
            L  db inherited loop breaking
            m  mode
            p  propagated constant
            u  user-defined

Cell or Port    From  To    Sense                     Flag     Reason
-------------------------------------------------------------------------
DISP1[7]                                              p     DISP1[7] = 0
U1/U1/reg[0]    E     D     hold_clk_rise             c     D = 0
U1/U1/reg[0]    E     D     setup_clk_rise            c     D = 0
U1/U1/reg[0]    E     E     clock_pulse_width_high    c     D = 0
```

Fanin and Fanout Logic

The `report_transitive_fanin` command reports the logic that fans in to specified sink objects (pins, ports, or nets). To show the transitive fanin of a pin in a design, use the `report_transitive_fanin` command. For example,

```
pt_shell> report_transitive_fanin -to FF1/D

****************************************
Report : transitive_fanin
Design : top
****************************************
Fanin network of sink 'FF1/D':

Driver Pin        Load Pin         Type          Sense
-----------------------------------------------------------
gate1/Y           FF1/D            net arc       same

Load Pin          Driver Pin       Type          Sense
-----------------------------------------------------------
gate1/A           gate1/Y          AN2           same
gate1/B           gate1/Y          AN2           same

Driver Pin        Load Pin         Type          Sense
-----------------------------------------------------------
IN1               gate1/A        (net arc)       same
IN2               gate1/B        (net arc)       same
```

The `report_transitive_fanout` command reports the logic that fans out from specified pins, ports, or nets. A pin is considered to be in the transitive fanout of a source if a timing path exists through combinational logic from the source to that pin. The fanout report stops at the inputs to registers (sequential cells).

If the report is for the current instance, the report stops at the boundaries of the current instance but shows the hierarchical boundary pins on the instance. To show the transitive fanout of all the clock sources in a design, use the `report_transitive_fanout` command. For example,

```
pt_shell> report_transitive_fanout -clock_tree


****************************************
Report : transitive_fanout
        -clock_tree
Design : top
****************************************

Fanin network of source 'CLK':

Driver Pin        Load Pin         Type          Sense
-------------------------------------------------------
CLK               FF3/CLK         (net arc)     same
CLK               FF2/CLK         (net arc)     same
CLK               FF1/CLK         (net arc)     same
```

Bus Report

The `report_bus` command reports information about buses (pins or ports) in the current instance or current design. To generate a report about buses in the design, use the `report_bus` command. For example,

```
pt_shell> report_bus

****************************************
Report : bus
Design : bus_design
****************************************

Bussed Port     Dir      From     To    Width
-----------------------------------------------
acnt            out      15       0     16
baddreg         out      15       0     16
bwordreg        out      15       0     16
caddreg         out      15       0     16
comreg          out       7       0      8
cs_badd         in        1       0      2
cs_bword        in        1       0      2
cs_cadd         in        1       0      2
cs_channel      in        3       0      4
cs_cword        in        1       0      2
cwordreg        out      15       0     16
dack            out       3       0      4
```

```
din             in        7       0       8
dreq            in        3       0       4
drqreg          out       3       0       4
modreg          out       7       0       8
mskreg          out       3       0       4
srqreq          out       3       0       4
stareg          out       3       0       4
tmpreg          out       7       0       8
wcnt            out      15       0      16
```

Design Constraint Checking

Paths that are incorrectly constrained might not appear in the violation reports, possibly allowing you to overlook paths with violations. For this reason, you should always use the `check_timing` command to check any new design or any design with new constraints (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 constraints, such as `create_clock`, `set_input_delay`, and `set_output_delay`. The following example shows a typical `check_timing` report:

```
pt_shell> check_timing

Information: Checking 'no_clock'.
Warning: There are 4 register clock pins with no clock.
Information: Checking 'no_input_delay'.
Information: Checking 'unconstrained_endpoints'.
Information: Checking 'generic'.
Information: Checking 'latch_fanout'.
Warning: There are 2 level-sensitive latches which fanout to
themselves.
Information: Checking 'loops'.
Warning: There are 6 timing loops in the design.
Information: Checking 'generated_clocks'.
```

By default, the `check_timing` command performs several types of constraint checking and issues a summary report like the one shown in the preceding example. To get a detailed report, use the `-verbose` option. To add to or subtract from the default list of checks, use the `-include` or `-exclude` option of the `check_timing` command or set the `timing_check_defaults` variable to specify the list of checks for subsequent `check_timing` commands. For more information, see the man page for the specific command or variable.

Table 13-2 describes some of the conditions that the `check_timing` command can detect. For a complete list, see the `check_timing` man page.

Table 13-2 Issues Detected With the check_timing Command

Potential problem	Report results
Unconstrained endpoints	Shows unconstrained register data pins or primary outputs that are not marked as false paths.
No clock fanin	Shows register clock pins that cannot be reached by a clock signal. Data pins on these same registers often also appear as unconstrained endpoints.
Multiple clock fanin	Shows register clock pins that are reached by more than one clock signal (see Figure 13-1).
Master-slave clock separation	Shows overlapping clocks on master-slave registers. Designs that contain master-slave latches with separate master and slave clock inputs often have a strict timing separation value between those clocks. This value ensures that the master and slave clocks do not overlap. If an overlap occurs, the master and slave latches become transparent. You can specify a minimum separation between the clocks.
Latch fanout	Shows two types of checks: - Latch self-loop (see Figure 13-2). This situation can be a problem because it is very delay sensitive. Data can flow around the self-loop many times during the active level. - Latch fanout to another latch of the same clock (see Figure 13-3). This situation is not usually a problem. A message informs you that a phi1 to phi1 path exists in the design. If you have a pure two-phase design, this helps isolate paths that do not conform.
Combinational feedback loops	Shows the combinational feedback loops, which are considered to be asynchronous. You can break loops by disabling timing arcs or PrimeTime automatically breaks them.
No input delay	Shows primary input ports with no input delay or unclocked input delay. For more information, see the `check_timing` man page.
Ignored attributes	Shows ignored timing attributes, such as the `max_time_borrow` attribute or timing exceptions.

Table 13-2 Issues Detected With the check_timing Command (Continued)

Potential problem	Report results
Generated clock network	Informs you if these criteria are not met: - The generated clock is in the fanout of a master clock and not in the fanout of any other clock source. - The master source pin of the generated clock has a valid clock-generated clock source. - There are no loops in the generated-clock network. For example, if clock D2 is generated from D1, D1 cannot be defined as a generated clock with its master source set to D2.
PrimeTime SI only: no driving cell, ideal clocks, partial input delays clock expandable	If crosstalk analysis is enabled with PrimeTime SI, the `check_timing` command performs additional checks that are specific to crosstalk analysis. For more information, see the `check_timing` man page or the *PrimeTime SI User Guide*.

Figure 13-1 shows register clock pins that are reached by multiple clock signals.

Figure 13-1 Multiple Clock Fanin

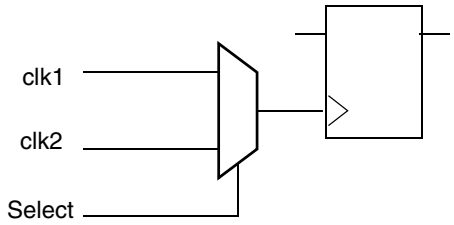


Figure 13-2 shows an example of a self-looping latch.

Figure 13-2 Latch Self-Loop

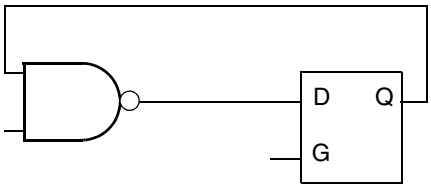
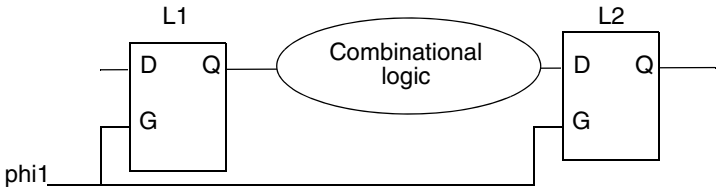


Figure 13-3 shows latch fanout to another latch of the same clock.

Figure 13-3 Latch Fanout to Another Latch of the Same Clock



Note:

If timing paths are unconstrained, the `check_timing` command only reports the unconstrained endpoints, not the unconstrained startpoints. Similarly, for paths constrained only by `set_max_delay`, `set_min_delay`, or both rather than `set_input_delay` and `set_output_delay`, the `check_timing` command only reports any unconstrained endpoints, not unconstrained startpoints.

The `check_timing` command allows interface logic models to be used at the chip-level. The `is_interface_logic_model` hierarchical instance attribute indicates whether an instance is an interface logic model or not. PrimeTime automatically sets this attribute to `true` when you create an interface logic model with the `create_ilm` command. The attribute allows the `check_timing` command to identify interface logic models and suppress false reporting of unconnected clock pins. For more information about interface logic models, see the *PrimeTime Modeling User Guide*.

Analysis Coverage

You can generate a report showing information about timing checks in the current design or current instance by using the `report_analysis_coverage` command. Generating information about timing checks is most critical for new designs.

Perform these checks right after you resolve errors found while using the `check_timing` command. Perform additional checks whenever significant changes are made to the design or the timing assertions. Analysis coverage checks are critical for sign-off. Follow this basic flow:

1. Run `link_design` and resolve link errors.
2. Run `check_timing` and resolve timing check errors.
3. Run `report_analysis_coverage` and resolve untested issues.
4. Perform the rest of the analysis.

The `report_analysis_coverage` command summarizes these checks:

- Setup
- Hold
- No-change
- Minimum period
- Recovery
- Removal
- Minimum pulse width
- Clock separation (master-slave)
- Clock gating setup
- Clock gating hold
- Output setup
- Output hold
- Maximum skew

The default report is a summary of checks organized by type. For each type of check, such as setup, the report shows the number and percentage of checks that meet constraints, violate constraints, and are untested. If there are no checks of a certain type, the report does not show that check type.

Use the report generated by the `report_analysis_coverage` command ensure that the analysis was complete. Static timing is considered exhaustive—all paths are checked. However, if the assertions are incomplete or if paths are disabled (using false paths, disabled arcs, case analysis, and so forth), some timing checks are not tested. You can use this report with the `check_timing` command to make sure the design and assertions are valid.

In some cases, you might want to use case analysis to analyze the design in different configurations. You can use the `report_analysis_coverage` command to determine which checks are untested in each configuration. If a check is untested in all configurations, you might want to do more analysis.

Use the `-status_details` option to show more information about the individual timing checks. The report shows all checks with the corresponding status. Untested checks have information about why they are untested, if the reason can be determined. See the `report_analysis_coverage` man page for the reasons. You can use the `-exclude_untested` option to sort the list of reasons.

To display the summary report, use the `report_analysis_coverage` command. For example,

```
pt_shell> report_analysis_coverage

****************************************
Report : analysis_coverage
Design : counter
Version: 2000.11
Date   : Wed Nov 15 13:40:04 2000
****************************************

Type of Check       Total      Met           Violated        Untested
------------------------------------------------------------------------------
setup               5          0 (  0%)      3 ( 60%)        2 ( 40%)
hold                5          3 ( 60%)      0 (  0%)        2 ( 40%)
------------------------------------------------------------------------------
All Checks          10         3 ( 30%)      3 ( 30%)        4 ( 40%)
```

To display the detailed report of untested setup checks, use the `report_analysis_coverage` command with its options. For example,

```
pt_shell> report_analysis_coverage -status_details {untested} -check_type {setup}

****************************************
Report : analysis_coverage
        -status_details {untested}
        -sort_by slack
        -check_type {setup }
Design : counter
Version: 2000.11
Date   : Thur November 30 13:41:09 2000
****************************************

Type of Check         Total        Met           Violated     Untested
------------------------------------------------------------------------
setup                   5          0 ( 0%)       3 ( 60%)     2 ( 40%)
------------------------------------------------------------------------
All Checks              5          0 (  0%)      3 ( 60%)     2 ( 40%)

Constrained                    Related   Check
Pin                            Pin       Type       Slack       Reason
------------------------------------------------------------------------
ffd/CR                         CP        setup      untested    no_clock
ffd/D                          CP        setup      untested    no_clock
```

Path Timing Report

You can use path timing reports to focus on particular timing violations and to determine the cause of a violation. By default, the `report_timing` command reports the path with the worst slack for each path group based on maximum delay.

A path timing report provides detailed timing information about any number of requested paths. The level of detail that you want to see in the output can vary. For example, you can view this information:

- Gate levels in the logic path
- Incremental delay value of each gate level
- Sum of the total path delays
- Amount of slack in the path
- Source and destination clock name, latency, and uncertainty
- On-chip variation (OCV) with clock reconvergence pessimism removed

Path Groups

PrimeTime organizes paths in the design into groups. This path grouping affects the generation of timing analysis reports. For example, the `report_timing` command, by default, reports the single path with the worst slack from each path group.

In Design Compiler, path grouping also affects design optimization. Each path group can be assigned a weight (also called cost function). The higher the weight, the more effort Design Compiler uses to optimize the paths in that group. You can assign weights to path groups in PrimeTime, but this weight information is not used in PrimeTime.

PrimeTime implicitly creates a path group each time you use the `create_clock` command to create a new clock. The name of the path group is the same as the clock name. PrimeTime assigns a path to that path group if the endpoint of the path is a flip-flop clocked by that clock.

PrimeTime also creates the following path groups implicitly:

- **clock_gating_default**: paths that end on combinational elements used for clock gating
- **async_defaul**: paths that end on asynchronous preset/clear inputs of flip-flops

- **default**: constrained paths that do not fall into any of the other implicit categories (for example, a path that ends on an output port)
- none: unconstrained paths

In addition to these implicit path groups, you can create your own user-defined path groups by using the `group_path` command. This command also lets you assign any particular path to a specific path group. To obtain information about the current set of path groups, use the `report_path_group` command.

To remove a path group, use the `remove_path_group` command. Paths in that group are implicitly assigned to the default path group.

To place all paths to ports with names matching OUT_1* into their own group called out1bus, use the following syntax:

```
pt_shell> group_path -name out1bus -to [get_ports OUT_1*]
```

Using the report_timing Command

The `report_timing` command provides options to control reporting of the following:

- Number of paths
- Types of paths
- Amount of detail
- Startpoints, endpoints, and intermediate points along the path

The `-significant_digits` option of the `report_timing` command lets you specify the number of digits after the decimal point displayed for time values in the generated report. This option only controls the number of digits displayed, not the precision used internally for analysis. For analysis, PrimeTime uses the full precision of the platform's fixed-precision, floating-point arithmetic capability.

Example 1

The `report_timing` command generates a report similar to the following example:

```
pt_shell> report_timing

****************************************
Report : timing
Design : FP_SHR
****************************************

Operating Conditions:
Wire Loading Model Mode: top
```

```
  Startpoint: a (input port)
  Endpoint: c_d (output port)
  Path Group: default
  Path Type: max

  Point                              Incr       Path

-----------------------------------------------------
  input external delay              10.00      10.00 r
  a (in)                             0.00      10.00 r
  m1/Z (MUX21H)                      1.00      11.00 r
  u1/S (FA1)                         1.00      12.00 r
  c_d/Z (AN2)                        1.00      13.00 r
  c_d (out)                          0.00      13.00 r
  data arrival time                            13.00

  max_delay                         15.00      15.00
  output external delay            -10.00       5.00
  data required time                            5.00
-----------------------------------------------------
  data required time                            5.00
  data arrival time                           -13.00
-----------------------------------------------------
  slack (VIOLATED)                 -8.00
```

Example 2

To show detailed information about the four worst paths in each path group, enter

```
pt_shell> report_timing -input_pins -max_paths 4
```

Example 3

You can use multiple `-through` options in a single command to specify paths that traverse multiple points in the design. For example,

```
pt_shell> report_timing -from A1 -through B1 -through C1 \
             -to D1
```

This means any path that starts at A1, passes through B1 and C1 in that order, and ends at D1.

```
pt_shell> report_timing -from A1 -through {B1 B2} \
            -through {C1 C2} -to D1
```

This means any path that starts at A1, passes through either B1 or B2, then passes through either C1 or C2, and ends at D1.

