PrimeTime® SI
User Guide

Version D-2009.12, December 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, 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, CustomExplorer, 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, StarRC, 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

5. Multiple Supply Voltage Analysis

6. Static Noise Analysis

Appendix C. SPEF Warning Messages

Index

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 SI Release Notes* in SolvNet.

To see the *PrimeTime SI 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, and then select a release in the list that appears.

About This User Guide

The *PrimeTime SI User Guide* describes the features and usage flow of PrimeTime SI, an optional tool that adds crosstalk timing and crosstalk noise analysis capabilities to PrimeTime.

Audience

This user guide is for engineers who use PrimeTime for static timing analysis and PrimeTime SI for crosstalk analysis. Users should already be familiar with PrimeTime and have some familiarity with crosstalk and signal integrity principles.

Related Publications

For additional information about PrimeTime SI, 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 and PrimeTime VX
- StarRC
- Library Compiler
- Liberty 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

PrimeTime SI Overview

PrimeTime SI (signal integrity) is an optional tool that adds crosstalk analysis capabilities to the PrimeTime static timing analyzer. PrimeTime SI calculates the timing effects of cross-coupled capacitors between nets and includes the resulting delay changes in the PrimeTime analysis reports. It also calculates the logic effects of crosstalk noise and reports conditions that could lead to functional failure.

This overview chapter includes the following sections:

Signal Integrity and Crosstalk

Signal integrity is the ability of an electrical signal to carry information reliably and resist the effects of high-frequency electromagnetic interference from nearby signals. Crosstalk is the undesirable electrical interaction between two or more physically adjacent nets due to capacitive cross-coupling. As integrated circuit technologies advance toward smaller geometries, crosstalk effects become increasingly important compared to cell delays and net delays. The reasons for this are apparent in Figure 1-1.

Figure 1-1 Net Capacitance With Different Feature Sizes

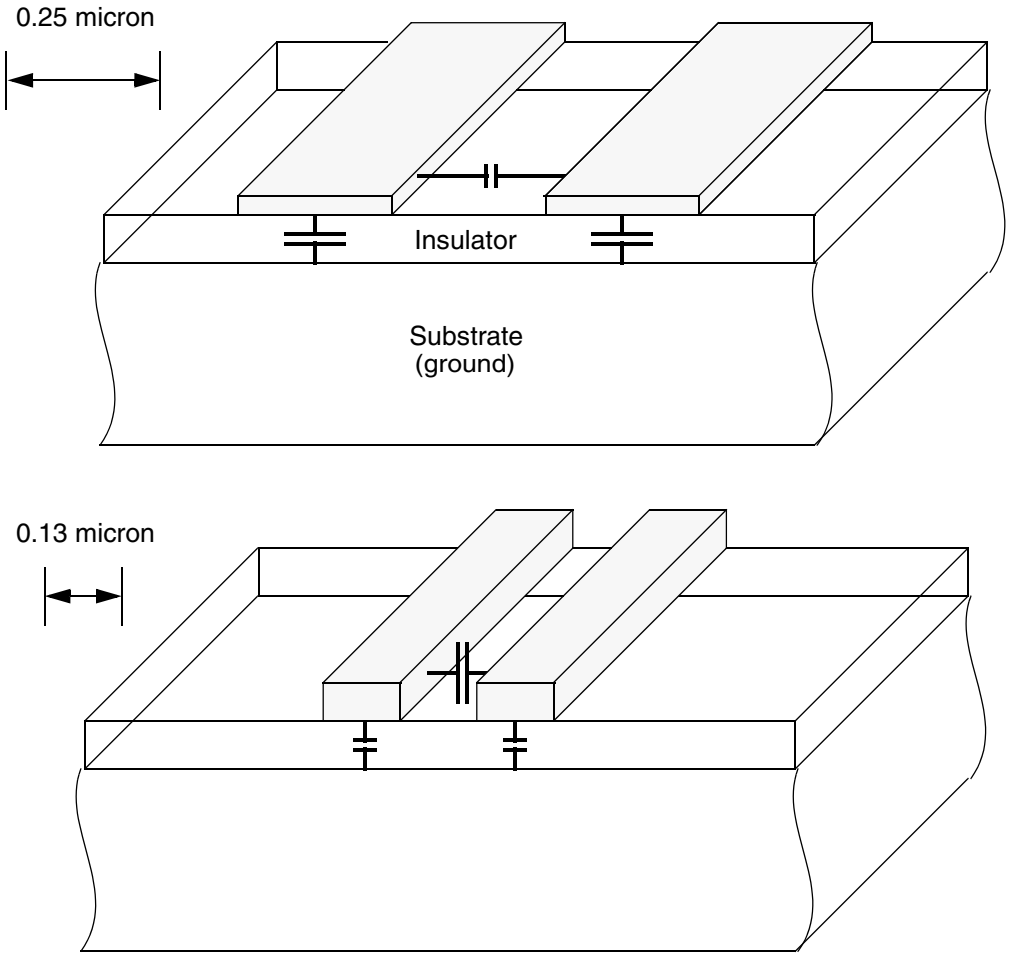


Figure 1-1 shows an enlarged view of two parallel metal interconnections in an integrated circuit, first for a 0.25-micron technology and then for a 0.13-micron technology.

As circuit geometries become smaller, wire interconnections become closer together and taller, thus increasing the cross-coupling capacitance between nets. At the same time, parasitic capacitance to the substrate becomes less as interconnections become narrower, and cell delays are reduced as transistors become smaller.

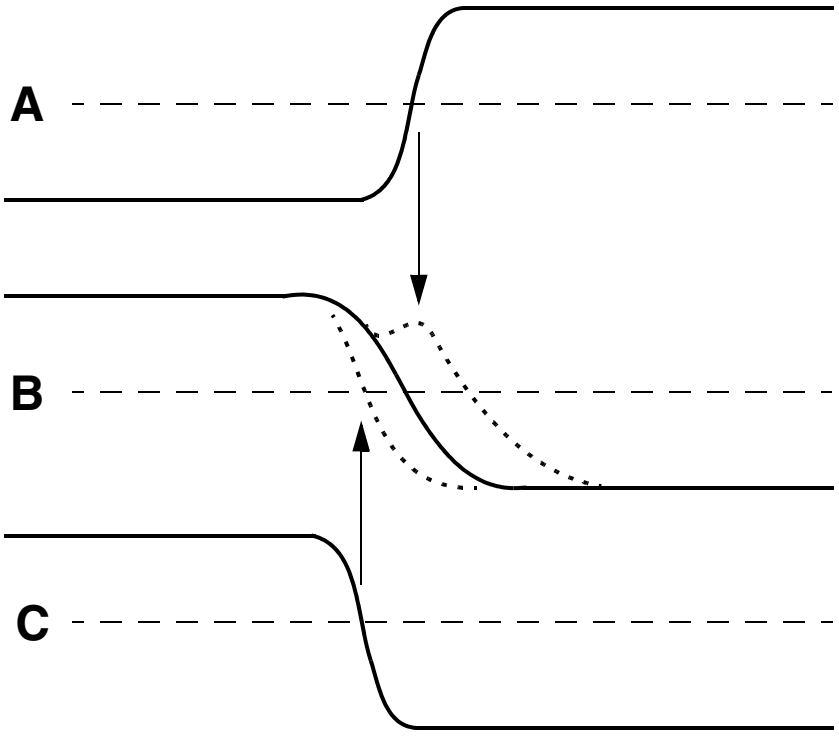
With circuit geometries at 0.25 micron and above, substrate capacitance is usually the dominant effect. However, with geometries at 0.18 micron and below, the coupling capacitance between nets becomes significant, making crosstalk analysis increasingly important for accurate timing analysis.

PrimeTime SI has the ability to analyze and report on two major types of crosstalk effects: delay changes and static noise.

Crosstalk Delay Effects

Crosstalk can affect signal delays by changing the times at which signal transitions occur. For example, consider the signal waveforms on the cross-coupled nets A, B, and C in Figure 1-2.

Figure 1-2 Transition Slowdown or Speedup Caused by Crosstalk



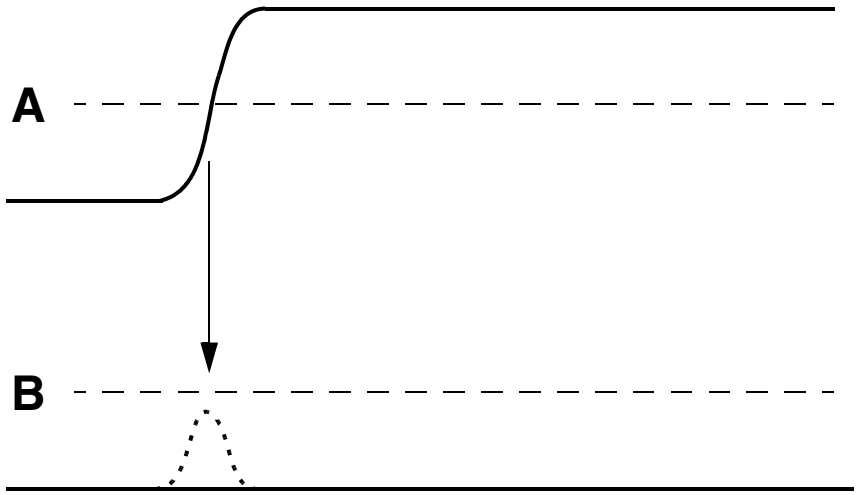
Because of capacitive cross-coupling, the transitions on net A and net C can affect the time at which the transition occurs on net B. A rising-edge transition on net A at the time shown in Figure 1-2 can cause the transition to occur later on net B, possibly contributing to a setup violation for a path containing B. Similarly, a falling-edge transition on net C can cause the transition to occur earlier on net B, possibly contributing to a hold violation for a path containing B.

PrimeTime SI determines the worst-case changes in delay values and uses this additional information to calculate and report total slack values. It also reports the locations and amounts of crosstalk delays so that you can change the design or the layout to reduce crosstalk effects at critical points.

Crosstalk Noise Effects

PrimeTime SI also determines the logic effects of crosstalk on steady-state nets. For example, consider the signal waveforms on the cross-coupled nets A and B in Figure 1-3.

Figure 1-3 Glitch Due to Crosstalk



Net B should be constant at logic zero, but the rising edge on net A causes a noise bump or glitch on net B. If the bump is sufficiently large and wide, it can cause an incorrect logic value to be propagated to the next gate in the path containing net B.

PrimeTime SI considers these effects and determines where crosstalk noise bumps have the largest effects. It reports the locations of potential problems so that they can be fixed.

You can choose to have PrimeTime calculate crosstalk delay effects only, crosstalk noise effects only, or both.

Aggressor and Victim Nets

A net that receives undesirable cross-coupling effects from a nearby net is called a victim net. A net that causes these effects in a victim net is called an aggressor net. Note that an aggressor net can itself be a victim net; and a victim net can also be an aggressor net. The terms aggressor and victim refer to the relationship between two nets being analyzed.

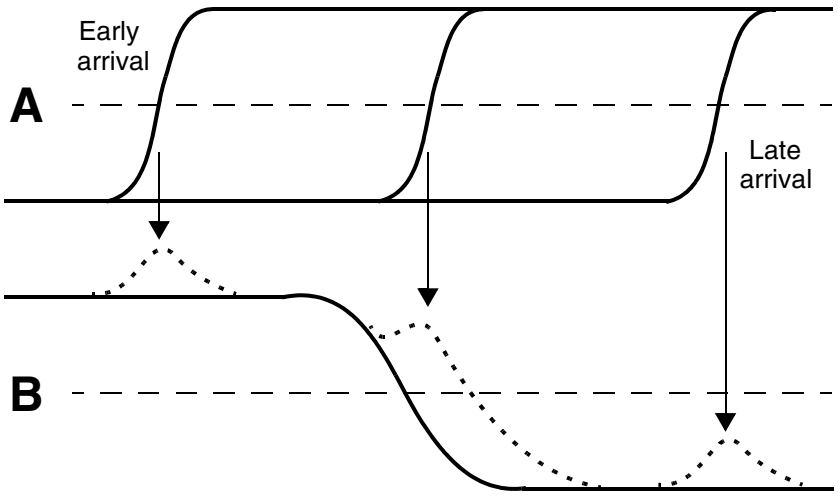
The timing impact of an aggressor net on a victim net depends on several factors:

- The amount of cross-coupled capacitance
- The relative times and slew rates of the signal transitions
- The switching directions (rising, falling)
- The combination of effects from multiple aggressor nets on a single victim net

PrimeTime SI takes all of these factors into account when it calculates crosstalk effects. It saves a lot of computation time by ignoring situations where the cross-coupling capacitors are too small to have an effect and by ignoring cases where the transition times on cross-coupled nets cannot overlap.

Figure 1-4 illustrates the importance of timing considerations for calculating crosstalk effects. The aggressor signal A has a range of possible arrival times, from early to late.

Figure 1-4 Effects of Crosstalk at Different Arrival Times



If the transition on A occurs at about the same time as the transition on B, it could cause the transition on B to occur later as shown in the figure, possibly contributing to a setup violation; or it could cause the transition to occur earlier, possibly contributing to a hold violation.

If the transition on A occurs at an early time, it induces an upward bump or glitch on net B before the transition on B, which has no effect on the timing of signal B. However, a sufficiently large bump can cause unintended current flow by forward-biasing a pass transistor. PrimeTime SI reports the worst-case occurrences of noise bumps.

Similarly, if the transition on A occurs at a late time, it induces a bump on B after the transition on B, also with no effect on the timing of signal B. However, a sufficiently large bump can cause a change in the logic value of the net, which can be propagated down the timing path. PrimeTime SI reports occurrences of bumps that cause incorrect logic values to be propagated.

Timing Windows and Crosstalk Delay Analysis

PrimeTime offers three analysis modes with respect to operating conditions: single, best case/worst case, and on-chip variation. PrimeTime SI uses the on-chip variation mode to derive the timing window relationships between aggressor nets and victim nets.

Using the on-chip variation mode, PrimeTime SI finds the earliest and the latest arrival times for each victim net and aggressor net. The range of switching times, from earliest to latest arrival, defines a timing window for the victim net, and defines another timing window for the aggressor net. Crosstalk timing effects can occur only when the victim and aggressor timing windows overlap.

PrimeTime SI performs crosstalk analysis using multiple iterations. For the first iteration, it ignores the timing windows and assumes that all transitions can occur at any time. This results in a pessimistic but fast analysis that gives approximate crosstalk delay values. In subsequent analysis iterations, PrimeTime SI considers the timing windows and eliminates some victim-aggressor relationships from consideration, based on the lack of overlap between the applicable timing windows.

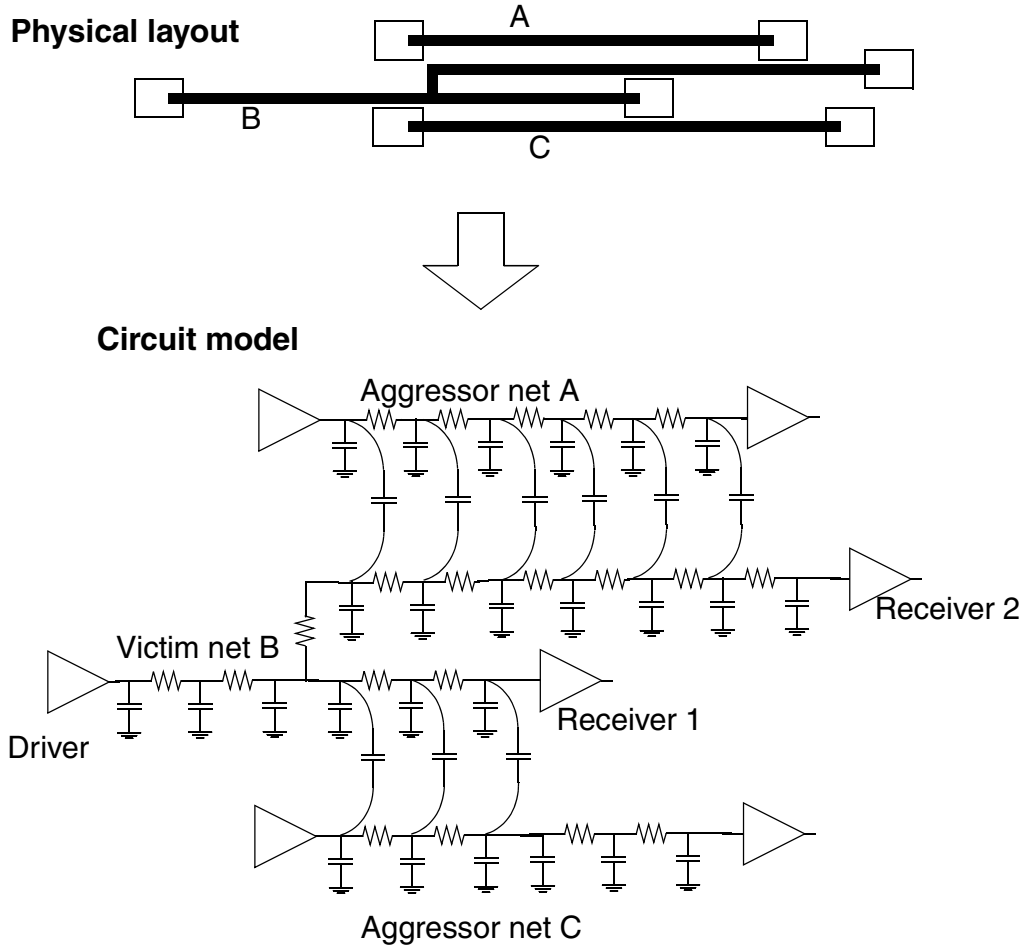
When an overlap occurs, PrimeTime SI calculates the effect of a transition occurring on the aggressor net at the same time as a transition on the victim net. The analysis takes into account the drive strengths and coupling characteristics of the two nets.

