PrimeTime®
Advanced Timing Analysis
User Guide

Version C-2009.06, June 2009

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, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC 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

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 Suite, then select a release in the list that appears at the bottom.

About This User Guide

The *PrimeTime Advanced Timing Analysis User Guide* describes advanced topics related to performing static timing analysis using PrimeTime. It supplements the basic information provided in the *PrimeTime Fundamentals User Guide*.

Audience

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

Related Publications

For additional information about PrimeTime, see Documentation on the Web, which is available through 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 SI, PrimeTime PX, and PrimeTime VX
- Design Compiler
- Library Compiler
- Library NCX

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as `pin1 [pin2 ... pinN]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
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 an electronic 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 on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at 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 or in the footer.

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 call to your local support center from the Web by going to https://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- 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 the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Overview of Advanced Topics

Prior to consulting this user guide, you should familiarize yourself with the *PrimeTime Fundamentals User Guide*, which covers basic topics, such as getting started, design data, clocks, analysis conditions, and timing reports. This user guide covers more advanced topics, such as tool command language (Tcl) features, using Design Compiler, context characterization, back-annotation, and attributes.

This chapter describes the content in each of the other chapters of this manual:

Topics Covered in This Manual

This manual covers a variety of topics for advanced users of PrimeTime. You should consider the *PrimeTime Fundamentals User Guide* a prerequisite for using this manual.

Chapter 2, “Tcl Advanced Features,” describes advanced features of the Tcl scripting language not covered in the *PrimeTime Fundamentals User Guide*, including Tcl packages and autoload, regular expressions (using `-regexp` to perform pattern matching), extending procedures, and the TclPro Toolkit.

Chapter 3, “Using PrimeTime With Other Synopsys Tools,” explains how to use PrimeTime effectively with Design Compiler. It explains the differences between PrimeTime and Design Compiler with respect to command syntax and timing analysis behavior. It also explains how to use Transcript, a tool that translates Design Compiler timing assertion scripts to PrimeTime scripts.

Chapter 4, “Context Characterization,” describes how to use the `characterize_context` command in PrimeTime to capture the timing context of subdesigns in the chip-level timing environment. This information can be used to set the timing constraints of a subdesign during synthesis or logic optimization in Design Compiler, or to perform timing analysis hierarchically in PrimeTime.

Chapter 5, “Advanced Analysis Techniques,” describes a variety of advanced analysis topics, including true and false path detection, time borrowing, asynchronous logic analysis, three-state bus analysis, parallel driver reduction, and data-to-data checking.

Chapter 6, “Advanced On-Chip Variation,” describes the advanced on-chip variation (OCV) technology. It provides information about the graph-based and path-based solution. It explains the flow and shows the reporting aspects for this technology.

Chapter 7, “Ideal Network Support,” describes the creation of ideal networks, on which no design rule constraints (DRCs) are run. This allows designers ignore networks (large unoptimized networks with high fanout and capacitance) in prelayout and, instead, focus on violations arising from other sources. The use of ideal networks reduces runtime because PrimeTime uses 'ideal timing' rather than internally calculated timing.

Chapter 8, “SDF Back-Annotation,” explains how to back-annotate a design with delay information contained in a Standard Delay Format (SDF) file, and how to write an SDF file from the PrimeTime database. Back-annotating delay information from a layout tool produces a more accurate timing analysis than estimating delays from wire load models.

Chapter 9, “Parasitic Back-Annotation,” explains how to back-annotate a design with detailed parasitic information (parasitic capacitors and resistors) for more accurate timing analysis. It explains the types of parasitic data accepted and the procedures for back-annotating, reporting, and removing parasitic data. It also explains the PrimeTime algorithm for automatically completing parasitic information that is incomplete.

Chapter 10, “Delay Calculation With Detailed Parasitics,” explains how PrimeTime calculates delays in the presence of detailed parasitics. Note that you do not need this information to use parasitic back-annotation. It is only provided in case you want to check the accuracy of the delay calculations performed by PrimeTime.

Chapter 11, “Low-Power Flow Support,” describes how PrimeTime supports using IEEE 1801™ (UPF) to specify low-power features of a design, such as multiple power supplies and power gating. It explains how PrimeTime reads UPF scripts and provides examples demonstrating the use of UPF commands and infrastructure.

Chapter 12, “Object Attributes,” describes the object attributes contained in the PrimeTime design database (for example, the `number_of_pins` attribute attached to each cell) and how to gather attribute information for generating a custom report.

Appendix A, “Writing Mapped SDF Files,” provides detailed information about controlling the format of the SDF file produced by the `write_sdf` command. It is a supplement to Chapter 7, “SDF Back-Annotation.”

2

Tcl Advanced Features

The PrimeTime command interface is based on the Tcl command language. Basic principles of Tcl usage are presented in the *PrimeTime Fundamentals User Guide*.

This chapter explains some advanced features of Tcl in the following sections:

Tcl Packages and Autoload

PrimeTime supports the standard Tcl package and autoload facilities. However, the load command is not supported, so packages that require shared libraries cannot be used. PrimeTime is shipped with the standard implementations of the packages that are part of the Tcl distribution. For reference, you can find these packages below the root of the Synopsys installation in the auxx/tcllib/lib/tcl8.4 directory.

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

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

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

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

```
package require mycompanyPT
```

For detailed information about Tcl packages, see the `package` command man page, Tcl reference books, or visit the following Web site:

http://tcl.activestate.com/doc/

Regular Expressions

Many commands that create explicit collections allow you to choose between simple wildcard patterns (using the `*` and `?` wildcard characters) and complete regular expressions. Such commands provide the `-regexp` option, which you can use to tell the collection engine to use the pattern matching you want. For example,

```
pt_shell> set gc [get_cells -regexp {i(1|2)_.*}]
```

Using `-regexp` causes PrimeTime to view the patterns argument as a regular expression. In addition, the pattern matching operators in the filter expression (`=~` and `!~`) also use regular expressions. The following two commands, which illustrate basic regular expressions, are equivalent:

```
pt_shell> get_cells blk* -filter "ref_name =~ AN*"

pt_shell> get_cells -regexp {blk.*} -filter "ref_name =~ AN.*"
```

To perform a case-insensitive search in PrimeTime, the `-regexp` and `-nocase` options must be combined.

PrimeTime uses the same regular expression engine as Tcl. For details about supported regular expression language and the `regexp` command, see the man page for `regexp` or `re_syntax`, or consult books on the subject in the engineering section of your local bookstore or library.

Using Regular Expressions With Implicit Collections

Commands that generate implicit collections do not directly support regular expressions. However, because an element of an implicit collection can be an explicit collection, you can use regular expressions indirectly. For example,

```
pt_shell> report_timing -from [list i1/* [get_pins -regexp $expr]]
```

Using Regular Expressions With Hierarchy

Mixing `-regexp` and `-hierarchical` options is different from mixing a wildcard pattern with `-hierarchical` searches. Hierarchical searches with regular expressions are always compared to the full name of the objects in question. The rules for hierarchical searches with regular expressions are as follows:

- Using `-regexp` alone matches leaf names in the current instance. For example,

  ```
  pt_shell> get_cells -regexp i1.*
  ```

- Using `-regexp` with `-hierarchical` matches full names, relative to the current instance, for each object found at or below the current instance. This is independent of the existence of hierarchy separators in the pattern. For example, to create a collection of `i1/i2/n1`, `i1/i21/n1`, `i1/i2/i3/n1`, and so forth, use the following syntax:

  ```
  pt_shell> get_cells -regexp i1/i2.*/n1
  ```

- Using `-regexp` does not provide a direct method to match leaf names at each level of the hierarchy. However, you can emulate a method of matching leaf names by using filters. For example, to use `-regexp` to do the same as the command `get_cells n1 -hierarchical`, use the `base_name` attribute, which is the leaf name of the cell. For example, enter

  ```
  pt_shell> get_cells -regexp -hierarchical ".*" \
            -filter {base_name == n1}
  ```

Anchoring Regular Expressions

By default, PrimeTime automatically anchors regular expression patterns. For example, the pattern `blk.*` is considered the same as `^blk.*$`, which is usually the intended behavior.

If you want a less restricted matching style, like the Tcl `regexp` command, prefix and suffix the pattern with `.*` to unanchor the pattern (that is, not do an exact match). For example, to get any cells that contain `U1`:

```
pt_shell> get_cells -regexp {.*U1.*}
```

This matches `U1`, `U11`, `U1A`, `U1_23`, plus `ZU1`, `ZZU1`, `hello_U1`, and so forth.

Using Regular Expressions With Buses

When using a regular expression to match buses, you must be very careful because the bus characters ([ ]) are part of the command language. In addition, the usage varies slightly depending on whether the command argument is a string or a list.

For a command argument which is a string, the following example shows the correct form. The expression argument to `filter_collection` is a string.

```
pt_shell> filter_collection -regexp [get_ports *] {full_name =~

a\[[0-1]\]}
```

This regular expression matches ports a[0] and a[1]. A single backslash (`\`) must precede the real square brackets.

For a command argument which is a list, the syntax depends on how you specify the list. Consider the following examples, which use the `get_ports` command. The “patterns” argument to `get_ports` is a list.

```
pt_shell> get_ports -regexp [list {a\[[0-1]\]}]
pt_shell> get_ports -regexp {{a\[[0-1]\]}}
```

These two commands are essentially equivalent. Proper list forms require single backslash quoting (\), just like string arguments. It is recommended that you use a properly formatted list for a list argument, especially in this situation. However, when you pass a single string into the “patterns” argument, double backslash quoting (`\\`) is required. For example,

```
pt_shell> get_ports -regexp {a\\[[0-1]\\]}
```

The double backslash is required because the promotion of the string to a list consumes one of the backslashes.

Extending Procedures

Tcl enables you to write reusable procedures. Basic Tcl procedures have positional arguments, allow default values for arguments, and allow a varying number of arguments. PrimeTime provides a set of extensions, which allows you or a process developer to define help text and semantic information for the arguments of a procedure. This is called defining procedure attributes.

The procedure attributes that you can define include

- A one-line informational text string that is shown with simple help
- The command group in which the procedure exists
- Whether the procedure can be modified
- Whether the body of the procedure can be viewed
- Help text and semantic rules for each argument of the procedure
- Whether the procedure can be abbreviated

The `define_proc_attributes` command defines attributes for a procedure. Use this command to modify the attributes of an existing procedure. After you have defined procedure attributes, you can parse the procedure arguments to your procedure with the `parse_proc_arguments` command.

Default Procedure Attributes

When you create a procedure by using the `proc` command,

- It is placed in the Procedures command group
- It has no help text for its arguments
- On invocation, you can abbreviate the procedure name, subject to the value of the `sh_command_abbrev_mode` variable
- View the body of the procedure using `info body`
- Possible to modify the procedure and its attributes

Changing the Aspects of a Procedure

You can use the `define_proc_attributes` command to change the aspects of a procedure listed previously.

The `define_proc_attributes` command enables you to:

- Define the help text and semantic rules for each argument.
- Make the help for a procedure look like the help for an application command.

Format for the -define_args Option

The value for the `-define_args` option is a list of lists. Each element has this format:

`arg_name option_help value_help` [*`data_type`*] [*`attributes`*]

`arg_name`

The name of the argument.

`option_help`

A short description of the argument.

`value_help`

Typically, the argument name for a positional argument or a one-word description for the value of a command-line option. The value has no meaning for a Boolean option.

`data_type`

Optional; any of `string` (the default), `list`, `boolean`, `int`, `float`, or `one_of_string`. Option validation can use the data type.

`attributes`

Optional; a list of attributes that can be any of the following.

Value	Description
`required`	The default. If you use `required`, you cannot use `optional`.
`optional`	If you use `optional`, you cannot use `required`.
`value_help`	When argument help is shown for the procedure, this attribute also shows the valid values for a `one_of_string` argument. For data types other than `one_of_string`, this is ignored.
`values`	List of allowable values for a `one_of_string` argument. This is required if the argument type is `one_of_string`. If you use values with other data types, an error appears.

Value	Description
merge_duplicates	When an option appears more than once in a command, the values are concatenated into a list of values. The parse_proc_arguments receives a single entry, which is a list. When merge_duplicates is not specified, the right most value for the option is used.
script	Identifies the argument as a Tcl script. This is used in the TclPro flow, so that tools like snps_checker will recurse into these arguments. For more information, see "TclPro Toolkit" on page 2-9.

There are two reasons for using the define_parse_attributes command: consistent formatted help and syntax and semantic checking. To complete syntax and semantic checking, you must use the parse_proc_arguments command. If you do not use parse_proc_arguments, the procedure cannot respond to -help. However, you can always use the help *procedure_name* -verbose command.

The following procedure adds two numbers and returns the sum. For demonstration purposes, some unused arguments are defined.

```
pt_shell> proc plus {a b} { return [expr $a + $b]}
pt_shell> define_proc_attributes plus -info "Add two numbers" \
            -define_args { \
            {a "first addend" a string required} \
            {b "second addend" b string required} \
            {"-verbose" "issue a message" "" boolean optional}}
pt_shell> help -verbose plus
Usage: plus    # Add two numbers
    [-verbose]          (issue a message)
    a                   (first addend)
    b                   (second addend)
pt_shell> plus 5 6
11
```

Parsing Arguments Passed in to a Tcl Procedure

The parse_proc_arguments command parses the arguments passed to a Tcl procedure that was defined with the define_proc_attributes command. The parse_proc_arguments command works only if you use one procedure argument: the Tcl keyword args, meaning any number of arguments. Then, you pass the arguments to the parse_proc_arguments, which does the semantic checking for you.

You can use the parse_proc_arguments command to parse arguments passed into a Tcl procedure. The value passed into the Tcl procedure is automatically converted to a valid one-of-string value option. It is no longer the original user-supplied string. Using the

parse_proc_arguments command within Tcl procedures enables support for argument validation and support of the -help option. Typically, parse_proc_arguments is the first command called within a procedure. You cannot use the parse_proc_arguments command outside of a procedure. For syntax information, see the man page.

When a procedure that uses the parse_proc_arguments command is invoked with the -help option, parse_proc_arguments prints help information (in the same style as using the help -verbose command) and then causes the calling procedure to return. Similarly, if any type of error exists with the arguments (missing required arguments, invalid value, and so forth), the parse_proc_arguments command returns a Tcl error and the calling procedure terminates and returns.

If you do not specify -help and the specified arguments are valid, the array variable result_array contains each of the argument values that you entered, subscripted with the argument name. The argument names are not the names of the arguments in the procedure definition; the argument names are the names of the arguments as defined using the define_proc_attributes command.

The following procedure shows how the parse_proc_arguments command typically is used. The procedure argHandler accepts an optional argument of each type supported by the define_proc_attributes command, then prints the options and values received.

```
proc argHandler {args} {
  parse_proc_arguments -args $args results
  foreach argname [array names results] {
    echo "  $argname = $results($argname)"
  }
}

define_proc_attributes argHandler \
  -info "Arguments processor' \
  -define_args {
  {{-Oos "oos help" AnOos one_of_string
      {required value_help {values {a b}}}}
   {-Int "int help" AnInt int optional}
   {-Float "float help" AFloat float optional}
   {-Bool "bool help" "" boolean optional}
   {-String "string help" AString string optional}
   {-List "list help" AList list optional}}
   {-IDup "int dup" AIDup int {optional merge_duplicates}}}
```

Invoking argHandler with -help generates the following output:

```
pt_shell> argHandler -help
Usage: argHandler    # argument processor
      -Oos AnOos            (oos help: Values: a, b)
      [-Int AnInt]          (int help)
      [-Float AFloat]       (float help)
      [-Bool]               (bool help)
```

```
      [-String AString]     (string help)
      [-List AList]         (list help)
      [-IDup AIDup]         (Int dup)
```

Invoking `argHandler` with an invalid option generates the following output and causes a Tcl error:

```
pt_shell> argHandler -Int z
Error: value 'z' for option '-Int' not of type 'integer'
(CMD-009)
Error: Required argument ‘-Oos’ was not found (CMD-007)
```

Invoking `argHandler` with valid arguments generates the following output:

```
pt_shell> argHandler -Int 6 -Oos a -IDup 2 -IDup 3
  -Int = 6
  -Oos = a
  -IDup = 2 3
```

Note:

Only the arguments entered are shown in the output of `argHandler`. To test to see if a specific argument was entered, you would do the following in your procedure:

```
if {[info exists results(-Bool)]}{
   ...
}
```

TclPro Toolkit

TclPro is an open-source toolkit for Tcl programming. These are some of the tools that come with TclPro:

- TclPro Checker (procheck)
  Static script syntax checker
- TclPro Compiler (procomp)
  Stand-alone bytecode compiler
- TclPro Debugger (prodebug)
  Tcl debugger

The Synopsys Syntax Checker (snps_checker), based on TclPro Checker, is included with the PrimeTime distribution. It is not necessary to download the TclPro tools to access syntax checking.

This section discusses how you can leverage these tools in the Synopsys environment. For more information about the TclPro tools, see the following Web address:

http://tcl.sourceforge.net

Installation Requirements

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

Checking the Syntax and Semantics of Your Scripts

The Synopsys Syntax Checker, based on the TclPro Checker, helps you find syntax and semantic errors in your Tcl scripts. Everything that you need for syntax and semantic checking is shipped with PrimeTime. In this environment, the following items are checked:

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

Limitations Using the Synopsys Syntax Checker

The following limitations apply to the Synopsys Syntax Checker:

- Command abbreviation is not checked. Use of abbreviated commands will show up as undefined procedures.
- Aliases created with the `alias` command are not expanded and will show up as undefined procedures.
- A few checks done when the application is running might not be checked using the snps_checker environment. For example, some cases where one option requires another option are not checked.

- Script size is an issue with snps_checker and TclPro 1.3 and 1.5. Scripts up to a few thousand lines can be reasonably checked, but beyond that, CPU time becomes a factor. Do not try to check extremely large scripts using snps_checker.
- PrimeTime allows you to specify Verilog-style bus names on the command line without rigid quoting, for instance, A[0]. This format with indices from 0 to 255 is checked. Wildcards * and % are also checked. Other forms, including ranges as A[15:0] would show as undefined procedures, unless represented as {A[15:0]}.
- User-defined procedures enhanced with the `define_proc_attribute` command are not checked. Because such procedures are declared with the `args` argument, no semantic errors are reported.

Running the Synopsys Syntax Checker

There are two ways to run snps_checker in the Synopsys environment. You can launch snps_checker from PrimeTime, or you can run it stand-alone.

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

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

```
package require snpsTclPro
```

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

The following example shows the script that is used in snps_checker:

```
create_clock [get_port CLK] -p
create_clock [get_ports CLK] -p -12.2

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

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

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

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

```
report_timing -through $x -thr $y -from a -from b -to z > r.out
                                          ^
/disk/scripts/ex1.tcl:11 (SnpsE-Excl) Can only specify one
of
these options for all_fanout: -from -clock_tree
all_fanout -from X -clock_tree
^
/disk/scripts/ex1.tcl:12 (numArgs) wrong # args
pwd foo
^
```

Bytecode-Compiled Files

Bytecode-compiled files are created using the TclPro Compiler, procomp. These files have the following advantages over ASCII scripts:

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

PrimeTime can load bytecode-compiled scripts. To load a bytecode-compiled file, you source it like you do other scripts. You do not need any files other than the application to load bytecode-compiled files.

Debugging Scripts

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

Launching Prodebug From PrimeTime

Running the TclPro debugger is dependent on the `SNPS_TCLPRO_HOME` environment variable. Once this variable is set, you can debug scripts using the `debug_script` command. For more information about the `SNPS_TCLPRO_HOME` variable, see “Installation Requirements” on page 2-10.

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

```
package require snpsTclPro
```

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

TclPro Limitations

TclPro is not supported on RS/6000. However, bytecode-compiled files can be loaded into this application on all platforms, including RS/6000; and snps_checker is supported on all UNIX platforms.

3

Using PrimeTime With Other Synopsys Tools

PrimeTime works well with other Synopsys tools. This chapter describes the differences between tools and how to use PrimeTime with other Synopsys tools, such as Design Compiler.

This chapter contains the following sections:

Using PrimeTime With Design Compiler

PrimeTime works very well with the Design Compiler synthesis tool. There are, however, some differences in command syntax and behavior between the two tools. The following sections describe how to make the two tools work together efficiently:

- Features Specific to PrimeTime
- Timing Analysis Differences
- Command Scripts
- Reading .db Files With Back-Annotated Data
- Path Groups in Design Compiler

Features Specific to PrimeTime

Some features of PrimeTime are not supported by Design Compiler, such as internal clocks, certain features of extracted timing models, and mode analysis (timing models with operating modes).

Design Compiler does not support the creation of clocks on internal pins of a leaf cell. You must specify these clocks at input or output pins that are contained in the fanin or fanout of the internal pin.

Design Compiler cannot analyze and optimize designs with timing models that use mode analysis. You must manually disable the inactive timing arcs to analyze and optimize a module in a particular operating mode.

Timing Analysis Differences

The differences in timing analysis behavior between PrimeTime and Design Compiler are described in the following sections.

Clocks

The following differences apply to clocks.

timing_input_port_default_clock Variable

This Boolean variable affects the behavior of PrimeTime when timing a path from an input port with no clock reference input delay. When this variable is `true` (the default), each such input port is given one input port clock so that the inputs are constrained. This also causes the clocks along the paths driven by these input ports to become related. When this variable is set to `false`, no such imaginary clock is assumed.

Design Compiler does not have this variable. It behaves like PrimeTime with the variable set to `false`.

timing_input_port_clock_shift_one_cycle Variable

This Boolean variable affects the behavior of PrimeTime when timing a path from an input port with no clocked input external delay and the variable for the input port default clock (`timing_input_port_default_clock`) is set to `true`.

When you set the `timing_input_port_clock_shift_one_cycle` variable to `true`, paths starting at such input ports are given one extra cycle (like `set_multicycle_path 2`) to meet timing constraints at clocked destination registers or output ports. When this variable is `false` (the default), no extra multicycle shift is applied.

Design Compiler does not have this variable. It behaves like PrimeTime with the variable set to `true`.

Paths

The following differences apply to paths.

Time Borrowing for Hold Checks

PrimeTime disables short path borrowing by default. (Short path borrowing is time borrowing for hold checks at level-sensitive latches.) Design Compiler always allows borrowing for short paths.

To simulate the Design Compiler behavior, set the `timing_allow_short_path_borrowing` variable to `true`. For more information about path borrowing, see the *Synopsys Timing Constraints and Optimization User Guide*.

Segmenting Paths: Load-Dependent Delays

In Design Compiler, if a timing report originates from a segmented path's new startpoint, Design Compiler transfers the load-dependent portion of the source gate's delay to the output net delay. This method can provide a better synthesis result, but it is not appropriate for static timing analysis. PrimeTime does not add the load-dependent delay to the net.

Transition Time

The following differences apply to transition time.

Bidirectional Pins

PrimeTime maintains separate transition time information for the driver and load parts of bidirectional inout pins.

Design Compiler keeps the maximum of driver and load transition times.

Clock Pins

PrimeTime propagates transition times to clock pins and uses this information for computation of clock-to-Q delay and transition time, and calculation of setup and hold delay.

Design Compiler uses 0.0 for computation of clock-to-Q transition time and uses propagated transition for calculation of clock-to-Q delay for setup and hold delay.

PrimeTime is more pessimistic than Design Compiler for setup. The best solution for pre-layout is the `set_clock_transition` command, which specifies a fixed transition time on clock pins. Both PrimeTime and Design Compiler use 0.0 as input transition time on D of level-sensitive latches when calculating transition time for D-to-Q arcs.

current_design Command

Design Compiler requires that you change the current design for locally specifying information such as wire load models:

```
dc_shell> current_design BOTTOM
dc_shell> set_wire_load_model -name small
dc_shell> current_design MID
dc_shell> set_wire_load_model -name medium
dc_shell> current_design TOP
dc_shell> set_wire_load_model -name large
```

In PrimeTime, you achieve this by using the `current_instance` command or by specifying hierarchical cell names. Do not use `current_design` for this purpose. For example,

```
pt_shell> current_design TOP
pt_shell> set_wire_load_model -name large
pt_shell> set_wire_load_model -name medium [get_cells U1]
pt_shell> set_wire_load_model -name small [get_cells U1/U2]
```

Command Scripts

You can use command scripts in PrimeTime. A command script is a sequence of pt_shell commands in a text file. The file can be in ASCII, gzip, or bytecode format. The `source` command executes scripts in PrimeTime.

Sharing Design Compiler and PrimeTime Scripts

You can share scripts when you use both Design Compiler and PrimeTime on the same design. The easiest way to do so is to use SDC, as described in “Synopsys Design Constraints Formatted Script Files” on page 3-6. Another method is to keep scripts that contain shared commands separate from scripts that contain tool-specific commands, as demonstrated in the following examples.

Note:

You cannot convert dc_shell Tcl scripts to pt_shell scripts, but you can source some dc_shell Tcl scripts directly into pt_shell. An alternative method is to write shared constraint files in Tcl using the Synopsys Design Constraints (SDC) format. For more information, see “Synopsys Design Constraints Formatted Script Files” on page 3-6.

Example

This dc_shell script includes another script called timing_assertions.scr:

```
current_design MID
set_wire_load_model -name medium
current_design TOP
set_wire_load_mode enclosed
set_wire_load_model -name large
set_max_area 3000
set_dont_touch u13
include timing_assertions.scr
compile
```

The following script is part of the timing_assertions.scr script, which contains commands that both Design Compiler and PrimeTime understand:

```
set_operating_conditions WCCOM
create_clock -period 10 CLK
set_input_delay ...
set_output_delay ...
```

To use the Design Compiler script in PrimeTime, use the timing assertions script and a script that contains PrimeTime commands. Some PrimeTime commands are identical in syntax as Design Compiler commands. However, as mentioned earlier, commands that are tool-specific should be specified differently.

Example

In this example, the pt_shell script contains commands that are equivalent to the commands in the dc_shell script shown earlier:

```
set_wire_load_mode enclosed
set_wire_load_model -name medium {u2 u4}
set_wire_load_model -name large
source timing_assertions.scr
set_max_area 3000
```

```
set_dont_touch u13
```

Synopsys Design Constraints Formatted Script Files

SDC formatted script files are Tcl scripts that use a subset of the commands supported by PrimeTime and Design Compiler. There are limitations placed on some of the supported commands. For example, not all options are supported in some cases. Although SDC is a subset of Tcl, it is not a full-function scripting language. It is intended to support the specification of timing constraints across various EDA tools. It is not intended for complex scripting.

Within SDC, there is only one design: the current design. You cannot change the current design with SDC.

SDC files are read into PrimeTime with the `read_sdc` command, and are written from PrimeTime using the `write_sdc` command. For more information about these commands, see the man pages.

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

Reading .db Files With Back-Annotated Data

You can improve the performance of reading netlist .db format files with back-annotation information into PrimeTime by using one of the following procedures.

Procedure 1

1. In dc_shell, save the .db format file with no back-annotation information and write the back-annotation information using SDF.
2. In pt_shell, read the .db format file (which is not back-annotated) and the generated SDF. This is faster than reading the back-annotated .db format file directly in PrimeTime.

Procedure 2

1. In dc_shell, save the timing assertions set on the design with the `write_script` command. Make sure you remove commands that are not timing related.

2. In pt_shell, read the back-annotated .db format file, but use the `-netlist_only` option of the PrimeTime `read_db` command to instruct PrimeTime to ignore the back-annotation.
3. Apply the timing assertions to the design by sourcing the generated script. Make sure the script is converted from a Design Compiler script to a PrimeTime script.

Path Groups in Design Compiler

In Design Compiler, you can use path groups to modify the cost function for optimization. You can assign paths or endpoints to named groups. Each group has a weighting value you can specify, which contributes to the maximum delay cost for the design.

PrimeTime checks each timing path in the design for maximum delay violations: it compares the actual path delay for rising and falling signals to a target path delay. The worst violation for a group is the path with the largest negative slack.

You can specify path groups in PrimeTime. When you request a path-timing report, PrimeTime lists a report for each path group. You can export this information to Design Compiler using the `write_script` or `write_context` commands or by budgeting the design.

When you create an explicit path group, you can also assign a critical range for the path group. The critical range defines a margin of delay for the path group during optimization in Design Compiler. A nonzero value allows the optimization of near-critical paths if the worst violation is not increased.

ECO Flow with PrimeTime, IC Compiler, and Star-RCXT

You can use PrimeTime along with IC Compiler and Star-RCXT in an ECO flow. This section describes possible ECO flows. It includes the following sections:

- Introduction to the PrimeTime ECO Flow
- Design and Library Requirements
- Recommended Usage Flow
- IC Compiler ECO Flow Examples
- Generating Compatible ECO Output for IC Compiler With the write_changes Command
- Clearing Netlist Changes
- ECO Parasitics

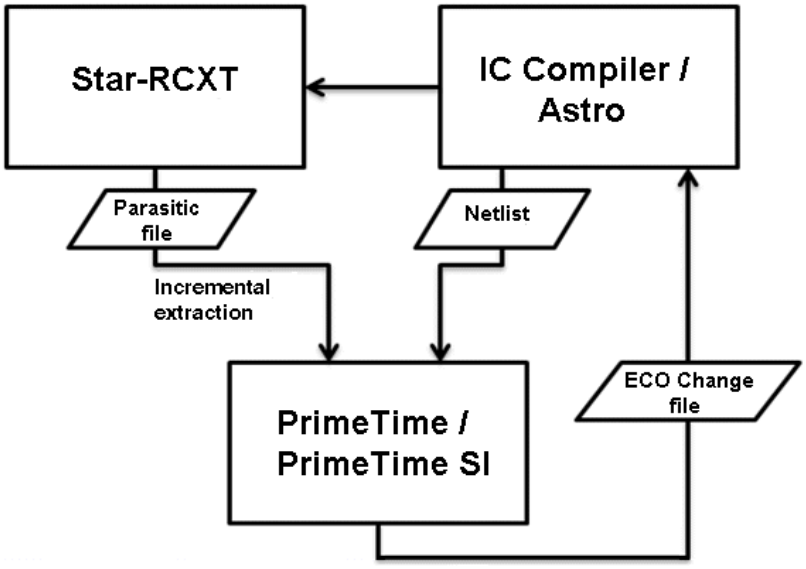
- Writing Astro Change Files

Introduction to the PrimeTime ECO Flow

In design flows, sign-off static timing analysis is performed after physical design implementation. It is common to observe timing violations during the initial phases of static timing analysis before physical implementation is finalized. At this stage, it might be beneficial to fix as many timing violations as possible across all static timing analysis scenarios. One approach is to perform the manual engineering change order (ECO) process on a cell-by-cell basis, using the what-if analysis capabilities in PrimeTime. However, this approach can result in long runtime due to several iterations of timing updates. To improve the turnaround time of early timing ECOs, an automated ECO fixing flow is desired.

PrimeTime allows for an ECO flow between PrimeTime, IC Compiler, and Star-RCXT, as shown in Figure 3-1. After performing timing and signal integrity analysis in PrimeTime and PrimeTime SI, you can perform ECO operations in PrimeTime and provide a netlist change list to IC Compiler to place and reroute the affected parts of the design. You can then give the updated layout to Star-RCXT, which extracts the incremental parasitics for the affected nets. You can then read the incremental parasitics file back into PrimeTime and PrimeTime SI to rerun the timing and signal integrity analysis and confirm the final timing.

Figure 3-1 ECO Flow



PrimeTime ECO capability is provided to help improve the turnaround time for early ECO fixing. In the early ECO phase, the design is expected to have a large number of violations and it consists of many scenarios. Based on the timing characteristics of the design,

PrimeTime ECO generates a list of netlist changes that can be imported to an implementation tool, such as IC Compiler. This capability allows you to quickly evaluate possible fixes within PrimeTime and reduce iterations in the physical implementation flow.

You can use the PrimeTime ECO fixing capabilities for setup slack-based fixing, hold slack-based fixing, and crosstalk delta delay fixing. PrimeTime ECO capabilities are available in the following analysis modes:

- PrimeTime static timing analysis
- PrimeTime SI crosstalk delay analysis
- PrimeTime VX variation-aware analysis
- Single-core analysis and distributed multi-scenario analysis
- Graph-based analysis
- Path-based analysis for hold fixing

Design and Library Requirements

As designs move from the functional to the timing closure phase, the early ECO flow can benefit from using PrimeTime sign-off analysis to provide fixing guidance to IC Compiler. Design for the early ECO flow using PrimeTime ECO should meet certain criteria. Designs should

- Be at the start of the ECO flow and they should also be routed
- Complete the clock tree design phase
- Include an IC Compiler database
- Contain STAR-RCXT setup files and .nxtgrd files for sign-off parasitic extraction

The library requirements for the PrimeTime ECO flow are that each cell type should have sufficient size for upsizing and downsizing to permit setup fixing. The library should also have sufficient buffer sizes available for insertion when hold fixing.

The PrimeTime ECO required inputs are libraries (.db and .lib), Verilog netlists, parasitic data (SPEF and SBPF), and Synopsys Design Constraint (SDC) timing constraints. PrimeTime ECO generates a change list of buffer insertion or cell upsizing and downsizing. It also supports the ASCII output format to interface with IC Compiler.

Recommended Usage Flow

Before proceeding, PrimeTime ECO ensures the requested change does not cause the following actions:

- Produce a negative impact on victim paths
- Cause a violation on the aggressor
- Introduce design rule checking violation on the driver output when sizing an aggressor driver or on the driver input when sizing a victim driver
- Degrade timing on crosstalk-related stages, such as when a victim is an aggressor to another victim

The following list of steps show the recommended PrimeTime ECO usage flow:

1. Improve setup slack-based violations in the design within PrimeTime. The following example enables setup fixing for all clock groups and violating endpoints:

   ```
   fix_eco_timing -type setup
   ```

   In setup slack-based fixing, cells are resized to meet timing requirements for violating endpoints. The expectation is that PrimeTime ECO, when used for early ECO fixing, reduces the number of timing violations without degrading worst negative slack. It looks at the solution to not only improve overall slack, but also delta delays.

   To specify the maximum area ratio increase when a cell is resized, you can set the `eco_alternative_area_ratio_threshold` variable only during setup fixing. The default value of this variable is 0 for unlimited area during cell resizing. A value of 1.2 would indicate that the resized cell could be no more than 20% larger than the current cell. You can also specify cells to exclude from ECO candidates during cell resizing by marking library cells with the `pt_dont_use` user-defined variable. To prevent certain library cell from being used during ECO, see the example script that creates this variable.

   ```
   define_user_attribute -quiet -type boolean -class lib_cell pt_dont_use
   proc set_pt_dont_use_lib_cell { lib_cell } {
     set_user_attr -quiet -class lib_cell [get_lib_cell -quiet $lib_cell]
    pt_dont_use true
   }
   ```

2. Use the `fix_eco_timing` command to fix hold slack-based violations in the design. For example, to enable hold fixing for all clock groups and ensure that slack is less than -0.1ns, use the following command:

   ```
   fix_eco_timing -type hold -slack_lesser_than 0.1 \
   -buffer_list {BUF1 BUF2 BUF3 BUF4}
   ```

PrimeTime ECO fixes hold violations by inserting a delay at pins that have negative hold slack. Buffers are inserted to introduce the delay while making sure that additional setup violations are not incurred. For hold slack-based fixing, PrimeTime ECO, when used for early ECO fixing, is expected to reduce the number of violations without degrading setup worst negative slack.

3. Use PrimeTime ECO with IC Compiler for early ECO fixing by completing the following tasks:

   a. Run PrimeTime ECO and examine the quality of results (QoR) improvements with ECO changes. You can also try more than one iteration of PrimeTime ECO or use different setting to determine the best ECO strategy.

   b. Generate the list of changes for IC Compiler, by executing the following command:

      ```
      write_changes -format icctcl
      ```

      The `write_changes` PrimeTime command creates an output file of ECO commands that have been issued in a session. There are different output formats to interface with different tools: icctcl, dctcl, ptsh, or text format. You use the `-format icctcl` option for IC Compiler.

   c. From PrimeTime, read the change list ECO file into IC Compiler.

   d. Execute the `legalize_placement` and `route_eco` IC Compiler commands.

   e. Run Star-RCXT extraction.

   f. Run PrimeTime to validate the QoR after implementation.

   g. Follow the above steps until you cannot achieve any further significant improvements.

4. Measure the QoR improvement with PrimeTime ECO by using the `report_analysis_coverage` and `report_constraints` report commands and the `report_qor` Tcl script.

IC Compiler ECO Flow Examples

This section shows examples of a typical ECO flow. The first shows an example if you were changing a netlist in PrimeTime by resizing a cell and inserting a buffer, you might follow this flow:

```
pt_shell> read_parasitics my_spef.spef \
           -keep_capacitive_coupling
pt_shell> update_timing
pt_shell> set eco_instance_name_prefix {UECO}
pt_shell> size_cell U1 bufx4
pt_shell> insert_buffer U5/Z bufx1
pt_shell> update_timing
pt_shell> report_timing
pt_shell> write_changes -format icctcl -output eco.tcl
pt_shell> save_session my_session
```

At this point, you can take your design to IC Compiler and then to Star-RCXT. After you are finished with those tools, you could return to PrimeTime to confirm the final timing and use the following commands:

```
pt_shell> restore_session my_session
pt_shell> read_parasitics -eco incr.spef -keep_capacitive_coupling
pt_shell> update_timing
pt_shell> report_timing
```

Generating Compatible ECO Output for IC Compiler With the write_changes Command

You can use the `write_changes` command to generate a compatible ECO output for IC Compiler. The following sections provide more information.

Specifying New Cell and New Net Names for insert_buffer

The `insert_buffer` command adds a buffer at one or more specified pins or ports. This command allows you to provide the names for new nets and cells added during buffer insertion. You can use a few methods to determine the new cell and net names. For example, you can use the `eco_instance_name_prefix` variable, the default naming rule, or the `-new_cell_names` and `-new_net_names` options of the `write_changes` command.

The output from the `write_changes` command in the icctcl format always specifies the new cell and net names for any `insert_buffer` operations, even if you did not explicitly specify them. This ensures an effortless interface between PrimeTime and IC Compiler, while maintaining a consistent netlist between the logical netlist in static timing analysis and the physical netlist in the implementation tools.

Controlling Library Name Prepending

The `eco_write_changes_prepend_libname_to_libcell` variable controls the naming conventions for reference library cells in the change list. The default is `true`. You should set this variable to `false` in the PrimeTime to IC Compiler flow. This ensures that the ECO file is processed correctly even if the logical library names used across implementation corners differ from those that are used in the PrimeTime analysis session.

You must set this variable before the netlist changes, as it controls how the changes are recorded in memory by PrimeTime. By default, the output Tcl file has the library cell definition with the library name. By setting this variable to `false`, the file contains the reference cell name only without the library. The following example demonstrates the output Tcl file for both settings.

Example 3-1 Output Tcl File When Variable Set to true

```
##############################################################################
# Change list, formatted for IC Compiler
##############################################################################
current_instance
insert_buffer -new_cell_names ECO_1 -new_net_names net_1[get_pins {I7/A}] lib1/buf4l
```

Example 3-2 Output Tcl File When Variable Set to false

```
##############################################################################
# Change list, formatted for IC Compiler
##############################################################################
current_instance
insert_buffer -new_cell_names ECO_1 -new_net_names net_1 [get_pins {I7/A}] buf4l
```

Clearing Netlist Changes

The `write_changes -reset` option clears any netlist change lists. When you use this option, PrimeTime no longer retains any history of previous netlist editing commands. This only clears the list of changes remembered by the `write_changes` command. It does not actually undo any of the design changes that have been made.

ECO Parasitics

The `read_parasitics -eco` option allows you to read in an incremental parasitics file from Star-RCXT. For more information about the `read_parasitics` command, see the man page.

Using the `-eco` option can shorten the overall ECO cycle time. After you have made the what-if changes based on the initial PrimeTime analysis, and these changes have been implemented in IC Compiler, you can extract only the parasitics related to the changes: for instance, buffer insertions, cell insertions, or net rerouting. This results in a reduced SPEF

or SBPF file. You can then read the ECO file into PrimeTime or PrimeTime SI using the `read_parasitics -eco` command. All the other options should be exactly the same as before, for example, `-path` or `-increment`.

If the number of changes is small, reading in the partial parasitics file triggers only an incremental timing update. This results in faster turnaround times when you are checking the feasibility of various ECO changes. After performing initial analysis in PrimeTime, you can either leave your pt_shell running or save a session and restore it later to consume ECO parasitics.

If the original parasitics were read in from many parasitics files, the ECO parasitics file can correspond to only one of those parasitics files. This is because Star-RCXT only allows ECO extraction when a full extraction has been done previously. PrimeTime tries to automatically determine to which original file the ECO parasitics file corresponds, but this is not always possible. You can use the `read_parasitics -original_file_name` option to specify the original parasitics file to which the ECO file corresponds.

Note:

You cannot use the parasitics files written by PrimeTime for future ECO flows if the source of the parasitics comes from multiple files. For example, if your design is annotated with multiple parasitics files, and you generate an ILM for it, you cannot use the resulting ILM parasitics file for ECO flows.

Writing Astro Change Files

When performing what-if analysis in PrimeTime and PrimeTime SI to investigate possible fixes for timing or system integrity problems, you can write out the netlist changes directly in native Astro formats. The `write_astro_changes` command writes out the structural netlist ECO operations and coupling separation directives. This is a more direct approach than the Verilog netlist method because it does not require importing a complete Verilog description into the design. Use the `write_astro_changes` command to generate an ECO change file.

Use the `write_astro_changes` command to write changes:

```
write_astro_changes  # Write design changes for Astro
  -format type             (Astro format to write: Values:
                            heco, scheme)
  [-dont_merge_changes]    (Don't merge together structural
                            changes)
  file_name                (Output file for design changes)
```

If the `heco` format is specified, structural changes are written in the Astro hierarchical ECO (HECO) format. IC Compiler can also read in HECO files written by PrimeTime. Structural netlist changes are any what-if operations which modify the netlist structure, such as `insert_buffer` or `size_cell`. Once written, they can be applied to a cell in Astro with the following command followed by ECO place and route operations:

```
auHECOByFile "lib" "cell" "heco_file.txt"
```

Note:

In IC Compiler, the HECO flow will become obsolete and is being replaced with the Tcl flow. In PrimeTime, you can use the `write_changes` command to generate an ECO output that is compatible for IC Compiler. For more information, see Recommended Usage Flow.

If the `scheme` format is specified, the coupling separation in the current PrimeTime SI analysis is written as scheme commands for Astro. These coupling separations are applied in a PrimeTime SI analysis using the `set_coupling_separation` and `remove_coupling_separation` commands.

Once written out as a scheme file, you can load these separation directives to apply them to the currently active cell in Astro, followed by a “Detail Route Search & Repair” operation:

```
load "separations.scm"
```

It is recommended that you use Astro W-2004.12-SP1 or later for this flow. For more information about the `write_astro_changes` command, see the man page.

4

Context Characterization

Context characterization is the process of deriving the timing context of a subdesign from its environment in the parent design. The resulting information can be used for hierarchical timing analysis in PrimeTime or for synthesis or logic optimization in Design Compiler.

This chapter is described in the following sections:

Context Characterization Overview

Context characterization captures the timing context of instances of subdesigns in the chip-level timing environment. The timing context of an instance includes clock information, input arrival times, output delay times, timing exceptions, design rules, propagated constants, wire load models, input drives, and capacitive loads.

You can use context characterization for the following purposes:

- To set the timing constraints of a subdesign during synthesis or logic optimization in Design Compiler.
- To perform timing analysis hierarchically in PrimeTime while observing chip-level timing constraints.

Setting Synthesis or Optimization Constraints

The context characterization steps for setting synthesis or optimization constraints in Design Compiler are as follows:

1. Read the top design into PrimeTime.
2. Identify the subdesigns that require timing optimization.
3. Characterize the timing context for each subdesign.
4. Generate a Design Compiler script containing the context information.
5. Read the subdesign into Design Compiler.
6. In Design Compiler, read the script generated in step 4.
7. Perform module-level optimization of the subdesigns.

Performing Subdesign Timing Analysis

The context characterization steps for performing subdesign timing analysis in PrimeTime are as follows:

1. Read the design into PrimeTime.
2. Identify the subdesigns for timing analysis.
3. Characterize the timing context for each subdesign.
4. Generate a PrimeTime script containing the timing assertions.

5. Read a subdesign into PrimeTime.
6. Read the script generated in step 4.
7. Analyze the timing of the subdesign.

Deriving the Context of a Subdesign

The `characterize_context` command derives the timing and environment context from instances of subdesigns in the timing environment. The context of an instance is defined as

- Waveforms of the clocks affecting the instance
- Input arrival times
- Output required times
- Timing exceptions
- Design rules
- Logic constants
- Case analysis
- Wire load models
- Input drives
- Capacitive loads

Use the `characterize_context` command with the `write_physical_annotations` and `write_context` commands to export timing and physical information for a block from the chip-level environment.

The `characterize_context` command does not capture delays and parasitics annotated on the internal nets of the instance being characterized. To capture this information, use the `write_physical_annotations` command without the `-boundary_nets` option. If you omit the options, PrimeTime characterizes all the information. For more information, see the `characterize_context` man page.

Example 1

To generate the timing-related context information for instance I1 and I2, enter

```
pt_shell> characterize_context -timing {I1 I2}
```

Example 2

To generate the design rule and the logic constant information from the context of instance I2, enter

```
pt_shell> characterize_context -design_rules I2 -constant_inputs
```

Clock Information

PrimeTime records all clock signals that affect the instance being characterized. Clock information includes signals that feed an input pin of the instance and signals that have sources within the instance. PrimeTime derives information about all clocks driving registers or ports that launch data signals to or receive data signals from the instance.

PrimeTime characterizes the clock waveform, clock signal latency, and the clock uncertainty information for each clock being characterized. PrimeTime does not characterize propagated skew information for clocks feeding input pins of the instance being characterized.

The `write_context` command writes characterized clock information as a series of `create_clock`, `set_clock_latency`, and `set_clock_uncertainty` commands.

Input and Output Delay Times

PrimeTime characterizes arrival times of data signals at input pins of the instance being characterized and their launching clocks. PrimeTime derives required times of data signals at output pins and their related clocks.

The `write_context` command writes input arrival times as `set_input_delay` commands and writes output required times as `set_output_delay` commands.

Point-to-Point Timing Exceptions

PrimeTime derives point-to-point timing exceptions that affect the instance being characterized. The derived exceptions include

- False paths
- Multicycle paths
- `max_delay` and `min_delay` exceptions between clocked startpoints and endpoints

The `characterize_context` command calculates arrival and required times along with port capacitances so that the characterized design has the same timing as the chip-level design.

PrimeTime writes point-to-point exceptions as a series of `set_false_path`, `set_multicycle_path`, `set_max_delay`, and `set_min_delay` commands, depending on their type.

Constant Logic Values on Inputs

The `characterize_context` command derives constant logic values that are propagated to input pins of the instance being characterized.

PrimeTime derives logic constant values as logic constants and case analysis values as case analysis. For more information about case analysis, see the chapter on that topic in the *PrimeTime Fundamentals User Guide*.

Input Drive Strength and Port Capacitance

PrimeTime derives the following input drive strength and port capacitance information:

Driver information

The drivers of input pins of the instance being characterized are characterized. If the input pin is driven by a port that has a linear drive specified, this drive is also characterized. Drive information is written as `set_drive` and `set_driving_cell` commands.

Port capacitance information

Both the pin capacitance and the wire capacitance on input and output pins are characterized. Pin capacitance is written as a `set_load` command. Wire capacitance is characterized as a fanout number that derives the number of external loads at the port. This number is used together with the wire load model to estimate the boundary net capacitance. If the wire load mode is `enclosed` or `segmented`, each pin of the instance being characterized has a wire load model specified for it, depending on the enclosing hierarchy of the boundary net to which it is connected.

The fanout number is saved as a `set_port_fanout_number` command for PrimeTime and as a `set_load -fanout_number` command for Design Compiler. The wire load model is saved as a `set_wire_load_model` command for PrimeTime and Design Compiler. In addition to the fanout number, PrimeTime sets a fixed wire capacitance on the port in certain situations to ensure that the pin-to-pin wire capacitance of the boundary net after characterization is the same as that in the top-level design before characterization. This fixed value is written as a `set_load` command.

Wire Load Models

The `characterize_context` command derives the wire load mode and model for characterized instances. PrimeTime always inherits the top-level wire load mode from subdesigns. The wire load model for subdesigns is determined as follows:

- If the top-level mode is “top,” the subdesigns inherit the top-level wire load model.
- If the top-level mode is “enclosed” or “segmented,” the subdesigns inherit the wire load model from the enclosing hierarchy of the net or net segment.

Design Rule Checks

The `characterize_context` command derives design rule checks such as `max_capacitance`, `min_capacitance`, `max_fanout`, `min_fanout`, `max_transition`, and `min_transition` from their context. It also derives `fanout_load` for output pins of the instance being characterized. Each design rule check is saved by means of the corresponding command; for example, `max_fanout` is saved as a `set_max_fanout` command.

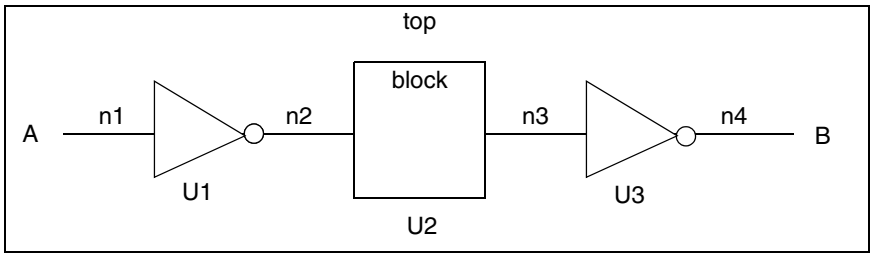
Annotated Delays and Parasitics

The `characterize_context` command does not capture delays and parasitics annotated on the internal nets of the instance being characterized. Use the `write_physical_annotations` command to capture this information.

Example 1

Figure 4-1 shows a simple combinational subdesign called block. Script 1 manually sets the port interface attributes and Script 2 starts characterization.

Figure 4-1 Simple Combinational Design



Setup Script

```
current_design top
```

```
create_clock -name clk -period 10
set_input_delay 1.0 -clock clk A
set_output_delay 2.0 -clock clk B
```

Script 1

```
current_instance block
create_clock -name clk -period 10
set_driving_cell -lib_cell INV {C}
# derive driving cell information
set_input_delay 3.3 C -clock clk
# arrival time of net n2 is 3.3
set_load 1.3 D
# load of U3 is 1.3
set_output_delay 3.5 D -clock clk
current_instance top
```

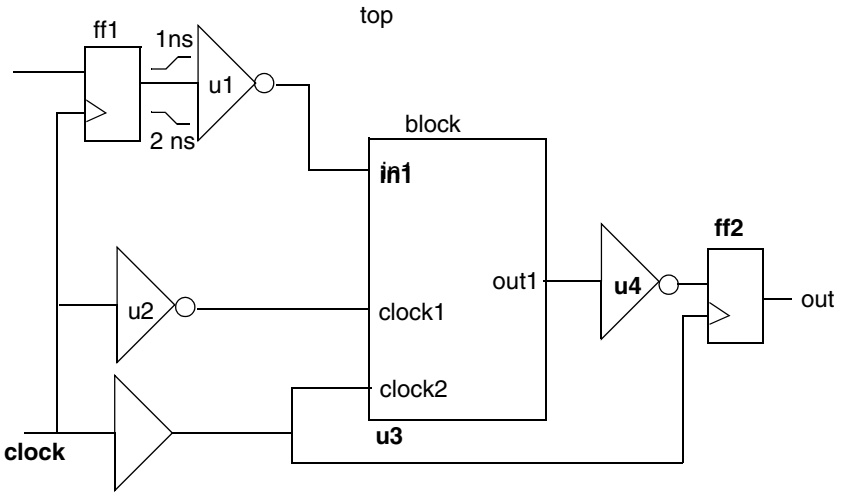
Script 2

```
current_design top
characterize_context U2
write_context -out U2.ptsh U2
```

Example 2

Figure 4-2 shows a subdesign block in the sequential design called top. Script 1 sets port interface attributes manually and Script 2 invokes characterization.

Figure 4-2 Combinational Subdesign Block in Sequential Design



Script 1

```
current_design top
create_clock -period 10 -waveform {0 5} clock
```

```
current_instance block
create_clock -name clock -period 10 -waveform {0 5} clock1
create_clock -name clock_bar -period 10 -waveform {5 10}
clock2
# delay from ff1/CP to u5/in1 is 1.8
set_input_delay -clock clock 1.8 in1
# delay of u4 plus ff2 setup time is 1.2
set_output_delay -clock clock 1.2 out1
set_driving_cell -lib_cell INV -input_transition_rise 1 in1
\
-input_transition_fall 2 in1
set_driving_cell -lib_cell CKINV clock1
set_driving_cell -lib_cell CKBUF clock2
# outside load on out1 net
set_load 0.85 out1
current_instance top
```

Script 2

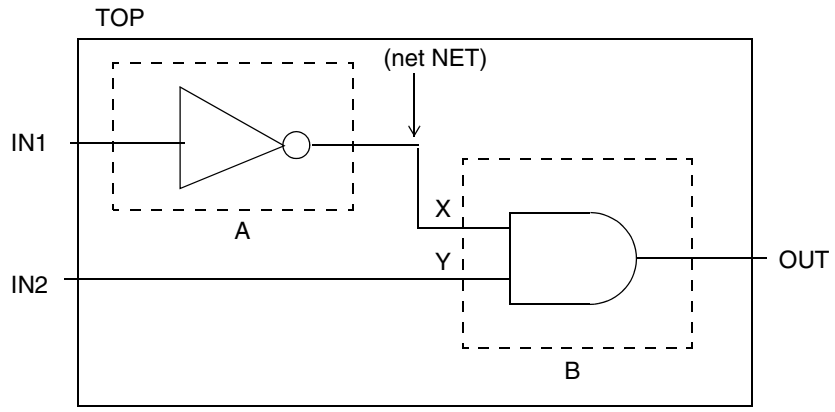
```
current_instance top
create_clock -period 10 -waveform {0 5} clock
characterize_context u5
write_context -out u5.ptsh u5
```

Input Delay and Port Capacitance

The `characterize_context` command calculates arrival and required times along with port capacitance so that the characterized design has the same timing as the chip-level design.

The design in Figure 4-3 is the basis for the load and arrival time calculations using the `characterize_context` command shown in the following sections.

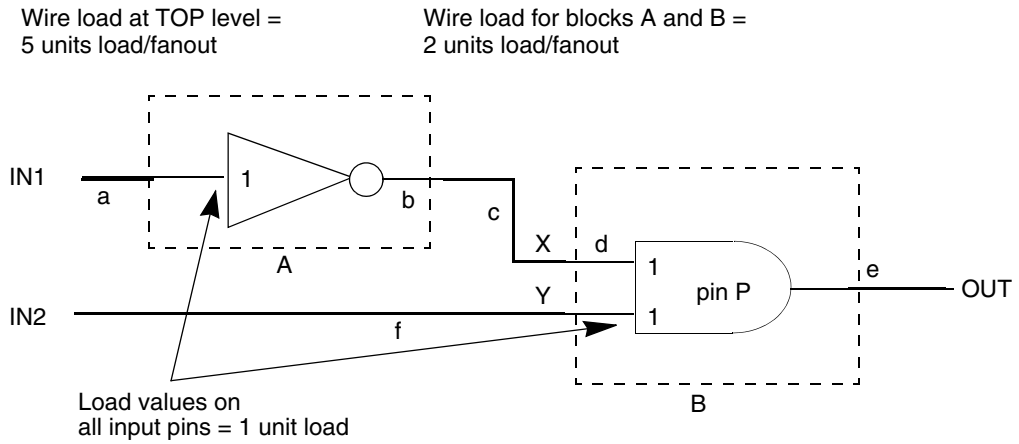
Figure 4-3 Example Hierarchical Design



Calculating Loads

Figure 4-4 shows how PrimeTime annotates the loads of ports and nets in the example in Figure 4-3 and provides some values for wire loads and input pin capacitance.

Figure 4-4 Design With Annotated Ports and Nets



In Figure 4-4, a through f represent wire segments used in the calculations. The example assumes a segmented wire load model that takes the interconnection net loads on the blocks into account and uses a linear function for the wire loads.

For the outside load for pin P in hierarchical block B, the calculation is

```
outside load =
   sum of the loads of all pins on the net loading P
       which are not in B
   + sum of the loads of all segments of net
       driving or loading P which are not in B
```

Each segment's load is

```
segment load =
   number of fanouts * wire load
```

Example

The outside load on input IN1 to block A is calculated by using 0 driving pins, a fanout count of 1 for segment a, and the TOP wire load of 5 loads per fanout:

```
load pins on driving net + load of segment a
   = 0 + (1 * 5)
   = 5
```

For the outside load on the output of block A, the calculation is

```
load pins on net
         + load of segment c
         + load of segment d
   = 1 (for load pin P)
     + (1 * 5)
     + (1 * 2)
   = 1 + 5 + 2
   = 8
```

For each segment in the calculation, the local wire load model is used to calculate the load. The calculation for block A's output pin uses the TOP wire load of 5 loads per fanout for segment c and block B's wire load of 2 loads per fanout for segment d.

For the outside load on the output of block B, the calculation is

```
load pins on net + load of segment e
   = 0 + (1 * 5)
   = 5
```

For the outside load on the input pin X of block B, the calculation is

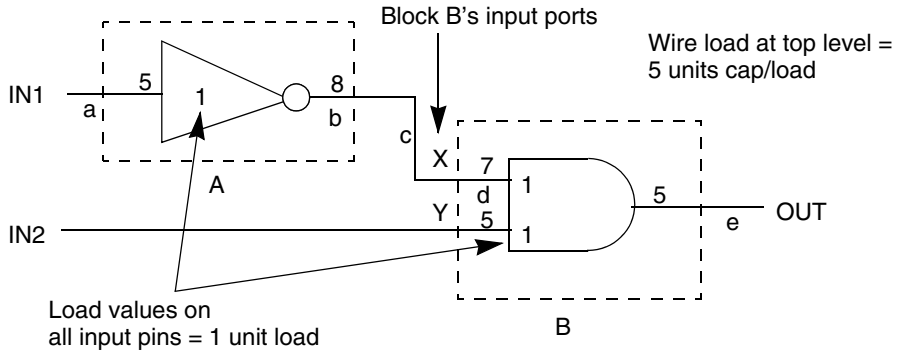
```
load pins on net
          + load of segment b
          + load of segment c
   = 0 + (1 * 2) + (1 * 5)
   = 7
```

For the outside load on the input pin Y, the calculation is

```
pin loads on driving net + load of segment f
   = 0 + 1 * 5
   = 5
```

Figure 4-5 shows the same example with the outside loads annotated after characterization.

Figure 4-5 Design With Outside Loads After Characterization



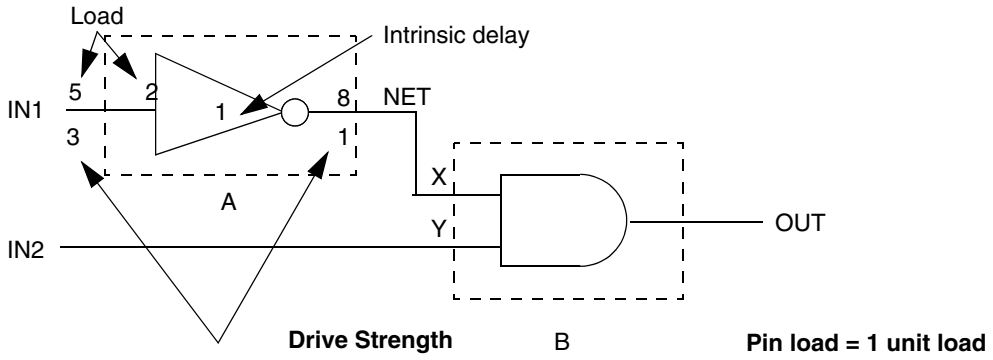
Calculating Input Delays

Because characterization provides accurate details of the outside loads, the input delays or path delays of input signals reflect only the delay through the last gate driving the port. They do not include the connect or transition delays.

For example, the characterized arrival times on the input pins of block B are calculated from the delay to the pin that drives the port being characterized, without the transition or connect delays. This section describes the timing calculations for characterizing block B.

Figure 4-6 shows the example design annotated with default drive strengths and intrinsic delays of block A and signal IN1.

Figure 4-6 Design Annotated for Timing Calculations



For input pin X, the delay calculation is

```
drive strength at IN1 * (wire load + pin load)
   + intrinsic delay of A's cell
   = 3 * (5 + 2 + 1)
     + 1
   = 25
```

Writing Physical Information

The `write_physical_annotations` command writes physical information for a hierarchical block in a design. The exported information includes annotated net and cell delays using SDF. PrimeTime can also export annotated parasitics as a series of pt_shell and dc_shell (dctcl mode) commands. For more information, see the `write_physical_annotations` man page.

Example 1

To write the annotated delays on nets for the hierarchical cell U1, enter

```
pt_shell> write_physical_annotations -sdf U1.sdf \
            -nets_only U1
```

To use the `-append` option to augment the script that PrimeTime generates by the `write_context` command with commands to import the generated delay and parasitics files, enter

```
pt_shell> characterize_context U2

pt_shell> write_context -format dtcl -out U2.dtcl U2

pt_shell> write_physical_annotations -sdf U2.sdf \
            -parasitics U2.rc -format dcsh -append U2.dcsh U2
```

Reporting the Timing Context

The `report_context` command reports the timing context derived by the `characterize_context` command. If you do not specify an option, all context information is reported.

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

Example 1

To report the timing-related context information for instance I1 and I2, use this command. The report shows clocks and their waveforms, point-to-point timing exceptions, input external delay, and output external delays.