```
pt_shell> report_timing -from A1 -through {B1 C1} -to D1
```

This means any path that starts at A1, passes through B1 and C1 in that order, and ends at D1. You cannot use more than one `-through` option in this syntax mode.

Example 4

To show the transition time and capacitance, use the following syntax:

```
pt_shell> report_timing -transition_time -capacitance
```

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
        -transition_time
        -capacitance
Design : counter
Version: 2000.11
Date   : Wed Nov 15 09:51:19 2000
****************************************

  Startpoint: ffa (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: ffd (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Point                          Cap      Trans     Incr       Path
  -----------------------------------------------------------------------------
  clock CLK (rise edge)                                0.00       0.00
  clock network delay (ideal)                          0.00       0.00
  ffa/CLK (DTC10)                                      0.00       0.00       0.00 r
  ffa/Q (DTC10)                          3.85          0.57       1.70       1.70 f
  U7/Y (IV110)                           6.59          1.32       0.84       2.55 r
  U12/Y (NA310)                          8.87          2.47       2.04       4.58 f
  U17/Y (NA211)                          4.87          1.01       1.35       5.94 f
  U23/Y (IV120)                          2.59          0.51       0.37       6.30 r
  U15/Y (BF003)                          2.61          0.88       0.82       7.12 f
  U16/Y (BF003)                          2.61          1.46       0.99       8.11 r
  U10/Y (AN220)                          2.63          0.46       1.04       9.15 r
  ffd/D (DTN10)                                        0.46       0.00       9.15 r
  data arrival time                                                          9.15
  clock CLK (rise edge)                                           10.00      10.00
  clock network delay (ideal)                                      0.00      10.00
  ffd/CLK (DTN10)                                                            10.00 r
  library setup time                                              -1.33       8.67
  data required time                                                          8.67
  -----------------------------------------------------------------------------

  data required time                                                          8.67
  data arrival time                                                          -9.15
  -----------------------------------------------------------------------------
  slack (VIOLATED)                                                           -0.48
```

Using the report_timing -exclude Option

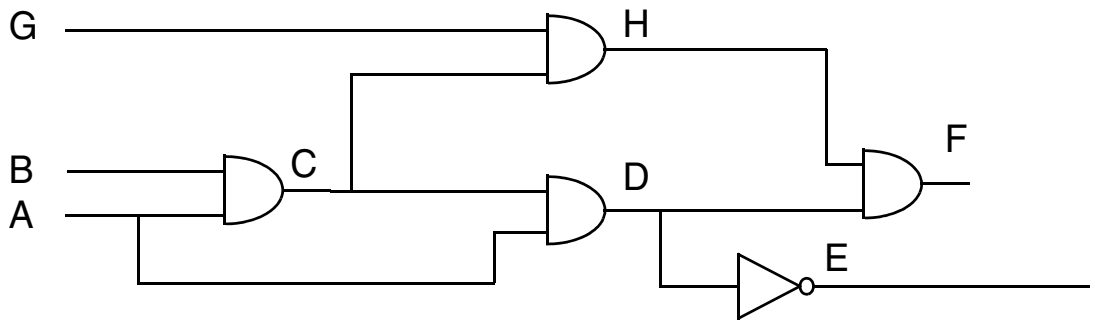
The `report_timing` command has the `-exclude`, `-rise_exclude`, and `-fall_exclude` options that allow you to skip paths that contain any excluded netlist objects. All of these options take a list of objects (pins, ports, nets, and cells). These options behave as follows:

- For pins and ports, they exclude all paths from, to, or through any excluded pin or port. This option does not apply to clock pins.
- For cells and nets, they exclude all paths from, to, or through all pins of any excluded cell or net.
- The `-exclude` option has higher precedence over the `-from`, `-to`, and `-through` options.
- The `-exclude` option does not work when you also specify the `-true` option.
- The `-rise_exclude` and `-fall_exclude` options prune paths with rise and fall transition, respectively.
- When used with the `-trace_latch_borrow` option, the `-exclude` option does not apply to the borrowing path.
- When used with the `-full_clock_expanded` or `-full_clock` options, the `-exclude` option does not apply to the clock path.
- Multiple levels of the `-exclude` options are not supported. That is, you cannot exclude a particular path by specifying a series of the `-exclude` options.

Example of Using the -exclude Option

For example, assume that you had a design with the paths shown in Figure 13-4, where C is both an endpoint and a startpoint.

Figure 13-4 Using the -exclude Option



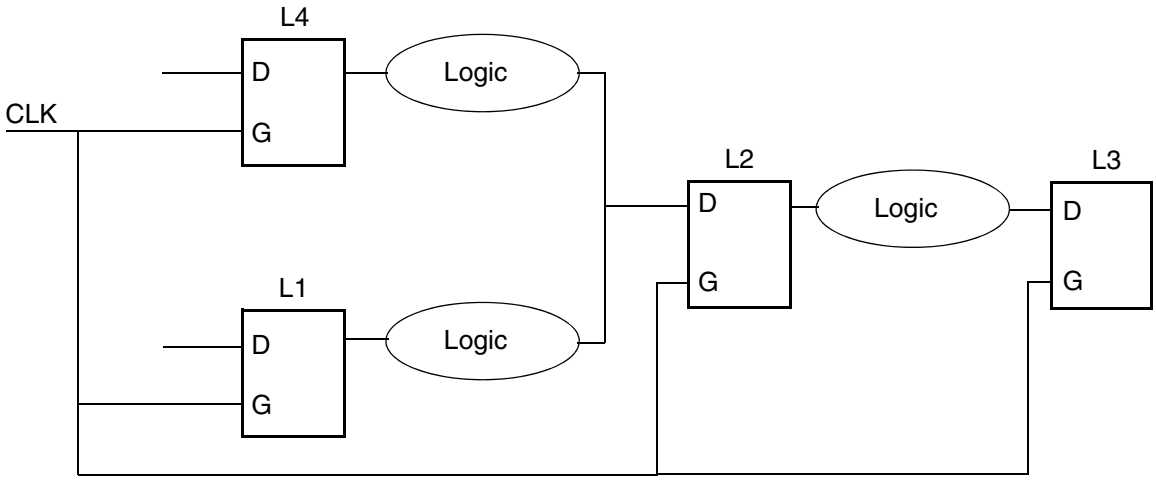
If you used the following command, only three paths would be reported: A-->D-->E, A-->D-->F, and G-->H-->F.

```
pt_shell> report_timing -nworst 2 -max_paths 3 -exclude C
```

Example of Using the -exclude Option With the -trace_latch_borrow Option

The `-exclude` option only applies to data paths, not borrowing paths. For example, assume you have a circuit like the one shown in Figure 13-5. In this circuit, LD2 borrows from both LD1 and LD4, but the path from LD1 is the worst path.

Figure 13-5 Using -exclude with -trace_latch_borrow



If you used the following commands on this circuit, both reports would show the worst path starting from LD2 and still borrowing from LD1, because the `-exclude` option does not apply to borrowing paths.

```
pt_shell> report_timing -exclude LD1 -to LD3/D
pt_shell> report_timing -exclude LD1 -to LD3/D -trace_latch_borrow
```

Timing Update Efficiency

PrimeTime has a built-in algorithm to efficiently update the timing of a design after its initial timing to accommodate a change in conditions. The algorithm reuses a portion of the computation done for the initial timing. For example, if you load and analyze a design using the `update_timing` or `report_timing` command, and then change the capacitance on a port with the `set_load` command, the subsequent `report_timing` command reuses results from the previous timing update, and only analyzes the timing changes resulting from the capacitance change. As a result, the second timing update takes much less time than the first one.

When any timing changes occur, timing is automatically updated by commands that need the information, such as the `report_timing` and `report_constraint` commands. You do not need to do a manual update.

Note:

The `update_timing -full` command causes a complete timing update and overrides the fast timing updates previously described. Avoid invoking `update_timing -full` unless it is necessary.

To update timing for the design manually, use the `update_timing` command. This command causes PrimeTime to recompute all the timing information.

The preceding algorithm trades off computational effort for memory usage. The `timing_update_effort` variable controls this tradeoff. The values for this variable are low, medium (the default), and high as explained in the following:

low

The computational effort is low (timing update is fastest), but the memory usage might increase significantly if the number of changes is very large. This memory usage increase is not bounded.

medium (the default)

The computational effort is low (timing update is fast), but the additional memory used over that used for the initial timing is bounded. This bound is 10 percent. If the 10 percent bound is not sufficient to accommodate all your changes, a message appears and PrimeTime automatically changes to a less efficient algorithm that is more conservative in memory usage. In this case, you might need to change the value to low. However, even with this small 10 percent bound, PrimeTime can accommodate a relatively large number of changes and it is unlikely that you need to change this default value.

high

The computational effort is high (timing update is slower), but there is no increase in the memory used over that used for the initial timing of the design.

When one or more of the following commands causes a change to the timing of the design, PrimeTime updates only the timing for the changed data, resulting in a faster timing update.

- `set_load`
- `remove_capacitance`
- `set_resistance`
- `remove_resistance`
- `set_port_fanout_number`
- `remove_port_fanout_number`
- `set_input_transition`
- `set_driving_cell`

- remove_driving_cell
- set_drive

Status Messages During Timing Update

PrimeTime can display messages during the timing update phase of an analysis, providing you with the current status of the update. For example, PrimeTime can issue messages when it is propagating constants, calculating delays, and calculating slack for paths. These messages can be very useful in debugging large designs and monitoring the progress of the analysis.

The timing_update_status_level variable controls the detail and number of progress messages that the timing update process issues. The allowed values are none, low, medium, or high. The default value is none, indicating no intermediate status messages issued.

You can report the progress of the update timing explicitly by using the update_timing command and set the timing_update_status_level variable to low, medium, or high, or for an implicit update use the report_timing command. The number of messages varies based on the value of the variable, for instance

- When the value is none (the default), no messages are shown.
- When the value is low, only the beginning and the end of the update are reported.
- When the value is medium, all messages for low are shown, as well as the beginning and end of the additional intermediate timing update steps: constant propagation, delay calculation, and slack computation.
- When the value is high, all messages for medium are shown. In addition, for the delay calculation step, large designs show the completion percentage. Also, for the slack computation step, the groups for which the computation is made are shown.

If you want a detailed status of a timing update, set the timing_update_status_level to high before you begin your analysis. To show the messages during update timing, use the following syntax:

```
pt_shell> set timing_update_status_level high
high

pt_shell> update_timing

Information: Updating design - Started                   (UITE-214)
Information: Updating design - Propagating Constants     (UITE-214)
Information: Updating design - Calculating delays        (UITE-214)
Information: Updating design - Calculating delays 10%... (UITE-214)
Information: Updating design - Calculating delays 20%... (UITE-214)
```

```
Information: Updating design - Calculating delays 30%... (UITE-214)
Information: Updating design - Calculating delays 40%... (UITE-214)
Information: Updating design - Calculating delays 50%... (UITE-214)
Information: Updating design - Calculating delays 60%... (UITE-214)
Information: Updating design - Calculating delays 70%... (UITE-214)
Information: Updating design - Calculating delays 80%... (UITE-214)
Information: Updating design - Calculating delays 90%... (UITE-214)
Information: Updating design - Calculating delays 100%...(UITE-214)
Information:Updating design  - Calculating slacks        (UITE-214)
Information: Updating design - Calculating slacks (max type)
                               for group 'CLK3'          (UITE-214)
Information: Updating design - Calculating slacks (min type)
                               for group 'CLK3'          (UITE-214)
Information: Updating design - Calculating slacks (max type)
                               for group 'CLK4'          (UITE-214)
Information: Updating design - Calculating slacks (min type)
                               for group 'CLK4'          (UITE-214)
Information: Updating design - Completed                 (UITE-214)
```

Some messages issued during the `read_parasitics`, `report_annotated_parasitics -check`, `read_sdf`, and an implicit or explicit `update_timing` commands have a default limit each time the command is invoked. If the number of messages exceeds this limit, a note stating that no further messages will be issued is added to the log file. In addition, a summary of the messages affected and the number suppressed is printed at the end of the PrimeTime session. If there are a large number of messages, the size of the log file is reduced. The `sh_message_limit` variable controls the default message limit and the `sh_limited_messages` variable controls the messages affected by this feature. Only messages issued during the commands mentioned are affected by the `sh_limited_messages` variable. For more information about these commands and variables, see the specific man page.

Clock Network Timing Report

The timing characteristics of the clock network are important in any high-performance design. To obtain information about the clock networks in a design, use the `report_clock_timing` command. This command reports the clock latency, transition time, and skew characteristics at specified clock pins of sequential elements in the network.

In the `report_clock_timing` command, you specify the type of report you want (latency, transition time, single-clock skew, interclock skew, or summary), the scope of the design to analyze, and any desired filtering or ordering options for the report. PrimeTime gathers the requested information and reports it in the specified order.

Latency and Transition Time Reporting

The information reported by the `report_clock_timing` command is based on the clock latency and transition times calculated by PrimeTime. This information is maintained for all clock pins of sequential devices in the design (flip-flops and latches).

The clock latency at a particular pin depends on the following analysis conditions:

- Constraint type: setup or hold
- Timing path role: launch or capture
- Transition type: rise or fall

To restrict the scope of the `report_clock_timing` command, you can optionally specify the pins to analyze and the conditions at those pins. For example,

```
pt_shell> report_clock_timing -type latency -to U1/CP \
            -hold -capture -rise
```

In this example, PrimeTime reports the latency at pin U1/CP for a hold check, for data capture at the flip-flop, for a rising edge at the clock pin. If you specify neither `-rise` nor `-fall`, PrimeTime considers the device type, in conjunction with its launch or capture role, to determine the appropriate transition to report.

Using the `-to` option, you can selectively restrict the scope of the design checked for the report. For example,

```
pt_shell> report_clock_timing -type latency \
              -to {U1/CP U7/CP} -hold -capture -rise
```

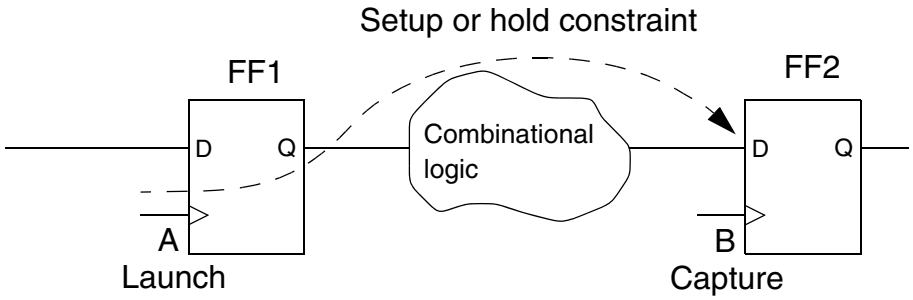
When a pin named in the “to” list is not a clock pin of a sequential element, PrimeTime replaces that pin with the set of clock pins in the transitive fanout of the named pin. (Here, “clock pin” means the clock pin of a flip-flop or the gate pin of a latch.)

Skew Reporting

To get a single-clock skew report, use `report_clock_timing -type skew`. This 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 `report_clock_timing -type interclock_skew`, as described in the section “Interclock Skew Reporting” on page 13-37.

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 13-6. PrimeTime calculates the latency at points A and B, and then subtracts latency at A from the latency at B to obtain the clock skew.

Figure 13-6 Clock Skew Calculation



Skew = (latency at B) – (latency at A) [– (clock reconvergence pessimism)]

To determine the latency at the two pins for skew calculation, PrimeTime considers the following conditions at each pin:

- Type of sequential device involved: rising or falling edge-sensitive; or high or low level-sensitive
- Type of constraint: setup or hold (which determines the path type, transition type, and library)
- Role of the sequential device in the constraint relationship: launch or capture

If CRPR is enabled, PrimeTime takes it into account for the skew calculation. For information about this subject, see “Clock Reconvergence Pessimism Removal” on page 12-19.

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 positive-edge-triggered capture flip-flop, you would use a command similar to the following:

```
pt_shell> report_clock_timing -from latch/G -to ff/CP \
            -type skew -setup
```

PrimeTime reports the skew between a pair of sequential devices only if they can communicate by one or more data paths in the specified “from” and “to” direction. The existence of such a data path is sufficient for reporting. PrimeTime does not check to see whether the path has been declared false.

To find the worst skew between any pair of sequential devices, you can specify `-from` without `-to` or `-to` without `-from`. For the unspecified pins, PrimeTime uses the set of all sequential device clock pins in the design that communicate with the specified pins in the specified direction. To restrict the clocks considered in the report, use the `-clock` *`clock_list`* option. To include clock uncertainty in the skew calculation, use the `-include_uncertainty_in_skew` option.

Like the `report_timing` command, the `report_clock_timing` command calculates skew based on the opening edge at the “to” device, even for a level-sensitive latch that allows time borrowing.

Interclock Skew Reporting

To 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 information, such as the following:

- Skew between pin A (clocked by CLK1) and pin B (clocked by CLK2)
- Worst local skew between all sequential devices clocked by CLK1 and all sequential devices clocked by CLK2
- Ten worst skews to all devices that communicate with pin A, irrespective of their domain

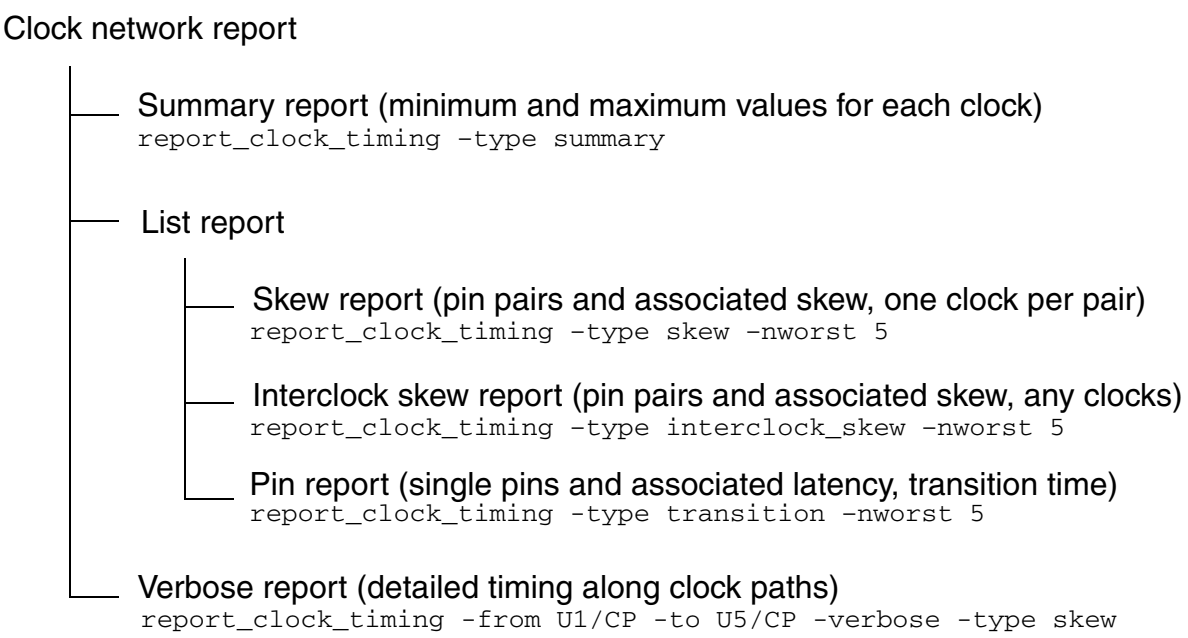
You can restrict the scope of the report by using the `-from_clock` *`from_clock_list`* option or the `-to_clock` *`to_clock_list`* option, or both options. Because of the potentially large number of clock pins that PrimeTime must analyze, it is a good idea to be as specific as possible when you specify an interclock skew report.