Cross-Coupling Models

Figure 1-5 shows the physical layout for a small portion of an integrated circuit, together with a detailed model of the circuit that includes cross-coupled capacitance. Each physical interconnection in the design has some distributed resistance along the conductor and

some parasitic capacitance to the substrate (ground) and to adjacent nets. The model divides each net into subnets and represents the distributed resistance and capacitance as a set of discrete resistors and capacitors.

Figure 1-5 Detailed Model of Cross-Coupled Nets



A detailed model such as this can provide a very accurate prediction of crosstalk effects in simulation. For an actual integrated circuit, however, a model might have too many circuit elements to process in a practical amount of time. Given a reasonably accurate (but sufficiently simple) network of cross-coupled capacitors from an external tool, PrimeTime SI can obtain accurate crosstalk analysis results in a reasonable amount of time.

Summary of PrimeTime SI Features

PrimeTime SI is an optional enhancement to PrimeTime that adds crosstalk analysis capabilities to the PrimeTime static timing analysis engine. You need a PrimeTime SI license in order to use its crosstalk analysis features.

If you need to account for possible timing effects of crosstalk, using PrimeTime SI is much easier, faster, and more thorough than using a circuit simulator such as SPICE. Instead of analyzing just a single path or a few paths for crosstalk effects, PrimeTime SI lets you analyze the whole circuit using the familiar PrimeTime analysis flow.

To use PrimeTime SI with PrimeTime, you only need to do the following additional steps:

- Enable PrimeTime SI by setting the `si_enable_analysis` variable to true.
- Back-annotate the design with cross-coupling capacitor information, as specified in a Standard Parasitic Exchange Format (SPEF) or Detailed Standard Parasitic Format (DSPF) file.
- Specify the parameters that determine the accuracy and speed of the crosstalk analysis effort, such as the number of analysis iterations and the capacitance values that can be safely ignored.

PrimeTime SI calculates the possible effects of crosstalk on signal delays and includes that information in all reports generated by the `report_timing` command and other timing analysis commands.

Manual Organization

This manual, the *PrimeTime SI User Guide*, provides detailed instructions on using PrimeTime SI.

Chapter 2, “Using PrimeTime SI,” concisely explains the usage flow and commands for doing static timing analysis with PrimeTime SI.

Chapter 3, “Graphical User Interface,” explains how to use the PrimeTime SI dialog boxes to set up the analysis and how to view the analysis results in histogram form.

Chapter 4, “Net Selection and Analysis Exit,” describes how to specify the nets that PrimeTime SI initially considers for crosstalk analysis, and how PrimeTime SI reselects nets for each crosstalk analysis iteration, and how to specify the parameters that control the termination of crosstalk analysis iterations.

Chapter 5, “Multiple Supply Voltage Analysis,” describes how to analyze designs with different power supply voltages for different cells.

Chapter 6, “Static Noise Analysis,” describes how to analyze designs for crosstalk noise effects and how to interpret reports on worst-case noise bumps, noise slack, and propagated noise.

Appendix A, “Crosstalk Attributes,” describes the crosstalk attributes that you can extract from the design using Tcl programs. You can use the extracted attribute information to debug crosstalk problems.

Appendix B, “SPICE Analysis,” explains how to have PrimeTime SI generate a SPICE deck to represent a set of critical paths, which you can then analyze with an external SPICE simulator.

Appendix C, “SPEF Warning Messages,” lists and describes the error messages that PrimeTime SI can return when you read in a Standard Parasitic Exchange Format (SPEF) file containing errors.

2

Using PrimeTime SI

Adding crosstalk analysis to the regular PrimeTime analysis flow involves enabling crosstalk analysis, providing cross-coupling capacitor information, and specifying the parameters that control the accuracy and speed of the analysis.

This chapter provides information on the following topics:

Usage Flow

Static timing analysis with PrimeTime SI uses the same command set, libraries, and Tcl scripts as ordinary PrimeTime analysis. You only need to enable crosstalk analysis, provide the cross-coupled capacitance information, and optionally specify the parameters that control the speed and performance of the crosstalk portion of the analysis. The addition of crosstalk analysis does not affect the speed or accuracy of the remaining portion of the timing analysis.

Here is an example of a script that uses crosstalk analysis, with the crosstalk-specific items shown in boldface:

```
set_operating_conditions -analysis_type on_chip_variation
set si_enable_analysis TRUE
read_db ./test1.db
current_design test1
link_design
read_parasitics -keep_capacitive_coupling SPEF.spf
create_clock -period 5.0 clock
check_timing -include { no_driving_cell ideal_clocks \
  partial_input_delay unexpandable_clocks }
report_timing
report_si_bottleneck
report_delay_calculation -crosstalk -from pin -to pin
```

The `set_operating_conditions` command sets the analysis type to `on_chip_variation`, which is necessary to allow PrimeTime SI to correctly handle the min-max timing window relationships. If you do not specify this analysis type explicitly, PrimeTime SI automatically switches to that mode.

Setting `si_enable_analysis` to true enables crosstalk analysis using PrimeTime SI.

The `-keep_capacitive_coupling` option in the `read_parasitics` command is necessary to maintain the cross-coupling status of capacitors that have been read into PrimeTime.

In addition to IEEE 1481 Standard Parasitic Exchange Format (SPEF), PrimeTime SI can also read parasitic data in Synopsys Binary Parasitic Format (SBPF) or Detailed Standard Parasitic Format (DSPF). To read data in this format, use a command similar to the following:

```
pt_shell> read_parasitics -keep_capacitive_coupling \
              -format SBPF mydata.sbpf
```

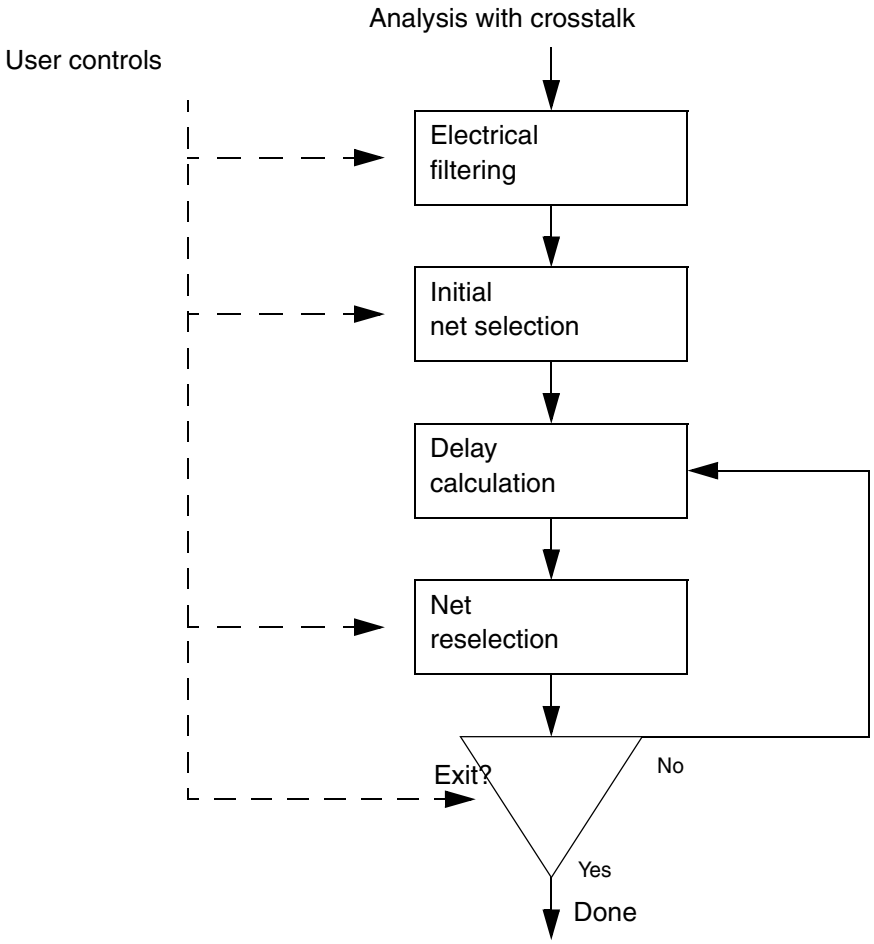
You might want to set some of the parameters that control the accuracy and speed characteristics of the crosstalk analysis, as explained later in this chapter. These parameters are controlled by a set of variables.

If significant crosstalk effects are apparent in the timing report, you will probably want to do some analysis to debug or correct the timing problem. To assist in this task, the PrimeTime SI graphical user interface lets you generate histograms of crosstalk delays and induced bump voltages. You can also create your own Tcl scripts to extract the crosstalk attributes from the design database, and then generate your own custom reports or histograms. The attributes you can examine are described in Appendix A, "Crosstalk Attributes."

How PrimeTime SI Operates

PrimeTime SI performs crosstalk analysis in conjunction with your regular PrimeTime analysis flow. With crosstalk analysis enabled, when you update the timing (for example, by using `update_timing` or `report_timing`), PrimeTime SI performs the steps shown in Figure 2-1.

Figure 2-1 PrimeTime SI Crosstalk Analysis Flow



The first step is called electrical filtering. This means removing from consideration the aggressor nets whose effects are too small to be significant, based on the calculated sizes of bump voltages on the victim nets. You can specify the threshold level that determines which aggressor nets are filtered.

After filtering, PrimeTime SI selects the initial set of nets to be analyzed for crosstalk effects from those not already eliminated by filtering. You can optionally specify that certain nets be included in, or excluded from, this initial selection set.

The next step is to perform delay calculation, taking into account the crosstalk effects on the selected nets. This step is just like ordinary timing analysis, but with the addition of crosstalk considerations.

Crosstalk analysis is an iterative process, taking multiple passes through the delay calculation step, to obtain accurate results. For the initial delay calculation (using the initial set of selected nets), PrimeTime SI uses a conservative model that does not consider transition timing windows. In other words, PrimeTime SI assumes that every aggressor net can have a worst-type transition (rising or falling) at the worst possible time, causing the worst possible slowdown or speedup of transitions on the victim net. The purpose of this behavior is to quickly obtain worst-case delay values so that PrimeTime SI can intelligently select the nets for the next analysis iteration.

In the second and subsequent delay calculation iterations, PrimeTime SI considers the possible times that transitions can occur and their directions (rising or falling), and removes from consideration any crosstalk delays that can never occur, based on the separation in time between the aggressor and victim transitions or the direction of the aggressor transition. The result is a more accurate, less pessimistic analysis of worst-case effects.

After each delay calculation iteration, PrimeTime SI selects a new set of nets for analysis in the next iteration, based on the results of the analysis just completed. This process is called net reselection. By default, PrimeTime SI selects only the nets in the critical path. You can optionally specify different reselection criteria, such as the amount of slack in the timing paths or the size of the crosstalk effect (delta delay).

After net reselection, PrimeTime SI determines whether it has performed enough iterations, based on the exit criteria. If so, it exits from the analysis loop and generates a timing report. Otherwise, it continues with the next delay calculation iteration.

By default, PrimeTime SI exits from the loop upon completion of the second iteration, which typically provides good results in a reasonable amount of time. However, you can optionally specify other types of exit criteria to get more iterations and obtain more accurate results, at the expense of additional runtime.

You can interrupt any crosstalk analysis iteration by pressing Control-c. PrimeTime SI finishes the current iteration, exits from the loop, and reports diagnostic information regarding the state of the analysis at the end of the iteration.

PrimeTime SI Variables

Several PrimeTime SI variables are available to control the crosstalk analysis parameters. Each variable has a default setting. To override the default, you can use the `set` command. For example, to set the variable that enables crosstalk analysis:

```
pt_shell> set si_enable_analysis true
```

Table 2-1 lists the PrimeTime SI variables and their default settings. The variables are described elsewhere in the manual and in the PrimeTime SI man pages.

Table 2-1 PrimeTime SI Variables

Variable	Default setting
si_analysis_logical_correlation_mode	true
si_ccs_aggressor_alignment_mode	lookahead
si_ccs_use_gate_level_simulation	true
si_enable_analysis	false
si_filter_accum_aggr_noise_peak_ratio	0.03
si_filter_per_aggr_noise_peak_ratio	0.01
si_ilm_keep_si_user_excluded_aggressors	false
si_noise_composite_aggr_mode	disabled
si_noise_endpoint_height_threshold_ratio	0.75
si_noise_limit_propagation_ratio	0.75
si_noise_slack_skip_disabled_arcs	false
si_noise_update_status_level	none
si_use_driving_cell_derate_for_delta_delay	false
si_xtalk_composite_aggr_mode	disabled
si_xtalk_composite_aggr_noise_peak_ratio	0.01
si_xtalk_composite_aggr_quantile_high_pct	99.73

Table 2-1 PrimeTime SI Variables (Continued)

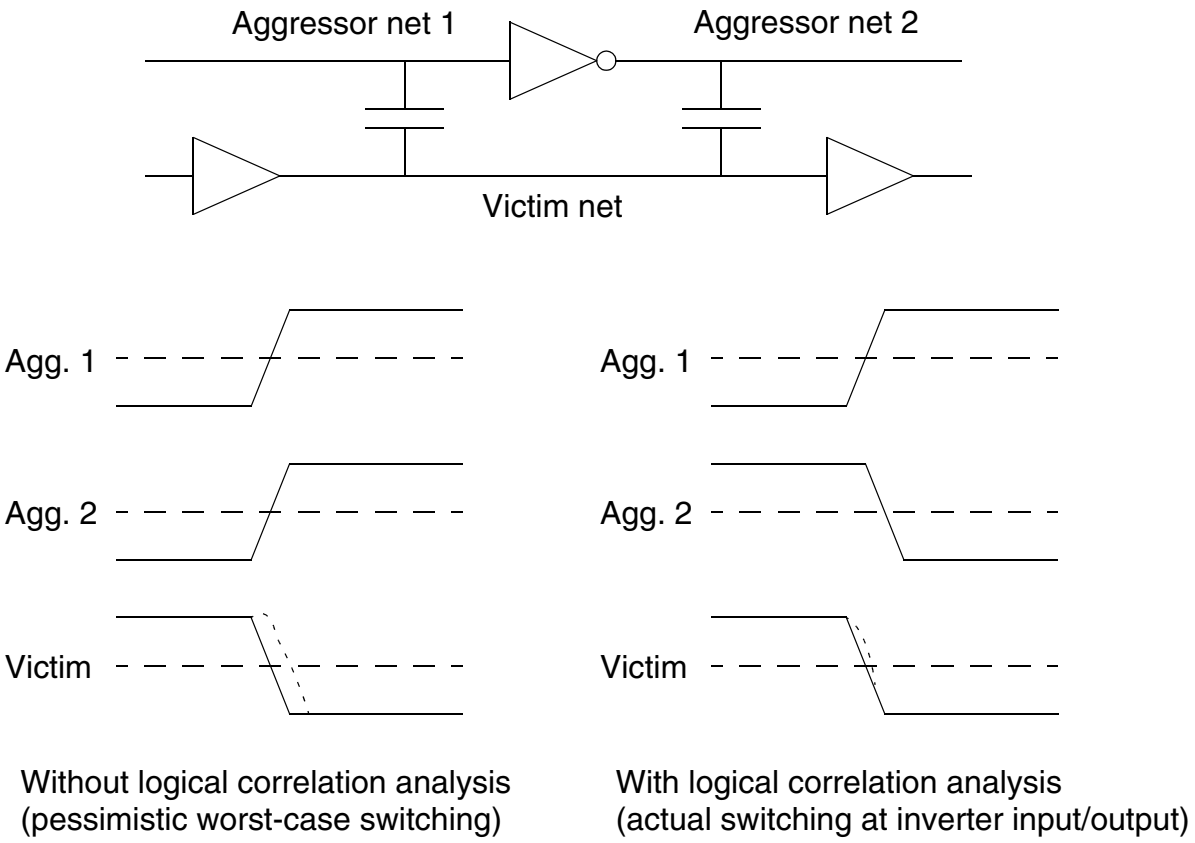
Variable	Default setting
si_xtalk_delay_analysis_mode	all_paths
si_xtalk_double_switching_mode	disabled
si_xtalk_exit_on_max_iteration_count	2
si_xtalk_exit_on_max_iteration_count_incr	2
si_xtalk_reselect_clock_network	true
si_xtalk_reselect_delta_delay	5
si_xtalk_reselect_delta_delay_ratio	0.95
si_xtalk_reselect_max_mode_slack	0
si_xtalk_reselect_min_mode_slack	0
si_xtalk_reselect_time_borrowing_paths	true
pba_enable_ccs_waveform_propagation	false

Logical Correlation

In a conservative analysis, the analysis tool assumes that all aggressor nets can switch together in a direction to cause a worst-case slowdown or speedup of a transition on the victim net. In some cases, due to a logical relationship between the signals, the aggressor nets cannot actually switch together in the same direction. Figure 2-2 shows an example of this situation.