```
pt_shell> report_context -timing {I1 I2}
```

Example 2

To report the environment and design rule context information for I2, use this command. The report shows design rule checks, wire load models, driving cell information about input pins, and capacitive load on input and output pins of I2.

```
pt_shell> report_context -env -design_rules I2
```

Generating Scripts for Characterized Contexts

The `write_context` command generates the timing of characterized contexts as a Design Compiler script or a PrimeTime script. Use this command to export context information to other tools. Use `write_context` with the `write_physical_annotations` and `characterize_context` commands to export timing and physical information for a block from the chip-level environment. If you do not specify any options, PrimeTime writes all context information. For more information, see the `write_context` man page.

Example 1

To write the timing-related context information for instance I1 and I2 as a `dc_shell` script to standard output, enter

```
pt_shell> write_context -timing -format dcsh {I1 I2}
```

Example 2

To write the environment and constant input context information for I2 as a PrimeTime shell script into a file called des1.ptsh file, enter

```
pt_shell> write_context -environment \
            -constant_inputs -output des1.ptsh I2
```

Removing Context Information

If you no longer need to report or write the context of an instance, you can delete the context information using the `remove_context` command. For the specified list of instances, the `remove_context` command deletes the timing context derived by the `characterize_context` command. If you do not specify an option, PrimeTime deletes all context information.

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

Example 1

To delete the timing-related context information for instances I1 and I2, enter

```
pt_shell> remove_context -timing {I1 I2}
```

Example 2

To delete the environment and design rule context information for instance I2, enter

```
pt_shell> remove_context -environment -design_rules I2
```

Limitations of Context Characterization

The following limitations apply to the `characterize_context` command:

Timing exceptions

To accurately constrain certain cases of timing exceptions involving registers, the `characterize_context` command can create virtual clocks and set input and output delays relative to that virtual clock instead of to the source register's clock. You might see these additional virtual clocks in the resulting constraint files.

The `characterize_context` command ignores

- Combinational timing exceptions starting outside the cell being characterized
- Timing exceptions specified with the `-through` option

The `characterize_context` command does not derive all combinational timing exceptions affecting paths in the instance being characterized. Combinational timing exceptions are exceptions defined between unclocked points in the design using the `set_max_delay` and `set_min_delay` commands.

Attributes in a design

The `characterize_context` command ignores clock latency or uncertainty and `max_time_borrow` attributes placed on a hierarchical boundary (generally not an issue because these attributes are usually placed on clocks and cells). The `characterize_context` command does not preserve `path_group` information when deriving output constraints for subdesigns; PrimeTime supports the default clock-based path groups.

Generated clock information

Generated clocks are expanded from the master clock and are characterized as separate clocks. They are saved by means of the `create_clock` command.

Modes

The `characterize_context` command does not characterize the mode information in the top design.

Back-annotation information

The `characterize_context` command does not derive back-annotation information, such as back-annotated delays and wire capacitance set on nets contained in the instance being characterized. Use the `write_physical_annotations` command in PrimeTime to do this.

Timing budgets

The `characterize_context` command generates subdesign constraints on the basis of absolute path delays in the current design. Timing budgets are not usually needed when the outputs of modules are registered.

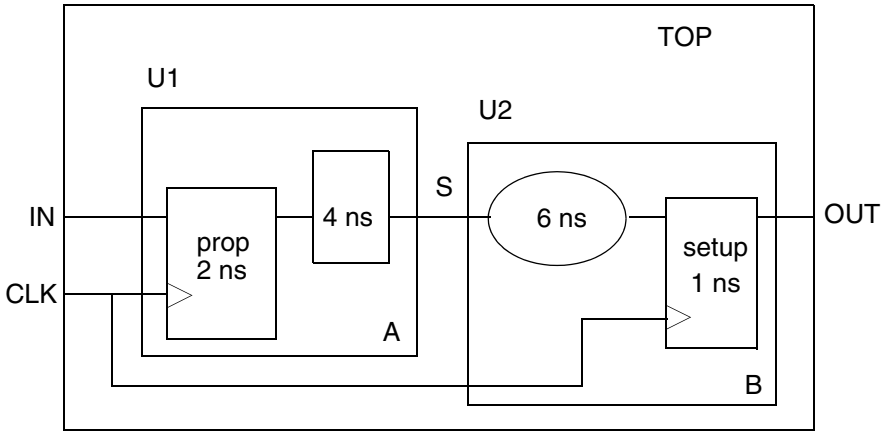
The `characterize_context` command generates constraints consistent with optimal timing budgets, if the design is already fully optimized and all paths have zero slack. The best design methodology in many cases is to use the `characterize_context` command to derive noncritical constraints and use timing budgets to constrain critical paths. Timing budgets give you more direct, repeatable control of optimization than characterized constraints. Timing budgets can also reduce the number of `characterize_context` and `compile` iterations necessary to converge on a coherent design hierarchy.

Example

This example illustrates a case where timing budgets produce better results than the `characterize_context` command.

Figure 4-7 shows a design targeted to run at 50 MHz. The path delay through the circuit is 13 ns (including propagation delay through the first flip-flop and the setup time requirement of the second flip-flop). This design meets the timing requirement, but it is faster than necessary.

Figure 4-7 50-MHz Design



You can improve the circuit area by recompiling this design.

- In PrimeTime, use these commands:

```
pt_shell> create_clock CLK -period 20 -waveform {0 10}
pt_shell> characterize_context {A B}
pt_shell> write_context -out A.dcsh -format dcsh A
pt_shell> write_context -out B.dcsh -format dcsh B
```

- In Design Compiler, use these commands:

```
dc_shell> current_design A
dc_shell> include A.dcsh
dc_shell> compile
dc_shell> current_design B
dc_shell> include B.dcsh
dc_shell> compile
dc_shell> current_design TOP
dc_shell> report_timing
```

PrimeTime generates these constraints for design A:

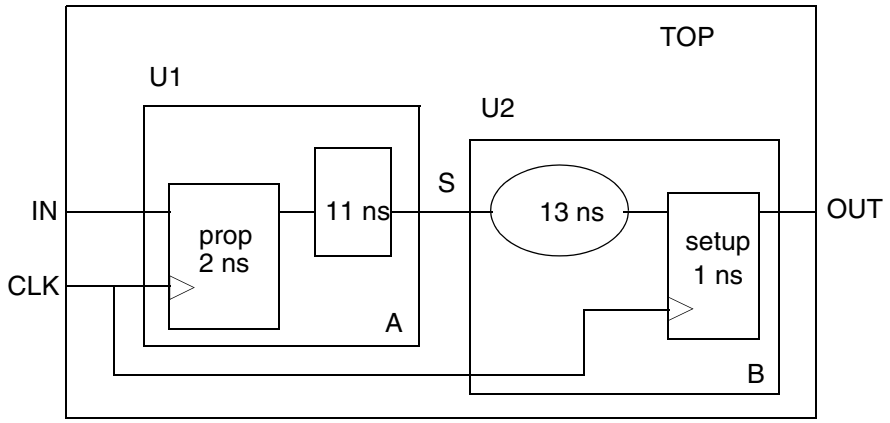
```
create_clock CLK -period 20 -waveform {0 10}
set_output_delay 7 -clock CLK S
```

PrimeTime generates these constraints for design B:

```
create_clock CLK -period 20 -waveform {0 10}
set_input_delay 6 -clock CLK S
```

When design A is compiled, Design Compiler can relax the path delay to 13 ns without violating the generated constraint. Similarly, when design B is compiled, Design Compiler can relax the path delay to 14 ns without violating the generated constraint. The resulting design, shown in Figure 4-8, does not run at the required speed.

Figure 4-8 Design Resulting From Generated Constraint



If you repeat the `characterize_context` and `compile` commands on the design, `characterize_context` generates different constraints.

PrimeTime generates these constraints for design A:

```
create_clock CLK -period 20 -waveform {0 10}
set_output_delay 14 -clock CLK S
```

PrimeTime generates these constraints for design B:

```
create_clock CLK -period 20 -waveform {0 10}
set_input_delay 13 -clock CLK S
```

This resulting design might have the same constraints as the original design. It runs at speed, but it is unnecessarily large. When two blocks are characterized at the same time, then compiled in parallel, each block is overconstrained by the entire amount of negative slack on critical paths between the block and underconstrained by the entire amount of positive slack on noncritical paths between the blocks. To solve this problem, you can use the `characterize_context` command to compile the blocks repeatedly.

- In PrimeTime, use these commands:

  ```
  pt_shell> create_clock CLK -period 20 -waveform {0 10}
  pt_shell> characterize_context A -format dcsh -out A.dcsh
  ```

  Use the following commands for design B:

  ```
  current_design TOP
  read_ddc A.ddc
  ```

```
link TOP
characterize_context B -format dcsh -out B.dcsh
```

- In Design Compiler, use these commands:

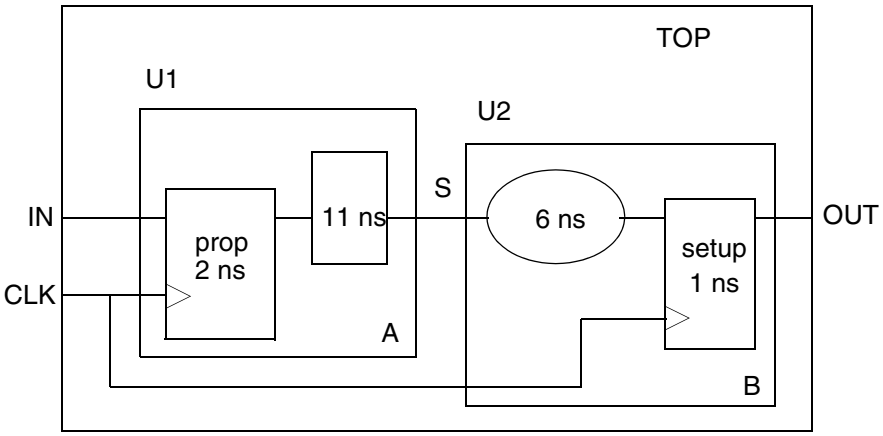
```
dc_shell> current_design A
dc_shell> include A.dcsh
dc_shell> compile
```

Use the following commands for design B:

```
current_design B
include B.dcsh
compile
current_design TOP
report_timing
```

Although this approach produces a smaller design that runs at speed, it might not produce the smallest design that runs at speed. In the preceding command sequence, the available slack is consumed by design A during the first compile. Figure 4-9 shows the resulting design.

Figure 4-9 Design Resulting From Compiling Blocks Repeatedly



To realize the optimal design, you must set time budgets without using the `characterize_context` command. Instead, in Design Compiler, use these commands:

```
current_design A
create_clock CLK -period 20 -waveform {0 10}
set_output_delay 9 -clock CLK S
compile
current_design B
create_clock CLK -period 20 -waveform {0 10}
set_input_delay 11 -clock CLK S
compile
current_design TOP
report_timing
```

5

Advanced Analysis Techniques

There are several advanced timing analysis techniques that you can use in PrimeTime. This chapter explains the analysis techniques, commands, and options.

This chapter contains the following sections:

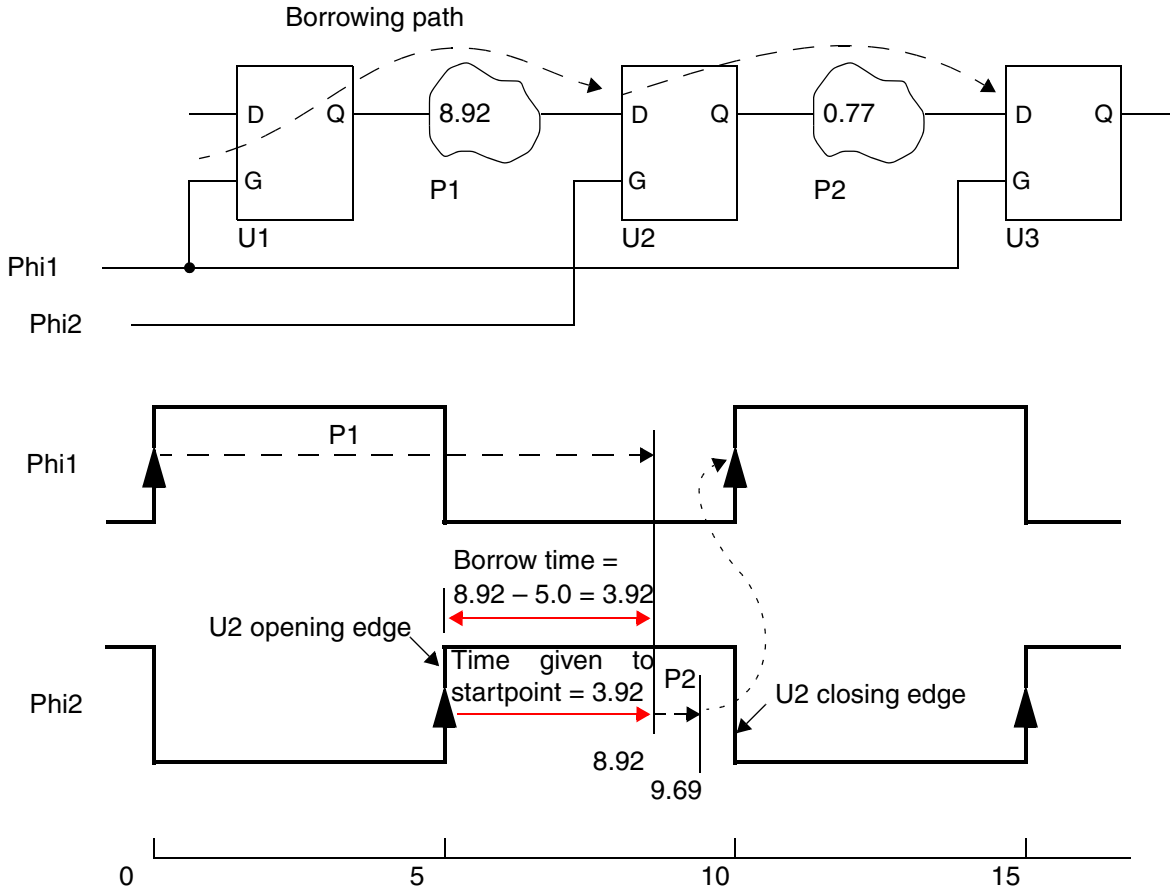
Time Borrowing in Latch-Based Designs

Transparent latches present unusual challenges for static timing analysis tools. A technique known as time borrowing (also known as “cycle stealing”) gives latch-based designs a distinct advantage over flip-flop based designs because a level-sensitive latch is transparent for the duration of an active clock pulse. This technique can relax the normal edge-to-edge timing requirements of synchronous designs. However, it is harder to control the timing of latch-based designs because of the multi-phase clocks used and the lack of “hard” clock edges at which events must occur.

Borrowing Time From Logic Stages

In a design using level-sensitive latches, a path can borrow time from the next logic stage by taking advantage of the fact that latches are transparent while the gate input is asserted. Figure 5-1 shows latch-based stages using a simple two-phase clocking scheme.

Figure 5-1 Latch-Based Timing Paths



A design using level-sensitive latches allows a combinational logic path with a delay longer than the available cycle time as long as it is compensated by shorter path delays in subsequent latch-to-latch stages. For the two-phase design, the available time for latch-to-latch paths is half the clock cycle.

In Figure 5-1, U1, U2, and U3 are positive-level-sensitive latches (active when G = 1), and P1 and P2 are combinational logic paths. For now, assume a library setup time of zero for the latches and zero delay from D to Q in transparent mode. For positive-level-sensitive latches, PrimeTime uses the rising (opening) edge as the reference edge.

The figure shows path delays of P1 = 8.92 and P2 = 0.77. There might appear to be a violation at U2 because the data arrives at U2 after the rising edge of phi2. However, because the U2 latch is transparent for 5 ns and P2 is less than that amount, path P1 can borrow the slack time (3.92 ns) from the path between U2 and U3. Therefore, the sum of P1 and P2 is 9.69, which is less than the required time of 10.00 at U3.

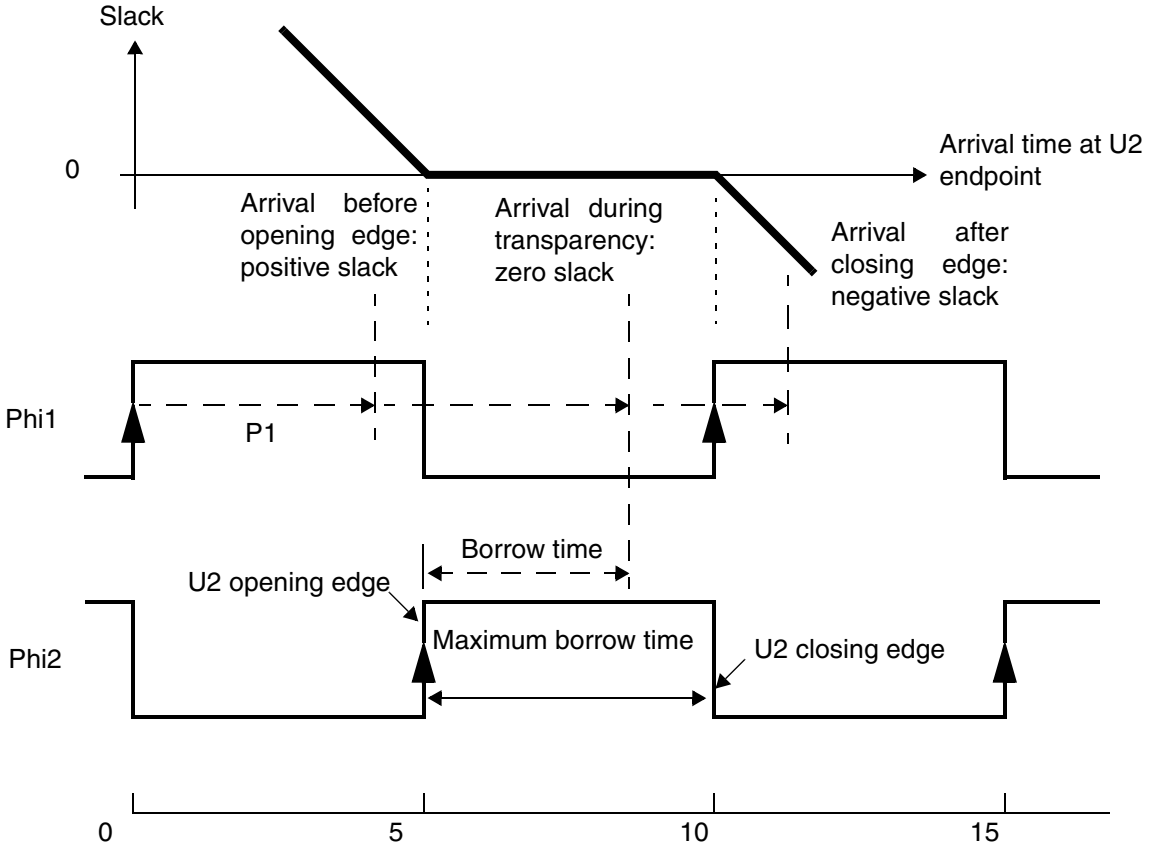
Latch Timing Reports

If the data signal arrives before the opening edge at the endpoint latch, PrimeTime models this behavior just as it would for a flip-flop. The opening edge of the clock (rising edge in this example) captures the data at the endpoint. The same clock edge launches the data at the startpoint of the next path.

On the other hand, if the data signal arrives while the latch is transparent (after the opening edge but before the closing edge at the endpoint latch), PrimeTime models the behavior at the next stage as a launch from the data pin of the latch, rather than from the clock pin. In this case, there is no timing violation and the slack is considered zero. The amount of time borrowed by the stage ending at the latch becomes the departure time for the next stage, subject to certain adjustments described in the next section, “Maximum Borrow Time Adjustments” on page 5-6. A data signal arriving after the closing edge at the endpoint latch is a timing violation.

Figure 5-2 shows how PrimeTime would calculate and report slack for a range of combinational delays through P1, which would result in different arrival times at U1 (refer to the schematic in Figure 5-1). Figure 5-2 does not consider the effects of latch setup time, latency, uncertainly, and clock reconvergence pessimism removal.

Figure 5-2 Latch-Based Timing Path Slack Calculation



Example

The following example shows how the `report_timing` command reports a timing path ending at a transparent latch with time borrowing. The “time borrowed from endpoint” and “time given to startpoint” statements in this report correlate with Figure 5-1.

```
****************************************
Report : timing
         -path short
         -delay max
         -max_paths 1
Design : time_borrow
****************************************
Wire Loading Model Mode: enclosed

Startpoint: U1 (positive level-sensitive latch clocked by
Phi1)
Endpoint: U2 (positive level-sensitive latch clocked by Phi2)
Path Group: Phi2
```

```
Path Type: max

Point                                          Incr       Path
-------------------------------------------------------------
-
clock Phi1 (rise edge)                         0.00       0.00
clock network delay (ideal)                    0.00       0.00
U1/G (LATCH)                                   0.00       0.00 r
U1/Q (LATCH)                                   0.57       0.57 r
...
U2/D (LATCH)                                   8.35       8.92 r
data arrival time                                         8.92

clock Phi2 (rise edge)                         5.00       5.00
clock network delay (ideal)                    0.00       5.00
U2/G (LATCH)                                   0.00       5.00 r
time borrowed from endpoint                    3.92       8.92
data required time                                        8.92
-------------------------------------------------------------
-
data required time                                        8.92
data arrival time                                        -8.92
-------------------------------------------------------------
-
slack (MET)                                               0.00

Time Borrowing Information
-------------------------------------------------
Phi2 nominal pulse width                       5.00
library setup time                            -0.46
-------------------------------------------------
max time borrow                                4.54
actual time borrow                             3.92
-------------------------------------------------


Startpoint: U2 (positive level-sensitive latch clocked by
Phi2)
Endpoint: U3 (positive level-sensitive latch clocked by Phi1)
Path Group: Phi1
Path Type: max


Point                                          Incr       Path
-------------------------------------------------------------
-
clock Phi2 (rise edge)                         5.00       5.00
clock network delay (ideal)                    0.00       5.00
time given to startpoint                       3.92       8.92
U2/D (LATCH)                                   0.00       8.92 r
U2/Q (LATCH)                                   0.53       9.45 r
...
```

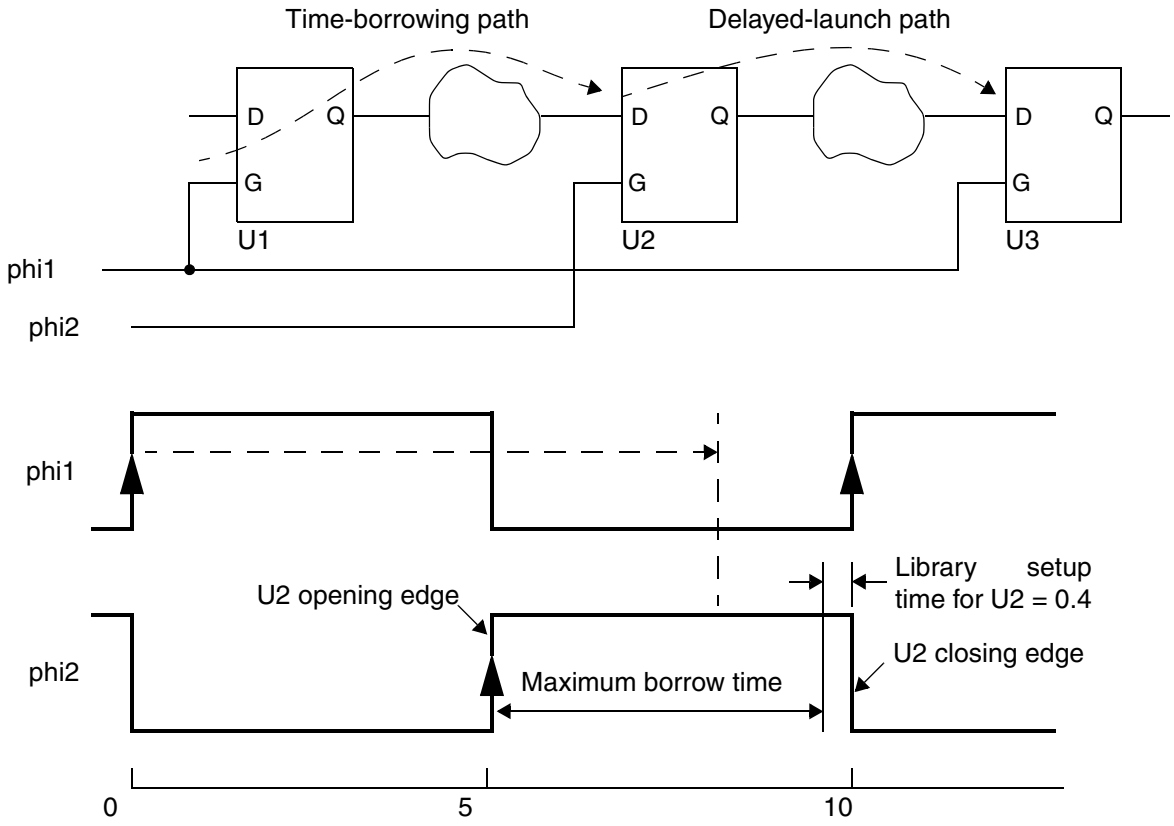
```
U3/D (LATCH)                              0.24       9.69 f
data arrival time                                    9.69

clock Phi1 (rise edge)                   10.00      10.00
clock network delay (ideal)               0.00      10.00
U3/G (LATCH)                              0.00      10.00 r
time borrowed from endpoint               0.00      10.00
data required time                                  10.00
-----------------------------------------------------------
-
data required time                                  10.00
data arrival time                                   -9.69
-----------------------------------------------------------
-
slack (MET)                                          0.31

Time Borrowing Information
-------------------------------------------------
Phi1 nominal pulse width                  5.00
library setup time                       -0.49
-------------------------------------------------
max time borrow                           4.51
actual time borrow                        0.00
-------------------------------------------------
```

For the U2 to U3 path, time must be added to compensate for the time borrowed, so PrimeTime adds 3.92 ns to the launch time of U2. This is reported in the second path's timing report. Because the P2 path has enough slack, neither path is in violation.

Maximum Borrow Time Adjustments

The maximum amount of time that can be borrowed at an endpoint latch is based on the clock pulse width (the time from the opening edge to the closing edge of the gate signal) as defined by the `create_clock` command, minus the library setup time of the latch. See Figure 5-3.

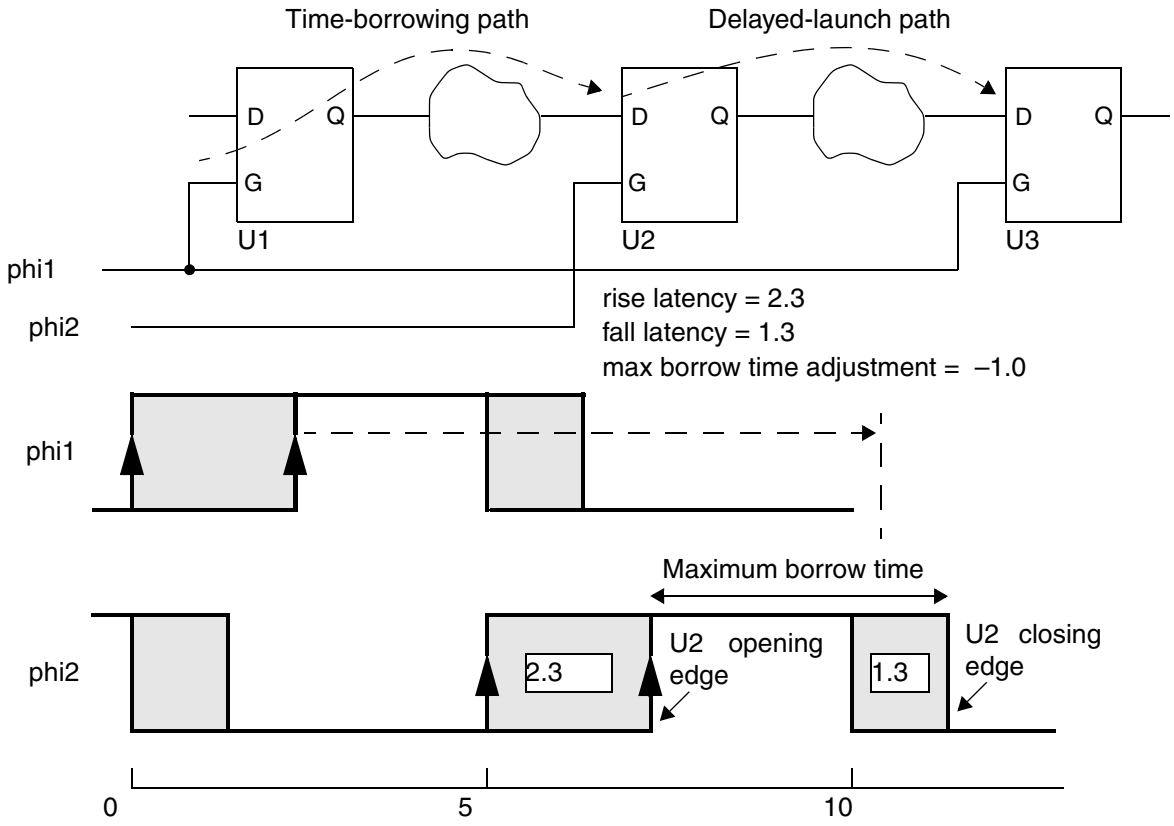
Figure 5-3 Maximum Borrow Time Reduced by Setup Requirement



For better accuracy, PrimeTime adjusts this amount further for the effects of clock latency, clock uncertainty, and clock reconvergence pessimism removal.

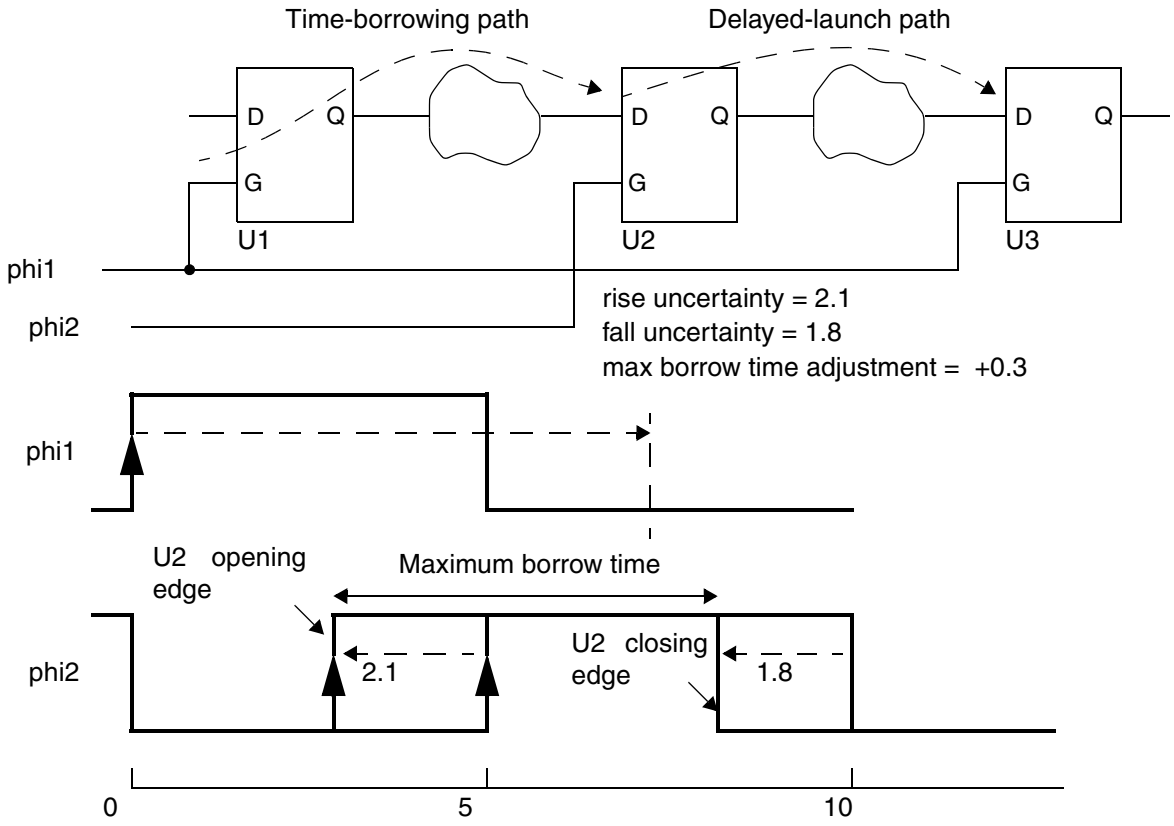
Clock latency can be defined with the `set_clock_latency` command or calculated by PrimeTime from propagated delays in the clock tree (enabled by `set_propagated_clock`). Latency values can be different for the rising and falling edges of the clock pulse, which can affect the pulse width and thus the maximum borrow time. See Figure 5-4.

Figure 5-4 Maximum Borrow Time Adjustment for Latency



Clock uncertainty can be defined with the `set_clock_uncertainty` command. For a setup check, PrimeTime considers the earliest possible arrival of capture clock edges. Differences in uncertainty between rising and falling edges can affect the clock pulse width and maximum borrow time. See Figure 5-5.

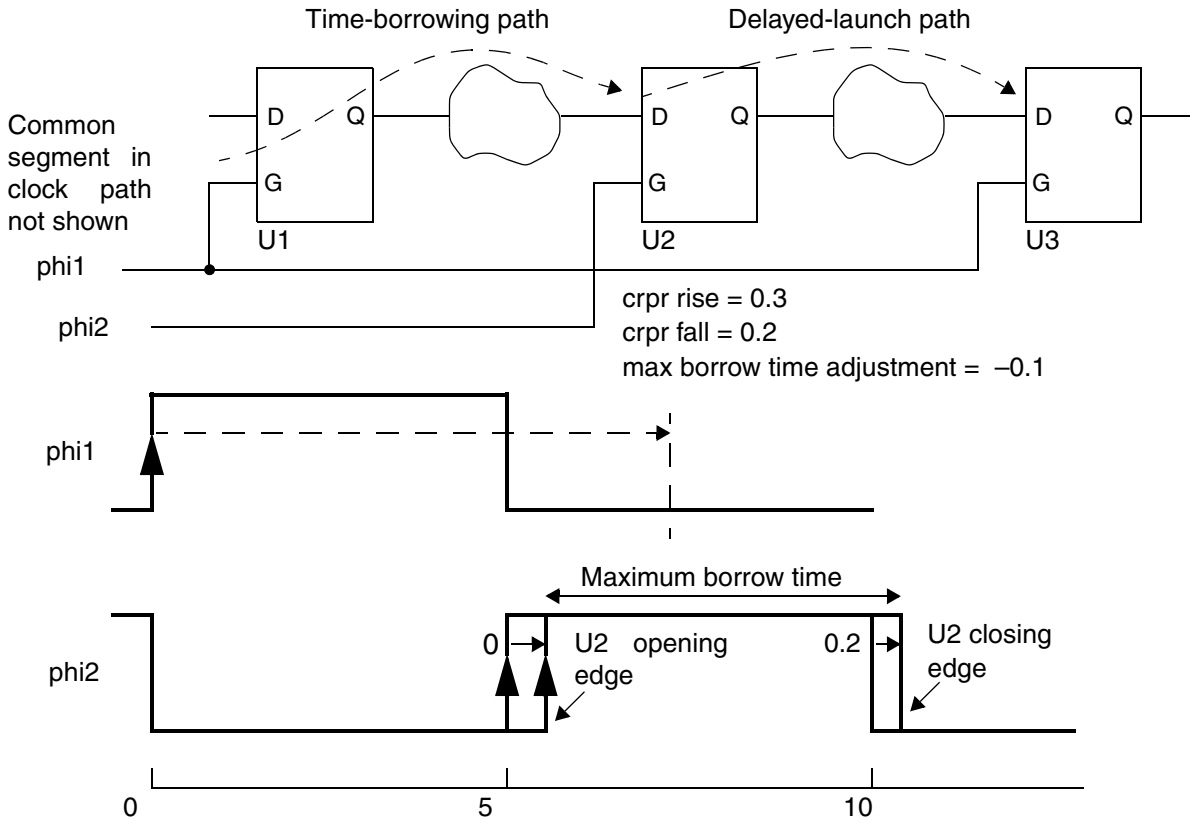
Figure 5-5 Maximum Borrow Time Adjustment for Uncertainty



Clock reconvergence pessimism removal (CRPR) is an analysis technique that corrects any inaccuracy resulting from a common segment in the launch and capture clock paths, as explained in the “Min-Max Analysis” chapter in the *PrimeTime Fundamentals User Guide*. You enable this feature by setting the `timing_remove_clock_reconvergece_pessimism` variable to `true`.

Application of CRPR shifts the clock edge times, like clock uncertainty. However, applying clock uncertainty makes the analysis more pessimistic, whereas applying CRPR makes the analysis less pessimistic, so the direction of the edge shift is in the positive direction rather than the negative direction. Differences in CRPR between rising and falling edges can affect the clock pulse width and maximum borrow time. See Figure 5-6.

Figure 5-6 Maximum Borrow Time Adjustment for CRPR



To calculate the maximum allowable borrow time, PrimeTime starts with the clock pulse width and then adjusts it for the applicable effects of clock latency, clock uncertainty, clock reconvergence pessimism removal, and library setup time of the endpoint latch. The `report_timing` command reports the clock pulse width and the adjustments as follows:

```
Time Borrowing Information
-------------------------------------------------
CLK nominal pulse width                   5.00
clock latency difference                 -1.00
clock uncertainty difference              0.30
CRPR difference                          -0.10
library setup time                       -0.40
-------------------------------------------------
max time borrow                           3.80
-------------------------------------------------
...
```

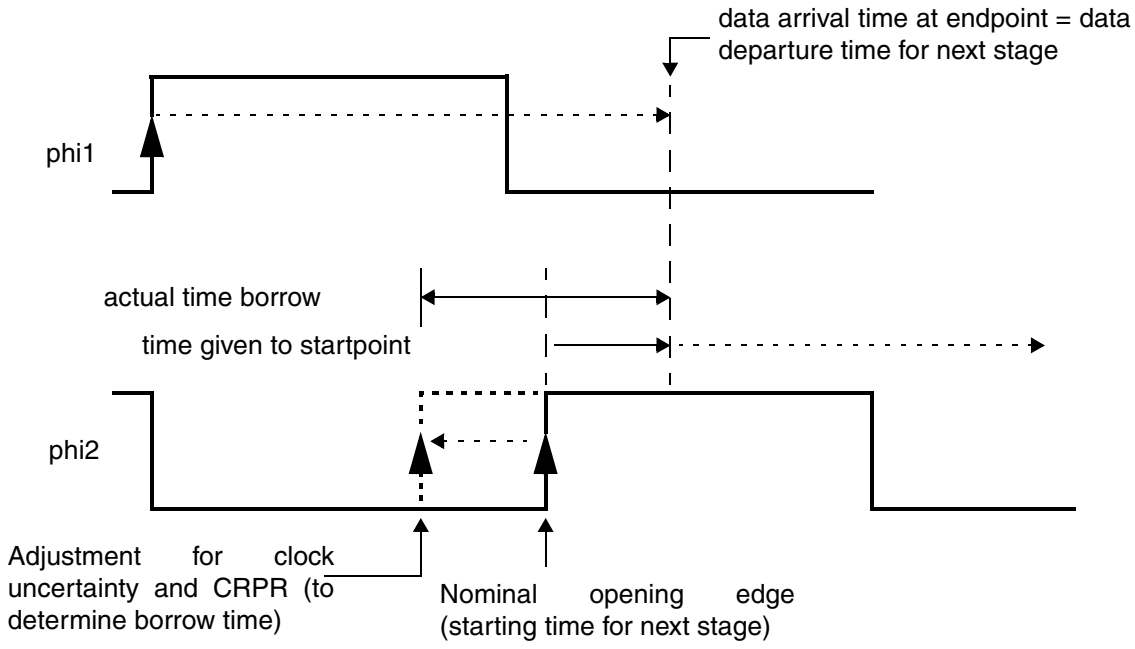
Time Borrowed and Time Given

PrimeTime calculates the amount of time borrowed in relation to the arrival time of the opening clock edge at the latch. PrimeTime first adjusts the arrival time of the opening edge to account for path-specific effects of both uncertainty and CRPR (if applicable). Then it compares the adjusted value to the signal arrival time at the data pin to determine the amount of time borrowed at the path endpoint, if any.

If uncertainty and CRPR exist for the opening edge of the latch, the time borrowed will be different from the amount of time given to the startpoint of the next stage. To determine the time given to the startpoint, PrimeTime subtracts the uncertainty and CRPR adjustments. This subtraction is necessary in transparent mode to make the launch at the next stage occur precisely when the signal arrives at the data pin, as illustrated in Figure 5-7.

Note, however, that when the `timing_early_launch_at_borrowing_latches` variable is disabled, the data arrival and launch times are not identical, owing to the deliberate application of a late clock latency to launch the next stage. This mode is recommended when CRPR is enabled. Note that the CRPR adjustment to the time given to the startpoint of the next stage is not applied in this mode. For more information, see the man page for this variable.

Figure 5-7 Borrow Time and Time Given to Startpoint in Transparent Mode



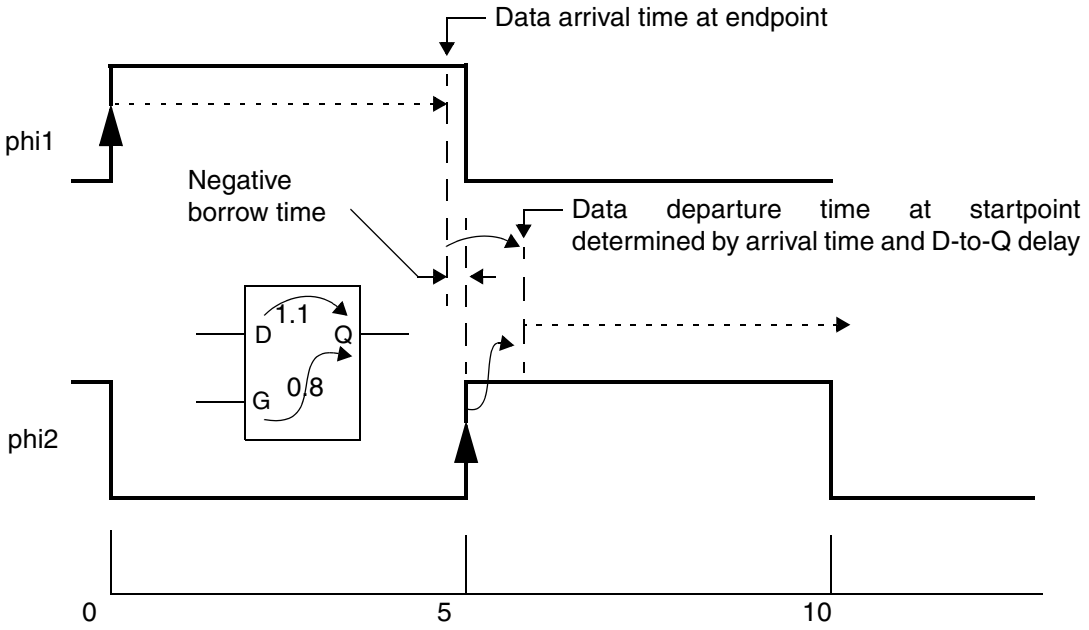
The `report_timing` command reports the amount of time borrowing and uncertainty/CRPR adjustments as follows:

```
Time Borrowing Information
-------------------------------------------------
CLK nominal pulse width                   5.00
clock latency difference                 -1.00
clock uncertainty difference              0.30
CRPR difference                          -0.10
library setup time                       -0.40
-------------------------------------------------
max time borrow                           3.80
-------------------------------------------------
actual time borrow                        3.40
open edge uncertainty                    -2.10
open edge CRPR                            0.30
-------------------------------------------------
time given to startpoint                  1.60
-------------------------------------------------
```

In most cases, data arrival before the opening clock edge results in no borrowing, whereas data arrival after the opening clock edge results in borrowing. When borrowing occurs, the arrival time minus the clock edge time is reported as “time borrowed from endpoint” from the perspective of the path segment ending at the latch. This same amount of time is reported as “time given to startpoint” from the perspective of the path segment starting from the latch.

However, in certain cases, data arrival just before the clock edge can result in borrowing. This happens when the D-to-Q delay of the latch is more than its clock-to-Q delay, and the arrival time is very close to the clock edge, as shown in Figure 5-8. Under these circumstances, the data departure time for the next path is determined by the data arrival time and the D-to-Q delay, rather than the clock-to-Q delay of the no-borrowing case.

Figure 5-8 Negative Borrow Time



PrimeTime accounts for this condition by allowing time borrowing. The arrival time minus the clock edge time is a negative number, so the amount of borrowing is negative. The borrowed amount is reported as “time given to endpoint” from the perspective of the path segment ending at the latch, and “time borrowed from startpoint” from the perspective of the path segment starting from the latch. This amount of time cannot exceed the difference between the D-to-Q delay and clock-to-Q delay of the latch.

Limiting Time Borrowing

The `set_max_time_borrow` command limits time borrowing to a specified amount for all latch endpoints within a specified scope of the design. You can use this command to set a more restrictive constraint on borrowing. At the specified latch endpoints, PrimeTime limits borrowing to the specified amount or to the default value determined by the adjusted pulse width, whichever is smaller.

If the `default_max_time_borrow` constraint is zero, no time borrowing is allowed and PrimeTime analyzes the latch like a flip-flop.

If the `default_max_time_borrow` constraint is negative, the data signal must be stable before the open edge of the clock. You can use this feature to ensure that the enable input arrives before the clock (on a gated-clock latch, for example). For more information, see the `set_max_time_borrow` man page.

To show the `max_time_borrow` attributes in the design, use the `report_exceptions` command. To remove the maximum time borrow limit set on specified objects using `set_max_time_borrow`, use the `remove_max_time_borrow` command.

These commands set maximum time borrowing of 2.5 on all latches clocked by clk:

```
pt_shell> create_clock -period 10 -waveform {0 5} clk

pt_shell> set_max_time_borrow 2.5 find(clock,clk)

pt_shell> report_timing -to latch1/D
```

The `user_max_time_borrow` constraint (from the `set_max_time_borrow` command) effectively reduces the pulse available for time borrowing from 5.0 to 2.5:

```
Time Borrowing Information
-------------------------------------------------
user max_time_borrow                     2.50
-------------------------------------------------
max time borrow                          2.50
actual time borrow                       0.00
-------------------------------------------------
```

The time borrowed from an endpoint is usually the same as the time given to the next startpoint when the endpoint and startpoint are both the D pin of the same latch. There can be a small difference due to uncertainty and CRPR adjustment of the opening edge of the latch. Sometimes, if the borrowing at D is small, the most critical path from the latch still comes from G; however, if you do the explicit report from D, then the time given is the same as the time borrowed.

Path-Based Timing Analysis

Static timing analysis tools are designed to be pessimistic in certain respects to ensure detection of all timing violations. For example, by default, PrimeTime considers both the worst arrival time and worst slew among all signals feeding into a path, even if the signal with the worst arrival time is different from the one with the worst slew.

You can control the behavior of PrimeTime with respect to worst slew with the `timing_slew_propagation_mode` variable. 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.

Although the worst arrival mode reduces the pessimism for the critical path, non-critical paths may have optimism for highly slew-sensitive paths. For more information, see the man page for this variable or the description in the *PrimeTime Fundamentals User Guide*.

A better method to reduce the pessimism for critical paths is to analyze those paths in isolation from other paths. This method is called path-based timing analysis. When recalculating a timing path, PrimeTime propagates the edge along each path of interest, ignoring slews from side arcs along the path, and it performs delay calculation to compute the path-specific slack. Thus, path-based analysis recalculates the timing in a timing path without considering outside paths that might otherwise affect the arrival times and slew values used in the calculation. The command annotates the path collection with new timing information such as arrival times, slews, and slack.

Timing path effects, such as clock reconvergence pessimism removal and crosstalk effects (delta delays), are also recomputed specifically to that path. Path-based analysis is useful when there are only a few violations remaining in the analysis, and you want to find out whether these violations are caused by pessimistic analysis of arrival and slew times.

One important property of path-based analysis is that the slack ordering of paths can change due to the path recalculation. In other words, the most critical path before recalculation may not be the most critical path after recalculation. Because of this property, multiple paths to an endpoint may need to be recalculated to determine the true post-recalculation critical path. It cannot be emphasized enough that recalculating the single worst failing path to an endpoint does not provide the worst recalculated slack to that endpoint.

There are two ways to recalculate a specific path or set of paths. One way is by using the `-pba_mode path` option of the `report_timing` or `get_timing_paths` command. No path search is performed to determine whether those paths are truly the worst recalculated paths. Here is an example of the path-specific recalculation approach:

```
pt_shell> report_timing -pba_mode path
```

In this example, the single worst path in each path group is obtained from the design, then recalculated, then reported. No searching is performed for other paths that might have worse slack after recalculation. It is possible that there are subcritical paths that would have worse slack after recalculation than the reported paths. This method is useful for quickly gaining an idea of how much improvement is provided by path-based analysis without the runtime of a full path search. The slack results either match those from a full path search or are slightly optimistic.

The second way is to perform an exhaustive path search for the worst recalculated timing. You do this by using the `-pba_mode exhaustive` option of the `report_timing` command. PrimeTime performs an exhaustive recalculated path search, recalculating as many paths as necessary to ensure that the paths returned are the worst recalculated paths that meet the specified options.

To ensure optimal path search runtime, the following usage is recommended:

- Use the `-pba_mode exhaustive` option with the `-slack_lesser_than` option to keep the search bounded.
- Perform an exhaustive path search only when `-pba_mode path` shows that the slacks are reasonably close to the desired threshold.
- Ensure that the reporting is as specific as possible. If reports are desired only for specific path groups or startpoints and endpoints, specify the required reporting options.
- Enable worst parallel-arc reporting by setting the `timing_report_use_worst_parallel_cell_arc` variable to `true`. For more information about this variable, see the man page.

To see the critical path in each path group while applying exhaustive path-based recalculation, use the following syntax:

```
pt_shell> report_timing -pba_mode exhaustive -slack_lesser_than 0
```

In the example above, if no paths are reported with negative slack, the design has no timing violations.

Path-based recalculation can also be combined with other timing path selection options, such as `-from/-through/-to` or `-max_paths/-nworst`. For `-pba_mode path`, the `-slack_lesser_than` option applies to the original slack used to obtain the paths. The reported recalculated slack might be improved.

When using `-max_paths` or `-nworst` with `-pba_mode` exhaustive, the path limits represent the desired path counts at the completion of the search. Additional paths are searched during the path search process to ensure a complete result. For example, to report the worst path to each violating endpoint with recalculation taken into account, use the following syntax:

```
pt_shell> report_timing -pba_mode exhaustive -nworst 1 \
           -slack_lesser 0 -max_paths 1000
```

Multiple paths to each failing endpoint are searched to find the worst recalculated failing path.

The `report_timing -pba_mode exhaustive` option does not support other options, such as `-true`, `-justify`, `-start_end_pair`, and some other postprocessing options, such as the `-slack_greater_than` option.

You can also use the `get_timing_paths -pba_mode path` command to recalculate a collection of timing paths. When passed a collection of timing paths with this option, the collection is recalculated and returned. For example,

```
pt_shell> set paths [get_timing_paths -max_paths 10]
pt_shell> set recalc_paths [get_timing_paths -pba_mode path $paths]
```

After a set of timing paths have been recalculated, you can use the `report_timing` command to report the recalculated timing values of the path collection. For example,

```
pt_shell> report_timing $recalc_paths
```

The recalculated values are accessible only by the `report_timing` and `get_attribute` commands, not by other commands, such as the `report_constraint` command. However, you can use the `report_attribute` command to report the newly calculated attribute values directly inside the path collection.

The path-based analysis feature allows control of whether or not clock paths and latch-based borrowing paths are recalculated. Clock and borrowing path information is controlled by the `pba_recalculate_full_path` variable.

When set to `false`, only the data portion of the path is recalculated. Clock and borrowing portions of the paths are not recalculated by the path-based engine if they are present. This makes it possible to avoid recalculation on these portions of the paths, but still have them included in the timing reports.

If the `pba_recalculate_full_path` variable is set to `true` (default), all path recalculation commands recalculate full clock paths, borrowing paths, and data paths. To recalculate the clock path, you must also include the `-path full_clock_expanded` option. To recalculate the borrow path, you must also include it the `-trace_latch_borrow` option. If these options were not specified, the corresponding path components cannot be recalculated even if the variable is set to `true`. For more information about this capability, see the `pba_recalculate_full_path` variable man page.

Note:

When the `pba_recalculate_full_path` variable is set to `true`, path-based analysis recalculates the borrowing at transparent latches only if the `-trace_latch_borrow` option has been specified. Otherwise, the existing borrowing at the original path's startpoint is used.

The `timing_report_recalculation_status` variable facilitates debugging and eases the runtime concern. It works for both the `get_timing_path -pba_mode exhaustive` and `report_timing -pba_mode exhaustive` commands. For more information about this variable, see the man page.

The amount of time needed for the integrated path search depends on a few factors:

- Amount of slack improvement provided by recalculation
- Number of paths involved in the search (`-slack_lesser than`, `-from/-through/-to`)
- Number of paths to be returned to the user (`-max_paths` and `-nworst`)

In particular, large `-nworst` values can significantly increase the runtime of the search. The `pba_exhaustive_endpoint_path_limit` variable helps to prevent excessive runtime by terminating the exhaustive path searching for an endpoint once a specific limit on the per-endpoint number of paths to recalculate has been reached.

When you perform a timing update on the original design (for example, with an `update_timing` command), any timing path collections created by the `get_timing_paths` command are automatically deleted.

Composite Current Source (CCS) Receiver Model for Path-Based Analysis

For path-based analysis, PrimeTime uses the actual receiver models in the path being analyzed. For the cells that are side loads, the worst-case receiver models are used. During path-based analysis, receiver models are derived using the effective capacitance (Ceff) on the output of the load cell. You save this capacitance for path-based analysis by setting the `rc_cache_min_max_rise_fall_ceff` variable to `true` (the default is false). Note that this variable is set to true automatically if crosstalk analysis is enabled or if the GUI was invoked prior to the `update_timing` command.

Fast Performance Analysis Mode

The fast performance analysis mode in PrimeTime provides increased performance when accuracy is less critical. This mode provides a single setting that configures PrimeTime and PrimeTime SI for higher performance for the many early analysis runs while maintaining reasonable accuracy. When accuracy is critical, such as during late stage timing and ECO closure runs, you should ensure that this mode is disabled. By default, fast analysis mode is turned off. You should enable fast analysis mode at the beginning of your script by using the following command:

```
set_program_options -enable_fast_analysis
```

Note:

You should turn on fast analysis mode before loading designs and libraries because enabling fast analysis might cause changes when loading them.

When you choose fast analysis mode, it gets higher priority over any other options or variables to provide higher performance. For example, it focuses default reporting on only violating paths. A read-only `sh_fast_analysis_mode_enabled` variable monitors whether fast analysis mode is enabled. The `report_design` command shows whether the fast analysis mode is enabled or disabled.

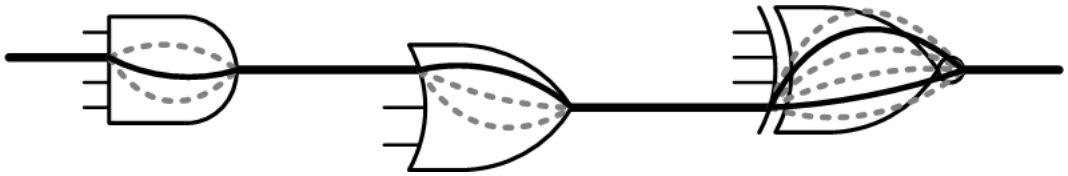
For more information about these command options, the variable, and an example report, see the specific man pages.

Parallel Arc Path Tracing

To ensure that the `report_timing` command uses only the worst arc in each sense set of parallel arcs for path tracing, set the `timing_report_use_worst_parallel_cell_arc` variable to `true`. The variable is set to `false` by default. Figure 5-9 shows the result if you enable this variable by setting it to `true`.