To display the names of the “from” clock and the “to” clock in the interclock skew report, use the `-show_clocks` option in the `report_clock_timing` command.

Clock Timing Reporting Options

The `report_clock_timing` command offers several different types of reports, as summarized in Figure 13-7. 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 13-7 Clock Network Timing Report Types



For a full description of the `report_clock_timing` command and its options, see the man page.

Summary Report

A summary report shows a list of the minimum and maximum latency, transition time, and clock skew values found in the requested scope of the clock network. For example,

```
pt_shell> report_clock_timing -type summary \
            -clock [get_clocks CLK1]
...
  Clock: CLK1
-------------------------------------------------------------
  Maximum setup launch latency:
      f2_2/CP                              6.11      rp-+
  Minimum setup capture latency:
      f1_2/CP                              1.00      rpi-+
  Minimum hold launch latency:
      f1_2/CP                              1.00      rpi-+
```

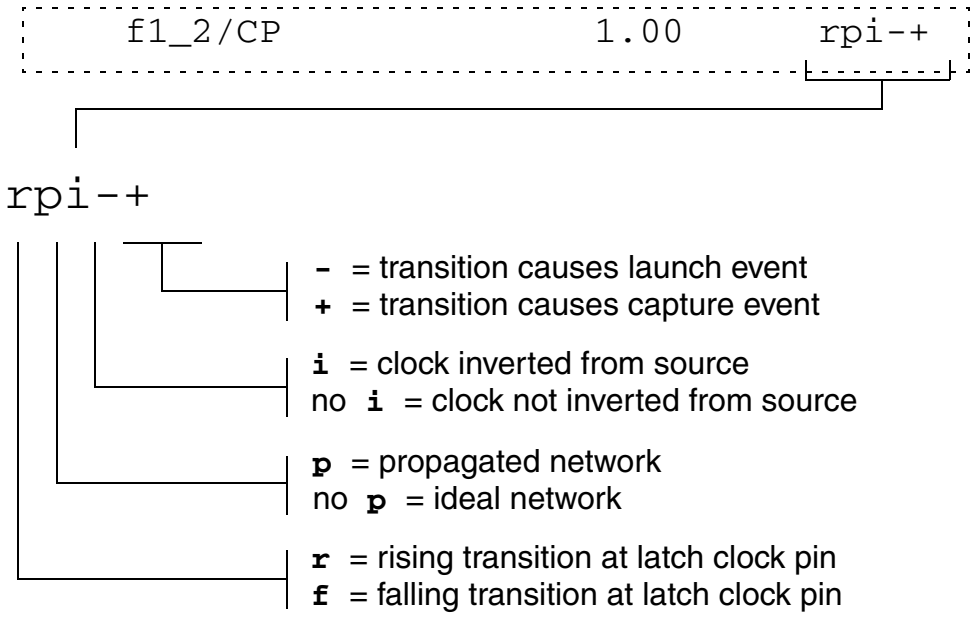
```
  Maximum hold capture latency:
      f2_2/CP                                6.11      rp-+
  Maximum active transition:
      l3_2/G                                 0.13      rpi-
  Minimum active transition:
      l2_3/G                                 0.00      rp-
  Maximum setup skew:
      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 13-8.

Figure 13-8 Latency, Transition Time, and Skew Condition Codes



To restrict the scope of the report, you can use the `-from` and `-to` options of the command. For example,

```
pt_shell> report_clock_timing -type summary -clock CLK1 \
            -from {A B C} -to {D E}
```

This command restricts the scope of the report to CLK1 latency and transition times at pins A through E, and restricts the scope of the skew report to CLK1 skew between the pin groups {A B C} and {D E}.

List Report

A list report provides latency, transition time, and skew information in greater detail than a summary report. You can have PrimeTime gather, filter, sort, and display a collection of clock pins according to a specified attribute of interest.

There are two types of lists you can generate: skew reports and pin reports. A skew report lists pin pairs and shows the skew value for each pair. A pin report lists individual pins and shows the transition time and latency values for each pin. The items are listed in order of skew, transition time, or latency, as specified by the options in the `report_clock_timing` command.

An example of a command to generate a skew report is as follows:

```
pt_shell> report_clock_timing -clock CLK1 -type skew -setup \
            -nworst 3
```

The report shows the three largest skew values listed in order of decreasing skew, together with the latency at each pin and the clock reconvergence pessimism used in the skew calculation:

```
  Clock: CLK1

  Clock Pin           Latency    CRP      Skew
-------------------------------------------------------
  f2_2/CP             6.11                         rp-+
  f2_1/CP             2.01      -0.10     4.00     rp-+

  l2_2/G              4.11                         rp-
  f1_2/CP             1.00      -0.10     3.01     rpi-+

  f2_2/CP             6.11                         rp-+
  l3_3/G              3.01      -0.10     3.00     rp-
-------------------------------------------------------
```

The rightmost column shows the conditions under which the reported values occurred at each corresponding pin, using the codes shown in Figure 13-8.

An example of a command to generate an interclock skew report is as follows:

```
pt_shell> report_clock_timing -type interclock_skew \
            -nworst 12 -setup -include_uncertainty_in_skew
```

The report shows the 12 largest skew values between clock pins 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 at this point indicate a possibly long runtime to generate the rest of the report.) An example of an interclock skew report is:

```
****************************************
Report : clock timing
         -type interclock_skew
 ...
****************************************
Number of startpoint pins:     1023
Number of endpoint pins:       2496
Number of startpoint clocks:   4
Number of endpoint clocks:     6

Clock Pin               Latency    Uncert    Skew
-------------------------------------------------------------
f2_2/CP                 6.11                           rp-+
f2_1/CP                 2.01       0.11      4.21      fp-+
...
...
```

To show the names of the two clocks for each entry in the interclock skew report, use the `-show_clocks` option of the command.

An example of a command to generate a pin report is as follows:

```
pt_shell> report_clock_timing -clock CLK1 -type transition \
            -nworst 5
```

The report shows the five largest transition times listed in decreasing order, together with the source, network, and total latency of the corresponding pins:

```
  Clock: CLK1
                  --- Latency ---
  Clock Pin   Source   Network   Total     Trans
-------------------------------------------------------------
  l3_2/G      0.10     4.00      4.10      0.13      rpi-
  f3_1/CP     0.11     3.00      3.11      0.12      rp-+
  l2_1/G      0.10     4.00      4.10      0.10      rpi-
  f3_3/CP     0.10     3.00      3.10      0.08      rpi-+
  l3_3/G      0.11     3.00      3.11      0.06      rp-
-------------------------------------------------------------
```

The rightmost column shows the conditions under which the reported values occurred, using the codes shown in Figure 13-8.

To get a list of the largest latency values rather than transition times, use `-type latency` instead of `-type transition`.

Verbose Path-Based Report

The most detailed type of clock network timing report is the verbose, path-based report. This type of report shows the calculation of skew, latency, and transition times along the clock path. For example,

```
pt_shell> report_clock_timing \
            -from f3_3/CP -to f2_2/CP -verbose \
            -hold -type skew -include_uncertainty_in_skew
```

This command produces a report showing the transition time, incremental delay, and total latency along the clock paths to the specified pins; and the clock skew between the two pins:

```
Clock: CLK1

  Startpoint: f3_3 (rising edge-triggered flip-flop clocked
    by CLK1')
  Endpoint: f2_2 (rising edge-triggered flip-flop clocked
    by CLK1)

  Point                      Trans       Incr       Path
  -----------------------------------------------------------
  clock source latency                   0.00       0.00
  clk3 (in)                  0.00        0.00       0.00 f
  bf3_3_1/Z (B1I)            0.01        1.00 H     1.00 f
  bf3_3_2/Z (B1I)            0.00        1.00 H     2.00 f
  if3_3_1/Z (IVA)            0.04        1.00 H     3.00 r
  f3_3/CP (FD1)              0.04        0.00       3.00 r
  startpoint clock latency                          3.00

  clock source latency                   0.11       0.11
  clk2 (in)                  0.00        0.00       0.11 r
  az_1/Z (B1I)               0.09        1.00 H     1.11 r
  az_2/Z (B1I)               0.13        1.00 H     2.11 r
  bf2_2_1/Z (B1I)            0.02        1.00 H     3.11 r
  if2_2_1/Z (IVA)            0.44        1.00 H     4.11 f
  bf2_2_2/Z (B1I)            0.01        1.00 H     5.11 f
  if2_2_2/Z (IVA)            0.13        1.00 H     6.11 r
  f2_2/CP (FD1)              0.13        0.00       6.11 r
  endpoint clock latency                            6.11
  -----------------------------------------------------------
```

```
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.

Limitations of Clock Network Reporting

The following limitations apply to the `report_clock_timing` command:

- When you ask for the skew between two clock pins, it calculates and reports the skew even if the path between the two clocks has been declared false.
- If the “from” and “to” lists in the command contain a large number of pins, execution of the command can take a long time due to the large number of paths that must be checked, especially for interclock skew reports.

Obtaining the Clock Network Using the get_clock_network_objects Command

The `get_clock_network_objects` command allows you to thoroughly view the clock network in your design. This command operates as a debugging tool similar to the such commands as the `get_cell` or `get_nets` command.

When you run the `get_clock_network_objects` command, PrimeTime returns a collection of clock network objects (including latches, flip flops, and black box IPs driven by the clock network). Use the `-type` option to instruct PrimeTime to return clock objects you specified. These clock object can belong or relate to one or several clocks domains (specified using the *`clock_list`* option) or all clock domains. The *`object_type`* can be one of the following: cell, register, net, pin, clock_gating_output.

If you want PrimeTime to include clock gating networks in the collection of clock networks, use the `-include_clock_gating_network` option.

The following example returns a collection of clock network pins of all clock domains in the design, not including pins of the clock gating networks.

```
pt_shell> get_clock_network_objects -type pin
```

Bottleneck Report

A bottleneck is a common point in the design that contributes to multiple violations. Bottleneck analysis helps you identify the worst bottlenecks, determine the likelihood of being able to significantly improve a bottleneck, and make a change to the netlist (guide synthesis, try a gate sizing change, and so forth).

PrimeTime bottleneck analysis associates a bottleneck with leaf cells in a design. You can generate a bottleneck report to know which gates or nets to change to improve the overall timing quality. Analyzing bottlenecks enables you to fix many paths with a single change (whether the change is logical or physical). In some cases, fixing many subcritical violating paths is preferable to fixing a few worst paths because a few violators might be resolved by place and route.

You can get bottleneck information in PrimeTime in three ways:

- Graphically, by generating a bottleneck histogram (Reports > Histograms > Timing Bottlenecks in the PrimeTime GUI).
- In text form, by using the `report_bottleneck` command.

  Note:
  If you intend to use the `report_bottleneck` command as part of your flow, it is recommended that you set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update. For more information, see “report_bottleneck” on page 2-34 or the man page.
- By writing a Tcl program that retrieves, organizes, and reports bottleneck attributes in the design. An example Tcl procedure is provided in *install_dir*/auxx/pt/examples/tcl/bottleneck_utils.tcl and “Example 2” later in this section.

You can specify one of three cost types for the report: `path_count (the default)`, `path_cost`, and `endpoint_cost`. Use the path type options as follows:

- To consider all violating paths equally, use `path_count`.
- To give greater weight to paths with larger violations, use `path_cost`.
- To consider only the slack of violating endpoints in the fanout of each cell (for faster analysis), use `endpoint_cost`.

The `report_bottleneck` command lists the worst 20 (or other specified number) leaf cells in the design, where worst means that they have the highest bottleneck cost.

To verify the result of bottleneck analysis, use the `report_timing` or `get_timing_paths` command.

This example reports the 20 worst bottleneck cells in the current design based on the number of paths through the cells with slack less than 0.0.

```
pt_shell> report_bottleneck

****************************************
Report : bottleneck
        -max_cells 20
        -nworst_paths 100
Design : Example
Version: 2000.11
****************************************

Bottleneck Cost = Number of violating paths through cell

                                                  Bottleneck
Cell                                Reference     Cost
-----------------------------------------------------------
ipxr/ir1/i2/i0/u3                   DFF1A         39656.00
ipxr/ir1/U14                        INV2          39654.00
ipxr/ir1/U16                        INV8          39654.00
ipxr/ir1/i2/i0/u4                   DFF1A         31806.00
ipxr/ir1/U15                        BUF8B         31804.00
ipxr/U1712                          MUX2I         23999.00
ipxr/U558                           INV4          23999.00
ipxr/U1370                          NAN4          22485.00
ipxr/x01                            INV2          22485.00
dmac/BufEn                          BUF3          22485.00
dmac/U574                           INV           22485.00
ipxr/U1335                          NAN6CH        21844.00
ipxr/U564                           MUX2I         20074.00
ipxr/U1694                          MUX2A         19936.00
ipxr/U1559                          BUF2C         14785.00
ipxr/U1484                          MUX2I         14252.00
ipxr/U1486                          MUX2I         12468.00
SccMOD/U929                         OR3           12135.00
ipxr/U1493                          MUX2I         11248.00
dmac/BiuSMOD/U879                   NAN2          10949.00
```

The following Tcl procedure validates the bottleneck cost for a cell to compare with `report_bottleneck` or the GUI bottleneck data. You can use the `print_report` command to list the paths.

```
proc verify_bottleneck_cell {cell slack_limit nworst
print_report} {
  set through_pins [get_pins -of_object [get_cell $cell] -filter
"direction!=in"]
  set path_count 0
  set path_cost 0.0
  set fanout_endpoint_cost 0.0
```

```
  foreach_in_collection through_pin $through_pins {
    set paths [get_timing_paths -through $through_pin \
       -slack_lesser_than $slack_limit -nworst $nworst]
    foreach_in_collection path $paths {
      incr path_count
      set this_path_cost [expr 0.0 - [get_attribute $path slack]]
      set path_cost [expr $this_path_cost + $path_cost]
    }
    if {$print_report} {
      report_timing -through $through_pin \
        -slack_lesser_than $slack_limit -nworst $nworst
    }
    set paths [get_timing_paths -through $through_pin \
       -slack_lesser_than $slack_limit -max_paths 10000]
    foreach_in_collection path $paths {
      set this_path_cost [expr 0.0 - [get_attribute $path slack]]
      set fanout_endpoint_cost [expr $fanout_endpoint_cost +
         $this_path_cost]
    }
  }

  echo "------------------------------------------------------------"
  echo "path_count = $path_count"
  echo "path_cost = $path_cost"
  echo "fanout_endpoint_cost = $fanout_endpoint_cost"
}
```

Global Slack Report

You can use the `report_global_slack` command to display the slack for a specified pin or port. By default all pins, except those of hierarchical cells and ports of the design, are reported.

Note:

You should time the design with the `timing_save_pin_arrival_and_slack` variable set to `true` before using this command. If the variable is `false` (the default), `report_global_slack` might have a longer runtime.

You can choose any combination from maximum or minimum and rise or fall to report a particular slack value. The `-max` and `-min` options are mutually exclusive as are the `-rise` and `-fall` options. Use the `object_list` option to list pins or ports. If no object list is provide, then the default is all pins. For more information about the `report_global_slack` command, see the man page.

Constraint Report

The `report_constraint` command produces a summary of the constraint violations in the design, including the amount by which a constraint is violated or met and the design object that is the worst violator. PrimeTime can report on the maximum area of a design and certain timing constraints. It can also verify whether the netlist meets specific pin limits.

Timing Constraints

There are several types of timing constraints, such as

- Maximum path delay and setup
- Minimum path delay and hold
- Recovery time, the minimum amount of time required between an asynchronous control signal (such as the asynchronous clear input of a flip-flop) going inactive and a subsequent active clock edge
- Removal time, the minimum amount of time required between a clock edge that occurs while an asynchronous input is active and the subsequent removal of the asserted asynchronous control signal
- Clock-gating setup and hold
- Minimum pulse width high or low at one or several clock pins in the network
- Minimum period at a clock pin
- Maximum skew between two clock pins of a cell

Design Rules

PrimeTime performs checks to ensure that the netlist meets the design rules defined by the ASIC vendor and specified in the design library. You can also specify design rule parameters in PrimeTime, using commands such as `set_max_capacitance`.