By default, PrimeTime SI considers the logical relationships between multiple aggressor nets where buffers and inverters are used, thus providing a more accurate (less pessimistic) analysis of multiple aggressor nets. Taking into account the logical correlation between different nets requires some CPU resources.

Figure 2-2 Logical Correlation



For a faster but more pessimistic analysis, you can disable logical correlation consideration. To do so, set the variable `si_analysis_ logical_correlation_mode` to false. This variable is set to true by default.

Electrical Filtering

In order to achieve accurate results in a reasonable amount of time, PrimeTime SI filters (removes from consideration) aggressor nets that are considered to have too small an effect on the final results. When filtering occurs, the aggressor net and the coupling capacitors connect to it are not considered for analysis between that victim net and aggressor net. If the bump height contribution of an aggressor on its victim net is very small (less than 0.00001 of the victim's nominal voltage), this aggressor is automatically filtered.

Filtering eliminates aggressors based on the size of the voltage bump induced on the victim net by the aggressor net. The bump sizes depend on the cross-coupling capacitance values, drive strengths, and resistance values in the nets. An aggressor net is filtered if the peak voltage of the noise bump induced on the victim net, divided by Vdd (the power supply voltage), is less than the value specified by this variable:

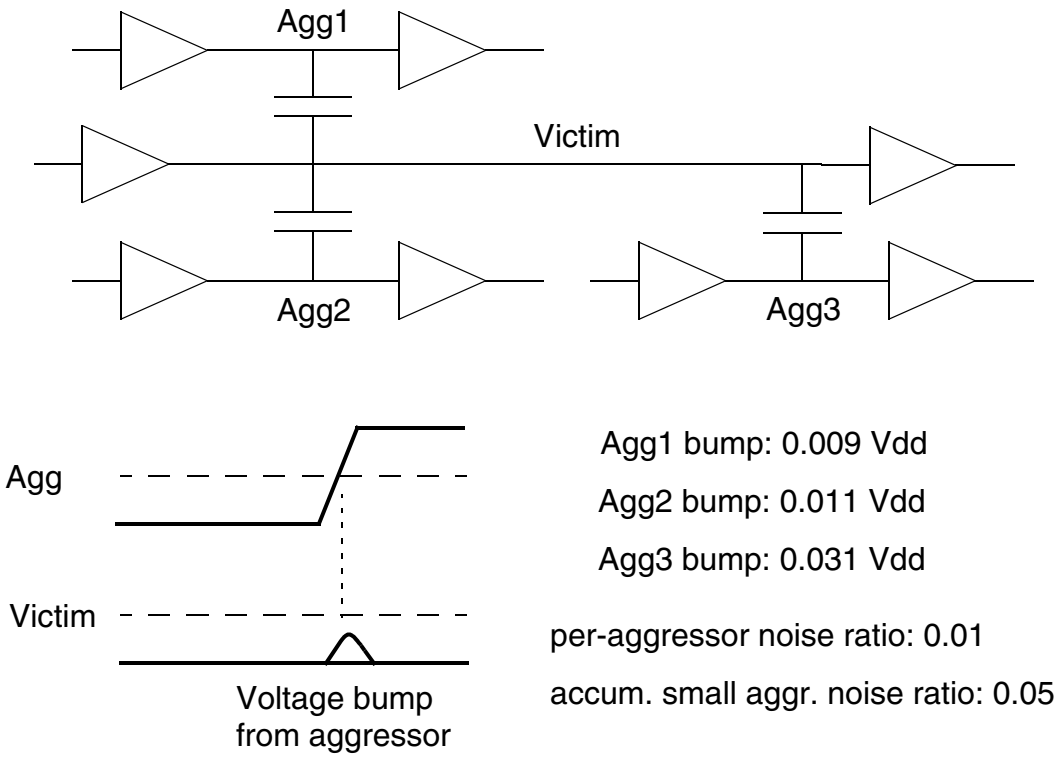
`si_filter_per_aggr_noise_peak_ratio`

By default, this variable is set to 0.01.

In addition, if a combination of smaller aggressors is below a different, larger threshold, all of those smaller aggressors are filtered. This threshold is set by the variable `si_filter_accum_aggr_noise_peak_ratio`. If the combined height of smaller noise bumps, divided by Vdd, is less than this variable setting, all of those aggressors are removed from consideration for that set of analysis conditions. The default setting for this variable is 0.03.

Figure 2-3 shows how PrimeTime SI compares voltage bump sizes for a case with three aggressor nets. It first calculates the voltage bumps induced on the victim net by transitions on each aggressor net. Then it considers the bump sizes in order. For this example, assume that `si_filter_accum_aggr_noise_peak_ratio` has been set to 0.05.

Figure 2-3 Voltage Bumps From Multiple Aggressors



PrimeTime SI filters out aggressor 1 immediately because the bump size, 0.009, is below the per-aggressor threshold, 0.01.

Then PrimeTime SI considers the combined bumps of smaller aggressors. The combination of aggressor 1 and 2 bump heights is 0.02, which is below the accumulated small aggressor threshold of 0.05, so both of those aggressors are filtered out. However, the combination of all three aggressor bump heights is 0.051, which is above the accumulated small aggressor threshold, so aggressor 3 is not filtered out, even though it is below the threshold by itself.

In summary, aggressor 1 is filtered due to the per-aggressor threshold alone, while both aggressors 1 and 2 are filtered out, but not aggressor 3, due to the accumulated small aggressor threshold.

You can set the thresholds higher for more filtering and less runtime, or lower for less filtering and increased accuracy.

Usage Guidelines

Most of the PrimeTime SI variables let you trade analysis accuracy against execution speed. For example, you can get more accuracy by running more delay calculation iterations, at the cost of more runtime. The following suggestions and guidelines will help ensure reasonable accuracy with a reasonable runtime.

Initial Analysis Without Crosstalk

First make sure that your test design is well-constrained and passes normal static timing analysis (without crosstalk analysis enabled). There should be no timing violations.

Capacitive Coupling Data

A good crosstalk analysis depends on getting an accurate and reasonably simple set of cross-coupling capacitors. Make sure that your parasitic capacitance extraction tool generates an IEEE-standard SPEF or DSPF file with coupling capacitors, and that the tool settings generate a reasonable number of capacitors.

If your extraction tool supports the filtering of small capacitors based on a threshold, it might be more efficient to let the extraction tool rather than PrimeTime SI do electrical filtering. To further trade accuracy for simplicity, you might consider limiting the number of coupling capacitors per aggressor-victim relationship.

PrimeTime SI ignores any cross-coupling capacitor between a net and itself. If possible, configure your extraction tool to suppress generation of such self-coupling capacitors.

When you read in the capacitive coupling data with the `read_parasitics` command, remember to use the `-keep_capacitive_coupling` option to retain the data.

Reading and Writing Parasitic Data

The `read_parasitics` command reads parasitic data from a file and annotates the information on the nets of the current design.

PrimeTime SI supports the reading of parasitic data in Standard Parasitic Exchange Format (SPEF), Detailed Standard Parasitic Format (DSPF), and Synopsys Binary Parasitic Format (SBPF). Data in SBPF format occupies less disk space and can be read much faster than the same data stored in the other formats.

To convert parasitic data to SBPF, follow this procedure:

1. Using the `read_parasitics` command, read the parasitic data from a file in SPEF or DSPF format. For example:

```
pt_shell> read_parasitics -keep_capacitive_coupling \
          file1.spef
```

2. Make any desired modifications to the parasitic data in the design.
3. Write the data to a file in SBPF format using the `write_parasitics` command:

```
pt_shell> write_parasitics -format SBPF file_name
```

4. You can quickly read the data back in at any time from the SBPF file using the `read_parasitics` command:

```
pt_shell> read_parasitics -format SBPF file_name
```

All capacitors are written except for net self-coupling capacitors, which are discarded.

For more information on the `read_parasitics` command, see the man page for the command.

Operating Conditions

PrimeTime SI uses on-chip variation of operating conditions to find the arrival window for each victim net and aggressor net. It automatically switches the analysis mode to on_chip_variation for crosstalk analysis if it is not already set to that mode.

These are the consequences of automatic switching to on_chip_variation mode:

- If you were already using on_chip_variation mode for non-crosstalk analysis before invoking PrimeTime SI, crosstalk analysis will continue in that mode.

- If you were using the single operating condition mode, crosstalk analysis will be done in on_chip_variation mode with the single operating condition setting for both min and max analysis. This is equivalent to using a single operating condition.
- If you were using bc_wc (best-case/worst-case) mode, crosstalk analysis will be done in on_chip_variation mode, using a mixture of the two extreme conditions for both min and max analysis.

To get best-case/worst-case analysis results (best-case conditions for min analysis and worst-case conditions for max analysis), you need two crosstalk analysis runs: one using only best-case conditions, and another using only worst-case conditions. For example, for non-crosstalk analysis, suppose that you are using:

```
pt_shell> set_operating_conditions -analysis_type bc_wc \
            -min BEST -max WORST
pt_shell> report_timing -delay_type min_max
```

Then, for crosstalk max analysis under WORST operating conditions, you would use:

```
pt_shell> set si_enable_analysis true
pt_shell> set_operating_conditions -analysis_type \
            on_chip_variation WORST
pt_shell> report_timing -delay_type max
```

Then, for crosstalk min analysis under BEST operating conditions, you would use:

```
pt_shell> set_operating_conditions -analysis_type \
            on_chip_variation BEST
pt_shell> report_timing -delay_type min
```

Using check_timing

The `check_timing` command can check for several conditions related to crosstalk analysis, making it easier to detect conditions that can lead to inaccurate crosstalk analysis results. It is recommended that you run `check_timing` after you set the constraints and before you start an analysis with `update_timing` or `report_timing`.

There are four types of checking that are specific to crosstalk analysis:

- `no_driving_cell` – The `check_timing` command reports any input port that does not have a driving cell and does not have case analysis set on it. When no driving cell is specified, that net is assigned a strong driver for modeling aggressor effects, which can be pessimistic.
- `ideal_clocks` – The `check_timing` command reports any clock networks that are ideal (not propagated). For accurate determination of crosstalk effects, the design should have a valid clock tree and the clocks should be propagated.

- `partial_input_delay` – The `check_timing` command reports any inputs that have only the minimum or only the maximum delay defined with `set_input_delay`. To accurately determine timing windows, PrimeTime SI needs both the earliest and latest arrival times at the inputs.
- `unexpandable_clocks` – The `check_timing` command reports any clocks that have not been expanded to a common time base. For accurate alignment of arrival windows, all of the synchronous and active clocks of different frequencies must be expanded to a common time base.

With the exception of `ideal_clocks`, crosstalk-related checks are now on by default. To enable ideal_clocks, you can either set the `timing_checks_default` variable or use the `-include` option of `check_timing`. For example:

```
pt_shell> check_timing -include { ideal_clocks }
```

Initial Crosstalk Analysis Run

For the first analysis run with crosstalk analysis, it is a good idea to use the default settings for the crosstalk variables so that you can obtain results quickly. For example:

```
pt_shell> set si_enable_analysis TRUE
pt_shell> report_timing
```

With the default variable settings, PrimeTime SI does the crosstalk analysis using two delay calculation iterations. In the first iteration, PrimeTime SI ignores the timing windows so that it can quickly get an estimate of crosstalk delay effects. In the second and final iteration, PrimeTime SI reselects only the nets in the critical path of each path group, and then does a detailed analysis of those nets considering the timing windows and transition types (rising or falling).

Using the default settings, you can quickly determine the following design and analysis characteristics:

- The effectiveness and accuracy of the current electrical filtering parameter
- The approximate overall signal integrity of the design (by the presence or absence of a large number of constraint violations)
- The approximate runtime behavior for the critical path in the current design
- The detailed timing effects calculated for the critical path

At this point, if no timing violations are reported, it is likely that your design will meet the timing specifications. If PrimeTime SI reports violations or small slack values, you need to do a more detailed analysis to find the causes of these conditions. Also, if you are near the end of the design cycle, you should do a more detailed analysis to confirm the results of the fast (default) analysis.

Additional Crosstalk Analysis Runs

There are many ways to modify the PrimeTime SI variable settings to obtain a more thorough and accurate analysis, at the cost of more execution time. A good starting point is to try the following:

```
pt_shell> set si_xtalk_reselect_min_mode_slack 0
pt_shell> set si_xtalk_reselect_max_mode_slack 0
pt_shell> report_timing
```

If you know that the clock is designed in such a way that it cannot be a victim net, then you can disable net reselection based on the change in delay calculated in the previous iteration. To do so, set the delta delay reselection parameters to very large numbers. For example:

```
pt_shell> set si_xtalk_reselect_delta_delay 100000
pt_shell> set si_xtalk_reselect_delta_delay_ratio 100000
```

As a result of these settings, even if the clock net has a large change in calculated delay due to crosstalk, it is not reselected for analysis; only data path nets that contribute to slack violations are reselected. This can speed up the analysis considerably because of the resources that would otherwise be used for analyzing the clock net.

However, if you need to analyze clock paths in the detailed analysis, you can set either one of the delta delay variables to some meaningful value, appropriate to your technology, so that nets are reselected for analysis if they have a large enough change in calculated delay due to crosstalk.

By selecting a combination of the critical slack range and delta delay threshold, you have the ability to trade off accuracy against runtime.

By default, PrimeTime SI terminates the delay calculation loop after two iterations, which usually provides results that are sufficiently accurate. It is recommended that you change the exit criteria only if additional iterations are necessary for more accurate (less pessimistic) results, and the incremental increase in accuracy is expected to be worth the additional cost in runtime.

When the analysis is done, you can inspect the results generated by the `report_timing -crosstalk_delta` command. To obtain more detailed information, you can use the built-in crosstalk histogram reports. You can also extract the crosstalk attribute values from the design database using your own Tcl scripts. The design attributes that you can extract are described in Appendix A, “Crosstalk Attributes”.