```
set timing_report_use_worst_parallel_cell_arc true
```

Figure 5-9 Worst Arc Used for Path Tracing



Only paths with the worst arc behaviors through the logic with a given rise and fall edge sequence are returned. The worst arc is the fastest arc for minimum delay tracing and it is the slowest arc for maximum delay tracing. Subcritical timing paths through other combinations of timing arcs with the same edge direction sequence are not returned. However, since timing arcs with different senses are still considered unique, different paths with unique rise and fall edge sequences through non-unate gates are still returned.

This might cut down substantially on the number of paths returned by large -nworst values, improving analysis efficiency. If the -nworst limit is not being reached, such as when reporting paths to a single endpoint, a lesser number of timing paths are returned. If -nworst has reached its limit, such as when reporting across an entire path group, the paths that are returned up to the limit might have a more varied topological exploration of the logic. This can help improve the efficiency of reporting scripts, bottleneck analysis, ECO scripts that are driven by timing_path collections.

This feature impacts both the `report_timing` and `get_timing_paths` commands when you specify an -nworst value greater than 1. Only parallel arcs of the same sense are affected. If there are different sense arcs in parallel, such as positive_unate and negative_unate across an XOR gate, they are still considered unique. This feature affects path tracing behavior at reporting time only. It does not affect the behavior of the `update_timing` command. You can change the value of the variable at any time without incurring a timing update penalty.

Support for Retain Arcs

PrimeTime can load retain arcs for timing models from library files, annotate the retain arcs from SDF input files, and report on these arcs. Retain arcs are similar to hold-check arcs and are typically used for modeling random access memory (RAM). They are defined between a clock pin and the data output of a RAM, and they are always defined in parallel with the parent arc, which is the ordinary or default delay arc between the same two pins. A retain arc does not generate an actual timing check during timing analysis, but is treated as another delay arc that is connected in parallel with its parent arc.

Clock-to-output retain arcs guarantee that the RAM output does not change for a specific interval of time after the clock edge. When the retain arc delay is less than its parent arc, the retain arc appears in a timing report for only the minimum delay paths. When the retain arc delay is longer than its parent arc, the retain arc can also appear in the maximum delay path report with no error messages or warnings.

To report all of the timing arcs for cells in a technology library, including retain arcs, use the `-timing_arcs` option with the `report_lib` command. Use the `check_timing -retain` command to check if the retain arc has a delay greater than its parent arc.

The `read_sdf` command supports retain arcs, but the default behavior of the `write_sdf` command does not support those arcs. To write out retain arcs, use SDF version 3.0 format, not the default version 2.1 format, by specifying the following syntax:

```
pt_shell> write_sdf -version 3.0 file
```

To map retaining information for arcs, use these functions:

- `min_rise_retain_delay`
- `min_fall_retain_delay`
- `max_rise_retain_delay`
- `max_fall_retain_delay`

True and False Path Detection

An advanced feature of PrimeTime is the ability to determine whether a specified path is a true path or false path. PrimeTime can also find the most critical true path in the design.

PrimeTime categorizes false paths into two main types: functional and delay-dependent.

Figure 5-10 shows an example of a functional false path that results from resource sharing. When mux1 is selected, the path through AND gate c_d is blocked.

Figure 5-10 Functional False Path Due to Resource Sharing

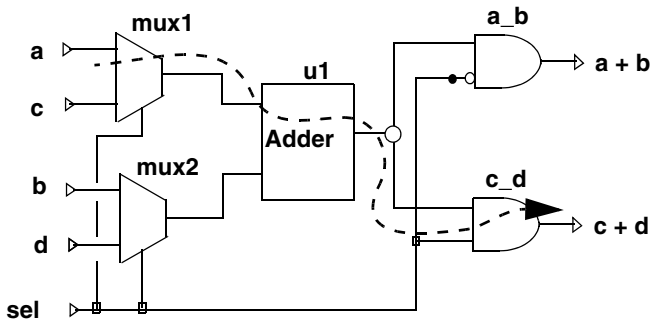
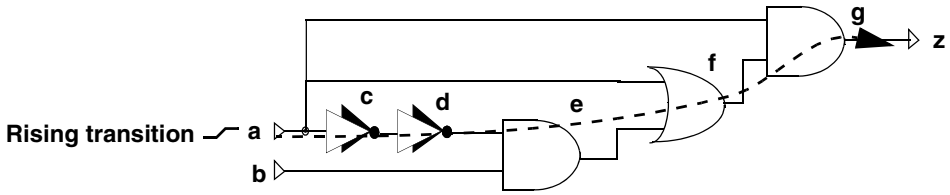


Figure 5-11 shows an example of a delay-dependent false path in carry-lookahead logic. The highlighted path (from a to g) is false for a rising transition at input a. The shorter direct path (from a, through f, to g) determines the longest true path.

Figure 5-11 Delay-Dependent False Path in Carry-Lookahead Logic



False path detection works in combination with case analysis. If you specify some pins in your design to be at a constant logic value, the false path detection features take these propagated logic constant values into account. See “Justifying Paths” on page 5-22.

Reporting True or False Paths

When reporting a timing path, the `report_timing -justify` option uses a justification process to check whether the reported path is a true path or a false path. True path justification takes into account the logic function of the design and tries to find a set of input vectors that can sensitize the reported path.

If PrimeTime cannot find an input vector, it considers the path a false path. If it finds an input vector, it considers the path a true path and shows the input vector at the end of the report.

The `report_timing -false` option is similar to the `report_timing -justify` option except that it only reports false paths. Although this is a convenient way to identify the false paths in the design, do not use this command to create a set of `set_false_path` commands; if the delays change in the design, the false paths might become true paths.

Reporting True Paths

The true path reporting feature is an advanced algorithm that detects the most critical true path. Use the `-justify` or `-true` options to the `report_timing` command to incorporate functional information into the critical path computation. With `-true`, the `report_timing` command is less pessimistic.

`-justify`

Provides a quick means to determine whether any path shown is true.

`-true`

Runs a search algorithm to find the most critical true path.

A path is true if PrimeTime can find an input vector that sensitizes the path. Sensitization is determined using floating mode delay, in which all nodes are assumed to be in the X-state (unknown) before the application of an input vector.

The sensitization criteria for each gate on the path are as follows:

- The online signal is a controlling value, and there are no side inputs with controlling values that arrived earlier (all side inputs must be noncontrolling or later arriving).
- The online signal is a noncontrolling value. All other side inputs are noncontrolling, and no side input arrives later than the online signal.

For these criteria, PrimeTime defines arrival time as the time when a simulated value arrives to replace the X-state, not the structurally longest arrival time that PrimeTime typically computes.

Justifying Paths

If you suspect that the paths that PrimeTime is reporting are false, use the `report_timing -justify` command to generate a timing report. For most designs, the structurally longest path is a true path, so the `-justify` option finds an input vector that sensitizes the path. You can proceed with confidence knowing that PrimeTime is reporting the true delay (and that timing optimization will be performed on the critical path).

The `-justify` option runs fast because it focuses on the single reported path (there is no searching and very little backtracking required). If PrimeTime finds an input vector, it displays it along with the path. If the path is false, PrimeTime reports it as such. You can see more false paths by combining the `-justify` option with the `-nworst` option. If the `-justify` option reports that the structurally longest path is false, use the `-true` option to find the longest true path.

The `report_timing` command with the `-justify` option generates a report similar to this one.

```
pt_shell> report_timing -justify

****************************************
Report : timing
         -justify
Design : FP_SHR
****************************************
Operating Conditions:
Wire Loading Model Mode: top

A False path:

  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

True-delay Input Vector
  ---------------------------------------
   ( False Path )
  ---------------------------------------
  True-delay conclusions:
-----------------------------------------------------------
  1. No input-vector could be found to justify reported path.
The
search completed, so reported path is false.
-----------------------------------------------------------
```

Note:

In this example, the path reported starts with the rising transition at input a (see Figure 5-11 on page 5-21 for more information).

Finding the Longest True Path

The `-true` option of the `report_timing` command runs a search algorithm to find the longest true path. For many designs, this algorithm runs very fast. However, some designs require a large amount of backtracking and the algorithm can take a long time to complete. PrimeTime provides a backtracking feature to control runtime.

In most cases, PrimeTime can find the longest true path fairly quickly using a low backtrack limit. Some designs require a high backtrack limit to find the real longest true path, but these designs are rare. The default backtrack limit is 1,000, which is a good compromise between runtime and accuracy.

The `report_timing` command with the `-true` option generates this report. In this example, particularly notice the lines that are in boldface.

```
pt_shell> report_timing -true

****************************************
Report : timing
         -true
Design : FP_SHR
****************************************

Operating Conditions:
Wire Loading Model Mode: top

  A True path:

  Startpoint: a (input port)
  Endpoint: a_b (output port)
  Path Group: default
  Path Type: max
Point                                  Incr       Path
-----------------------------------------------------------
  input external del                  10.00      10.00
  a (in)                               0.00      10.00 r
  m1/Z (MUX21H)                        1.00      11.00 r
  u1/S (FA1)                           1.00      12.00 f
  a_b/Z (AN2)                          1.00      13.00 f
  a_b (out)                            0.00      13.00 f
  data arrival time                              13.00

  max_delay                           15.00      15.00
  output external delay                0.00      15.00
  data required time                             15.00
-----------------------------------------------------------
  data required time                             15.00
  data arrival time                             -13.00
-----------------------------------------------------------
  slack (MET)                                     2.00
```

```
 True-delay Input Vector
  ---------------------------------------
  b (in)                                r
  cond (in)                             f
  a (in)                                r
  ---------------------------------------
  True-delay conclusions:
-------------------------------------------------
  1. A path of length 13.00 (slack 2.00) was proven true.
The search completed, so there are no true paths that
have worse slack.
-------------------------------------------------
```

Changing the Backtrack Limit

To change the backtrack limit, use the `true_delay_prove_true_backtrack_limit` variable.

When the search algorithm reaches this backtrack limit, it stops searching for true paths and saves the longest true path found. This path is proved true, but because the search reached the backtrack limit, longer true paths might exist but are not found. The proved true path provides a lower bound on the delay of the design. The algorithm then goes to a second phase in which it tries to prove paths false, starting with the structurally longest path and working downward. This second phase provides a tighter upper bound on the true delay of the design.

Generally, the prove-false phase reaches the backtrack limit. When both phases are complete, the true delay is between the lower (prove-true) value and the upper (prove-false) value.

Changing the Prove-False Backtrack Limit

To change the prove-false backtrack limit, use the `true_delay_prove_false_backtrack_limit` variable.

For almost all real designs, the search algorithm completes the search with the default settings and finds the longest true path. However, some designs are more difficult because they have millions of false paths close to the delay value of the longest true path. For these designs, whether you are proving true (going up from 0.0) or proving false (going down from the structurally longest path), the algorithm eventually gets stuck because of all the false paths. At this point, the algorithm has useful information (a lower and upper bound on the true delay of the design). If you want the exact answer, set the backtrack limit very high so that the search algorithm continues processing until it completes the search.

When PrimeTime reaches the backtrack limit, the `report_timing` command prints conclusions at the end of the report that explain the results, as shown in the following example.

```
True-delay conclusions:
  ---------------------------------------------------------
  1. A path of length 42.00 (slack -42.00) was proven true.
     The search completed, so there are no true paths that
     have worse slack.
  ---------------------------------------------------------
True-delay conclusions:
---------------------------------------------------------
  1. A path of length 23.00 (slack -23.00) was proven true.
     However, The search reached the backtrack limit of 1000,
     so there may be other true paths that have worse slack
     but were not found. You may want to increase the prove-
     true limit by increasing
     true_delay_prove_true_backtrack_limit.

  2. All paths of length 36.60 or longer were proven false.
     To get a tighter bound you can increase the prove-false
     limit by increasing
     true_delay_prove_false_backtrack_limit.
---------------------------------------------------------
```

Using the True Delay Function

The `-true_threshold` option of the `report_timing` command provides faster yes or no answers than the `-true` option. For example, if you want to know whether a path of length 18.0 or greater exists, define a true threshold of 18.0. Defining a true threshold accelerates the algorithm in two ways:

- It does not consider any paths shorter than the defined value.
- If it finds a path of this length or greater, it stops searching immediately.

The true delay algorithm works by trying to find an input vector that sensitizes the current path. The input vector applies rising or falling values at time zero to a design previously initialized to all X-state values. The time at which the last output transitions from X becomes the delay of the design for that vector. PrimeTime propagates the rising and falling values through the design using timed simulation and the floating delay sensitization criteria.

Long Path Example

In a carry-bypass adder, the long path of the carry chain is made into a false path through the addition of logic to provide a shorter path to the output (in the paths where the carry would propagate down the entire chain).

Examples

For a carry-bypass adder design, the `report_timing` command reports the carry chain as the longest path. In this example, particularly notice the lines that are boldface. To avoid getting this type of report, you can declare the long path to be a false path.

```
pt_shell> report_timing

****************************************
Report : timing
Design : false12cbp2
****************************************

  Startpoint: s (input port)
  Endpoint: x[11] (output port)
  Path Group: default
  Path Type: max

  Point                                 Incr       Path
  -----------------------------------------------------
  input external d                      0.00       0.00 r
  s (in)                                0.00       0.00 r
  U27/z1937 (*GEN*22                    1.00       1.00 f
  U257/z2167 (*GEN*21                   1.00       2.00 f
  U233/z2143 (*GEN*20                   1.00       3.00 f
  U234/z2144 (*GEN*205)                 1.00       4.00 r
  U239/z2149 (*GEN*185)                 1.00       5.00 r
  U108/z2018 (*GEN*48)                  1.00       6.00 r

  U199/z2109 (*GEN*270)                 1.00      37.00 r
  U200/z2110 (*GEN*46)                  1.00      38.00 r
  U167/z2077 (*GEN*129)                 1.00      39.00 r
  U1/z1911 (*GEN*283)                   1.00      40.00 r
  x[11] (out)                           0.00      40.00 r
  data arrival time                               40.00

  max_delay                             0.00       0.00
  output external delay                 0.00       0.00
  data required time                               0.00
  -----------------------------------------------------
  data required time                               0.00
  data arrival time                              -40.00
  -----------------------------------------------------
  slack (VIOLATED)                               -40.00

pt_shell> true_delay_prove_true_backtrack_limit = 1000000
1000000

pt_shell> report_timing -true

****************************************
Report : timing
         -true
```

```
Design : false12cbp2
**************************************

  A True path:

  Startpoint: s (input port)
  Endpoint: y[11] (output port)
  Path Group: default
  Path Type: max

                                    Incr       Path
  -----------------------------------------------------
  input external d                  0.00       0.00 f
  s (in)                            0.00       0.00 f
  U27/z1937 (*GEN*22                1.00       1.00 r
  U248/z2158 (*GEN*6                1.00       2.00 f
  U231/z2141 (*GEN*2                1.00       3.00 r
  U232/z2142 (*GEN*21               1.00       4.00 f
  U240/z2150 (*GEN*99)              1.00       5.00 r
  U108/z2018 (*GEN*48)              1.00       6.00 f

  U198/z2108 (*GEN*272)             1.00      20.00 f
  U199/z2109 (*GEN*270)             1.00      21.00 r
  U200/z2110 (*GEN*46)              1.00      22.00 f
  U167/z2077 (*GEN*129)             1.00      23.00 r
  U13/z1923 (*GEN*82)               1.00      24.00 f
  y[11] (out)                       0.00      24.00 f
  data arrival time                           24.00

  max_delay                         0.00       0.00
  output external delay             0.00       0.00
  data required time                           0.00
  -----------------------------------------------------
  data required time                           0.00
  data arrival time                          -24.00
  -----------------------------------------------------
  slack (VIOLATED)                           -24.00
  True-delay Input Vector
  ---------------------------------------
  c[11] (in)                           r
  a[11] (in)                           r
  d[11] (in)                           f
  b[11] (in)                           f
  c[10] (in)                           r
  a[10] (in)                           r
  d[10] (in)                           f
  b[10] (in)                           f
  d[9] (in)                            r
  b[9] (in)                            r
  c[9] (in)                            f
  a[9] (in)                            f
  d[8] (in)                            r
  b[8] (in)                            r
```

```
 c[6] (in)                              f
 a[6] (in)                              f
 c[5] (in)                              r
 a[5] (in)                              r
 c[4] (in)                              r
 a[4] (in)                              r
 d[4] (in)                              f
 b[4] (in)                              f
 c[3] (in)                              r
 a[3] (in)                              r
 d[3] (in)                              f
 b[3] (in)                              f
 c[2] (in)                              r
 a[2] (in)                              r
 d[2] (in)                              f
 c[0] (in)                              r
 a[0] (in)                              r
 d[0] (in)                              r
 b[0] (in)                              r
 c[1] (in)                              f
 a[1] (in)                              f
 d[1] (in)                              r
 b[1] (in)                              f
 s (in)                                 f
 ...
 ---------------------------------------

 True-delay conclusions:
 -------------------------------------------------------
 1. A path of length 24.00 (slack -24.00) was proven true.
    The search completed, so there are no true paths that
    have worse slack.
-------------------------------------------------------
```

Asynchronous Logic Analysis

To simplify the timing verification designs with asynchronous logic, isolate the asynchronous logic into separate blocks, then disable the timing of these blocks.

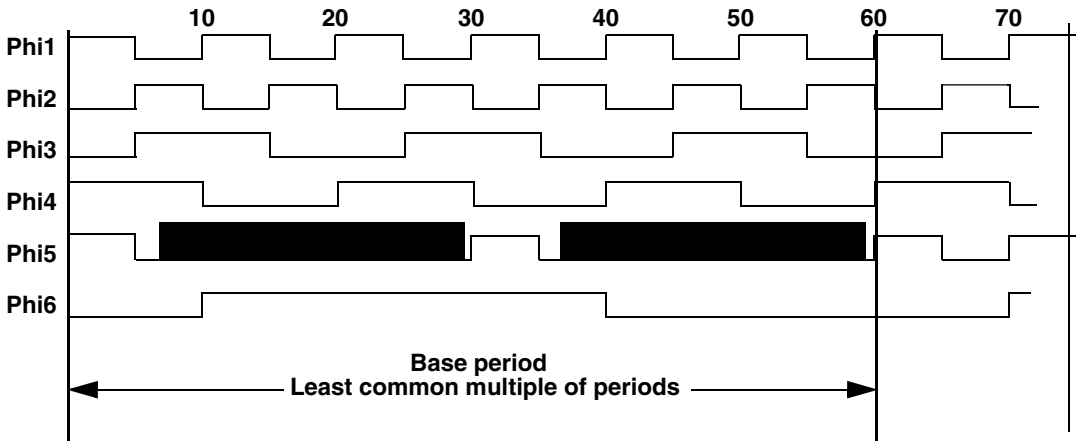
PrimeTime does not support self-timed asynchronous logic where no global clock is used. You can isolate this type of logic in a level of hierarchy, then use full-timing gate-level simulation to verify valid timing and functionality.

PrimeTime can analyze designs:

- With no combinational feedback loops; loops containing flip-flops or latches are adequate (combinational feedback loops are automatically broken)
- Without unclocked memory elements, such as RS latches

- With a single clock or multiple clocks fanning in to each register clock pin
- With known and fixed phase relationship between the clocks at the start and end registers of every path (Interacting clocks must have a single base period over which all clock waveforms repeat—see Figure 5-12.)

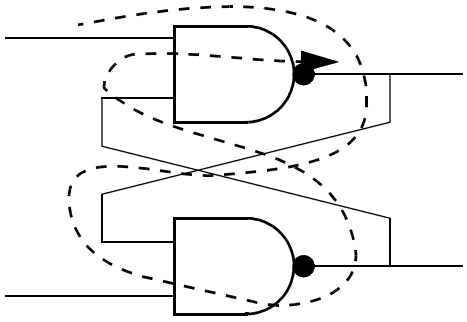
Figure 5-12 Base Period of Clocks



Combinational Feedback Loop Breaking

A combinational feedback loop is a path that can be traced though combinational logic back to its starting point. Figure 5-13 shows an example. To analyze such a path, PrimeTime must break the loop (stop tracing the path) at some point within the loop. You can check a design for the presence of combinational feedback loops with the command `check_timing -include loops`.

Figure 5-13 Combinational Feedback Loop



By default, PrimeTime identifies each feedback loop and disables one of the timing arcs of the loop, such as the timing arc from one input to one output of a NAND gate in the loop. In some cases, this approach can result in some real paths not being reported because the paths are broken by the disabled arcs.

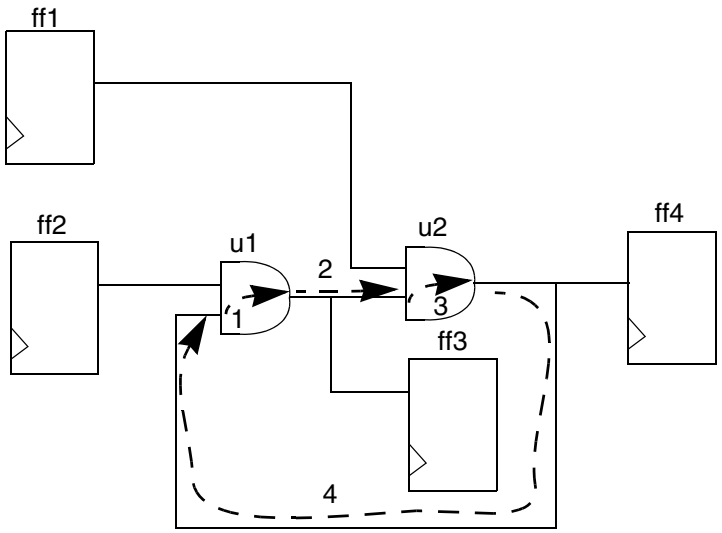
Dynamic Loop Breaking

An optional technique called dynamic loop breaking guarantees that all valid paths of the design will be reported. To enable dynamic loop breaking, set the `timing_dynamic_loop_breaking` variable to true. By default, the variable is set to false and PrimeTime does static loop breaking.

If the scope of the design that must be updated is large or if the loops are complex, using dynamic loop breaking might significantly increase the runtime or memory usage for a timing update.

Figure 5-14 shows an example where dynamic loop breaking helps to ensure that all paths get reported. In this example, no timing arc of the combinational loop can be broken without a valid path of the design also being broken. Arcs #1 and #4 cannot be broken because breaking them would break the valid path ff1 – u2 – u1 – ff3. Arcs #2 and #3 cannot be broken because breaking them would break the valid path ff2 – u1 – u2 – ff4.

Figure 5-14 Dynamic Loop Breaking



Specifying Loop Break Points

You can have multiple places in a feedback loop that are broken. If you want a feedback loop to be broken at a different timing arc from the one selected by default, use the `set_disable_timing` command to explicitly break the loop at the desired point.

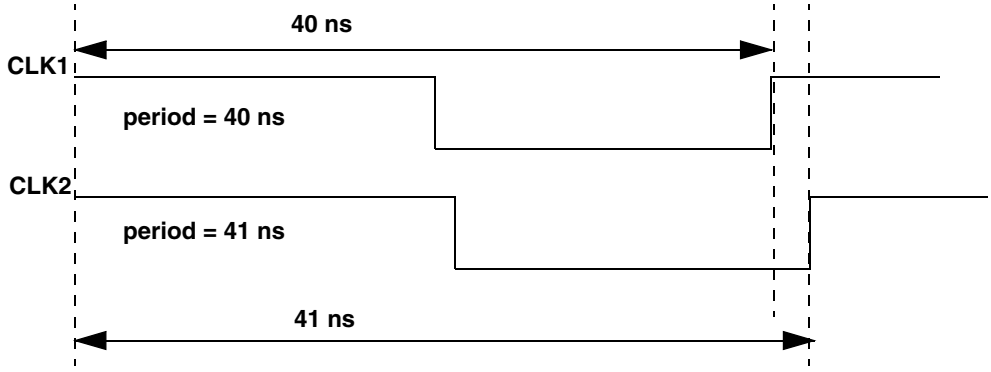
Design Compiler also uses loop breaking to analyze timing. The points at which Design Compiler and PrimeTime break a loop can be different, possibly leading to different timing results. If you want to ensure consistent loop breaking between the two tools, set the `timing_keep_loop_breaking_disabled_arcs` variable to true in PrimeTime. In that case, PrimeTime inherits the loop-breaking choices from the .ddc or .db file generated by Design Compiler. By default, this variable is set to false. After changing this variable setting, a timing update is necessary to see the change.

The report generated by `report_disable_timing` distinguishes between arcs disabled by PrimeTime and those disabled by inheritance from the .ddc or .db file. For more information, see the man page for the command.

Unrelated Clocks

Sometimes a design has paths between unrelated clocks. Unrelated clocks have different frequencies that do not have a reasonable base period. PrimeTime attempts to find a base period and phase relationship anyway, which typically is not useful (see Figure 5-15).

Figure 5-15 Unrelated Clock Waveforms



The clocks shown in Figure 5-15 do not expand to a reasonable base period, and the resulting setup requirement is quite restrictive. In this case, you might want to verify the timing with dynamic simulation rather than use PrimeTime.

Example

To exclude asynchronous paths from static timing analysis to improve runtimes and avoid false violations, enter

```
pt_shell> set_false_path -from [get_clocks clk40] \
              -to [get_clocks clk41]
```

In some cases, you might know the required minimum and maximum path delays for the combinational logic between two registers of unrelated clocks. In these cases, specify the path delay constraint using the `set_max_delay` and `set_min_delay` commands for that path.

Three-State Bus Analysis

By default, PrimeTime checks for setup and hold violations in three-state bus designs. It checks the worst delay path to ensure that the latched signals are stable and are not unknown (X) values. This ensures proper timing checks for transient bus contention and transient floating bus conditions.

PrimeTime considers that a disabling transition on a three-state cell can cause either a 1 or 0 delay on the output. By default, PrimeTime considers three_state_disable and three_state_enable arcs (as defined in the library) during path tracing. Although this is different from propagating an X value, the effect for static timing is the same as for propagating an X value.

Limitations of the Checks

The three-state bus checks have these limitations:

- PrimeTime checks the potential for setup or hold errors due to bus contention or float conditions. PrimeTime does not check logical and power violations for bus contention or float conditions.
- The analysis is pessimistic; some reported violations might never happen because of state dependencies.
- In some simulators, Z does not propagate to X until after the charge decay time. PrimeTime does not model this effect; it uses the gate delay equations to propagate Z and X. This might be pessimistic compared to some simulators.

Disabling the Checks

If you know that bus contention or floating buses do not occur in your design, you can disable these checks by setting these variables to true:

`timing_disable_bus_contention_check`

When set to true, this variable disables propagation of maximum delay along three_state_disable timing arcs and minimum delay along three_state_enable arcs. These checks are valid only during transient bus contention. The default is false.

`timing_disable_floating_bus_check`

When set to true, this variable disables propagation of minimum delay along three_state_disable timing arcs and maximum delay along three_state_enable arcs. These checks are valid only during floating bus conditions. The default is false.

Bus Contention

Some designs rely on a bus configuration in which many three-state drivers control the bus. In most designs, no two drivers with different logical outputs can be simultaneously enabled at the steady state (when the enable pins of the three-state drivers assume their steady-state values for any clock cycle). When multiple drivers drive the same bus, it is called bus contention.

Although you can design a circuit so that no steady-state bus contention occurs, a design might contain transient bus contention conditions. Transient bus contention conditions occur during the transition of the bus control from one driver to another. During this short transient period, the logical value for the bus is unknown (X value) if the drivers contending for control of the bus are imposing conflicting logical values. For static timing analysis, setup checks ensure that this X value, assumed during transient bus contention, is not latched. The setup check is measured from the time the bus becomes stable.

Floating Buses

A floating bus condition can arise for three-state bus configuration designs. A floating bus condition occurs when no driver is enabled. In this case, the bus immediately gets an unknown (X) value. For static timing analysis, hold checks ensure that this X value, assumed when the bus switches to the floating mode, is not latched. The hold checks are measured to the time the bus becomes floating.

Three-State Buffers

Two timing arc types are used to describe timing behavior of three-state buffers. These timing arc types are defined in the library.

three_state_disable timing arc

Specifies the time the three-state pin takes to go from a high or low state to the high-impedance state.

three_state_enable timing arc

Specifies the time a three-state pin takes to go from the high-impedance state to a high state or low state.

Performing Transient Bus Contention Checks

Figure 5-16 shows a circuit used to describe how PrimeTime performs transient bus contention checks and floating bus checks.

Figure 5-16 Circuit Example

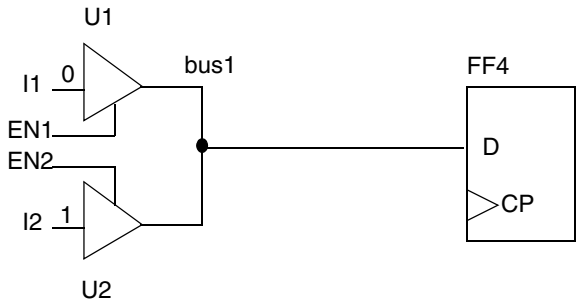
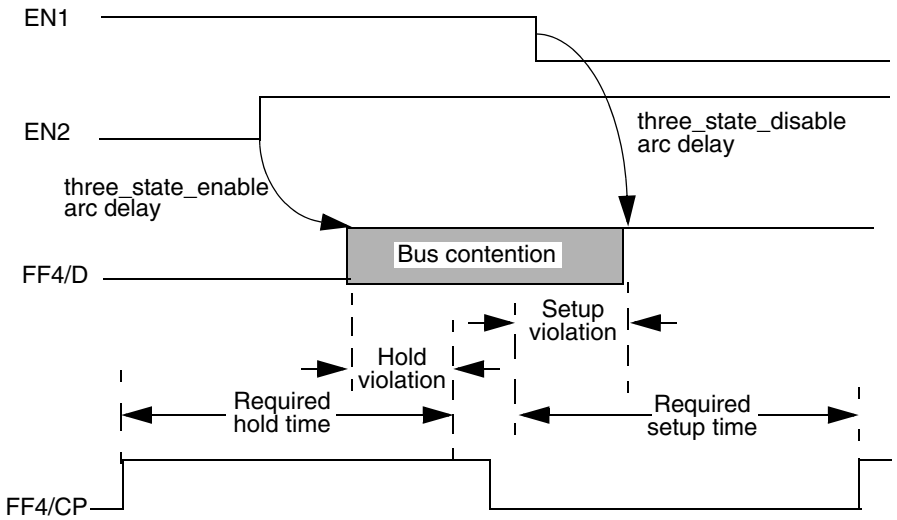


Figure 5-17 shows how PrimeTime performs transient bus contention checks.

Figure 5-17 Transient Bus Contention Check



In Figure 5-17, if EN1 turns off after EN2 turns on, there is a potential bus contention for some time. The signal arriving at FF4/D is X state until the contention is over and the new bus value propagates along the path. If I1=0 and I2=1, the waveforms for the circuit in Figure 5-16 are as shown in Figure 5-17.

The setup violation for FF4/CP is seen only if the `three_state_disable` arc delay is considered. A transition to the Z state must be propagated as a possible transition to logic 0 or logic 1 to find all possible cases. A similar situation occurs for the hold check. The hold violation is seen only if the `three_state_enable` arc delay is considered.

This bus contention region is bounded by the minimum `three_state_enable` arc delay of any bus driver from one side and by the maximum `three_state_disable` arc delay from the other side. If you know that such bus contention regions can never occur, you can disable checking for both setup and hold violations that occur due to this bus contention region.

You can disable the checks by setting the `timing_disable_bus_contention_check` variable to `true`, causing PrimeTime to ignore the three_state_enable arc delay for hold violation checking and the three_state_disable arc delay for setup violation checking.

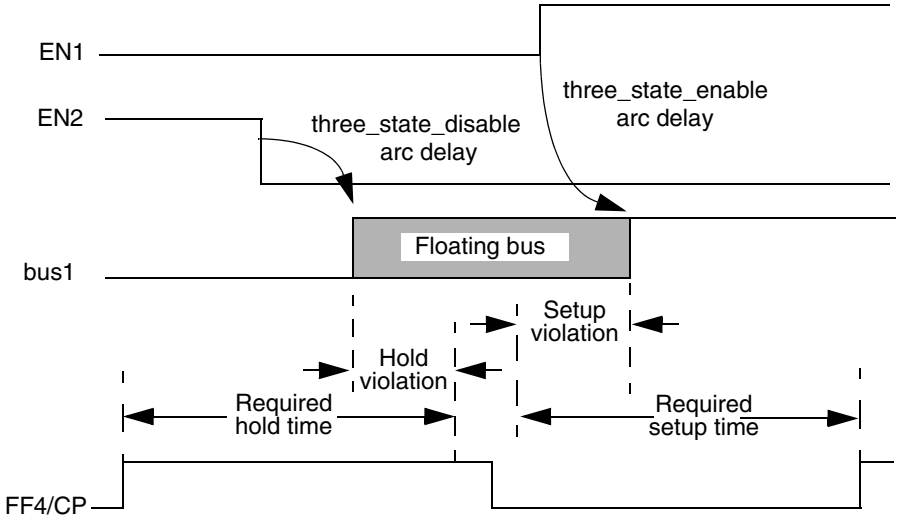
Even when you set this variable to `true`, PrimeTime considers the `three_state_enable` arc delay for setup violation checking and the `three_state_disable` arc delay for hold violation checking. This occurs during a floating bus region, which is described in the next section.

Performing Floating Bus Checks

A floating bus occurs when the bus is at a valid logic value and then begins to float. Timing analysis ensures that the float does not propagate an X to the register data pin until after the hold check. PrimeTime assumes the same setup and hold relationships as for any other data signals.

Figure 5-18 shows another situation that can arise that yields a transient floating bus condition.

Figure 5-18 Floating Bus Contention Check



In Figure 5-18, the hold violation for FF4/CP is seen only if the three_state_disable arc delay is considered. A transition to the Z state must be propagated as a possible transition to logic 0 or logic 1 to find all possible cases. A similar situation happens for the setup check. The setup violation is seen only if the three_state_enable arc delay is considered.

Because the `three_state_enable` arc delays are considered, by default PrimeTime checks for all setup and hold violations that occur due to the floating bus region. The floating bus region is bounded by the minimum `three_state_disable` arc delay of any bus driver from one side and by the maximum `three_state_enable` arc delay from the other side. PrimeTime ignores charge decay here (it assumes that the logical value for the bus immediately becomes unknown—(X)—when the bus is floating).

If you know that such floating bus regions can never occur, you can disable checking for both setup and hold violations that occur due to this bus contention region. To disable the checks, set the `timing_disable_floating_bus_check` variable to `true`. In this case, PrimeTime ignores the three_state_disable arc delay for hold violations checking and the three_state_enable arc delay for setup violations checking. Even when you set this variable to `true`, the `three_state_enable` arc delay is considered for setup violation checking and the `three_state_disable` arc delay is considered for hold violation checking. This occurs during a bus contention region, which was described in the previous section.

Fast Multidrive Delay Analysis

In large designs with annotated parasitics, the runtime for performing RC delay calculation can be large for massively multidriven networks. The primary bottleneck in this process is the runtime required to compute the trip times for measuring delays and slews at the load pins.

For massively multi-driven networks with homogeneous drivers (drivers with similar input skews, slews, and operating conditions), a calculation mode is available that significantly reduces the runtime. This mode shorts all of the driver nodes together with a network of resistors and uses a single driver model at the first driver node for all waveform calculations. The drive strength of the single driver is scaled up to be equivalent to the whole set of original drivers. The delay calculation results for the single driver are copied to all of the other drivers, including delay, slew, driver model parameters, and effective capacitance.

Usage of the fast multidrive analysis mode is based on the `rc_driver_count_threshold_for_fast_multidrive_analysis` variable. This variable is an integer that specifies the maximum number of network drivers that are handled without invoking the fast multidrive analysis mode. The fast mode is invoked when the number of drivers exceeds the variable setting. By default, the variable is set to `9`. Setting the variable to `0` disables the mode entirely (the same as setting it to an extremely large number).

When PrimeTime uses the fast multidrive analysis mode, it generates an RC-010 warning message that reports the number of drivers, the number of loads, the input slew and input skew spreads, and the matching or mismatching of driver library timing arcs and driver operating conditions. This information can help determine the accuracy of the analysis.

The accuracy of this mode is best when the driver library arcs are the same and operate under the same context (operating conditions, input slew, input skew, and so on), and operate with the same network impedance. These conditions are often typical of mesh networks. Any differences in operating conditions and input slews can cause small differences in output delay and output slew. For accuracy, the differences in input skews should be small compared to the delays through the network. For maximum accuracy, you can either disable the fast multidrive analysis mode or annotate delays and slews onto the network.

Note:

For more information, see“Parallel Driver Reduction” on page 5-39 and “Reducing SDF for Clock Mesh/Spine Networks” in Chapter 8.

Parallel Driver Reduction

Because clock signals must typically drive a large number of loads, clocks are often buffered to provide the drive strength necessary to reduce clock skew to an acceptable level. A large clock network might use a large number of drivers operating in parallel. These drivers can be organized in a “mesh” or “spine” pattern to distribute the signal throughout the chip.

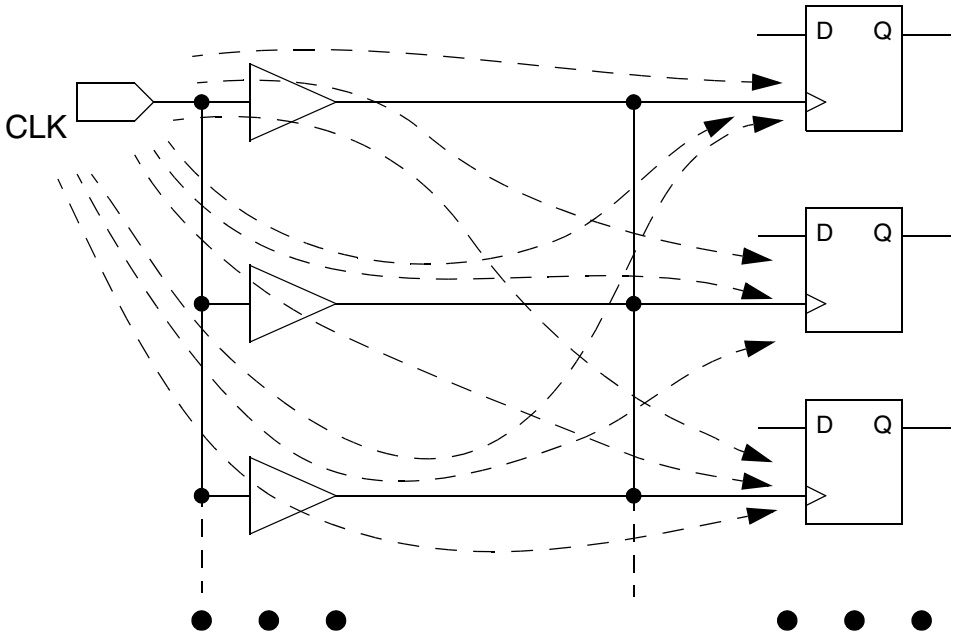
If a design has 1,000 drivers in parallel driving 1,000 loads, PrimeTime must keep track of one million different timing arcs between the drivers and loads. Timing analysis of such a network can consume a large amount of CPU and memory resources.

For better performance, PrimeTime can reduce the number of timing arcs that must be analyzed. The reduction process is illustrated in Figure 5-19. When the reduction feature is enabled, PrimeTime selects one driver in a parallel network and analyzes the timing arcs through that driver only.

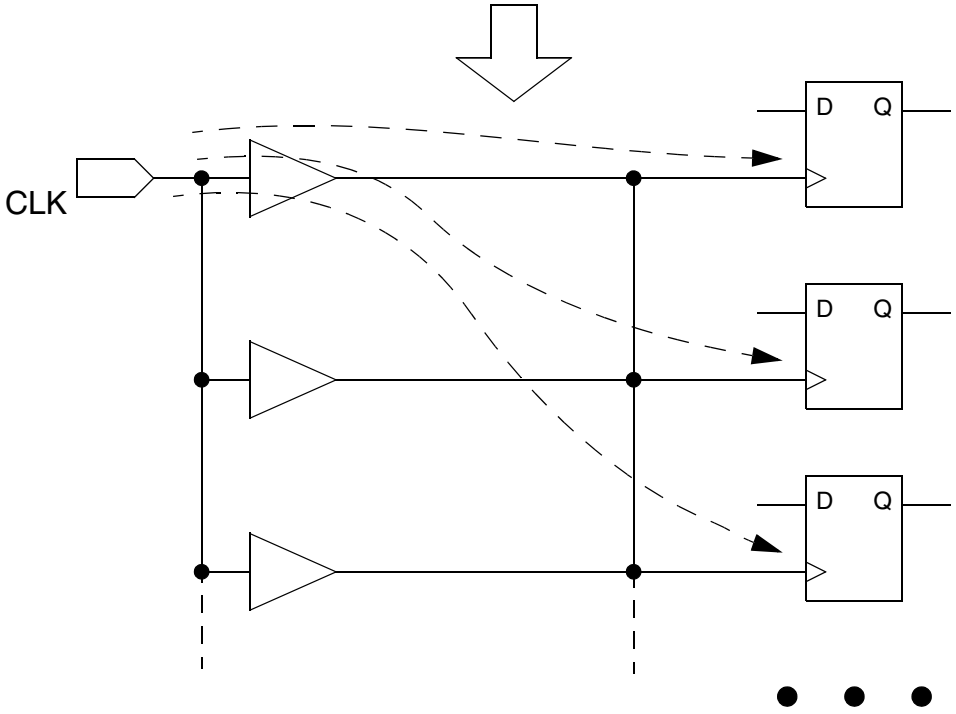
Note:

For related information, see “Fast Multidrive Delay Analysis” on page 5-38 and “Reducing SDF for Clock Mesh/Spine Networks” in Chapter 8.

Figure 5-19 Parallel Driver Reduction




```
pt_shell> set timing_reduce_multi_drive_net_arcs true
pt_shell> set timing_reduce_multi_drive_net_arcs_threshold 1000
pt_shell> link_design
```



Invoking Parallel Driver Reduction

Parallel driver reduction is controlled by the following variables:

- `timing_reduce_multi_drive_net_arcs`
- `timing_reduce_multi_drive_net_arcs_threshold`

The first of these two variables can be set to `true` or `false` to enable or disable parallel driver reduction. By default, it is set to false and no reduction is done. When it is set to true, during design linking, PrimeTime checks for the presence of nets with multiple drivers. If it finds a net with driver-load combinations exceeding the threshold specified by the threshold variable, it reduces the number of timing arcs associated with the drivers of that net.

The threshold variable specifies the minimum number of timing arcs that must be present to trigger a reduction. By default, it is set to 10,000, which means that PrimeTime reduces the timing arcs of a net if the number of drivers multiplied by the number of loads on the net is more than 10,000. In typical designs, this large number only occurs in clock networks.

PrimeTime performs driver reduction for a net when all of the following conditions are true:

- Number of driver-load combinations is more than the variable-specified threshold (10,000 by default)
- Driver cells are nonsequential library cells (not flip-flops or latches and not hierarchical)
- All drivers of the net are instances of the same library cell
- All the drivers are connected to the same input and output nets

To expand a reduced network to its original form or to perform driver reduction with a different threshold, you must relink the design.

Working With Reduced Drivers

After layout is complete and detailed parasitic information becomes available, it is not possible to annotate this information on a reduced network. To get accurate results, you can use an external simulator such as SPICE to get detailed delay information for the network. Then you can annotate the clock latency values and transition times on the individual clock pins of the sequential cells, while still allowing PrimeTime to treat the reduced network as ideal, with zero delay. This technique provides reasonably accurate results, while being very efficient because the clock network timing only needs to be analyzed once, even for multiple PrimeTime analysis runs.

If you back-annotate the design with the `read_sdf` command, any annotations on the reduced timing arcs are ignored. PrimeTime issues a PTE-048 message when this happens. You can suppress these messages with the following command:

```
pt_shell> suppress_message PTE-048
```

If you write out SDF with the `write_sdf` command, no interconnect delays are generated for the reduced network.

Reducing parallel drivers only affects the timing arcs analyzed by PrimeTime, not the netlist. Therefore, it does not affect the output of the `write_changes` command. For information about RC delay calculation with massively multidriven networks, see "Fast Multidrive Delay Analysis" on page 5-38.

Data-to-Data Checking

PrimeTime can perform setup and hold checking between two data signals, neither of which is defined to be a clock, at any two pins in the design. This feature can be useful for checking the following types of timing constraints:

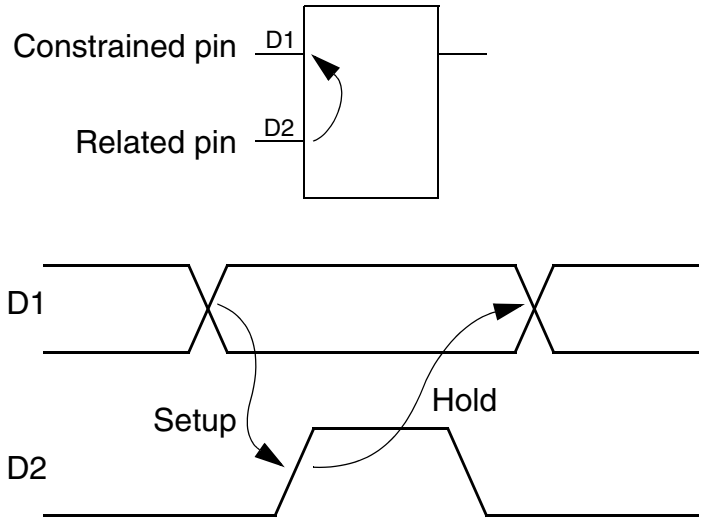
- Constraints on handshaking interface logic
- Constraints on asynchronous or self-timed circuit interfaces
- Constraints on signals with unusual clock waveforms that cannot be easily specified with the `create_clock` command
- Constraints on skew between bus lines
- Recovery and removal constraints between asynchronous preset and clear input pins

A timing constraint between two data (nonclock) signals is called a nonsequential constraint. You can define such checks in PrimeTime by using the `set_data_check` command, or you can define them for a library cell by setting nonsequential constraints for the cell in Library Compiler. You should use data checks only in situations such as those described above. Data checks should not be considered a replacement for standard sequential checking.

Data Check Examples

Figure 5-20 shows a simple example of a cell that has a nonsequential constraint. The cell has two data inputs, D1 and D2. The rising edge of D2 is the active edge that might be used to latch data at D1. Pin D1 is called the constrained pin and Pin D2 is called the related pin. In a sequential setup or hold check, pin D2 would be considered the clock pin. However, for any of a number of reasons, it might be desirable to consider the signal at D2 a data signal, and not define it to be a clock.

Figure 5-20 Simple Data Check Example



In this example, the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge. If these nonsequential constraints are not already defined for the library cell, you can define them in PrimeTime. To do so, use commands similar to the following:

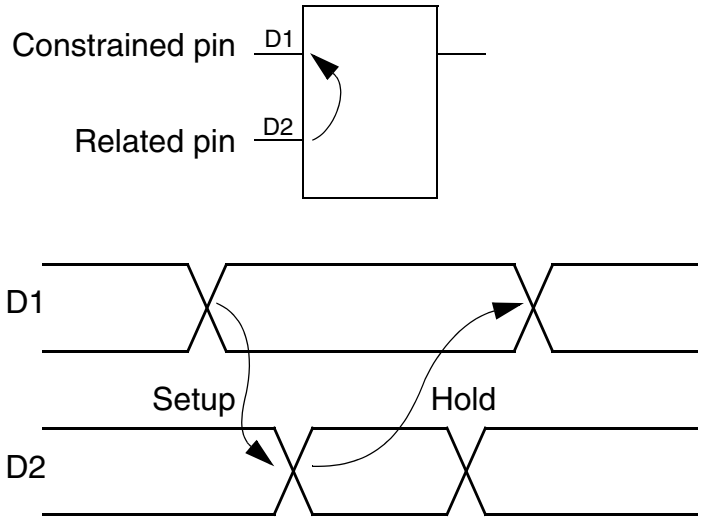
```
pt_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
pt_shell> set_data_check -rise_from D2 -to D1 -hold 6.0
```

The “from” pin is the related pin and the “to” pin is the constrained pin. If the data checks apply to both rising and falling edges on the related pin, use `-from` instead of `-rise_from` or `-fall_from`, as shown in the following example:

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

The resulting timing checks are illustrated in Figure 5-21.

Figure 5-21 Data Checks on Rising and Falling Edges

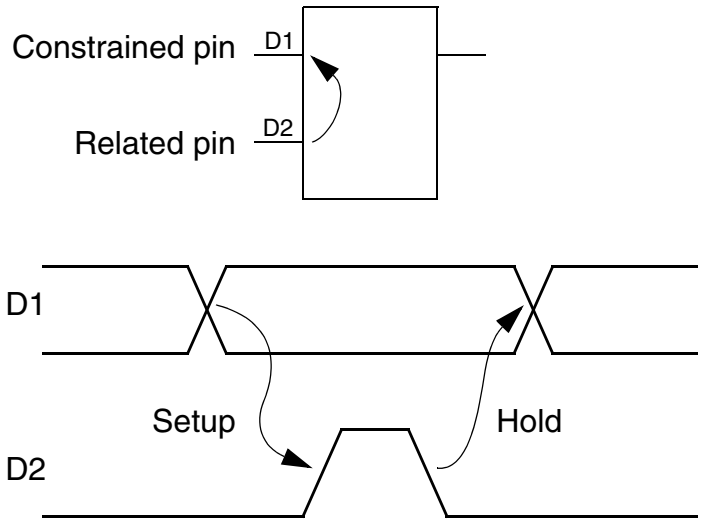


You can define a no-change data check by specifying only a setup check from the rising edge and a hold check from the falling edge:

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

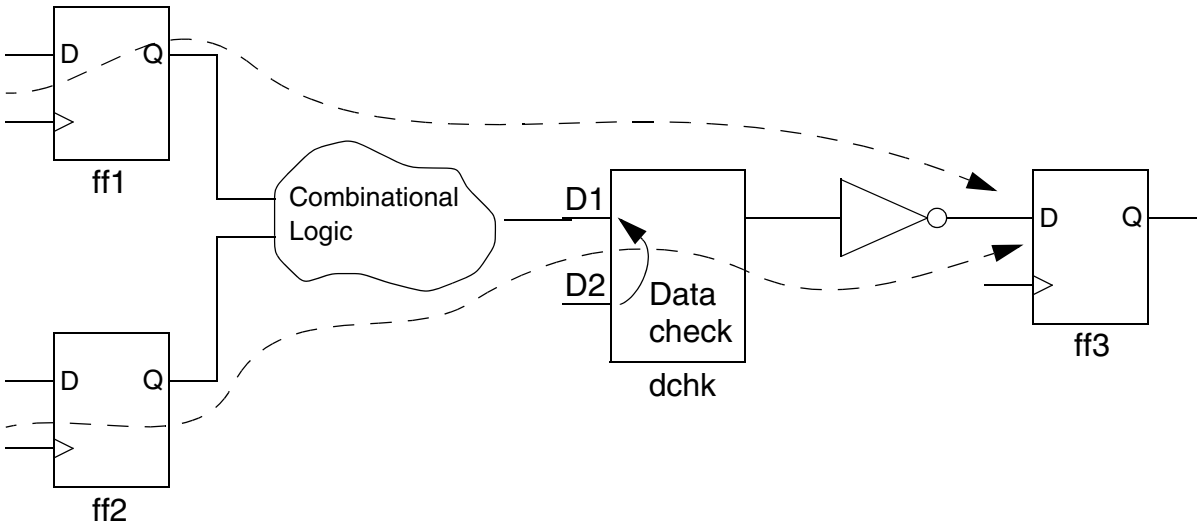
PrimeTime interprets this as a no-change check on a positive-going pulse. The resulting timing check is illustrated in Figure 5-22.

Figure 5-22 No-Change Data Check



Data checks are nonsequential, so they do not break timing paths. For example, in Figure 5-23, the data check between D1 and D2 does not interrupt the timing paths shown by the dashed-line arrows. If you were to define the signal at D2 to be a clock, the check would be sequential and the paths would be terminated at D1.

Figure 5-23 Timing Paths Not Broken by Data Checks



You can specify a data check pin as a path endpoint for the `report_timing` command. In that case, PrimeTime reports the data checks that apply to the pin. For example, for the circuit shown in Figure 5-23, `report_timing -to dchk/D1` generates a data check report, whereas `report_timing -through dchk/D1` generates a timing report on standard paths that pass through the specified pin.

To remove data checks set with the `set_data_check` command, use the `remove_data_check` command. For more information about either command, see the specific man page.

Data Checks and Clock Domains

In a data check, signals arriving at a constrained pin or related pin can come from different clock domains. PrimeTime checks the signal paths separately and puts them into different clock groups, just like standard sequential checks.

If the related pin has signals from multiple clock domains, you might want to specify which clock domain to analyze at that pin for the data check. To specify a clock domain to analyze, either use the `-clock` *`clock_name`* option of the `set_data_check` command, or disable all clocks other than the clock of interest.

Library-Based Data Checks

PrimeTime performs data checking for any cell that has nonsequential timing constraints defined in the library cell, as long as the signal at the related pin is not defined to be a clock in PrimeTime. If the signal is defined to be a clock, PrimeTime converts the nonsequential checks to sequential checks and does not block this clock signal from further propagation. If a combinational arc from the related pin exists (such as with an integrated clock gating cell), the clock is free to continue propagation down this arc.

In Library Compiler, you define nonsequential constraints on a cell by specifying a related pin and by assigning the following `timing_type` attributes to the constrained pin:

```
non_seq_setup_rising
non_seq_setup_falling
non_seq_hold_rising
non_seq_hold_falling
```

For more information about defining nonsequential constraints in Library Compiler, see the Library Compiler user guides.

Defining nonsequential constraints in the library cell results in a more accurate analysis than using the `set_data_check` command because the setup and hold times can be made sensitive to slew of the constrained pin and the related pin. The `set_data_check` command is not sensitive to slew.

To specify which clock domain to use at the related pin for data checks defined in library cells, you can use the `set_data_check ... -clock` *`clock_name`* command. The `remove_data_check` command does not remove data checks defined in library cells.

Data Propagation Through Generated Clocks

To support the use of multiple levels of undefined generated clocks, use the `timing_propagate_through_unclocked_registers` variable. When this variable is set to `true`, PrimeTime propagates data through a flip-flop or level-sensitive latch when there is no defined clock driving the clock pin. When the variable is set to false (the default), the lack of a defined clock prevents data propagation through a flip-flop or latch.

Netlist Editing

PrimeTime provides commands for editing the netlist. Editing the netlist is performed to address timing issues without having to iterate through logic synthesis. You might want to edit the netlist because of a timing violation caused by a large net capacitance, which in turn is caused by a large net fanout. For addressing common timing problems, you can increase

the size of the driver cell using the `size_cell` command, or insert a buffer in the net to increase its drive using the `insert_buffer` command. These capabilities are useful, especially late in a design cycle when the logic functionality is frozen, and only small edits can be made.

For a list of editing commands, see Table 5-1.

Table 5-1 Netlist Editing Commands

Object	Task	Command
Buffer	Inserting a buffer	`insert_buffer`
	Removing a buffer	`remove_buffer`
Cell	Changing the size (or drive strength) of a cell	`size_cell`
	Creating a cell	`create_cell`
	Renaming a cell	`rename_cell`
	Removing a cell	`remove_cell`
Net	Adding a new net to the design	`create_net`
	Connecting nets to pins in the design	`connect_net`
	Disconnecting nets to pins in the design	`disconnect_net`
	Renaming a net	`rename_net`
	Removing a net	`remove_net`

To perform netlist editing, all you need is a linked design. You can edit any hierarchical cell in the netlist.

Note:

It is recommended that you not perform substantial edits to the design for the purposes of adding new logic, removing out-of-date logic, or fixing functional errors. To perform moderate design changes and to generate new netlists, use Design Compiler or a similar tool.

When the editing of the netlist is complete, you can see the impact of the changes using the standard timing analysis and reports. If you want to preserve a record of the changes that you have made, use the `write_changes` command to save the changes to a disk file. The changes recorded are a sequence of editing commands.

If you want the library file names to be prepended to the library cell names in the change list arising out of netlist changes, set the `eco_write_changes_prepend_libfile_to_libcell` variable to `true`. By default this variable is set to false. You can discard the netlist changes by reading and relinking your design.

Edits to the top level of a design cannot be undone without removing the design, reading the original design again, and relinking it. However, edits to lower levels of the hierarchy can be undone by swapping in the original version of the design with the `swap_cell` command.

Automatic Uniquifying of Blocks

By editing a hierarchical block that is one instance of multiple instances in a design, you make that block unique. PrimeTime “uniquifies” that block for you automatically. For example, if the edited block is an instance of BLOCK, PrimeTime renames it BLOCK_0 (or some similar name, avoiding any names already used in the design).

Making an instance of a design in PrimeTime unique is somewhat different from other Synopsys tools because a new design is not created at the time the block becomes unique. A placeholder for the design is created, similar to the placeholder that exists when a design is removed from memory by using the `link_design -remove_sub_designs` command.

Designs created by automatic uniquifying can be listed using the `list_designs -all` command. These designs become real only if and when you relink the design.

When a block is edited, it is uniquified, along with all blocks above it up to the top of the hierarchy. Messages are generated when uniquifying occurs. For example, if you use the `size_cell` command to change the size of cell i1/low/n1, it causes the cells to be uniquified:

```
pt_shell> size_cell i1/low/n1 class/NR4P
Uniquifying 'i1/low' (low) as 'low_0'.
Uniquifying 'i1' (inter) as 'inter_0'.
Sized 'i1/low/n1' with 'class/NR4P'
1
```

In addition to uniquifying on demand, blocks are marked as having been edited. As soon as you edit a block, its parent has been edited, on up the hierarchy to the design, which itself has been edited. This information is available from a Boolean attribute, `is_edited`, available on a design and on hierarchical cells. After you use the `size_cell` command, the `is_edited` attribute is on i1/low block. For example,

```
pt_shell> get_attribute [get_cells i1/low] is_edited
true
```

Resolving Library Cells

Many of the netlist editing commands take a library cell as argument. This argument can be a library cell object or the name of a library cell. The former can be obtained using the `get_lib_cells` command. The latter can be either the full name of the library cell (such as, class/AND2), in which case there is no ambiguity about what to link the cell to; or just the base name of the library cell (such as, AND2), in which case the library cell base name must be resolved to a library. The process of resolving library cell base names to libraries is described in this section.

Library cell resolving for netlist editing commands is used primarily for netlist editing in distributed multi-scenario, where you cannot use full names of library cells because each scenario might have a different set of libraries. However, you can also use this feature in nonmulti-scenario PrimeTime. For more information about the distributed multi-scenario analysis flow, see the *PrimeTime Distributed Multi-Scenario Analysis User Guide*.

You can resolve library cells base names from the cell's current library or from a specific library you define using the `-current_library` option or `-library` option, respectively.

You can use the `-current_library` option with the commands `size_cell`, `get_alternative_lib_cells`, and `report_alternative_lib_cells`. This option instructs PrimeTime to resolve the specified library cell base name from the existing cell's currently linked library only. This option excludes PrimeTime from searching any other libraries.

The commands `size_cell`, `insert_buffer`, and `create_cell` offer the `-libraries` option, with which you can specify a list of libraries to resolve a specified library cell name. The libraries are searched in the order specified in the command. This option excludes PrimeTime from searching any other libraries. The `-libraries` option compliments the `-libraries` option with the `get_alternative_lib_cells` and `report_alternative_lib_cells`.

In the absence of these options, PrimeTime resolves library cell base names from libraries in the following order:

- Link library of the current cell for `size_cell`, `get_alternative_lib_cells`, or `report_alternative_lib_cells`
- Libraries specified by the `link_path_per_instance` variable
- Libraries specified by the `link_path` variable

You can use the `-base_names` option with the `get_alternative_lib_cells` command to cause the command to return the alternative library cells as a list of library cell base names (a string) rather than a collection.

Estimating Delay Changes

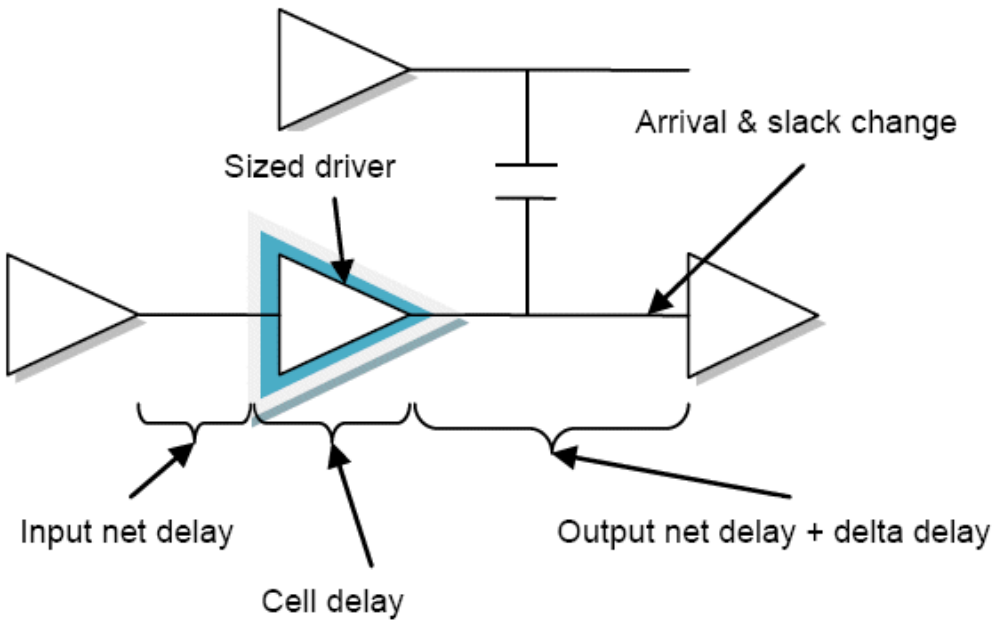
When performing ECO design modifications, it is important to fully understand the consequences of the timing changes before modifying the design. Making the change and evaluating the resulting timing can cause you to incur the cost of an incremental timing update.

To improve the efficiency of the ECO flow, you can use the `estimate_eco` command. This command quickly computes and displays the ECO alternatives based on current timing values of a particular stage. The estimate computation takes into account the incoming transition times, output loading, fanout loading, detailed parasitics, if present, and driver characteristics of the candidate cells.

The results are expected to reasonably predict, but not necessarily match, the actual timing resulting from subsequently performing that ECO command. Therefore, you should not rely on the `estimate_eco` command to get an accurate timing report because the results are not guaranteed. Rather, you should use the command to help choose the best design change among multiple options by showing the relative timing impact of each option. In particular, this command allows you to write efficient ECO scripts that can quickly choose optimal ECOs without relying on an expensive change/update/check/modify inner loop.

Two types of ECO fixing methods available with the `estimate_eco` command are sizing and buffer insertion. Both of these methods change the netlist and, as a result, change the delay values of the associated stages. For example, sizing a victim driver cell changes its crosstalk delay because of the driver resistance change. Moreover, it changes the previous stage delay and current stage delay due to input capacitance and driver resistance changes as shown in Figure 5-24.

Figure 5-24 Timing Breakdowns



When using the `estimate_eco` command, you should use the following flow:

1. Show the available options.

In this stage, you evaluate possible ECO options. For example, you might want to size up a driver cell, buf3, in a path that has a setup timing violation. To find out the effects of substituting different drivers, you would use the following command:

```
pt_shell> estimate_eco -type size_cell -max -rise

delay type : max_rise
lib cell            area      stg_delay    arrival     slack
-------------------------------------------------------------------------
mylib90/CKNXD12     19.05     0.0462       0.0462      0.1474
mylib90/INVD24      21.17     0.0265       0.0265      0.1671
mylib90/INVD20      18.35     0.0291       0.0291      0.1645
mylib90/CKNXD8      14.11     0.0610       0.0610      0.1326
mylib90/INVD3        3.53     0.1295       0.1295      0.0212
*mylib90/INVD2       2.82     0.1936       0.1936     -0.0013
mylib90/CKNXD1       2.82     0.4271       0.4271     -0.3214
```

Note:

The above report shows all possible alternative library cells and associated timings, with the asterisk (*) identifying the current cell. The rise or fall timing represents the worst timing through the cell output pins in the corresponding direction. The stg_delay column shows the delay of the inserted buffer or resized cell and the driven net. The arrival and slack columns represent the arrival time and slack at the load pins of the newly inserted buffer or resized cell.

2. Estimate the stage delay improvements.

   After deciding on the approaches for fixing the timing violation, you analyze the delay changes at the stage in which the netlist change is to occur. For example, you might want to insert a buffer to the input pin, buf6/I, and estimate how much the stage delay would change. The command syntax and sample output is as follows:

```
pt_shell> estimate_eco -type insert_buffer -inverter_pair \
          -max -rise -verbose \
          -lib_cell

cell name : buf3
current lib cell: mylib90/INVD2
buffer lib cell : mylib90/INVD2
max_rise                    current      estimate
-----------------------------------------------------------------
input net delay             0.000000     0.000000
stage delay                 0.371022     0.371022
  cell delay                0.206952     0.206952
  output net delay          0.023531     0.023531
  buffer delay              0.000000     0.071992
  delta delay               0.140539     0.140539
arrival time                0.371022     0.443014
slack                       0.123400     0.045612
```

3. Commit changes and verify.

If you are sure about the fix, you can run `size_cell`, `insert_buffer`, or both and then run the `update_timing` command to see real changes. Since the `estimate_eco` command shows only estimated delays based on current-stage information, the real delay values after the change can be different due to the interaction between previous, next, and coupled stages.

Note:

The timing update in this process cannot guarantee real silicon delays because PrimeTime does not have place and route information.

For more information about the `estimate_eco` command, including recommended methods for using the timing estimation data in scripts, see the man page.

Sizing Cells

You can replace the library cell referenced by a leaf cell in the netlist with a new cell by using the `size_cell` command. The result is a cell with new characteristics with respect to delay calculation, which results in different timing.

When you use the `size_cell` command and the targeted cell is not functionally equivalent, an error message appears. For example,

```
pt_shell> size_cell o_reg1 class/NR4P
Error: Could not size 'o_reg1' ('class/FD2') with 'class/
NR4P':
Cells not functionally equivalent. (NED-005)
Error: No changes made. (NED-040)
0
```

You can use the `get_alternative_lib_cells` command to find a compatible library cell.