The `report_constraint` command checks the following design rules if they are defined in the library or design:

- Total net capacitance (minimum and maximum limits)
- Pin transition time (minimum and maximum limits)
- Sum of fanout load attributes on net (minimum and maximum limits)

Usually only maximum capacitance and maximum transition are considered important. You can set the design rule constraint separately for rising and falling transitions, capacitance, and different clock domains and data paths for these clock domains.

Generating a Default Report

A default report displays brief information about the worst evaluation for each constraint in the current design and the overall cost. To report the constraints of a design, use the `report_constraint` command. For example,

```
pt_shell> report_constraint

****************************************
Report : constraint
Design : TOP
****************************************


Weighted
Group (max_delay/setup)       Cost     Weight     Cost
--------------------------------------------------------
C1                            1.50      1.00      1.50
C2                            0.00      1.00      0.00
--------------------------------------------------------
max_delay/setup                                   1.50


Weighted
Group (min_delay/hold)        Cost     Weight     Cost
--------------------------------------------------------
C1                            0.00      1.00      0.00
C2                            0.00      1.00      0.00
--------------------------------------------------------
min_delay/hold                                    0.00


Constraint                                        Cost
--------------------------------------------------------
max_delay/setup                                   1.50  (VIOLATED)
min_delay/hold                                    0.00
```

Reporting Violations

The `-all_violators` option of the `report_constraint` command displays information about all instances of constraint evaluations that were violated. To report violations in your design, use the following syntax:

```
pt_shell> report_constraint -all_violators

****************************************
Report : constraint
         -all_violators
Design : TOP
****************************************

   max_delay/setup     ('C1' group)

   Endpoint            Slack
   -------------------------------------
   OUT1                -1.50(VIOLATED)
   OUT2                -1.50(VIOLATED)
```

The `-verbose` option of the `report_constraint` command provides detailed information. For example,

```
pt_shell> report_constraint -all_violators -verbose

****************************************
Report : constraint
         -all_violators
         -verbose
Design : TOP
****************************************

  Startpoint: ff3 (rising edge-triggered flip-flop clocked by C1)
  Endpoint: OUT1 (output port clocked by C1)
  Path Group: C1
  Path Type: max
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock C1 (rise edge)                     0.00       0.00
  clock network delay (ideal)              0.00       0.00
  ff3/CP (FF)                              0.00       0.00 r
  ff3/Q (FF)                               1.00       1.00 r
  OUT1 (out)                               0.00       1.00 r
  data arrival time                                   1.00
  clock C1 (rise edge)                     3.00       3.00
  clock network delay (ideal)              0.00       3.00
  output external delay                   -3.50      -0.50
  data required time                                 -0.50
-----------------------------------------------------------
```

```
data required time                                    -0.50
data arrival time                                     -1.00
-------------------------------------------------------------
slack (VIOLATED)                                      -1.50
Startpoint: ff4 (rising edge-triggered flip-flop clocked by C2)
Endpoint: OUT2 (output port clocked by C1)
Path Group: C1
Path Type: max
Point                                    Incr       Path

-------------------------------------------------------------
clock C2 (rise edge)                     0.00       0.00
clock network delay (ideal)              0.00       0.00
ff4/CP (FF)                              0.00       0.00 r
ff4/Q (FF)                               1.00       1.00 r
OUT2 (out)                               0.00       1.00 r
data arrival time                                   1.00
clock C1 (rise edge)                     3.00       3.00
clock network delay (ideal)              0.00       3.00
output external delay                   -3.50      -0.50
data required time                                  -0.50
-------------------------------------------------------------
data required time                                    -0.50
data arrival time                                     -1.00
-------------------------------------------------------------
slack (VIOLATED)                                      -1.50
```

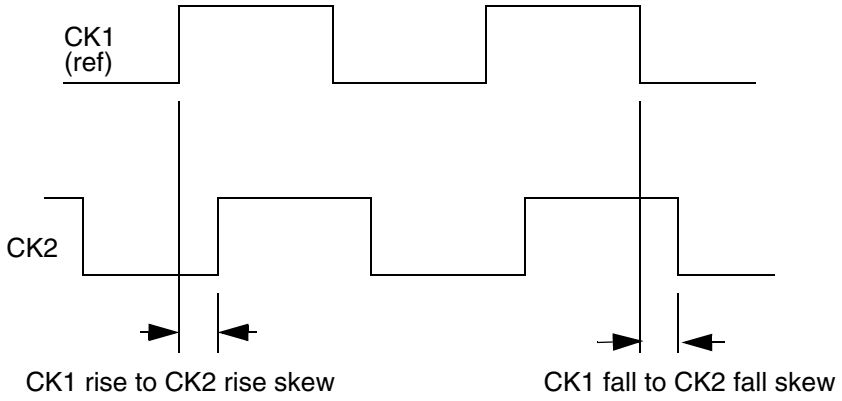
Maximum Skew Checks

You sometimes need skew timing checks on sequential devices with more than one clock signal. The skew constraint defines the maximum separation time allowed between two clock signals. You can describe skew constraints in the Synopsys library format (.lib). The following Library Compiler keywords describe skew constraints:

- `skew_rising`
- `skew_falling`

Using Library Compiler, you identify a skew constraint by defining the `timing_type` attribute as one of these two keywords in a timing group. The `related_pin` attribute specifies a reference clock pin. Figure 13-9 describes a timing diagram for skew constraint.

Figure 13-9 Timing Diagram for Skew Constraint



When the timing type is `skew_rising`, the timing constraint interval is measured from the rising edge of the reference pin (specified in `related_pin`) to a transition edge of the constrained pin. The `intrinsic_rise` value is the maximum skew time between the reference pin rising to the constrained pin rising. The `intrinsic_fall` value is the maximum skew time between the reference pin rising to the constrained pin falling.

When the timing type is `skew_falling`, the timing constraint interval is measured from the falling edge of the reference pin (specified in `related_pin`) to a transition edge of the constrained pin. The `intrinsic_rise` value is the maximum skew time between the reference pin falling to the constrained pin rising. The `intrinsic_fall` value is the maximum skew time between the reference pin falling to the constrained pin falling. You can specify either the `intrinsic_rise` or `intrinsic_fall` value. You cannot specify both.

In PrimeTime, use the `report_constraint` command with the `-max_skew` option to report maximum skew checks. PrimeTime reports only maximum skew timing checks. The `-max_skew` option checks the maximum separation time allowed between two clock signals. The default displays all timing and design rule constraints.

PrimeTime calculates the actual skew for the specific skew check by taking the absolute value of the difference between delayed reference and constrained clock edges. The ideal reference clock edge is delayed by minimum clock latency to the reference pin; the ideal

constrained clock edge is delayed by maximum clock latency to the constrained pin. Clock uncertainties are not considered for the skew timing checks. To report maximum skew for all violators, use the following syntax:

```
pt_shell> report_constraint -max_skew -all_violators
                   Required           Actual
   Pin             Skew               Skew            Slack
-------------------------------------------------------------------------
   ff3/c (r->f)    0.15               5.47           -5.32  (VIOLATED)
   ff2/c (f->f)    0.14               2.32           -2.18  (VIOLATED)
   ff2/c (r->r)    0.12               1.18           -1.06  (VIOLATED)
   ff4/c (f->f)    0.14               0.84           -0.70  (VIOLATED)
   ff4/c (r->r)    0.12               0.42           -0.30  (VIOLATED)
```

To report detailed information, use the following syntax:

```
pt_shell> report_constraint -max_skew -verbose

Constrained Pin: ff2/c
  Reference Pin: ff2/cn
  Check: max_skew

  Point                                          Incr          Path
  ---------------------------------------------------------------------
  clock sclk (rise edge)                         0.00          0.00
  clock network delay (ideal)                    1.40          1.40 r
  ff2/cn                                         0.00          1.40
  reference pin arrival time                                   1.40

  clock mclk (rise edge)                         0.00          0.00
  clock network delay (ideal)                    1.60          1.60 r
  ff2/c                                          0.00          1.60 r
  constrained pin arrival time                                 1.60
  ---------------------------------------------------------------------
  allowable skew                                               0.12
  actual skew                                                  0.20
  ---------------------------------------------------------------------
  slack (VIOLATED)                                            -0.08
```

No-Change Timing Checks

Certain signals need no-change timing checks to make sure that they do not switch during the active interval of a periodic signal such as a clock. A no-change check is described in the library.

Performing a no-change check is equivalent to performing a setup check against the active edge transition and a hold check against the inactive edge transition. For a sequential element with an active-high clock, the no-change setup check is performed against the rising clock edge and the hold check is performed against the falling clock edge.

The `report_timing` and `report_constraint` commands report no-change checks as library no-change setup time and library no-change hold time.

In the following report, pay particular attention to the lines in bold.

```
Startpoint: EN1 (level-sensitive input port clocked by CLK)
Endpoint: UCKLENH (positive nochange timing check clocked
by CLK')
Path Group: CLK
Path Type: max

Point                                   Incr          Path
-----------------------------------------------------------
clock CLK (rise edge)                   0.00          0.00
clock network delay (ideal)             0.00          0.00
input external delay                    1.00          1.00 r
EN1 (in)                                0.00          1.00 r
U24/A (N1A)                             0.00          1.00 r
U24/Z (N1A)                             0.07          1.07 f
U25/A (N1B)                             0.00          1.07 f
U25/Z (N1B)                             0.08          1.15 r
U26/A (N1C)                             0.00          1.15 r
U26/Z (N1C)                             0.07          1.21 f
U27/A (N1D)                             0.00          1.21 f
U27/Z (N1D)                             0.04          1.25 r
UCKLENH/EN (LD1QC_HH)                   0.00          1.25 r
data arrival time                                     1.25

clock CLK' (rise edge)                  2.00          2.00
clock network delay (ideal)             0.00          2.00
UCKLENH/G (LD1QC_HH)                    0.00          2.00 r
library nochange setup time            -0.32          1.68
data required time                                    1.68
-----------------------------------------------------------
data required time                                    1.68
data arrival time                                    -1.25
-----------------------------------------------------------
slack (MET)                                           0.43
```

Unit Reporting

The `report_units` command generates a list of the units used in the current design. The following shows a sample report:

```
pt_shell> report_units

****************************************
Report : units
Design : Design1
Version: X-2005.06

Date   : Wed Mar 30 21:04:08 2005
****************************************
Units
------------------------------------------
Capacitive_load_unit        : 5.82e-15 Farad
Current_unit                : 0.001 Amp
Resistance_unit             : 171821 Ohm
Time_unit                   : 1e-09 Second
Voltage_unit                : 1 Volt
1
```

The report is always generated in the standard units as shown above, regardless of the definition in the main library. For example, the capacitance is always reported in farads, even though the main library might have picofarads. The same applies to all other parameters: the current is in amps, resistance in ohms, time in seconds, and voltage in volts.

The report scales the value based on the main library definition, so it always shows the correct value in standard units. For example, in the example above, the current unit is reported as 0.001 Amp, but in the library it has been defined as 1 mA.

Clock-Gating and Recovery/Removal Checks

By default, PrimeTime automatically determines clock gating and performs clock-gating setup and hold checks. To disable reporting clock-gating setup and hold checks, set the `timing_disable_clock_gating_checks` variable to `true`.

By default, PrimeTime performs recovery and removal checks. To disable reporting recovery and removal checks, set the `timing_disable_recovery_removal_checks` variable to `true`.

14

Command Interface and Tcl

The PrimeTime command interface is based on the Tcl command language. This product includes software developed by the University of California, Berkeley and its contributors.

Tcl concepts and the command interface are described in the following sections:

Tcl Syntax and PrimeTime Commands

The PrimeTime user interface is based on Tcl version 8.4. Using Tcl, you can extend the PrimeTime command language by writing reusable procedures. The Tcl language has a straightforward syntax. Every Tcl script is viewed as a series of commands, separated by a new-line character or semicolon. Each command consists of a command name and a series of arguments.

There are two types of PrimeTime commands: application and built-in commands. Each type is described in the following sections. Other aspects of Tcl version 8.4 are also described. If you need more information about the Tcl language, consult books on the subject in the engineering section of your local bookstore or library.

Using Tcl Special Characters

The characters listed in Table 14-1 have special meaning for Tcl in certain contexts.

Table 14-1 Special Characters

Character	Meaning
`$`	Dereferences a variable.
`( )`	Used for grouping expressions.
`[ ]`	Denotes a nested command.
`\`	Used for escape quoting.
`" "`	Denotes weak quoting. Nested commands and variable substitutions still occur.
`{ }`	Denotes rigid quoting. There are no substitutions.
`;`	Ends a command.
`#`	Begins a comment.

Basic System Commands

Table 14-2 lists some common tasks and the basic system commands for performing them.

Table 14-2 Common Tasks and Their System Commands

To do this	Use this
List the current PrimeTime working directory.	`pwd`
Change the PrimeTime working directory to a specified directory or, if no directory is specified, to your home directory.	`cd` *`directory`*
List the files specified, or list all files in the working directory if no arguments are specified. Requires an `sh` program in your path.	`ls` *`directory_list`*
Search for a file using `search_path`.	which *`filename`*
Generate the current date and time.	`date`
Execute an operating system command. You should always use the `exec` command before using the `sh` command. This Tcl built-in command has some limitations. For example, it does not perform any expansion. For more information, see the `exec` man page.	`exec` *`command`*
Execute an operating system command. Unlike `exec`, this command performs file name expansion. Requires an `sh` program in your path.	`sh` *`command`*
Return the value of an environment variable.	getenv *`name`*
Set the value of an environment variable. Any changes to environment variables apply only to the current PrimeTime process and to any child processes launched by the current PrimeTime process.	setenv *`name value`*
Display the value of one or all environment variables.	printenv *`variable_pattern`*

Using the Result of a Command

PrimeTime commands return a result that is interpreted by other commands as a string, Boolean, integer, and so forth. With nested commands, the result can be used as the following:

- Conditional statement in a control structure
- Argument to a procedure
- Value to which a variable is set

An example of using a result is as follows:

```
if {[expr $a + 11] <= $b} {
   echo "Done"
}
   return $b
```

Using Built-In Commands

Most built-in commands are intrinsic to Tcl. Their arguments do not necessarily conform to the PrimeTime argument syntax. For example, many Tcl commands have options that do not begin with a dash, but do have a value argument. The Tcl `string` command has a `compare` option that you use like this:

```
string compare string1 string2
```

PrimeTime Extensions and Restrictions

Generally, PrimeTime implements all the Tcl built-in commands. However, PrimeTime adds semantics to some Tcl built-in commands and imposes restrictions on some elements of the language. The differences are

- Tcl `rename` command is limited to procedures you have created.
- Tcl `load` command is not supported.
- Unable to create an unknown command.
- Auto-exec feature found in tclsh is not supported; however, autoload is supported.
- Tcl `source` command has additional options: `-echo` and `-verbose` that are nonstandard to Tcl.
- `history` command has additional options, `-h` and `-r`, nonstandard to Tcl, and the form `history` *n*. For example, `history 5` lists the last five commands.

- PrimeTime command processor processes words that look like bus (array) notation (words that have square brackets, such as `a[0]`), so that Tcl does not try to execute the index as a nested command. Without this processing, you would need to rigidly quote such array references, as in `{a[0]}`.
- Always use braces (`{}`) around all control structures and procedure argument lists. For example, quote the `if` condition as follows:

```
if {! ($a > 2) } {
  echo "hello world"
}
```

Redirecting and Appending Output

You can direct the output of a command, procedure, or a script to a specified file in two ways:

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

The UNIX style redirection operators cannot be used with built-in commands. You must use the `redirect` command when using built-in commands. See "Using the redirect Command" on page 14-6.

You can use either of the following two commands to redirect command output to a file:

```
redirect temp.out {report_timing -nworst 3}

report_timing -nworst 3 > temp.out
```

You can use either of the following two commands to append command output to a file:

```
redirect -append temp.out {report_timing -nworst 3}

report_timing -nworst 3 >> temp.out
```

Note:

The Tcl built-in command `puts` does not respond to redirection of any kind. Instead, use the `echo` command, which responds to redirection.

Using the redirect Command

In an interactive session, the result of a redirected command that does not generate a Tcl error is an empty string. For example,

```
pt_shell> redirect -append temp.out { history -h }
pt_shell> set value [redirect blk.out {plus 12 34}]
pt_shell> echo "Value is <$value>"
Value is <>
```

Screen output from a redirected command occurs only when there is an error. For example,

```
pt_shell> redirect t.out { create_clock -period IO [get_port CLK]}
Error: Errors detected during redirect
       Use error_info for more info. (CMD-013)
```

This command had a syntax error because IO is not a floating-point number. The error is in the redirect file.

```
pt_shell> exec cat t.out
Error: value 'IO' for option '-period' not of type 'float'
(CMD-009)
```

The `redirect` command is more flexible than traditional UNIX redirection operators. The UNIX style redirect operators `>` and `>>` are not part of Tcl and cannot be used with built-in commands. You must use the `redirect` command with built-in commands. For example, you can redirect expr $a > 0 only with the following:

```
redirect file {expr $a > 0}
```

With the `redirect` command you can redirect multiple commands or an entire script. As a example, you can redirect multiple `echo` commands as follows:

```
redirect e.out {
    echo -n "Hello"
    echo "world"
}
```

Getting the Result of Redirected Commands

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

```
redirect p.out {
       set rvs [read_verilog a.v]
}
if {$rvs == 0} {
      echo "read_verilog failed! Returning..."
      return
}
```

Using the Redirection Operators

Because Tcl is a command-driven language, traditional operators usually have no special meaning unless a particular command, such as `expr`, imposes some meaning. PrimeTime commands respond to `>` and `>>` but, unlike UNIX, PrimeTime treats the `>` and `>>` as arguments to the command. Therefore, you must use white space to separate these arguments from the command and the redirected file name. For example,

```
echo $my_variable >> file.out; # Right
echo $my_variable>>file.out; # Wrong!
```

Keep in mind that the result of a command that does not generate a Tcl error is an empty string. To use the result of commands you are redirecting, you must use the `redirect` command.