Timing Window Overlap Analysis

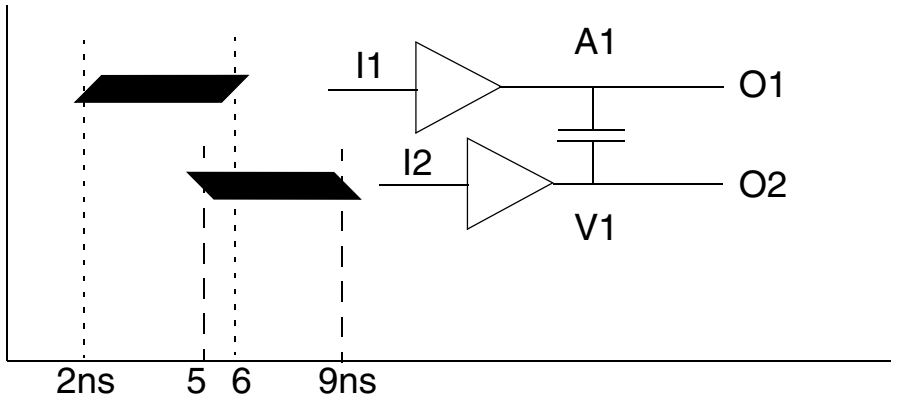
The crosstalk effect on a net depends on the alignment of the victim and aggressor switching time. The following sections discuss this effect.

Overview

Depending on how the victim and aggressor switching times align, a net could become slower or faster depending on the switching directions. PrimeTime SI calculates the crosstalk effect based on the timing window of the aggressors and victim. This process is referred as timing window overlap analysis or aggressor alignment.

During timing window overlap analysis, PrimeTime SI calculates the crosstalk delta delay per load pin of a net. For this purpose, the timing arrival windows are used by PrimeTime SI, because it encapsulates all the timing paths passing through the net. If the aggressor partially overlaps with the victim's timing window, the partial effect (smaller delta delay) is considered. Figure 2-4 illustrates how timing windows can overlap.

Figure 2-4 Victim-Aggressor Switching Time Alignment



In this example, victim V1 is coupled with aggressor A1. The timing arrival windows are 2ns to 6ns for the aggressor, and 5ns to 9ns for the victim. Since the victim timing window overlaps with the aggressor's timing window, the signal integrity engine calculates the crosstalk delta delay due to this aggressor.

Sometimes, when timing windows from different clocks exist on a victim and aggressor net, PrimeTime SI considers the different combinations of these clocks with respect to the clock periods.

Multiple Aggressors

When there are multiple aggressors in the design, the signal integrity engine finds the combination of aggressors that could produce the worst crosstalk effect and calculates the crosstalk delta delay for this combination. A multiple aggressor is shown in Figure 2-5.

Figure 2-5 Multiple Aggressor Alignment

In this example, the victim has three aggressors: A1 is stronger than A2, and A2 is stronger than A3. Since aggressor A1's window does not overlap with the victim's window, and A2 is stronger than A3, the signal integrity engine will calculate the crosstalk delay due to aggressor A2. The A1 and A3 will not be considered for delta delay calculation.

The `report_delay_calculation -crosstalk` command will report the attributes for aggressors A1 and A3 as follows:

```
N - aggressor does not overlap for the worst case alignment
```

Asynchronous Clocks

If the victim timing window clock and the aggressor timing window clocks are asynchronous, they have no fixed timing relationship with each other. The aggressor will be treated as infinite window with respect to the victim. The `report_delay_calculation -crosstalk` command will report this as follows:

```
I - aggressor has Infinite arrival with respect to the victim
```

For multiple aggressors, if the aggressor clocks are synchronous with each other, but asynchronous with the victim, the timing relationships between the aggressors are respected, but they are still treated as infinite windows with respect to the victim.

Crosstalk Delay Analysis for All Paths

In the default mode, PrimeTime SI calculates the maximum possible delta delay (worst crosstalk effect) for the victim and aggressor arrival windows. This ensures that crosstalk delta delay is considered for all paths passing through the victim net, and that the maximum delta delay value is applied on that net. This guarantees that all the paths going through the victim net are conservative.

To use this type of analysis, set the `si_xtalk_delay_analysis_mode` variable to `all_paths` mode.

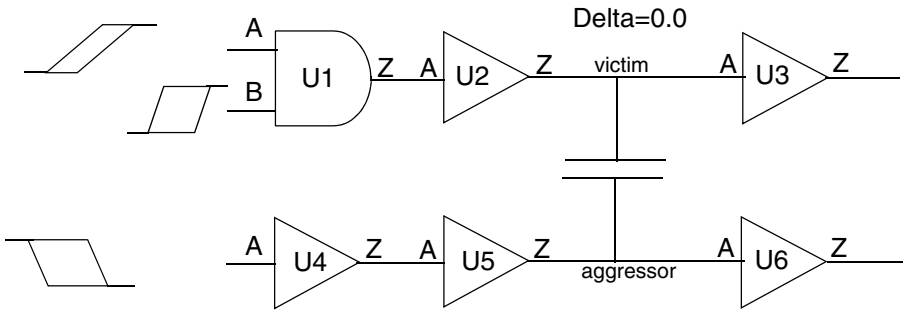
The disadvantage of this approach is that the largest crosstalk delta delay value is applied to the critical paths, making them pessimistic. When a path is recalculated using path-based analysis, this pessimism is removed. You can also remove this pessimism by using other available crosstalk delay analysis modes.

Crosstalk Delay Analysis for the Worst Path

When you set the `set si_xtalk_delay_analysis_mode` variable to `worst_path` mode, PrimeTime SI calculates crosstalk delta delay for only the worst arrival paths. This option guarantees a conservative calculation for the worst arrival path, but could introduce optimism in sub-critical paths. Because this mode respects any false paths that you set, it calculates the true worst arrival path. The true worst-arrival signal decides the timing slack of the design, so the design slack is conservative when you use this mode.

The `worst_path` mode is less conservative than the `all_paths` mode, but in some scenarios subcritical paths might become optimistic, as illustrated in Figure 2-6.

Figure 2-6 Potential Optimism in Sub-critical Paths



In this example, the delta delay calculated for the path through pin U1-B is 0.0. However, using a delta delay of 0.0 for the path through pin U1-A makes it optimistic, as it overlaps with the aggressor window. In this type of scenario, you could run into the following types of issues:

- If you issue a `report_timing -nworst N`, where N is greater than 1, the report will include paths with optimistic slack.
- The `report_si_bottleneck` command may not report all bottlenecks.

When you set the `si_xtalk_delay_analysis_mode` variable to `worst_path` mode, delay calculation is done for each clock in the clock network. This is useful when multiple clocks are propagated in the clock network from the clock selection multiplexors.

Crosstalk Delay Analysis for All Violating Paths

When you set the `si_xtalk_delay_analysis_mode` variable to `all_violating_paths` mode, PrimeTime SI calculates crosstalk delta delay for all violating paths (paths with negative slack) as well as for the worst path. This is the recommended mode for much of the design process. This mode is less pessimistic than `all_paths` mode (the default), and more conservative than `worst_path` mode.

In `all_violating_paths` mode, all paths with negative (or zero) slack are guaranteed to have conservative delta delay, path arrival, and slack. This mode is recommended for the fixing flow, because the results of the `report_timing -slack_lesser_than 0.0 -nworst N` command, where N is greater than 1, will always be conservative. It also reports all crosstalk bottlenecks when the `report_si_bottleneck -slack_lesser_than 0.0` command.

Like `worst_path` mode, this mode respects any false paths that you set, and it calculates crosstalk delay for the true worst arrival path.

When you set the `si_xtalk_delay_analysis_mode` variable to `all_violating_paths` mode, delay calculation is done for each clock in the clock network. This is useful when multiple clocks are propagated in the clock network from the clock selection multiplexors.

For the clock network and unconstrained paths, the worst-path arrival signal is used for alignment.

Clock Groups

When multiple clocks exist in a design, you can use the `set_clock_groups` command to specify the relationships between the clocks. Doing so allows PrimeTime SI to correctly analyze the crosstalk interactions between the clocks. This is the command syntax:

```
set_clock_groups
  -group clock_list
   [-physically_exclusive |
    -logically_exclusive] |
     -asynchronous]
   [-allow_paths]
   [-name name]
```

The `-group` option specifies the names of the clocks that belong to a group, whereas the `-name` option assigns an arbitrary name to the group. The remaining options specify the relationship between the clocks in the group. The clocks in a group can be defined as logically exclusive, physically exclusive, or asynchronous.

Two clocks defined to be logically exclusive of each other have no logical timing paths between them. PrimeTime does not check the logical timing between the clocks, but PrimeTime SI still checks for possible crosstalk interaction between them.

Two clocks defined to be physically exclusive of each other have no logical timing paths between them, and furthermore, are considered physically isolated from each other. PrimeTime does not check the logical timing between the clocks and PrimeTime SI assumes no possible crosstalk interaction between them.

Two clocks defined to be asynchronous with each other have no timing relationship at all. PrimeTime does not check the logical timing between the clocks, but PrimeTime SI still checks for possible crosstalk interaction between them, using infinite arrival windows.

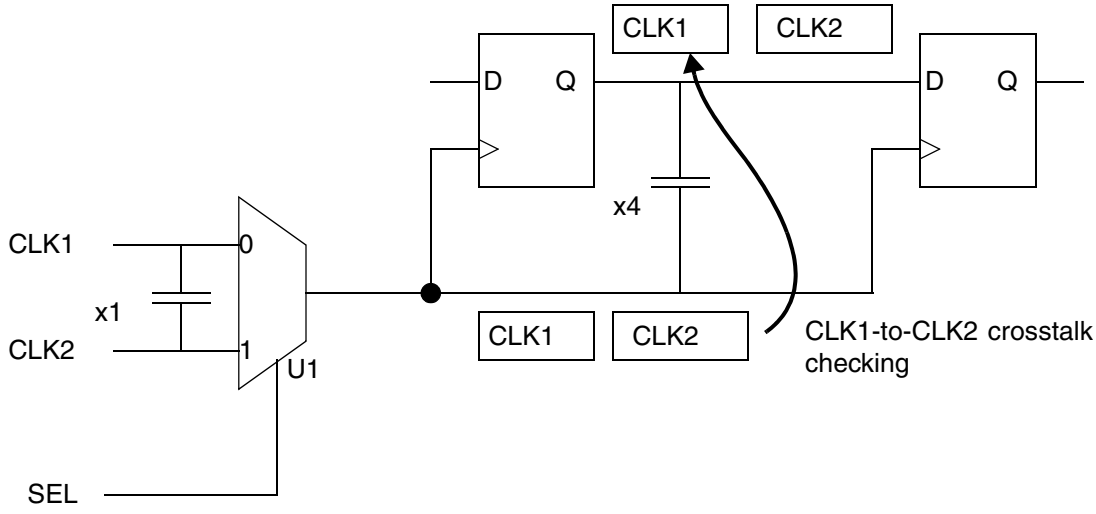
The `-allow_paths` can be used with asynchronous clocks to restore analysis of the timing paths between clock groups, but still using infinite alignment windows for crosstalk analysis.

Logically and Physically Exclusive Clocks

The most accurate way to analyze multiple clocks is to run a separate analysis for each possible combination of clocks. If there are many such combinations, you can use the distributed multi-scenario analysis (DMSA) feature of PrimeTime to run the analyses and get unified results. However, modern designs often use many clocks, perhaps hundreds or even thousands, to save power. If the number of clocks is very large, it might not be practical to analyze each combination separately. In that case, you can analyze multiple clocks simultaneously in a single run, and use `set_clock_groups` to specify the timing relationships between groups of clocks.

For example, consider the circuit shown in Figure 2-7. Only one of the two input clocks is enabled at any given time. However, by default, PrimeTime SI considers the crosstalk across capacitor x4, between the overlapping arrival windows of CLK1 and CLK2.

Figure 2-7 Circuit with Multiplexed Clocks



The most accurate way to handle this situation is to use case analysis, first setting the MUX control signal to 0 and then to 1. This method ensures that there is no interaction between the clocks and correctly handles all crosstalk situations. However, it requires two analysis runs. For a design with many clocks, it might not be practical to analyze every possible combination of enabled clocks.

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

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

The `-logically_exclusive` option causes PrimeTime to suppress any logical (timing path) checking between CLK1 and CLK2, similar to setting a false path constraint between the clocks. However, PrimeTime SI still computes crosstalk delta delays across coupling capacitor x4 between the two clocks, which is pessimistic if the two clocks are not simultaneously present on the nets.

To eliminate this pessimism, you can define the clocks to be physically exclusive:

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

PrimeTime SI does not compute any delta delays between the clocks defined to be physically exclusive, thereby eliminating the pessimistic analysis of crosstalk between CLK2 and CLK1 across capacitor x4. However, crosstalk across capacitor x1 is also eliminated, which can be optimistic if the MUX is deep inside the chip and x1 is significant.

To correctly handle both cross-coupling capacitors, instead of declaring the original clocks to be exclusive, you can define two generated clocks at the output of the MUX and define them to be physically exclusive:

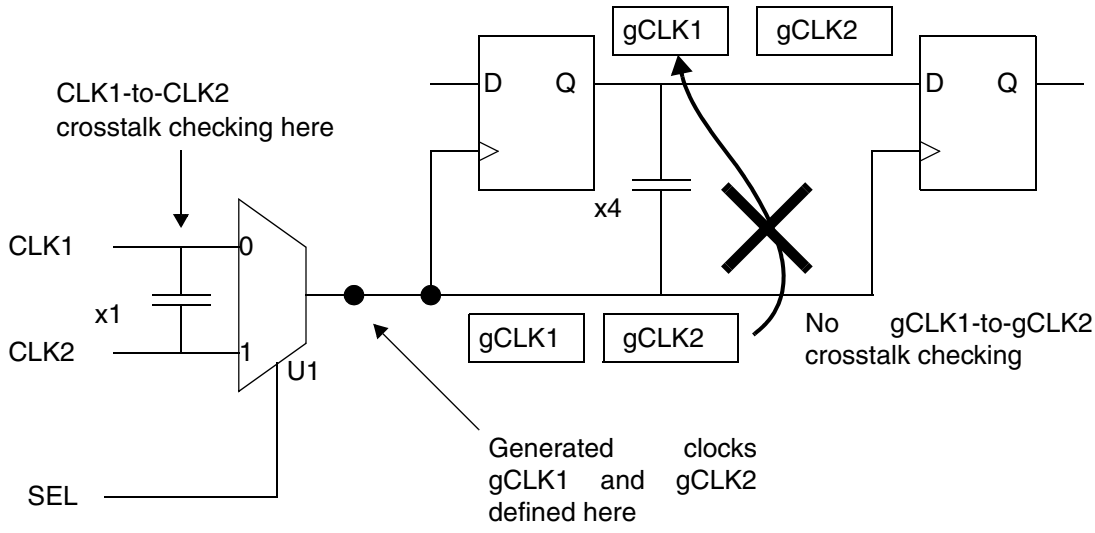
```
pt_shell> create_generated_clock -name gCLK1 \
          -source [get_ports CLK1] -divide_by 1 \
          -add -master_clock [get_clocks CLK1] \
          [get_pins U1/z]

pt_shell> create_generated_clock -name gCLK2 \
          -source [get_ports CLK2] -divide_by 1 \
          -add -master_clock [get_clocks CLK2] \
          [get_pins U1/z]

pt_shell> set_clock_groups -physically_exclusive \
          -group {gCLK1} -group {gCLK2}
```

In that case, PrimeTime SI computes delta delays between CLK1 and CLK2 across x1 before the MUX, but not between gCLK1 and gCLK2 across x4 or elsewhere in the generated clock tree. See Figure 2-8.

Figure 2-8 Circuit with Multiplexed Clocks



The definition of physically exclusive clock groups removes pessimism by eliminating crosstalk analysis where no crosstalk should exist. The pessimism removal applies to both crosstalk timing analysis and crosstalk noise analysis. It is the user's responsibility to correctly define the clock groups. PrimeTime SI does not verify that a particular setting makes sense for the design.