```
pt_shell> get_alternative_lib_cells o_reg1
{"class/FD2P"}
```

For example, to size the cell o_reg1 class/FD2P, enter

```
pt_shell> size_cell o_reg1 class/FD2P
Sized 'o_reg1' with 'class/FD2P'
1
```

A library might contain several alternative library cells. For example, `get_alternative_lib_cells` might return this:

```
pt_shell> get_alternative_lib_cells o_reg1
{"class/FD2P1", "class/FD2P2", "class/FD2P3"}
```

It might not be obvious which cell to use for replacement. You can use the `report_alternative_lib_cells` command to obtain the slack at the outputs of the leaf cell. For example, you can use the `report_alternative_lib_cells` command for a report like this:

```
pt_shell> report_alternative_lib_cells o_reg1

****************************************
Report : alternative_lib_cells o_reg1 -delay_type max
Design : counter/0 (unknown)
Version: 2000.11
Date   : Wed Nov 15 17:53:16 2000
****************************************
Alternative                     Slack
Library Cells
-------------------------------------------------------------
class/FD2P3                      1.88(r)
class/FD2 *                     -1.02(r)
class/FD2P1                     -2.88(r)
class/FD2P2                     -2.98(r)
```

In the report, it indicates that the cell class/FD2P3 would yield a design that will produce positive slack at the output of the leaf cell. In this case, you would use the class/FD2P3 to size the leaf cell. To verify the state of the design, you can use the `report_timing` command.

Inserting Buffers

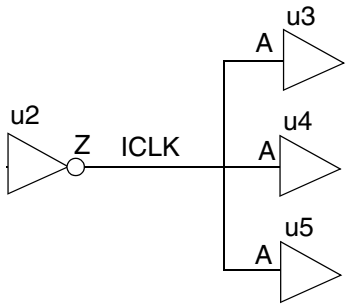
You can use the `insert_buffer` command to add a buffer at one or more pins in the netlist. To remove a buffer from the netlist, you can use the `remove_buffer` command. For more information, see the `insert_buffer` and `remove_buffer` man pages.

Note:

If you insert a buffer when parasitics are present, the parasitics may not be preserved. See the `insert_buffer` man page for more information.

In the circuit shown in Figure 5-25, pin u2/Z drives net ICLK, and ICLK has three loads: u3/A, u4/A, and u5/A.

Figure 5-25 Inserting Buffer

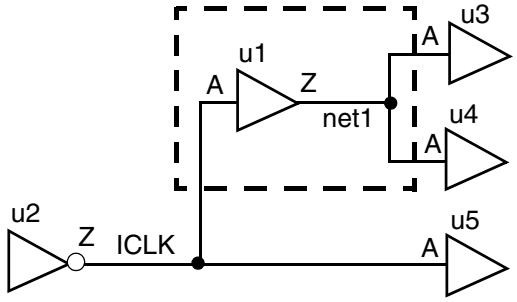


To buffer u3/A and u4/A, enter,

```
pt_shell> insert_buffer {u3/A u4/A} class/B1I
```

The `insert_buffer` command creates a new cell, u1, and a new net, net1. The input of the new buffer is always the existing net, in this case, ICLK. The output of the new buffer is always the new net, in this case, net1. The new circuit is shown in Figure 5-26.

Figure 5-26 New Circuit



To change the prefix used to name the newly inserted buffers, you can use the `eco_instance_name_prefix` variable. By default it is set to “U”.

Swapping Cells

The `swap_cell` command replaces the cells in the cell_list with the design or library cell you specify in the `new_design` option.

Before the swap, the `swap_cell` command also checks whether the pinout of the cell you are swapping out and the pinout of the design or library cell you are swapping in are the same. For every pin of the cell you are swapping out, there must be a pin with the same name in the cell you are swapping in.

The swap_cell command always relinks the part of the design that has been replaced. Do not use link_design following a swap_cell; this often undoes the work of the swap_cell command. The PrimeTime data model is instance based. When you use the swap_cell command, an instance is modified. For example, U2/U0 is an instance of design X, and you swap U2/U0 for a different design. PrimeTime did not modify the design X; it modified the instance U2/U0. Therefore, an initial link reverts to the old design.

PrimeTime does not save information about cells that were swapped; that information is maintained only until the next link. However, the fact that the swap occurred is carried in the change list for the design, and can be exported using the write_changes command.

By default, the swap_cell command restores the constraints that were on the design before the swap occurred. If you are doing multiple swaps, it is far more efficient to save the constraints using the write_script command and then use the swap_cell command with the -dont_preserve_constraints option.

When you have completed the swaps, restore the constraints by sourcing your script that you had written with the write_script command. If you are swapping one library cell for another, consider using the size_cell command, which is much more efficient.

For example, to replace cells u1, u2, and u3 in TOP with the A design (in A.db), enter

```
pt_shell> read_db A.db
pt_shell> current_design TOP
pt_shell> swap_cell {u1 u2 u3} A.db:A
```

To replace cells u1, u2, and u3 in TOP with an LC3 lib_cell in the misc_cmos library, enter

```
pt_shell> swap_cell {u1 u2 u3} [get_lib_cell \
          misc_cmos/LC3]
```

Renaming Cells or Nets

The rename_cell and rename_net commands allow you to change the name of a cell or net. When renaming a cell or a net using hierarchical names, make certain that the old and new names are at the same level of hierarchy. You can refer to a cell or net by its full hierarchical name as long as it is below the current instance.

To successfully rename a cell or net, you must meet the following conditions:

- PrimeTime must be able to find the cell or net.
- No more than one cell or net matches the cell or net pattern you specify.
- No leaf cell or net is found to match the new name you specify.
- New name must be at the same hierarchical level as the cell or net that you are renaming.

The following example successfully renames a cell, cellA to cellB, in the current instance (middle).

```
pt_shell> rename_cell cellA cellB
Information: Renamed cell 'cellA' to 'cellB' in design
'middle'.
(NED-008)
1
```

The following example renames a net at a level below the current instance. In this example, u1 and u1/low are instances of designs that have multiple instances; therefore, they are uniquified as part of the editing process.

```
pt_shell> rename_net [get_net u1/low/w1] u1/low/netA
Uniquifying 'u1/low' (low) as 'low_0'.
Uniquifying 'u1' (inter) as 'inter_0'.
Information: Renamed net 'w1' to 'netA' in 'middle/u1/low'.
(NED-
009)
1
```

PrimeTime saves the name changes using `rename_cell` and `rename_net` for output with `write_changes`. For additional information, see the man page for `rename_cell` or `rename_net` for additional information.

User Function Class Support

For some complex library cells, the `user_function_class` attribute is required to describe the functional behavior of the cell instead of the `function_id` attribute. With support for the library cell attribute `user_function_class`, what-if analysis cell resizing is now possible for these complex cells.

Although PrimeTime scales cells with the same `function_id` attribute, it is possible for users to define their own function class. For example,

```
data buffers
clock buffers
delay buffers (used to fix hold times)
```

The foundry could provide these function classes to ensure that the proper cells are chosen for upsizing. You can set the `function_id` attribute in Design Compiler or in the source .lib file. For more information about setting this attribute, see the Design Compiler documentation.

Interdependent Setup and Hold Pessimism Reduction

Setup and hold pessimism reduction (SHPR) allows you to reduce pessimism in slack computation in path-based analysis. This is done by using the information in a library that supports interdependent setup and hold characterization data.

In a traditional library, the setup and hold constraint arcs for a sequential cell have a single fixed behavior. Setup might be characterized conservatively to allow hold constraint arcs to be characterized aggressively or vice versa. In some libraries, both the setup and hold constraints might be characterized conservatively (large setup/hold requirements) to avoid any possibility of failure during operation. In other libraries, both constraints might be characterized aggressively (small setup/hold requirements) to achieve maximum performance. If both setup and hold are characterized aggressively, failures can result unless care is taken to avoid data pulse widths that are too narrow to be reliably captured by the clock edge.

In reality, the magnitude of the setup and hold constraint requirements depend on each other. In a library with SHPR data, PrimeTime is able to understand this interdependence between setup and hold and trade off the setup and hold checks against each other to adapt to the needs of each sequential cell to the upstream logic during path-based analysis. By using the SHPR data, PrimeTime is able to use aggressive setup/hold constraints while still ensuring the minimum data pulse width requirements for reliable capture are met.

Use Model for SHPR

PrimeTime supports three modes for SHPR as explained in the sections below. Some user-controlled constraints of SHPR optimization are also available.

- Setup-Preferred Slack Improvement
- Hold-Preferred Slack Improvement
- Total Negative Slack Improvements

Setup-Preferred Slack Improvement

For some designs that prefer fast clock frequency (and assuming the hold time violation is easy to fix), the setup-preferred slack improvement mode can be desirable. This mode optimizes only total negative setup slacks of `max_rise` and `max_fall` paths ending at a specific flip-flop, with the trade off of corresponding hold slacks.

Hold-Preferred Slack Improvement

For some designs in which the setup time violation can be easily fixed by lowering clock frequency, hold time violation fixing becomes the primary consideration. The hold-preferred slack improvement mode is designed for such a case. This mode optimizes only total negative hold slacks of `min_rise` and `min_fall` paths ending at a specific flip-flop, with the trade off of corresponding setup slacks.

Total Negative Slack Improvements

The total negative slack mode minimizes the sum of negative slacks of all four types of timing paths ending at a particular flip-flop. By default, this mode allows trade off between the positive slack of one type of timing path and the negative slack of another type of timing path, without incurring new timing violations. This mode gives the same priorities on both negative setup slack and negative hold slack. This is the default mode of SHPR.

SHPR Optimization Constraints

For the three modes just described, user controls are provided to influence the slack trade-off. This is known as the SHPR optimization constraint, which defines the limit of trade off of the positive setup or hold slack. When one type of slack is sacrificed to improve the other type of slack, you can specify the slack limit for the sacrificial slack trade off.

SHPR Optimization Mechanism

In SHPR, a sequential capturing endpoint can be traded off in only two independent scenarios:

- `setup_rise` against `hold_fall`
- `setup_fall` against `hold_rise`

Figure 5-27 depicts these two scenarios.

Figure 5-27 Two Setup and Hold Trade off Scenarios

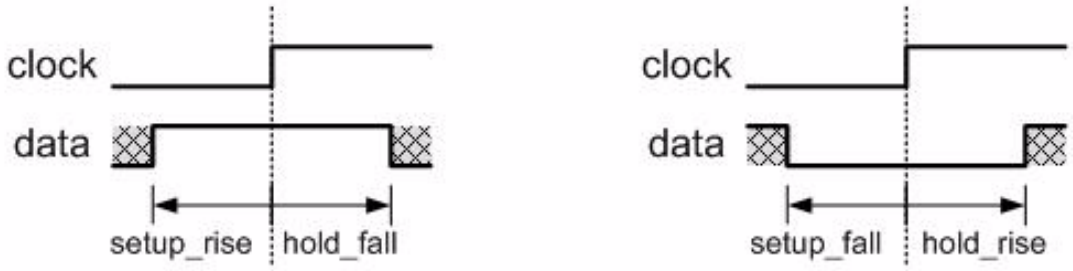
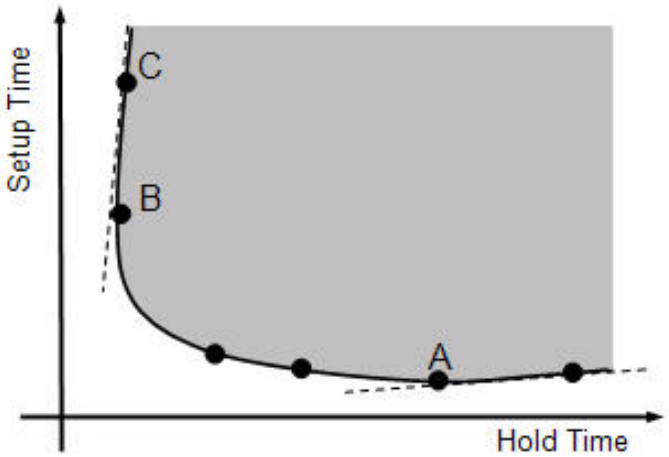


Figure 5-28 shows a typical interdependent setup and hold value curve. On the curve, any points beyond point A or B have a positive slope. Points A and B are called the turning points.

Figure 5-28 Interdependent Setup and Hold Value Curve



To be conservative, PrimeTime does not allow any extrapolation beyond the boundary points of the characterized SHPR curve.

SHPR User Interface

To set the optimization constraints for SHPR, use the `set_setup_hold_pessimism_reduction` command. The valid modes are total, setup, and hold. You can use the different modes to control how setup slack is traded off for hold slack or vice versa.

In setup mode (also known as setup-preferred mode), the goal is to prefer positive setup slack at the expense of hold slack. In hold mode (also known as hold-preferred mode), the goal is to prefer positive hold slack at the expense of setup slack. In total mode (the default), the goal is to trade off setup and hold slack in a way that minimizes the sum of the negative rise/fall and setup/hold slack. This mode results in the smallest overall set of setup and hold timing violations.

It might not be desirable to introduce a large violation in the sacrificial slack type to gain only a small improvement in the preferred slack type. You can use the `-setup_cutoff` and `-hold_cutoff` options of the `set_setup_hold_pessimism_reduction` command to keep the trade off reasonable. In setup-preferred mode, hold violations are made no worse than the `-hold_cutoff` value to improve setup. In hold-preferred mode, setup violations are made no worse than the `-setup_cutoff` value to improve hold. The default value for both cutoff options is negative infinity, which allows any amount of slack worsening in the sacrificial slack to improve the preferred slack.

In both setup and hold mode, the preferred slack type is improved only as far as it takes to reach zero slack and no further.

To disable SHPR, use the `remove_setup_hold_pessimism_reduction` command, which has the following options:

- `-setup_cutoff`: reset cutoff setup slack
- `-hold_cutoff`: reset cutoff hold slack

If you do not add an option, the SHPR feature is completely disabled.

SHPR Examples

The following sections outline some examples of SHPR.

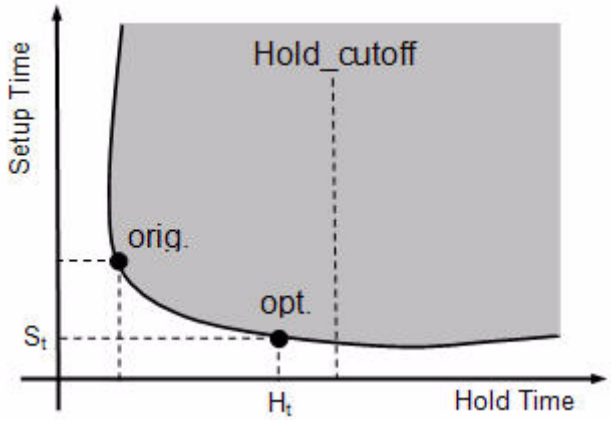
Setup Preferred Slack Improvement Example

In the following example, the mode is setup preferred slack improvement and the cutoff hold slack is -100 ps.

```
pt_shell>set_setup_hold_pessimism_reduction -mode setup \
         -hold_cutoff -100
```

The targeted setup time that results in zero slack in the max timing path is determined as S_t in Figure 5-29 below.

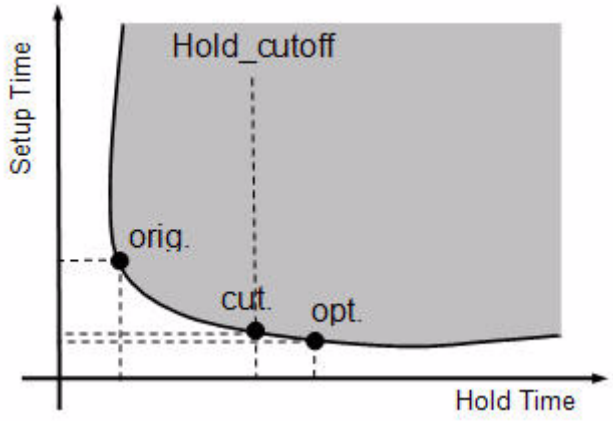
Figure 5-29 Successful Setup Slack Improvement



Using S_t as the input, you can get the corresponding hold time H_t , by using the interdependent setup and hold value curve. If H_t does not reach the `hold_cutoff` value, S_t and H_t are the optimized values of setup and hold time, as shown in Figure 5-29.

Instead, if H_t is beyond the `hold_cutoff` value, as shown below in Figure 5-30, the corresponding setup and hold times of the cutoff point are considered as the optimized setup and hold values.

Figure 5-30 Setup Slack Improvement Stops in Cutoff Hold Slack



Total Slack Improvement Example

The total mode is a special case. It does not find the minimal negative slack of min and max timing paths along the whole interdependent setup and hold curve. Instead, one of the following two cases with the minimal total negative slack of min and max timing paths is chosen:

- Case 1: [setup-preferred slack improvement mode, `hold_cutoff`]
- Case 2: [hold-preferred slack improvement mode, `setup_cutoff`]

Liberty Format Extension

The Liberty format and Library Compiler support interdependent setup and hold tables, starting in the Y-2006.06 release. The `interdependence_id` tag is used to identify the interdependent setup and hold tables. Figure 5-31 shows the library format needed to support SHPR.

Figure 5-31 Example of Library Format Needed to Support SHPR

```
timing() {
  related_pin : "CP";
  timing_type : setup_rising;
  rise_constraint(setup_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx");
  }
  fall_constraint(setup_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx");
  }
}
timing() {
  related_pin : "CP";
  timing_type : setup_rising;
  interdependence_id : 1;
  rise_constraint(setup_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx");
  }
  fall_constraint(setup_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx", \
       "xxxx, xxxx, xxxx");
  }
}
...
```

```
timing() {
  related_pin : "CP";
  timing_type : hold_rising;
  rise_constraint(hold_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy");
  }
  fall_constraint(hold_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy");
  }
}
timing() {
  related_pin : "CP";
  timing_type : hold_rising;
  interdependence_id : 1;
  rise_constraint(hold_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy");
  }
  fall_constraint(hold_template_3x3) {
    index_1 ("0.0264, 0.1728, 0.6736");
    index_2 ("0.0264, 0.1728, 1.3424");
values("yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy", \
       "yyyy, yyyy, yyyy");
  }
}
...
```

For more information, see the *Library Compiler Technology and Symbol Libraries Reference Manual*.

6

Advanced On-Chip Variation

Advanced on-chip variation (OCV) is a technology that recognizes the need for closer integration between design methodology and fabrication process variation. This technology reduces unnecessary pessimism by analyzing the physical location and logic depth of paths in a design. This chapter explains the graph-based and path-based advanced OCV solutions and provides you with flow and reporting information.

This chapter contains the following sections:

Introduction

Advanced OCV is an optional accuracy improvement that determines derating factors based on metrics of path logic depth and the physical distance traversed by a particular path. A longer path that has more gates tends to have less total variation because the random variations from gate-to-gate tend to cancel each other out. A path that spans a larger physical distance across the chip tends to have larger systematic variations. Advanced OCV is less pessimistic than a traditional OCV analysis, which relies on constant derate factors that do not take path-specific metrics into account.

The advanced OCV analysis determines path-depth and location-based bounding box metrics to calculate a context-specific advanced OCV derate factor to apply to a path, replacing the use of a constant derate factor.

The advanced OCV solution works with all other PrimeTime features and affects all reporting commands. This solution works in both the Standard Delay Format (SDF)-based and the delay calculation based flows. It is compatible with distributed multi-scenario analysis and multicore analysis.

Advanced OCV Flow

The advanced OCV solution in PrimeTime is integrated into the graph-based and path-based analysis capabilities, which enables a successive refinement strategy of analysis that is already used today with traditional static timing analysis in PrimeTime. In advanced OCV flows, sign-off is performed using graph-based advanced OCV, with path-based advanced OCV as an optional capability to refine the analysis of critical paths. Graph-based and path-based analysis with advanced OCV is analogous to graph-based and path-based analysis in the traditional static timing analysis context.

Graph-Based Advanced OCV Solution

Graph-based advanced OCV analysis is a fast, design-wide analysis performed during the `update_timing` command. It allows designers to exploit reduced derating pessimism across the entire design to reduce silicon area and improve design performance. You set the `timing_aocvm_enable_analysis` variable to `true` to enable the graph-based advanced OCV analysis.

By construction, graph-based analysis always provides a conservative analysis compared to path-based analysis to prevent an optimistic calculation. With graph-based advanced OCV analysis, PrimeTime computes conservative values of depth and distance metrics. To perform graph-based advanced OCV analysis, conservative values of path-depth and location-based bounding box are chosen to bound the worst-case path through a cell. For

example, in graph-based advanced OCV analysis, the depth count for a cell does not exceed that of the path of minimum depth that traverses that cell. Otherwise, the analysis might be optimistic for the minimum-depth path.

Graph-based advanced OCV is a prerequisite for further margin reduction using path-based advanced OCV analysis on selected critical paths. For most designs, graph-based advanced OCV is sufficient for sign-off.

Path-Based Advanced OCV Solution

The advanced OCV metrics computation is different for graph-based and path-based advanced OCV. In graph-based advanced OCV, the depth and distance metrics are calculated for all paths through each timing arc. In path-based mode, the advanced OCV metrics are calculated precisely for each timing path.

To further reduce pessimism and improve accuracy of advanced OCV results, use the `report_timing` command with the `-pba_mode` option to analyze paths in isolation from other paths. For example,

```
pt_shell> report_timing -pba_mode path
```

The PrimeTime path-based analysis engine performs both advanced OCV path-based analysis and regular path-based analysis by default. For the advanced OCV flow, it is strongly recommended that you set the `pba_aocvm_only_mode` variable to `true` so that only advanced OCV path-based analysis is applied. For more information about path-based analysis, see “Path-Based Timing Analysis” on page 5-14.

Although it is strongly advised that advanced OCV path-based analysis usage should always follow an advanced OCV graph-based analysis, advanced OCV path-based analysis within a traditional OCV flow is also permitted. When you have not enabled advanced OCV graph-based analysis, the derate factors computed by PrimeTime during advanced OCV path-based analysis are bounded by constant derates specified using the `set_timing_derate` command. The specification of these constant timing derate factors is required for this flow and it is your responsibility to ensure they are conservative.

Importing Advanced OCV Information

PrimeTime allows you to specify advanced OCV information using derate tables, which are required for an advanced OCV analysis. Guard-band timing derates allow you to optionally model nonprocess related effects in an advanced OCV flow. Importing advanced OCV information is described in more detail in the following sections:

- Specifying Derate Tables

- File Format for Advanced OCV
- Specifying Random Coefficients
- Guard-Banding in Advanced OCV

Specifying Derate Tables

You can use the `read_aocvm` command to read advanced OCV derate tables from a disk file. Derate tables are annotated onto one or more design objects and are applied directly to timing arc delays. The allowed design object classes are hierarchical cells, library cells, and designs. For more information about the `read_aocvm` command, see the man page.

You can use the `write_binary_aocvm` command to create binary encoded advanced OCV files from ASCII-advanced OCV files. This command is used to protect sensitive process-related information. The read_aocvm command can read binary and compressed binary advanced OCV files that were created using the `write_binary_aocvm` command. No additional arguments are required to read binary or compressed binary advanced OCV files.

You can use the `report_aocvm` command to report advanced OCV derate table data. An advanced OCV derate table imported from a binary or compressed binary file is not visible in the `report_aocvm` output.

File Format for Advanced OCV

The advanced OCV file format allows you to specify multiple advanced OCV derate tables. It supports the following table types:

- One-dimensional tables in either depth or distance
- Two-dimensional tables in both depth and distance

The file format is defined so that you can associate an advanced OCV table with a group of objects. The objects are selected internally within PrimeTime based using the following definitions:

- `OBJECT_TYPE  design | lib_cell | cell`
- `OBJECT_SPEC  [patterns]`

In the `OBJECT_SPEC` definition, the patterns field is optional. It specifies the object name and an expression that you want to evaluate based on the attributes of the object.

You can use regular expression matching for the patterns field of the OBJECT_SPEC definition. It is the same as the regexp Tcl command that you can use for design object gathering commands, such as the get_cells, get_lib_cells, and get_designs commands, in PrimeTime.

You can use any of the options of the related object-gathering command in the patterns field. For example, if the OBJECT_TYPE is lib_cell, you can use any of the arguments of the related get_lib_cells command in the patterns field. For more information about these commands, see the specific man pages.

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

Note:

All field descriptions are required, unless otherwise stated. To add a comment in any location within the file, use double forward slashes (//).

Table 6-1 shows the syntax definition for the advanced OCV file format.

Table 6-1 Advanced OCV File Format Syntax

Field specifier	Definition	Field description
version	VERSION	Advanced OCV version number.
object_type	OBJECT_TYPE	design | lib_cell | cell
rf_type	RF_TYPE	rise | fall | rise fall
delay_type	DELAY_TYPE	cell | net | cell net
derate_type	DERATE_TYPE	early | late
object_spec	[patterns]	string
depth	[DEPTH]	A set of M floats, where M can be zero.
distance	[DISTANCE]	A set of N floats, where N can be zero. Note: You should use nanometers (nm) for the distance field coordinates in the advanced OCV tables.

Table 6-1 Advanced OCV File Format Syntax (Continued)

Field specifier	Definition	Field description
`table`	`[TABLE]`	A set of N x M floats. There are also the following special cases: • If N==0, the table is of size M. • If M==0, the table is of size N. • If M==0 and N==0, the table is of size 1. Linear interpolation is used to determine points that have not been defined in the table. PrimeTime does not extrapolate beyond the lowest or highest values specified in the table.

The following example of an advanced OCV file sets an early advanced OCV table for the whole design, which applies to all cell and nets:

```
version             1.0
object_type:        design
rf_type:            rise fall
delay_type:         cell net
derate_type:        early
object_spec:        top
depth:              0 1 2 3
distance:           100 200
table:              0.87 0.93 0.95 0.96 \
                    0.83 0.85 0.87 0.90
```

The following example of an advanced OCV file sets an early advanced OCV table for the whole design, which applies to all nets:

```
version             1.0
object_type:        design
rf_type:            rise fall
delay_type:         net
derate_type:        early
object_spec:        top
depth:              0 1 2 3
distance:           100 200
table:              0.88 0.94 0.96 0.97 \
                    0.84 0.86 0.88 0.91
```

The following is an example of an advanced OCV file that sets a late advanced OCV table for all hierarchical cells matching "*" with an operating condition voltage of 1.2 V:

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

Specifying Random Coefficients

Random coefficients can be used to calculate cell path depths. A complex cell can consist internally of a larger number of transistors, so it might be necessary to associate that cell with a logic depth count that is larger than 1 (the default logic depth count for a cell arc). For instance, a buffer is often implemented as two inverters in series. In that case, the buffer cell could be assigned a random derate coefficient of 2.0 to set the logic depth count.

You can use the `set_aocvm_coefficient` command to set random advanced OCV coefficients on library cells. Random advanced OCV coefficients are not required for an advanced OCV analysis.

```
pt_shell> set_aocvm_coefficient 2.0 [get_lib_cells lib1/BUF2]
```

Guard-Banding in Advanced OCV

Guard-band timing derate allows you to model non-process related effects in an advanced OCV flow. The `-aocvm_guardband` option is available in the `set_timing_derate`, `report_timing_derate`, and `reset_timing_derate` commands.

You can use the `set_timing_derate` command to specify a guard-band derate factor. The `-aocvm_guardband` option is applicable only in an advanced OCV context. The derate factor that is applied to an arc is a product of the guard-band derate and the advanced OCV derate. The guard-band derate has no impact outside the context of advanced OCV.

To report only guard-band derate factors, specify the `-aocvm_guardband` option of the `report_timing_derate` command. If you do not specify either the `-variation` or `-aocvm_guardband` option, only the deterministic derate factors are reported. These two options are mutually exclusive.

To reset guard-band derate factors, you can use the `-aocvm_guardband` option of the `reset_timing_derate` command.

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

Advanced OCV Reporting

Internally calculated advanced OCV derates are shown in the derate column of the `report_timing` command if you have specified the `-derate` option.

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

In a graph-based advanced OCV analysis, you can specify a timing arc in the `object_list` of the `report_aocvm` command. The graph-based depth and distance metrics and advanced OCV derate factors on timing arc objects are displayed.

```
pt_shell> report_aocvm [get_timing_arcs -of_objects [get_cells U1]]
```

If you specify a timing path in the `object_list`, path metrics (distance, launch depth, and capture depth) for that path are displayed.

```
pt_shell> report_aocvm [get_timing_paths -path_type full_clock_expanded \
          pba_mode path]
```

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

Configuring an Advanced OCV Analysis

You can configure an advanced OCV analysis using the `timing_aocvm_analysis_mode` variable. You can include the following analysis modes in the variable:

`clock_network_only`

When you specify this option, the derivation and application of advanced OCV derates is restricted to arcs that are in the clock network. Clock network advanced OCV depth and distance metrics are calculated, considering only clock network topology. If constant derates have been annotated for data network objects, the data network receives constant on-chip variation derating. Otherwise, the data network is not derated.

`combined_launch_capture_depth`

Separate launch and capture depths are calculated in advanced OCV by default. By specifying this option, launch and capture paths are considered together and a combined depth is calculated for the entire path.

`single_path_metrics`

When you specify this option, single path metrics are calculated for all path objects. This behavior is backwardly compatible with the Tcl-based location aware on-chip variation (LOCV) solution.

For more information about the analysis modes, see the `timing_aocvm_analysis_mode` variable man page.

7

Ideal Network Support

PrimeTime allows for the creation of ideal networks, on which no design rule constraints (DRC) run. It is desirable that designers are able to ignore networks (large unoptimized networks with high fanout and capacitance) in pre-layout and focus instead on violations arising from other sources. Using ideal networks reduces runtime because PrimeTime uses “ideal timing” rather than internally calculated timing. In this way, ideal networks are similar to ideal clock networks, but they can also be applied to data networks.

The setting of an ideal network is described in the following sections:

Introduction to Ideal Networks

PrimeTime provides the ability to indicate ideal networks - sets of connected ports, pins, nets, and cells - and have them exempt from timing updates and DRC fixing. That is, ideal networks ignore max_capacitance, max_fanout, and max_transition design rule checks. When you specify the source of the ideal network, the pins, ports, nets, and cells contained therein are treated as ideal objects. You or ideal propagation must mark ideal objects.

Propagating Ideal Network Properties

When you specify the source objects (ports, leaf-level pins) of an ideal network, the nets, cells, and pins in the transitive fanout of the source objects can be treated as ideal.

Propagation of the ideal network property is governed by the following rules:

- Pin is marked as ideal if it is one of the following:
  - Pin is specified in the object list of the `set_ideal_network` command
  - Driver pin and its cell are ideal
  - Load pin attached to an ideal net
- Net is marked as ideal if all its driving pins are ideal.
- Combinational cell is marked as ideal if either all input pins are ideal or it is attached to a constant net and all other input pins are ideal.

  Note:
  Ideal network propagation can traverse combinational cells, but it stops at sequential cells.

PrimeTime propagates the ideal network during a timing update and propagates again from ideal source objects as necessary to account for changes in the design, for example, ECOs.

Using Ideal Networks

Using the `set_ideal_network` command, specify which networks you want set as ideal. Indicate the sources of the ideal network with the `object_list` argument. For example, the following command sets the ideal network property on the input port P1, which is propagated along the nets, cells, and pins in the transitive fanout of P1:

```
pt_shell> set_ideal_network P1
```

You can use the `-no_propagate` option to limit the propagation of the ideal network to only the nets and pins that are electrically connected to the ideal network source. If, for example, you enter

```
pt_shell> set_ideal_network -no propagate net1
```

only the net, net1, its driver pins, and its load pins are marked as ideal. No further propagation is performed.

If you do not use the `-no_propagate` option with this command, PrimeTime sets the ideal property on the specified list of pins, ports, or nets and propagates this property according to the propagation rules defined in the previous section.

You can remove the ideal network you set by using the `remove_ideal_network` command. To generate a report querying the port, pins, nets, or cells in the design to check whether or not they are ideal, use the `report_ideal_network` command. The following report shows the port, pin, nets, and cells identified as being part of an ideal network:

```
pt_shell> report_ideal_network -timing -load_pin -net -cell

****************************************
Report : ideal_network
        -timing
        -load_pin
        -net
        -cell
Design : middle
Version: Y-2006.06-DEV
Date   : Tue Feb 21 03:30:19 2006
****************************************

Source ports      Latency                  Transition
and pins        Rise       Fall            Rise       Fall
                min   max   min   max       min   max   min max
-------------------------------------------------------------
middle/p_in   --  --   --    --       --     --   --   --

Internal pins         Latency              Transition
with ideal timingRise      Fall            Rise       Fall
                 min max   min   max       min   max   min max
-------------------------------------------------------------
n3_i/B           5.62 5.62 --  --          --    --   --  --
n0_i/Z           1.34 --  1.34 --          --    --   --  --

Boundary pins         Latency              Transition
                Rise       Fall       Rise            Fall
             min   max  min   max  min   max      min    max
-------------------------------------------------------------
middle/p_out  --     --  --     --   --   --       --     --
Nets
```

```
------------------------------------------------------------
p_in
p_out
n2
n1
n0

Cells
------------------------------------------------------------
n3_i
n2_i
n1_i
n0_i

1
```

Using Ideal Latency

By default, the delay of an ideal network is zero. You can specify ideal latency on pins and ports in an ideal network by using the `set_ideal_latency` command. Ideal latency is accumulated along a path in the same way as a delay value.

Note that ideal latency takes effect only if the object is ideal. If it is not in an ideal network, PrimeTime issues a warning to that effect in the `report_ideal_network` command.

Remove latency from a pin or port by using the `remove_ideal_latency` command.

Using Ideal Transition

The transition time of an ideal network is zero by default. You can, however, specify an ideal transition time value with the `set_ideal_transition` command.

If you set an ideal transition value on an object, the value is propagated from that object along the ideal network either to the network boundary pins or until another ideal transition value is encountered.

Ideal transitions you have annotated on pins or ports take effect only if the object is ideal. If the object is not ideal, PrimeTime issues a warning in the `report_ideal_network` command informing you of this. Use the `remove_ideal_transition` command to remove the transition time you set on a port or pin.

8

SDF Back-Annotation

You can back-annotate a design with detailed delay information from a Standard Delay Format (SDF) file for more accurate timing analysis.

SDF back-annotation is described in the following sections:

Overview of SDF Back-Annotation

For initial static timing analysis, PrimeTime estimates net delays based on a wire load model. Actual delays depend on the physical placement and routing of the cells and nets in the design.

A floor planner or router can provide more detailed and accurate delay information, which you can provide to PrimeTime for a more accurate analysis. This process is called delay back-annotation. Back-annotated information often is provided in an SDF file.

You can read SDF back-annotated delay information in these ways:

- Read the delays and timing checks from an SDF file.
- Annotate delays, timing checks, and transition times from the command line without using the SDF format.

PrimeTime supports SDF v1.0 through 2.1 and a subset of v3.0 features. In general, it supports all SDF constructs except for the following:

- `PATHPULSE`, `GLOBALPATHPULSE`
- `NETDELAY`, `CORRELATION`
- `PATHCONSTRAINT`, `SUM`, `DIFF`, `SKEWCONSTRAINTS`

It also supports the following subset of SDF v3.0 constructs:

- `RETAIN`
- `RECREM`
- `REMOVAL`
- `CONDELSE`

If you do not have an SDF file, you can specify delays in an analyzer script file containing capacitance and resistance parasitics annotation commands. For more information, see Chapter 9, “Parasitic Back-Annotation.”

Reading SDF Files

The `read_sdf` command reads instance-specific pin-to-pin leaf cell and net timing information from an SDF version 1.0, 1.1, 2.0, 2.1, or 3.0 file, and uses the information to annotate the current design.

Instance names in the design must match instance names in the timing file. For example, if the timing file was created from a design using VHDL naming conventions, the design you specify must use VHDL naming conventions.

Annotating Timing From a Subdesign Timing File

When you specify the `-path` option, the `read_sdf` command annotates the current design with information from a timing file created from a subdesign of the current design. When you specify a subdesign, you cannot use the net delays to the ports of the subdesign to annotate the current design.

Annotating Load Delay

The load delay, also known as extra source gate delay, is the portion of the cell delay caused by the capacitive load of the driven net. Some delay calculators consider the load delay part of the net delay; other delay calculators consider the load delay part of the cell delay. By default, the `read_sdf` command assumes the load delay is included in the cell delay in the timing file being read. If your timing file includes the load delay in the net delay instead of in the cell delay, use the `-load_delay` option with the `read_sdf` command.

Annotating Timing Checks

When the timing file contains the following checks, they are used to annotate the current design.

For check	SDF construct is
setup and hold	`SETUP`, `HOLD`, and `SETUPHOLD`
recovery	`RECOVERY`
removal	`REMOVAL`
minimum pulse width	`WIDTH`
minimum period	`PERIOD`
maximum skew	`SKEW`
no change	`NOCHANGE`

Reading the File

The `read_sdf` command reads an SDF file. After reading an SDF file, PrimeTime reports the following:

- Number of errors found while reading the SDF file (for example, pins not found in the design)
- Number of annotated delays and timing checks
- Unsupported SDF constructs found in the SDF file, with the number of occurrences of each SDF construct
- Process, temperature, and voltage values found in the SDF file
- Annotated delays and timing checks of the design (with the `report_annotated_delay` and `report_annotated_check` commands)

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

Example 1

This command reads from disk the SDF format file adder.sdf, which contains load delays included in the cell delays and uses its information to annotate the timing on the current design.

```
pt_shell> read_sdf -load_delay cell adder.sdf
```

Example 2

These commands read the timing information of instance u1 of design MULT16 from the disk file mult16_u1.sdf, and annotates the timing on the design MY_DESIGN. The load delay is included in the net delays.

```
pt_shell> current_design MY_DESIGN
pt_shell> read_sdf -load_delay net -path u1 mult16_u1.sdf
```

Example 3

This command reads timing information and annotates the current design with the worst timing when the timing file has different timing conditions for the same pin pair. The load delay is assumed to be included in the cell delay.

```
pt_shell> read_sdf -cond_use_max boo.sdf
```

Example 4

This command reads minimum and maximum timing information and annotates the current design with delays corresponding to minimum and maximum operating conditions. When reporting minimum delay, PrimeTime uses delays annotated for the minimum condition. When reporting maximum delays, PrimeTime uses delays annotated for the maximum condition.

```
pt_shell> read_sdf -analysis_type bc_wc boo.sdf
```

Example 5

This command reads minimum and maximum timing information from two separate SDF files and annotates the current design with delays corresponding to minimum and maximum operating conditions. When reporting minimum delays, PrimeTime uses delays annotated for the minimum condition. When reporting maximum delays, PrimeTime uses delays annotated for the maximum condition.

```
pt_shell> read_sdf -analysis_type bc_wc \
            -min_file boo_bc.sdf -max_file boo_wc.sdf
```

Removing Annotated Timing Checks and Delays

To remove timing checks annotated with the `read_sdf` command, use the `reset_design` or `remove_annotated_check` command. To remove delays read and annotated with the `read_sdf` command, use the `reset_design` or `remove_annotated_delay` command.

Managing Large Files

You can compress large files such as SDC, SDF, and script files to a UNIX gzip file when you use the following `write` commands with the `-compress` option:

- `write_sdc`
- `write_sdf`
- `write_sdf_constraints`
- `write_script`
- `write_ilm_script`

Note:

If you intend to use the `write_sdf_constraints` command as part of your flow, it is recommended that you set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update. For more information, see “Generating Timing Constraints for Place and Route” on page 8-21 or the man page.

An example of writing an SDF file to a gzip file is as follows:

```
pt_shell> write_sdf -compress gzip 1.sdf.gz
```

The following commands read gzip compressed files:

- `read_verilog`
- `read_parasitics`
- `read_sdf`
- `read_sdc`

Each command recognizes gzip compressed files automatically. For example, to read a gzip file, enter

```
pt_shell> read_sdf 1.sdf.gz
```

Reporting Delay Back-Annotation Status

Because SDF files are usually large (about 100 MB or larger) and contain many delays, it is a good idea to verify that all nets are back-annotated with delays (and timing checks, where applicable).

Reporting Annotated or Nonannotated Delays

You can check and identify nets that have not been back-annotated with delays in SDF. PrimeTime checks and reports on cell delays and net delays independently. When reporting net delays, PrimeTime considers three types of nets:

- Nets connected to primary input ports
- Nets connected to primary output ports
- Internal nets that are not connected to primary ports

This distinction is important because according to the SDF standard, only internal nets are annotated in SDF.

The `report_annotated_delay` command reports the number of annotated and nonannotated delay arcs. For more information, see the `report_annotated_delay` man page.

To produce a report of annotated delays, enter

```
pt_shell> report_annotated_delay
```

PrimeTime displays a report similar to the following:

```
                              |           |           |    NOT    |
Delay type                    |    Total  | Annotated | Annotated |
------------------------------+-----------+-----------+-----------+--
cell arcs                     |       16  |       14  |        2  |
cell arcs (unconnected)       |        5  |        5  |        0  |
internal net arcs             |        3  |        3  |        0  |
net arcs from primary inputs  |       11  |       11  |        0  |
net arcs to primary outputs   |        4  |        4  |        0  |
------------------------------+-----------+-----------+-----------+----
                              |       39  |       37  |        2  |
```

Reporting Annotated or Nonannotated Timing Checks

You can create a report showing the number of annotated timing checks of all cell timing arcs within your design. For more information, see the `report_annotated_check` man page.

To produce an annotated timing check report, enter

```
pt_shell> report_annotated_check
```

PrimeTime displays a report similar to the following:

```
*****************************
report: annotated_check
Design: my_design
*****************************
                                          |NOT        |
                               |Total     |Annotated  |Annotated  |
-------------------------------+----------+-----------+-----------|
cell setup arcs                |   2368   |   2368    |   0       |
cell hold arcs                 |   2368   |   2368    |   0       |
cell recovery arcs             |    676   |    676    |   0       |
cell removal arcs              |      0   |      0    |   0       |
cell min pulse width arc       |    135   |    135    |   0       |
cell min period arcs           |    822   |    822    |   0       |
cell max skew arcs             |    716   |    716    |   0       |
cell nochange arcs             |      0   |      0    |   0       |
-------------------------------+----------+-----------+-----------|
                               |   7085   |   7085    |   0       |
```

Faster Timing Updates in SDF Flows

The following variable offers the ability to use zero slew for arcs annotated with SDF delays, thereby reducing runtime for analyzing designs with all (or almost all) arcs annotated with SDF:

`timing_use_zero_slew_for_annotated_arcs`

By default, this variable is set to false. For each point in a path, PrimeTime calculates slew from the input slew and capacitance, and propagates the calculated slew forward from that point in the path.

In a design that uses SDF-annotated delays on all arcs (or almost all arcs, such as 95% or more), you can forego the higher accuracy of slew calculation in favor of faster runtime. To make this choice, set the variable to true. In that case, for each arc that is fully annotated by either `read_sdf` or `set_annotated_delay`, PrimeTime skips the delay and slew calculation and sets a slew of zero on the load pin of the annotated arc. As a result, timing updates can be completed in significantly less time.

For any arcs that are not annotated, PrimeTime estimates the delay and output slew using the best available input slew. For a block of arcs that are not annotated, PrimeTime propagates the slew throughout the block using the `worst_slew` mode if the `timing_slew_propagation_mode` is set to `worst_slew`.

When you use this feature, it is recommended that you disable prelayout slew scaling by setting the `timing_prelayout_scaling variable` to `false`.

Annotating Conditional Delays From SDF

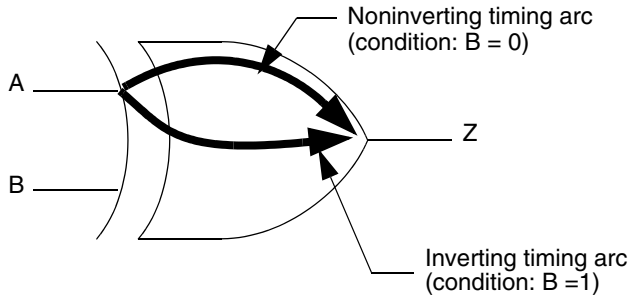
Delays and timing checks specified in an SDF file can be conditional. The SDF condition is usually an expression based on the value of some inputs of the annotated cell. The way PrimeTime annotates these conditional delays depends on whether the Synopsys library specifies conditional delays.

- If the Synopsys library contains conditional arcs, all conditional delays specified in SDF file are annotated. The condition string specified in the Synopsys library with the `sdf_cond` construct must exactly match the condition string in the SDF file.
- If the Synopsys library does not contain conditional arcs (there is no `sdf_cond` construct in the Synopsys library), the maximum or minimum delays of all conditional delays from SDF are annotated. To specify whether to annotate the minimum or maximum delays, use the `-cond_use_max` or `-cond_use_min` option of the `read_sdf` command.

If your library contains state-dependent delays, using a Synopsys library containing conditional arcs enables more accurate annotation from the SDF.

PrimeTime uses the condition specified in the SDF only to annotate the delays and timing checks to the appropriate timing arc specified in the Synopsys library. If you have conditional timing arcs in your SDF, and your library is defined correctly to support conditional arcs, you can use case analysis (or constant propagation) to enable the desired conditional arc values. Consider the example 2-input XOR gate in Figure 8-1.

Figure 8-1 Example of State-Dependent Timing Arcs

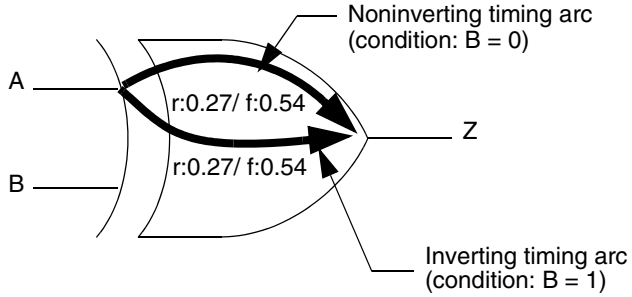


The SDF for the delay from A to Z looks like this:

```
(INSTANCE U1)
(DELAY
 (ABSOLUTE
      (COND B (IOPATH A Z (0.21) (0.54) ) )
      (COND ~B (IOPATH A Z (0.27) (0.34) ) )
 )
)
```

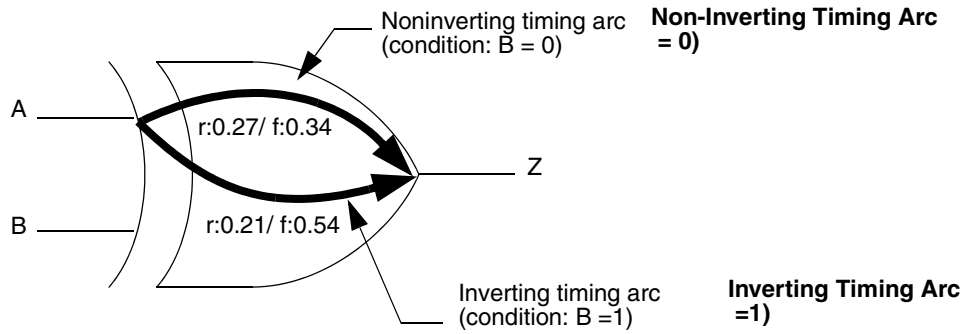
If the Synopsys library contains no conditions, the annotation from SDF uses the worst-case delay for all timing arcs from A to Z. Mapping from the library the timing arc that corresponds to a given condition is not possible. In this case, the annotated delays are as shown in Figure 8-2.

Figure 8-2 Annotated Delays When the Synopsys Library Contains No Conditions



If the Synopsys library contains conditions, the annotation can identify which timing arc corresponds to the SDF condition. In this case, the annotated delays are as shown in Figure 8-3.

Figure 8-3 Annotated Delays When the Synopsys Library Contains Conditions



Note:

An `IOPATH` statement in the SDF file annotates all arcs between two pins. However, even if an `IOPATH` statement follows a `COND IOPATH` statement, the `COND IOPATH` statement takes precendence over the `IOPATH` statement.

The `IOPATH` delay for the A to Z arc varies depending on whether the B input is a 1 or 0. To select the B = 0 delays, set a case analysis on the B input pin. For example, enter

```
pt_shell> set_case_analysis 0 [get_pins U1/B]
```

Note:

It is possible for a constant to propagate to a pin that selects a conditional arc. This can occur through normal logic constant propagation from a tie-high or tie-low condition, or through a `set_case_analysis` that was set on another pin that propagates a constant to the pin.

Writing an SDF File

You might want to write out the back-annotated delay information to use for gate-level simulation or another purpose. You can use the `write_sdf` command to write the delay information in SDF version 1.0, 2.1, or 3.0 format. The default output format is version 2.1. For example, to write an SDF file, enter

```
pt_shell> write_sdf -version 2.1 -input_port_nets mydesign.sdf
```

Note:

If you use a utility other than the `write_sdf` command to write out the SDF file, you should ensure that the annotations are explicitly specified where the SDF version permits.

For more information about the `write_sdf` command, see the man page. For information about specifying the format of the written SDF, see Appendix A, “Writing Mapped SDF Files.”

The SDF written by PrimeTime has the style described in the following sections:

SDF Constructs

The SDF written by PrimeTime uses the following SDF constructs:

- DELAYFILE, SDFVERSION, DESIGN, DATE, VENDOR, PROGRAM, VERSION, DIVIDER, VOLTAGE, PROCESS, TEMPERATURE, TIMESCALE
- CELL, CELLTYPE, INSTANCE
- ABSOLUTE, COND, CONDELSE, COSETUP, DELAY, HOLD, INTERCONNECT, IOPATH, NOCHANGE, PERIOD, RECOVERY, RECREM, RETAIN, SETUP, SETUPHOLD, SKEW, TIMINGCHECK, WIDTH

  Note:

  The following constructs are supported in SDF version 3.0 only: CONDELSE, RETAIN, RECREM, REMOVAL.

- Posedge and negedge identifiers

The SDF written by the `write_sdf` command does not use the following SDF constructs: INCREMENT, CORRELATION, PATHPULSE, GLOBALPATHPULSE, PORT, DEVICE, SUM, DIFF, SKEWCONSTRAINT, PATHCONSTRAINT. However, the `write_sdf_constraints` command supports the PATHCONSTRAINT construct.

SDF Delay Triplets

The SDF delay triplet values depend on whether your design uses a single operating condition or minimum and maximum operating conditions. For a single operating condition, the SDF delay triplet has three identical values, for example: (1.0:1.0:1.0).

For minimum and maximum operating conditions, the SDF triplet contains only two delays for the minimum operating condition and the maximum operating condition, respectively: (1.0::2.0). The typical delay of the SDF triplet is not used. The SDF delays written by PrimeTime specify the following transitions: 0 to 1, 1 to 0, 0 to Z, Z to 1, 1 to Z, and Z to 0.

SDF Conditions and Edge Identifiers

PrimeTime takes advantage of the edge identifiers as well as conditions if edge identifiers are specified in the library with `sdf_cond`. PrimeTime uses edge identifiers for timing checks and cell delays.

PrimeTime writes an edge identifier (`POSEDGE` or `NEGEDGE`) for a combinational cell delay arc when the positive edge delay differs from the negative edge delay, and the input net transition differs between rise and fall on the input pin of the delay arc. As a result, two timing arc `IOPATH` delays can be generated for a given timing arc. For example,

```
(CELL
  (CELLTYPE "XOR")
  (INSTANCE U1)
  (DELAY
    (ABSOLUTE
    (IOPATH (posedge A) Z (0.936:0.936:0.936)
(1.125:1.125:1.125))
    (IOPATH (negedge A) Z (1.936:1.936:1.936)
(2.125:2.125:2.125))
    (IOPATH (posedge B) Z (0.936:0.936:0.936)
(1.125:1.125:1.125))
    (IOPATH (negedge B) Z (1.936:1.936:1.936)
(2.125:2.125:2.125))
    )
  )
)
```

This is common for library cells such as multiplexers and exclusive-OR cells. Other SDF writers might only generate one set of delay triplets for the positive and negative edges. PrimeTime writes both for the highest accuracy. However, some logic simulators do not support edge identifiers on combinational and sequential timing arcs and expect to see only one timing arc. To write an SDF that is compatible with these simulators, use the `-no_edge` option with the `write_sdf` command. For example,

```
pt_shell> write_sdf -no_edge mydesign.sdf
```

With the `-no_edge` option, PrimeTime generates only one timing arc, with the worst-case delay triplets for the positive and negative transitions. Note that the simulation timing delays might be pessimistic as a result of using the `-no_edge` option.

Reducing SDF for Clock Mesh/Spine Networks

A design with a large clock mesh or spine network can produce an unreasonably large SDF file because of the extremely large number of nets in the clock network. In these cases, you can choose to have PrimeTime reduce the SDF file by combining SDF values that differ by a negligible amount, and using the SDF 3.0 PORT construct to represent the combined nets.

Note:
For related information, see “Parallel Driver Reduction” on page 5-39 and “Fast Multidrive Delay Analysis” on page 5-38.

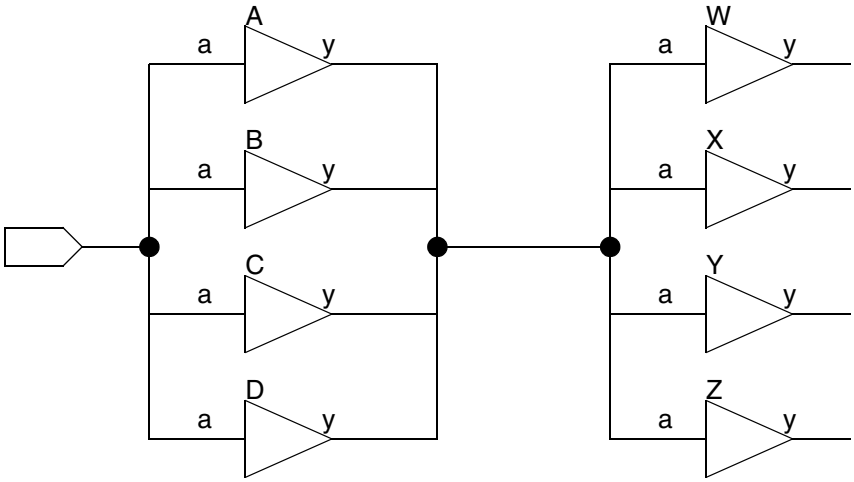
The SDF reduction features operate under the control of four variables:

- `sdf_enable_port_construct` – Boolean variable, when set to true, enables the PORT construct to be used for writing SDF. It is set to false by default.
- `sdf_enable_port_construct_threshold` – Floating-point value that specifies the absolute delay difference, in picoseconds, below which the PORT construct is used. The default setting is 1 ps.
- `sdf_align_multi_drive_cell_arcs` – Boolean variable, when set to true, causes PrimeTime to unify the small differences in driver cell outputs to networks that constitute a mesh or spine, which can cause simulation failure. It is set to false by default.
- `sdf_align_multi_drive_cell_arcs_threshold` – Floating-point value that specifies the absolute delay difference, in picoseconds, below which multi-drive arcs are aligned. The default setting is 1 ps.

PORT Construct

The `sdf_enable_port_construct` variable determines whether the PORT statement is used to replace multiple INTERCONNECT statements. The PORT statement will be used in all parallel nets (within specified threshold) in the design except those parallel nets which are driven by tristate buffers. If the PORT statement is used, the `sdf_enable_port_construct_threshold` variable determines the maximum allowable absolute difference in delay arc values for interconnections to be combined. For example, consider the clock network shown in Figure 8-4.

Figure 8-4 Parallel Buffers Driving Parallel Buffers



If using the PORT construct is disabled (the default), the `write_sdf` command writes out this clock network using SDF syntax similar to the following:

```
(INSTANCE top)
(DELAY
  (ABSOLUTE
    (INTERCONNECT A/y W/a (0.01 ::0.03))
    (INTERCONNECT A/y X/a (0.02 ::0.04))
    (INTERCONNECT A/y Y/a (0.01 ::0.04))
    (INTERCONNECT A/y Z/a (0.02 ::0.03))
    (INTERCONNECT B/y W/a (0.01 ::0.03))
    (INTERCONNECT B/y X/a (0.03 ::0.05))
    ...
    (INTERCONNECT D/y Y/a (0.02 ::0.05))
    (INTERCONNECT D/y Z/a (0.01 ::0.03))
  )
)
```

There are 16 interconnection combinations listed.

If the PORT construct is enabled and if the variation in delay values is within the specified threshold, the `write_sdf` command reduces the SDF as follows:

```
(INSTANCE top)
(DELAY
  (ABSOLUTE
    (PORT W/a (0.02 ::0.04))
    (PORT X/a (0.03 ::0.05))
    (PORT Y/a (0.02 ::0.06))
    (PORT Z/a (0.03 ::0.07))
  )
)
```

Four PORT statements replace 16 INTERCONNECT statements.

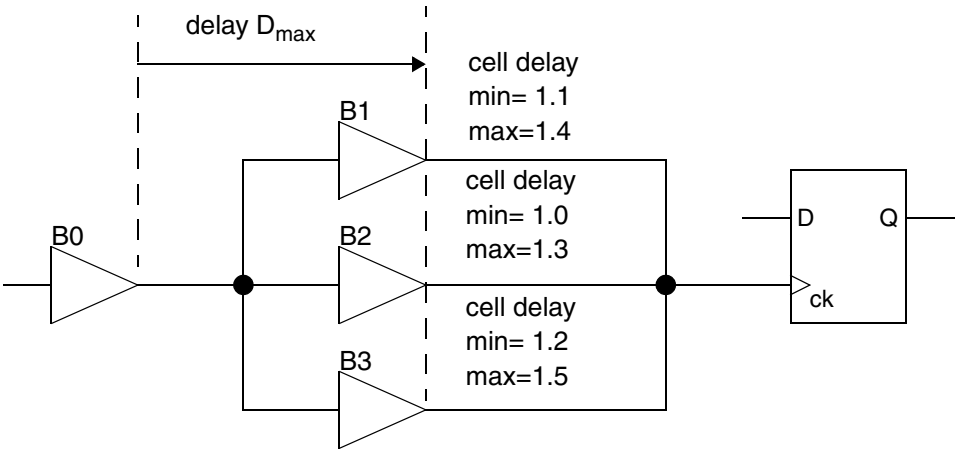
Normalizing Multi-Driven Arcs for Simulation

When the sdf_align_multi_drive_cell_arcs and sdf_enable_port_construct variables are enabled, PrimeTime aligns similar delay values of cell timing arcs where multiple drivers drive a common net.

Note:
If the common net is driven by tristate buffers, PrimeTime will not align cell delays.

The sdf_align_multi_drive_cell_arcs_threshold variable then determines the maximum allowable absolute difference in delay arc values for normalizing to occur. Normalizing means using a single worst-case delay arc value to represent multiple drivers. This can prevent errors from occurring when the SDF file is used for circuit simulation. For example, consider the cell delays in the parallel driver network shown in Figure 8-5.

Figure 8-5 Cell Delays in a Parallel Driver Network



PrimeTime aligns the cell arcs if the sdf_align_multi_drive_cell_arcs variable is set to true, the delay are values are within the sdf_align_multi_drive_cell_arcs_threshold variable setting, the cells in the clock network are combinational (not sequential), and each cell has one input and one output.

The normalizing process adjusts both the net delays and cell delays in the clock network as needed to ensure that the complete path delays are the same. Net delays are adjusted only if the sdf_enable_port_construct variable is set to true and the delays are within the threshold variable setting, in which case the PORT statement is used to represent the net arcs connected to the load pin “ck” in the example. In the case of the nets connecting B0 to cells B1 through B3, no changes are made to the delay values because each of the three cells has only one fanin net.

In the case of the cell delays, the worst delay from B0 to the output pin of cells B1 through B3 is calculated to be Dmax. The cell arcs of B1 through B3 are adjusted to make the delay of each path from B0 to the output pins of the cells to be equal to Dmax. In this example, the cells are given the same delays because the net arcs from B0 to the cells B1 through B3 all have the same delays. Taking the worst delay means using the minimum of min delays and maximum of max delays, the most pessimistic values. In the case of cells with multiple arcs, the worst cell arc will be used to represent the cell delay.

The `sdf_align_multi_drive_cell_arcs` variable should be set to true only to generate SDF for simulation. Avoid reading the generated SDF back into PrimeTime for timing analysis, as the data is pessimistic.

If multi-drive normalizing is disabled, the `write_sdf` command writes the following description of the example network.

Net delays:

```
(INTERCONNECT B1/z FF1/ck (0.01 ::0.04))
(INTERCONNECT B2/z FF1/ck (0.03 ::0.05))
(INTERCONNECT B3/z FF1/ck (0.02 ::0.04))
```

Cell delays:

```
(CELLTYPE "buf")
(INSTANCE B1)
(DELAY
  (ABSOLUTE
    (DEVICE (1::8) (4::7))
  )
)
(CELLTYPE "buf")
(INSTANCE B2)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (3::11))
  )
)
(CELLTYPE "buf")
(INSTANCE B3)
(DELAY
  (ABSOLUTE
    (DEVICE (1::5) (5::8))
  )
)
```

If multi-drive normalizing is enabled, and if the delay differences are under the specified threshold, the `write_sdf` command writes the following description of the example network.

Net delays:

```
(PORT FF1/ck (0.03 ::0.05))
```

Cell delays:

```
(CELLTYPE "buf" )
(INSTANCE B1)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (5::11))
  )
)
(CELLTYPE "buf" )
(INSTANCE B2)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (5::11))
  )
)
(CELLTYPE "buf" )
(INSTANCE B3)
(DELAY
  (ABSOLUTE
    (DEVICE (2::9) (5::11))
  )
```

Writing VITAL Compliant SDF Files

If you want to write out SDF that is VITAL compliant, use the following command:

```
pt_shell> write_sdf -no_edge_merging -exclude {"no_condelse"} \
          file.sdf
```

If the simulator you plan to use cannot handle negative delays, you can use the `-no_negative_values` option with the `timing_checks`, `cell_delays`, or `net_delays` values. You should use the `-no_negative_values timing_checks` option when you want to zero out all negative timing check values, such as setup, hold, recovery or removal SDF statements. Use the `-no_negative_values cell_delays` option when you want to zero out all negative cell delay values, such as IOPATH and port SDF statements. Use the `-no_negative_values net_delays` option when you want to zero out all negative net delays, such as interconnect SDF statements. For example, if you want to zero out timing checks and net delays, use the following syntax:

```
pt_shell> write_sdf -no_edge_merging -no_negative_values \
     {timing_checks net_delays} -exclude {"no_condelse"} file.sdf
```

Removing Annotated Delays and Checks

You can remove annotated delays and timing checks using these commands:

- remove_annotated_delay
- remove_annotated_check

Removing Annotated Delays

The remove_annotated_delay command removes annotated cell and net delays from the current design. Delays are annotated either from an SDF file by using the read_sdf or set_annotated_delay commands.

Examples

To remove all annotated net and cell delays from the current design, enter

```
pt_shell> remove_annotated_delay -all
1
```

To remove annotated delays from some cells, enter

```
pt_shell> remove_annotated_delay [get_cells u1*]
1
```

The following command removes annotated delays between pins. When there are no annotated delays between the specified pins, a warning message appears.