The UNIX style redirect operators `>` and `>>` are not part of Tcl and cannot be used with built-in commands. You must use the `redirect` command with built-in commands.

Using Command Aliases

You can use aliases to create short forms for the commands you commonly use. For example, this command duplicates the function of the dc_shell `include` command when using PrimeTime:

```
pt_shell> alias include "source -echo -verbose"
```

After creating a new alias, you can use it by entering this command:

```
pt_shell> include commands.pt
```

When you use aliases, keep the following points in mind:

- PrimeTime recognizes an alias only when it is the first word of a command.
- An alias definition takes effect immediately but lasts only until you exit the PrimeTime session. To save commonly used alias definitions, store them in the .synopsys_pt.setup file.
- You cannot use an existing command name as an alias name; however, aliases can refer to other aliases.
- Aliases cannot be syntax checked. They look like undefined procedures. For more information about syntax checking, see the *PrimeTime Advanced Timing Analysis User Guide*.

Interrupting Commands

If you enter the wrong options for a command or enter the wrong command, you can interrupt command processing by pressing Control-c. The time the command takes to respond to an interrupt (to stop what it is doing and return to the prompt) depends on the size of the design and the function of the command being interrupted.

If PrimeTime is processing a script file and you interrupt one of the commands in the script, the processing is interrupted and PrimeTime does not process any other commands in that script. If you press Control-c three times before a command responds to your interrupt, pt_shell is interrupted and exits with this message:

```
Information: Process terminated by interrupt.
```

You cannot interrupt some commands, such as the `update_timing` command. To stop these commands, you must terminate the pt_shell process at the system level. The exceptions to this behavior are documented with the applicable commands.

Using Scripts

You can use the `source` command to execute scripts in PrimeTime. A script file, also called a command script, is a sequence of pt_shell commands in a text file.

By default, the `source` command executes the specified script file without showing the commands or the system response to the commands. The `-echo` option causes each command in the script to be displayed as it is executed. The `-verbose` option causes the system response to each command to be displayed.

Within a script file you can execute any PrimeTime command. The script can be simple ASCII or gzip compressed. The source command is also used to load bytecode-compiled scripts. For more information, see the *PrimeTime Advanced Timing Analysis User Guide*.

Adding Comments

You can add block comments to script files by beginning comment lines with the pound sign (#). Add inline comments using a semicolon to end the command, followed by the pound sign to begin the comment. For example,

```
#
# Set the new string
#

set newstr "New"; # This is a comment.
```

Controlling Script Processing When Errors Occur

By default, when a syntax or semantic error occurs while executing a command in a script, PrimeTime discontinues processing the script. To change the default behavior, use the `sh_continue_on_error` and `sh_script_stop_severity` variables.

To force PrimeTime to continue processing the script no matter what, set the `sh_continue_on_error` variable to `true`. This is not usually recommended because the remainder of the script might not perform as expected if a command fails due to syntax or semantic errors, such as an invalid option.

To get PrimeTime to stop the script when certain kinds of messages are issued, use the `sh_script_stop_severity` variable. This is set to `none` by default. Set it to `E` to get the script to stop on any message with error severity. Set it to `W` to get the script to stop on any message with warning severity.

Note:

The `sh_script_stop_severity` variable has no impact if the `sh_continue_on_error` variable is set to `true`.

Finding Scripts Using the search_path Variable

By default, the `source` command treats its `script_file_name` argument as an absolute file name. To cause PrimeTime to find the `script_file_name` using the search path, set the `sh_source_uses_search_path` variable to `true`. The default is `false`. For example,

```
pt_shell> set sh_source_uses_search_path true
```

Logging Source File Contents

When you source a script, the `source` command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment.

To disable this logging, set the `sh_source_logging` variable to `false`. The default is `true`. For example,

```
pt_shell> set sh_source_logging false
```

Listing and Running Previous Commands

The `history` command lists the commands used in a pt_shell session. With no arguments, the `history` command lists the last *n* commands that you entered. The default is `20`.

The `history` command is complex and can generate various forms of output. This section mentions some commonly used features.

To review the last 20 commands you entered, use the `history` command. Something similar to the following is displayed:

```
pt_shell> history
1 source basic.tcl
2 read_db middle.db
.
.
18 current_design middle
19 link
20 history
```

To change the length of the history buffer, use the `keep` option. For example, this command specifies a history of 50 commands:

```
pt_shell> history keep 50
```

To limit the history list to three commands and to display them in reverse order, enter

```
pt_shell> history -r 3
20 history info 3
19 link
18 current_design middle
```

You can also redirect the output of the `history` command to create a command script:

```
pt_shell> redirect my_script {history -h}
```

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

Rerunning Previously Entered Commands

You can rerun and recall previously entered commands by using the exclamation point (`!`) operator. Table 14-3 lists the shortcuts you can use to rerun commands. Place these shortcuts at the beginning of a command line. They cannot be part of a nested command or a script.

Table 14-3 Shortcut for Rerunning Commands

To rerun	Use
The last command	`!!`
The *n*th command from the last	`!-n`
The command numbered *n* (from a history list)	`!n`
The most recent command that started with `text` (Note: `text` must begin with a letter or underscore (_) and can contain numbers.)	`!text`

You can recall the `current_design` command, by entering

```
pt_shell> history
1 source basic.tcl
2 read_db middle.db
3 current_design middle
4 link
5 history
```

You can then rerun the third command, by entering

```
pt_shell> !3
```

Modifying the Previous Command Before Rerunning

You can modify and rerun the previous command executed using the `^x^y` form. This recalls the last command, replaces all instances of `x` with `y`, then executes the command. This is different from many UNIX shells, which only replace the first instance of `x` with `y`. For example, the following command was typed incorrectly, and was easily corrected:

```
pt_shell> sorce sort.tcl
Error: unknown command 'sorce' (CMD-005)
pt_shell> ^sorce^source
source sort.tcl
1
```

Use more characters to help isolate the change. In the previous example, changing `sor` to `sour` would have modified the command, but it also would have changed the argument to `sourt.tcl`, which was not intended.

Suppressing Warning and Informational Messages

PrimeTime provides commands for suppressing and redisplaying warning and informational messages.

Note:
You cannot suppress error messages and fatal error messages.

Table 14-4 summarizes the commands for controlling the output of warning and informational messages. You can use these commands within a procedure (for example, to turn off specific warnings). If you suppress a message *n* times, you must unsuppress the message the same number of times to enable its output again. Use these commands carefully. For more information, see the man pages.

Table 14-4 Commands for Controlling Message Output

Task	Command
Disable on-screen reporting of one or more messages	`suppress_message`
Enable on-screen reporting of previously disabled messages	`unsuppress_message`
Display the currently suppressed message IDs	`print_suppressed_messages`
Disable on-screen reporting of messages after a specified number of occurrences	`set_message_info`
Obtain information about the total number of messages generated, the number of occurrences of a specified message, or the limit set on reporting of a message	`get_message_info`

Using Variables

PrimeTime variables can be divided into two categories: application variables and user-defined variables. The application variables are standard variables that control the behavior of PrimeTime. You create user-defined variables for any desired purpose. The following table summarizes commands that manage variables.

Task	Command

Assign a value to a variable	`set`
	For example,
	`set x 27`

List variables with their current values	`printvar`
	`echo $variable_name`
	`set variable_name`

Substitute the value of a variable	`$`. For example, `set x 27  set y $x`

Remove user-defined variables	`unset`
	For example,
	`unset x`
	Note: You cannot remove application variables.

These commands set and remove variables:

```
pt_shell> set search_path ". /usr/synopsys/libraries"
. /usr/synopsys/libraries

pt_shell> set adir "/usr/local/lib"
/usr/local/lib

pt_shell> set my_path "$adir $search_path"
/usr/local/lib . /usr/synopsys/libraries

pt_shell> unset adir

pt_shell> unset my_path
```

You can set variables to the result of a command as follows:

```
pt_shell> set x [get_ports *]
```

Listing Variables

The `printvar` command list variables. Table 14-5 summarizes the `printvar` command and its options.

Table 14-5 Commands for Listing Variables

To list this	Use this
All variables, alphabetically with their current values	`printvar`
All variables that match a specified pattern, alphabetically with their current values	`printvar` *`pattern*`*
A specified variable with its current value	`printvar` *`variable_name`* `echo $`*`variable_name`* `set` *`variable_name`*
PrimeTime Suite variables only	`printvar -application`
User-defined variables	`printvar -user_defined`

To list the path variables, enter

```
pt_shell> printvar *path
link_path                      = "* lib.db"
search_path                    = ". /usr/designs /usr/libraries"
sh_source_uses_search_path = "false"
```

User-Defined Variables

PrimeTime supports any number of user-defined variables. Variables are either scalar or arrays.

The syntax of an array reference is:

```
array_name(element_name)
```

Arrays can be managed using the `array` command. For more information about the `array` command, see the referenced Tcl documentation. The following examples show how to set, unset, and examine variables:

```
pt_shell> set x 3
3
pt_shell> set A(1) 27
27

pt_shell> set A(hello) there
there

pt_shell> array names A
1 hello

pt_shell> unset x
pt_shell>
```

By default, when you create a new variable by using the `set` command at the pt_shell prompt, a message similar to the following appears:

```
pt_shell> set new 2
Information: Defining new variable 'new'. (CMD-041)
```

This message can be useful in interactive mode because it warns you when you create a new variable unintentionally. For example, if you intend to set a system variable but mistype it slightly, the warning message tells you that you created a new variable instead of setting the existing system variable.

The following variables control the generation of CMD-041 messages:

- `sh_new_variable_message` (default: `true`)
- `sh_new_variable_message_in_proc` (default: `false`)
- `sh_new_variable_message_in_script` (default: `false`)

By default, the CMD-041 message is generated in interactive mode, but not during execution of procedures or scripts (when you are unlikely to be looking for messages). To enable generation of these messages during execution of procedures or scripts, set the variables appropriately. Note that doing so can cause a significant increase in runtime. For more information, see the man pages for the variables.

Using Collections

PrimeTime builds an internal database of the netlist and the attributes applied to the database. This database consists of several classes of objects, such as designs, libraries, ports, cells, nets, pins, and clocks. Most PrimeTime commands operate on these objects.

A collection is a group of objects exported to the Tcl user interface. Collections have an internal representation (the objects), and sometimes a string representation. The string representation is generally used only for error messages.

You can create collections of objects, then apply a set of commands to interact with those collections. Collections can be homogeneous (contain objects of one type) or heterogeneous (contain objects of many types).

Note:

In some ways creating collections of objects in PrimeTime is significantly different from doing so in Design Compiler. For an explanation of the differences, see the *PrimeTime Advanced Timing Analysis User Guide*.

The collection commands are divided into the following categories:

- Commands that create collections of objects for use by another command
- Commands that manipulate collections
- Commands that query objects for you to view

You can use wildcards and filtering criteria to narrow the focus of a collection. You can store collections in variables for use in setting attributes, or for performing custom reporting.

Creating Collections

In PrimeTime, you can pass a collection to a command by using the result of a command that creates the collection or by using a variable that contains a collection. For example, the following command creates a collection of nets and passes that collection to a variable named mynets:

```
pt_shell> set mynets [get_nets *]
```

Primary Commands That Create Collections

Table 14-6 lists the primary commands that create collections of objects.

Table 14-6 Primary Commands for Creating Collections

To create this collection	Use this command
Cells or instances	`get_cells`
Designs	`get_designs`
Libraries	`get_libs`
Library cells	`get_lib_cells`
Library cell pins	`get_lib_pins`
Nets	`get_nets`
Pins	`get_pins`
Ports	`get_ports`

When used at the command prompt, any command that creates a collection implicitly queries the collection. You can view the contents of a collection by using the `query_objects` command. See "Querying Objects" on page 14-20. Do not use `echo`, `puts`, or `printvar`, as this only returns the string representation of the collection, a name arbitrarily assigned by PrimeTime to serve as a pointer to the collection.

Primary Collection Command Example

The `get_cells` command creates a collection of cells in the design. For example, enter the following command to query the cells that begin with *o* and reference an FD2 library cell:

```
pt_shell> get_cells "o*" -filter {ref_name == FD2}
{"o_reg1", "o_reg2", "o_reg3", "o_reg4"}
```

Although the output looks like a list, it is not. The output is only a display of the `get_cells` result. The collection was not saved in a variable, nor passed to another command, so it was deleted as soon as it was displayed. Given a collection of pins, use the following commands to query the cells connected to those pins:

```
pt_shell> set pins [get_pins o*/CP]
{"o_reg1/CP", "o_reg2/CP"}
pt_shell> get_cells -of_objects $pins
{"o_reg1", "o_reg2"}
```

Enter the following command to remove the wire load model from the `i1` and `i2` cells:

```
pt_shell> remove_wire_load_model [get_cells {i1 i2}]
Removing wire load model from cell 'i1'.
Removing wire load model from cell 'i2'.
1
```

The primary collection commands, such as the `get_cells` and `get_pins` commands, can take collections of the same type as arguments. For example, you can use a syntax similar to the following:

```
get_cells [get_cells u1]
```

This feature is often useful for writing procedures that can take a pattern or a collection as an argument.

Other Commands That Create Collections

Table 14-7 lists other commands that create collections of objects.

Table 14-7 Other Commands for Creating Collections

To create this collection	Use this command
Cells or instances	`all_instances`
Clocks	`get_clocks, all_clocks`
Fanin of pins, ports, or nets	`all_fanin`
Fanout of pins, ports, or nets	`all_fanout`
Interface logic objects	`get_ilm_objects`
Objects connected to a net, pin, or port	`all_connected`
Path groups	`get_path_groups`
Ports	`all_inputs, all_outputs`
Quick timing model ports	`get_qtm_ports`
Register cells or pins	`all_registers`
Timing paths for further processing	`get_timing_paths`

Each collection created by a command, such as the `get_nets` command has its own context. You cannot add to, remove from, or compare collections from different contexts such as different designs, different libraries, different sets of paths created by different `get_timing_paths` commands, or different sets of arcs created by different `get_timing_arcs` commands. To compare objects, create one large collection containing all the objects of interest, and then sort, filter, or manipulate the objects in that collection. For example, the following commands create two different sets of paths that cannot be compared:

```
pt_shell> set paths [get_timing_paths ...]
pt_shell> set more_paths [get_timing_paths ...]
```

Even if the two collections represent the exact same paths, they are different objects, so comparing them reports a mismatch.

Saving Collections

You can save collections by assigning the result of a command to a variable. To save a collection for later use, set a variable equal to the collection, as in this command sequence:

```
pt_shell> set data_ports [get_ports {data[*]}]
{"data[2]", "data[1]", "data[0]"}

pt_shell> set_input_delay 2.0 $data_ports
1
```

The result of the first command is a collection. PrimeTime automatically displays the contents of the collection. Then the collection is passed into the `set_input_delay` command. You can deallocate the collection by unsetting the variable to which the collection was assigned (`data_ports`).

A collection is active only as long as it is referenced. Typically, a collection is referenced when a variable is set to the result of a command that creates it, or when it is passed as an argument to a command or a procedure. For example, if you save a collection of ports using the `set myports [get_ports *]` command, either of the following commands delete the collection referenced using the `myports` variable:

```
pt_shell> unset myports

pt_shell> set myports "newvalue"
```

Collections can be implicitly deleted when they go out of scope, in other words, when the parent of the objects within the collection is deleted. For example, if your collection of ports is owned by a design, the collection is implicitly deleted when the design that owns the ports

is deleted. When a collection is implicitly deleted, the variable that referenced the collection still holds a string representation of the collection. However, since the collection is gone, this string value is not useful.

Querying Objects

To view the contents of a collection, use the `query_objects` command. This command searches for and displays objects in the PrimeTime database. Do not use `echo`, `puts`, or `printvar`, as doing so returns only the string representation the collection, which is a name arbitrarily assigned by PrimeTime to serve as a pointer to the collection.

The output of the `query_objects` command is similar to the dc_shell `find` command. The `query_objects` command does not have a meaningful return value. It displays the objects found and returns the empty string. For more information, see the `query_objects` man page.

Implicit Query

When commands that create collections are issued from the command prompt, they implicitly query the collection. You can control how many objects are displayed from the collection by using the `collection_result_display_limit` variable.