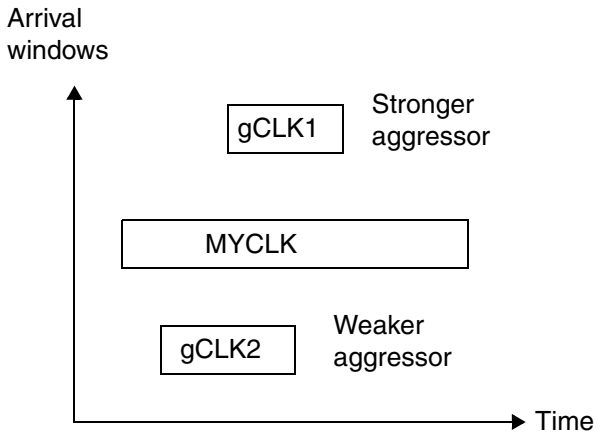
PrimeTime detects only group-to-group conflicts, not implied conflicts. For example, if you declare CLK1 and CLK2 to be asynchronous to each other, they are both still synchronous to CLK3 by default. This is an implied three-way conflict that PrimeTime does not detect. The user is responsible for resolving such conflicts by correct use of the `set_clock_groups` command.

An exclusive or asynchronous path group definition has higher priority than a `set_false_path` timing exception. Furthermore, the `reset_path` command does not cancel path group relationships set with the `set_clock_groups` command.

Path-Based Physical Exclusion Analysis

If two or more clocks are defined to be physically exclusive of each other, no more than one of those clocks can operate as an aggressor to a given victim net. For example, consider the crosstalk alignment diagram in Figure 2-9.

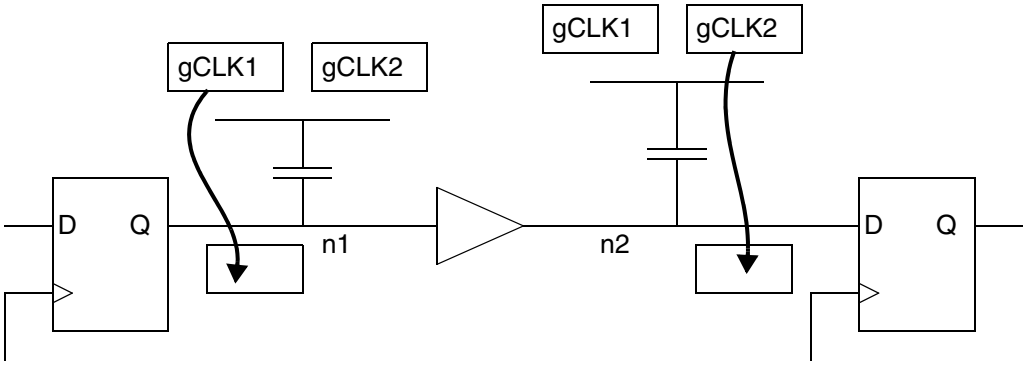
Figure 2-9 Crosstalk Alignment Diagram



The physically exclusive clock signals gCLK1 and gCLK2 are aggressors to clock signal MYCLK. Both of the aggressors overlap the arrival window of MYCLK. However, because gCLK1 and gCLK2 are physically exclusive, only one can be an aggressor at that victim net at any given time. PrimeTime SI chooses the stronger aggressor that causes a larger delta delay (in this example, gCLK1), and does not consider the weaker one (gCLK2).

For each net, a different aggressor could be the stronger one. For example, consider the circuit in Figure 2-10. The two clocks gCLK1 and gCLK2 are physically exclusive, and both operate as aggressors to victim nets n1 and n2 in a timing path.

Figure 2-10 Different Exclusive Aggressors on a Path



Because of the different arrival times at the two victim nets, gCLK1 is the aggressor for net n1 and gCLK2 is the aggressor for net n2. This analysis is correct for each individual net, but it is pessimistic for the path as a whole because gCLK1 and gCLK2 are physically exclusive. They cannot both contribute to a decrease in the slack for the path.

You can optionally have PrimeTime consider the worst possible set of allowable (non-exclusive) clocks resulting in the least slack for each path. This whole-path analysis reduces pessimism, but requires slightly more runtime than considering nets individually. Even with the increased runtime, it is still typically much faster than repeated analysis runs that consider all the clock combinations separately. For the most accurate worst-case, whole-path analysis of physically exclusive clocks, set the following variable to true:

```
pba_enable_path_based_physical_exclusivity
```

By default, it is set to false. It is suggested that you leave this variable set to false for most analysis runs, and set it to true only for final signoff of the design timing.

Infinite Alignment Windows

The `-allow_paths` option can be used with the `-asynchronous` option to restore logical (timing path) checking between clock groups, while still using infinite alignment windows for crosstalk analysis. This option allows the timing paths between the clock paths to remain in place, but applies infinite windows between the clock groups for conservative crosstalk analysis.

For example, the following command defines clocks CLK1 and CLK2 to be asynchronous for purposes of crosstalk analysis (infinite arrival windows), without affecting normal logical timing checks between the two clocks:

```
pt_shell> set_clock_groups -asynchronous -allow_paths \
          -group {CLK1} -group {CLK2}
```

You can restrict checking of some or all clock-to-clock paths by using the `set_false_path` command in conjunction with the `set_clock_groups` command.

High Capacity Analysis Mode

The current trend in most of the new designs is an increase in the amount and complexity of core logic. This leads to an increase in the size and complexity of the design netlist, constraints, parasitics, and so on.

In order to analyze this large amount of data efficiently, PrimeTime SI has an optional high capacity mode targeted for very large designs with clock reconvergence pessimism removal (CRPR) enabled. The high capacity mode makes runtime trade-offs between performance and capacity, while producing the same results as normal analysis. High capacity mode is compatible with all analysis flows (including flat and hierarchical), but is most useful for PrimeTime SI with CRPR enabled.

You enable high capacity mode using the `set_program_options -enable_fast_analysis` command. No other changes are required in your existing scripts or flows. You must issue the `set_program_options` command prior to loading designs or libraries. You are not required to change your existing flow, except to enable high capacity mode prior to loading designs and libraries.

For example, the following enables high capacity mode with the default settings.

```
pt_shell> set_program_options -enable_fast_analysis
Information: enabling high capacity analysis mode....
(PTHC-001)
```

You can set the level of capacity effort using the `sh_high_capacity_effort` variable, which should be set before running `set_program_options`. This variable provides simple heuristics for trade-off between capacity and performance; it does not impact the analysis results. The values available are `low`, `medium` (the default), and `high`. For example, to set the caching effort to high, use

```
pt_shell> set sh_high_capacity_effort high
```

Adaptive CRPR Mode

Clock reconvergence pessimism removal (CRPR) can require a considerable amount of runtime under certain conditions, for example, when there is a significant variation in the clock network due to crosstalk. To minimize the possibility of excessive runtime while maintaining the enhanced accuracy of pessimism removal in the critical paths, you can enable CRPR in an adaptive mode.

In the adaptive CRPR mode, PrimeTime SI calculates CRPR only for the critical path set (the violating portion of the design), thereby reducing the complexity of CRPR analysis. The critical path set is the set of all paths with a slack of less than zero, for both setup and hold timing constraints. The adaptive mode operates only when crosstalk analysis is enabled and the maximum iteration count is set to 2 or more.

The adaptive CRPR mode uses less runtime than the default CRPR mode, but retains the increased accuracy of pessimism removal for the critical path set. The most significant performance improvement occurs when the critical path set is small and the memory consumption of the normal CRPR mode is large.

The `timing_crpr_enable_adaptive_engine` variable controls the adaptive CRPR mode. Setting the variable to `true` causes PrimeTime SI to perform pessimism removal only on the critical paths, as long as crosstalk analysis is enabled and the number of crosstalk iterations is set to 2 or more. The default setting is `false`, which causes PrimeTime to perform CRPR on all paths.

You might see some differences in the results between standard and adaptive CRPR. This is because nets chosen for reselection in the second and subsequent analysis iterations depend on slack values, and slack values depend on CRPR calculations to some extent. Typically, more nets are reselected for analysis in the second iteration using adaptive CRPR because no CRPR is done in the first iteration. The resulting analysis is more accurate (less pessimistic) because more nets are reselected for analysis.

Crosstalk Analysis with Composite Aggressors

Some complex designs require an efficient method of analyzing multiple aggressors to a single victim net. In general, such designs have the following characteristics:

- A large number of aggressors per victim net
- Little or no filtering of aggressors
- Crosstalk calculations are performed in high effort mode

The composite aggressor feature makes an effective tradeoff between runtime and accuracy by evaluating all “small” aggressors, including those that are filtered, in a fast but conservative manner. It can also reduce pessimism by utilizing an optional statistical analysis for the aggressors in the composite aggressor group.

In finer geometries, nets have a large number of small aggressors. Filtering these small aggressors completely ignores their effects. However, evaluating all of them in a detailed analysis can result in extremely long runtimes. The composite aggressor feature provides a way to consider the effects of many small aggressors in a reasonable amount of runtime.

A composite aggressor is a single composite waveform that represents the effects of all the small aggressors, including those that are filtered. This concept is illustrated in Figure 2-11 and Figure 2-12.

Figure 2-11 Composite Aggressor Mode Disabled

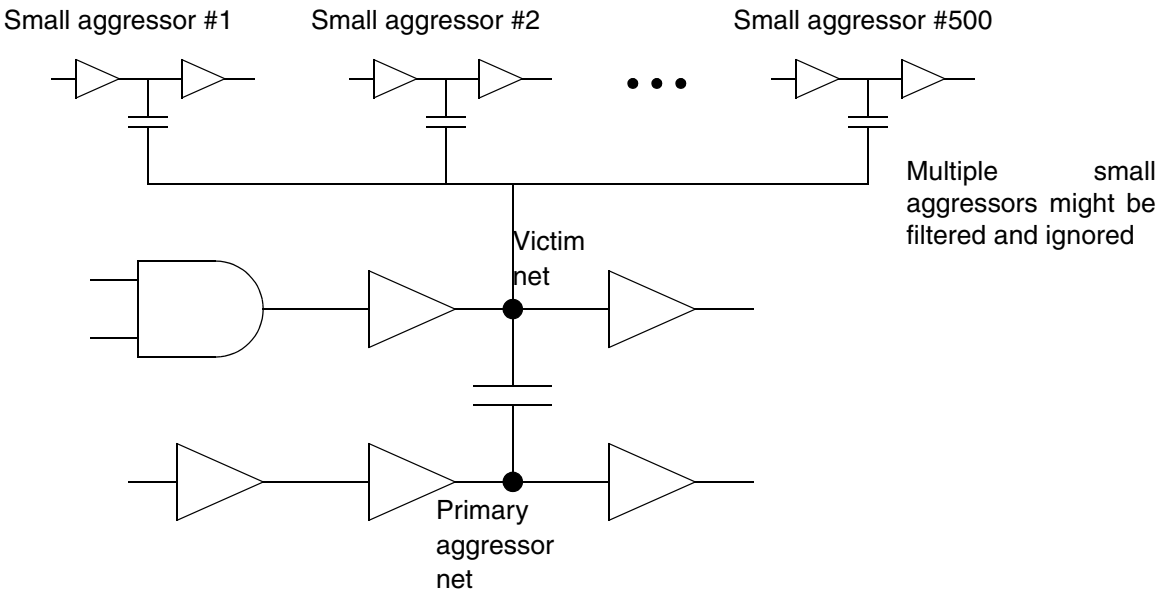
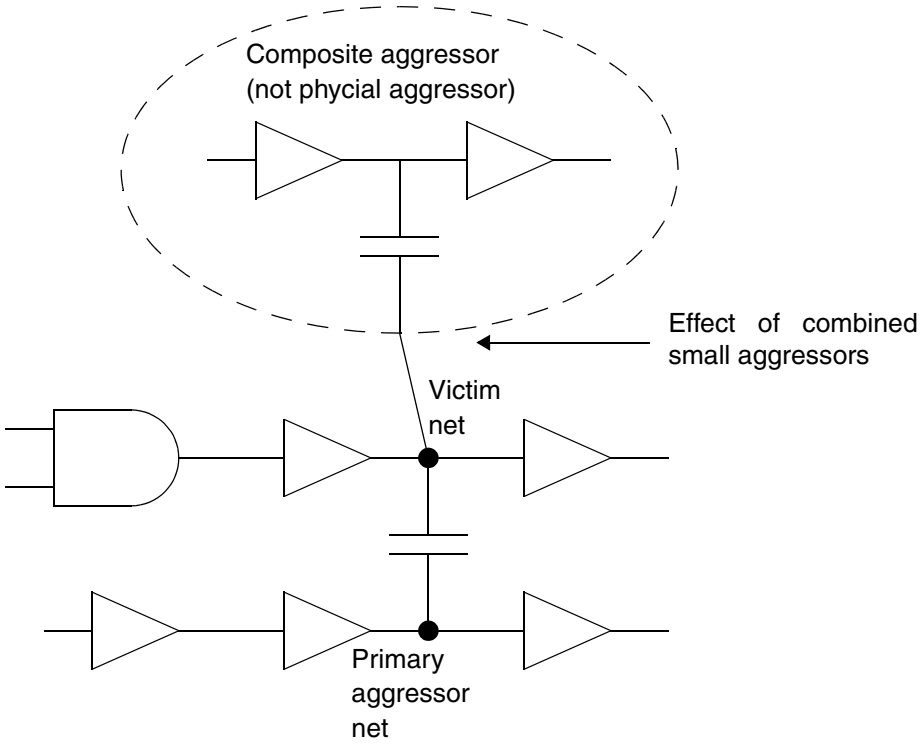


Figure 2-12 Composite Aggressors Enabled



Aggressors (1, 2, ... 500) are small aggressors in Figure 2-11. Some of these might be aggressors that were filtered out before you enabled the composite aggressor feature. They are all replaced by one composite aggressor (waveform) in Figure 2-12. Note that the composite aggressor represents the combined effect of the group of physical small aggressors and is not made of physical cells.

Enabling Composite Aggressors

To enable the composite aggressor feature, use the following variable:

```
si_xtalk_composite_aggr_mode
```

This variable can be set to any one of the following values:

- `disabled` (default) – The composite aggressor feature is disabled and filtered aggressors are ignored. Unfiltered aggressors are all fully considered in the analysis.
- `normal` – PrimeTime SI combines the effect of some small aggressors (possibly including filtered ones) into a single composite aggressor, thereby accounting for their worst-case effects in an efficient manner.

- `statistical` – The normal composite aggressor feature is enabled in statistical mode, which reduces the effect of the composite aggressor to reduce the pessimism inherent in combining all of the worst-case small aggressors.

The following variable sets a bump height threshold below which an aggressor is treated as part of the composite aggressor:

```
si_xtalk_composite_aggr_noise_peak_ratio
```

This variable sets the bump height threshold as a fraction of the power supply voltage. The default setting is 0.01, which causes aggressors with a bump height below 1 percent of the power supply voltage to be considered “small” and become part of the composite aggressor. In addition, any aggressors meeting the normal electrical filtering criteria (both peak and accumulated) will also become part of the composite aggressor instead of being discarded.

In normal composite aggressor mode, all of the aggressors below the filtering thresholds are combined to generate the composite aggressor. Since these aggressors are defined as small, they are combined in a conservative fashion to minimize the impact on runtime.

In the statistical mode, all of the small aggressors below the threshold are still included as part of the composite aggressor. However, some statistical analysis is applied to adjust the composite bump height to a lower level. This analysis considers the finite probability that all the small aggressors will switch simultaneously to adversely affect the victim. This adjustment results in a more realistic, less pessimistic analysis than the normal composite aggressor mode.

Operation of the statistical mode is based on the assumption that each aggressor contributing to the composite aggressor can switch in the rising or falling direction to either help or hurt the victim, and that the possible range for the composite aggressor bump height is any value from zero and the worst-case value.

Based on these assumptions, PrimeTime SI statistically combines the individual aggressors below the threshold. It determines largest composite aggressor bump height having a probability of occurrence no greater than a certain threshold value. You can set this value with the following variable:

```
si_xtalk_composite_aggr_quantile_high_pct
```

By default, this variable is set to 99.73, representing a probability of 99.73 percent that the generated composite bump will be at least as large as the actual effect of the small aggressors. This is three standard deviations (3 sigma) from the mean value of a normal distribution. To specify a different probability, set the variable to the desired percentage. For example, choose a smaller value such as 90 to generate a smaller, less conservative composite bump that is closer to the mean value.