```
pt_shell> remove_annotated_delay -from ffb/Q
1
pt_shell> remove_annotated_delay -from ffb/Q -to u1/A
Warning: No annotated delays from 'ffb/Q' to 'u1/A'
0
```

For more information, see the remove_annotated_delay man page.

Removing Annotated Checks

The remove_annotated_check command removes annotated timing checks from cells in the current design. Timing checks are annotated either from an SDF file by using the read_sdf command, or the set_annotated_check command. Timing check types include setup, hold, recovery, removal, and no-change.

Examples

To remove all annotated cell timing checks from the current design, enter

```
pt_shell> remove_annotated_check -all
```

To remove only setup checks from all cells in the current design, enter

```
pt_shell> remove_annotated_check -setup [get_cells *]
1
```

The following command removes checks between pins. The error condition is shown when the pins are not on the same cell, and when there are no annotated checks, a warning message appears.

```
pt_shell> remove_annotated_check -from ffb/CP -to ffa/D
Error: Cannot remove annotated check from 'ffb/CP' to 'ffa/
D':
pins are on different cells (PTE-032)
0
pt_shell> remove_annotated_check -from ffa/CP -to ffa/D \
            -setup -hold
1
pt_shell> remove_annotated_check -from ffa/CP -to ffa/D
Warning: No annotated timing checks were removed. (PTE-031)
0
```

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

Setting Annotations From the Command Line

You can annotate delays, timing checks, and transition times from the command line to make a limited number of changes for debugging. To manually set annotations, use these commands:

- `set_annotated_delay`
- `set_annotated_check`
- `set_annotated_transition`

Annotating Delays

The `set_annotated_delay` command sets cell or net delays from the command line. To list annotated delay values, use the `report_annotated_delay` command. To set a cell delay, you specify the delay from a cell input to an output of the same cell. To set a net delay, you specify the delay from a cell output to a cell input.

To remove the annotated cell or net delay values from a design, use the `remove_annotated_delay` or `reset_design` command.

Example 1

This example annotates a cell delay of 20 units between input pin A of cell instance U1/U2/U3 and output pin Z of the same cell instance. The delay value of 20 includes the load delay.

```
pt_shell> set_annotated_delay -cell -load_delay cell 20 \
            -from U1/U2/U3/A -to U1/U2/U3/Z
```

Example 2

This example annotates a rise net delay of 1.4 units between output pin U1/Z and input pin U2/A. The delay value for this net does not include load delay.

```
pt_shell> set_annotated_delay -net -rise 1.4 -load_delay \
            cell -from U1/Z -to U2/A
```

Example 3

This example annotates a rise net delay of 12.3 units between the same output pins. In this case the net delay value does include load delay.

```
pt_shell> set_annotated_delay -net -rise 12.3 -load_delay \
            net -from U1/Z -to U2/A
```

Example 4

This example annotates a fall cell delay of 21.2 units on the enable arc of the tristate cell instance U8.

```
pt_shell> set_annotated_delay -cell -fall -of_objects \
          [get_timing_arcs -from U8/EN -to U8/Z -filter

sense==enable_low] 21.2
```

Annotating Timing Checks

The `set_annotated_check` command sets the setup, hold, recovery, removal, or no-change timing check value between two pins.

To list annotated timing check values, use the `report_annotated_check` command. To remove annotated timing check values from a design, use the `remove_annotated_check` or `reset_design` command. To see the effect of `set_annotated_check` for a specific instance, use the `report_timing` command.

Example

This example annotates a setup time of 2.1 units between clock pin CP of cell instance u1/ff12 and data pin D of the same cell instance.

```
pt_shell> set_annotated_check -setup 2.1 -from u1/ff12/CP \
```

```
        -to u1/ff12/D
```

Annotating Transition Times

The `set_annotated_transition` command sets the transition time on any pin of a design. Transition time (also known as slew) is the amount of time it takes for a signal to change from low to high or from high to low.

Example

This example annotates a rising transition time of 0.5 units and a falling transition time of 0.7 units on input pin A of cell instance U1/U2/U3.

```
pt_shell> set_annotated_transition -rise 0.5 [get_pins U1/U2/U3/A
pt_shell> set_annotated_transition -fall 0.7 [get_pins U1/U2/U3/A
```

Generating Timing Constraints for Place and Route

You can use the `write_sdf_constraints` command to write path timing constraints in SDF versions 1.0 and 2.1 formats to a constraints file. Use the constraint file to constrain layout tools to meet critical timing goals.

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

1. Set the `timing_save_pin_arrival_and_slack` variable to `true` to store slacks at all pins in the design.
2. Update timing by using the `update_timing` command.

   Note:

   If you intend to use `write_sdf_constraints` command as part of your flow, it is recommended that you set the `timing_save_pin_arrival_and_slack` variable to `true` before your first timing update.
3. Use the `write_sdf_constraints` to write constraints for the current design.

When you use the `write_sdf_constraints` command, keep the following limitations in mind:

- PrimeTime does not generate capacitance constraint files.
- PrimeTime does not support some Design Compiler `write_constraints` command options, including `-cover_nets` and `-load_delay net | cell`.

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

The following example shows a timing constraint file in SDF v2.1 format.

```
(DELAYFILE
(SDFVERSION "OVI 2.1")
(DESIGN "counter")
(DATE "Thur Nov 30 16:09:53 2000")
(VENDOR "lsi_10k")
(PROGRAM "Synopsys PrimeTime")
(VERSION "2000.11")
(DIVIDER /)
(VOLTAGE 5.00:5.00:5.00)
(PROCESS "NOMINAL")
(TEMPERATURE 25.00:25.00:25.00)
(TIMESCALE 1ns)
(CELL
  (CELLTYPE "counter")
  (INSTANCE)
  (TIMINGCHECK
    (PATHCONSTRAINT ffb/CP ffb/QN w/B w/Z q/A q/Z j/C j/Z
ffd/D
(9.100:9.100:9.100) (9.100:9.100:9.100) )
    (PATHCONSTRAINT ffb/CP ffb/QN v/A v/Z r/A r/Z g/C g/Z
ffc/D
(9.100:9.100:9.100) (9.100:9.100:9.100) )
    (PATHCONSTRAINT ffa/CP ffa/QN u/A u/Z s/A s/Z d/C d/Z
ffb/D
(9.100:9.100:9.100) (9.100:9.100:9.100) )
    (PATHCONSTRAINT ffa/CP ffa/QN t/B t/Z a/C a/Z ffa/D
(9.150:9.150:9.150) (9.150:9.150:9.150) )
     )
  )
)
```

The following example shows the syntax to write a timing constraint file in SDF v2.1 format for the most critical path in each path group in the current design:

```
pt_shell> write_sdf_constraints cstr.sdf
```

The following example shows the syntax to write a timing constraint file in SDF v1.0 format for the 10 most critical paths in each path group in the current design (named counter):

```
pt_shell> write_sdf_constraints -max_paths 10 \
             -version 1.0 counter_cstr.sdf
```

Providing Constraint Coverage for the Entire Design

To constrain a design fully for placement, every net or every net arc (connection between two cells) generally requires a constraint. You can ensure that a design is fully constrained by forcing the `write_sdf_constraints` command to generate constraints for all paths in the design. Generation of the resulting constraint file requires a large amount of CPU time and disk space.

Some place and route tools require only the worst path through any point (other data is not used). In these cases, generating the worst path through every driver-load pin pair in the design is sufficient to provide full constraint coverage. In that case, the total number of paths for the design is less than or equal to the total number of leaf cell input pins plus the number of primary outputs (often, points are covered by multiple paths).

The `write_sdf_constraints` command's `-cover_design` option generates just enough unique paths to provide constraint coverage for the entire design. The overall runtime with the `-cover_design` option is greater, but memory and disk space requirements are significantly less.

Note:

The `-cover_design` option ensures that a constraint is placed on every driver-load pin pair in the design, but does not fully constrain the design. The coverage is minimal; it contains less information than a constraint file with all paths enumerated. Theoretically, a place and route tool can meet all the given constraints and still have timing violations.

The `-cover_design` option is most useful for large designs in which generating data for all paths is not reasonable, or when the targeted place and route tool uses only the worst path through any point. A point can be a net or an edge, where an edge is any connection between two leaf-level cells.

For large designs in which generating data for all paths is not reasonable, using the `-cover_design` option might be the only way to ensure that every net is constrained at least once. When the targeted place and route tool uses only the worst path through any point, constraints other than the worst are ignored by the place and route tool.

To write a timing constraint file in SDF v2.1 format with just enough paths to cover the worst path through every driver-load pin pair, enter

```
pt_shell> write_sdf_constraints -cover_design \
            counter_cstr.sdf
```

9

Parasitic Back-Annotation

You can back-annotate a design with detailed parasitic information (resistance and capacitance) for more accurate timing analysis.

Parasitic back-annotation is described in the following sections:

Parasitic Data

PrimeTime allows parasitic back-annotation of detailed circuit delay information in the form of lumped capacitance, lumped resistance, reduced pi model, or detailed RC network. The reduced pi model and detailed RC network are more accurate than lumped capacitance and lumped resistance, but using them requires setting environment variables and uses more CPU time and memory.

Note:

To reduce CPU time usage to a minimum, use an SDF file rather than parasitic data to back-annotate the design. In that case, PrimeTime does not have to calculate the delays. For more information, see Chapter 8, “SDF Back-Annotation”.

In case of conflict between different types of parasitic data, PrimeTime uses the data on each net in the following order:

1. Lumped resistance and capacitance values annotated with the `set_resistance` and `set_load` commands, if any.
2. Detailed parasitics annotated with the `read_parasitics` command, if any.
3. Wire load models obtained from the technology library or specified with the `set_wire_load_model` command.

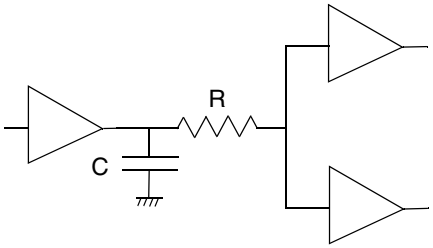
Setting lumped resistance or capacitance with the `set_resistance` or `set_load` command temporarily overrides the wire load model or detailed parasitics for a net. Removing lumped resistance or capacitance from a net with the `remove_resistance` or `remove_capacitance` command causes the net to revert back to its previous form of parasitic data.

You can set lumped resistance and capacitance separately. For example, you can read in detailed parasitics for a net, and then override just the capacitance for that net with the `set_load` command. In that case, PrimeTime still uses the detailed parasitic information to calculate the net resistance.

Lumped Parasitics

You can annotate resistance and capacitance (RC) on nets, as shown in Figure 9-1. Even if you annotated all the delays of the design with SDF, you might want to annotate parasitics to perform certain design rule checks, such as maximum transition or maximum capacitance.

Figure 9-1 Lumped RC



Setting Net Capacitance

The `set_load` command sets the net capacitance values on ports and nets within the design. If the current design is hierarchical, you must link it with the `link` command.

By default, the total capacitance on a net is the sum of all of pin, port, and wire capacitance values associated with the net. A capacitance value you specify overrides the internally estimated net capacitance.

You can use the `set_load` command for nets at lower levels of the design hierarchy. Specify these nets as BLOCK1/BLOCK2/NET_NAME.

If you use the `-wire_load` option, the capacitance value is set as a wire capacitance on the specified port and the value is counted as part of the total wire capacitance (not as part of the pin or port capacitance).

To view capacitance values on ports, use `report_port`. To view capacitance values on nets, use the `report_net` command. For more information, see the `set_load` man page.

Setting Net Resistance

The `set_resistance` command sets net resistance values in the design. The specified resistance value overrides the internally estimated net resistance value.

You can also use the `set_resistance` command for nets at lower levels of the design hierarchy. You can specify these nets as BLOCK1/BLOCK2/NET_NAME.

To view resistance values, use the `report_net` command. To remove resistance values annotated on specified nets, use the `remove_resistance` command. To remove resistance annotated on the entire design, use the `reset_design` command.

Reduced and Detailed Parasitics

PrimeTime provides features that enable you to annotate reduced parasitics and detailed parasitics.

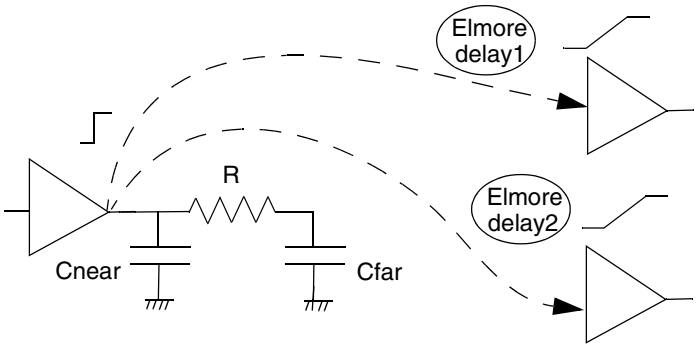
Annotating Reduced Parasitics

Reduced resistance and capacitance represents the RC within a design from a driver standpoint: two capacitors and one resistor, as shown in Figure 9-2. The net delays are typically annotated with an Elmore delay, which is the most commonly used representation. For this model the following rules apply:

- RC is represented in terms of a pi model (two C and one R).
- Net delay from the driver to the net fanout provided is an Elmore delay.
- The RC pi model is used to compute the slew at the driver pin.
- Slew degradation (transition time) from the driver pin to each net load pin is computed if the Elmore delay is significant (more than 20 percent of driver transition time). The slew degradation is based on a published paper by E. G. Friedman and J. H. Mulligan, Jr., titled “Ramp Input Response of RC Tree Network,” in Analog Integrated Circuits and Signal Processing, Volume 14, No. 1/2, pp. 53-58, September 1997.

The model in Figure 9-2 calculates the cell delay and the driver output slew more accurately than the lumped RC model. Reduced RC is annotated with RSPF or SPEF.

Figure 9-2 Reduced RC

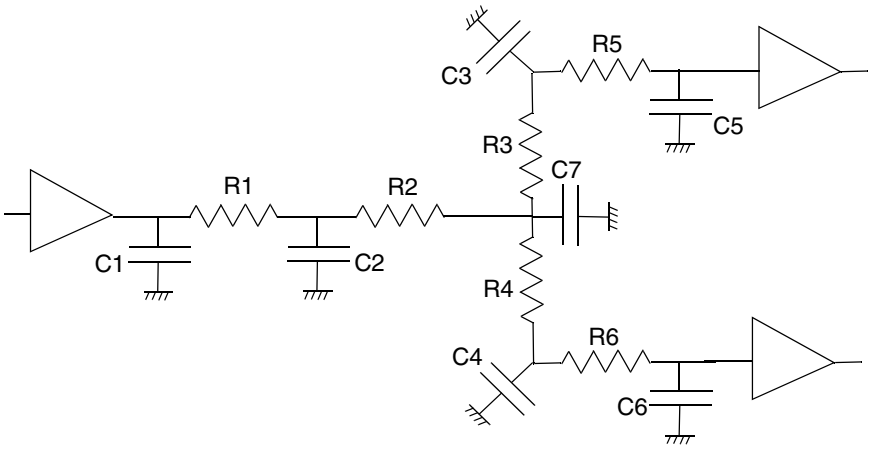


Annotating Detailed Parasitics

You can annotate detailed parasitics into PrimeTime and annotate each physical segment of the routed netlist in the form of resistance and capacitance (see Figure 9-3). Annotating detailed parasitics is very accurate but more time-consuming than annotating lumped parasitics. Because of the potential complexity of the RC network, PrimeTime takes longer to calculate the pin-to-pin delays in the netlist.

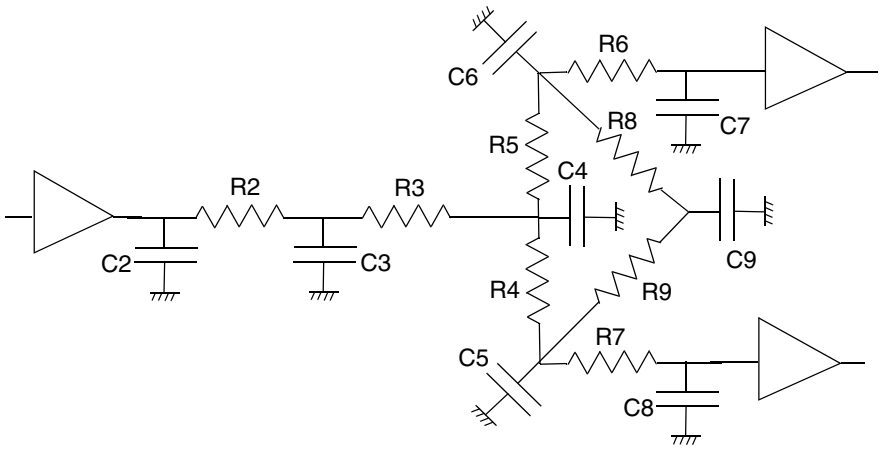
This RC network is used to compute effective capacitance (Ceffective), slew, and delays at each subnode of the net. PrimeTime can read detailed RC in DSPF and SPEF formats.

Figure 9-3 Detailed RC



You can use this model for netlists that have critical timing delays, such as clock trees. This model can produce more accurate results, especially in deep submicron designs where net delays are more significant compared to cell delays. The detailed RC network supports meshes, as shown in Figure 9-4.

Figure 9-4 Meshed RC



Supported File Formats for Parasitic Annotation

PrimeTime supports these formats for parasitic annotation:

- Cadence Design Systems Reduced Standard Parasitic Format (RSPF) and Detailed Standard Parasitic Format (DSPF)
- Open Verilog International (OVI) SPEF

Limitations

The following limitations apply to RSPF and DSPF constructs:

- SPICE inductors (Lxxx) and lines (Txxx) are allowed in the DSPF file, but they are ignored.
- Physical coordinates of pins and instances are ignored.
- The `BUSBIT` construct is not supported.
- Instances listed in the RSPF and DSPF files in the SPICE section are ignored. They are not checked to determine whether they match the current design loaded in PrimeTime.
- Resistors cannot be connected to ground. PrimeTime ignores such resistors and displays a warning about them.
- Capacitors must be connected to ground. PrimeTime ignores node-to-node coupling capacitors and displays a warning about them.

The following limitations apply to SPEF constructs:

- Inductors are ignored.
- Poles and residue descriptions on reduced nets are not supported.
- Resistors cannot be connected to ground. PrimeTime ignores resistors connected to ground and displays a warning.
- Cross-coupling capacitors are split to ground unless you are using PrimeTime SI and crosstalk analysis is enabled.

Characterization Trip Points

The characterization trip points of a design (waveform measurement thresholds) affect the calculation of delays and transition times by PrimeTime. PrimeTime establishes the trip points as follows:

1. If pin-level thresholds are defined, the application tools use those. Pin-level thresholds override library-level thresholds with the same name.

2. If pin-level thresholds are not defined, but library-level thresholds are, the application tools use the library-level thresholds.
3. If neither pin-level nor library-level thresholds are defined, and the PrimeTime environment variables are not set, the application tools use the thresholds defined in the main library (the first library in the link path).
4. If neither pin-level nor library-level thresholds are defined, the thresholds are determined by PrimeTime environment variables.
5. If none of the above are defined, the application tools use default trip points at 20 percent and 80 percent of the rail voltage for transition time calculations and 50 percent of the rail voltage for delay calculations.
6. If the trip points defined by any of these methods are not valid (for instance, 0 percent and 100 percent), the trip points will be set to 5 percent and 95 percent of the rail voltage for transition time calculations and 50 percent of the rail voltage for delay calculations.

Note:

In earlier releases of PrimeTime, the trip points were obtained in a different manner. To revert to this earlier behavior, set the `lib_thresholds_per_lib` variable to `false`. For more information, see the man page for the variable.

Figure 9-5 and Figure 9-6 show some signal waveforms, the trip points used to calculate transition times (slews) and delays, and the variables that can be used to define the trip points. The default trip point values define slew as the time that a signal takes to change from 20 to 80 percent or from 80 to 20 percent of the rail voltage, and they define cell delay as the amount of time from the input signal reaching 50 percent of the rail voltage to the output signal reaching 50 percent of the rail voltage.

Figure 9-5 Slew Transition Points

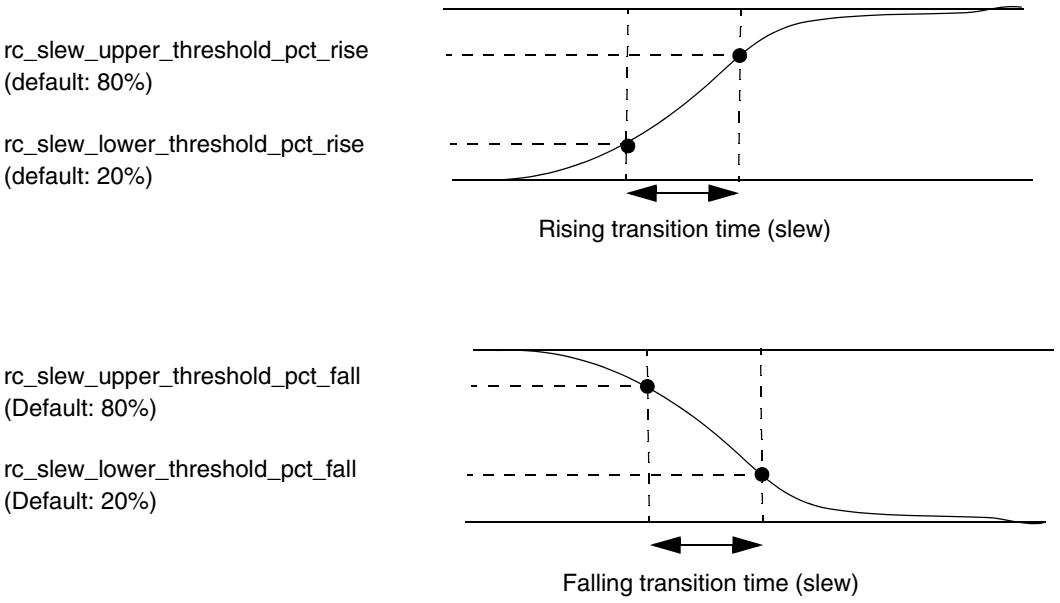
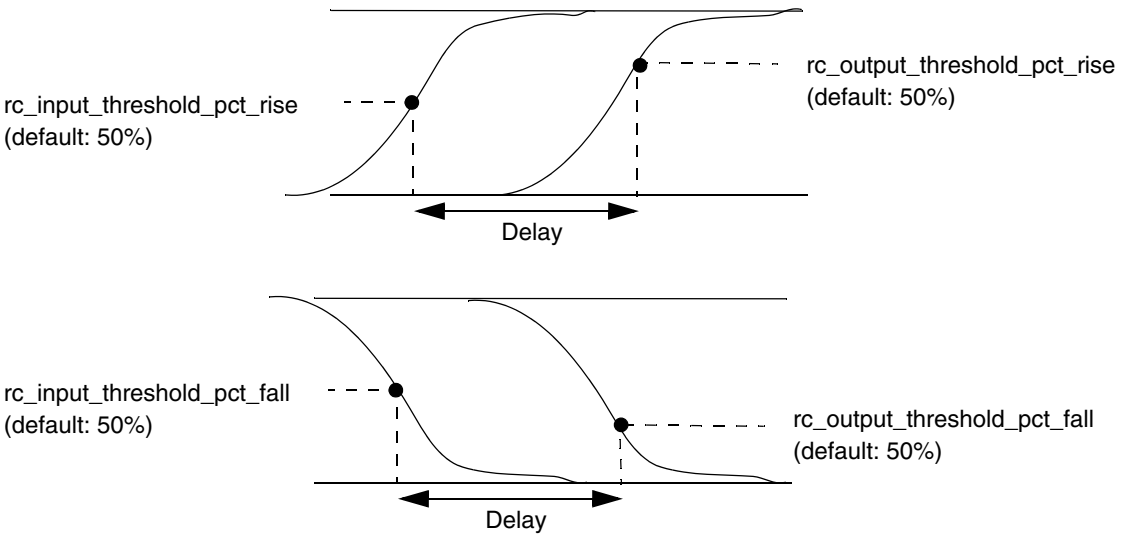


Figure 9-6 Input/Output Transition Points



The following table lists the variables that can be used to specify the trip points in the absence of trip point definitions in the technology libraries.

Variable	Default
rc_slew_lower_threshold_pct_rise	20
rc_slew_lower_threshold_pct_fall	20
rc_slew_upper_threshold_pct_rise	80
rc_slew_upper_threshold_pct_fall	80
rc_slew_derate_from_library	1
rc_input_threshold_pct_fall	50
rc_input_threshold_pct_rise	50
rc_output_threshold_pct_fall	50
rc_output_threshold_pct_rise	50

To get the current value of a variable, use the `printvar` command.

To check the threshold levels defined in a library, use the `report_lib` command. To check the threshold settings used for a particular delay calculation arc, use the command `report_delay_calculation -thresholds`.

Example

In the absence of trip point definitions in the libraries used in the design, the following commands specify a set of threshold levels that PrimeTime will use to calculate delays and slews:

```
pt_shell> set rc_slew_derate_from_library 0.4
pt_shell> set rc_slew_lower_threshold_pct_fall 30
pt_shell> set rc_slew_lower_threshold_pct_rise 30
pt_shell> set rc_slew_upper_threshold_pct_fall 70
pt_shell> set rc_slew_upper_threshold_pct_rise 70
pt_shell> set rc_input_threshold_pct_rise 50
pt_shell> set rc_input_threshold_pct_fall 50
```

The four “slew threshold” variables specify that slews were characterized by measuring the transition times from 30 to 70 percent and from 70 to 30 percent of the rail voltage. The “slew derate” variable, which is set to 0.4, specifies that the transition times were extrapolated to the rail voltages (0 to 100 percent of the supply voltage); the range of 30 to 70 percent is a

span of 40 percent of the supply voltage. The two “input threshold” and two “output threshold” variables specify that delays were calculated from trip points at 50 percent of the rail voltage.

Reading Parasitics Files

The `read_parasitics` command reads a parasitic data file in RSPF, DSPF, SPEF (version IEEE 1481-1999), or SBPF format and annotates the current design with the parasitic information. An RSPF, DSPF, or SPEF file can be ASCII text or it can be compressed with gzip. Specifying the format in the command is optional because the reader can automatically determine the file type.

Net and instance pin names in the design must match the names in the parasitics file. For example, if you create the parasitics file from a design using VHDL naming conventions, the design name must use VHDL naming conventions.

When reading parasitics files, PrimeTime assumes by default that capacitance values specified in the SPEF files do not include the pin capacitance. The pin capacitance values used by PrimeTime are the values specified in the Synopsys design libraries; any pin capacitance values specified in SPEF are ignored.

The reduced and detailed RC networks specified in SPEF files are used to compute effective capacitance dynamically during delay calculation. Note that the capacitance value reported by most report commands (such as `report_timing` and `report_net`) is the lumped capacitance, also known as Ctotal. Ctotal is the sum of all capacitance values of a net as specified in the SPEF, to which pin capacitance is also added.

Parasitic data files are often very large and time-consuming for PrimeTime to read. To minimize the read time, make sure that the data file is on a local disk that can be accessed directly (not across a network). Having enough memory (to avoid using disk swap space) is also helpful. Compressing an SPEF file using gzip can improve overall processing time because the file is so much smaller. The most efficient format is SBPF (Synopsys Binary Parasitic Format); consider using it if your tools support it. For example, to read parasitics for the current design from a file called sub_design1.rspf using the RSPF file format, enter

```
pt_shell> read_parasitics -format rspf sub_design1.rspf
```

To optimize performance, it is recommended that you create a single parasitics file in the SBPF format. You can use the following syntax:

```
pt_shell> write_parastics -format SBPF file_name
```

You can read multiple parasitics files. For example, you might have separate parasitics files for your subblocks and a separate file for your top-level interconnect. Parasitics for chip-level timing are often read incrementally. Hierarchical parasitics for each instance are read separately and stitched with top-level parasitics. The recommended flow for reading multiple hierarchical incremental parasitics files is as follows:

```
read_parasitics A.sbpf -incremental –path [all_instances -hier BLKA]/
read_parasitics B.sbpf -incremental
read_parasitics C.sbpf -incremental
read_parasitics D.sbpf -incremental
read_parasitics chip_file_name –increment
report_annotated_parasitics -check
```

The `-incremental` option enables the `read_parasitics` command to perform additional processing that allows PrimeTime to stitch together the parasitics. By default, the command does a check for incomplete annotated nets on the nets it annotates. It then executes the `report_annotated parasitics -check` command. You can use the `-path` option to take a list of instances so that multiple instantiations of the same block level design can all be annotated together. This can significantly reduce both the runtime and memory required for reading parasitics in PrimeTime. By executing the commands in the recommended flow above and explicitly using the `report_annotated_parasitics -check` command, all of the parasitics files are read in, then the entire design is checked to confirm and report the successful annotation of all nets. The report contains all of the nets in the design, not just the nets that are annotated for the last command.

Some messages issued during `read_parasitics`, `report_annotated_parasitics -check`, `read_sdf`, and an implicit or explicit `update_timing` command have a default limit each time the command is invoked. A summary of the messages affected and other useful information is stored in the log file and also printed at the end of the PrimeTime session. The `sh_message_limit` variable controls the default message limit and the `sh_limited_messages` variable controls the messages affected by this feature. For more information about the commands and variables, see the specific man page.

Scaling Parasitic Values

You can scale parasitic values that have been read into the design. There are three separate scaling factors for resistors, ground capacitors, and (for PrimeTime SI users) coupling capacitors. To scale the parasitics, you use the `scale_parasitics` command.

Each factor is multiplied against all back-annotated values of the specified type throughout the design. A factor greater than 1.0 increases the parasitic component values, whereas a factor less than 1.0 decreases the parasitic component values. Factors must be greater than or equal to 0.0.

The `scale_parasitics` command operates immediately to modify the parasitics in memory. If you use the `scale_parasitics` command more than once in a session, the factor is applied to the scaled values, not the original values read from the parasitic data file. If you write the design parasitics with the `write_parasitics` command, the scaled values (not original values) are written.

Net-Specific Parasitic Scaling

By using the `scale_parasitics` command with a list of nets, you can apply different scaling factors to individual nets instead of using a global scaling factor. You can scale resistance, ground capacitance, and coupling capacitance values.

The `report_scale_parasitics` command allows you to see a report of the scale parasitics. The `reset_scale_parasitics` command allows you to reset the parasitics to the global value. You can inspect the changes to the scaled parasitics using the `report_annotated_parasitics -list_annotated` command.

Note:

Net-specific parasitic scaling does not incur a full timing update, only an incremental update.

Ground-Capacitance and Resistance Scaling

You can scale all ground capacitances on a net by a given scaling factor. A scaling factor is a positive floating point number greater than zero. For example,

```
pt_shell> scale_parasitics [get_net n1] -ground 1.3
```

You can scale all resistances of a net by another scaling factor:

```
pt_shell> scale_parasitics [get_net n1] -resistance 1.22
```

If you scale the net multiple times or you scale globally, followed by net-specific scaling, the last command issued takes effect with respect to the original (not the previous value). For example, if the following sequence is executed, the ground capacitance of net n1 is scaled by 1.2, *not* 1.3 multiplied by 1.2:

```
pt_shell> scale_parasitics -ground 1.3 ## scale all nets
pt_shell> scale_parasitics [get_net n1] -ground 1.2 ## scale n1
```

Coupling-Capacitance Scaling

You can scale all the coupling capacitances of a net by a scaling factor.

```
pt_shell> scale_parasitics -coupling 1.3 [get_net n1]
```

If you scale the two coupled nets by different factors, then the later command takes effect. For example,

```
pt_shell> scale_parasitics -coupling 1.3 [get_net n1]
pt_shell> scale_parasitics -coupling 1.2 [get_net n2]
```

The net effect is that the all the coupling of net n1 is scaled by 1.3 except the coupling to n2, and all the coupling of net n2 is scaled by 1.2 including the coupling to n1.

Resetting Scale Parasitics

You can use the `reset_scale_parasitics` command to reset the scaled nets to their original values. You can only issue this command on nets that have been previously scaled.

```
pt_shell> reset_scale_parasitics [get_nets n1]
```

The `reset_scale_parasitics` command returns the values to the original values, not necessarily the previous values. For example,

```
pt_shell> scale_parasitics -ground 1.3 ## scale all nets
pt_shell> scale_parasitics [get_net n1] -ground 1.2 ## scale n1
pt_shell> reset_scale_parasitics [get_net n1] ## Unscale n1
```

The net effect is that there is no scaling on net n1.

You can also use the `reset_scale_parasitics` command to regain original coupling values:

```
pt_shell> scale_parasitics -coupling 1.3 [get_net n1]
pt_shell> scale_parasitics -coupling 1.2 [get_net n2]
pt_shell> reset_scale_parasitics [get_net n1]
```

The net effect of this sequence of commands is that the coupling of n1 is unchanged, and all coupling of n2 except the coupling between n1 and n2 is scaled by 1.2.

Reporting Scale Parasitics

You can use the `report_scale_parasitics` command to look at the current scale factors of all scaled nets or a list of scaled nets:

```
pt_shell> report_scale_parasitics [get_nets n1]

Net name     Ground cap factor   Resistance factor   Coupling factor
  n1                --                  1.2              0.98
```

This means that net n1 is not scaled for ground capacitance: its resistance is scaled by a factor of 1.2, and its coupling capacitance is scaled by a factor of 0.98.

Note:

Be aware of the following possible scenario:

```
pt_shell> scale_parasitics -coupling 1.3 [get_net n1]
pt_shell> scale_parasitics -coupling 1.2 [get_net n2]
pt_shell> report_scale_parasitics [get_net n1]
```

This will show a coupling scale factor of 1.3 even though the coupling between net n1 and net n2 is scaled by a different factor, 1.2.

Examples

The following three examples illustrate net-specific parasitic scaling usage:

- Single net scaling
- Physical block (net) scaling
- Scaling due to net separation

Single Net Scaling

This example illustrates scaling the resistance and ground capacitance of a selected net. The output of the `report_annotated_parasitics` command without any scaling is as follows:

```
****************************************
Report : annotated_parasitics
      -internal_nets
      -boundary_nets
      -list_annotated
      -max_nets 10
Design : Chip
Version: X-2005.06-Beta1-DEV
Date   : Tue Mar 29 15:47:40 2005
****************************************
1.  ADDR[1] (driver: port ADDR[1])

  C in pF        Node    Pin/Port
  -------------------------------------------------------------
  0.00842337       1     ADDR[1] (driver) [+pin_cap=0pF]
  0.0170247        2     --
  0.00531228       3     --
  0.00702817       4     --
  0.00487712       5     --
  0.00397756       6     --
  0.00590386       7     --
  0.0122464        8     --
  0.00170559       9     --
  0.00611351      10     --
  0.00304156      11     --
  0.00395301      12     --
```

```
0.00607125      13     --
0.00227354      14     --
0.0119327       15     --
0.00564987      16     --
0.00749174      17     --
0.00585252      18     --
0.0083778       19     --
0.0020378       20     --
0.00111427      21     U527/A1 (load) [+pin_cap=0.00103pF]
0.00159791      22     U459/B1 (load) [+pin_cap=0.00112pF]
1e-06           23     U2053/A1 (load) [+pin_cap=0.00075pF]
1e-06           24     U483/A1 (load) [+pin_cap=0.00107pF]
1e-06           25     U534/A1 (load) [+pin_cap=0.00191pF]
1e-06           26     U503/A1 (load) [+pin_cap=0.00189pF]
-------------------------------------------------------------
0.13201      Total

CC in pF     Local Node                Other Node
-------------------------------------------------------------
0.00197175    13                       n4250:13
0.00139652    17                       n4250:6
0.00218098     3                       n2124:2
0.00141374     1 ADDR[1] (driver)      REG_DATA[22]:2
0.000183647   16                       n8376:2
0.000175112    6                       n13565:2
0.000433937    6                       n13599:4
0.000633856    8                       n4750ASTipoNet3769:1
0.000346092    8                       n1946:1
0.00274598     3                       n3117:1
0.00033236    17                       n7632:1
0.00067407    12                       n11321:1
0.000256377    2                       BW0_30_:2
0.000994872    3                       n225:2
0.000264488   15                       n1785:2
0.00157215     9                       n5870ASThfnNet347:5
0.00130062     2                       REG_DATA[26]:2
0.00170201     4                       n2248ASThfnNet432:10
0.00157756    14                       n2248ASThfnNet432:13
-------------------------------------------------------------
0.0201561    Total

R in Kohm    Left node                 Right Node
-------------------------------------------------------------
0.198293       1 ADDR[1] (driver)        2
0.0190159      2                         8
0.00350119     3                        11
0.027464       3                         5
0.00235238     4                        14
0.00688478     4                        11
0.00561526     5                        20
0.000852706    6                        16
0.0239033      6                        21 U527/A1 (load)
0.0380269      6                        22 U459/B1 (load)
```

```
  0.0397274      6                          12
  0.0257463      7                           9
  0.0313717      7                          12
  0.00521104     8                          19
  0.00360633     9                          10
  0.0152421     10                          13
  0.0923591     12                          21 U527/A1 (load)
  0.146931      12                          22 U459/B1 (load)
  0.00413988    13                          17
  0.00264083    14                          17
  0.0060443     15                          18
  0.00321024    15                          19
  0.0329064     16                          23 U2053/A1 (load)
  0.0285095     16                          24 U483/A1 (load)
  0.0187162     16                          25 U534/A1 (load)
  0.0200831     16                          26 U503/A1 (load)
  0.00884356    18                          20
  0.0230755     26 U503/A1 (load)  25 U534/A1 (load)
  0.0667172     21 U527/A1 (load)  22 U459/B1 (load)
  0.00850544    23 U2053/A1 (load)         24 U483/A1 (load)
  ------------------------------------------------------------
  0.909497     Total
```

Net Type	Total	Lumped	RC pi	RC network	Coupled network	Not Annotated
Internal nets	0	0	0	0	0	0
-Driverless nets		0	0	0	0	0
Boundary/port nets	1	0	0	0	1	0
- Driverless nets		0	0	0	0	0
	1	0	0	0	1	0

After the parasitics are scaled for resistance and ground capacitance, the output of the `report_annotated_parasitics` command would be as follows:

```
1
****************************************
Report : annotated_parasitics
      -internal_nets
      -boundary_nets
      -list_annotated
      -max_nets 10
Design : Chip
Version: X-2005.06-Beta1-DEV
Date   : Tue Mar 29 15:48:57 2005
****************************************

1.  ADDR[1] (driver: port ADDR[1])
  C in pF        Node    Pin/Port
  ------------------------------------------------------------------
```

```
0.0168467        1     ADDR[1] (driver) [+pin_cap=0pF]
0.0340494        2     --
0.0106246        3     --
0.0140563        4     --
0.00975424       5     --
0.00795512       6     --
0.0118077        7     --
0.0244928        8     --
0.00341118       9     --
0.012227        10     --
0.00608312      11     --
0.00790602      12     --
0.0121425       13     --
0.00454708      14     --
0.0238654       15     --
0.0112997       16     --
0.0149835       17     --
0.011705        18     --
0.0167556       19     --
0.0040756       20     --
0.00222854      21     U527/A1 (load) [+pin_cap=0.00103pF]
0.00319582      22     U459/B1 (load) [+pin_cap=0.00112pF]
2e-06           23     U2053/A1 (load) [+pin_cap=0.00075pF]
2e-06           24     U483/A1 (load) [+pin_cap=0.00107pF]
2e-06           25     U534/A1 (load) [+pin_cap=0.00191pF]
2e-06           26     U503/A1 (load) [+pin_cap=0.00189pF]
-----------------------------------------------------------------
0.264021     Total

CC in pF     Local Node                Other Node
-----------------------------------------------------------------
0.00197175    13                       n4250:13
0.00139652    17                       n4250:6
0.00218098     3                       n2124:2
0.00141374     1 ADDR[1] (driver)      REG_DATA[22]:2
0.000183647   16                       n8376:2
0.000175112    6                       n13565:2
0.000433937    6                       n13599:4
0.000633856    8                       n4750ASTipoNet3769:1
0.000346092    8                       n1946:1
0.00274598     3                       n3117:1
0.00033236    17                       n7632:1
0.00067407    12                       n11321:1
0.000256377    2                       BW0_30_:2
0.000994872    3                       n225:2
0.000264488   15                       n1785:2
0.00157215     9                       n5870ASThfnNet347:5
0.00130062     2                       REG_DATA[26]:2
0.00170201     4                       n2248ASThfnNet432:10
0.00157756    14                       n2248ASThfnNet432:13
-----------------------------------------------------------------
0.0201561    Total
```

```
  R in Kohm    Left node                  Right Node
  ------------------------------------------------------------------
  0.396586       1 ADDR[1] (driver)        2
  0.0380318      2                         8
  0.00700238     3                        11
  0.054928       3                         5
  0.00470476     4                        14
  0.0137696      4                        11
  0.0112305      5                        20
  0.00170541     6                        16
  0.0478066      6                        21 U527/A1 (load)
  0.0760538      6                        22 U459/B1 (load)
  0.0794548      6                        12
  0.0514926      7                         9
  0.0627434      7                        12
  0.0104221      8                        19
  0.00721266     9                        10
  0.0304842     10                        13
  0.184718      12                        21 U527/A1 (load)
  0.293862      12                        22 U459/B1 (load)
  0.00827976    13                        17
  0.00528166    14                        17
  0.0120886     15                        18
  0.00642048    15                        19
  0.0658128     16                        23 U2053/A1 (load)
  0.057019      16                        24 U483/A1 (load)
  0.0374324     16                        25 U534/A1 (load)
  0.0401662     16                        26 U503/A1 (load)
  0.0176871     18                        20
  0.046151      26 U503/A1 (load)  25 U534/A1 (load)
  0.133434      21 U527/A1 (load)  22 U459/B1 (load)
  0.0170109     23 U2053/A1(load)  24 U483/A1 (load)
  ------------------------------------------------------------------
  1.81899      Total
```

Net Type	Total	Lumped	RC pi	RC network	Coupled network	Not Annotated
Internal nets	0	0	0	0	0	0
-Driverless nets		0	0	0	0	0
Boundary/port nets	1	0	0	0	1	0
- Driverless nets		0	0	0	0	0
	1	0	0	0	1	0

Note that only the resistance and the ground capacitance have been scaled by 2. All delay calculation and PrimeTime SI analysis would be updated by the new values of these parasitics.

Physical Block (Net) Scaling

This example illustrates how you can scale each net in a physical block in your design. Foundries can provide different scaling values for different conditions of a block (process, temperature, or voltage), and the use of parasitics scaling can reflect this variation.

To use this methodology, link the top level of the design (Chip, in this example). You can then switch to the physical block of interest by using the following:

```
current_instance A

set gnets [get_net *] # This will produce a collection of
                      # nets for block A.

scale_parasitics -ground 1.1 -resis 1.1 -coupling 1.1 $gnets

current_instance Chip

...

# Proceed with the analysis
```

Each net in block A, will be scaled as specified by the `scale_parasitics` command. Keep in mind that nets that cross hierarchies will also be scaled. You can inspect the changes to the parasitics using the `report_annotated_parasitics -list_annotated` command:

Scaling Due to Net Separation

PrimeTime now has the capability to scale the coupling capacitance due to net separation. By spacing nets, you can reduce the coupling capacitance between the nets. This can be reflected using the `scale_parasitics` command. For example,

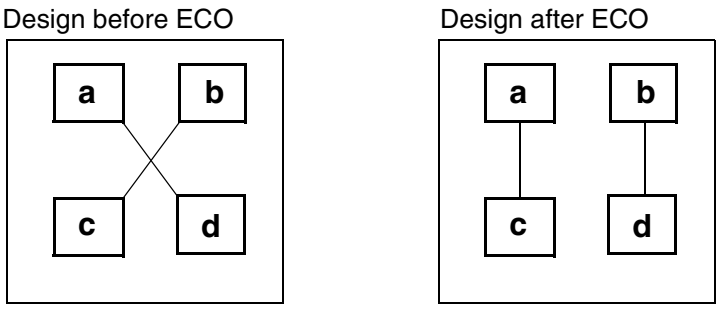
```
pt_shell> scale_parasitics -coupling 0.75 [get_net Net1]
```

Each of the coupling capacitances for Net1 will be scaled. Therefore, in this case both capacitors, from Net1 to Net2, and from Net2 to Net1, will be scaled.

Incremental Timing Analysis

PrimeTime can perform an incremental update on a design that is already analyzed when only a small number of nets are affected. The maximum number of nets that can be affected without requiring a full timing update depends upon the total design size. For example, if you have a full-chip annotated file and an engineering change order (ECO) annotated file, and you want to analyze the impact of your ECO, you can run an analysis with the `report_timing` command on a full-chip annotated file, and then repeat the analysis with just the ECO changes applied. See Figure 9-7.

Figure 9-7 Incremental Update Before and After Engineering Change



To set the annotation on most—or all—nets, enter

```
pt_shell> read_parasitics full_chip.dspf
pt_shell> report_timing
```

To override the annotations on a small number of nets, enter

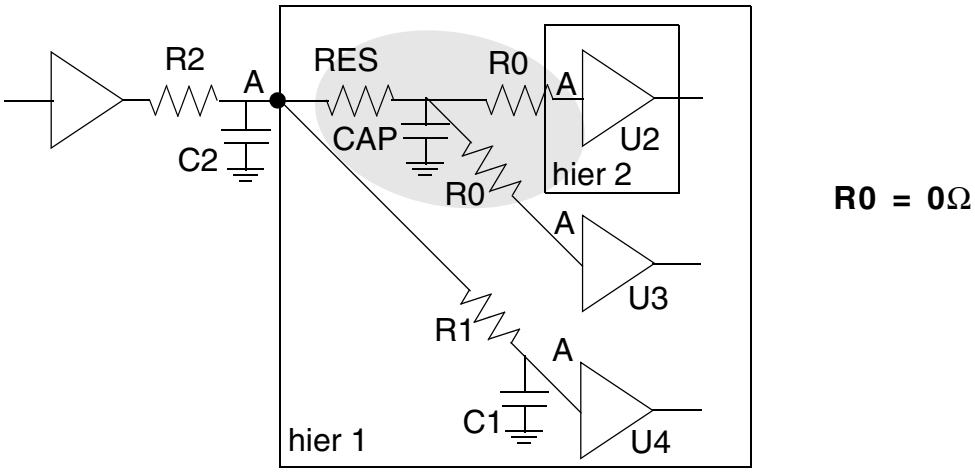
```
pt_shell> read_parasitics eco.dspf
pt_shell> report_timing
```

Incomplete Annotated Parasitics

PrimeTime can complete parasitics on nets that have incomplete parasitics. The following conditions apply to RC network completion:

- The nets that have an RC network must be back-annotated from DSPF or SPEF files.
- PrimeTime cannot complete effective capacitance calculation for a net that has incomplete parasitics. Instead, PrimeTime reverts to using wire load models for delay calculation on these nets.
- Partial parasitics can be completed on a net only if all the missing segments are between two pins (either boundary pins or leaf pins). As shown in Figure 9-8, the missing segment is completed with a fanout of two.

Figure 9-8 Missing Segment Completed Using the Fanout of Two



The shaded area shown in Figure 9-8 is completed when you use the `complete_net_parasitics` command or the `-complete_with` option of the `read_parasitics` command.

Selecting a Wire Load Model for Incomplete Nets

Once the missing segments are identified, PrimeTime selects the wire load model to complete the missing parts of the net as follows:

- The wire load mode is taken from the top-level hierarchy. If the wire load mode does not exist, the default from the main library is taken.
- If the wire load mode is ‘enclosed,’ the wire load for the missing segment is the hierarchy that encloses the missing segment.
- If the wire load mode is ‘top,’ the wire load for the top level of hierarchy is used.
- If the wire load model does not exist on the enclosing hierarchy, the wire load model of the parent hierarchy is taken. If the parent hierarchy does not exist, the wire load model of the parent of the parent is used, this process continues until the top-level hierarchy is reached.
- The default wire load model from the main library is used if the wire load model cannot be obtained from the previous steps.
- Zero resistance and capacitance are used if no wire load model could be obtained using the `-complete_with wlm` option.

Completing Missing Segments on the Net

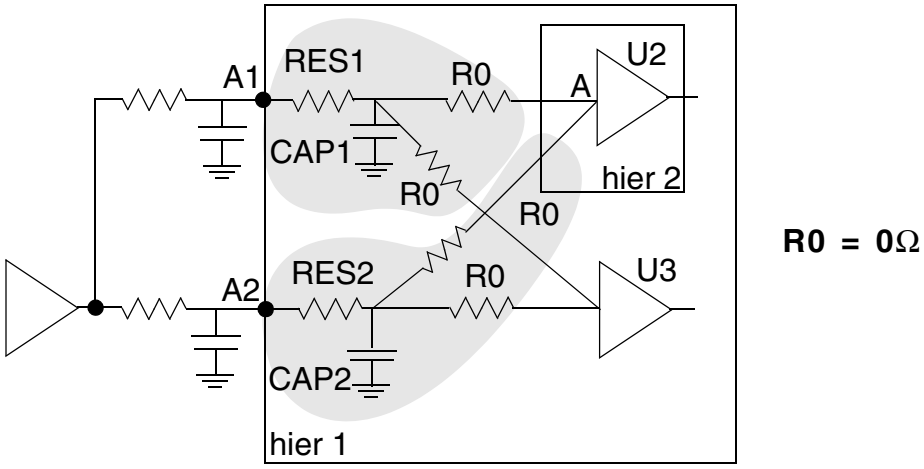
PrimeTime completes multiple pin to pin segments with a single RC pair based on the value of the option `-complete_with`, which can be `zero` or `wlm` (wire load model).

Here is a command sequence that completes the missing segments:

```
pt_shell> read_ddc design.ddc
pt_shell> read_parasitics incomplete_parasitics.spef
pt_shell> complete_net_parasitics -complete_with wlm
```

As shown in Figure 9-9, if the `-complete_with wlm` option is used, the resistance (RES) and capacitance (CAP) values are estimated based on the wire load model. If `-complete_with zero` is used, the values assigned to RES and CAP are zero. In both cases, the resistance that connects the net to the loads is 0 ohms.

Figure 9-9 Multidrive Segments



You can see how networks get completed by comparing the attribute `rc_network` defined for the incomplete nets before and after completion.

After the segment is completed, it cannot be undone. For this reason, if you are using multiple `read_parasitics` commands with the `-increment` option, be sure to use the `-complete_with` option only with the very last `read_parasitics` command. Otherwise, PrimeTime will complete the network before you have finished reading in all of the parasitic data files. To cancel the effects of all `read_parasititics` and `complete_net_parasitics` commands, use the `remove_annotated_parasitics` command.

Do not use the `complete_net_parasitics` command when you have parasitics with errors. However, you can manually fix the DSPF or SPEF files and reread them. Use the `complete_net_parasitics` command only when the following assumption holds true: The

relevant portion of a net's delay is already annotated with detailed parasitics, and you want PrimeTime to complete the remaining, less significant portion of the net with zero or wire load model based resistance and capacitance.

Reporting Annotated Parasitics

Parasitic data files can be large (100 MB or larger) and can contain many parasitics. You can use the `report_annotated_parasitics` command after reading an SPEF, RSPF, or DSPF file to verify that PrimeTime back-annotated all cell drivers in the design. For more information, see the `report_annotated_parasitics` man page.

To create a report for annotated parasitic data files, enter

```
pt_shell> report_annotated_parasitics -check
```

The `-check` option causes PrimeTime to verify that all RC networks are complete. PrimeTime displays a report similar to this:

```
****************************************
Report : annotated_parasitics
         -check
Design : top_rc1
Version: 2000.11
Date   : Thur Nov 16 14:22:37 2000
****************************************
                   |          |          |    RC     |    Not    |
   Pin type        |  Total   |  RC pi   | network   |Annotated|
-------------------+---------+---------+---------+---------
+
internal net drive|   23649 |       0 |   23649 |       0 |
-------------------+---------+---------+---------+---------
+
design input port |      34 |       0 |      34 |       0 |
-------------------+---------+---------+--------+----------
+
                   |   23683 |       0 |   23683 |       0 |
```

Removing Annotated Parasitics

The `remove_annotated_parasitics` command removes annotated parasitics from nets. You can remove annotated parasitics from all nets or just a specified set of nets. The `reset_design` command also removes all parasitics read and annotated using the `read_parasitics` command.

10

Delay Calculation With Detailed Parasitics

This chapter describes how PrimeTime calculates delays in the presence of detailed parasitics and the types of models used to calculate these delays.

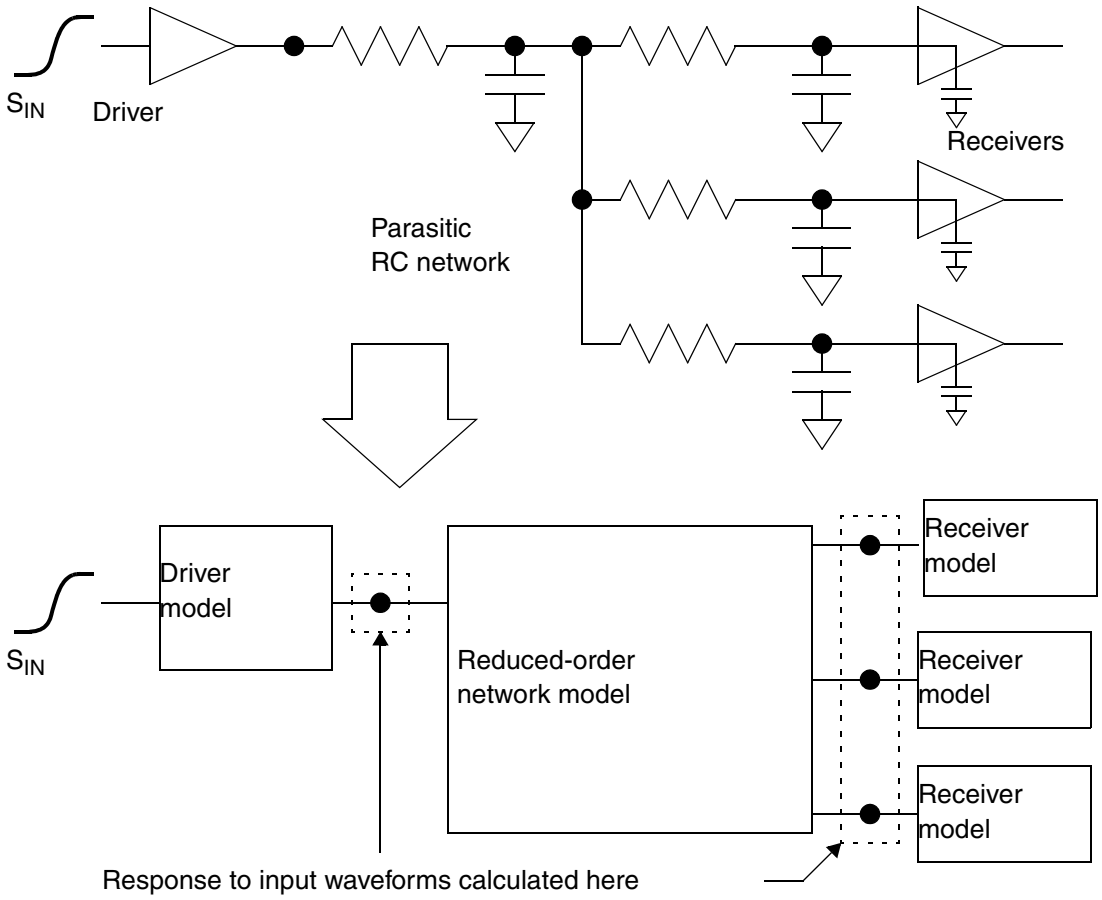
This chapter contains the following sections:

Overview of Delay Calculation

To perform static timing analysis, PrimeTime must accurately calculate the delay and slew (transition time) at each stage of each timing path. A stage consists of a driving cell, the annotated RC network at the output of the cell, and the capacitive load of the network load pins. The goal is to compute the response at the driver output and at the network load pins, given the input slew or waveform at the driver input, using the least amount of runtime necessary to get accurate results.

To perform stage delay calculation accurately and efficiently, PrimeTime uses models to represent the driver, RC network, and capacitive loads on the net. See Figure 10-1. An ideal model would produce exactly the same delays and slews as a SPICE simulation at the output of the driver and at the input of each receiver.

Figure 10-1 Models Used to Calculate Stage Delays and Slews



The driver model is intended to reproduce the response of the driving cell's underlying transistor circuitry when connected to an arbitrary RC network, given a specific input slew.

The reduced-order network model is a simplified representation of the full annotated network that has nearly the same response characteristics as the original network. PrimeTime uses the Arnoldi reduction method to create this model.

The receiver model is intended to represent the complex input capacitance characteristics of a cell input pin, including the effects of rise/fall direction, the slew at the pin, the receiver output load, the state of the cell, and the voltage and temperature conditions.

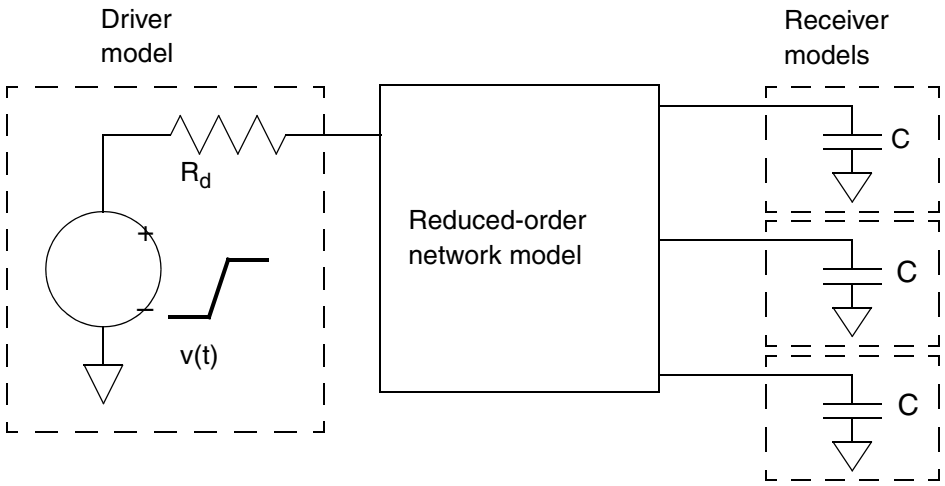
PrimeTime supports two types of driver models for RC delay calculation, known as basic and advanced. When the `rc_driver_model_mode` and `rc_receiver_model_mode` variables are set to `basic`, RC delay calculation uses only the driver and receiver models derived from the more basic nonlinear delay model (NLDM) present in the cell libraries. When the variables are set to `advanced`, RC delay calculation uses the newer, more advanced Composite Current Source (CCS) timing model driver and receiver models, if data for those models are present in the cell libraries; otherwise, the basic models are used. The default setting for both variables is `advanced`.

The advanced CCS timing model has many advantages, one of which is the solution to the problem described by the RC-009 warning message. This warning occurs when the drive resistance of the driver model is much less than the network impedance to ground. The CCS timing model is also better at handling the Miller Effect, dynamic IR drop, and multivoltage analysis.

Nonlinear Delay Models (NLDM)

The NLDM is the earlier, established method of representing the driver and receiver of a path stage. The driver model uses a linear voltage ramp in series with a resistor (a Thevenin model), as shown in Figure 10-2. The resistor helps smooth out the voltage ramp so that the resulting driver waveform is similar to the curvature of the actual driver driving the RC network.

Figure 10-2 NLDM Driver and Receiver Models



The driver model has three model parameters: the drive resistance R_d, the ramp start time t_Z, and the ramp duration delta *t*. PrimeTime chooses parameter values to match the output waveforms as closely as possible. It builds a different simplified driver model for each gate timing arc (for example, from U1/A to U1/Z) and for each sense (for example, rising edge).

When the drive resistor is much less than the impedance of the network to ground, the smoothing effect is reduced, potentially reducing the accuracy of RC delay calculation. When this condition occurs, PrimeTime adjusts the drive resistance to improve accuracy and issues an RC-009 warning. For more information, see the man page on the RC-009 warning.

The NLDM receiver model is a capacitor that represents the load capacitance of the receiver input. A different capacitance value may apply to different conditions such as rising/falling transitions or min/max timing analysis. A single capacitance value, however, applies to a given timing check, which does not support accurate modeling of the Miller Effect.

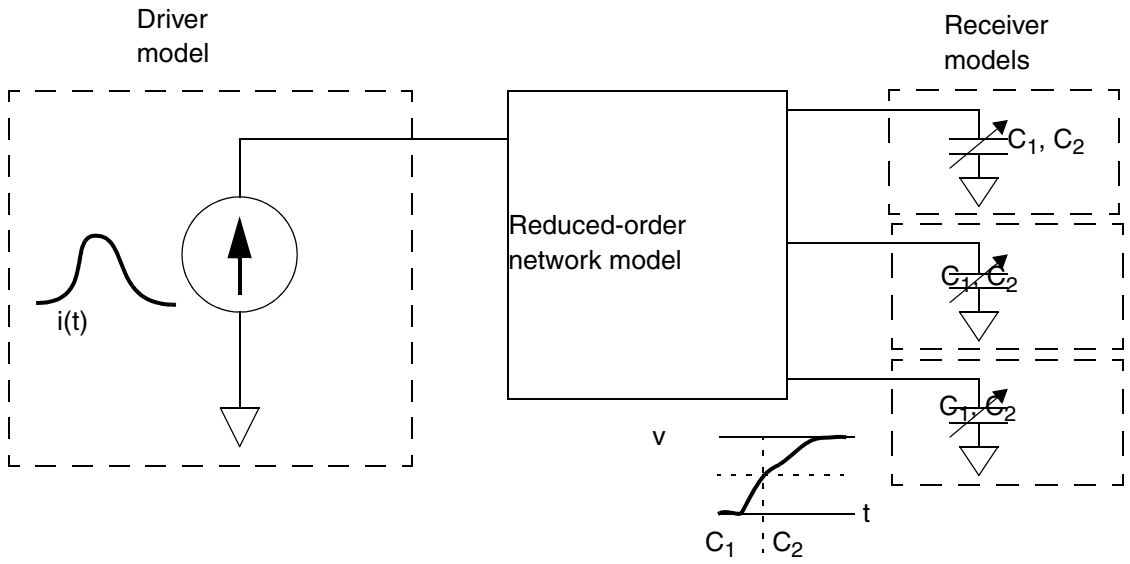
Note:

The Miller Effect is the effective change in capacitance between transistor terminals that occurs with a change in voltage across those terminals.

CCS Timing Models

With the advent of smaller nanometer technologies, the CCS timing approach of modeling cell behavior has been developed to address the effects of deep submicron processes. The driver model uses a time-varying current source, as shown in Figure 10-3. The advantage of this driver model is its ability to handle high-impedance nets and other non-monotonic behavior accurately.

Figure 10-3 CCS Timing Driver and Receiver Models



The CCS timing receiver model uses two different capacitor values rather than a single lumped capacitance. The first capacitance is used as the load up to the input delay threshold. When the input waveform reaches this threshold, the load is dynamically adjusted to the second capacitance value. This model provides a much better approximation of loading effects in the presence of the Miller Effect.

In some cases, different input signals can affect the input capacitance of the receiver. Conditional pin-based models are used in the library to describe the pin capacitance for different input signals. If there are conditional pin-based receiver models in the library, PrimeTime considers all receiver models and chooses the worst among the enabled pin-based and arc-based receiver models for the analysis.

In PrimeTime, the CCS timing analysis requires detailed parasitics. It uses library information in the following order for delay calculation with detailed parasitics:

1. CCS timing driver and receiver models, if both are available.

2. CCS timing driver model, if available, and lumped pin capacitance for the receiving cells.
3. NLDM delay and transition tables and pin capacitance for the receiving cells. (You can use a CCS timing receiver model only with a CCS timing driver model.)

After a timing update, to determine whether CCS timing data was used for delay calculation, you can use the `report_delay_calculation` command. In the command, specify the cell or net by using the `-from` and `-to` options, or specify a timing arc using the `-of_objects` option. The following examples show CCS timing driver and receiver model data used for a cell and net delay calculation.

```
pt_shell> report_delay_calculation -from cell1/A -to cell1/Z
.........
arc sense: positive_unate
arc type:  cell

Calculation  Rise   Rise    Fall   Fall   Slew    Rail
Thresholds:  Delay  Slew    Delay  Slew   Derate  Voltage Temp.
-------------------------------------------------------------
   from-pin  50     30->70  50     70->30 0.400   1.100   125.0
     to-pin  50     30->70  50     70->30 0.400   1.100   125.0

RC network on pin 'cell1/Z' :
-----------------------------------------------------
Number of elements = 8 Capacitances + 7 Resistances
Total capacitance  = 0.03062 pF
Total capacitance  = 0.03062 (in library unit)
Total resistance   = 0.017983 Kohm

Advanced driver-modeling used for rise and fall.

                           Rise        Fall
---------------------------------------------------------------------
Input transition time  = 0.100000    0.100000  (in library unit)
Effective capacitance  = 0.002625    0.002800  (in pF)
Effective capacitance  = 0.002625    0.002800  (in library unit)
Output transition time = 0.060388    0.047040  (in library unit)
Cell delay             = 0.050937    0.045125  (in library unit)

pt_shell> report_delay_calculation -from [get_pins cell/Z] \
          -to [get_pins receiver/A]
   ...
   From pin: cell/Z
   To pin:   receiver/A
   Main Library Units:  1ns  1pF  1kOhm
   arc sense: unate
   arc type:  net
   Calculation  Rise   Rise   Fall   Fall   Slew    Rail
   Thresholds:  Delay  Slew   Delay  Slew   Derate  Voltage Temp
   ------------------------------------------------------------
      from-pin  50     30->70   50   70->30  1.000  0.900  125.0
        to-pin  50     30->70   50   70->30  1.000  0.900  125.0

      RC network on pin 'cell/Z' :
      -----------------------------------------------------
      Number of elements = 8 Capacitances + 7 Resistances
```

```
  Total capacitance  = 0.003062 pF
  Total capacitance  = 0.003062 (in library unit)
  Total resistance   = 0.017983 Kohm
  Advanced receiver-modeling used for rise and fall.

  Advanced Receiver Model
                                    Rise        Fall
 ---------------------------------------------------------------------------
Receiver model capacitance       1 = 0.001966    0.001888  (in library unit)
Receiver model capacitance       2 = 0.002328    0.002160  (in library unit)

                              Rise       Fall
 ---------------------------------------------------------------------
 Net delay                = 0.000024    0.000064   (in library unit)
 Transition time          = 0.059192    0.037599   (in library unit)
 From_pin transition time = 0.059078    0.037666   (in library unit)
 To_pin transition time   = 0.059192    0.037599   (in library unit)
 Net slew degradation     = 0.000114   -0.000067   (in library unit)
```

Guidelines to Address the CCS Extrapolation Warning Message (RC-011)

Extrapolation warning messages (RC-011) occur when RC delay-calculation is attempted using a slew or load that is much larger than the maximum library slew or load indices of a cell or pin. Large extrapolations can cause inaccurate results.

The following sections describe the recommendations to address the driver extrapolation and receiver load extrapolations:

- Guidelines for Characterizing Design Rule Constraints (DRC)
- Guidelines for Fixing RC-011 Warning Messages

Guidelines for Characterizing Design Rule Constraints (DRC)

The `max_transition` pin attributes are normally present on the input and output pins of library cells. For input pins, the `max_transition` attribute value should not exceed the maximum slew index in the NLDM and CCS driver and CCS receiver_capacitance2 tables. The lowest value of the maximum slew index between the NLDM and CCS tables should be used as a reference. The tables used as reference are for the rising and falling timing arcs from the relevant input pin for which the `max_transition` attribute is being characterized. You should take both the arc-based and pin-based tables into account.

The `max_capacitance` pin attributes are normally present on the output pins of library cells. For output pins, the `max_capacitance` attribute value should not exceed the maximum load index in the NLDM and CCS driver as well as CCS receiver_capacitance1 and receiver_capacitance2 tables. You should use the lowest value of the maximum load index between the NLDM and CCS tables as a reference. The tables used as reference are for the

rising and falling timing arcs to the relevant output pin for which the max_capacitance attribute is being characterized. You should take both the arc-based and pin-based tables into account.

Guidelines for Fixing RC-011 Warning Messages

If the libraries used in the design follow the guidelines outlined in the Guidelines to Address the CCS Extrapolation Warning Message (RC-011) section, all RC-011 messages are addressed by fixing the max_transition and max_capacitance violations reported by the report_constraint command. If the libraries are not compliant with these guidelines, you should consider RC-011 warning messages to be important. You need to address the DRCs in the design to fix these warnings.

Scaling With CCS Timing Libraries

PrimeTime supports voltage and temperature scaling by interpolating between data in separate libraries that have been characterized at different nominal voltage and temperature values. The delay (CCS timing driver model and receiver model) and timing constraints are scaled. In addition, scaling occurs if there is a mixture of CCS and NLDM data. Scaling between the libraries is done during runtime of the tool.

Invoking Scaling

You can invoke voltage and temperature scaling by using the define_scaling_lib_group command. This command specifies the scaling relationships between libraries that have been characterized at different voltages and temperatures and invokes both delay and constraint scaling. You can define scaling relationships between multiple libraries as shown in the following example:

```
pt_shell> define_scaling_lib_group \
            {lib_0.9V_0C.db lib_1.1V_0C.db lib_1.3V_0C.db}
```

This command should be issued after the design has been read in. If the design is not already linked, the define_scaling_lib_group command automatically links the design and creates scaling relationships between the libraries in each group. If the design is already linked, this command creates scaling relationships without an additional link. There is no restriction on the number of libraries you can specify in the command, and you can define multiple scaling groups to cover different portions of your design. Each library, however, can be part of only one scaling group.

Guidelines for Scaling

When using the `define_scaling_lib_group` command:

1. Read in the design netlist before defining scaling relationships.
2. Only one library in the group should be in the link path. The remaining libraries are read in automatically during creation of the scaling group. Each library can be part of only one scaling group.
3. While creating scaling relationships, PrimeTime reports the following information:

```
Completing scaling library groups ...
Loading db file './lib/lib_0.9V_0C.db'
Loading db file './lib/lib_1.3V_0C.db'
... the scaling library groups are complete.
```

   During this process, PrimeTime does consistency checking between the libraries. The cell names, pin names, and timing arcs need to be identical between libraries, and the output capacitance table indices must be the same for each timing arc.

   All libraries in the scaling group must be consistent in terms of the existence of the receiver model. For example, for a specific library pin, either all libraries have a CCS receiver model or all the libraries do not have any CCS receiver model.

4. You can set the supply voltages and temperature of each cell using the `create_operating_conditions` and `set_operating_conditions` commands. If you are using IEEE 1801 (UPF) as part of a low-power design, you should use the `set_voltage` and `set_temperature` commands. Ensure that each specified voltage and temperature value is within the range of the applicable scaling library group.
5. If a library is removed from memory with the `remove_lib` command, the affected library group is removed without warning.

The `report_delay_calculation` command provides additional information when scaling is used for the delay calculation. It reports which transitions (rise and/or fall) used scaling and which libraries were used for delay calculation.

```
pt_shell> report_delay_calculation -from [get_pins cell/A] \
           -to [get_pins cell/Z]

...

arc sense: negative_unate
arc type:  cell

   RC network on pin 'cell/Z' :
   ----------------------------------------------------
   Number of elements = 4 Capacitances + 3 Resistances
   Total capacitance  = 0.015272 pF
```

```
   Total capacitance  = 15.271626 (in library unit)
   Total resistance   = 0.027793 Kohm

   Advanced driver-modeling used for rise and fall.
   Scaling library arc group used for rise and fall.
   Scaling libraries used for driver model :
     /remote/testcase/lib/ccs_lib_1.05.db:ccs_1.05
     /remote/testcase/lib/ccs_lib_0.85.db:ccs_0.85

                              Rise      Fall
-----------------------------------------------------------
 Input transition time  =  0.600000  0.600000 (in library unit)
 Effective capacitance  =  0.015272  0.015272 (in pF)
 Effective capacitance  = 15.271626 15.271626 (in library
unit)
 Output transition time =  0.091507  0.088209 (in library
unit)
 Cell delay             =  0.031275  0.112788 (in library unit)
```

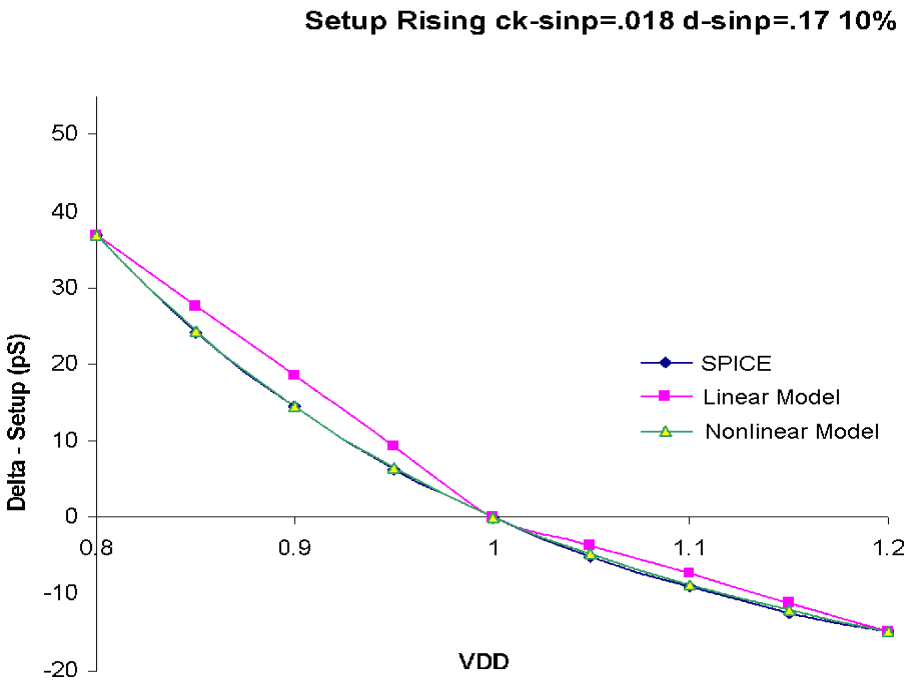
The `report_lib_groups` command can be used to find all libraries with scaling relationships in the design. For example,

```
pt_shell> report_lib_groups -scaling -show {voltage temperature}

...

           Group    Library      Temperature   Voltage

-----------------------------------------------------------
           Group 1
                    lib_0.85      125.00        0.85
                    lib_1.05      125.00        1.05
```

Scaling Interpolation for Constraints

For setup, hold, recovery, removal, minimum pulse width, and minimum period constraints, PrimeTime uses a nonlinear interpolation method for voltage variation. Figure 10-4 shows a comparison of SPICE data, linear interpolation, and nonlinear interpolation for a D flip-flop setup constraint in the voltage range between 0.8 and 1.2 volts. The nonlinear interpolation method used by PrimeTime produces more accurate results than linear interpolation.

Figure 10-4 Nonlinear Interpolation of Timing Constraint



Scaling of Design Rule Constraints

If scaling has occurred, the `report_constraint` command can interpolate the design rule constraint values between the scaling libraries that were used for calculation on the corresponding pin. Design rule constraint support for composite current source (CCS) scaling now eliminates any possibility of missed design rule constraint violations.

11

Low-Power Flow Support

PrimeTime supports the use the IEEE 1801™ Unified Power Format (UPF) Standard, Version 1.0, for specifying the low-power features of the design. PrimeTime correctly analyzes the timing of the design in the presence of multivoltage supplies and voltage values set on specific supply nets with the `set_voltage` command.

The low-power flow support is described in the following sections:

Multivoltage Analysis

PrimeTime supports timing analysis with different power supply voltages on different cells in the design. PrimeTime calculates delays and slews, PrimeTime SI calculates crosstalk delay and noise effects, and PrimeTime PX calculates power consumption based on the actual supply voltage on each supply pin of each cell.

To perform a multivoltage analysis, you must provide information about the voltage on each power supply pin of each cell in the design. There are three different ways to provide this information:

- Specify the power intent of the design using UPF commands and related PrimeTime commands, including `create_power_domain`, `create_supply_net`, `create_supply_port`, `connect_supply_net`, and `set_voltage`. This method was introduced in the A-2007.12 release and is the preferred method.
- Use commands in the “power domain mode” of the Z-2007.06 release. The commands in this mode are `create_power_net_info`, `create_power_domain`, `connect_power_domain`, `connect_power_net_info`, `set_operating_conditions -object_list`, and `set_voltage`. To use the power domain mode from the Z-2007.06 release, you must set the `power_domains_compatibilty` variable to `true`, which disables the other two methods. For more information, see “Power Domain Mode of Release Z-2007.06” on page 11-20.

  Note:
  In the power domain mode, the `create_power_domain` command has a different effect from the command of the same name executed in the default UPF mode.
- Set the supply voltages on specific blocks in the design by using either the `set_operating_conditions -object_list` or `set_rail_voltage` command or both commands. Using the `set_rail_voltage` command requires a PrimeTime SI license. This method was available prior to the Z-2007.06 release and is still supported. For more information about using this method, see “Multivoltage Method Prior to Release Z-2007.06” on page 11-22.

The three methods are mutually exclusive. By default, if you use any UPF commands, UPF is the voltage specification method for the PrimeTime session. If you do not use any UPF commands in the session, you can use the method available prior to the Z-2007.06 release by using either the `set_operating_conditions -object_list` or `set_rail_voltage` command or both commands.

Most of this chapter applies to the recommend UPF flow. For information that is specific to the two older flows, see the applicable sections at the end of this chapter, “Power Domain Mode of Release Z-2007.06” on page 11-20 and “Multivoltage Method Prior to Release Z-2007.06” on page 11-22.

Unless you are using the older method of specifying all supply voltages using `set_operating_conditions` or `set_voltage`, the library must contain power and ground (PG) pin information for each cell in the design. The Liberty syntax for specifying PG pin information in the library uses the `voltage_map` and `pg_pin` statements. The `voltage_map` statement defines the power supplies and default voltage values, whereas the `pg_pin` statements specify the power supply associated with each pin of each cell, as well as some of the power-related pin parameters. For more information about library requirements for multi-voltage analysis, see the chapter called “Advanced Low Power Modeling” in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*, which is available in the Library Compiler collection on SolvNet.

The second essential requirement for multi-voltage analysis is Liberty library data that accurately describes timing behavior at specific voltages. PrimeTime supports voltage scaling by interpolating between data in separate libraries that have been characterized at different supply voltages. You invoke voltage and temperature scaling by using the `define_scaling_lib_group` command. Note that the behavior of PrimeTime is different from that of Design Compiler, which links the design at precise corner voltages. For more information, see the man page for the `define_scaling_lib_group` command or “Scaling With CCS Timing Libraries” on page 10-8.

UPF Commands

Multiple power supplies at different voltages can supply power to different domains occupying different areas of the chip. Some power domains can be selectively shut off to conserve power during periods of inactivity. Level-shifter cells convert signals leaving one domain and entering another, while isolation cells supply constant signal levels at the outputs of domains that are shut down. Power-down domains can contain retention registers that can retain logic values during the power-down period. Power-switch cells, operating under the control of a power-controller block, switch the power on and off to specific domains. You can specify all of these low-power aspects of a design in the UPF language.

A standard PrimeTime license includes UPF support. No additional license is required specifically for UPF; however, power analysis requires a PrimeTime PX license and IR drop annotation requires a PrimeTime SI license.

For background information about the Synopsys low-power flow and using UPF commands in the flow, see the *Synopsys Low-Power Flow User Guide*, which is available in the PrimeTime Suite collection on SolvNet. The same user guide is also available in other product documentation collections because you can use the UPF syntax with a wide range of products, including Design Compiler, IC Compiler, Formality, MVSIM-VCS, MVRC, and PrimeRail.

PrimeTime uses UPF-specified power intent and the `set_voltage` command to determine the voltage on each power supply pin of each cell. Based on the UPF-specified power intent and `set_voltage` information, PrimeTime builds a virtual model of the power network and propagates the voltage values from UPF supply nets to the PG pins of leaf instances. Since PrimeTime does not directly read power state table, the `set_voltage` command should be consistent with a specific state in UPF power state table (PST) that you intent to verify.

You can enter the UPF commands at the shell prompt. You can also source these commands in a command file or with the `source` or `load_upf` command. Any loaded UPF information is removed upon relinking a design, just like timing assertions loaded from a Synopsys Design Constraints (SDC) file.

The following is a typical sequence of commands used in a PrimeTime timing analysis with UPF-specified power intent. The UPF-related power commands are highlighted in bold.

```
# Read libraries, designs
 ...
read_lib l1.db
read_verilog d1.v
 ...

# Read UPF file
# (containing commands like the ones shown below)

 create_power_domain ...
 create_supply_net ...
 create_supply_port ...
 create_power_switch ...
 connect_supply_net ...
 set_scope block1
 load_upf block1_upf.tcl

# Read SDC and other timing assertions
source d1.tcl

# Define scaling library groups for voltage and temperature scaling
define_scaling_lib_group library_list

# Read SDC and other timing assertions
 ...
set_voltage -object_list supply_net_name
set_voltage -cell ... -pg_pin_name ... value
# (sets voltage on supply nets or IR drop on cell power pins;
#  PrimeTime SI license required for -cell and -pg_pin_name options)
set_temperature -object_list cell_list value

# Perform timing, signal integrity analysis
report_timing
```

PrimeTime reads and uses the UPF information, but it does not modify the power domain description in any structural or functional way. Therefore, it does not write out any UPF commands with the `write_script` command and there is no `save_upf` command.

PrimeTime supports a subset of the commands and command options in the UPF 1.0 standard (available from http://www.accellera.org). The unsupported commands are either unrelated to the functions of PrimeTime (for example, synthesis and physical implementation) or represent capabilities that have not been implemented.

The supported power supply commands can be divided into the following categories:

- Power domain commands:

```
connect_supply_net
create_power_domain
create_power_switch
create_supply_net
create_supply_port
set_domain_supply_net
```

- Isolation and retention commands:

```
set_retention
set_retention_control
set_isolation
set_isolation_control
```

- Flow commands:

```
load_upf
set_scope
set_design_top
upf_version
```

- Related non-UPF commands:

```
check_timing -include supply_net_voltage
check_timing -include unconnected_pg_pins
check_timing -include signal_level
get_power_domains
get_power_switches
get_supply_nets
get_supply_ports
report_power_domain
report_power_network
report_power_pin_info
report_power_switch
report_supply_net
set_level_shfiter_strategy
set_level_shifther_threshold
set_related_supply_net
set_temperature
```

```
set_voltage
```

Some UPF commands are essential for synthesis and other implementation tools, but they supply information that is either not used during PrimeTime analysis or is available from other sources. PrimeTime accepts these commands as valid UPF syntax, but otherwise ignores them. It does not provide man pages for these commands. The following UPF commands are ignored by PrimeTime:

```
add_port_state
add_pst_state
create_pst
map_isolation_cell
map_level_shifter_cell
map_power_switch
map_retention_cell
name_format
set_level_shifter
set_power_switch
```

The following commands are rejected by PrimeTime as unknown syntax. They are either not supported by the Synopsys UPF command subset in all Synopsys tools or they exist only in the RTL UPF.

```
add_domain_elements
bind_checker
create_hdl2upf_vct
create_upf2hdlvct
merge_power_domains
save_upf
set_pin_related_supply
```

The following sections describe the supported UPF commands and their usage in PrimeTime. For background information or more information about using each command throughout the synthesis, implementation, and verification flow, see the UPF 1.0 specification, available as a free download from Accellera at http://www.accellera.org. For more information about the Synopsys low-power flow and Synopsys usage of UPF commands, including usage in PrimeTime, see the *Synopsys Low-Power Flow User Guide*.

Virtual Power Network

In the PrimeTime multivoltage analysis flow, you typically read in the design netlist, source a UPF command script that defines the power supply intent, and then source an SDC script that specifies the timing constraints for analysis. From the design netlist and UPF commands, PrimeTime builds a virtual power network that supplies power to all the PG pins of all the leaf-level cells in the design netlist.

PrimeTime builds the power supply network based on UPF commands, including `create_power_domain`, `create_supply_net`, `create_supply_port`, `set_domain_supply_net`, and `connect_supply_net`. It determines which cells belong to which power domains and traces the supply connections through the supply nets and supply ports down to the leaf level.

The `set_retention` and `set_retention_control` UPF commands determine the supply connections to the retention registers. Similarly, the `set_isolation` and `set_isolation_control` UPF commands determine the connections of the backup supply nets of the isolation cells.

The `set_level_shifter` UPF command is not supported in PrimeTime; therefore, the `connect_supply_net` command must explicitly specify the power supply connections of the PG pins of level shifters. Design Compiler automatically generates these `connect_supply_net` commands when it inserts level shifters and writes them out with the `save_upf` command, so you do not need to create these commands yourself when you read the UPF produced by Design Compiler. To specify the behavior of signal level mismatch checking by `check_timing -include signal_level`, use the `set_level_shifter_strategy` and `set_level_shifter_threshold` non-UPF commands. These commands do not affect PG connectivity.

The `set_related_supply_net` non-UPF command associates a supply net to one or more ports of the design. This is the command syntax:

```
set_related_supply_net -object_list objects supply_net_name
```

You use the `object_lists` option to specify the list of ports associated with supply nets. In the absence of this command, PrimeTime assumes that ports are supplied by the voltage of the design operating condition.

Setting Voltage and Temperature

To determine the supply voltage on each PG pin of each cell in the design, PrimeTime uses the following sources of information, in order of increasing priority:

- default supply voltage in library cell definition (`voltage_map` in Liberty syntax)
- `set_operating_conditions` applied at the design level
- `set_voltage` on a supply net
- `set_voltage` on a PG pin (PrimeTime SI license required)

By default, PrimeTime uses the supply voltages in the library cell definitions, as specified by the `voltage_map` statement in Liberty syntax for the library cell. For details, see the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*, available in the Library Compiler collection of manuals on SolvNet.

You can override the library-specified voltages by using the `set_operating_conditions` command to set the operating conditions at the design level (in other words, without using the `-object_list` option of the command). For example,

```
pt_shell> set_operating_conditions WCIND
```

The supply voltages associated with the named operating condition are specified in the library definition for the operating condition or by the `create_operating_condition` command.

You can override both the library-specified and operating-condition voltage settings with the `set_voltage` command for specific supply nets or specific PG pins in the design.

The command specifies one or more supply nets (`-object_list` *`supply_nets`*) or PG pins (`-cell` *`cell`* `-pg_ping` *`pg_pin`*) in the design and applies a specified voltage to the listed objects. You can specify either a single voltage (*`max_voltage`*) or both the minimum-delay and maximum-delay voltages (`-min` *`min_voltage`* and *`max_votltage`*) used for analysis. Note that the `-min` *`min_voltage`* option specifies the voltage used for minimum-delay analysis, which is typically the larger voltage value. A PrimeTime SI license is required to use the `-cell` and `-pg_pin_name` options.

For example, to set a voltage of 1.20 on the supply net VDD1, you would use the following command:

```
pt_shell> set_voltage 1.20 -object_list VDD1
```

To set a maximum-delay voltage of 0.91 and a minimum-delay voltage of 1.18 on the PWR pin of cell XA91, you would use the following command (PrimeTime SI license required):

```
pt_shell> set_voltage 0.91 -min 1.18 -cell XA91 -pg_pin_name PWR
```

You can similarly specify the operating temperature for specific cells by using the `set_temperature` command, overriding the design operating condition settings. You can use this command to model the effects of on-chip temperature variation. For example, to set the temperature to 130 degrees Celsius on cell XA91:

```
pt_shell> set_temperature 130 -object_list [get_cells XA91]
```

You can use the `report_power_pin_info` command to determine the voltage of a PAD cell. A PAD cell is defined as such in the library by having its `is_pad` attribute set to `true` in the Liberty description of the cell. PrimeTime detects ports connected to PAD cells and determines the port voltage in the following order of increasing priority:

1. The nominal voltage of the main library
2. The nominal voltage of the driving library cell
3. The design operating condition
4. The pad voltage
5. The explicit port voltage, set with the `set_related_supply_net` command in UPF mode or the `set_operating_conditions -object_list port` command in non-UPF mode

For more information about the `report_power_pin_info` command, see "Reporting Commands" on page 11-12 or the man page.

Analysis With Multiple Voltages

You can provide a set of libraries characterized at a range of supply voltage and temperature values, and then scale the cell data between these libraries. This is the preferred way to provide multivoltage library data. The `define_scaling_lib_group` command specifies the list of CCS libraries that have been characterized at different voltages and temperatures, and also invokes delay and constraint scaling between those libraries. PrimeTime interpolates between the specified libraries to determine the cell behavior at the exact voltage and temperature conditions that have been set on each cell. For more information, see the man page for the `define_scaling_lib_group` command or "Scaling With CCS Timing Libraries" on page 10-8.

Another way to provide multivoltage library data is to use the `link_path_per_instance` variable to direct PrimeTime to use different libraries for different cell instances. This variable, when set prior to linking the current design, overrides the default `link_path` setting for selected leaf cell or hierarchical cell instances. The format is a list of lists. Each sublist consists of a pair of elements: a set of instances and a `link_path` specification to be used for those instances. For example,

```
set link_path {* lib1.db}
set link_path_per_instance [list
    [list {ucore} {* lib2.db}]
    [list {ucore/usubblk} {* lib3.db}]]
```

The listed instances can be hierarchical blocks that represent individual voltage domains. The corresponding libraries might be characterized at different supply voltages or under varying conditions such as body bias. If a given block matches multiple entries in the per-instance list, the more specific entry overrides the more general entry. In the 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”).

Using either type of library data method, PrimeTime uses the scaled or library-specified supply voltage information to accurately determine the delays and slews of signals.

For post-layout analysis using detailed annotated parasitics, PrimeTime determines the delays and slews using analog waveforms. For pre-layout analysis (in the absence of detailed annotated parasitics), PrimeTime performs geometric-like scaling of the transition times and net delays, taking into account the respective voltage swings of the driver and load, and the logic thresholds. In either case, PrimeTime reports transition times in terms of local-library thresholds and local voltages.

The use of local trip points and voltages can cause an apparent improvement in transition time along nets in a path. For example, a driver cell might have a transition time of 1.0 ns measured between 20 percent and 80 percent of VDD, while the load on the same net has a transition time of 0.67 ns measured between 30 percent and 70 percent of VDD:

1.0 * (70–30)/(80–20) = .67

To disable prelayout scaling, set the `timing_prelayout_scaling` variable to `false`.

Multivoltage Reporting and Checking

Several commands are available for reporting the power supply network and checking the network for errors or unusual conditions, including `get_*` commands, `report_*` commands, and `check_timing`.

Collection (get_*) Commands

The `get_*` commands each create a collection of objects for reporting purposes. You can create a collection of existing power domains, power, switches, supply nets, or supply ports with the following commands.

```
get_power_domains
get_power_switches
get_supply_nets
get_supply_ports
```

The `get_power_domains` command returns a collection of power domains previously created with the `create_power_domain` command. You can pass the collection to another command for further processing, for example, to extract the name, scope, and elements attributes associated the domains. The `create_power_domain` command options allow you to limit the collection to the power domains meeting specified criteria. For example, the `-of_objects` option causes the collection to include only the power domains to which the

specified objects belong. The *patterns* option limits the collection to power domains whose names match the specified pattern. The `get_power_switches`, `get_supply_nets`, and `get_supply_ports` commands operate in a similar manner and have similar command options.

Table 11-1 lists the attributes of UPF power domain, supply net, supply port, and power switch objects.

Table 11-1 UPF Collection Objects

Command and Object Class	Attribute	Attribute Type	Comment
`get_power_domains` `upf_power_domain`	name	string	name as entered
	full_name	string	hierarchical name
	elements	list of cells	
	scope	cell	
`get_supply_nets` `upf_supply_net`	name	string	name as entered
	full_name	string	hierarchical name
	domains	list of domains	
	resolve	string	one_hot, etc.
	voltage_min	float	minimum-delay voltage
	voltage_max	float	maximum-delay voltage
	static_prob	float	probability of logic 1 (for power analysis)
`get_supply_ports` `upf_supply_port`	name	string	name as entered
	full_name	string	hierarchical name
	domain	domain	
	direction	string	in, out
	scope	string	

Table 11-1 UPF Collection Objects (Continued)

Command and Object Class	Attribute	Attribute Type	Comment
`get_power_switches` `upf_power_switch`	name	string	name as entered
	full_name	string	hierarchical name
	domain	domain	
	output_supply_port_name	string	name as entered
	output_supply_net	supply net	
	input_supply_port_name	string	name as entered
	input_supply_net	supply net	
	control_port_name	list of strings	names as entered
	control_net	list of nets	
	on_state	list of strings	format: state_name input_supply_port boolean_function

Reporting Commands

These are the reporting commands associated with the power supply network:

```
report_power_domain
report_power_network
report_power_pin_info
report_power_switch
report_supply_net
```

The `report_power_domain` command reports the power domains previously defined with `create_power_domain` commands. The reports includes the names of the PG nets of each domain. For example,

```
pt_shell> report_power_domain [get_power_domains B]
...
  Power Domain : B
  Scope : H1
  Elements :  PD1_INST  H2  H3

  Connections :    -- Power --          -- Ground --
    Primary          H1/H7/VDD            H1/H7/VSS
-------------------------------------------------------
```

The `report_power_network` command generates a report on all connectivity of the entire power network, through ports and switches. You can restrict the report to one or more specified nets. For example,

```
pt_shell> report_power_network -nets H1/VDD
 ...

Supply Net:  H1/VDD

  Connections:
     Name       Object type          Domain
    ------------------------------------------------------------
     VDD          Supply_port          D1
     H1/VDD       Supply_port          D2
     U1/VDD       PG_power_pin         D1
     SW1/IN       Switch_input         D1
```

The `report_power_pin_info` command reports the PG connectivity of leaf-level cells or library cells used in the design. For example,

```
pt_shell> report_power_pin_info [get_cells -hierarchical]
...

Note: Power connections marked by (*) are exceptional

Cell          Power Pin   Type                 Voltage        Power Net Connected
              Name                           Max        Min
-------------------------------------------------------------------------------
PD0_INST/I0   PWR         primary_power     0.9300     1.2700   int_VDD_5
PD0_INST/I0   GND         primary_ground    0.0000     0.0000   A_VSS
PD0_INST/I1   PWR         primary_power     0.9300     1.2700   int_VDD_5
PD0_INST/I1   GND         primary_ground    0.0000     0.0000   A_VSS
I0            PWR         primary_power     1.0000     1.0000   int_VDD_4 (*)
I0            GND         primary_ground    0.0000     0.0000   int_VSS_4 (*)
I1            PWR         primary_power     1.1500     1.1500   T_VDD
I1            GND         primary_ground    0.0000     0.0000   T_VSS
PD1_INST/I0   PWR         primary_power     1.0000     1.0000   int_VDD_4
PD1_INST/I0   GND         primary_ground    0.0000     0.0000   int_VSS_4
PD1_INST/I1   PWR         primary_power     1.0000     1.0000   int_VDD_4
PD1_INST/I1   GND         primary_ground    0.0000     0.0000   int_VSS_4

pt_shell> report_power_pin_info [get_lib_cells -of [get_cells -hier]]
...
```

```
Cell          Power Pin Name   Type                Voltage
-------------------------------------------------------------
INVX2         PWR              primary_power       1.0000
INVX2         GND              primary_ground      0.0000
BUFX2         PWR              primary_power       1.0000
BUFX2         GND              primary_ground      0.0000
AND2X1        PWR              primary_power       1.0000
AND2X1        GND              primary_ground      0.0000
OR2X1         PWR              primary_power       1.0000
OR2X1         GND              primary_ground      0.0000
```

The `report_power_switch` command reports the power switches in the design previously created with the `create_power_switch` command. Here is an example of a power switch report:

```
pt_shell> report_power_switch
  ...

  Total of 3 power switches defined for design 'top'.

  Power Switch : sw1
-------------------------------------------------------------
    Power Domain : PD_SODIUM
    Output Supply Port : vout VN3
    Input Supply Port : vin1 VN1
    Control Port: ctrl_small ON1
    Control Port: ctrl_large ON2
    Control Port: ss SUPPLY_SELECT
    On State : full_s1 vin1 { ctrl_small & ctrl_large & ss }
-------------------------------------------------------------
  ...
```

The `report_supply_net` command reports the supply net information for a power domain or specified object in a power domain. For example,

```
pt_shell> report_supply_net
...

  Total of 14 power nets defined.

  Power Net 'VDD_Backup' (power)
  ------------------------------------------------------
  Backup Power Hookups:                                A
  ------------------------------------------------------

  Power Net 'VSS_Backup' (ground)
  ------------------------------------------------------
  Backup Ground Hookups:                               A
  ------------------------------------------------------

  Power Net 'T_VDD' (power,switchable)
  ------------------------------------------------------
  Voltage states:                                  {1.2}
```

```
  Voltage ranges:                                  {1.1 1.3}
  Max-delay voltage:                                    1.15
  Min-delay voltage:                                    1.15
  Primary Power Hookups:                                   T
  -------------------------------------------------------

  Power Net 'A_VDD' (power)
  -------------------------------------------------------
  Max-delay voltage:                                    1.15
  Min-delay voltage:                                    1.15
  Primary Power Hookups:                                   A
  -------------------------------------------------------
```

Using the check_timing Command

You can use the `check_timing` command to check the validity of the power supply connections in the design, including the following types of checks:

- voltage set on each supply net segment (`supply_net_voltage`)
- supply net connected to each PG pin of every cell (`unconnected_pg_pins`)
- compatible signal levels between driver and load pins (`signal_level`)

Voltage Set on Each Supply Net Segment

Every supply net segment must have a voltage set on it with the `set_voltage` command. To verify that this is the case, include a supply net voltage check in the `check_timing` command, as in the following example:

```
pt_shell> check_timing -include supply_net_voltage -verbose
Information: Checking 'no_clock'.
Information: Checking 'no_input_delay'.
 ...
Information: Checking 'supply_net_voltage'.
Warning: There are '2' supply nets without set_voltage. (UPF-029)
  VG
  VS1
Information: Checking 'pulse_clock_non_pulse_clock_merge'.
```

Supply Net Connected to Each PG Pin of Every Cell

Each PG pin of every cell should be connected to a UPF supply net. To verify that this is the case, include an unconnected PG pin check in the `check_timing` command, as in the following example:

```
pt_shell> check_timing -include unconnected_pg_pins
 ...
```

Each UPF supply net connection can be either explicit (as specified by the `connect_supply_net` command) or implicit (due to the assignment of the cell to a power domain, isolation strategy, retention strategy, and so on).

Compatible Driver-to-Load Signal Levels

To have PrimeTime check for mismatching voltage levels between cells that use different supply voltages, use the following `check_timing` command:

```
pt_shell> check_timing -include signal_level
```

This type of timing check traverses all nets and determines whether the output voltage level of each driver is sufficient to drive the load pins on the net. PrimeTime reports any driver-load pairs that fail the voltage level check. It assumes that there is no voltage degradation along the net. You can fix a violation by changing the supply voltages or by inserting a level shifter between the driver and load.

PrimeTime performs signal level checking by comparing the input and output voltage levels defined in the library for the pins of leaf-level cells. The checked signal levels are based on either the gate noise immunity (or signal level range) margins defined by the Liberty syntax `input_voltage` and `output_voltage` or a comparison of driver/load supply voltages. Driver/load supply voltage mismatch reporting can be controlled with the following commands:

```
set_level_shifter_strategy
   [-rule all | low_to_high | high_to_low]

set_level_shifter_threshold
   [-voltage voltage_value]
   [-percent percent_value]
```

The `set_level_shifter_strategy` command specifies the type of strategy used for reporting voltage mismatches: `all`, `low_to_high`, or `high_to_low`. The `low_to_high` strategy reports the voltage level mismatches when a source at a lower voltage drives a sink at a higher voltage. The `high_to_low` strategy reports the voltage level mismatches when a source at a higher voltage drives a sink at a lower voltage. The `all` strategy (the default) reports both types of mismatches.

The `set_level_shifter_threshold` command specifies the absolute and relative mismatch amounts that trigger an error condition. If there is a voltage difference between driver and load, the difference must be less than both the absolute and relative thresholds to be considered acceptable. For example, the following command sets the voltage difference threshold to 0.1 volt and the percentage threshold to 5 percent:

```
pt_shell> set_level_shifter_threshold -voltage 0.1 -percent 5
```

A voltage difference that is more than 0.1 volt or more than five percent of the driver voltage is reported as a mismatch. The default thresholds are both zero, so if you set one threshold to a nonzero value, you should set the other to a nonzero value as well. To disable either absolute or percentage threshold checking, set its threshold to a large value.

The percentage difference is determined as follows:

abs(driver(VDD)–load(VDD))/driver(VDD)*100

PrimeTime reports either of the following conditions as an “incompatible voltage” error:

- Driver VOmax > Load VImax
- Driver VOmin < Load VImin

PrimeTime reports either of the following conditions as a “mismatching driver-load voltage” warning:

- Driver VOH < Load VIH
- Driver VOL > Load VIL

Here is an example of a voltage mismatch report:

```
pt_shell> check_timing -verbose -include signal_level
 ...
Error: There are 2 voltage mismatches
  MIN-MAX - driver vomax > load vimax:

Driver      Voltage   Load          Voltage   Margin
-------------------------------------------------------
u2/Z        2.50      u3/A          0.90      -1.60
u3/Z        2.10      u5/A          1.47      -0.63

Warning: There is 1 voltage mismatch
  MIN-MAX - driver vol > load vil:

Driver      Voltage   Load          Voltage   Margin
-------------------------------------------------------
u2/Z        0.30      u3/A          0.20      -0.10
```

The technology library defines the input and output voltages in terms of the supply voltage in Liberty syntax. PrimeTime correctly calculates the voltages for comparison. For example, a library might define the input and output voltage levels as follows:

- VIL = 0.3 * VDD, VIH = 0.7 * VDD
- VOL = 0.4, VOH = 2.4
- VImin = –0.5, VImax = VDD + 0.5

- VOmin = –0.3, VOmax = VDD + 0.3

PrimeTime calculates the input and output voltages, taking into account the supply voltages that apply to each cell.

For proper transition time scaling for cells with multiple power rails, PrimeTime requires definition of the `input_signal_level` and `output_signal_level` attributes on multi-rail cells. For `check_timing -include signal_level`, PrimeTime uses the definitions of the `input_voltage` and `output_voltage` attributes on all pins.

Even without `input_voltage` or `output_voltage` specified, PrimeTime still attempts to report the 100 worst mismatches based on rail voltages not being exactly equal. For example:

```
pt_shell> check_timing -include signal_level -verbose
 ...
Warning: There are 2 voltage mismatches
  MAX-MAX - driver rail != load rail:
  The 100 worst voltage mismatches:

Driver      Voltage   Load           Voltage   Margin
-----------------------------------------------------
u1/Y        4.75      u3/A           3.00      -1.75
u3/Y        3.00      ff2/CLK        4.75      -1.75

Warning: There are 2 voltage mismatches
  MIN-MIN - driver rail != load rail:
  The 100 worst voltage mismatches:

Driver      Voltage   Load           Voltage   Margin
-----------------------------------------------------
u1/Y        5.25      u3/A         n 3.00      -2.25
u3/Y        3.00      ff2/CLK        5.25      -2.25
...
```

The following table summarizes the effects of the threshold and strategy settings.

Signal level check type	Strategy impact			Threshold impact	Explanation
	All	Low_to_high	High_to_low		
Driver VOmax > Load VImax	None	None	None	None	Driver exceeding breakage voltage of the load cell defined by the library is a fatal condition. It cannot be filtered.
Driver VOmin < Load VImin	None	None	None	None	Library-defined signal level margins define safe region of operation and cannot be suppressed.
Driver VOH < Load VIH	None	None	None	None	
Driver VOL > Load VIL	None	None	None	None	
Driver VDD != Load VDD	None	Filter any mismatch VDDdriver > VDDload	Filter any mismatch VDDdriver > VDDload	Further filter any mismatch smaller than the tolerance	The user-defined tolerance and strategy are fully used.

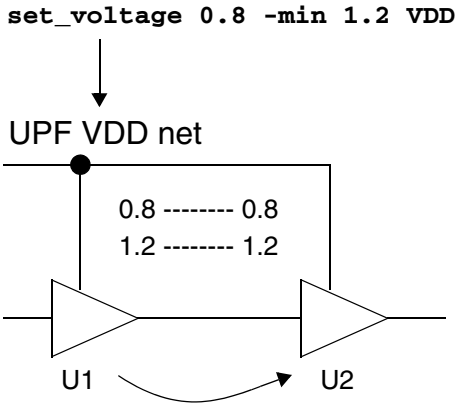
Impact of Correlated Supplies on Signal Level Checking

The signal level check correctly considers the case where minimum and maximum voltages are set on the power nets that supply both of the cells being compared. For example, suppose that you set the voltage for supply VDD as follows:

```
pt_shell> set_voltage 0.8 -min 1.2 VDD   # corner voltages
```

This command sets the supply voltage to 0.8 for maximum-delay analysis and 1.2 Volts for minimum-delay analysis. Suppose that the same VDD supply applies to both the driver and receiver cells of a net, as shown in Figure 11-1.

Figure 11-1 Signal Level Checking With Supply Correlation



PrimeTime considers the fact that the supply voltages of the driver and receiver are correlated. When the supply voltage is low for U1, it is also low for U2, so there is no mismatch. The same is true for the case where the supply voltage is high.

Power Domain Mode of Release Z-2007.06

If you want to use the power domain flow based on setting object-specific voltages and operating conditions, set the `power_domains_compatibility` variable to `true`. This causes PrimeTime to enter the “power domain mode,” which enables the Z-2007.06 multivoltage syntax, disables the UPF syntax, and removes any UPF-specified power supply data in the design. Due to differences in command syntax and data infrastructure, the Z-2007.06 power domain capabilities are not compatible with the UPF features.

When PrimeTime is in the power domain mode, you can view the list of power domain commands with the following command:

```
pt_shell> help "power domains"
Power Domains:
 connect_power_domain   # Connect power domain
 connect_power_net_info # Connect power net to cell power pin
 create_power_domain    # Create power domain object
 create_power_net_info  # Create power net info object
 get_power_domains      # Create a collection of power domains
 report_power_domain    # Report power domain
 report_power_net_info  # Report all power net objects
 report_power_pin_info  # Report power pin info
 set_voltage            # Set voltage
```

There is some overlap in command syntax between the power domain mode and the UPF mode. The interpretation of a command such as `create_power_domain` depends on the operating mode.

Physical tools such as IC Compiler and Design Compiler create power distribution networks based on user-specified connectivity in Tcl syntax. PrimeTime SI can use the same set of commands to build a virtual model of the power distribution network. The power domain model, together with the `set_voltage` command, allows PrimeTime SI to determine the voltage at the power pins of each leaf-level instance. These are the power domain commands:

- The `create_power_net_info` command specifies the name of a power supply net or ground net in the design. If it is a power supply, the command also specifies the voltage and whether the power can be switched off.
- The `create_power_domain` command specifies the name of a power domain and lists the hierarchical cells associated with the domain. If no list is provided, the power domain applies to the top level; there can be no more than one such top-level domain. The command also specifies whether the domain can be powered down and if so, the control net.
- The `connect_power_domain` command creates logical power connections for a specified power domain. It specifies the primary, backup, and internal PG nets for a specified power domain. All cells in the power domain inherit the specified power connections.

  The backup PG nets are for always-on logic, retention registers, isolation cells, and enable-level-shifter cells. The internal PG nets are for the switching cells inside the power domain.
- The `connect_power_net_info` command makes power net connections for a specific power pin of a leaf cell. The pin-level connections override the domain-level connections made with the `connect_power_domain` command.
- The `set_voltage` command defines the operating voltage on the power nets defined by the `create_power_nets_info` command. You can specify a single voltage or a minimum and a maximum voltage for the power net. If you do not use this command, the available operating condition settings are used.

If you link the design again, the power domain information is discarded.

A sequence of commands similar to the following can be used to perform timing analysis of a multivoltage design:

```
# Enable power domains (non-UPF) mode
set power_domains_compatibility TRUE
# Read libraries, designs
...
read_lib l1.db
```

```
read_verilog d1.v
...

# Read libraries, design, SDC, and
# update design and setup link to HSPICE

create_power_net_info –power power_net_name

create_power_domain domain_name –object_list object_list

connect_power_domain –primary_power_net power_net \
  -backup_power_net power_net

connect_power_net_info object_list \
  –power_pin_name pin_name –power_net_name net_name

# Read SDC and other timing assertions
source d1.tcl

# Read SDC and other timing assertions
set_voltage on power nets or IR drop on power pins

# Perform timing, signal integrity analysis
report_timing
```

The following commands report information about the power rails.

- The `report_power_domain` command reports the power domains in the design and their connections. If you specify a list of objects, only the power domains of those objects are reported.
- The `report_power_net_info` command reports the names of the power nets and their associated power domain hookups.
- The `report_power_pin_info` command reports power pin information for library cells or for leaf-level cells in the current design. Specify a list of library cells to find out the names, types, and voltage specifications of the power pins in the library cells. Specify a list of leaf-level cells in the design to find out the power net connections to the power pins of those cells.

Multivoltage Method Prior to Release Z-2007.06

To perform multivoltage analysis, you can specify multiple voltages by applying different operating conditions to different cells with the `set_operating_conditions -object_list` command, by directly annotating voltage drop information with the `set_voltage` command, or by linking blocks to different libraries. (Using the `set_voltage` command

requires a PrimeTime SI license.) Any subsequent timing analysis takes into account the supply voltages specified for the design. The `report_cell` command reports the instance-specific operating conditions and supply voltage information.

These commands were available prior to the Z-2007.06 release and are still supported. However, if you use any UPF commands in a session, the pre-Z-2007.06 multivoltage specification method is disabled, and only UPF commands can be used for specifying power supply information. In that case, you can no longer use `set_operating_conditions -object_list` or `set_voltage`, and any multivoltage information previously set with those commands is discarded.

In the pre-Z-2007.06 multivoltage flow, there are two ways to set supply voltages on a linked design:

- Create different operating conditions having different rail voltages, either by defining them in the technology library or by using the `create_operating_conditions` command. Then apply different operating conditions to different hierarchical blocks or leaf-level cells in the design with the `set_operating_conditions -object_list` command. This method is appropriate for specifying voltage islands on the chip.
- Use the `set_voltage` command to set rail voltages directly on hierarchical blocks or leaf-level cells, overriding any applicable operating condition voltages. This method is appropriate for back-annotating voltage drop information from a power rail analysis tool.

You can combine these two methods in the same design. To do so, first apply the operating conditions to define the voltage islands, then apply the voltage drop information. Rail voltages set with the `set_voltage` command override those set with operating conditions on a cell-by-cell basis. However, they do not override operating conditions set at lower levels of hierarchy.

If you do not set conditions with `set_operating_conditions`, PrimeTime SI uses the default operating conditions of the individual libraries that the cells came from. To have the cells use the operating conditions from the main library instead (the first library defined in the link path), set the `default_oc_per_lib` variable to `false`.

To specify different NLDM library cells for different instances in the design, set the `link_path_per_instance` variable to a list, with each list element consisting of a list of instances and the corresponding link paths that override the default link path for each of those instances. For an example, see “Analysis With Multiple Voltages” on page 11-9.

Setting Operating Conditions on Cells

By default, the `set_operating_conditions` command sets the operating conditions for the whole current design. To override global operating conditions and set the conditions for a hierarchical block or cell instance, use the `-object_list` option and list the applicable cells. For example:

```
pt_shell> set_operating_condtions WCCOM5.0
pt_shell> set_operating_condtions \
          -object_list ALU/A4 WCCOM3.5
```

The first command specifies the set of operating conditions for the whole chip. The second command overrides the whole-chip settings for the cell ALU/A4.

You can also set operating conditions on ports. The specified operating conditions apply to the driving cells for the ports (as specified by the `set_driving_cell` command).

With different operating conditions applied to different levels of the cell hierarchy, the lowest-level setting for a cell has priority over higher-level settings. To get a report on the applicable minimum and maximum operating condition settings for a cell, use the `report_cell` command.

Setting Rail Voltages Directly on Cells

To set the rail voltage on a hierarchical block or leaf-level cell (irrespective of operating conditions), use the `set_rail_voltage` command. Using this command requires a PrimeTime SI license. The command is used in a manner similar to the `set_operating_conditions -object_list` command. For example,

```
pt_shell> set_rail_voltage -max -rail_value 1.2 [get_cells {ALU/a5}]

pt_shell> set_rail_voltage -min -rail_value 1.4 [get_cells {ALU/a5}]
```

12

Object Attributes

An attribute is a string or value associated with an object in the design that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can write programs in Tcl to get attribute information from the design database and generate custom reports on the design.

The following sections describe attributes and how to use them:

Using Attributes

PrimeTime provides a set of commands for setting, reporting, listing, and creating attributes, as summarized in Table 12-1.

Table 12-1 Attribute Commands

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

The `get_attribute`, `report_attribute`, `set_user_attribute`, and `remove_user_attribute` commands accept an object specification, which can be a collection or a list of collections. The object specification for the `get_attribute` command is limited to one collection containing one object.

Defining User Attributes

The `define_user_attribute` command defines a new attribute. PrimeTime has a set of attributes that it considers application attributes. You can view these attributes by using `list_attributes -application`. You must define any other attribute before you can use it in PrimeTime. One such use is importing an attribute from the .ddc or .db file.

You can apply attributes to most object classes in PrimeTime. These can mark interesting cells or nets, store values you have computed, and so on. PrimeTime cannot use these attributes, but you can use them in scripts, procedures, and so on. You can list the attributes you define using the `list_attributes` command. When you define an attribute, decide on the appropriate data type. For more information, see the `define_user_attribute` man page.

Use the following commands to define attributes of various types. You can specify more than one class. Enter

```
pt_shell> define_user_attribute attr_s -class {cell net} -type string
pt_shell> define_user_attribute attr_i -class cell -type int
```

The following command sequence defines attribute `attr_irl` as >= 0 and <= 100, attribute `attr_ir2` as >= 0 with no maximum, and attribute `attr_ir3` as <= 100 with no minimum.