Use the following command to see the ports that match the pattern `in*`:

```
pt_shell> get_ports in*
{"in0", "in1", "in2"}
```

You can also use the following commands to see the same ports.

```
pt_shell> query_objects [get_ports in*]
{"in0", "in1", "in2"}

pt_shell> query_objects -class port in*
{"in0", "in1", "in2"}
```

Using Wildcard Characters

Many commands that create explicit collections allow you to choose between simple wildcard patterns, using the `*` and `?` wildcard characters, or complete regular expressions, using the `-regexp` option. The search capability is the same whether you use wildcards or the `-regexp` option to describe the patterns. For more information about regular expressions, see the *PrimeTime Advanced Timing Analysis User Guide* and specific man page.

When using wildcards, PrimeTime uses some UNIX glob-style matching operators, including the following:

- `*` – Matches 0 to *n* characters
- `?` – Matches exactly one character

For example, given ports `i0`, `i1`, `in0`, and `in1`, notice the queries in this sequence of commands:

```
pt_shell> get_ports i*
{"i0", "i1", "in0", "in1"}

pt_shell> query_objects [get_ports i?]
{"i0", "i1"}
```

Commands that create explicit collections, such as the `get_cells` command, allow you to search in the current instance or hierarchically by using a `-hierarchical` option. The rules for different kinds of searches are as follows:

- Using a wildcard pattern alone matches leaf names in the current instance. For example,

  ```
  get_cells i1*
  ```

- Using a wildcard pattern with the `-hierarchical` option matches leaf names at each level of the hierarchy. For example, to find all cells in the hierarchy with the leaf name containing `n1`,

  ```
  get_cells *n1* -hierarchical
  ```

- When you search for a wildcard pattern that contains the hierarchy separator, PrimeTime breaks up the pattern around the hierarchy separator and matches each piece at progressively deeper levels of the hierarchy. For example, to find the cells in `i1` that begin with `i2`, then return a collection of cells in each `i1/i2*` that has a leaf name of `n1`, enter

  ```
  get_cells i1/i2*/n1
  ```

If an object you specify has a name that contains a backslash character (`\`) or any wildcard characters (`*` or `?`), it is better to use the `-exact` option to find the object. Using the `-exact` option makes the primary collection creation command, such as the `get_cells` or `get_ports` command, consider the *patterns* argument to be a list of exact strings rather than a list of patterns with wildcards. For example,

```
get_cells -exact [list {*cell*1}] ;# finds cell actually named
*cell*1
get_cells -exact [list {a\b1}]    ;# finds cell actually named a\b1
```

Wildcard matching uses the normal * and ? operators. Without the -exact option, each wildcard match character needs to be backslash-escaped in order to be considered exactly (not a wildcard). For example,

```
get_cells [list {\*cell\*1}]
get_cells [list {a\\b1}]
```

Remember that the *patterns* argument to the collection creation command is a list. If you do not supply a well-formed list, additional backslashes are necessary to communicate these special characters.

Filtering Collections

You can filter collections by using the -filter option with the primary commands that create collections (see commands listed in Table 14-6), or you can use the filter_collection command.

Using -filter Options

Many commands that create collections accept a -filter option that specifies a filter expression. A filter expression is a string composed of a series of logical expressions describing a set of constraints you want to place on a collection.

Each subexpression of a filter expression is a relation contrasting an attribute name (such as area or direction) with a value (such as 43 or input), by means of an operator, such as == or !=).

The following command gets the cells in U1 that have an area no greater than 12 or reference a design (or library cell) named AN2P, AO2P, and so on. The command then assigns the collection to the Cells variable.

```
pt_shell> set Cells [get_cells "U1/*" \
            -filter {area <= 12 || ref_name =~ "A*P"}]
```

For more information, see the chapter called “Object Attributes” in the *PrimeTime Advanced Timing Analysis User Guide*.

Using Filter Operators

The filter language supports the following logical operators:

- AND or && – Logical AND (not case-sensitive)
- OR or || – Logical OR (not case-sensitive)

You can group logical expressions with parentheses to enforce order; otherwise, PrimeTime evaluates the expressions from left to right.

The filter language supports the following relational operators:

- == Equal
- != Not equal
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- =~ Matches pattern
- !~ Does not match pattern

In the following filter expression,

```
{area <= 12 || ref_name =~ "A*P"}
```

In the above example, `area` is an attribute (or identifier), `<=` is a relational operator, `12` is a value, and `||` is the logical OR operator.

The filter language also supports the following existence operators:

- defined
- undefined

An existence operator determines whether an attribute is defined for an object. For example,

```
sense == setup_clk_rise and defined(sdf_cond)
```

The right side of a relation can consist of a string or a numeric literal. You do not need to enclose strings in quotation marks. This method is useful because a filter expression is usually the value for an argument, and the entire expression is enclosed in quotation marks.

As shown below, you do not need to enclose the word *in* with quotation marks.

```
pt_shell> set iports [get_ports * -filter {direction == in}]
```

However, if an expression contains characters that are part of the filter language syntax, you must use curly braces to enclose the expression and double quotation marks to enclose string operands. As shown in the example below, parentheses are part of the filter language, so they are double quoted; and the complete expression is grouped in curly braces.

```
pt_shell > set x [filter_collection $ports {full_name =~ "D(*)"}
```

Relation Rules

Parsing a filter expression can fail because of the following:

- Syntax problems
- Invalid attribute name
- Type mismatch between an attribute and the value

Here are the basic relational rules:

- String attributes can be compared with any operator
- Numeric attributes cannot be compared with patterns
- Boolean attributes can be compared only with `==` and `!=`

  The value is either `true` or `false`.

Using Pattern Match Filter Operators

The pattern match filter operators, `=~` and `!~`, behave differently in different circumstances. They do one of two types of pattern matching:

- Simple wildcard matching (the default)
- Full regular expression matching

Commands that provide a `-filter` option also provide a `-regexp` option to enable you to change the filtering to full regular expressions. The `filter_collection` command also provides a `-regexp` option to enable full regular expressions.

If the patterns are equivalent, the `-regexp` option has no impact on the results of the `get_cells`, `get_pins`, and `get_nets` commands. For example, the following two commands retrieve the same results:

```
get_cell -regexp U1.*
get_cell U1*
```

Similarly, the following two commands retrieve the same results:

```
get_cell -regexp U1.* -hier
get_cell U1* -hier
```

Using the filter_collection Command

The `filter_collection` command is useful for filtering an existing collection. However, it is usually less efficient than using the `-filter` option (described in "Filtering Collections" on page 14-22).

The filter_collection command takes a collection and a filter expression as arguments. The result of the filter_collection command is a new collection or an empty string if no objects match the criteria.

The following commands are equivalent:

```
get_cells * -filter "ref_name =~ A*P"

filter_collection [get_cells *] "ref_name =~ A*P"
```

The filter_collection command is useful for translating Design Compiler scripts that use the -filter option. In addition, if you derive several collections from one larger collection, using the filter_collection command might be more efficient than using the -filter option, as shown in this series of commands:

```
pt_shell> set acells [get_cells "*"]
{"u1", "u2", "u3", "u4"}

pt_shell> set ands [filter_collection $acells \
            "ref_name =~ AN*"]
{"u1", "u2"}
pt_shell> set ors [filter_collection $acells \
            "ref_name =~ OR*"]
{"u3", "u4"}
```

Optionally, the filter_collection command accepts the -regexp option, which changes =~ and !~ to perform real regular expression matching.

Using Implicit Collections As Arguments

Many PrimeTime commands have arguments that accept a list of patterns. In such an argument, each pattern is either a string representation of a collection or a string that matches one of a command-specific series of object classes. For example, the set_input_delay command accepts ports and pins, in that order. Therefore, the following command is a valid construct:

```
pt_shell> set_input_delay 5.0 [list out4 [get_pins U1/*] n*1]
```

In the previous command, set_input_delay first tries to find the out4 port and if it is not found, tries to find the out4 pin. The pin collection of U1/* is explicit. The n*1 pattern is matched against ports. If there is no match, it is matched against pins.

The patterns out4 and n*1 are the implicit elements of the collection. For convenience, you can use the name of the individual object rather than an explicit collection, such as [get_ports out4], especially for singular objects.

Iterating Over the Elements of a Collection

The `foreach_in_collection` command iterates over a collection. You can nest the `foreach_in_collection` command within other control structures, including another `foreach_in_collection` command. For more information, see the man page.

During each iteration, the `itr_var` variable is set to a collection of exactly one object. Any command that accepts one of the collections also accepts the `itr_var` variable because they are of the same type (a string representation of a collection). To generate a separate report for each cell, use these commands:

```
pt_shell> foreach_in_collection itr [get_cells o*] {
            redirect [format "%s%d" $fbase $ndx]
            report_cell
          }
```

Note:

You can use the `foreach` command to iterate over a list, but not a collection. Attempting to do so causes the collection to be deleted. For more information, see “Iterating Over a List: foreach” on page 14-37.

Limitations of the foreach_in_collection Command

The `foreach_in_collection` iterator command is sensitive to changes in the netlist. If you are iterating over the elements of a collection and some netlist changes cause the collection to be deleted, the iteration terminates. A message appears indicating that the collection was deleted. For this reason, it is not recommended to use commands, such as the `swap_cell` command within the `foreach_in_collection` command. Instead, use the command to create a list of cells that you want to swap out, then make one call to `swap_cell` at the end.

Individual elements of some collections cannot be exported from within the `foreach_in_collection` command. For example,

```
set the_cell ""
foreach_in_collection c [get_cells *] {
 if { [get_attribute $c mark] == "true" } {
  set the_cell $c
  break
 }
}
```

In this example, when the iteration is completed, `the_cell` is a collection of one cell. For many collections, this is acceptable; however, for `timing_paths`, `timing_arcs` and `lib_timing_arcs`, this is not allowed. In these cases, when you get to the end of the iteration, any collections created from the base collection are deleted.

Removing From and Adding to a Collection

PrimeTime allows you to remove objects from or add objects to a collection by using the `remove_from_collection` and `add_to_collection` commands.

You can remove objects with the `filter_collection` command or by filtering objects when you initially create the collection. However, there are some constructs, such as removing the elements in one collection from another collection, that you cannot accomplish with filtering. Use the `remove_from_collection` and `add_to_collection` commands for this and other purposes.

The arguments to both commands are a collection and a specification of the objects that you want to add or remove. The specification can be a list of names, wildcard strings, or other collections.

The result of the commands is a new collection, or an empty string if the operation results in zero elements (in the case of the `remove_from_collection` command). The first argument (the base collection) is a read-only argument. The syntax is as follows:

```
add_to_collection collection object_spec [-unique]
remove_from_collection collection object_spec
```

collection

The base collection. The base collection is copied to the result collection and objects matching `object_spec` are added to or removed from the new collection. The base collection can be the empty collection.

object_spec

Lists named objects or collections to add or remove. The object class of each element in this list must be the same as the base collection. If the name matches an existing collection, the collection is used. Otherwise, PrimeTime searches for the objects in the database using the object class of the base collection.

`-unique`

The `-unique` option for the `add_to_collection` command eliminates duplicate objects from the resulting collection.

When the *collection* argument to the `add_to_collection` command is the empty collection, some special rules apply to the *object_spec* argument. If the *object_spec* argument is not empty, at least one homogeneous collection must exist in the *object_spec* list. The first homogeneous collection in the *object_spec* list becomes the base collection and sets the object class for the function.

Examples

This command sets the `ports` variable for collection of all ports except for `CLOCK`:

```
pt_shell> set ports [remove_from_collection \
            [get_ports "*"] CLOCK]
```

This command results in a collection of all cells in the design except those in the top level of hierarchy:

```
pt_shell> set lcells [remove_from_collection \
            [get_cells * -hier] [get_cells *]]
```

You can add objects to a collection with the `add_to_collection` command:

```
pt_shell> set isos [add_to_collection [get_cells i*]\
            [get_cells o*]]
```

Most collections are homogeneous since commands, such as the `get_ports` or `get_cells` command, create homogeneous collections. Some commands, such as the `all_connected` command, create heterogeneous collections. You can create heterogeneous collections using the `add_to_collection` command. For example,

```
pt_shell> set a [get_ports PH*]
{"PH1", "PH2"}
pt_shell> set b [get_pins -regexp {reg(0|1)/CP}]
{"reg0/CP", "reg1/CP"}
pt_shell> query_objects -verbose [add_to_collection $a $b]
{"port:PH1", "port:PH2", "pin:reg0/CP", "pin:reg1/CP"}
```

Sorting Collections

You can sort a collection using the `sort_collection` command. PrimeTime sorts according to the list of attributes you specify. Sorts are ascending by default. You can reverse the order using the `-descending` option. In an ascending sort, PrimeTime first sorts Boolean attributes with the objects that have the attribute set to `false`, then sorts the objects that have the attribute set to `true`. In the case of a sparse attribute, objects that do not have the attribute follow the objects that have the attribute. Not all collections can be sorted, such as timing paths, timing arcs, or library timing arcs. For example, enter the following command to sort the ports by direction, then by full name.

```
pt_shell> sort_collection [get_ports *] \
            {direction full_name}
{"in1", "in2", "out1", "out2"}
```

For example, to get a collection of hierarchical cells decreasing alphabetically, followed by leaf cells decreasing alphabetically, sort the collection of cells using the *is_hierarchical* and *full_name* attributes. In the following example, `i1` and `i2` are hierarchical, and `u1` and `u2` are leaf cells.

```
pt_shell> set c [get_cells {u2 i2 u1 i1}]
{"u2", "i2", "u1", "i1"}
pt_shell> set cs [sort_collection -descending $c \
            {is_hierarchical full_name}]
{"i2", "il", "u2", "u1"}
```

The *is_hierarchical* attribute is Boolean, so in a descending sort, cells that have the attribute set to `true` (`i1` and `i2`) are first. To resolve the equality for the *is_hierarchical* attribute on the `i1` and `i2` cells, use the *full_name* attribute.

Using Collection Utility Commands

PrimeTime provides these additional commands for manipulating collections:

- `sizeof_collection`
- `compare_collections`
- `copy_collection`
- `index_collection`

Note:
Some collections, such as timing paths, cannot be indexed, copied, or compared.

Determining the Size of a Collection

The `sizeof_collection` command returns the number of elements in a collection. You can use the empty string (the empty collection) as an argument. `sizeof_collection ""` is always 0.

Examples

To gauge the size of a design, enter

```
pt_shell> sizeof_collection [get_cells * -hier]
```

To determine whether a collection command yielded results, enter

```
pt_shell> if {[sizeof_collection [get_cells U2/U2]] != 0}
```

Comparing Collections

The `compare_collections` command compares the contents of two collections, object for object. You can specify that PrimeTime compare the objects in order. If the objects in both collections are the same, the result is 0 (like the result of string compare). If the objects are different, the result is nonzero.

Copying Collections

The `copy_collection` command duplicates a collection, resulting in a new collection. The base collection remains unchanged.

The `copy_collection` command is an efficient mechanism for duplicating an existing collection. However, copying a collection and having multiple references to the same collection are significantly different. Not all collections can be copied, such as timing paths, timing arcs, or library timing arcs.

Examples

If you create a collection and save a reference to it in the `c1` variable, assigning the value of `c1` to another `c2` variable creates a second reference to the same collection:

```
pt_shell> set c1 [get_cells "U1*"]
{"U1", "U10"}
pt_shell> set c2 $c1
{"U1", "U10"}
pt_shell> echo $c1 $c2
_sel3 _sel3
```

The previous commands do not copy the collection; only the `copy_collection` command creates a new collection that is a duplicate of the original.

This command sequence shows the result of copying a collection:

```
pt_shell> set collection1 [get_cells "U1*"]
{"U1", "U10"}
pt_shell> set collection2 [copy_collection $collection1]
{"U1", "U10"}
pt_shell> compare_collections $collection1 $collection2
0
```

Indexing Collections

The `index_collection` command creates a collection of one object that is the *n*th object in another collection. Objects in a collection are numbered 0 through *n*-1. Not all collections can be indexed, such as timing paths, timing arcs, or library timing arcs.

Although collections that result from commands, such as `get_cells` command, are not really ordered, each has a predictable, repeatable order: The same command executed *n* times (for example, `get_cells *`) creates a collection of cells in the same order.

The following example shows how to extract the first object in a collection:

```
pt_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
pt_shell> index_collection $c1 0
{"u1"}
```

Using Lists

Lists are an important part of Tcl; they are used to represent groups of objects. Tcl list elements can consist of strings or other lists. Table 14-8 shows the Tcl commands you can use with lists.

Table 14-8 Tcl Commands to Use With Lists

Command	Task
`concat`	Concatenates two lists and returns a new list.
`join`	Joins elements of a list into a string.
`lappend`	Creates a new list by appending elements to a list (modifies the original list).
`lindex`	Returns a specific element from a list; this command returns the indexed element if it is there, or an empty string if it is not there.
`linsert`	Creates a new list by inserting elements into a list (it does not otherwise modify the list).
`list`	Returns a list formed from its arguments.
`llength`	Returns the number of elements in a list.
`lrange`	Extracts elements from a list.
`lreplace`	Replaces a specified range of elements in a list.
`lsearch`	Searches a list for a regular expression.
`lsort`	Sorts a list.
`split`	Splits a string into a list.

Most publications about Tcl contain extensive descriptions of lists and the commands that operate on lists. Note these facts about Tcl:

- This is the preferred method of entering a list:

  ```
  [list a b c d]
  ```

  The entries `"a b c d"`, `{a b c d}`, and `[list a b c d]` might all work equally well. However, the preferred method is more reliable.