Excluding Nets from Statistical Mode

If you want to use statistical mode but have certain nets in your design for which you do not want to statistically reduce the aggressor impact, you can disable statistical analysis for just those nets by using the following commands:

- `set_si_delay_disable_statistical`
- `remove_si_delay_disable_statistical`

These commands take a list of nets as an argument. The commands have no effect if a net is not part of the composite aggressor group.

Reporting Composite Aggressors

To see which unfiltered aggressors are part of a composite aggressor, use the following reporting command:

```
report_delay_calculation -crosstalk
```

Only aggressors that meet the electrical filtering threshold are reported, although all are considered in the composite aggressor. Aggressors that are part of a composite aggressor are labeled with a “C” in the report. A line in the report lists the composite aggressor mode (`disabled`, `normal`, `statistical`, `physical`, or `normal_physical`).

If you want to see all aggressors, including those that fall below the electrical filtering thresholds, you can use the `get_attribute` command. The four attributes that show a collection in composite aggressor group are

- `si_xtalk_composite_aggr_min_rise`
- `si_xtalk_composite_aggr_min_fall`
- `si_xtalk_composite_aggr_max_rise`
- `si_xtalk_composite_aggr_max_fall`

You use them for four different analysis types: min_rise, min_fall, max_rise, and max_fall. For example, the following command gets aggressor information for min_rise analysis.

```
pt_shell> get_attribute -class net vict1 \
          si_xtalk_composite_aggr_min_rise
{"aggr1", "aggr2", "aggr3"}
```

Path-Based Analysis

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. A path-based analysis consists of three steps:

1. Use the `get_timing_paths` command to create a path collection that is to be analyzed in isolation from other paths.
2. Use the `get_recalculated_timing_paths` command to recalculate the delay and slack for the path collection.
3. Use the `report_timing -of_objects` command to get a timing report for the path collection.

In a path-based timing analysis, PrimeTime SI recalculates the crosstalk effects using new victim arrival times and slews taken from the path collection, but using aggressor arrival windows determined by the previous timing update.

For more information on path-based timing analysis, see the section on that subject in the *PrimeTime Advanced Timing Analysis User Guide*, in the "Advanced Analysis Techniques" chapter.

Advanced Delay Calculation Using CCS Models

PrimeTime SI supports two types of library models: composite current source (CCS) and nonlinear delay model (NLDM). CCS is a newer, more advanced technology that provides greater accuracy for technologies at 65 nm and below. NLDM is an older technology that provides good results for technologies above 65 nm.

For libraries that have both CCS timing and CCS noise models, PrimeTime SI applies an advanced, gate-level delay calculation method that uses the CCS timing and noise models. This type of analysis results in greater accuracy for each cross-coupled network of aggressor and victim nets with their drivers, receivers, and parasitics.

PrimeTime SI performs the advanced delay calculation for nets reselected for analysis in the final iteration of crosstalk delay analysis, where the amount of coupling is large enough to benefit from the additional analysis. It uses CCS models for the drivers and receivers of the victim net, as well as for the aggressor nets. It builds a multi-input, multi-output reduced-order model of the coupled interconnects. It then performs a time-step delay calculation using the model, resulting in piecewise-linear waveforms at the inputs of the victim receivers.

In earlier stages of the design flow, you can optionally disable the advanced delay calculation mode so that PrimeTime SI uses conventional analysis techniques for all crosstalk delay analysis, which saves runtime. To disable advanced delay calculation:

```
pt_shell> set si_ccs_use_gate_level_simulation false
```

By default, the variable is set to `true` and advanced delay analysis is enabled. For final sign-off analysis, it is recommended that you set the variable back to `true` to get the highest possible accuracy.

By default, PrimeTime SI aligns the aggressor transitions to produce the worst delay effect for the whole path, which might be different from the alignment that produces the worst stage delay. This is because the worst delay effect for the whole path depends on the cumulative delay changes along multiple stages along the path. The worst alignment for a single stage might result in smaller delay effects at other stages along the path.

By default, to find the worst alignment for a path, PrimeTime SI looks ahead of the victim receiver of the current stage being analyzed, to consider the path-wide effects. If you want to use the aggressor alignment to get the worst stage delay rather than the worst path delay, set the variable `si_ccs_aggressor_alignment_mode` to `stage`. By default, this variable is set to `lookahead`. The lookahead method is always used during path-based analysis, irrespective of the variable setting.

PrimeTime SI generally uses the standard Synopsys predriver model to drive the primary inputs for timing analysis. However, for best possible accuracy using the advanced delay calculation method, it should use the same type of predriver that was used to characterize the cells, which might be either the Synopsys predriver or a simple ramp.

PrimeTime SI gets the predriver modeling information from the library, if available. If the library does not have the predriver information, you can specify the predriver type explicitly in PrimeTime SI by using the following command:

```
set_library_driver_waveform
  [-type ramp | standard ]
  [library_objects]
```

The `-type` setting specifies the predriver type, either a simple ramp or the standard Synopsys predriver. If you specify one or more library objects (a collection of libraries or library cells) in the command, it applies only to those objects. Otherwise, the command applies to the whole design. You can query the `lib_pin` attributes `driver_waveform_rise` and `driver_waveform_fall` to find out the type of predriver that has been applied.

Setting the predriver type overrides any predriver specification in the applicable library. Setting the driver waveform type triggers a timing update if crosstalk analysis is enabled.

To reduce memory usage, for each piecewise-linear waveform at a receiver input pin, PrimeTime SI computes a simpler waveform that has the same path delay effects as the original waveform. Then it stores the equivalent waveform information on each input pin, using less memory than the full piecewise-linear waveform. Experiments have verified that using the equivalent waveform typically produces very accurate delay calculation results.

For even higher accuracy during path-based analysis, PrimeTime SI can annotate the piecewise-linear waveforms in the design and propagate those waveforms, instead of using equivalent waveforms, in combination with using the advanced delay calculation mode.

(Path-based analysis is performed with either the `get_timing_paths -recalculate` command or the `get_recalculated_timing_paths` command.) To enable this feature, set the following variable:

```
pt_shell> set pba_enable_ccs_waveform_propagation true
```

By default, this variable is set to `false`. When it is set to `true`, PrimeTime SI invokes the advanced delay calculation mode during path-based analysis, irrespective of the `si_ccs_use_gate_level_simulation` variable setting, and performs delay calculation using the actual piecewise-linear waveforms along the full length of each path reselected for path-based analysis. This results in more accuracy in cases where the actual waveform shape is significantly different from the waveform derived from the characterized cell model. Such differences can result from capacitive coupling, resistive shielding, the receiver backward Miller effect, and other conditions.

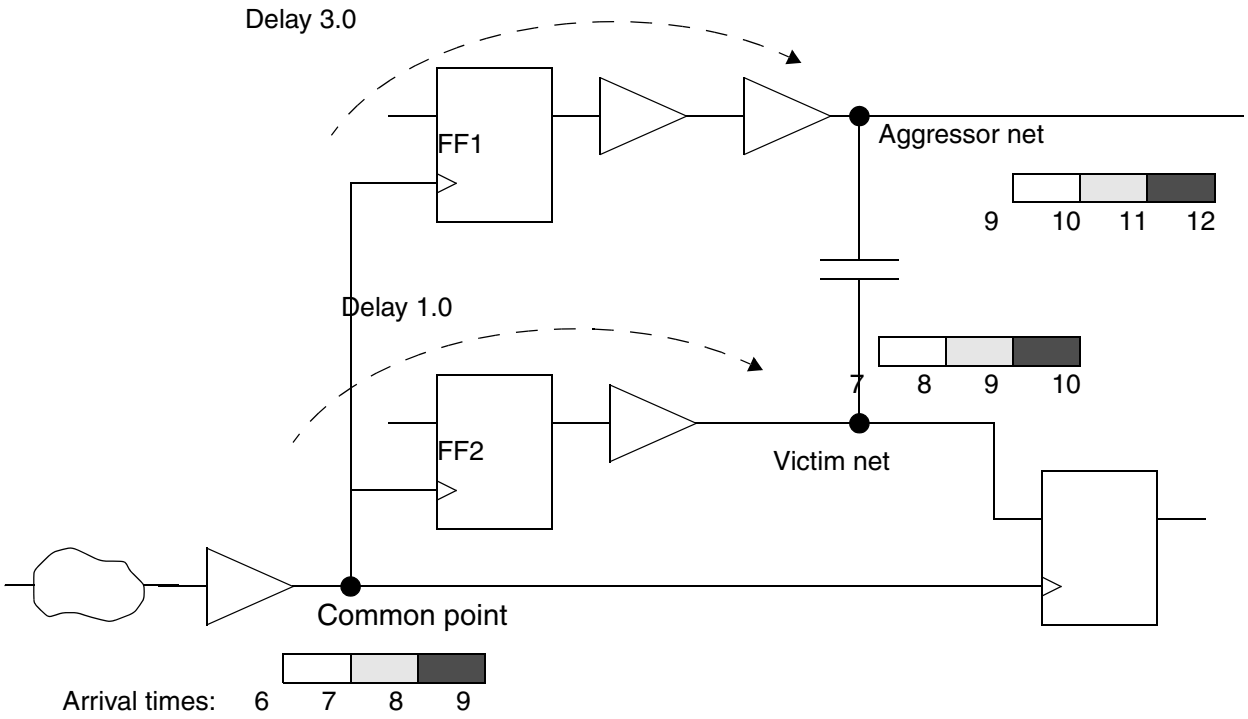
Clock On-Chip Variation Pessimism Reduction

The `set_timing_derate` command can be used to model the effects of on-chip variation (OCV). The command specifies a factor by which the delays of long path delays are increased or the delays of short paths are decreased. When there is a common segment between the launch and capture clock paths, the clock reconvergence pessimism removal (CRPR) algorithm, if enabled, removes the pessimism caused by the derating of delay times along the common segment.

However, by default, pessimism removal occurs only after the final slack has been calculated for the timing path. No pessimism removal occurs during the calculation of crosstalk arrival windows. If the clock paths leading up to the aggressor net and victim net share a common segment, then the calculated arrival windows are wider than they should be, possibly causing the aggressor and victim arrival windows to marginally overlap. This can cause a crosstalk situation to occur that would not occur with accurate CRPR accounting during arrival window overlap analysis.

Figure 2-13 is a simple example that demonstrates the pessimism caused by derating a long clock path. There is some cross-coupling capacitance between two wires in the fanout of FF1 and FF2, both of which are clocked by the same clock signal.

Figure 2-13 Clock Reconvergence Pessimism in Crosstalk Arrival Windows



In the absence of derating, the arrival windows are 1.0 time units wide. Because of the differences in delay along the paths leading up to the aggressor and victim nets, the windows do not overlap and no crosstalk delay effects are possible. For example, if the aggressor transition occurs between 9.0 and 10.0 time units, the victim transition has already occurred, at some time between 7.0 and 8.0 time units.

However, with derating applied, the long clock path leading up to the common point has an early arrival at time 6.0 and a late arrival at time 9.0. This widens the aggressor and victim windows to 3.0 time units, causing them to overlap and produce a crosstalk situation where none could actually exist in the real circuit.

To get the best possible accuracy during path-based analysis, you can optionally have CRPR applied during arrival window overlap analysis, thus removing the pessimism caused by different early and late arrival times at the common point leading up to the aggressor and victim. To invoke this option, set the following variable to `true`:

```
pba_enable_xtalk_delay_ocv_pessimism_reduction
```

The default setting is `false`. When the variable is set to `true`, and if CRPR is enabled, during path-based analysis, PrimeTime SI applies CRPR during calculation of aggressor and victim arrival windows, resulting in more accurate arrival windows, at the cost of some

additional runtime. Path-based analysis occurs for the paths reselected with `get_timing_paths -recalculate` or `get_recalculated_timing_paths`. CRPR is enabled when `timing_remove_clock_reconvergence_pessimism` is set to true.

Annotated Delta Delays

Instead of allowing PrimeTime SI to calculate delta delays resulting from crosstalk, you can set delta delay values explicitly with the `set_annotated_delay -net -delta_only` command. This feature lets you annotate delta delays calculated by an external tool and then perform timing analysis in the presence of those delta delays.

For example, to set a delta delay of 0.12 time units on the net arc from output pin U1/Z to input pin U2/A for maximum-delay analysis, you would use the following command:

```
pt_shell> set_annotated_delay -net -max -delta_only \
          -from U1/Z -to U2/A 0.12
```

If you run PrimeTime with crosstalk analysis disabled (`si_enable_anlaysis` set to false), PrimeTime applies the annotated delta delay to the path. However, if you run a crosstalk analysis, a delta delay value calculated by PrimeTime SI overrides any delta delay value set manually with the `set_annotated_delay -delta_only` command.

If you annotate both a delta delay and an overall delay for the same timing arc by using the `set_annotated_delay` command, both with and without the `-delta_only` option, the annotated overall delay is assumed to include any delta delay and the `-delta_only` value is not used.

You can similarly set delta transition times on specified ports or pins with the `set_annotated_transition -delta_only` command. For more information, see the man pages for `set_annotated_delay` and `set_annotated_transition`.

Timing Reports

There are several different commands for generating reports on crosstalk effects:

- `report_timing` generates a slack timing report that includes crosstalk delay effects.
- `report_si_bottleneck` helps determine the major victim nets or aggressor nets that are causing multiple violations.
- `report_delay_calculation -crosstalk` provides detailed information on crosstalk calculations for a particular victim net.

- `report_si_double_switch` helps determine those victim nets with double-switch violations in the design as described in "Fixing Double-Switching Violations" on page 2-44.
- `report_noise` generates a report on static noise effects (noise bumps on quiet victim nets as described in Chapter 6, "Static Noise Analysis."

Crosstalk report_timing Command

When crosstalk analysis is enabled, the report generated by the `report_timing` command shows the path delays, including crosstalk effects. To see detailed crosstalk information in the report, use the `-crosstalk_delta` option with the `report_timing` command. For example:

```
pt_shell> report_timing -transition_time -crosstalk_delta \
          -input_pins -significant_digits 4
```

Using the `-crosstalk_delta` option causes input pins to be displayed in the report, even if you do not use the `-input_pins` option.