```
pt_shell> define_user_attribute attr_ir1 -class cell -type int \
          -range_min 0 -range_max 100
pt_shell> define_user_attribute attr_ir2 -class cell -type int -range_min
0
pt_shell> define_user_attribute attr_ir3 -class cell -type int -range_max

100
```

The following command defines attribute `attr_oo` for cells. The attribute is a string, but you can set it only to `A`, `B`, `C`, or `D`. Enter

```
pt_shell> define_user_attribute attr_oo \
           -class cell -type string -one_of {A B C D}
```

Importing User-Defined Attributes

To load user-defined attributes (attributes that are not PrimeTime application attributes) from a .ddc or .db file into PrimeTime, you must define the attributes in PrimeTime before you read any designs.

Assume you created a Boolean attribute in Design Compiler called `MarkedPin` and placed this attribute on several pins. In PrimeTime, before you read any designs you must define the `MarkedPin` attribute. For example, enter

```
pt_shell> define_user_attribute -class pin -type Boolean -import

MarkedPin
```

After you define the attribute in this way, PrimeTime extracts it from the .ddc or .db file as it would extract any other attribute. For example, enter

```
pt_shell> list_attributes -class pin

****************************************
Report : List of Attribute Definitions
Design :
Version: 2000.11
Date   : Tues Nov 12 13:25:04 2000
****************************************
```

```
Properties:
    A - Application-defined
    U - User-defined
    I - Importable from db (for user-defined)

Attribute Name     Object       Type          Properties  Constraints
-------------------------------------------------------------------
MarkedPin          pin          Boolean       U,I
```

Attributes appear on design objects only after the design is linked.

User-defined attributes can be imported from both top-level designs and lower-level designs. To import user-defined attributes at the top level, there are no special considerations. You can use the `define_user_attribute -import` command; however, note that a top-level design attribute is attached to a design, but an inherited (lower-level) design attribute is attached to an instance of that design, which is a cell. Therefore, to import and inherit a lower-level design attribute, the attribute must be defined for both designs and cells. The `-import` option is only needed at the design level.

Similarly, a top-level port attribute is attached to a port, but an inherited port attribute is attached to an instance of that port, which is a pin. Therefore, to import and inherit a port attribute, the attribute must be defined for both ports and pins. The `-import` option is only needed at the port level.

To import and inherit a design attribute, the attribute must be defined for both designs and cells, although it only needs to be imported for the design. To import and inherit a port attribute, the attribute must be defined for both ports and pins, although it only needs to be imported for the port. A required attribute is defined automatically if necessary to import a related attribute. After the design is linked, you can see that user-defined attributes have been applied. For example,

```
pt_shell> report_attribute [get_pins */CP]

****************************************
Report : Attribute
Design : M
Version: 2000.11
Date   : Tues Nov 14 13:33:25 2000
****************************************

Design          Object          Type     Attribute Name   Value
-------------------------------------------------------------------
M             o_reg1/CP       Boolean    MarkedPin       true
M             o_reg2/CP       Boolean    MarkedPin       true
M             o_reg3/CP       Boolean    MarkedPin       true
M             o_reg4/CP       Boolean    MarkedPin       true
```

You can create a collection of all pins with this attribute. Enter

```
pt_shell> set ipins [get_pins * -hier -filter \
            "MarkedPin == true"]
```

Saving Design Attributes

By default, PrimeTime saves nothing when it exits. To save design attributes you applied during a session, use the `write_script` command.

The `write_script` command writes the following information:

- Clock-related information: Clock creation, generated clock creation, clock latency, clock uncertainty, and interclock uncertainty
- Timing-related information: Disable timing, maximum time borrow
- Point-to-point exceptions: false path, minimum delay, maximum delay, multicycle path, group path, input delay, output delay, annotated delay, and annotated checks
- Net attributes: Capacitance and resistance
- Port attributes: Fanout, capacitance, and resistance
- Design environment: Wire load model, operating condition, drive, driving cell, and input transition
- Design rules: minimum capacitance, maximum capacitance, minimum transition, maximum transition, minimum fanout, and maximum fanout

You can save the design attributes in Synopsys pt_shell, dc_shell, or dctcl script formats, then use the script to re-create the attributes on the design. For more information, see the `write_script` man page.

Attribute Names and Usage

The PrimeTime `get_attribute` command supports the attributes listed in the tables in this appendix. PrimeTime attributes are read-only (you cannot set them) unless otherwise specified. For information about attributes used in PrimeTime SI, see the *PrimeTime SI User Guide*. Find the attributes by class in these tables:

- `cell` object class, Table 12-2
- `clock` object class, Table 12-3
- `design` object class, Table 12-4
- `lib` object class, Table 12-5

- lib_cell object class, Table 12-6
- lib_pin object class, Table 12-7
- lib_timing_arc object class, Table 12-8
- net object class, Table 12-9
- path_group object class, Table 12-10
- pin object class, Table 12-11
- port object class, Table 12-12
- timing_arc object class, Table 12-13
- timing_path object class, Table 12-14
- timing_point object class, Table 12-15

Cell Object Class Attributes

Table 12-2 lists the cell object class attributes.

Table 12-2 Attributes of the cell Object Class

Attribute name	Type	Description
area	float	The area of a cell. If the cell is hierarchical, this includes net area.
base_name	string	The leaf name of a cell. For example, the base_name of cell U1/U2/U3 is U3.
disable_timing	Boolean	This attribute is true if the timing for the cell has been marked as disabled in set_disable_timing. You can set and unset the disable_timing attribute.

Table 12-2 Attributes of the cell Object Class (Continued)

Attribute name	Type	Description
dont_touch	Boolean	Identifies cells to be excluded from optimization in Design Compiler. Values are undefined by default. Cells with the dont_touch attribute set to true are not modified or replaced during compilation in Design Compiler. Set with the set_dont_touch command and used by characterize_context and create_timing_context. For more information, see the set_dont_touch Design Compiler man page.
early_cell_check_derate_factor early_clk_cell_derate_factor early_clk_net_delta_derate_factor early_clk_net_derate_factor early_data_cell_derate_factor early_data_net_delta_derate_factor early_fall_cell_check_derate_factor early_fall_clk_cell_derate_factor early_fall_clk_net_delta_derate_factor early_fall_clk_net_derate_factor early_fall_data_cell_derate_factor early_fall_data_net_delta_derate_factor early_fall_data_net_derate_factor early_rise_cell_check_derate_factor early_rise_clk_cell_derate_factor early_rise_clk_net_delta_derate_factor early_rise_clk_net_derate_factor early_rise_data_cell_derate_factor early_rise_data_net_delta_derate_factor early_rise_data_net_derate_factor	float	Early timing derate factors, specified using the set_timing_derate command, that apply to the cell. The net derates listed here are only defined for hierarchical cells.
escaped_full_name	string	Contains the name of the cell. Any literal hierarchy characters are escaped with a backslash.
full_name	string	The complete name of a cell. For example, the full name cell U3 within cell U2 within cell U1 is U1/U2/U3. The full_name attribute is not affected by current_instance.

Table 12-2 Attributes of the cell Object Class (Continued)

Attribute name	Type	Description
is_combinational	Boolean	A cell is combinational if it is nonsequential or non-three-state and all of its outputs compute a combinational logic function. The report_lib command reports such a cell as not being a black box.
is_edited	Boolean	This attribute is true if the hierarchical cell has been uniquified as a result of a netlist editing (ECO) change. It is only defined for hierarchical cells.
is_fall_edge_triggered	Boolean	This attribute is true if a cell is used in the design as a falling-edge-triggered flip-flop.
is_hierarchical	Boolean	This attribute is true for any pins of a hierarchical cell. It is false for pins of library cells, also known as leaf cells.
is_ideal	Boolean	This attribute is true if the cell has been marked ideal using the set_ideal_network command.
is_mux	Boolean	This attribute is true if a cell is a multiplexer.
is_negative_level_sensitive	Boolean	This attribute is true if a cell is used in the design as a negative level-sensitive latch.
is_positive_level_sensitive	Boolean	This attribute is true if a cell is used in the design as a positive level-sensitive latch.
is_rise_edge_triggered	Boolean	This attribute is true if a cell is used in the design as a rising-edge-triggered flip-flop.
is_sequential	Boolean	A cell is sequential if it is not combinational.

Table 12-2 Attributes of the cell Object Class (Continued)

Attribute name	Type	Description
is_three_state	Boolean	This attribute is true if a cell is a three-state device.
late_cell_check_derate_factor late_clk_cell_derate_factor late_clk_net_delta_derate_factor late_clk_net_derate_factor late_data_cell_derate_factor late_data_net_delta_derate_factor late_data_net_derate_factor late_fall_cell_check_derate_factor late_fall_clk_cell_derate_factor late_fall_clk_net_delta_derate_factor late_fall_clk_net_derate_factor late_fall_data_cell_derate_factor late_fall_data_net_delta_derate_factor late_fall_data_net_derate_factor late_rise_cell_check_derate_factor late_rise_clk_cell_derate_factor late_rise_clk_net_delta_derate_factor late_rise_clk_net_derate_factor late_rise_data_cell_derate_factor late_rise_data_net_delta_derate_factor late_rise_data_net_derate_factor	float	Late timing derate factors, specified using the set_timing_derate command, that apply to the cell. The net derates listed here are only defined for hierarchical cells.
number_of_pins	integer	Number of pins on the cell. The number of pins can be different before and after linking. For example, if some pins were unconnected in a Verilog instance, after linking to the lower-level design, additional pins can be created on the cell.
object_class	string	The class of the object. This is a constant equal to cell.
ref_name	string	The name of the design or library cell of which the cell is (or will be) an instantiation. Also known as the reference name. The linker looks for a design or library cell by this name to resolve the reference.

Table 12-2 Attributes of the cell Object Class (Continued)

Attribute name	Type	Description
timing_model_type	string	Returns the timing model type of a cell. Valid values: • ITS (interface timing specification) • QTM (quick timing model) • extracted • none (normal library model) You can set the timing_model_type attribute.
upf_isolation_strategy	string	Strategy name (created by the set_isolation command) that was used to derive rail supply and ground nets of the cell.
upf_retention_strategy	string	Strategy name (created by the set_retention command) that was used to derive backup rail supply and ground nets.
voltage_max	float	The voltage value of the maximum or single operating condition for the cell. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the supply voltage value for the operating condition.

Table 12-2 Attributes of the cell Object Class (Continued)

Attribute name	Type	Description
voltage_min	float	The voltage value of the minimum operating condition for the cell. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the supply voltage value for the operating condition.
wire_load_model_max	string	The name of the wire load model effective on a hierarchical cell for the maximum operating condition. You can set the wire_load_model_max attribute.
wire_load_model_min	string	The name of the wire load model effective on a hierarchical cell for the minimum operating condition (valid in bc_wc or on_chip_variation analysis types). You can set the wire_load_model_min attribute.
wire_load_selection_group_max	string	The name of the wire load selection group on a hierarchical cell for the maximum operating condition. You can set the wire_load_selection_group_max attribute.
wire_load_selection_group_min	string	The name of the wire load selection group on a hierarchical cell for the minimum operating condition (valid in bc_wc or on_chip_variation analysis types). You can set the wire_load_selection_group_min attribute.

Clock Object Class Attributes

Table 12-3 lists the clock object class attributes.

Table 12-3 Attributes of the clock Object Class

Attribute name	Type	Description
clock_latency_fall_max	float	The maximum fall latency (insertion delay) for a clock. Set with the set_clock_latency command.
clock_latency_fall_min	float	The minimum fall latency (insertion delay) for a clock. Set with the set_clock_latency command.
clock_latency_rise_max	float	The maximum rise latency (insertion delay) for a clock. Set with the set_clock_latency command.
clock_latency_rise_min	float	The minimum rise latency (insertion delay) for a clock. Set with the set_clock_latency command.
clock_network_pins	collections	The collection of pin and port objects that make up the propagation path of this clock.
clock_source_latency_early_fall_max	float	The maximum early falling source latency. Set with the set_clock_latency command.
clock_source_latency_early_fall_min	float	The minimum early falling source latency. Set with the set_clock_latency command.
clock_source_latency_early_rise_max	float	The maximum early rising source latency. Set with the set_clock_latency command.
clock_source_latency_early_rise_min	float	The minimum early rising source latency. Set with the set_clock_latency command.
clock_source_latency_late_fall_max	float	The maximum late falling source latency. Set with the set_clock_latency command.

Table 12-3 Attributes of the clock Object Class (Continued)

Attribute name	Type	Description
clock_source_latency_late_fall_min	float	The minimum late falling source latency. Set with the set_clock_latency command.
clock_source_latency_late_rise_max	float	The maximum late rising source latency. Set with the set_clock_latency command.
clock_source_latency_late_rise_min	float	The minimum late rising source latency. Set with the set_clock_latency command.
full_name	string	The name of the clock. This is set with the create_clock command. It is either the name given with the -name option, or the name of the first object to which the clock is attached. Once set, this attribute is read-only.
hold_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with the set_clock_uncertainty command.
is_active	Boolean	This attribute is true if the clock is active (the default state). Clocks can be made active or inactive with set_active_clocks.
is_generated	Boolean	This attribute is true if the clock is a generated clock. Set with the create_generated_clock command.
max_capacitance_clock_path_fall	float	A floating-point number that establishes an upper limit for the falling maximum capacitance for all pins in this clock path. Set with the set_max_capacitance command.

Table 12-3 Attributes of the clock Object Class (Continued)

Attribute name	Type	Description
max_capacitance_clock_path_rise	float	A floating-point number that establishes an upper limit for the rising max capacitance for all pins in this clock path. Set with the set_max_capacitance command.
max_capacitance_data_path_fall	float	A floating-point number that establishes an upper limit for the falling max capacitance for all pins in the data path launched by this clock. Set with the set_max_capacitance command.
max_capacitance_data_path_rise	float	A floating-point number that establishes an upper limit for the rising max capacitance for all pins in the data path launched by this clock. Set with the set_max_capacitance command.
max_fall_delay	float	A floating-point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_max_delay command.
max_rise_delay	float	A floating-point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_max_delay command.
max_time_borrow	float	A floating-point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with the set_max_time_borrow command.
max_transition_clock_path_fall	float	A floating-point number that establishes an upper limit for the falling max transition for all pins in this clock path. Set with the set_max_transition command.

Table 12-3 Attributes of the clock Object Class (Continued)

Attribute name	Type	Description
max_transition_clock_path_rise	float	A floating-point number that establishes an upper limit for the rising max transition for all pins in this clock path. Set with the set_max_transition command.
max_transition_data_path_fall	float	A floating-point number that establishes an upper limit for the falling max transition for all pins in the data path launched by this clock. Set with the set_max_transition command.
max_transition_data_path_rise	float	A floating-point number that establishes an upper limit for the rising max transition for all pins in the data path launched by this clock. Set with the set_max_transition command.
min_fall_delay	float	A floating-point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_min_delay.
min_rise_delay	float	A floating-point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_min_delay command.
object_class	string	The class of the object. This is a constant, equal to “clock”. You cannot set the object _class attribute.
period	float	The clock period (or cycle time) is the shortest time during which the clock waveform repeats. For a simple waveform with one rising and one falling edge, the period is the difference between successive rising edges. Set with the create_clock -period command.

Table 12-3 Attributes of the clock Object Class (Continued)

Attribute name	Type	Description
propagated_clock	Boolean	Specifies that clock latency (insertion delay) be determined by propagating delays from the clock source to destination register clock pins. If this attribute is not present, ideal clocking is assumed. Set with the set_propagated_clock command.
setup_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with the set_clock_uncertainty command.
sources	collection	This is a collection of the source pins or ports of the clock. The sources are defined with the create_clock command.
waveform	string	This is a string representation of the clock waveform. For example, a clock rising at 2.5 and falling at 5.0 has a waveform attribute value of {2.5 5}.The waveform is defined with the create_clock command.

Design Object Class Attributes

Table 12-4 lists the design object class attributes.

Table 12-4 Attributes of the design Object Class

Attribute name	Type	Description
analysis_type	string	The analysis type: single, bc_wc, or on_chip_variation.
area	float	The total area of the design. This is the sum of the areas of all leaf cells and nets.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
designWare	Boolean	This attribute specifies if a cell came from a DesignWare design.
dont_touch	Boolean	Identifies designs to be excluded from optimization in Design Compiler. Values are undefined by default. Designs with the dont_touch attribute set to true are not modified or replaced during compile in Design Compiler. Set with the set_dont_touch command and used by the characterize_context and create_timing_context commands.
early_cell_check_derate_factor early_clk_cell_derate_factor early_clk_net_delta_derate_factor early_clk_net_derate_factor early_data_cell_derate_factor early_data_net_delta_derate_factor early_data_net_derate_factor early_fall_cell_check_derate_factor early_fall_clk_cell_derate_factor early_fall_clk_net_delta_derate_factor early_fall_clk_net_derate_factor early_fall_data_cell_derate_factor early_fall_data_net_delta_derate_factor early_fall_data_net_derate_factor early_rise_cell_check_derate_factor early_rise_clk_cell_derate_factor early_rise_clk_net_delta_derate_factor early_rise_clk_net_derate_factor early_rise_data_cell_derate_factor early_rise_data_net_delta_derate_factor early_rise_data_net_derate_factor	float	Early timing derate factors, specified using the set_timing_derate command, that apply to the design.
extended_name	string	The complete, unambiguous name of a design. The extended_name of the design is the source_file_name attribute followed by a colon (:) followed by the full_name attribute. For example, the extended_name of design TOP read in from /u/user/simple.db is /u/user/simple.db:TOP.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
full_name	string	The name of a design. For example, the full_name of design TOP read in from /u/user/simple.db is TOP. This name can be ambiguous because several designs of the same name can be read in from different files.
is_current	Boolean	This attribute is true for the current design. This attribute changes for all designs when you use the current_design command.
is_edited	Boolean	This attributed is true if the design has been uniquified as a result of a netlist editing (ECO) change.
late_cell_check_derate_factor late_clk_cell_derate_factor late_clk_net_delta_derate_factor late_clk_net_derate_factor late_data_cell_derate_factor late_data_net_delta_derate_factor late_data_net_derate_factor late_fall_cell_check_derate_factor late_fall_clk_cell_derate_factor late_fall_clk_net_delta_derate_factor late_fall_clk_net_derate_factor late_fall_data_cell_derate_factor late_fall_data_net_delta_derate_factor late_fall_data_net_derate_factor late_rise_cell_check_derate_factor late_rise_clk_cell_derate_factor late_rise_clk_net_delta_derate_factor late_rise_clk_net_derate_factor late_rise_data_cell_derate_factor late_rise_data_net_delta_derate_factor late_rise_data_net_derate_factor	float	Late timing derate factors, specified using the set_timing_derate command, that apply to the design.
max_area	float	A floating-point number that represents the target area of the design. The units must be consistent with the units used from the technology library during optimization. The max_area value is set in Design Compiler using the set_max_area command.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
max_capacitance	float	A floating-point number that sets the default maximum capacitance design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with the set_max_capacitance command.
max_fanout	float	A floating-point number that specifies the default maximum fanout design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with the set_max_fanout command.
max_transition	float	A floating-point number that specifies the default maximum transition design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with the set_max_transition command.
min_capacitance	float	A floating-point number that sets the default minimum capacitance design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with the set_min_capacitance command.
min_fanout	float	A floating-point number that specifies the default minimum fanout design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with the set_min_fanout command.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
min_transition	float	A floating-point number that specifies the default minimum transition design rule limit for the design. The units must be consistent with those of the technology library used during optimization. Set with the set_min_transition command.
object_class	string	The class of the object. This is a constant equal to design.
operating_condition_max	string	The name of the maximum or single operating condition for the design. Set with the set_operating_conditions command.
operating_condition_min	string	The name of the minimum operating condition for the design. This attribute is not valid in single operating condition analysis.
process_max	float	The process value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command.
process_min	float	The process value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. Operating conditions are associated with a design using the set_operating_conditions command.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
rc_input_threshold_pct_rise rc_input_threshold_pct_fall rc_output_threshold_pct_rise rc_output_threshold_pct_fall rc_slew_derate_from_library rc_slew_lower_threshold_pct_fall rc_slew_lower_threshold_pct_rise rc_slew_upper_threshold_pct_fall rc_slew_upper_threshold_pct_rise	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimeTime uses to calculate delays and transition times. These attributes from a design object return the values obtained from the main library (the first library in the link path). For more information, see "Characterization Trip Points" on page 9-6.
source_file_name	string	The name of the file from which the design was read. For example, the source_file_name of design TOP read in from /u/user/simple.db is /u/user/simple.db.
temperature_max	float	The temperature value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the ambient temperature value for the operating condition.
temperature_min	float	The temperature value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the ambient temperature value for the operating condition.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
tree_type_max	string	The tree_type value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. The tree_type value is used in prelayout interconnect delay estimation, and can have a value of best_case, balanced_case, balanced_resistance (cmos2 only), or worst_case.
tree_type_min	string	The tree_type value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. The tree_type value is used in prelayout interconnect delay estimation, and can have a value of best_case, balanced_case, balanced_resistance (cmos2 only), or worst_case.
voltage_max	float	The voltage value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the supply voltage value for the operating condition.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
`voltage_min`	`float`	The voltage value of the minimum operating condition for the design. The operating condition is defined in a library or by the `create_operating_conditions` command. You associate operating conditions with a design using the `set_operating_conditions` command. This attribute represents the supply voltage value for the operating condition
`wire_load_min_block_size`	`float`	Specifies the smallest hierarchical cell that has automatic wire load selection by area applied. If an automatic wire load selection group is specified as the default in the main library, or through the `set_wire_load_selection_group` command, it is applied to all hierarchical cells larger than the specified minimum block size.

Table 12-4 Attributes of the design Object Class (Continued)

Attribute name	Type	Description
wire_load_mode	string	Determines which wire load model to use to compute wire capacitance, resistance, and area for nets in a hierarchical design that has different wire load models at different hierarchical levels. Allowed values are top, which tells PrimeTime to use the wire load model at the top hierarchical level; enclosed, which tells PrimeTime to use the wire load model on the smallest design that encloses a net completely; and segmented, which indicates to break the net into segments, one within each hierarchical level. In the segmented mode, each net segment is estimated using the wire load model on the design that encloses that segment. The segmented mode is not supported for wire load models on clusters. If no value is specified for this attribute, PrimeTime searches for a default in the first library in the link path. If no default is found, top is the default. Set with set_wire_load_mode.
wire_load_model_max	string	The name of the design's wire load model for maximum conditions. Set with set_wire_load_model.
wire_load_model_min	string	The name of the design's wire load model for minimum conditions. This attribute is not valid for single operating condition analysis. Set with set_wire_load_model.
wire_load_selection_group_max	string	The name of the design's wire load selection group for maximum conditions. Set with set_wire_load_selection_group.
wire_load_selection_group_min	string	The name of the design's wire load selection group for minimum conditions. Set with set_wire_load_selection_group.

Library Object Class Attributes

Table 12-5 lists the library object class attributes.

Table 12-5 Attributes of the lib Object Class

Attribute name	Type	Description
default_max_capacitance	float	The library default maximum capacitance design rule limit.
default_max_fanout	float	The library default maximum fanout design rule limit.
default_max_transition	float	The library default maximum transition design rule limit.
default_min_capacitance	float	The library default minimum capacitance design rule limit.
default_min_fanout	float	The library default minimum fanout design rule limit.
default_min_transition	float	The library default minimum transition design rule limit.
extended_name	string	The complete, unambiguous name of a library. The extended_name of the library is the source_file_name attribute followed by a colon (:) followed by the full_name attribute. For example, the extended_name of library tech1 read in from /u/user/lib1.db is /u/user/lib1.db:tech1.
full_name	string	The name of a library. For example, the full_name of library tech1 read in from /u/user/lib1.db is tech1. This name can be ambiguous because several libraries of the same name can be read in from different files.
lib_scaling_group	collection	The collection of libraries in the scaling library group to which the library belongs, which is set with the define_scaling_lib_group command.
object_class	string	The class of the object. This is a constant equal to lib.

Table 12-5 Attributes of the lib Object Class (Continued)

Attribute name	Type	Description
source_file_name	string	The name of the file from which the library was read. For example, the source_file_name of library tech1 read in from /u/user/lib1.db is /u/user/lib1.db.

Library Cell Object Class Attributes

Table 12-6 lists the library cell object class attributes.

Table 12-6 Attributes of the lib_cell Object Class

Attribute name	Type	Description
area	float	A floating-point value representing the area of a library cell.
base_name	string	The name of a library cell. For example, the base_name of library cell tech1/AN2 is AN2.
disable_timing	Boolean	This attribute is true if the timing for the library cell has been specified to be disabled using the set_disable_timing command.
dont_touch	Boolean	Identifies library cells to be excluded from optimization. Values are undefined by default. Library cells with the dont_touch attribute set to true are not modified or replaced during compile. Set with the set_dont_touch command.
full_name	string	The fully qualified name of a library cell. This is the name of the library followed by the library cell name. For example, the full_name of library cell AN2 in library tech1 is tech1/AN2.
function_id	string	The name of the function that is created by Library Compiler.
is_combinational	Boolean	This attribute is true if the library cell is not sequential.

Table 12-6 Attributes of the lib_cell Object Class (Continued)

Attribute name	Type	Description
`is_fall_edge_triggered`	`Boolean`	This attribute is true if library cell is used in the design as a falling-edge-triggered flip-flop.
`is_mux`	`Boolean`	This attribute is true if a library cell is a multiplexer.
`is_negative_level_sensitive`	`Boolean`	This attribute is true if the library cell is used in the design as a negative level-sensitive latch.
`is_pll_cell`	`Boolean`	This attribute is `true` if the library cell is used in the design as a phase locked loop (PLL) cell.
`is_positive_level_sensitive`	`Boolean`	This attribute is true if the library cell is used in the design as a positive level-sensitive latch.
`is_rise_edge_triggered`	`Boolean`	This attribute is true if the library cell is used in the design as a rising-edge-triggered flip-flop.
`is_sequential`	`Boolean`	This attribute is true if the library cell is sequential.
`is_three_state`	`Boolean`	This attribute is true if a library cell is a three-state device.
`number_of_pins`	`integer`	Number of pins on the library cell.
`object_class`	`string`	The class of the object. This is a constant, equal to `lib_cell`.
`timing_model_type`	`string`	Returns the timing model type of a library cell. The values are • ITS (interface timing specification) • QTM (quick timing model) • extracted • none (normal library model)

Table 12-6 Attributes of the lib_cell Object Class (Continued)

Attribute name	Type	Description
user_function_class	string	This attribute is used to size complex cells that would otherwise not be sizable due to limitations of library tools in generating the function_id attribute for those complex cells.

Library Pin Object Class Attributes

Table 12-7 lists the library pin object class attributes.

Table 12-7 Attributes of the lib_pin Object Class

Attribute name	Type	Description
base_name	string	The leaf name of the library cell pin. For example, the base_name of tech1/AN2/Z is Z.
clock	Boolean	A Boolean value that is set to true when a clock attribute is attached to a lib_pin in the library definition.
direction	string	The direction of a pin. Value can be in, out, inout, or internal.
disable_timing	Boolean	This attribute is true if the library pin has been specified to be disabled using the set_disable_timing command.
drive_resistance_fall	float	A floating-point value representing the linear drive resistance for falling delays of a library pin.
drive_resistance_rise	float	A floating-point value representing the linear drive resistance for rising delays of a library pin.
driver_waveform_fall driver_waveform_rise	string	This attribute indicates the type of driver waveform for the library pin, either ramp or standard.

Table 12-7 Attributes of the lib_pin Object Class (Continued)

Attribute name	Type	Description
fanout_load	float	A floating-point value representing the fanout load value of a library pin. This value is used in computing max_fanout design rule cost.
full_name	string	The fully qualified name of a library cell pin. This is the name of the library followed by the library cell name followed by a pin name. For example, the full_name of pin Z on library cell AN2 in library tech1 is tech1/AN2/Z.
has_ccs_noise_above_high has_ccs_noise_above_low has_ccs_noise_below_high has_ccs_noise_below_low	Boolean	This attribute indicates whether CCS noise information is present for a pin in a library.
has_ccs_receiver_fall	Boolean	This attribute indicates whether CCS information is present for a pin in a library, for receiver fall analysis.
has_ccs_receiver_rise	Boolean	This attribute indicates whether CCS information is present for a pin in a library, for receiver rise analysis.
is_async_pin	Boolean	This attribute is true if a library pin is an asynchronous preset/clear pin.
is_clear_pin	Boolean	This attribute is true if a library pin is an asynchronous clear pin.
is_clock_pin	Boolean	This attribute is true if a library pin is a clock pin of sequential cell.
is_data_pin	Boolean	This attribute is true if a library pin is a data pin of a sequential cell.
is_fall_edge_triggered_clock_pin	Boolean	This attribute is true if the library pin is used in the design as a falling-edge-triggered flip-flop clock pin.
is_fall_edge_triggered_data_pin	Boolean	This attribute is true if the library pin is used in the design as a falling-edge-triggered flip-flop data pin.

Table 12-7 Attributes of the lib_pin Object Class (Continued)

Attribute name	Type	Description
is_mux_select_pin	Boolean	This attribute is true if a library pin is a select pin of a multiplexer device.
is_negative_level_sensitive_clock_pin	Boolean	This attribute is true if the library pin is used in the design as a negative edge-triggered flip-flop clock pin.
is_negative_level_sensitive_data_pin	Boolean	This attribute is true if the library pin is used in the design as a negative edge-triggered flip-flop data pin.
is_pll_feedback_pin	Boolean	This attribute is `true` if the library pin is a feedback pin of a phase locked loop (PLL) cell.
is_pll_output_pin	Boolean	This attribute is `true` if the library pin is an output pin of a phase locked loop (PLL) cell.
is_pll_reference_pin	Boolean	This attribute is `true` if the library pin is a reference pin of a phase locked loop (PLL) cell.
is_positive_level_sensitive_clock_pin	Boolean	This attribute is true if the library pin is used in the design as a positive edge-triggered flip-flop clock pin.
is_positive_level_sensitive_data_pin	Boolean	This attribute is true if the library pin is used in the design as a positive edge-triggered flip-flop data pin.
is_preset_pin	Boolean	This attribute is true if a library pin is an asynchronous preset pin.
is_rise_edge_triggered_clock_pin	Boolean	This attribute is true if the library pin is used in the design as a rising-edge-triggered flip-flop clock pin.
is_rise_edge_triggered_data_pin	Boolean	This attribute is true if the library pin is used in the design as a rising-edge-triggered flip-flop data pin.
is_three_state	Boolean	This attribute is true if the library pin is a three-state driver.

Table 12-7 Attributes of the lib_pin Object Class (Continued)

Attribute name	Type	Description
is_three_state_enable_pin	Boolean	This attribute is true if a library pin is an enable pin of a three-state device.
is_three_state_output_pin	Boolean	This attribute is true if a library pin could output a three-state signal.
is_unbuffered	Boolean	
load_of_pin_capacitance	float	
max_capacitance	float	A floating-point value representing the maximum capacitance design rule limit for a library pin.
max_fanout	float	A floating-point value representing the maximum fanout design rule limit for a library pin.
max_transition	float	A floating-point value representing the maximum transition time design rule limit for a library pin.
min_capacitance	float	A floating-point value representing the minimum capacitance design rule limit for a library pin.
min_fanout	float	A floating-point value representing the minimum fanout design rule limit for a library pin.
min_transition	float	A floating-point value representing the minimum transition time design rule limit for a library pin.
object_class	string	The class of the object. This is a constant, equal to lib_pin.
pin_capacitance	float	A floating-point value representing the capacitance of a library pin.

Table 12-7 Attributes of the lib_pin Object Class (Continued)

Attribute name	Type	Description
rc_input_threshold_pct_rise rc_input_threshold_pct_fall rc_output_threshold_pct_rise rc_output_threshold_pct_fall rc_slew_derate_from_library rc_slew_lower_threshold_pct_fall rc_slew_lower_threshold_pct_rise rc_slew_upper_threshold_pct_fall rc_slew_upper_threshold_pct_rise	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimeTime uses to calculate delays and transition times. These attributes from a library pin return the values obtained from the library to which the pin belongs. For more information, see "Characterization Trip Points" on page 9-6.
si_has_immunity_above_high si_has_immunity_above_low si_has_immunity_below_high si_has_immunity_below_low	Boolean	This attribute indicates whether NLDM noise immunity information is present for a pin in a library.

Library Timing Arc Object Class Attributes

Table 12-8 lists the library timing arc object class attributes.

Table 12-8 Attributes of the lib_timing_arc Object Class

Attribute name	Type	Description
from_lib_pin	collection	A collection containing the from library pin of the library timing arc.
has_ccs_driver_fall	Boolean	This attribute indicates whether the library timing arc has CCS Timing driver information, either true or false, for fall analysis.
has_ccs_driver_rise	Boolean	This attribute indicates whether the library timing arc has CCS Timing driver information, either true or false, for rise analysis.
has_ccs_noise_above_low has_ccs_noise_below_high has_ccs_noise_below_low has_ccs_noise_above_high	Boolean	These attributes indicate whether CCS Noise information is present in the timing arc object in a library.
has_ccs_receiver_fall	Boolean	This attribute indicates whether CCS receiver information is present in the timing arc object in a library, for fall analysis.

Table 12-8 Attributes of the lib_timing_arc Object Class (Continued)

Attribute name	Type	Description
has_ccs_receiver_rise	Boolean	This attribute indicates whether CCS receiver information is present in the timing arc object in a library, for rise analysis.
is_disabled	Boolean	Returns true if the library timing arc is disabled.
is_user_disabled	Boolean	Returns true if the library timing arc is disabled by using the command set_disable_timing.
mode	string	Returns the mode string of the library timing arc.
object_class	string	The class of the object. This is a constant equal to library_arc.
sdf_cond	string	A string representing the SDF condition of the library timing arc.
sense	string	A string representing the sense of the library timing arc.
si_has_immunity_above_low si_has_immunity_below_high si_has_immunity_below_low si_has_immunity_above_high	Boolean	These attributes indicate whether noise immunity information is present in the timing arc object in a library.
si_has_iv_above_low si_has_iv_below_high si_has_iv_below_low si_has_iv_above_high	Boolean	These attributes indicate whether the library timing arc contains output steady-state information in the form of I/V relationships (polynomials or tables).
si_has_propagation_above_low si_has_propagation_below_high si_has_propagation_below_low si_has_propagation_above_high	Boolean	These attributes indicate whether the library timing arc contains information on how bumps present at the arc input are propagated across the arc to the output.
si_has_resistance_above_low si_has_resistance_below_high si_has_resistance_below_low si_has_resistance_above_high	Boolean	These attributes indicate whether the library timing arc contains output steady-state information in the form of simple steady drive resistance.

Table 12-8 Attributes of the lib_timing_arc Object Class (Continued)

Attribute name	Type	Description
to_lib_pin	collection	A collection containing the to library pin of the library timing arc.
when	string	A string representing the when string of the library timing arc.

Net Object Class Attributes

Table 12-9 lists the net object class attributes.

Table 12-9 Attributes of the net Object Class

Attribute name	Type	Description
aggressors	string	This attribute shows the aggressor nets that impact the victim net.
area	float	The estimated area of a net. The net area is calculated using a wire load model.
ba_capacitance_max	float	A floating-point value representing the back-annotated capacitance on a net for maximum conditions. Set with set_load or read_parasitics.
ba_capacitance_min	float	A floating-point value representing the back-annotated capacitance on a net for minimum conditions. Set with set_load or read_parasitics.
ba_resistance_max	float	A floating-point value representing the back-annotated resistance on a net for maximum conditions. Set with set_resistance or read_parasitics.

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description
ba_resistance_min	float	A floating-point value representing the back-annotated resistance of a net for minimum conditions. Set with set_resistance or read_parasitics.
base_name	string	The leaf name of a net. For example, the base name of net i1/i1z1 is i1z1. You cannot set this attribute.
coupling_capacitors	string	This attribute lists the cross-coupling capacitance values in pF. With this attribute the nets are explicitly identified. For example, {u1a/A (n1) u2b/Z (n2) 0.40 not_filtered} {n1:3 (n1) n2:5 (n2) 0.03 filtered_by_accum_noise_peak} {in_port (n1) n3:7 (n3) 0.01 filtered_by_accum_noise_peak}
dont_touch	Boolean	Identifies nets to be excluded from optimization in Design Compiler. Values are undefined by default. Nets with the dont_touch attribute set to true are not modified or replaced during compile with Design Compiler. Set with the set_dont_touch command.

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description
early_clk_net_derate_factor early_data_net_derate_factor early_fall_clk_net_delta_derate_factor early_fall_clk_net_derate_factor early_fall_data_net_delta_derate_factor early_fall_data_net_derate_factor early_rise_clk_net_delta_derate_factor early_rise_clk_net_derate_factor early_rise_data_net_delta_derate_factor early_rise_data_net_derate_factor	float	Early timing derate factors, specified using the set_timing_derate command, that apply to the net.
effective_aggressors	string	This attribute lists the effective aggressors for the net. Effective aggressors are aggressors that are analyzed. For example, get_attribute -class net n5 \ effective_aggressors n7 (For more information, see the si_xtalk_bumps attribute.)
effective_coupling_ capacitors	string	This attribute lists the effective cross-coupling capacitance values in pF. Only the capacitors that are not excluded are shown.
escaped_full_name	string	Contains the name of the cell. Any literal hierarchy characters are escaped with a backslash.
full_name	string	The complete name of a net. For example, the full_name of net i1z1 within cell i1 is i1/i1z1. The full_name attribute is not affected by current instance. The full_name attribute is read-only.
has_detailed_parasitics	Boolean	This attribute is true if any part of the net has annotated detailed parasitics (even if only one segment of a net at different levels of hierarchy).

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description
has_valid_parasitics	Boolean	This attribute is true if the net has an annotated pi model, or if all segments of the net are annotated with properly connected detailed parasitics that form a valid representation of physical interconnection between drivers and loads.
is_ideal	Boolean	This attribute is `true` if the net has been marked ideal using the `set_ideal_network` command.
late_clk_net_derate_factor late_data_net_derate_factor late_fall_clk_net_delta_derate_factor late_fall_clk_net_derate_factor late_fall_data_net_delta_derate_factor late_fall_data_net_derate_factor late_rise_clk_net_delta_derate_factor late_rise_clk_net_derate_factor late_rise_data_net_delta_derate_factor late_rise_data_net_derate_factor	float	Late timing derate factors, specified using the `set_timing_derate` command, that apply to the net.
net_resistance_max	float	A floating-point value representing the resistance of a net for maximum conditions. May be computed from wire load models or set using `set_resistance` or `read_parasitics`.
net_resistance_min	float	A floating-point value representing the resistance of a net for minimum conditions. May be computed from wire load models or set using `set_resistance` or `read_parasitics`.

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description

number_of_aggressors	integer	This attribute shows the number of aggressor nets to a victim net. For example,
		`get_attribute -class net n5 \`
		`number_of_aggressors 1`
		Note that any coupled net is an aggressor net.

number_of_coupling_capacitors	integer	This attribute shows the number of coupling capacitors.

number_of_effective_aggressors	integer	This attribute shows the number of effective aggressor nets to a victim net. Only the effective aggressors are used for analysis. For example,
		`get_attribute -class net n5 \`
		`number_of_effective_aggressors 1`

number_of_effective_coupling_capacitors	integer	This attribute lists the number of effective coupling capacitors on a net. Only the effective coupling capacitors are used for the analysis.
object_class	string	The class of the object. This is a constant, equal to net.
pin_capacitance_max	float	A floating-point value representing the sum of all pin capacitances of a net for maximum conditions. You cannot set this attribute.
pin_capacitance_min	float	A floating-point value representing the sum of all pin capacitances of a net for minimum conditions. You cannot set this attribute.

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description
rc_annotated_segment	Boolean	This attribute is true if the specific net segment has annotated parasitics. For two segments at different levels of hierarchy (for example, n1 and h1/n1), the attribute values can differ.
rc_network	string	A string describing the parasitic data that has been back-annotated on the net, including resistor values (in KOhms) and capacitor values (in pF).
si_is_selected	Boolean	This attribute exists on each coupled net. It indicates whether the net was reselected at least once (true) or was never reselected (false) for crosstalk analysis in the most recent timing update. Reselection can occur only in the second and subsequent iterations of crosstalk analysis.
si_xtalk_bumps	string	This attribute lists each aggressor net and the voltage bumps that rising and falling aggressor transitions induce on the victim net (worst of rising min or max bumps and worst of falling min or max bumps, each expressed as a decimal fraction of the rail-to-rail voltage), or gives the reason that an aggressor net has no effect on the victim net.
si_xtalk_bumps_max_fall si_xtalk_bumps_max_rise si_xtalk_bumps_min_fall si_xtalk_bumps_min_rise	string	Each of these attributes lists each aggressor net and the voltage it induces on the victim net (expressed as a decimal fraction of the rail-to-rail voltage) for a given delay change type and transition type: maximum fall, maximum rise, minimum fall, or minimum rise.

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description
si_xtalk_composite_aggr_min_rise si_xtalk_composite_aggr_min_fall si_xtalk_composite_aggr_max_rise si_xtalk_composite_aggr_max_fall	collection	Each of these attributes list a collection of aggressors in a potential composite aggressor group for a given delay change type and transition type: maximum fall, maximum rise, minimum fall, or minimum rise.
si_xtalk_used_ccs_min_rise si_xtalk_used_ccs_min_fall si_xtalk_used_ccs_max_rise si_xtalk_used_ccs_max_fall	Boolean	Each attribute set to true means that the net was analyzed for crosstalk delay using CCS timing models for that type of timing constraint and transition.
total_capacitance_max	float	A floating-point value representing the sum of all pin capacitances and the wire capacitance of a net for maximum conditions. You cannot set this attribute.
total_capacitance_min	float	A floating-point value representing the sum of all pin capacitances and the wire capacitance of a net for minimum conditions. You cannot set this attribute.
total_coupling_capacitance	float	This attribute lists the total cross-coupling capacitance a victim net has in pF.
total_effective_coupling_capacitance	float	This attribute lists the total effective cross capacitance a victim net has in pF. Only effective values are used during the analysis.
user_global_coupling_separated	Boolean	If true, this net has been globally separated with the set_coupling_separation command.

Table 12-9 Attributes of the net Object Class (Continued)

Attribute name	Type	Description
user_pairwise_coupling_separated	collection	The collection of nets which have been pairwise-separated with the set_coupling_separation command.
wire_capacitance_max	float	A floating-point value representing the wire capacitance of a net for maximum conditions. The value can be computed from wire load models or set using set_load or read_parasitics.
wire_capacitance_min	float	A floating-point value representing the wire capacitance of a net for minimum conditions. The value can be computed from wire load models or set using set_load or read_parasitics.

Path Group Object Class Attributes

Table 12-10 lists the path group object class attributes.

Table 12-10 Attributes of the path_group Object Class

Attribute name	Type	Description
full_name	string	The name of the path group. Path groups are created by group_path or implicitly using create_clock. You cannot set this attribute.
object_class	string	The class of the object. This is a constant, equal to path_group. You cannot set this attribute.

Table 12-10 Attributes of the path_group Object Class (Continued)

Attribute name	Type	Description
`weight`	`float`	A floating-point value representing the cost function weight assigned to this path group. The weight of a group specifies how much the group influences the total maximum delay cost and can be used to guide optimization. You can specify the weight with `group_path`.

Pin Object Class Attributes

Table 12-11 lists the pin object class attributes.

Table 12-11 Attributes of the pin Object Class

Attribute name	Type	Description
`actual_fall_transition_max`	`float`	A floating-point value representing the largest falling transition time for a pin.
`actual_fall_transition_min`	`float`	A floating-point value representing the smallest falling transition time for a pin.
`actual_rise_transition_max`	`float`	A floating-point value representing the largest rising transition time for a pin.
`actual_rise_transition_min`	`float`	A floating-point value representing the smallest rising transition time for a pin.
`annotated_fall_transition_delta_max`	`float`	This attribute shows the additional transition time added to the maximum falling transition time on a pin. The additional transition time is set by either the `set_annotated_transition -delta_only` command or by PrimeTime SI during the crosstalk analysis.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
annotated_fall_transition_delta_min	float	This attribute shows the additional transition time added to the minimum falling transition time on a pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_rise_transition_delta_max	float	This attribute shows the additional transition time added to the maximum rising transition time on a pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_rise_transition_delta_min	float	This attribute shows the additional transition time added to the minimum rising transition time on a pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.
arrival_window	string	The minimum and maximum arrivals for rise and fall transitions. To get the arrival_window attribute on pins that are not endpoints, set the timing_save_pin_arrival_and_slack variable to true.
cached_c1_max_fall	float	A floating-point value representing the C1 CCS receiver model for a maximum fall transition.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
cached_c1_max_rise	float	A floating-point value representing the C1 CCS receiver model for a maximum rise transition.
cached_c1_min_fall	float	A floating-point value representing the C1 CCS receiver model for a minimum fall transition.
cached_c1_min_rise	float	A floating-point value representing the C1 CCS receiver model for a minimum rise transition.
cached_c2_max_fall	float	A floating-point value representing the C2 CCS receiver model for a maximum fall transition.
cached_c2_max_rise	float	A floating-point value representing the C2 CCS receiver model for a maximum rise transition.
cached_c2_min_fall	float	A floating-point value representing the C2 CCS receiver model for a minimum fall transition.
cached_c2_min_rise	float	A floating-point value representing the C2 CCS receiver model for a minimum rise transition.
case_value	string	The user-specified logic value of a pin or port propagated from a case analysis or logic constant in the design.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
ceff_params_max	string	Returns an access to the parameters internally computed by PrimeTime to compute the effective capacitance of a driver pin of a cell, respectively for maximum operating conditions. The returned parameters are rd, t0, delta_t, and Ceff; they represent how a driver is modeled for computing the effective capacitance.
ceff_params_min	string	Returns an access to the parameters internally computed by PrimeTime to compute the effective capacitance of a driver pin of a cell, respectively for minimum operating conditions. The returned parameters are rd, t0, delta_t, and Ceff; they represent how a driver is modeled for computing the effective capacitance.
clock_latency_fall_max	float	The user-specified maximum fall latency (insertion delay) of a pin in the clock network. Set with set_clock_latency.
clock_latency_fall_min	float	The user-specified minimum fall latency (insertion delay) of a pin in the clock network. Set with set_clock_latency.
clock_latency_rise_max	float	The user-specified maximum rise latency (insertion delay) of a pin in the clock network. Set with set_clock_latency.
clock_latency_rise_min	float	The user-specified minimum rise latency (insertion delay) of a pin in the clock network. Set with set_clock_latency.
clock_source_latency_early_fall_max	float	The maximum early falling source latency. Set with set_clock_latency.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
clock_source_latency_early_fall_min	float	The minimum early falling source latency. Set with set_clock_latency.
clock_source_latency_early_rise_max	float	The maximum early rising source latency. Set with set_clock_latency.
clock_source_latency_early_rise_min	float	The minimum early rising source latency. Set with set_clock_latency.
clock_source_latency_late_fall_max	float	The maximum late falling source latency. Set with set_clock_latency.
clock_source_latency_late_fall_min	float	The minimum late falling source latency. Set with set_clock_latency.
clock_source_latency_late_rise_max	float	The maximum late rising source latency. Set with set_clock_latency.
clock_source_latency_late_rise_min	float	The minimum late rising source latency. Set with set_clock_latency.
clocks	string	The collection of clock objects which propagate through this pin. It is undefined if no clocks are present.
constant_value	string	The logic value of a pin tied to logic constant zero or one in the netlist.
direction	string	The direction of a pin. Value can be in, out, inout, or internal. The pin_direction attribute is a synonym for direction. Directions can change as a result of linking a design, as references are resolved.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
disable_timing	Boolean	Disables timing arcs. This has the same effect on timing as not having the arc in the library. Set with set_disable_timing.
driver_model_scaling_libs_max	collection	The collection of library objects used for driver model scaling, where applicable, for libs max analysis.
driver_model_scaling_libs_min	collection	The collection of library objects used for driver model scaling, where applicable, for libs min analysis.
driver_model_type_max_fall	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for max fall analysis.
driver_model_type_max_rise	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for max rise analysis.
driver_model_type_min_fall	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for min fall analysis.
driver_model_type_min_rise	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for min rise analysis.
effective_capacitance_max	float	The effective capacitance for maximum operating conditions. Valid only for driver pins attached to annotated RC networks.
effective_capacitance_min	float	The effective capacitance for minimum operating conditions. Valid only for driver pins attached to annotated RC networks

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
escaped_full_name	string	Contains the name of the cell. Any literal hierarchy characters are escaped with a backslash.
fanout_load	float	A floating-point value representing the fanout load value of a pin. This value is used in computing max_fanout design rule cost.
full_name	string	The complete name of a pin to the top of the hierarchy. For example, the full name of pin Z on cell U2 within cell U1 is U1/U2/Z. The setting of the current instance has no effect on the full name of a pin. See also the lib_pin_name attribute.
hold_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with set_clock_uncertainty.
ideal_latency_max_fall ideal_latency_max_rise ideal_latency_min_fall ideal_latency_min_rise	float	Ideal delay value annotated on a pin in an ideal network, using the set_ideal_latency command.
ideal_transition_max_fall ideal_transition_max_rise ideal_transition_min_fall ideal_transition_min_rise	float	Ideal transition time annotated on a pin in an ideal network, using the set_ideal_transition command.
is_async_pin	Boolean	This attribute is true if a pin is an asynchronous preset/clear pin.
is_clear_pin	Boolean	This attribute is true if a pin is an asynchronous clear pin.
is_clock_gating_pin	Boolean	This attribute is true if a pin is a pin of a clock-gating cell.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
is_clock_pin	Boolean	This attribute is true if a pin is a clock pin of a sequential cell.
is_clock_used_as_clock is_clock_used_as_data	Boolean	When both are true, the clock through the specified pin acts as both clock and data. When is_clock_used_as_clock is true and is_clock_used_as_data is false, the clock through the specified pin acts only as a clock and never as data. When is_clock_used_as_data is true and is_clock_used_as_clock is false, the clock through the specified pin acts only as data and never as clock.
is_data_pin	Boolean	This attribute is true if a pin is a data pin of a sequential cell.
is_driver_scaled_max_fall	Boolean	Specifies whether the pin uses scaling between libraries for the driver model, either true or false, for max fall analysis
is_driver_scaled_max_rise	Boolean	Specifies whether the pin uses scaling between libraries for the driver model, either true or false, for max rise analysis.
is_driver_scaled_min_fall	Boolean	Specifies whether the pin uses scaling between libraries for the driver model, either true or false, for min fall analysis.
is_driver_scaled_min_rise	Boolean	Specifies whether the pin uses scaling between libraries for the driver model, either true or false, for min rise analysis.
is_fall_edge_triggered_clock_pin	Boolean	This attribute is true if a pin is used in the design as a falling-edge-triggered flip-flop clock pin.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
is_fall_edge_triggered_data_pin	Boolean	This attribute is true if a pin is used in the design as a falling-edge-triggered flip-flop data pin.
is_hierarchical	Boolean	This attribute is true for any pins that are instantiations of another design, and false for pins that are instantiations of a library pin (also known as leaf pins).
is_ideal	Boolean	This attribute is `true` if the pin has been marked ideal using the `set_ideal_network` command.
is_mux_select_pin	Boolean	This attribute is true if a pin is the select pin of a multiplexer device.
is_negative_level_sensitive_clock_pin	Boolean	This attribute is true if a pin is used in the design as a negative edge-triggered flip-flop clock pin.
is_negative_level_sensitive_data_pin	Boolean	This attribute is true if a pin is used in the design as a negative edge-triggered flip-flop data pin.
is_port	Boolean	This attribute is true for a pin or a port. Pins of ports are accessible only from a `timing_point`.
is_positive_level_sensitive_clock_pin	Boolean	This attribute is true if a pin is used in the design as a positive edge-triggered flip-flop clock pin.
is_positive_level_sensitive_data_pin	Boolean	This attribute is true if a pin is used in the design as a positive edge-triggered flip-flop data pin.
is_preset_pin	Boolean	This attribute is true if a pin is an asynchronous preset pin.
is_receiver_scaled_max_fall	Boolean	Specifies whether the pin uses scaling between libraries for the receiver model, either true or false, for max fall analysis.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
is_receiver_scaled_max_rise	Boolean	Specifies whether the pin uses scaling between libraries for the receiver model, either true or false, for max rise analysis.
is_receiver_scaled_min_fall	Boolean	Specifies whether the pin uses scaling between libraries for the receiver model, either true or false, for min fall analysis.
is_receiver_scaled_min_rise	Boolean	Specifies whether the pin uses scaling between libraries for the receiver model, either true or false, for min rise analysis.
is_rise_edge_triggered_clock_pin	Boolean	This attribute is true if a pin is used in the design as a rising-edge-triggered flip-flop clock pin.
is_rise_edge_triggered_data_pin	Boolean	This attribute is true if a pin is used in the design as a rising-edge-triggered flip-flop data pin.
is_three_state	Boolean	This attribute is true if a pin is a three-state driver.
is_three_state_enable_pin	Boolean	This attribute is true if a pin is an enable pin of a three-state device.
is_three_state_output_pin	Boolean	This attribute is true if a pin could output a three-state signal.
lib_pin_name	string	The leaf pin name. For example, the lib_pin_name of pin U2/U1/Z is Z. This attribute is read-only.
max_capacitance	float	A floating-point value representing the maximum capacitance design rule limit for a pin.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
max_fall_arrival	float	A floating-point value that specifies the arrival time for the longest path with a falling transition on a pin. In best-case, worst-case mode, this value is for the worst-case operating condition.
max_fall_delay	float	A floating-point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_max_delay.
max_fall_slack	float	A floating-point value representing the worst slack at a pin for falling maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated.
max_fanout	float	A floating-point value representing the maximum fanout design rule limit for a pin.
max_rise_arrival	float	A floating-point value that specifies the arrival time for the longest path with a rising transition on a pin. In best-case, worst-case mode, this value is for the worst-case operating condition.
max_rise_delay	float	A floating-point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_max_delay.
max_rise_slack	float	A floating-point value representing the worst slack at a pin for rising maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
max_time_borrow	float	A floating-point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the technology library. Set with set_max_time_borrow.
max_transition	float	A floating-point value representing the maximum transition time design rule limit for a pin.
min_capacitance	float	A floating-point value representing the minimum capacitance design rule limit for a pin.
min_fall_arrival	float	A floating-point value that specifies the arrival time for the shortest path with a falling transition on a pin. In best-case worst-case mode, this value is for the best-case mode.
min_fall_delay	float	A floating-point value that specifies the minimum falling delay on clocks, pins, cells, or on paths between such objects. Set with set_min_delay.
min_fall_slack	float	A floating-point value representing the worst slack at a pin for falling minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated.
min_fanout	float	A floating-point value representing the minimum fanout design rule limit for a pin.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
min_rise_arrival	float	A floating-point value that specifies the worst hold slack of all paths passing through a pin with a rising transition at a pin. In best-case, worst-case operating conditions, this value is for the best-case condition. If all such paths are unconstrained, the value is infinity.
min_rise_delay	float	A floating-point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_min_delay.
min_rise_slack	float	A floating-point value representing the worst slack at a pin for rising minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated.
min_transition	float	A floating-point value representing the minimum transition time design rule limit for a pin.
object_class	string	The class of the object. This is a constant, equal to pin.
pin_capacitance_max	float	A floating-point value representing the capacitance of a pin for maximum conditions.
pin_capacitance_min	float	A floating-point value representing the capacitance of a pin for minimum conditions.
pin_direction	string	The direction of a pin. Value can be in, out, inout, or internal. This attribute exists for backward compatibility with dc_shell. See the direction attribute.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
power_rail_voltage_min power_rail_voltage_max power_rail_bidir_input_min power_rail_bidir_input_max	float	Specifies the minimum or maximum power rail voltage set on the pin. In the case of a bidirectional pin, the output voltage is returned by default. To access the input voltage of a bidirectional pin, use the "_bidir_input_" attributes.
propagated_clock	Boolean	Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with set_propagated_clock.
rc_input_threshold_pct_rise_max rc_input_threshold_pct_fall_max rc_output_threshold_pct_rise_max rc_output_threshold_pct_fall_max rc_slew_derate_from_library_max rc_slew_lower_threshold_pct_rise_max rc_slew_upper_threshold_pct_rise_max rc_slew_lower_threshold_pct_fall_max rc_slew_upper_threshold_pct_fall_max	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimeTime uses to calculate delays and transition times. These attributes on an instance pin return the library threshold values obtained from the maximum library in a min-max analysis. For more information, see "Characterization Trip Points" on page 9-6.
rc_input_threshold_pct_rise_min rc_input_threshold_pct_fall_min rc_output_threshold_pct_rise_min rc_output_threshold_pct_fall_min rc_slew_derate_from_library_min rc_slew_lower_threshold_pct_rise_min rc_slew_upper_threshold_pct_rise_min rc_slew_lower_threshold_pct_fall_min rc_slew_upper_threshold_pct_fall_min	float	The characterization trip points (waveform measurement thresholds) and slew derating factor that PrimeTime uses to calculate delays and transition times. These attributes on an instance pin return the library threshold values obtained from the minimum library in a min-max analysis. For more information, see "Characterization Trip Points" on page 9-6.
receiver_model_scaling_libs_max	collection	The collection of library objects used for receiver model scaling, where applicable, for libs max analysis.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
receiver_model_scaling_libs_min	collection	The collection of library objects used for receiver model scaling, where applicable, for libs min analysis.
receiver_model_type_max_fall	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type max fall analysis.
receiver_model_type_max_rise	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type max rise analysis.
receiver_model_type_min_fall	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type min fall analysis.
receiver_model_type_min_rise	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type min rise analysis.
setup_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with set_clock_uncertainty.
si_noise_bumps_above_high si_noise_bumps_below_high si_noise_bumps_above_low si_noise_bumps_below_low	string	This attribute returns a string that lists the aggressor nets for the input pin and their corresponding coupled bump heights and widths, considering noise bumps in the above-high, below-high, above-low, or below-low region. The format of the string is: {{*aggr1_name height width*} {*aggr2_name height width*} ... } Height is in volts and width is in library time units.

Table 12-11 Attributes of the pin Object Class (Continued)

Attribute name	Type	Description
si_noise_height_factor_above_high si_noise_height_factor_above_low si_noise_height_factor_below_high si_noise_height_factor_below_low	float	This attribute returns the noise height derating factor for the specified pins, in the above-high, below-high, above-low, or below-low region.
si_noise_height_offset_above_high si_noise_height_offset_above_low si_noise_height_offset_below_high si_noise_height_offset_below_low	float	This attribute returns the noise height derating factor for the specified pins, in the above-high, below-high, above-low, or below-low region.
si_noise_prop_bumps_above_high si_noise_prop_bumps_below_high si_noise_prop_bumps_above_low si_noise_prop_bumps_below_low	string	This attribute returns a string that shows the bump height and width at the input pin caused by noise propagation, in the above-high, below-high, above-low, or below-low region. The format of the string is: {*height width*} Height is in volts and width is in library time units.
si_noise_slack_above_high si_noise_slack_below_high si_noise_slack_above_low si_noise_slack_below_low	float	This attribute returns the amount of noise slack for the pin in the above-high, below-high, above-low, or below-low region.
si_noise_total_bump_above_high si_noise_total_bump_above_low si_noise_total_bump_below_high si_noise_total_bump_below_low	string	This attribute returns a string that shows the total bump height and width at the input pin caused by crosstalk and noise propagation, in the above-high, below-high, above-low, or below-low region. The format of the string is: {*height width*} Height is in volts and width is in library time units.
si_noise_width_factor_above_high si_noise_width_factor_above_low si_noise_width_factor_below_high si_noise_width_factor_below_low	float	This attribute returns the noise width derating factor for the specified pins, in the above-high, below-high, above-low, or below-low region.
user_case_value	string	The user-specified logic value of a pin or port.

Port Object Class Attributes

Table 12-12 lists the port object class attributes.

Table 12-12 Attributes of the port Object Class

Attribute name	Type	Description
actual_fall_transition_max	float	A floating-point value representing the largest falling transition time for a port. You cannot set this attribute.
actual_fall_transition_min	float	A floating-point value representing the smallest falling transition time for a port. You cannot set this attribute.
actual_rise_transition_max	float	A floating-point value representing the largest rising transition time for a port. You cannot set this attribute.
actual_rise_transition_min	float	A floating-point value representing the smallest rising transition time for a port. You cannot set this attribute.
annotated_fall_transition_delta_max	float	This attribute shows the additional transition time added to the maximum falling transition time on a port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_fall_transition_delta_min	float	This attribute shows the additional transition time added to the minimum falling transition time on a port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
annotated_rise_transition_delta_max	float	This attribute shows the additional transition time added to the maximum rising transition time on a port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_rise_transition_delta_min	float	This attribute shows the additional transition time added to the minimum rising transition time on a port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during the crosstalk analysis.
arrival_window	string	The minimum and maximum arrivals for rising and falling transitions.
case_value	string	The user-specified logic value of a pin or port propagated from a case analysis or logic constant in the design.
ceff_params_max	string	The parameters used to find the maximum effective capacitance for each timing arc feeding a driver pin attached to an annotated RC network. The parameters specify a linear driver model (rd = drive resistance, t0 = start of voltage ramp, delta_t = duration of voltage ramp, ceff = effective capacitance).

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
ceff_params_min	string	The parameters used to find the minimum effective capacitance for each timing arc feeding a driver pin attached to an annotated RC network. The parameters specify a linear driver model (rd = drive resistance, t0 = start of voltage ramp, delta_t = duration of voltage ramp, ceff = effective capacitance).
clock_latency_fall_max	float	The user-specified maximum fall latency (insertion delay) for clock networks through a port. Set with set_clock_latency.
clock_latency_fall_min	float	The user-specified minimum fall latency (insertion delay) for clock networks through a port. Set with set_clock_latency.
clock_latency_rise_max	float	The user-specified maximum rise latency (insertion delay) for clock networks through a port. Set with set_clock_latency.
clock_latency_rise_min	float	The user-specified minimum rise latency (insertion delay) for clock networks through a port. Set with set_clock_latency.
clock_source_latency_early_fall_max	float	The maximum early falling source latency. Set with set_clock_latency.
clock_source_latency_early_fall_min	float	The minimum early falling source latency. Set with set_clock_latency.
clock_source_latency_early_rise_max	float	The maximum early rising source latency. Set with set_clock_latency.
clock_source_latency_early_rise_min	float	The minimum early rising source latency. Set with set_clock_latency.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
clock_source_latency_late_fall_max	float	The maximum late falling source latency. Set with set_clock_latency.
clock_source_latency_late_fall_min	float	The minimum late falling source latency. Set with set_clock_latency.
clock_source_latency_late_rise_max	float	The maximum late rising source latency. Set with set_clock_latency.
clock_source_latency_late_rise_min	float	The minimum late rising source latency. Set with set_clock_latency.
clocks	collection	The collection of clock objects which propagate through this port. It is undefined if no clocks are present.
constant_value	string	The logic value of a port tied to logic constant zero or one in the netlist.
direction	string	The direction of a port. Value can be in, out, inout, or internal. The port_direction attribute is a synonym for direction. You cannot set this attribute.
disable_timing	Boolean	This attribute is true if the timing for the port has been marked as disabled with the set_disable_timing command.
drive_resistance_fall_max	float	A floating-point value representing the linear drive resistance for falling delays and maximum conditions, associated with an input or inout port. Set with the set_drive command.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
drive_resistance_fall_min	float	A floating-point value representing the linear drive resistance for falling delays and minimum conditions, associated with an input or inout port. Set with the set_drive command.
drive_resistance_rise_max	float	A floating-point value representing the linear drive resistance for rising delays and maximum conditions, associated with an input or inout port. Set with the set_drive command.
drive_resistance_rise_min	float	A floating-point value representing the linear drive resistance for rising delays and minimum conditions, associated with an input or inout port. Set with the set_drive command.
driver_model_scaling_libs_max	collection	The collection of library objects used for driver model scaling, where applicable, for libs max analysis.
driver_model_scaling_libs_min	collection	The collection of library objects used for driver model scaling, where applicable, for libs min analysis.
driver_model_type_max_fall	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for max fall analysis.
driver_model_type_max_rise	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for max rise analysis.
driver_model_type_min_fall	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for min fall analysis.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
driver_model_type_min_rise	string	The driver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for min rise analysis.
driving_cell_dont_scale	Boolean	When true, indicates not to scale the transition time on the port using the driving cell. Otherwise the transition time is scaled by operating condition factors. Set with set_driving_cell.
driving_cell_fall_max	string	A string that names a library cell from which to copy max fall drive capability to be used in fall transition calculation for the port. Set with set_driving_cell.
driving_cell_fall_min	string	A string that names a library cell from which to copy the minimum fall drive capability to be used in fall transition calculation for the port. Set with set_driving_cell.
driving_cell_from_pin_fall_max	string	A string that names the driving_cell_fall_max input pin to be used to find timing arc max fall drive capability. Set with set_driving_cell.
driving_cell_from_pin_fall_min	string	A string that names the driving_cell_fall_min input pin to be used to find timing arc minimum fall drive capability. Set with set_driving_cell.
driving_cell_from_pin_rise_max	string	A string that names the driving_cell_rise_max input pin to be used to find timing arc rise drive capability. Set with set_driving_cell.
driving_cell_from_pin_rise_min	string	A string that names the driving_cell_rise_min input pin to be used to find timing arc rise drive capability. Set with set_driving_cell.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
driving_cell_library_fall_max	string	A string that names the library in which to find the driving_cell_fall_max. Set with set_driving_cell.
driving_cell_library_fall_min	string	A string that names the library in which to find the driving_cell_fall_min. Set with set_driving_cell.
driving_cell_library_rise_max	string	A string that names the library in which to find the driving_cell_rise_max. Set with set_driving_cell.
driving_cell_library_rise_min	string	A string that names the library in which to find the driving_cell_rise_min. Set with set_driving_cell.
driving_cell_max_fall_itrans_fall	float	The value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the falling transition at the from_pin of the driving cell that is used to compute the falling transition value at a pin that drives the port.
driving_cell_max_fall_itrans_rise	float	The value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the falling transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
driving_cell_max_rise_itrans_fall	float	The value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the rising transition value at the from_pin of the driving cell that is used to compute falling transition value at the pin that drives the port.
driving_cell_max_rise_itrans_rise	float	The value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the rising transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.
driving_cell_min_fall_itrans_fall	float	The value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum falling transition value at the from_pin of the driving cell that is used to compute the falling transition value at the pin that drives the port.
driving_cell_min_fall_itrans_rise	float	The value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum falling transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
driving_cell_min_rise_itrans_fall	float	The value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum rise transition value at the from_pin of the driving cell that is used to compute the falling transition value at the pin that drives the port.
driving_cell_min_rise_itrans_rise	float	The value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum rise transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.
driving_cell_multiply_by	float	A floating-point value that multiplies the transition time of the port marked with this attribute. Set with set_driving_cell.
driving_cell_no_design_rule	Boolean	This attribute is true if driving cell information has been set on a port with set_driving_cell -no_design_rule. If true, the driving cell's design rule limits (max_capacitance and so forth) are not used for the port.
driving_cell_pin_fall_max	string	A string that names the driving_cell_fall_max output pin to be used to find timing arc fall drive capability. Set with set_driving_cell.
driving_cell_pin_fall_min	string	A string that names the driving_cell_fall_min output pin to be used to find timing arc fall drive capability. Set with set_driving_cell.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
driving_cell_pin_rise_max	string	A string that names the driving_cell_rise_max output pin to be used to find timing arc rise drive capability. Set with set_driving_cell.
driving_cell_pin_rise_min	string	A string that names the driving_cell_rise_min output pin to be used to find timing arc rise drive capability. Set with set_driving_cell.
driving_cell_rise_max	string	A string that names a library cell from which to copy max rise drive capability to be used in rise transition calculation for the port. Set with set_driving_cell.
driving_cell_rise_min	string	A string that names a library cell from which to copy the minimum rise drive capability to be used in rise transition calculation for the port. Set with set_driving_cell.
effective_capacitance_max	float	The effective capacitance for maximum operating conditions. Valid only for driver pins attached to annotated RC networks.
effective_capacitance_min	float	The effective capacitance for minimum operating conditions. Valid only for driver pins attached to annotated RC networks.
escaped_full_name	string	Contains the name of the cell. Any literal hierarchy characters are escaped with a backslash.
fanout_load	float	Specifies the fanout load on output ports. Set with set_fanout_load.
full_name	string	The name of a port. You cannot set this attribute.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
hold_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with set_clock_uncertainty.
ideal_latency_max_fall ideal_latency_max_rise ideal_latency_min_fall ideal_latency_min_rise	float	Ideal delay value annotated on a port in an ideal network, using the set_ideal_latency command.
ideal_transition_max_fall ideal_transition_max_rise ideal_transition_min_fall ideal_transition_min_rise	float	Ideal transition time annotated on a port in an ideal network, using the set_ideal_transition command.
input_transition_fall_max	float	A floating-point value representing the fixed transition time for falling delays, maximum conditions associated with an input or inout port. Set with set_input_transition.
input_transition_fall_min	float	A floating-point value representing the fixed transition time for falling delays and minimum conditions associated with an input or inout port. Set with set_input_transition.
input_transition_rise_max	float	A floating-point value representing the fixed transition time for rising delays and maximum conditions associated with an input or inout port. Set with set_input_transition.
input_transition_rise_min	float	A floating-point value representing the fixed transition time for rising delays and minimum conditions associated with an input or inout port. Set with set_input_transition.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
`is_clock_used_as_clock` `is_clock_used_as_data`	`Boolean`	When both true, the clock through the specified pin acts as both clock and data. When `is_clock_used_as_clock` is `true` and `is_clock_used_as_data` is `false`, the clock through the specified pin acts only as a clock and never as data. When `is_clock_used_as_data` is `true` and `is_clock_used_as_clock` is `false`, the clock through the specified pin acts only as data and never as clock.
`is_driver_scaled_max_fall`	`Boolean`	Specifies whether the port uses scaling between libraries for the driver model, either true or false, for max fall analysis.
`is_driver_scaled_max_rise`	`Boolean`	Specifies whether the port uses scaling between libraries for the driver model, either true or false, for max rise analysis.
`is_driver_scaled_min_fall`	`Boolean`	Specifies whether the port uses scaling between libraries for the driver model, either true or false, for min fall analysis.
`is_driver_scaled_min_rise`	`Boolean`	Specifies whether the port uses scaling between libraries for the driver model, either true or false, for min rise analysis.
`is_ideal`	`Boolean`	This attribute is `true` if the port has been marked ideal using the `set_ideal_network` command.
`is_receiver_scaled_max_fall`	`Boolean`	Specifies whether the port uses scaling between libraries for the receiver model, either true or false, for max fall analysis.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
is_receiver_scaled_max_rise	Boolean	Specifies whether the port uses scaling between libraries for the receiver model, either true or false, for max rise analysis.
is_receiver_scaled_min_fall	Boolean	Specifies whether the port uses scaling between libraries for the receiver model, either true or false, for min fall analysis.
is_receiver_scaled_min_rise	Boolean	Specifies whether the port uses scaling between libraries for the receiver model, either true or false, for min rise analysis.
max_capacitance	float	A floating-point number that sets the maximum capacitance value for input, output, or bidirectional ports, and designs. The units must be consistent with those of the technology library used during optimization. Set with set_max_capacitance.
max_fall_delay	float	A floating-point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_max_delay.
max_fall_slack	float	A floating-point value representing the worst slack at a port for falling maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated. You cannot set this attribute.
max_fanout	float	Specifies the maximum fanout load for the net connected to this port. PrimeTime ensures that the fanout load on this port is less than the specified value. Set with set_max_fanout.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
max_rise_delay	float	A floating-point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_max_delay.
max_rise_slack	float	A floating-point value representing the worst slack at a port for rising maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.
max_transition	float	Specifies the maximum transition time for the net connected to this port. The compile command ensures that value. Set with set_max_transition.
min_capacitance	float	A floating-point number that sets the minimum capacitance value for input and bidirectional ports. The units must be consistent with those of the technology library used. Set with set_min_capacitance.
min_fall_delay	float	A floating-point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_min_delay.
min_fall_slack	float	A floating-point value representing the worst slack at a port for falling minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.
min_fanout	float	A floating-point value representing the minimum fanout design rule limit for a port. Set with set_min_fanout.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
min_rise_delay	float	A floating-point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with set_min_delay.
min_rise_slack	float	A floating-point value representing the worst slack at a port for rising minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.
min_transition	float	A floating-point value representing the minimum transition time design rule limit for a port. Set with set_min_transition.
object_class	string	The class of the object. This is a constant, equal to port. You cannot set this attribute.
pin_capacitance_max	float	A floating-point value representing the pin capacitance of a port for maximum conditions (wire capacitance is not included). Set with set_load.
pin_capacitance_min	float	A floating-point value representing the pin capacitance of a port for minimum conditions (wire capacitance is not included). Set with set_load.
port_direction	string	The direction of a port. Value can be in, out, or inout. This attribute exists for backward compatibility with dc_shell. See the direction attribute. You cannot set this attribute.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
propagated_clock	Boolean	Specifies that the clock edge times be delayed by propagating the values through the clock network. Affects all sequential cells in the transitive fanout of this port. If this attribute is not present, PrimeTime assumes ideal clocking. Set with set_propagated_clock command.
receiver_model_scaling_ libs_max	collection	The collection of library objects used for receiver model scaling, where applicable, for libs max analysis.
receiver_model_scaling_libs_min	collection	The collection of library objects used for receiver model scaling, where applicable, for libs min analysis.
receiver_model_type_max_fall	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type max fall analysis.
receiver_model_type_max_rise	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type max rise analysis.
receiver_model_type_min_fall	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type min fall analysis.
receiver_model_type_min_rise	string	The receiver model type, either “basic” (NLDM) or “advanced” (CCS Timing), for type min rise analysis.
setup_uncertainty	float	A floating-point value that specifies the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with set_clock_uncertainty.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
si_noise_active_aggressors_above_high si_noise_active_aggressors_above_low si_noise_active_aggressors_below_high si_noise_active_aggressors_below_low	collection	This attribute returns a collection of active aggressor nets for the input port, considering crosstalk noise bumps in the above-high, below-high, above-low, or below-low region. An active aggressor is an aggressor that contributes to the worst-case noise bump on the victim net.
si_noise_bumps_above_high si_noise_bumps_above_low si_noise_bumps_below_high si_noise_bumps_below_low	string	This attribute returns a string that lists the aggressor nets for the input port and their corresponding coupled bump heights and widths, considering noise bumps in the above-high, below-high, above-low, or below-low region. The format of the string is: {{*aggr1_name height width*} {*aggr2_name height width*} ... } Height is in volts and width is in library time units.
si_noise_height_factor_above_high si_noise_height_factor_above_low si_noise_height_factor_below_high si_noise_height_factor_below_low	float	This attribute returns the noise height derating factor for the specified ports, in the above-high, below-high, above-low, or below-low region.
si_noise_height_offset_above_high si_noise_height_offset_above_low si_noise_height_offset_below_high si_noise_height_offset_below_low	float	This attribute returns the noise height offset derating factor for the specified ports, in the above-high, below-high, above-low, or below-low region.
si_noise_prop_bumps_above_high si_noise_prop_bumps_above_low si_noise_prop_bumps_below_high si_noise_prop_bumps_below_low	string	This attribute returns a string that shows the bump height and width at the input port caused by noise propagation, in the above-high, below-high, above-low, or below-low region. The format of the string is: {*height width*} Height is in volts and width is in library time units.

Table 12-12 Attributes of the port Object Class (Continued)

Attribute name	Type	Description
si_noise_slack_above_high si_noise_slack_above_low si_noise_slack_below_high si_noise_slack_below_low	float	This attribute returns the amount of noise slack for the port in the above-high, below-high, above-low, or below-low region.
si_noise_total_bump_above_high si_noise_total_bump_above_low si_noise_total_bump_below_high si_noise_total_bump_below_low	string	This attribute returns a string that shows the total bump height and width at the input port caused by crosstalk and noise propagation, in the above-high, below-high, above-low, or below-low region. The format of the string is: {*height width*} Height is in volts and width is in library time units.
si_noise_width_factor_above_high si_noise_width_factor_above_low si_noise_width_factor_below_high si_noise_width_factor_below_low	float	This attribute returns the noise width derating factor for the specified ports, in the above-high, below-high, above-low, or below-low region.
user_case_value	string	The user-specified logic value of a pin or port.
wire_capacitance_max wire_capacitance_min	float	A floating-point value representing the wire capacitance of a port for maximum/minimum conditions (pin capacitance is not included). Set with set_load.
wire_load_model_max	string	The name of a wire load model (for maximum conditions) that can be used for prelayout wire load estimation of the net connected to a port. Set with set_wire_load_model.
wire_load_model_min	string	The name of a wire load model (for minimum conditions) that can be used for prelayout wire load estimation of the net connected to a port. Set with set_wire_load_model.

Timing Arc Object Class Attributes

Table 12-13 lists the timing arc object class attributes.

Table 12-13 Attributes of the timing_arc Object Class

Attribute name	Type	Description
annotated_delay_delta_max_fall	float	Specifies a delay that is added to the maximum falling delay of a timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_delay_delta_max_rise	float	Specifies a delay that is added to the maximum rising delay of a timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_delay_delta_min_fall	float	Specifies a delay that is added to the minimum falling delay of a timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during the crosstalk analysis.
annotated_delay_delta_ min_rise	float	Specifies a delay that is added to the minimum rising delay of a timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during the crosstalk analysis.
delay_max_fall	float	A floating-point value representing the maximum falling delay of the timing arc.
delay_max_rise	float	A floating-point value representing the maximum rising delay of the timing arc.
delay_min_fall	float	A floating-point value representing the minimum falling delay of the timing arc.
delay_min_rise	float	A floating-point value representing the minimum rising delay of the timing arc.

Table 12-13 Attributes of the timing_arc Object Class (Continued)

Attribute name	Type	Description
from_pin	collection	A collection containing the from pin of the timing arc.
is_annotated_fall_max	Boolean	Returns true if the maximum falling delay of the timing arc is back-annotated.
is_annotated_fall_min	Boolean	Returns true if the minimum falling delay of the timing arc is back-annotated.
is_annotated_rise_max	Boolean	Returns true if the maximum rising delay of the timing arc is back-annotated.
is_annotated_rise_min	Boolean	Returns true if the minimum rising delay of the timing arc is back-annotated.
is_cellarc	Boolean	Returns true if the timing arc is a cell arc, and false for net arcs.
is_constraint_scaled_max_fall	Boolean	Specifies whether the constraint timing arcs uses scaling between libraries for the driver model, either true or false, for max fall analysis.
is_constraint_scaled_max_rise	Boolean	Specifies whether the constraint timing arcs uses scaling between libraries for the driver model, either true or false, for max rise analysis.
is_constraint_scaled_min_fall	Boolean	Specifies whether the constraint timing arcs uses scaling between libraries for the driver model, either true or false, for min fall analysis.
is_constraint_scaled_min_rise	Boolean	Specifies whether the constraint timing arcs uses scaling between libraries for the driver model, either true or false, for min rise analysis.
is_disabled	Boolean	Returns true if the timing arc is disabled.
is_user_disabled	Boolean	Returns true if the timing arc is disabled by using the set_disable_timing command.

Table 12-13 Attributes of the timing_arc Object Class (Continued)

Attribute name	Type	Description
mode	string	Returns the mode string of the timing arc.
object_class	string	The class of the object. This is a constant equal to “timing_arc.”
sdf_cond	string	Returns the SDF condition of the timing arc.
sdf_cond_end	string	Returns the SDF condition at the endpoint of the timing arc. Variable sdf_enable_cond_start_end must be true.
sdf_cond_start	string	Returns the SDF condition at the startpoint of the timing arc. Variable sdf_enable_cond_start_end must be true.
sense	string	Returns the sense of the timing arc.
to_pin	collection	A collection containing the to-pin of the timing arc.
when	string	Returns the when string of the timing arc.

Timing Path Object Class Attributes

Table 12-14 lists the timing path object class attributes.

Table 12-14 Attributes of the timing_path Object Class

Attribute name	Type	Description
`arrival`	`float`	The arrival time at the endpoint of the timing_path.
`capture_clock_paths`	`collection`	Returns timing path collections for the capture clock. The path corresponds to the output you see from `report_timing -path full_clock_expanded`. If the capturing registers are clocks by a regular clock, it returns only one path in the collection. If it's a generated clock, the first path in the collection is the master clock path, followed by each dependent generated clock until it reaches the register's clock pin.
`clock_uncertainty`	`float`	The clock uncertainty of the timing_path. The uncertainty can be defined with the `set_clock_uncertainty` command.
`common_path_pessimism`	`float`	The value of the clock reconvergence common path pessimism. This attribute is defined only if you are using on-chip variation analysis and have specified the `-report_clock_reconvergence_pessimism` option to `get_timing_paths`.
`crpr_common_point`	`collection`	Returns the clock pin corresponding to the common pin used for CRP calculation for the path. If the launch/capture clock is ideal, the clock source will be returned as the common point. If the launch clock is different than the capture clock, the attribute will not exist on the path.

`depth_cell_capture`
`depth_cell_launch`
`depth_net_capture`
`depth_net_launch`
	`float`	Advanced on-chip variation (OCV) calculated depth for a path.

Table 12-14 Attributes of the timing_path Object Class (Continued)

Attribute name	Type	Description
distance_cell distance_net	float	Advanced on-chip variation (OCV) calculated distance for a path.
dominant_exception	string	Exists only if the path has at least a timing exception. If so, it specifies the type of the dominant exceptions: false_path, multicycle_path, and min_max_delay. See the -exceptions option of the report_timing command.
endpoint	string	The timing endpoint name of the timing_path, for example, U1/U5/par_reg/D.
endpoint_clock	string	The clock name of the clock at the path endpoint.
endpoint_clock_close_edge_type	string	The type of clock edge (rise or fall) that closes (latches) the data.
endpoint_clock_close_edge_value	float	The value of the closing edge of the endpoint clock.
endpoint_clock_is_inverted	Boolean	Returns true if the endpoint clock has been inverted.
endpoint_clock_is_propagated	Boolean	Returns true if the endpoint clock is a propagated clock, false if it is an ideal clock. You can set a clock as propagated using the set_propagated_clock command.
endpoint_clock_latency	float	The latency of the endpoint clock. If the clock is propagated, it is the computed latency (or delay) from the clock source to the endpoint. You can set clock latency using the set_clock_latency command.
endpoint_clock_open_edge_type	string	The type of clock edge (rise or fall) that opens the endpoint latch. If the endpoint is edge-triggered, the open and close edges are the same.
endpoint_clock_open_edge_value	float	The value of the opening edge of the endpoint clock.

Table 12-14 Attributes of the timing_path Object Class (Continued)

Attribute name	Type	Description
endpoint_clock_pin	string	The complete path name of the endpoint clock pin, for example, U23/U_reg/out_reg[2]/CP.
endpoint_hold_time_value	float	The value of the register hold time at the timing endpoint. For example, for a flip-flop this would be the library hold time for the flip-flop cell.
endpoint_is_level_sensitive	Boolean	Returns true if the endpoint is a level-sensitive device, for example, a latch. Returns false if the endpoint is edge-triggered.
endpoint_output_delay_value	float	The value of the output delay of the timing endpoint. You can set the output delay value with set_output_delay.
endpoint_recovery_time_value	float	The value of the recovery time at the timing endpoint. Recovery and removal times are often defined for the asynchronous set and clear pins of registers.
endpoint_removal_time_value	float	The value of the removal time at the timing endpoint. Recovery and removal times are often defined for the asynchronous set/clear pins of registers.
endpoint_setup_time_value	float	The value of the setup time at the timing endpoint. Recovery and removal times are often defined for the asynchronous set and clear pins of registers.
endpoint_unconstrained_reason	string	Exists only if the path endpoint is unconstrained. If so, it specifies the reason why the endpoint is unconstrained. Possible reasons include no_capture_clock, dangling_end_point, and fanin_of_disabled.

Table 12-14 Attributes of the timing_path Object Class (Continued)

Attribute name	Type	Description
launch_clock_paths	collection	Returns timing path collections for the launch clock. The path corresponds to the output you see from report_timing -path full_clock_expanded. If the launching registers are clocks by a regular clock, it returns only one path in the collection. If it's a generated clock, the first path in the collection is the master clock path, followed by each dependent generated clock until it reaches the register's clock pin.
object_class	string	The class of the object. This is a constant, equal to timing_path. You cannot set this attribute.
path_group	collection	The path group of the timing path.
path_type	string	The type of timing path (maximum or minimum). A path for a setup check is path_type of maximum.
points	collection	Returns a collection of the timing points that comprise a timing path. For example, the timing points listed in the left-hand column of a report_timing command correspond to this collection. A single timing path can consist of many timing points. You can iterate through each timing point using foreach_in_collection.
required	float	The required time value for the timing path. Corresponds to the data required time of a timing report.
slack	float	The slack of the timing path. Negative values represent violated paths. Corresponds to the slack of a timing report.
startpoint	collection	The startpoint of the timing path. Corresponds to the startpoint in the header of a timing report.

Table 12-14 Attributes of the timing_path Object Class (Continued)

Attribute name	Type	Description
startpoint_clock	collection	The startpoint clock name of the timing path.
startpoint_clock_is_inverted	Boolean	Returns true if the startpoint clock is inverted.
startpoint_clock_is_propagated	Boolean	Returns true if the startpoint clock is a propagated clock, false if it is an ideal clock. You can set a clock as propagated using the set_propagated_clock command.
startpoint_clock_latency	float	The latency of the startpoint clock. If the clock is propagated, it is the computed latency (or delay) from the clock source to the endpoint. You can set clock latency using the set_clock_latency command.
startpoint_clock_open_edge_type	string	The type of clock edge (rise or fall) that launches the data.
startpoint_clock_open_edge_value	float	The value of the opening edge of the startpoint clock.
startpoint_input_delay_value	float	The value of the startpoint input delay.
startpoint_is_level_sensitive	Boolean	Returns true if the startpoint is a level-sensitive device, such as a latch. Returns false if the startpoint is edge-triggered.
startpoint_unconstrained_reason	string	Exists only if the path startpoint is unconstrained. If so, it specifies the reason why the startpoint is unconstrained. Possible reasons include no_launch_clock, dangling_start_point, and fanout_of_disabled.
time_borrowed_from_endpoint	float	Returns the amount of time borrowed from the timing endpoint. Time borrowing occurs in paths involving level-sensitive devices.

Table 12-14 Attributes of the timing_path Object Class (Continued)

Attribute name	Type	Description
time_lent_to_startpoint	float	Returns the amount of time lent to the timing startpoint. Time borrowing occurs in paths involving level-sensitive devices.
transparent_latch_paths	collection	For paths with a transparent latch D-pin startpoint, if the startpoint is borrowing, the attribute returns the chain of "upstream" borrowing paths that lead up to the borrowing startpoint. To use the attribute, you must gather the path using the -trace_latch_borrow option.

Timing Point Object Class Attributes

Table 12-15 lists the point object class attributes.

Table 12-15 Attributes of the timing_point Object Class

Attribute name	Type	Description
arrival	string	The arrival time at the timing point, but not accounting for the following. - Clock latency to the startpoint clock - Time lent to the startpoint (due to latch borrowing) - Input delay - startpoint_clock_open_edge_value These must be added to the arrival attribute value to arrive at the total arrival time based on the desired startpoint.
object	string	The object at this point in the timing path.
object_class	string	The class of the object. This is a constant, equal to timing_point. You cannot set this attribute.
rise_fall	string	Returns rise if the timing point is a rising-edge delay. Returns fall if the timing point is a falling-edge delay.
slack	string	The slack value at the timing point.
transition	string	The transition value at the timing point.

Using Paths to Generate Custom Reports

You can use the `get_timing_paths` command to create a collection of paths for custom reporting and other processing. You can assign these timing paths to a variable or pass them into another command. For a list of supported timing path object class attributes, see Table 12-14 on page 12-79.

Each use of the `get_timing_paths` command has its own context. You cannot compare, add, or remove objects taken from different contexts. For example,