- Special characters pose some challenges for lists. A list of two bused port names might look like this:

  ```
  {{a[0]} {a[0]}}
  ```

Note:
Do not use commas to separate list items.

Example

If you are attempting to perform command or variable substitution, the form with braces does not work. For example, given that variable `a` is set to 5, these commands yield different results:

```
pt_shell> set a 5
5

pt_shell> set b {c d $a [list $a z]}
c d $a [list $a z]

pt_shell> set b [list c d $a [list $a z]]
c d 5 {5 z}
```

- You can nest lists, as shown in the previous example.
- Any list is also a well-formed Tcl command.
- You can use the `concat` command or other Tcl commands to concatenate lists.

Using Other Utilities

Tcl contains several other commands that handle the following:

- Strings and regular expressions (commands such as `format`, `regexp`, `regsub`, `scan`, and `string`).
- File operations (commands such as `file`, `open`, and `close`).

- Launching system subprocesses (commands such as `exec`).
- The `date` command returns the current date and time. It does so without creating a subprocess. This should be used instead of the `exec date` command.

Quoting Values

You can quote values in these ways:

- Escape individual special characters with the backslash (`\`) so that they are interpreted literally. Note that the backslash character itself is difficult to use from the user interface, so its use in netlist objects is discouraged. (The number of times a backslash needs to be escaped depends on whether the option to which it is passed is a string or a list.)
- Enclose a set of words separated by spaces within braces (`{}`). This syntax is called rigid quoting because variable, command, and backslash substitutions do not occur.
- Enclose a set of words separated by spaces within double quotation marks (`""`). This syntax is called weak quoting because variable, command, and backslash substitutions can still occur.

The PrimeTime shell allows you to use `blk[0]` without quoting even though brackets (`[]`) are part of the language. Although pt_shell can help in some cases, rigidly quoting these strings, as in `{blk[0]}`, is always better.

In the following examples, the value of variable `a` is 5. Both commands are valid but have different results:

```
pt_shell> set s "temp = data[$a]"
temp = data[5]

pt_shell> set s {temp = data[$a]}
temp = data[$a]
```

Nesting Commands

You can nest commands within other commands (also known as command substitution) by enclosing the nested command within brackets (`[]`). Tcl imposes a depth limit of 1000 for command nesting.

PrimeTime makes one exception to the use of command nesting with square brackets so it can recognize netlist objects with bus references. PrimeTime accepts a string, such as `data[63]`, as a name rather than as the word *data* followed by the result of command `63`. Without this extension, you would have to rigidly quote `data[63]`, as `{data[63]}`, or escape the square brackets, as in `data\[63\]`.

Here is an example of how to nest a command:

```
pt_shell> set title "Gone with the Wind"
Gone with the Wind
```

Here is another example of nesting a command:

```
pt_shell> set lc_title [string tolower $title]
gone with the wind
```

Evaluating Expressions

Tcl supports expressions. However, arithmetic operators are not included as part of the base Tcl language syntax. Instead, use the `expr` command to evaluate expressions. The `expr` command can also evaluate logical and relational expressions.

This construct is correct:

```
set a [expr (12*$p)]
```

This construct is incorrect:

```
set a (12 * $p);
```

Flow Control

Flow control commands—`if`, `while`, `for`, `foreach`, `break`, `continue`, and `switch`—determine the execution order of other commands. You can use any pt_shell command in a control flow command, and you can include other control flow commands.

Using the if Command

An `if` command has two or more arguments:

- An expression to evaluate.
- A script to conditionally execute based on the result of the expression.

You can extend the `if` command to contain an unlimited number of `elseif` clauses and one `else` clause.

- An `elseif` argument to the `if` command requires two additional arguments: an expression and a script.
- An `else` argument requires only a script.

This example shows the correct way to specify `elseif` and `else` clauses. Notice that the `else` and `elseif` clauses appear on the same line as the closing brace (`}`). This syntax is required because a new line indicates a new command. If the `elseif` clause is on a separate line, it is treated as a command, which it is not. The `elseif` clause is an argument to the `if` command.

```
if {$x == 0} {
  echo "Equal"
} elseif {$x > 0} {
  echo "Greater"
} else {
  echo "Less"
}
```

Note:
If you are comparing strings, use `string compare $x $y` rather than `$x==$y`.

Using while Loops

The `while` command is similar to the same construct in the C programming language. The `while` command has two arguments:

- An expression
- A script

This example shows the `while` command can print squared values from 0 to 10:

```
set p 0
while {$p <= 10} {
  echo "$p squared is: [expr $p * $p]"; incr p
}
```

Using for Loops

The `for` command is similar to the same construct in the C programming language. The `for` command has the following arguments:

- Initialization script
- Loop termination expression
- Iterator script
- Actual working script

This example shows how to rewrite the `while` loop shown earlier as a `for` loop.

```
for {set p  0} {$p <= 10} {incr p} {
  echo "$p squared is: [expr $p * $p]"
}
```

Iterating Over a List: foreach

The `foreach` command is similar to the same construct in the C shell. This command iterates over the elements in a list and contains the following arguments:

- Iterator variable
- List
- Script to execute (the script references the iterator's variable)

You cannot use `foreach` to iterate over a PrimeTime collection. Attempting this destroys the collection. For information about iterating over a collection, see "Iterating Over the Elements of a Collection" on page 14-26.

Example 1

This statement prints an array:

```
foreach el [lsort [array names a]] {
   echo "a\($el\) = $a($el)"
}
```

Example 2

This statement searches in the search_path for several files, then reports whether the files are directories:

```
foreach f [which {t1 t2 t3}] {
  echo -n "File $f is "
  if { [file isdirectory $f] == 0 } {
    echo -n "NOT "
  }
  echo "a directory"
}
```

Terminating a Loop

The `break` and `continue` commands terminate a loop before the termination condition is reached, as follows:

- The `break` command causes the innermost loop to terminate.
- The `continue` command causes the current iteration of the innermost loop to terminate.

Using the switch Command

The `switch` command is equivalent to an `if` tree that compares a variable to a number of values. One of a number of scripts is executed, based on the value of the variable:

```
switch $x {
a {incr t1}
b {incr t2}
c {incr t3}
}
```

The `switch` command has other forms and several complex options. For more examples of the `switch` command, see books about the Tcl language.

Using Procedures

To extend the Tcl command language, you can write reusable Tcl procedures. You can write new commands that can use an unlimited number of arguments, and the arguments can contain default values. You can use a varying number of arguments.

This procedure prints the contents of an array:

```
proc array_print {arr} {
  upvar $arr a
  foreach el [lsort [array names a]] {
    echo "$arr\($el\) = $a($el)"
  }
}
```

Keep in mind the following points about procedures:

- Procedures can use any supported PrimeTime command or other procedure you define.
- Procedures can be recursive.
- Procedures can contain local variables and reference variables outside their scope.

- Arguments to procedures can be passed by value or by reference.
- PrimeTime provides extensions to Tcl procedures that allow you to define the syntax and semantics of arguments. With these features, you can write extensions to PrimeTime that look like commands, and you can easily parse the arguments to your procedure. For more information, see the section called “Extending Procedures” in the “Tcl Advanced Topics” chapter of the *PrimeTime Advanced Timing Analysis User Guide*.

Using the proc Command to Create Procedures

You can use the `proc` command to create a new Tcl procedure. You cannot create a procedure using the name of an existing built-in or application command. However, if a procedure with the name you specify exists, the new procedure usually replaces the existing procedure. When you invoke the new command, PrimeTime executes the contents of the body.

The `proc` command returns an empty string. When a procedure is invoked, the return value is the value specified in a return command. If the procedure does not execute an explicit return, the return value is the value of the last command executed in the body of the procedure. If an error occurs while the body of the procedure executes, the procedure returns that error.

When the procedure is invoked, a local variable is created for each formal argument to the procedure. The value of the local variable is the value of the corresponding argument in the invoking command or the default value of the argument. Arguments with default values are not specified in a procedure invocation. However, there must be enough actual arguments for all the formal arguments that do not have defaults.

Except for one special case, no extra arguments can exist. The special case permits procedures with variable numbers of arguments. If the last formal argument has the name `args`, a call to the procedure can contain more actual arguments than the procedure has formal arguments. In this case, all actual arguments, starting at the one assigned to `args`, are combined into a list (as if you had used the `list` command). The combined value is assigned to the local variable `args`.

The arguments to Tcl procedures are named positional arguments. You can program positional arguments with default values; you can include optional arguments by using the special argument name `args`.

When the body is executed, variable names typically refer to local variables, which are created automatically when referenced and deleted as the procedure returns. One local variable is automatically created for each argument of the procedure. You can only access global variables by invoking the `global` or the `upvar` command.

PrimeTime allows you to program extended information for a procedure and its arguments. For more information, see the *PrimeTime Advanced Timing Analysis User Guide* and also the man page.

Programming Default Values for Arguments

To set up a default value for an argument, make sure the argument is located in a sublist that contains two elements: the name of the argument and the default value. This procedure reads a favorite library by default, but reads a specific library if its name is given:

```
proc read_lib {{lname favorite.db}} {
  read_db $lname
}
```

Specifying a Varying Number of Arguments

You can specify a varying number of arguments by using the `args` argument. You can specify that at least some arguments must be passed into a procedure; you can then deal with the remaining arguments as necessary.

Example 1

To report the square of at least one number, use

```
proc squares {num args} {
  set nlist $num
  append nlist ""
  append nlist $args
  foreach n $nlist {
    echo "Square of $n is [expr $n*$n]"
  }
}
```

Example 2

This procedure adds two numbers and returns the sum:

```
pt_shell> proc plus {a b} { return [expr $a + $b]}
pt_shell> plus 5 6
11
```

For more information about writing procedures, see books about the Tcl language.

Displaying the Body of a Procedure

The `info` command with the body argument displays the body (contents) of a procedure. The `proc_body` command is a synonym for the `info body` command.

If you define the procedure with the `hide_body` attribute, you cannot use the `info body` command to view the contents of the procedure. The *proc_name* argument is the name of the procedure in the following syntax:

```
info body proc_name
proc_body proc_name
```

The following example shows the output of the `info body` command for a procedure named plus:

```
pt_shell> proc plus {a b} {return [expr $a + $b]}

pt_shell> info body plus
return [expr $a + $b]
```

Displaying the Formal Parameters of a Procedure

The `info` command with the `args` argument displays the names of the formal parameters of a procedure. The *proc_name* argument is the name of the procedure in the following syntax:

```
info args proc_name
```

The following example shows the output of the `info args` command for the plus procedure:

```
pt_shell> proc plus {a b} { return [expr $a + $b] }

pt_shell> info args plus
a b
```

Glossary

active edge
The low-to-high or high-to-low transition of a clock signal that causes data to be latched into a flip-flop.

aggressor net
A net that causes undesirable crosstalk effects on another net (called the victim net) due to parasitic cross-capacitance between the two nets.

annotation
A piece of information attached to an object in the design, such as a delay value attached to a net; or the process of attaching such a piece of information to an object in the design.

advanced OCV
Advanced on-chip variation (OCV) is a Synopsys technology that reduces unnecessary pessimism by using the location and logic depth of each path analyzed.

arc
A set of timing constraints associated with a particular path.

Arcadia
A Synopsys tool that extracts RC (resistance and capacitance) information from a chip layout database.

ASCII
A standard form of plain, unformatted text (stands for American Standard Code for Information Exchange).

ASIC
Application-specific integrated circuit; a fabricated electronic device designed to perform a specific task.

ASIC vendor
A company that manufactures integrated circuits designed by others, using standard circuit elements having defined functional and timing characteristics.

asynchronous
The lack of a timing relationship between two different clocks or signals (transitions can occur at any time with respect to each other).

async_default path group
The group of paths that end on asynchronous preset/clear inputs of flip-flops.

attribute
A string or value associated with an object in the design that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can find out the values of attributes by using the `get_attribute` command.

back-annotation
The process of applying detailed parasitic data to a design, allowing a more accurate timing analysis of the final circuit. The data comes from an external tool after completion of layout.

borrowing
An analysis technique in which a portion of a path is allowed to borrow timing margin from the next stage of the design, when the next stage is a level-sensitive latch operating in transparent mode.

capture
The processes of clocking data into a latch or flip-flop at the endpoint of a path, thus getting the data that was launched by an earlier clock edge at the startpoint of the path.

cell
A component within an integrated circuit with known functional and timing characteristics, which can be used as an element in building a larger circuit.

cell delay
The delay from the input to the output of a cell (logic gate) in a path.

clock
A periodic signal that latches data into sequential elements. You define a signal to be a clock and specify its timing characteristics by using the `create_clock` or `create_generated_clock` command.

clock path
A timing path that starts at a clock input port or a pin of an internal cell and ends at a clock input pin of a sequential element.

clock gating
The control of a clock signal by a combinational logic element such as a multiplexer or AND gate, to either shut down the clock at selected times or to modify the clock pulse characteristics.

clock_gating_default path group
The group of paths that end on combinational elements used for clock gating.

clock latency
See latency.

clock reconvergence pessimism
An inaccuracy of static timing analysis that occurs when two different clock paths partially share a common physical path segment, and the analysis tool uses the minimum delay of the shared segment for one path and the maximum delay of the same segment for the other path. Correction of this inaccuracy is called clock reconvergence pessimism removal (CRPR).

clock skew
See skew.

closing edge
The edge of a clock signal that latches data into a level-sensitive latch, ending the transparent mode and desensitizing the data input.

collection
A set of objects taken from the loaded design. For example, the `set mylist [get_clocks "CLK*"]` command creates a collection of clocks and sets a variable called “mylist” to the collection. Then you can use commands that operate on the collection, such as `remove_clock $mylist`. You can also generate and operate on a collection within a single command, such as `remove_clock [get_clocks "CLK*"]`.

combinational feedback loop
A combinational logic network in which a signal path makes a complete loop back to its starting point. PrimeTime must break a loop (stop tracing the signal) at some point within the loop.

combinational logic
A logic network that only contains elements that have no memory or internal state. Combinational logic can contain AND, OR, XOR, and inverter elements, but cannot contain flip-flops, latches, registers, or RAM.

common point
In a path where clock reconvergence pessimism exists, the common point is the last cell output in the path segment shared between the launch and capture clock paths.

conditional timing arc
A timing arc whose delay values depend on some condition such as the logical state of an input.

conservative
An analysis algorithm or technique that favors pessimism (rather than optimism) to ensure detection of all violations.

constraint
A timing specification or requirement that applies to a path or a design. A timing constraint limits the allowed range of time that signals can arrive at a device input or be valid at a device output.

context characterization
The process of using the `characterize_context` command to capture the timing context of a subdesign in the chip-level timing environment. This process allows stand-alone timing analysis of the subdesign in PrimeTime or optimization of the subdesign in Design Compiler. The timing context of an instance includes clock information, input arrival times, output delay times, timing exceptions, design rules, propagated constants, wire load models, input drives, and capacitive loads.

context independent
Accurate in different contexts. Note that this term refers to the usability, not behavior, of a timing model. For example, a timing model created with the `extract_model` command is context independent, which means that the model can be used accurately in different contexts (its behavior changes according to the context in which it is used).

CPU time
The amount of time used by the central processing unit of the computer to accomplish a task, as reported by the software. This is a measure of computation resources required for the task. CPU time is less than the actual elapsed time.

critical path
The path in a path group that has the largest violation (or the least timing slack) of all paths in that path group.

crosstalk
An undesirable effect on a net caused by parasitic capacitance between the net and physically adjacent nets. Signal transitions on the adjacent nets can cause noise bump or a change in the transition time on the affected net.

current design
The loaded design currently considered to be the top-level design in a design hierarchy. You can specify the current design with the `current_design` command.

current instance
The instance (hierarchical cell) in a design hierarchy on which instance-specific commands operate by default. You can set the current instance with the `current_instance` command, which works like the cd command in UNIX.

data check
A timing check between two data signals, such as handshaking interface signals or recovery/removal checks between preset and clear pins. The two signals being checked are called the constrained and related signals. The related signal is the data signal treated as the clock in the timing relationship. For example, a setup data check verifies that the constrained signal is stable before the active edge of the related signal.

data path
A timing path that starts at a data launch point and ends at a data capture point. The term “path” usually means a data path, although there are other types of paths such as clock paths and asynch_default paths.

.db (database) format
A Synopsys file format used for storing designs, .lib technology libraries, clock information, back-annotated parasitic information, and timing models. You can read some or all of this information from a .db file into PrimeTime by using the `read_db` command. Tools such as Design Compiler and Formality also use .db files.

default path group
The set of constrained timing paths that do not fall into any of the other implicit categories (for example, a path that ends on an output port). The implicit path categories are paths that end on clocked sequential elements, clock_gating_default paths, and async_default paths.