Here is an example of a timing report with crosstalk effects:

```
****************************************
Report : timing
        -path full
        -delay max
        -input_pins
        -max_paths 1
        -transition_time
        -crosstalk_delta
Design : diagsys
Version: 2001.08-SI1
Date   : Tue May  1 21:23:27 2001
****************************************


  Startpoint: reset (input port)
  Endpoint: hostif_0/host_data_regx28x
            (recovery check against rising-edge clock clock)
  Path Group: **async_default**
  Path Type: max

  Point                          DTrans   Trans    Delta     Incr       Path
  ------------------------------------------------------------------------------
  clock (input port clock) (rise edge)                      0.0000     0.0000
  input external delay                                      0.0000     0.0000 f
  reset (in)                              0.0000            0.0000     0.0000 f
  U26/A (BF1T2)                  0.0000   0.0066   0.0000   0.0028 &   0.0028 f
  U26/Z (BF1T2)                           1.0368            0.7040 &   0.7068 f
  hostif_0/reset (hostif_mstest_1)        0.0000            0.0000     0.7068 f
  hostif_0/U1836/A (IV)          0.0023   1.0414   0.0087   0.0388 &   0.7456 f
  hostif_0/U1836/Z (IV)                   0.7593            0.5008 &   1.2463 r
```

```
hostif_0/U1835/A (BF1T4)      0.0000   0.7593   0.0000   0.0002 &   1.2466 r
hostif_0/U1835/Z (BF1T4)               1.1855            0.7458 &   1.9924 r
hostif_0/host_data_regx28x/CD (FD2S)
                              0.0000   1.1891   0.0597   0.1036 &  2.0960 r
data arrival time                                                    2.0960

clock clock (rise edge)                0.0000            5.0000     5.0000
clock network delay (ideal)                              0.0000     5.0000
hostif_0/host_data_regx28x/CP (FD2S)                                5.0000 r
library recovery time                                   -0.2470     4.7530
data required time                                                   4.7530
---------------------------------------------------------------------------
data required time                                                   4.7530
data arrival time                                                   -2.0960
---------------------------------------------------------------------------
slack (MET)                                                          2.6571

Startpoint: hostif_0/access_state_regx0x
            (rising edge-triggered flip-flop clocked by clock)
Endpoint: hostif_0/host_address_regx21x
          (rising edge-triggered flip-flop clocked by clock)
Path Group: clock
Path Type: max

Point                          DTrans   Trans    Delta     Incr        Path
---------------------------------------------------------------------------
clock clock (rise edge)                0.0000            0.0000     0.0000
clock network delay (ideal)                              0.0000     0.0000
hostif_0/access_state_regx0x/CP (FD2SP)
                                       0.0000            0.0000     0.0000 r
hostif_0/access_state_regx0x/Q (FD2SP)
                                       0.7233            0.6764 &   0.6764 r
hostif_0/U1752/A (ND2)         0.0000  0.7233   0.0115   0.0164 &   0.6928 r
hostif_0/U1752/Z (ND2)                 0.4238            0.3678 &   1.0606 f
hostif_0/U1751/A (IV)          0.0000  0.4238   0.0114   0.0117 &   1.0722 f
hostif_0/U1751/Z (IV)                  0.2866            0.1890 &   1.2612 r
hostif_0/U2275/C (ND3)         0.0000  0.2866   0.0066   0.0067 &   1.2679 r
hostif_0/U2275/Z (ND3)                 0.4255            0.2220 &   1.4899 f
hostif_0/U2276/A (IV)          0.0000  0.4255   0.0189   0.0191 &   1.5089 f
hostif_0/U2276/Z (IV)                  0.5016            0.2960 &   1.8050 r
hostif_0/U1736/A (ND2)         0.0000  0.5016   0.0389   0.0389 &   1.8439 r
hostif_0/U1736/Z (ND2)                 0.4531            0.3601 &   2.2040 f
hostif_0/U2266/B (NR4X05)      0.0000  0.4531   0.0219   0.0228 &   2.2268 f
hostif_0/U2266/Z (NR4X05)              0.6603            0.3408 &   2.5676 r
hostif_0/U2264/C (AO7CNP)      0.0000  0.6603   0.0200   0.0200 &   2.5875 r
hostif_0/U2264/Z (AO7CNP)              0.2220            0.6179 &   3.2054 f
hostif_0/U2271/C (AO7X05)      0.0000  0.2221   0.0077   0.0085 &   3.2139 f
hostif_0/U2271/Z (AO7X05)              1.7745            0.6765 &   3.8904 r
hostif_0/U2272/A (IVP)         0.0000  1.7745   0.1337   0.1347 &   4.0250 r
hostif_0/U2272/Z (IVP)                 1.0285            0.9720 &   4.9970 f
hostif_0/U1718/B (ND2)         0.0000  1.0286   0.0246   0.0314 &   5.0285 f
hostif_0/U1718/Z (ND2)                 0.9974            0.6208 &   5.6492 r
hostif_0/U1894/A (IV4)         0.0000  0.9974   0.0561   0.0581 &   5.7073 r
hostif_0/U1894/Z (IV4)                 0.4672            0.4603 &   6.1676 f
hostif_0/U1895/A (BF1T4)       0.0000  0.4673   0.0116   0.0155 &   6.1831 f
hostif_0/U1895/Z (BF1T4)               0.4958            0.5413 &   6.7244 f
hostif_0/U1589/B (ND2)         0.0000  0.4968   0.0298   0.0443 &   6.7688 f
hostif_0/U1589/Z (ND2)                 0.2541            0.2019 &   6.9706 r
```

```
  hostif_0/U1781/A (ND4X05)     0.0000   0.2541   0.0413   0.0414 &   7.0120 r
  hostif_0/U1781/Z (ND4X05)              0.8272            0.4066 &   7.4186 f
  hostif_0/U2044/A (MUX21)      0.0000   0.8272   0.6563   0.6568 &   8.0754 f
  hostif_0/U2044/Z (MUX21)               0.1661            0.4333 &   8.5087 f
  hostif_0/host_address_regx21x/D (FD2S)
                                0.0000   0.1661   0.0120   0.0121 &   8.5208 f
  data arrival time                                                    8.5208

  clock clock (rise edge)                0.0000            5.0000     5.0000
  clock network delay (ideal)                              0.0000     5.0000
  hostif_0/host_address_regx21x/CP (FD2S)                                5.0000 r
  library setup time                                      -0.3633     4.6367
  data required time                                                   4.6367
  ------------------------------------------------------------------------------
  data required time                                                   4.6367
  data arrival time                                                   -8.5208
  ------------------------------------------------------------------------------
  slack (VIOLATED)                                                    -3.8841
```

1

PrimeTime SI calculates and reports crosstalk effects on a per-stage basis. A stage consists of one cell together with its fanout net. PrimeTime SI stores all the delta delay and delta slew per-stage values on the path's input pin.

The report contains columns labeled Dtrans (delta transition) and Delta (delta delay) to show the contribution of crosstalk to the delay per stage in the path. The column labeled Incr (increment) already includes the calculated delta delay. An ampersand character (&) in the Incr column indicates the presence of parasitic data.

You can customize your reports by using a Tcl script that comes with PrimeTime SI. The Tcl script for customizing reports is

install_path/auxx/pt/examples/tcl/custom_timing_1.tcl

Bottleneck Reports

If the `report_timing` command reports a large number of crosstalk violations, the `report_si_bottleneck` command can help determine the major victim nets or aggressor nets that are causing multiple violations, in the same way that the `report_bottleneck` command determines the causes of multiple min/max delay violations.

The `report_si_bottleneck` command reports the nets having the highest "cost function," or highest contribution to undesirable crosstalk effects that cause timing violations. You can choose any one of four different cost functions:

- `delta_delay` – lists the victim nets having the largest absolute delta delay, among all victim nets with less than a specified slack
- `delta_delay_ratio` – lists the victim nets having the largest delta delay relative to stage delay, among all victim nets with less than a specified slack

- `total_victim_delay_bump` – lists the victim nets having the largest sum of all unfiltered bump heights (as determined by the net attribute `si_xtalk_bumps`), irrespective of delta delay, among all victim nets with less than a specified slack
- `delay_bump_per_aggressor` – lists the aggressor nets that cause crosstalk delay bumps on victim nets, listed in order according to the sum of all crosstalk delay bumps induced on affected victim nets, counting only those victim nets having less than a specified slack

By default, the specified slack level is zero, which means that costs are associated with timing violations only. If there are no violations, there are no costs and the command does not return any nets. To get a more extensive report, you can set the slack threshold to a larger value. For example, to get a list of all the victim nets with a delay violation or within 2.0 time units of a violation, listed in order of delta delay:

```
pt_shell> report_si_bottleneck -cost_type delta_delay \
            -slack_lesser_than 2.0
```

Unless you specify either `-max` or `min`, the command considers both maximum-delay (setup) and maximum-delay (hold) constraints.

Nets reported by the bottleneck command can be targeted for further investigation with the `report_delay_calculation -crosstalk`. If you find that the reported violations are valid, you can use command and “what-if” repair commands such as `size_cell` and `set_coupling_separation` to see the effects of different repair strategies.

By default, clock nets are not included in the bottleneck analysis because there are no slack values associated with those nets. However, you can include clock nets in the analysis by using the `-include_clock_nets` option.

You might want to investigate situations where many aggressors affect a single net. To get a bottleneck report that only includes these situations, use the `-minimun_active_aggressor` option. For example:

```
pt_shell> report_si_bottleneck -cost_type delta_delay \
            -minimun_active_aggressor 3
```

In this example, the bottleneck command only reports nets where three or more active aggressors are affecting the net.

For more information, see the man page for the `report_si_bottleneck` command.

Crosstalk Net Delay Calculation

You can use the `report_delay_calculation` command with the `-crosstalk` option to get detailed information about crosstalk calculations done by PrimeTime SI for a particular victim net. The command lists the aggressor nets and describes how they affect the victim net.

The `-crosstalk` option can be used only with a net timing arc, not a cell timing arc. To get information about a victim net, specify the net driver pin and load pin as in this example:

```
pt_shell> report_delay_calculation -crosstalk \
            -from [get_pins g1/Z] -to [get_pins g2/A]
```

The command reports the following information:

- The number of cross-coupled aggressor and active aggressors remaining after filtering
- Victim analysis information such as active/inactive status and reselection status
- Detailed information on each active aggressor such as bump height and window alignment status
- Reasons for inactive aggressor status such as filtering, case analysis, or bump height too small
- Delta delay and delta slew (reported with or without the -crosstalk option)

  Note:
  In PrimeTime SI, the delta slew for setup analysis is always positive or zero, and the delta slew for hold analysis is always negative or zero.

For more information, see the man page for the `report_delay_calculation` command.

Reporting Crosstalk Settings

If you want to check your crosstalk settings, you can use the following commands:

```
report_si_delay_analysis
report_si_noise_analysis
report_si_aggressor_exclusion
```

The `report_si_delay_analysis` command reports the crosstalk settings made with the following delay analysis commands:

```
set_si_delay_analysis
remove_si_delay_analysis
set_si_delay_disable_statistical
remove_si_delay_disable_statistical
```

```
set_coupling_separation
remove_coupling_separation
```

Similarly, the `report_si_noise_analysis` command reports the crosstalk settings made with the following noise analysis commands:

```
set_si_noise_analysis
remove_si_noise_analysis
set_si_noise_disable_statistical
remove_si_noise_disable_statistical
set_coupling_separation
remove_coupling_separation
```

By default, crosstalk settings for all nets in the design are reported. If you provide a list of one or more nets, the commands report the crosstalk settings applied directly to those nets.

The commands `report_si_delay_analysis` and `report_si_noise_analysis` do not trigger a timing or noise update because they only report (not change) existing attributes in the design.

For net-specific reporting, only crosstalk settings directly applied to the specified net are reported. This does not include global coupling separations or exclusions on other nets which implicitly affect the specified net. For example, consider a design where net n1 couples to nets n2 and n3. If you apply the following global coupling separation to net n1, the bump from n1 is disabled in the delay calculation reports for nets n2 and n3:

```
pt_shell> set_coupling_separation n1
```

However, if you request a report for net n2, no crosstalk settings are reported because the global coupling separation was applied to n1, and affects n2 implicitly:

```
pt_shell> report_si_delay_analysis n2
```

If desired, the crosstalk settings can be reported for all nets coupled to n2:

```
pt_shell> report_si_delay_analysis [get_attribute \
          [get_nets n2] effective_aggressors]
```

If the coupling separation was explicitly applied between n1 and n2 using the `-pairwise` option, then the crosstalk constraint would be shown in the reports for n2 and n3:

```
pt_shell> set_coupling_separation n1 -pairwise {n2 n3}
```

The `report_si_aggressor_exclusion` command reports the crosstalk settings made with the command

```
pt_shell> report_si_aggressor_exclusion
```

Net-specific reporting for the command `report_si_aggressor_exclusion` reports all exclusive groups to which the specific net belongs. To find out which of the aggressors were considered active and which of them were screened as quiet, you can use the `report_delay_calculation` and `report_noise_calculation` commands, respectively, for crosstalk delay and noise analysis. Note that the different aggressor nets from the same exclusive group can be active during different types of analysis based on their coupling information and worst-case alignment.

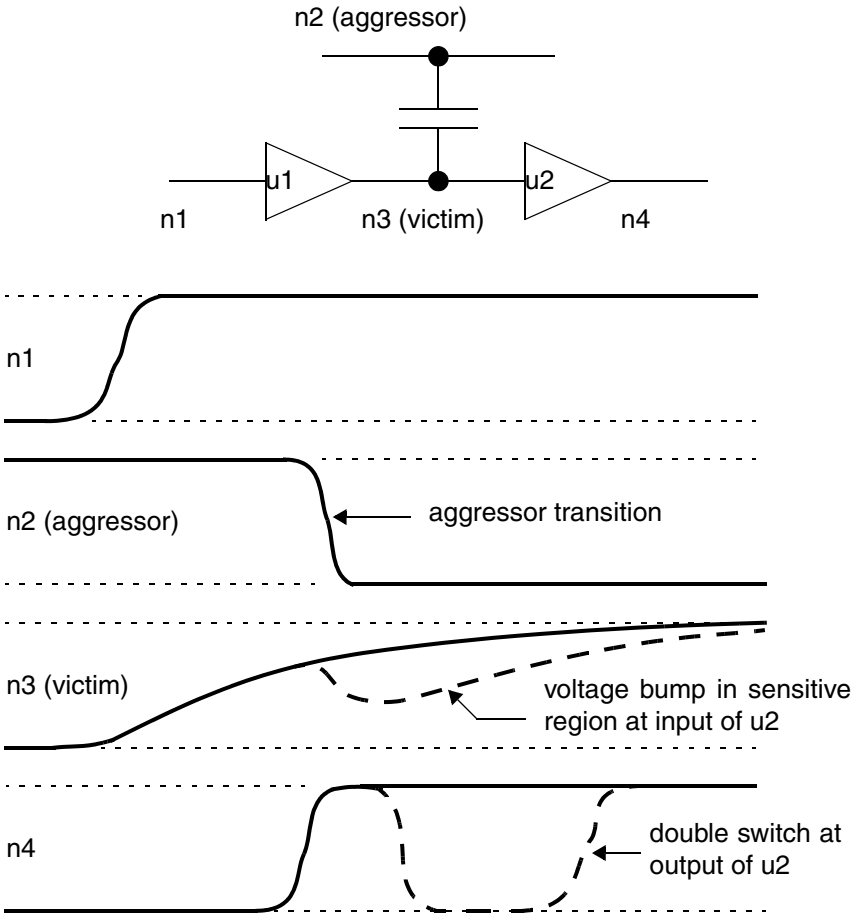
For more information about these commands, refer to their man pages.

Double-Switching Detection

When you use the `update_timing` command, PrimeTime SI detects timing violations resulting from the effects of crosstalk on transitions occurring on victim nets. The slowdown or speedup of a transition can trigger a setup or hold timing violation. When you use the `update_noise` command, PrimeTime SI detects functional errors resulting from the effects of crosstalk on steady-state nets. A large noise bump on a steady-state net can cause an incorrect logic value to be propagated.

In addition to crosstalk timing errors and steady-state functional errors, PrimeTime SI can also detect functional errors resulting from crosstalk effects on switching victim net. These types of errors are called double-switching errors. An example is shown in Figure 2-14.

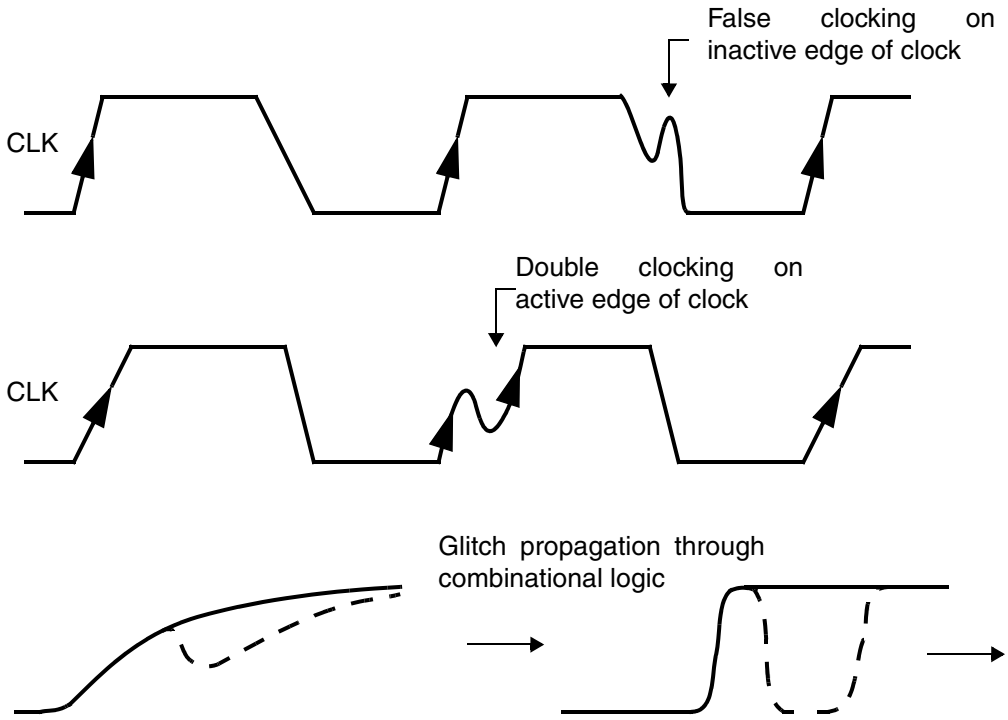
Figure 2-14 Double-Switching Error Example



In this example, a rising transition on net n1 is propagated through buffers u1 and u2 to nets n3 and n4. Because of the low drive of buffer u1 and the capacitive load of net n3, the transition on n3 is relatively slow. In the presence of crosstalk (indicated by the dashed lines in the figure), an aggressor transition causes a voltage bump in the sensitive voltage region at the input of buffer u2. This causes the output of the buffer to switch twice.