```
pt_shell> set paths1 [get_timing_paths -nworst 10]
...
pt_shell> set paths2 [get_timing_paths -nworst 100]
...
```

Even though the variables paths1 and paths2 contain some of the same objects, the collections cannot be compared because they come from different contexts. To do such comparisons, create one large collection containing all the objects of interest, and then perform filtering or other manipulation on that collection.

Use the `foreach_in_collection` command to iterate among the paths in the collection. The collection commands `index_collection`, `copy_collection`, `add_to_collection`, and `remove_from_collection` are not applicable to timing path collections. You can use the `get_attribute` command to obtain information about the paths.

One attribute of a timing path is the points collection. A point corresponds to a pin or port along the path. Iterate through these points using the `foreach_in_collection` command and get the attributes on them using the `get_attribute` command. For more information, see the man pages for the `foreach_in_collection` and `get_timing_paths` commands.

Example

You can find some detailed examples of custom timing reports in *install_directory*/auxx/pt/examples/tcl (where *install_directory* is the installation directory for PrimeTime).

The following procedure prints out the startpoint name, endpoint name, and the slack of the worst path in each path group:

```
proc custom_report_worst_path_per_group {} {
    echo [format "%-20s %-20s %7s" "From" "To" "Slack"]
    echo "------------------------------------------------------------"
    foreach_in_collection path [get_timing_paths] {
        set slack [get_attribute $path slack]
        set startpoint [get_attribute $path startpoint]
        set endpoint [get_attribute $path endpoint]
        echo [format "%-20s %-20s %s" [get_attribute $startpoint
full_name] \
```

```
                    [get_attribute $endpoint full_name] $slack]
    }
}

pt_shell> custom_report_worst_path_per_group

From                  To                       Slack
-------------------------------------------------------
ffa/CP                QA                     0.1977
ffb/CP                ffd/D                  3.8834
```

This example shows worst negative slack, total negative slack, and total positive slack for the current design:

```
proc report_design_slack_information {} {
  set design_tns 0
  set design_wns 100000
  set design_tps 0
  foreach_in_collection group [get_path_groups *] {
    set group_tns 0
    set group_wns 100000
    set group_tps 0
    foreach_in_collection path [get_timing_paths -nworst 10000 \
       -group $group] {
      set slack [get_attribute $path slack]
      if {$slack < $group_wns} {
        set group_wns $slack
        if {$slack < $design_wns} {
          set design_wns $slack
        }
      }
      if {$slack < 0.0} {
        set group_tns [expr $group_tns + $slack]
      } else {
        set group_tps [expr $group_tps + $slack]
      }
}
    set design_tns [expr $design_tns + $group_tns]
    set design_tps [expr $design_tps + $group_tps]
    set group_name [get_attribute $group full_name]
    echo [format "Group '%s' Worst Negative Slack : %g" $group_name
$group_wns]
    echo [format "Group '%s' Total Negative Slack : %g" $group_name
$group_tns]
    echo [format "Group '%s' Total Positive Slack : %g" $group_name
$group_tps]
    echo ""
  }
  echo "----------------------------------------"
  echo [format "Design Worst Negative Slack : %g" $design_wns]
  echo [format "Design Total Negative Slack : %g" $design_tns]
  echo [format "Design Total Positive Slack : %g" $design_tps]
```

```
}


pt_shell> report_design_slack_information

Group 'CLK' Worst Negative Slack : -3.1166
Group 'CLK' Total Negative Slack : -232.986
Group 'CLK' Total Positive Slack : 4.5656

Group 'vclk' Worst Negative Slack : -4.0213
Group 'vclk' Total Negative Slack : -46.1982
Group 'vclk' Total Positive Slack : 0

----------------------------------------
Design Worst Negative Slack : -4.0213
Design Total Negative Slack : -279.184
Design Total Positive Slack : 4.5656
```

Using Arcs to Generate Custom Reports

To create a collection of timing arcs for custom reporting and other processing, use the `get_timing_arcs` command. You can assign these timing arcs to a variable and get the desired attributes for further processing. For a list of supported timing arc object class attributes, see Table 12-13 on page 12-76.

Use the `foreach_in_collection` command to iterate among the arcs in the collection. You can use the `get_attribute` command to obtain information about the arcs. However, you cannot copy, sort, or index the collection of arcs. You can also use the `-filter` option of `get_timing_arcs` to obtain just the arcs that satisfy specified conditions, but you cannot use the `filter_collection` command to filter an already generated collection of arcs.

One attribute of a timing arc is `from_pin`, which is the pin or port from which the timing arc begins. In the same way, `to_pin` is the pin or port at which the timing arc ends. For more information about the `from_pin` and `to_pin` attributes, use the `get_attribute` command.

Example

To list the maximum rise delay value of all the timing arcs of the cell U1 that have the positive_unate arc, enter

```
pt_shell> set arcs [get_timing_arcs -of_objects U1 -filter \
             "sense == positive_unate"]
_sel3
pt_shell> foreach_in_collection arc $arcs {
             echo [get_attribute $arc delay_max_rise]
             }
0.846862
0.846862
```

Creating a Collection of Library Arcs

To create a collection of library arcs for custom reporting and other processing, use the `get_lib_timing_arcs` command. You can assign these library arcs to a variable and get the desired attributes for further processing.

Use the `foreach_in_collection` command to iterate among the library arcs in the collection. You can use the `get_attribute` command to obtain information about the arcs. However, you cannot copy, sort, or index a collection of library arcs. You can also use the `-filter` option of `get_lib_timing_arcs` to obtain just the library arcs that satisfy specified conditions, but you cannot use the `filter_collection` command to filter a collection of library arcs. For a list of supported library timing arc object class attributes, see Table 12-8 on page 12-32.

One attribute of a library timing arc is `from_lib_pin`, which is the library pin from which the timing arc begins. In the same way, `to_lib_pin` is the library pin at which the timing arc ends. To obtain information about the `from_lib_pin` and `to_lib_pin` attributes, use the `get_attribute` command. For more information, see the man page for the `foreach_in_collection` command.

Example

To list the senses of timing arcs starting from the clock pin of a flip-flop library cell, enter

```
pt_shell> set larcs [get_lib_timing_arcs -from class/FD1/CP]
_sel5
pt_shell> foreach_in_collection larc $larcs \ {
           { echo [get_attribute $larc sense] }
hold_clk_rise
setup_clk_rise
rising_edge
rising_edge
```

Reporting Library Data and Driver Information

To report library data and the resulting driver model parameters for a specified library arc and conditions, you can use the `report_driver_model` command. The reported information is useful for validating library data and driver models used in delay calculation with annotated parasitics. For more information, see the `report_driver_model` man page.

A

Writing Mapped SDF Files

The Standard Delay Format (SDF) mapping feature of PrimeTime lets you specify the output format of the SDF file. You start by creating an SDF map file, which specifies the syntax and timing arcs written to the SDF file for cells in the library. For example, if you want to change the SDF output for a flip-flop in the library, you define a map for that cell. You apply the mapping by using the `-map` option of the `write_sdf` command.

The SDF mapping feature is described in the following sections:

Specifying Timing Labels in the Library

To reference delays of timing arcs in the SDF mapping mechanism, you need to specify a label to the timing arcs in the library. The `timing_label` attribute is written in the timing group of the library file to label a timing arc. This will serve to label arcs for supplying values for the following SDF constructs: `IOPATH`, `SETUP`, `HOLD`, `SETUPHOLD`, `RECOVERY`, `REMOVAL`, `RECREM`, `NOCHANGE`, and `WIDTH`.

The following example shows how to label a timing arc A_Z.

```
cell(IV_I) {
  area : 1;
  pin(A) {
    direction : input;
    capacitance : 1;
  }
  pin(Z) {
    direction : output;
    function : "A'";
    timing() {
      related_pin : "A";
      timing_label : "A_Z";
      rise_propagation(onebyone) {
        values("0.380000");
      }
      rise_transition(tran) {
        values("0.03, 0.04, 0.05");
      }
      fall_propagation(prop) {
        values("0.15, 0.17, 0.20");
      }
      fall_transition(tran) {
        values("0.02, 0.04, 0.06");
      }
        }
          }
```

Specifying the min_pulse_width Constraint

You can specify the minimum pulse width arcs in two ways in the library: by attaching attributes to the pins or by having a `min_pulse_width` group section inside the pin section. When specifying the `min_pulse_width` pin constraint, use one of the following styles when writing the library file.

Style 1

Specify a `min_pulse_width` group within the pin group (in the library file).

- You should place the `timing_label_mpw_high` attribute in the `min_pulse_width` group to refer to the `constraint_high` attribute.
- You should place the `timing_label_mpw_low` attribute in the `min_pulse_width` group to refer to the `constraint_low` attribute.

Style 2

Specify the constraints in the library by using the `min_pulse_width_high` and `min_pulse_width_low` attributes in the pin group.

- You should place the `timing_label_mpw_high` attribute in the pin group to refer to the `min_pulse_width_high` attribute.
- You should place the `timing_label_mpw_low` attribute in the pin group to refer to the `min_pulse_with_low` attribute.

Accessing the min_pulse_width Constraint

To access the `min_pulse_width` values, use these functions:

- `min_rise_delay(label_high)`
  Returns the value of the `min_pulse_width_high` attribute.
- `max_rise_delay(label_high)` Same as `min_rise_delay(label_high)`.
- `min_fall_delay (label_low)`
  Returns the value of the `min_pulse_width_low` attribute.
- `max_fall_delay(label_low)` Same as `min_fall_delay(label_low)`.

For information about functions that support SDF mapping, see Table A-2 on page A-7.

Specifying the min_period Constraint on a Pin

Specify the `min_period` constraint on a pin as an attribute in the pin section. Use the `min_period(pin_name)` function to take the pin name as argument and to return the value of the `min_period` constraint.

Using SDF Mapping

To specify the SDF mapping file when writing out SDF for a design, use the `write_sdf` command with the `-map` option. To specify how bus bit names are written in the mapped SDF output, you can use the `-context` option. For example,

- `-context verilog`
  The bus bit names are printed using the %s[%d] format.
- `-context vhdl`
  The bus bit names are printed using the %s(%d) format.

  Note:
  Only names that are specified using the `bus()` function are affected. For more information, see Table A-2 on page A-7.

To write SDF with the `-context` and `-map` option, enter

```
pt_shell> write_sdf -context verilog -map example.map example.sdf
```

```
Output
File: exmaple.sdf
(DELAYFILE
 (SDFVERSION "OVI V2.1" )
 (DESIGN      "ADDER" )
 (DATE        "Tue Feb3 09:33:06 1999" )
 (VENDOR      "nyaa")
 (PROGRAM     " " )
 (VERSION     "3.24" )
 (DIVIDER     /)
 (VOLTAGE 0:0:0)
 (PROCESS "")
 (TEMPERATURE 0:0:0 )
 (TIMESCALE   1 ns)
(CELL
 (CELLTYPE "ADDER")
 (INSTANCE )
 (DELAY
  (ABSOLUTE
   (INTERCONNECT ...)
  )
 )
)
(CELL
 (CELLTYPE  "EXMP")
 (INSTANCE U15)
 (DELAY
  (ABSOLUTE
   (IOPATH A Z (1.8::4.3) (1.8::4.0))
   (IOPATH IN[0] Z (3.0::1.2) (3.0::1.0))
  )
 )
)
(CELL
 (CELLTYPE "FD02"
  (INSTANCE B1/U12/DFF)
  (DELAY
```

```
   (ABSOLUTE
    (IOPATH (posedge CP) Q(1.2::1.2) (1.5::1.5))
    (IOPATH (posedge CP) QN(1.2::1.2) (1.5::1.5))
    (IOPATH (negedge CD) Q (::) (2.1::2.4))
    (IOPATH (negedge CD) QN (2.1::2.4) (::))
   )
   (TIMINGCHECK
    (SETUP D (posedge CP)) (2.1::2.5))
    (HOLD D (posedge CP)) (1.1::1.4))
    (WIDTH (negedge CP) (5.0))
    (RECOVERY (posedge CD) (posedge CP)) (2.5))
    (PERIOD (posedge CP) (5.0))
   )

  )
 )
)
```

Note:

When SDF for cells are not present in the SDF mapping file, they are written with the default `write_sdf` format.

SDF Mapping Notation

The notation used in presenting the syntax of the SDF mapping language follows. In the following list, `item` is a symbol for a syntax construct item.

- `item?` – Item is optional in the definition (it can appear once or not at all).
- `item*` – Item can appear zero or any number of times.
- `item+` – Item can appear one or more times (but cannot be omitted).
- `KEYWORD` – Keywords are shown in uppercase bold for easy identification but are case-insensitive.
- `VARIABLE` – Is a symbol for a variable. Variable symbols are shown in uppercase for easy identification, but are case-insensitive. Some variables are defined as one of a number of discrete choices; other variables represent user data such as names and numbers.

SDF Mapping Comments

Comment lines in the SDF mapping language are in this format:

```
[white_space]* [#] [any char except new-line char]*
```

If the line has a pound sign (`#`) as the first non-whitespace character, it is treated as a comment line and is ignored.

Note:

Quoted strings can contain new-line characters in them, and can span multiple lines. Comments cannot be embedded inside a quoted string. For definitions of SDF mapping functions, see Table A-2 on page A-7.

SDF Mapping Variables

Table A-1 lists the variables used in the SDF mapping language.

Table A-1 SDF Mapping Variables

Variable	Description
SDFMAP_QSTRING	A string of any legal SDF characters and spaces, including tabs and new-line characters, optionally enclosed by double quotation marks.
SDFMAP_NUMBER	A nonnegative (zero or positive) real number, for example, 0, 1, 3.4, .7, 0.5, 2.4e-2, 3.4e2
SDFMAP_RNUMBER	A positive, zero, or negative real number, for example, 0, 1, 0.0, –3.4, .7, –0.3, 2.4e-2, 3.4e4
SDFMAP_DNUMBER	A nonnegative integer number, for example, +12, 23, 0
SDFMAP_TSVALUE	A real number followed by a unit.
SDFMAP_IDENTIFIER	The name of an object. It is case sensitive. The following characters are allowed: alphanumeric characters, the underscore character (_), and the escape character (\). If you want to use a nonalphanumeric character as a part of an identifier, you must prefix it with the escape character. White spaces are not allowed with the hierarchy divider character (/).
SDFMAP_MAP	A positive integer value preceded with the dollar sign ($), for example, $10, $3, $98. Used to specify a placeholder for a value in the mapping file.
SDFMAP_FUNCTION	A string that denotes the function name. Supported functions are given in Table A-2.

Supported SDF Mapping Functions

Table A-2 lists the functions supported when writing mapped SDF files.

Table A-2 SDF Mapping Functions

Function	Usage	Return value	Comment
`in`	`in(float)`	`float`	Natural logarithm.
`exp`	`exp(float)`	`float`	Exponentiation.
`sqrt`	`sqrt(float)`	`float`	Square root.
`max`	`max(float, float)`	`float`	Two-argument maximum function.
`min`	`min(float,float)`	`float`	Two-argument minimum function.
`bus`	`bus(string)`	`string`	Returns a transformed string by replacing the bus delimiter characters in the string with the appropriate bus delimiter characters as specified in the `write_sdf` command. The `-context verilog` option causes the string to be transformed to the %s[%d] format. The `-context vhdl` option causes the string to be transformed to the %s(%d) format.
`pin`	`pin(string)`	`string`	Returns the string argument passed to it.
`max_fall_delay`	`max_fall_delay(label)`	`float`	Maximum fall delay value associated with that timing label.
`max_fall_delay_bus`	`max_fall_delay_bus` (label, from_pin, to_pin)	`float`	Maximum fall delay value to be used when the timing label is associated with a bus arc.

Table A-2 SDF Mapping Functions (Continued)

Function	Usage	Return value	Comment
max_fall_retain_delay	max_fall_retain_delay (label, from_pin, to_pin)	float	Maximum fall delay value to be used when the timing label is associated with a retain arc.
max_rise_delay	max_rise_delay(label)	float	Maximum rise delay value associated with that timing label.
max_rise_delay_bus	max_rise_delay_bus (label, from_pin, to_pin)	float	Maximum rise delay value is used when the timing label is associated with a bus arc.
max_rise_retain_delay	max_rise_retain_delay (label, from_pin, to_pin)	float	Maximum rise delay value to be used when the timing label is associated with a retain arc.
min_fall_delay	min_fall_delay(label)	float	Minimum fall delay value associated with that timing label.
min_fall_delay_bus	min_fall_delay_bus (label, from_pin, to_pin)	float	Minimum fall delay function to be used when the timing label is associated with a bus arc.
min_fall_retain_delay	min_fall_retain_delay (label, from_pin, to_pin)	float	Minimum fall delay value to be used when the timing label is associated with a retain arc.
min_period	min_period(pin)	float	Returns the minimum period value associated with the pin name “pin” of the current cell.
min_rise_delay	min_rise_delay(label)	float	Minimum rise delay value associated with that timing label.
min_rise_delay_bus	min_rise_delay_bus (label, from_pin, to_pin)	float	Minimum rise delay value to be used when the timing label is associated with a bus arc.
min_rise_retain_delay	min_rise_retain_delay (label, from_pin, to_pin)	float	Minimum rise delay value to be used when the timing label is associated with a retain arc.

SDF Mapping File Syntax

The following example gives the complete syntax of a mapped SDF file.

```
mapping_file     ::=  map_header cell_map+

map_header       ::=  sdf_version sdf_map_name?
hierarchy_divider?
                          bus_delimiter?

sdf_map_name                    ::=  $SDF_MAP_NAME
SDFMAP_QSTRING
sdf_version            ::=  $SDF_VERSION SDFMAP_QSTRING
hierarchy_divider      ::=  $SDF_HIERARCHY_DIVIDER SDFMAP_HCHAR
bus_delimiter          ::=  $SDF_BUSBIT SDFMAP_QSTRING

cell_map               ::=  $SDF_CELL cell_name format_string
var_map
                            $SDF_CELL_END
cell_name                        ::=  SDFMAP_QSTRING
format_string          ::=  SDFMAP_QSTRING

var_map                ::= var_map_line+
var_map_line:          := SDFMAP_MAP expression

expression             ::= expression binary_operator expression
                       ||= unary_operator expression
                       ||= expression ? expression : expression
                       ||= ( expression )
                       ||= function_call
                       ||= SDFMAP_NUMBER

function_call          ::= SDFMAP_FUNCTION ( func_args? )
func_args              ::= func_arg
                       ||= func_args , func_args
func_arg               ::= SDFMAP_IDENTIFIER
                       ||= expression

binary_operator        ::= +
                       ||= -
                       ||= *
                       ||= /
                       ||= &
                       ||= |
                       ||= >
                       ||= <
                       ||= =
binary_operator        ::= !
                       ||= -
```

SDF Mapping Assumptions

The SDF mapping file reader assumes that the SDF format string read from the SDF mapping file has valid SDF syntax. For instance, it assumes that when the proper substitutions are made for the placeholders, the resulting SDF is syntactically (and also semantically) correct. The SDF map parser does not attempt to parse the format string to check for possible user errors. It performs the placeholder substitutions and prints the resulting string to the SDF output file.

To check the validity of the generated mapped SDF file, you can read the mapped file by using the `read_sdf` command. For example,

```
pt_shell> read_sdf -syntax_only mapped.sdf
```

Bus Naming Conventions

When you specify mapping names with the `bus(string)` function, the SDF writer replaces the bus bit delimiting characters (specified by the construct `$SDF_BUSBIT QSTRING`) by using the appropriate delimiting characters.

For example, suppose the following lines are in the mapping file:

```
$SDF_BUSBIT "<>"
....
$23 bus(output_bus<5>)
```

When you specify the `write_sdf` command with the `-context verilog` option, the SDF writer prints the name as `output_bus[5]`. For more information, see “Using SDF Mapping” on page A-3. If more than one set of matching bus delimiters is found in a name, the SDF writer replaces only the matching set of delimiters at the end of the name string. If you do not specify bus delimiters, the names are printed unchanged.

Header Consistency Check for SDF Mapping Files

Each SDF mapping file defines SDF version and SDF bus delimiter characters in its header. When PrimeTime reads multiple SDF mapping files during one `write_sdf` command, it assumes all the headers have the same SDF version and bus delimiter characters.

Labeling Bus Arcs

A pin in the library file can be a bus. In the following example, the timing group on the pin defines multiple arcs.

Example 1

```
bus(A) {
    bus_type  : bus_11;
    direction : input ;
timing() {
      related_pin : "CK" ;
      timing_type : setup_rising ;
         timing_label : "tas";
      intrinsic_rise : 1.12000 ;
      intrinsic_fall : 1.12000 ;
```

In this example (assuming A is an 11-bit bus), 11 setup arcs are defined between bus A and CK. When you label such an arc, since only one timing label is present, PrimeTime attaches the same label (tas) to all arcs defined by the statement. When you access individual bit arcs, you should use the bus versions of the max/min rise/fall delay functions. For example, to refer to the arc corresponding to A[5], which is from CK to A[5], use the mapping functions shown in the following examples.

Example 2

```
max_rise_delay_bus(tas, CK, A[5]) #To get the max rise delay
min_rise_delay_bus(tas, CK, A[5]) #To get the min rise delay
max_fall_delay_bus(tas, CK, A[5]) #To get the max fall delay
min_fall_delay_bus(tas, CK, A[5]) #To get the min fall delay
```

Example 3

```
bus (In)    {
  bus_type : "bus8";
  direction   :   input;
  capacitance :   1.46;
  fanout_load :   1.46;
}
bus(Q){
  bus_type : "bus8";
  direction   :   output;
  timing () {
    timing_label     : “In_to_Q”
      timing_sense     : non_unate;
      intrinsic_rise   : 2.251622;
      rise_resistance  : 0.020878;
      intrinsic_fall   : 2.571993;
      fall_resistance  : 0.017073;
      related_pin      : "In";
   }
}
```

To refer to the arc from In[6] to Q[6] and print the following SDF line:

```
(IOPATH In[6] Q[6] (1.8::4.3) (1.8::4.0))
```

The SDF mapping file format string should look like:

```
(IOPATH $1 $2 ($3::$4) ($5::$6))
```

The resulting SDF function mapping is as follows:

```
$1 bus(I1[6])
$2 bus(Q[6])
$3 min_rise_delay_bus(In_to_Q, In[6], Q[6])
$4 max_rise_delay_bus(In_to_Q, In[6], Q[6])
$5 min_fall_delay_bus(In_to_Q, In[6], Q[6])
$6 max_fall_delay_bus(In_to_Q, In[6], Q[6])
```

Note:

Avoid using a bus arc in nonbus max/min rise/fall delay functions.

SDF Mapping Limitations

The SDF mapping file syntax has the following limitations:

- You cannot use wildcard characters.
- You cannot specify instance-specific mapping format. An SDF mapping format must be specified for a library cell. The format is applied when writing SDF for every cell that is an instance of that library cell.
- If the SDF mapping file does not provide a format for every cell present in the library, PrimeTime prints these cells using the default format of the `write_sdf` command.

Mapped SDF File Examples

The following examples are different types of mapped SDF files.

Library File for Cells EXMP and FF1

A complete SDF mapping example for cells EXMP and FF1 is shown in this section. The `min_pulse_width` arcs are labeled by using Style 2.

The following is an example of a library file, example.lib.

```
library(example) {define("timing_label_mpw_low", "pin",
"string");
define("timing_label", "timing", "string");
define("timing_label_mpw_low", "pin", "string");


/* If using style #1 to specify min_pulse_width info
          */
/* define(“timing_label_mpw_low”, “min_pulse_width”,
“string”);
```

```
*/
/* define("timing_label_mpw_high", "min_pulse_width",
"string");*/

type(two_bit) {
    base_type : array;
    data_type : bit;
    bit_width : 2;
    bit_from : 1;
    bit_to : 0;
    downto : true;
  }
cell (EXMP) {
  version :   1.00;
  area    :   1.000000;
  cell_footprint : "EXMP";
    bus(IN) {
    bus_type : two_bit;
      direction : input;
      pin(IN[1:0]) {
        capacitance : 1;
      }
    }

    pin (A)    {
      direction   :   input;
      capacitance :   0.49;
      fanout_load :   0.49;
      max_transition : 4.500000;
   }
   pin (B)    {
      direction   :   input;
      capacitance :   0.51;
      fanout_load :   0.51;
      max_transition : 4.500000;
   }
   pin(Y){
      direction   :   output;
      capacitance :   0.00;
      function : "(A & B & IN[0] & IN[1])";
      timing () {
       related_pin : "A";
       timing_label : "A_Z";
       rise_transition (transitionDelay){ ...}
       fall_transition (transitionDelay){...}
       rise_propagation (propDelay){...}
       fall_propagation (propDelay){...}
      }
      timing () {
       related_pin : "B";
       timing_label : "B_Z";
       rise_transition (transitionDelay){...}
       fall_transition (transitionDelay){...}
```

```
        rise_propagation (propDelay){...}
        fall_propagation (propDelay) { ...}
       }
       timing () {
        related_pin : "IN[0]";
        timing_label : "IN[0]_Z";
        rise_transition (transitionDelay){...}
        fall_transition (transitionDelay)...}{
        rise_propagation (propDelay){...}
        fall_propagation (propDelay){...}
       }
       timing () {
        related_pin : "IN[1]";
        timing_label : "IN[1]_Z";
        rise_transition (transitionDelay){...}
        fall_transition (transitionDelay){...}
        rise_propagation (propDelay){...}
        fall_propagation (propDelay){...}
       }
       ...
    }

cell(FD2) {
  area : 9;
  pin(D) {
    direction : input;
    capacitance : 1;
    timing() {
         timing_label : "setup_D";
      timing_type : setup_rising;
      intrinsic_rise : 0.85;
      intrinsic_fall : 0.85;
      related_pin : "CP";
    }
    timing() {
         timing_label : "hold_D";
      timing_type : hold_rising;
      intrinsic_rise : 0.4;
      intrinsic_fall : 0.4;
      related_pin : "CP";
    }
  }
  pin(CP) {
    direction : input;
    capacitance : 1;
    min_period : 5.0
    min_pulse_width_high: 1.5
    min_pulse_width_low: 1.5
    timing_label_mpw_low: "min_pulse_low_CP"
    timing_label_mpw_high: "min_pulse_high_CP"

 }
  pin(CD) {
```

```
    direction : input;
    capacitance : 2;
    timing() {
         timing_label : “recovery_rise_CD”;
      timing_type : recovery_rising;
      intrinsic_rise : 0.5;
      related_pin : "CP";
    }
  }
....
pin(Q) {
    direction : output;
    function : "IQ";
    internal_node : "Q";
    timing() {
         timing_label : “CP_Q”;
      timing_type : rising_edge;
      intrinsic_rise : 1.19;
      intrinsic_fall : 1.37;
      rise_resistance : 0.1458;
      fall_resistance : 0.0523;
      related_pin : "CP";
    }
    timing() {
         timing_label : “clear_Q”;
      timing_type : clear;
      timing_sense : positive_unate;
      intrinsic_fall : 0.77;  /* CP -> Q intrinsic - 0.6 ns */
      fall_resistance : 0.0523;
      related_pin : "CD";
    }
  }
  pin(QN) {
    direction : output;
    function : "IQN";
    internal_node : "QN";
    timing() {
         timing_label : “CP_QN”;
      timing_type : rising_edge;
      intrinsic_rise : 1.47;
      intrinsic_fall : 1.67;
      rise_resistance : 0.1523;
      fall_resistance : 0.0523;
      related_pin : "CP";
    }
    timing() {
         timing_label : “preset_QN”;
      timing_type : preset;
      timing_sense : negative_unate;
      intrinsic_rise : 0.87;  /* CP -> QN intrinsic - 0.6 ns */
      rise_resistance : 0.1523;
      related_pin : "CD";
    }
```

```
   }
 }
}
```

Note:

The `min_pulse_width_low/high` constraints have been defined as attributes inside the pin group. The timing label attributes `timing_label_mpw_low` and `timing_label_mpw_high` are used to refer to these constraints.

Mapped SDF File

The following is an example of a mapped SDF file, example.map.

```
# This is a comment
$SDF_VERSION "OVI 2.1"
$SDF_BUSBIT "[]"
$SDF_CELL EXMP
"(DELAY
   (ABSOLUTE
    (IOPATH $1 $2 ($3::$4) ($5::$6))
    (IOPATH $7 $2 ($8::$9) ($10::$11))
   )
)"
$3 min_rise_delay(A_Z)
$4 max_rise_delay(A_Z)
$5 min_fall_delay(A_Z)
$6 max_fall_delay(A_Z)
$1 pin(A)
$2 pin(Z)
$7 bus(IN[0])
$8 min_rise_delay(IN[0]_Z)
$9 max_rise_delay(IN[0]_Z)
$10 min_fall_delay(IN[0]_Z)
$11 max_fall_delay(IN[0]_Z)
$SDF_CELL_END

$SDF_CELL DFF
"(DELAY
    (ABSOLUTE
     (IOPATH (posedge $1) $2 ($3::$4) ($5::$6))
     (IOPATH (posedge $7) $8 ($9::$10) ($11::$12))
     (IOPATH (negedge $13) $2 (::) ($16::$17))
     (IOPATH (negedge $13) $8($18::$19) (::))
    )
)
(TIMINGCHECK
    (SETUP $20(posedge $1)) ($30::$31))
    (HOLD $20 (posedge $1)) ($32::$33))
    (WIDTH (negedge $1) ($34))
    (RECOVERY (posedge $13) (posedge $1)) ($36))
    (PERIOD (posedge $1) ($37))
)"
```

```
$1 pin(CP)
$2 pin(Q)
$3 min_rise_delay(CP_Q)
$4 max_rise_delay(CP_Q)
$5 min_fall_delay(CP_Q)
$6 max_fall_delay(CP_Q )
$7 pin(CP)
$8 pin(QN)
$9 min_rise_delay(CP_QN)
$10 max_rise_delay(CP_QN)
$11 min_fall_delay(CP_QN)
$12 max_fall_delay(CP_QN)
$13 pin(CD)
$16 min_fall_delay(clear_Q)
$17 max_fall_delay(clear_Q)
$18 min_rise_delay(preset_QN)
$19 max_rise_delay(preset_QN)
$20 pin(D)
$30 min_rise_delay(setup_D)
$31 max_rise_delay(setup_D)
$32 min_rise_delay(hold_D)
$33 max_rise_delay(hold_D)
$34 min_fall_delay(min_pulse_low_CP)
$36 max_rise_delay(recovery_rise_CD)
$37 min_period(CP)
```

Three-State Buffers

This section gives an example of the following files for a three-state noninverting buffer.

- Library file
- Mapped SDF file

Library File

The following is a detailed example of a library file for a three-state noninverting buffer.

```
/
*------------------------------------------------------------
Internal Tristate Non-Inverting Buffer, Positive Enable, 1x
Drive
------------------------------------------------------------*
/
define("timing_label", "timing", "string");
define(“timing_label_mpw_low”, “pin”, “string”);
define(“timing_label_mpw_low”, “pin”, “string”);

cell(BTS) {
  area : 63 ;
  pin(Z) {
    direction : output ;
```

```
    function : "A";
     max_capacitance : 0.14000 ;
     capacitance : 0.00420 ;
    three_state : "E'";
     timing() {
       related_pin : "A" ;
       timing_sense : positive_unate ;
       timing_label : "A_Z";
       cell_rise(table_1) {
         values ( ... ) ;
       }
       rise_transition(table_1) {
         values ( ... ) ;
       }
       cell_fall(table_1) {
         values ( ... ) ;
       }
       fall_transition(table_1) {
         values ( ... ) ;
       }
       intrinsic_rise : 0.31166 ;
       intrinsic_fall : 0.40353 ;
     }
     timing() {
       related_pin : "E" ;
       timing_label : "E_Z_3s_enable";
       cell_rise(table_1) {
         values ( ... ) ;
       }
       rise_transition(table_1) {
         values ( ... ) ;
       }
       cell_fall(table_1) {
         values ( ... ) ;
       }
       fall_transition(table_1) {
         values ( ... ) ;
       }
       intrinsic_rise : 0.22159 ;
       intrinsic_fall : 0.27933 ;
     }
     timing() {
       related_pin : "E" ;
timing_type : three_state_disable ;
       timing_label : "E_Z_3s_disable";
       cell_rise(table_10) {
         values ( ... ) ;
       }
       rise_transition(table_10) {
         values ( ... ) ;
       }
       cell_fall(table_10) {
         values ( ... ) ;
```

```
      }
      fall_transition(table_10) {
        values ( ... ) ;
      }
      intrinsic_rise : 0.30693 ;
      intrinsic_fall : 0.19860 ;
    }
  }
  pin(A) {
    direction : input ;
    capacitance : 0.00423 ;
  }
  pin(E) {
    direction : input ;
    capacitance : 0.01035 ;
  }
}
```

Mapped SDF File

The following is an example of a mapped SDF file.

```
$SDF_VERSION "OVI 2.1"
$SDF_BUSBIT "[]"
$SDF_CELL BTS
 "(DELAY
    (ABSOLUTE
      (IOPATH $1 $3 ($4::$5) ($6::$7))
      (IOPATH $2 $3 ($8::$9) ($10::$11) ($12::$13) ($8::$9)
($14::$15) ($10::$11))
  )"
####
  $1 pin(A)
  $2 pin(E)
  $3 pin(Z)
####
  $4 min_rise_delay(A_Z)
  $5 max_rise_delay(A_Z)
  $6 min_fall_delay(A_Z)
  $7 max_fall_delay(A_Z)
#####
  $8  min_rise_delay(E_Z_3s_enable)
  $9  max_rise_delay(E_Z_3s_enable)
  $10 min_fall_delay(E_Z_3s_enable)
  $11 max_fall_delay(E_Z_3s_enable)
  $12 min_rise_delay(E_Z_3s_disable)
  $13 max_rise_delay(E_Z_3s_disable)
  $14 min_fall_delay(E_Z_3s_disable)
  $15 max_fall_delay(E_Z_3s_disable)
####
```

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W