Design Compiler
The Synopsys core tool that can synthesize a design defined by a hardware description language into an optimized, technology-dependent, gate-level design. Design Compiler supports a wide range of flat and hierarchical design styles and can optimize both combinational and sequential designs for speed, area, and power. Design Compiler and PrimeTime work well together, but they are independent tools that can be used separately.

distributed multicore analysis
A flow that enables you to use multiple cores to improve performance by distributing the timing update and optimizing handling in other major flow areas.

edge-sensitive latch
A flip-flop; a register element that captures data on each active edge on its clock input. The active edge is a transition from low to high or high to low, depending on the device type.

endpoint
The place in a design where a timing path ends. This is where data gets captured by a clock edge or where the data must be available at a specific time. Every endpoint must be either a register data input pin or an output port.

exception
See timing exception.

exception transformation
The conversion of the current set of exception-setting commands into a new set of commands by the `transform_exceptions` command. The command operates by eliminating invalid, redundant, and overridden paths in the original command set.

exclusive clocks
Two clocks that are never active at the same time (for example, because they are multiplexed).

extracted timing model
A timing model created from a gate-level netlist by the `extract_model` command. This type of model represents the timing characteristics (arc values) of the block. However, it does not contain any functional information, so it cannot be synthesized.

false path
A path that exists in a design that should not be analyzed for timing, such as a path between two multiplexed blocks that are never enabled at the same time. For proper analysis by PrimeTime, you need to define false paths as timing exceptions with the `set_false_path` command or remove the related objects from analysis with the `set_disable_timing` command.

fanin
The set ports and pins that affect a specified timing endpoint (called the sink). A pin is considered to be in the timing fanin of a sink if there is a timing path through combinational logic from the pin to that sink. Fanin tracing stops at the clock pins of registers (sequential cells).

For more information, see the man pages for the `all_fanin` and `report_transitive_fanin`. commands.

fanout
In PrimeTime, the term fanout usually means timing fanout, also known a transitive fanout. Timing fanout is the set ports and pins at timing endpoints that are affected by a specified source port or pin. A pin is considered to be in the timing fanout of a source if there is a timing path through combinational logic from the source to that pin. Fanout tracing stops at the inputs to registers (sequential cells).

Note that timing fanout is not the same as direct fanout. Direct fanout is the number of cell inputs connected to a cell output.

For more information, see the man pages for the `all_fanout` and `report_transitive_fanout` commands.

flip-flop
A memory element controlled by an edge-sensitive clock input. Typically, a flip-flop has an input D, and output Q, a clock input, and possibly asynchronous set and clear inputs. When the active edge occurs on the clock input, the value on the input D is latched and then held constant at the output Q. The active edge can be either rising or falling, depending on the type of flip-flop.

gated clock
A clock signal that can be modified by logic within the design, such as a clock that can be turned off to save power.

generated clock
A clock signal that is generated internally by the integrated circuit itself; a clock that does not come directly from an external source. An example of a generated clock is a divide-by-2 clock generated from the system clock, having half the frequency of the system clock. You specify the characteristics of a generated clock by using the `create_generated_clock` command.

glitch
A noise bump that is large enough to cause a logic failure.

GUI
Graphical user interface; the PrimeTime graphical windows that offer menus, dialog boxes, buttons, and graphical presentation of design information and analysis results.

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

I-V characteristics (steady-state)
The current-voltage characteristics of a cell output; the current as a function of voltage while the output is held constant at logic one or logic zero. PrimeTime SI uses this information to calculate crosstalk noise effects.

island
See voltage island.

inout pin
A bidirectional pin of a cell; a pin that can operate as both an input and an output.

input delay
A constraint that specifies the minimum or maximum amount of delay from a clock edge to the arrival of a signal at a specified input port. PrimeTime uses this information to check for timing violations at the input port and in the transitive fanout from that input port.

intellectual property
Information owned by one company that is licensed for use by another company. For example, one company might offer a license to use a chip submodule design (such as a microprocessor core) in another company's application-specific circuit. The owner of the submodule design can provide the layout information, electrical specifications, and a timing model to the other company, without revealing the submodule netlist.

interface logic model
A timing model created from a gate-level netlist by the `write_ilm_netlist` command and related commands. This type of model represents the timing characteristics (arc values) of the block and includes the interface logic of the original netlist. However, it does not include any functional information from the inside of the original block, so it cannot be synthesized.

latch
A memory element controlled by level-sensitive gate input. The output of a latch follows the input as long as the gate signal is active. When the gate signal goes from active to inactive, the input value is latched and the output remains constant at that value. The inactive-to-active edge of the gate signal is called the opening edge. The active-to-inactive edge of the gate signal is called the closing edge.

latency
The amount of time that a clock signal takes to be propagated from the clock source to a specific point inside the design. The clock signal is “hidden” (latent) for this amount of time.

launch
The processes of clocking data out of a latch or flip-flop at the startpoint of a path, releasing data that is captured by another clock edge at the endpoint of the path.

layout
The process of generating the geometric location, size, and form of components and connections for an integrated circuit. From layout information, an external tool can generate accurate information about the parasitic resistance and capacitance of the components and connections. You can then back-annotate the design with this information in PrimeTime for a more accurate timing analysis.

leaf level
The level of design hierarchy containing the lowest-level library cells, like the leaves of a tree.

level-sensitive latch
A register that allows data to pass through from input to output while its gate input is active, and holds its data output constant when the gate input is inactive.

library
A database containing information about the functional and timing characteristics of circuit elements used in a design, also called the technology library or .lib technology library when specified in the syntax of Library Compiler. Technology libraries are typically provided by the ASIC vendor.

Library Compiler
A Synopsys tool used to create library databases for PrimeTime, Design Compiler, and other CAE tools. Library Compiler reads the description of an ASIC library from a text file (.lib file) and compiles the description into a database in .db format for synthesis and analysis, or into a set of VHDL libraries for VHDL simulation.

man page
A description of a command, variable, or message code displayed by using the `man` command in PrimeTime. For example, entering `man create_clock` at the pt_shell prompt displays the man page describing the `create_clock` command. This is similar to the UNIX man command (which is derived from the word “manual”).

multicore analysis
A flow that enables you to use multiple cores to improve performance by threading or distributing the timing update and optimizing handling in other major flow areas. This flow includes the distributed multicore analysis and threaded multicore analysis flows.

multicycle path
A path that is designed to take more than one clock cycle for the data to propagate from the startpoint to the endpoint. For example, a multiplier circuit (a slow combinational logic element) might be designed to take four clock cycles to generate a valid result; the circuit is designed to capture the result after this amount of time has elapsed. For proper analysis, you need to define such a path as a timing exception with the `set_multicycle_path` command.

native
A command or process in PrimeTime that works entirely in PrimeTime itself and does not require an outside program.

net arc
A timing arc from a driver pin to a load pin connected to the same net, having a nonzero delay due to parasitic net capacitance.

net delay
The amount of delay from the output of a cell to the input of the next cell in a timing path. This delay is the result of parasitic capacitance of the interconnection between the two cells.

network latency
The amount of time a clock signal takes to propagate from the clock definition point in the design to a register clock pin.

NLDM
Nonlinear Delay Model, a standard, table-based format for specifying cell delays in .lib technology libraries.

noise
A temporary deviation or change in the analog voltage of a steady-state net, such as a crosstalk-induced voltage bump at the output of a CMOS gate.

noise bump
An occurrence of noise that appears as hump shape (often modeled as a triangle) when plotted as voltage versus time.

noise immunity curve
A function that specifies the maximum size of a noise bump that can be allowed at the input of a cell without generating propagated noise at the output of the cell. The function is often approximated as a hyperbola.

noise margin
DC noise margin is the difference between the most extreme output voltage for a logic level (such as VOLmax) and the most extreme input threshold considered the same logic level (such as VILmax). AC noise margin is the maximum size of a noise bump, in volts, that can be allowed at the input of a cell without generating a logic failure at the output of the cell.

noise slack
The amount by which a logic failure is avoided when a noise bump occurs at the input of a cell, equal to the failure threshold voltage minus the bump height, multiplied by the bump width. The units are in library voltage units times library time units.

no-change constraint
The amount of time that a data signal must remain unchanged before, during, and after a pulse of a control signal (for example, an address signal that must be stable during a write pulse). The data signal must be stable for a specified setup time before the leading edge of the control pulse, during the control pulse, and for a specified hold time after the trailing edge of the control pulse.

opening edge
The edge of a clock signal that asserts a gate input of a level-sensitive latch. The latch is transparent from the opening edge to the closing edge of the clock signal.

operating conditions
The process, voltage, and temperature conditions under which a circuit operates, which affect the timing characteristics of the cells used in the design.

optimistic
An analysis algorithm or technique with a known accuracy limitation, when the inaccuracy might indicate that a circuit has more timing slack than the real circuit. As a result, a violation in the real circuit might not be detected.

output delay
A constraint that specifies the minimum or maximum amount of delay from an output port to the external sequential device that captures data from that output port. This constraint establishes the times at which signals must be available at the output port in order to meet the setup and hold requirements of the external sequential element.

parasitic capacitance
Capacitance of a net due to the physical proximity between the net interconnections and adjacent nets or the substrate; or due to charge storage effects of P-N junctions in the net.

parasitic resistance
Resistance of an interconnection or net due to the finite conductivity of the interconnect material.

path
A point-to-point sequence through a design that starts at a register clock pin or an input port, passes through any number of combinational logic elements, and ends at a register data input pin or an output port. Data launched at the path startpoint by a clock edge is propagated through the combinational logic to the path endpoint, where the data gets captured by another clock edge.

path endpoint
See endpoint.

path group
A group of related paths, grouped either implicitly by the `create_clock` command or explicitly by the `group_path` command. By default, paths whose endpoints are clocked by the same clock are assigned to the same path group. Path groups affect the reporting of violations in PrimeTime. They also affect the relative amount of effort used by Design Compiler to optimize different paths.

path inspector
A window in the PrimeTime GUI used to display detailed information about a timing path such as the path schematic, path element tables, timing waveforms, path delay profiles, and text-format reports.

path startpoint
See startpoint.

pessimistic
An analysis algorithm or technique with a known accuracy limitation, when the inaccuracy might indicate that a circuit has less timing slack than the real circuit. As a result, a violation might be reported that would not actually exist in the real circuit.

pin
An input or output of a cell instance used in a design. The timing constraints for pins are specified in the .lib technology library.

port
An input or output of a design. You specify timing constraints for ports with the `set_input_delay` and `set_output_delay` commands.

post-layout
After layout; the time at which detailed parasitic data becomes available for back-annotation on the design.

PrimeTime
The Synopsys stand-alone, full-chip, gate-level static timing analysis tool that breaks down a design into a set of timing paths, calculates the delay of each path, and checks to see whether all timing constraints are met. The timing analysis is done without functional simulation, so no test vectors are required.

PrimeTime PX
A PrimeTime add-on feature that accurately analyzes full-chip power dissipation of cell-based designs. It provides vector-free and vector-based peak power and average power analysis.

PrimeTime SI
A PrimeTime add-on feature that adds crosstalk analysis capabilities to PrimeTime. PrimeTime SI calculates the timing effects of cross-coupled capacitors between nets and includes the resulting delay changes in the PrimeTime analysis reports.

PrimeTime VX
A PrimeTime add-on feature that increases the accuracy of timing analysis by considering the statistical variation and distribution of process parameters.

procedure (Tcl)
A user-defined command created by the `proc` command in PrimeTime. A procedure, once defined, can be executed like a standard command. A procedure can take arguments and can use local and externally defined variables.

propagated noise
A noise bump on a net caused by noise on the input of the gate that is driving the net. For example, for an inverter whose input is zero and output is one, a positive bump on the input can result in a negative bump on the output.

pt_shell
The text-based, command-line form of PrimeTime that lets you enter commands, run scripts, and view results in text form. By contrast, the GUI (graphical user interface) form of PrimeTime also provides pull-down menus, dialog boxes, command buttons, and graphical presentation of analysis results.

quick timing model
A temporary timing model you create with PrimeTime commands for a block when there is no netlist and no other type of timing model available. You use `create_qtm_model` to create a new model, commands like `create_qtm_port` and `create_qtm_delay_arc` to specify the model characteristics, and `save_qtm_model` to save the completed model.

RC
Resistor-Capacitor, a simple parasitic model that consists of a single resistor and capacitor to represent all of the net parasitics.

reconvergence pessimism
See clock reconvergence pessimism.

recovery constraint
The minimum amount of time required between an asynchronous control signal (such as the asynchronous clear input of a flip-flop) going inactive and a subsequent active clock edge. This is like a setup check for the active clock edge. If the active clock edge occurs too soon after the asynchronous input goes inactive, the register data is uncertain because the clocked input data might not be valid.

register
A leaf-level instance of a flip-flop (controlled by an edge-sensitive clock input) or latch (controlled by a level-sensitive gate input).

related signal
The data signal treated as a clock in a data-to-data timing check. The other signal being checked is called the constrained signal.

removal constraint
The minimum amount of time required between a clock edge that occurs while an asynchronous input is active and the subsequent removal of the asserted asynchronous control signal. This is like a hold check for the removal of the

asynchronous control signal. If the asynchronous input signal goes inactive too soon after a clock edge, the register data is uncertain because the clocked input data might be valid and conflict with the asynchronous control signal.

RSPF
Reduced Standard Parasitic Format, a format used to store circuit parasitic information for back-annotation. This format is defined by Cadence Design Systems, Inc.

SBPF
Synopsys Binary Parasitic Format, a format used to store circuit parasitic information for back-annotation. Parasitic data in this format occupies less disk space and can be read much faster than the same data stored in SPEF format.

script
A text file containing a sequence of pt_shell commands, which can be executed with the `source` command.

SDF
Standard Delay Format, a standard file format used to store circuit information for back-annotation, including delay information (such as pin-to-pin cell delays and net delays) and timing checks (such as setup, hold, recovery, and removal times). PrimeTime can read and write SDF files with the `read_sdf` and `write_sdf` commands.

sensitize
To find or apply an input vector that causes a particular path to be logically active and to propagate data.

sequential logic
A logic network that contains elements that have memory or internal state, such as flip-flops, latches, registers, or RAM.

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

signal integrity
The immunity of a signal or net against crosstalk effects; the characteristic of a net that is not affected by transitions or noise on physically adjacent nets.

skew
The amount of shift or variation away from the nominal, expected time that a clock transition occurs; or the difference in the amount of shift between two different points in a clock network.

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

slew
The amount of time it takes for a signal to change from low to high or from high to low; also known as transition time.

source latency
The amount of time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design.

SPDM
Scalable polynomial delay model, a standard, polynomial-based format for specifying cell delays in .lib technology libraries.

SPEF
Standard Parasitic Exchange Format, a format used to store circuit parasitic information for back-annotation. This format is defined by Open Verilog International (OVI).

SPICE
Simulation program with integrated circuit emphasis, a time-based simulation tool that analyzes circuit operation at the level of transistors, capacitors, and resistors. Many different forms of this simulator are available from different sources.

startpoint
The place in a design where a timing path starts. This is where data gets launched by a clock edge or where the data is expected to be available at a specific time. Every startpoint must be either a register clock pin or an input port.

static timing analysis
An efficient analysis technique that determines whether a circuit violates any timing requirements by considering path delays and timing constraints. Compared with gate-level simulation, static timing analysis is faster and more thorough, and does not require test vectors. However, it is not a replacement for simulation because it does not check the functionality of the circuit.

static noise analysis
An analysis technique used by PrimeTime SI that determines worst-case noise effects in the design so that those effects can be minimized or eliminated. This technique considers the cross-coupling capacitance between physically adjacent aggressor and victim nets and the steady-state (one or zero) load characteristics of the victim net.

synchronous
A timing relationship between two clocks or signals in which specific transitions occur at the same time or have a phase difference that stays fixed over time.

Tcl
Tool command language, a standard command scripting language on which the pt_shell interface is based.

Tcl procedure
See procedure (Tcl).

technology library
See library.

three-state
An output of a device (such as a bus driver) that can be in any of three logical states: low (0), high (1), or the high-impedance (Z) state.

time borrowing
See borrowing.

timing arc
See arc.

timing exception
An exception to the default (single-cycle) timing behavior assumed by PrimeTime. For PrimeTime to correctly analyze a circuit, you must specify each place in the design that does not conform to the default behavior. Examples of timing exceptions include false paths, multicycle paths, and paths that require a specific minimum or maximum delay time different from the default calculated time.

timing fanin/fanout
See fanin or fanout.

timing model
A circuit model that contains the timing characteristics of the block, but not necessarily any functional information. These models are often used in hierarchical timing analysis, when analyzing a very large chip design would be too time-consuming if done as a single, flat design.

transforming exceptions
See exception transformation.

transition time
The amount of time it takes for a signal to change from low to high or from high to low; also known as slew.

transitive fanin/fanout
See fanin or fanout.

transparent
The state of a level-sensitive latch while the gate input is asserted. During this time, signals pass through from input to output and the device operates just like a buffer.

true path
A path that is logically possible to operate in the design; a path that can be sensitized and can propagate data.

uncertainty (clock uncertainty)
The amount of variation away from the nominal, ideal-clock times at which clock edges occur.

vector
A specific set of values applied to the inputs of a device for testing or analysis purposes, or the resulting set of values at the outputs of a device; or a sequence of such values used for sensitizing a path in the design for analysis purposes.

victim net
A net that has undesirable crosstalk effects due to parasitic cross-capacitance between it and another net (called the aggressor net).

voltage island
A physical portion of a chip that uses a different power supply voltage from other portions of the chip.

wire load model
A net resistance and capacitance model used for timing analysis before layout, in the absence of back-annotated delay values or parasitic information. PrimeTime estimates the wire load based on the fanout of each net and library-specified parameters.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W