Double-switching errors such as this can cause incorrect circuit operation by *false clocking* on the inactive edge of a clock signal, by *double clocking* on the active edge of a clock signal, or glitch propagation through combinational logic. These effects are illustrated in Figure 2-15. The false clocking and double clocking examples cause undesired clocking of rising-edge-sensitive registers.

Figure 2-15 Undesirable Effects of Double-Switching Errors



Double-switching errors are caused by crosstalk coupling capacitance between a strong aggressor transition acting on a sensitive victim. These are the same conditions that cause static noise errors on steady-state nets, so most double-switching cases are detected by static noise analysis (`update_noise`). However, there can cases where crosstalk effects are large enough to cause double-switching errors, but not large enough to cause steady-state functional failures. These double-switching cases are not detected by `update_noise`.

Invoking Double-Switching Error Detection

To enable double-switching error detection, set the `si_xtalk_double_switching_mode` variable to either `clock_network` or `full_design`. Setting the variable to `clock_network` checks for false clocking, double clocking, and double switching in the clock network only. Setting the value to `full_design` checks these same conditions in the data paths as well as in the clock network.

This is the recommend procedure for double-switching error detection.

- Perform static timing analysis without crosstalk. Ensure that there are no timing violations and maximum transition violations.

- Perform static timing analysis and static noise analysis with crosstalk. Fix any reported crosstalk timing and noise violations.
- Enable double-switching crosstalk analysis and perform static timing analysis. Fix any reported double-switching errors.

Remember that most double-switching error conditions are detected by ordinary static noise analysis.

PrimeTime SI detects double-switching during `update_timing` and marks the net attributes of the nets that have double-switching. To get the attribute information you can use:

```
pt_shell> get_attribute [get_net clk_net] \
          si_has_double_switching
```

You can use this attribute in a script to find and fix the double-switching conditions detected in the design.

How Double-Switching Is Detected

There are several different conditions that affect the determination of double-switching, including the cross-capacitance value, the aggressor and victim drive characteristics, the aggressor arrival time and direction (rise or fall) with respect to the victim transition, the load coupling capacitance of the victim net, and the input sensitivity of the cells driven by the victim net.

PrimeTime SI uses the CCS noise model to propagate the coupled waveform for each stage and to check for potential double-switching. Therefore, in order to perform double-switching error detection, the design must use a cell library with CCS noise models. Double-switching detection is done for first and subsequent iterations of crosstalk analysis.

For each detected double-switching functional failure, PrimeTime SI sets the net attribute `si_has_double_switching` to true. It also measures the severity of the double-switching error and reports it as double-switching slack. A victim with a higher risk of double-switching is reported to have a more negative slack. The cases with the worst double-switching slack should be fixed first in an engineering change order (ECO).

The slack value is stored in the net attribute `si_double_switching_slack`. For nets without any double-switching error, this attribute is set to POSITIVE. If there is no CCS noise model available in the library, the switching bump is unconstrained and the slack attribute is INFINITY.

Reporting Double-Switching Violations

After a timing update with crosstalk analysis and double-switching error detection enabled, you can report the presence of double-switching errors detected in the design. Use the `report_si_double_switching` command to report all the victim nets that have double-switching. Note that many of these victims could also cause noise violations. The syntax for the command is

```
report_si_double_switching
  [-clock_network]
  [-rise]
  [-fall]
  [-nosplit]
  [nets]
```

The `-clock_network` option reports the double-switching violations for the clock network only. This option is independent of the `si_xtalk_double_switching_mode` variable setting being either `clock_network` or `full_design`.

Use the `-rise` option to report double-switching violations for rising victims only, or the `-fall` option to report falling victims only. Specify a list of nets to check only those nets. Otherwise, the whole clock network or the full design is checked, depending on the variable `si_xtalk_double_switching_mode`.

Here is an example of a double-switching report:

```
pt_shell> report_si_double_switching -nosplit
...
Victim Switching Actual      Required    Double Switching
Net    Direction Bump Height Bump Height Slack
~~~~   ~~~~~~~~~ ~~~~~~~~~~~ ~~~~~~~~~~~ ~~~~~~~~~~~~~~~~
I2     max_rise  0.69        0.42        -0.26 (Violating)
I1     max_fall  0.50        0.30        -0.20 (Violating)
```

This design has two potential double-switching errors. Net I2 has higher chance of having a double-switching error and should have a higher priority for fixing.

The command `report_delay_calculation -crosstalk` shows whether there are double-switching violations on a victim net, as long as the `si_xtalk_double_switching_mode` variable is set to either `clock_network` or `full_design`.

Fixing Double-Switching Violations

There are several ways to fix double-switching violations. For example, you can decrease the cross-capacitance by spacing or shielding the adjacent wires, or upsize the victim net driver to reduce the transition time.

The ECO fixing command `create_eco_astro_constraints` can be used to generate a script that supplies fixing constraints for Astro to reroute the design. The script has an option, `-constraint_type double_switching`, that generates constraints for fixing all the detected double-switching violations.

Fixing Crosstalk Violations

When PrimeTime SI finds a crosstalk violation, you need to correct the violation using a layout tool such as Synopsys IC Compiler. The repair flow is typically an iterative process. For example, you can make some layout changes to correct the crosstalk violations, and then send the updated SPEF and Verilog data to PrimeTime SI to verify that the problems are corrected, and that there are no new problems. If PrimeTime SI finds any violations in the modified design, the process must be repeated.

For a faster repair flow, you can perform "what-if" analysis of certain design changes entirely within PrimeTime SI. For analyzing these changes, PrimeTime SI uses a fast "incremental" analysis, taking just a fraction of the time needed for a full analysis, because it needs to update only the portion of the design affected by the changes.

"What If" Incremental Analysis

To correct crosstalk violations, you can increase the drive strength of victim nets by increasing the sizes of the driving cells using `size_cell` or by inserting buffers using `insert_buffer`. Another technique is to move apart adjacent victim/aggressor nets with the `set_coupling_separation` command. If there are any other custom netlist operations that you would like to perform, they can be done by removing the cell and reconnecting a new cell using low-level commands such as `connect_net`, `disconnect_net`, `create_cell`, and so on.

These are the types of changes that are compatible with incremental analysis:

- `size_cell`
- `insert_buffer`, `remove_buffer`
- `set_coupling_separation`, `remove_coupling_separation`
- `connect_net`, `disconnect_net`, `remove_net`
- `create_cell`, `remove_cell`

If there are any other changes, `update_timing` performs a full (not incremental) timing update.

For an incremental update, PrimeTime SI does the following:

1. Identifies the nets that are directly affected by the “what-if” changes, and their aggressors.
2. Performs electrical filtering of the affected nets.
3. Performs the first update iteration on the fanout cone of the affected nets, with the depth of the update cone limited to changes in slew.
4. Performs the second and any subsequent iterations using standard reselection criteria, but only on the nets in the affected cone.

The number of crosstalk analysis iterations performed in incremental mode is determined separately from full crosstalk iterations, using the following variable:

```
si_xtalk_exit_on_max_iteration_count_incr
```

This variable can be set to a positive integer that specifies the maximum number of incremental iterations to perform.

After you decide on a set of changes, you can then export those changes to IC Compiler for physical implementation with the `write_changes` command. IC Compiler can read and use the generated data for modifying the design.

The results of a “what-if” crosstalk analysis can be slightly different from a full analysis because of the iterative nature of the analysis and the interdependence of timing windows and crosstalk delay results. For final sign-off of the design, you should perform a full analysis using SPEF/Verilog data from IC Compiler or another layout tool.

Generating ECO Fixing Constraints

When performing final sign-off static timing analysis, a few timing violations or crosstalk issues might remain that you need to fix. You must go back to place-and-route and fix these violations. It is possible that the implementation tool cannot identify these violations. In that case, you need to manually intervene and make the necessary fixes. This engineering change order (ECO) fixing often requires multiple iterations between the place-and-route tool and PrimeTime to converge on a solution that removes all violations.

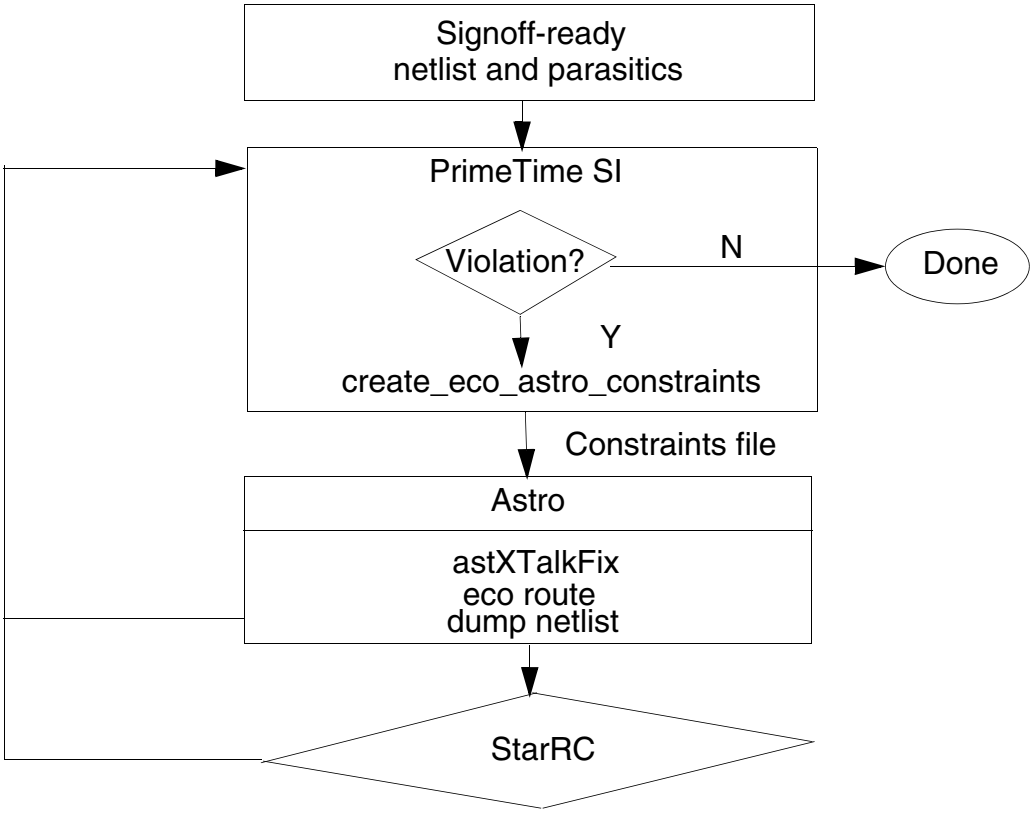
In order to address this issue efficiently, PrimeTime SI generates ECO fixing constraints that can be read by IC Compiler and used to direct optimization. Using these constraints, IC Compiler can fix timing violations very efficiently, resulting in faster closure. This is the general procedure:

- PrimeTime SI identifies the nets that need fixing and provides timing and target timing (constraint) information to IC Compiler.

- IC Compiler, which has all the physical information to determine the best way to fix the problem, has full flexibility to do so.

By addressing these issues as a set of constraints, fewer iterations are required and little, if any, intervention is required on your part. This should reduce overall turnaround time to timing closure. The flow is illustrated in Figure 2-16. You can use the flow to resolve both crosstalk-related problems and related violations.

Figure 2-16 ECO Fixing Flow



PrimeTime SI examines paths with timing violations, timing bottlenecks, and crosstalk violations at the same time. Then it performs intersection analysis on those paths, identifies which nets to fix, and creates a minimum number of constraints while satisfying all the fixing requirements for IC Compiler.

For this purpose, the `create_eco_astro_constraints` command works in conjunction with the astXTalkFix feature in IC Compiler. To use this feature in PrimeTime SI, you issue the command:

```
pt_shell> create_eco_astro_constraints \
          -output astro_eco.cst
```

The `create_eco_astro_constraints` command has the following syntax:

```
create_eco_astro_constraints
  [-output file_name]
  [-constraint_type constraint_type_list]
  [-delay_type max|min|min_max]
  [-effort_level low|high]
  [-validate]
  [-group path_group_name]
  [-max_constraints number_of_constraints]
  [-max_iteration number_of_iterations]
  [-verbose]
  [object_list]
```

This command generates constraints, by default giving the highest priority to delta delay reduction and stage reduction. It first tries to fix timing by focusing on reducing crosstalk along the violating paths. If this doesn't fix the timing, it generates stage delay reduction constraints. The flow can also target fixing non-crosstalk issues if they exist.

You can check the new timing with the generated constraints by using the `-validate` option. PrimeTime SI reads the constraints back in and runs an incremental timing update to see how much timing improvement has been achieved. This option is especially useful when you want to check your ECO fix before place-and-route.

To create constraints for specific clock domains, use the `-group` option. To limit the number of constraints generated for IC Compiler, use the `-max_constraint` option. You can use the `-delay_type` option to generate constraints for setup, hold, or both. For more information, see the `create_eco_astro_constraints` command man page.

This flow typically gives much more predictable results, with less effort and manual intervention, than manual ECO fixing. After performing the ECO using the generated constraint file as input, there should be improvement in slack or number of violations.

You can capture results of `report_timing` and `report_constraints` to determine worst negative slack, total negative slack, and number violations from the initial or baseline sign-off runs. After generating constraints, executing the ECO in IC Compiler and generating new parasitics, you can then rerun PrimeTime SI using the new netlist and parasitics and capture the same data.

The following example shows constraint generation with validation. With validation, you can predict what the result will be if all of the constraints are met. PrimeTime SI performs validation by annotating the target or constraint timing onto the design and running an incremental timing update.

```
restore_session PT_sessions/initial_result.session
create_eco_astro_constraints  -validate -output \
 const/eco1.cst
report_constraint
```

```
report_timing
save_session -replace PT_sessions/eco1.session
exit
```

This next example limits constraints to 50 and generates constraints only for myclk path group.

```
restore_session PT_sessions/initial_result.session
create_eco_astro-constraints -constraint_type delta_delay \
  -validate -max_constraints 50 \
  -group myclk -output const/eco2.cst
report_constraint
report_timing -group myclk
save_session -replace PT_sessions/eco2.session
exit
```

For reference, you can use the following IC Compiler information. These are the astTalkFix options.

```
setFormField "XTalk Fix" "Do From File" "1"
setFormField "XTalk Fix" "Sizing Only" "0"
setFormField "XTalk Fix" "User Specified Noise" "1"
setFormField "XTalk Fix" "Preserve Setup" "1"
setFormField "XTalk Fix" "Preserve Hold" "1"
setFormField "XTalk Fix" "Preserve TransCap" "1"
setFormField "XTalk Fix" "From File Name" ""
setFormField "XTalk Fix" "From File Name" "const/eco1.cst"
```

If sizing only is selected, buffer insertion is not done during crosstalk fixing.

ECO Estimation

The `estimate_eco` command quickly estimates timing changes for a path resulting from cell sizing and buffer insertion, without running a timing update, thus helping you choose the best cell size or buffer to use for fixing a particular timing violation. The command lists the available cell sizes to replace a given cell, or the available buffer cells to insert at a pariticular pin, and shows an estimate of the new delay, arrival, and slack values at the path stage for each choice. The timing estimation includes crosstalk effects. After you choose the cell size or buffer, you can then use the `size_cell` or `insert_buffer` command to make the change in the netlist and run an incremental timing update to determine the full effects of the change.

For more information about using the `estimate_eco` command, see the *PrimeTime Advanced Timing Analysis User Guide*.

3

Graphical User Interface

The PrimeTime SI graphical user interface (GUI) displays crosstalk analysis results as described in the following sections:

- Analysis Flow Using the GUI
- Crosstalk GUI Analysis
- Noise GUI Analysis

Analysis Flow Using the GUI

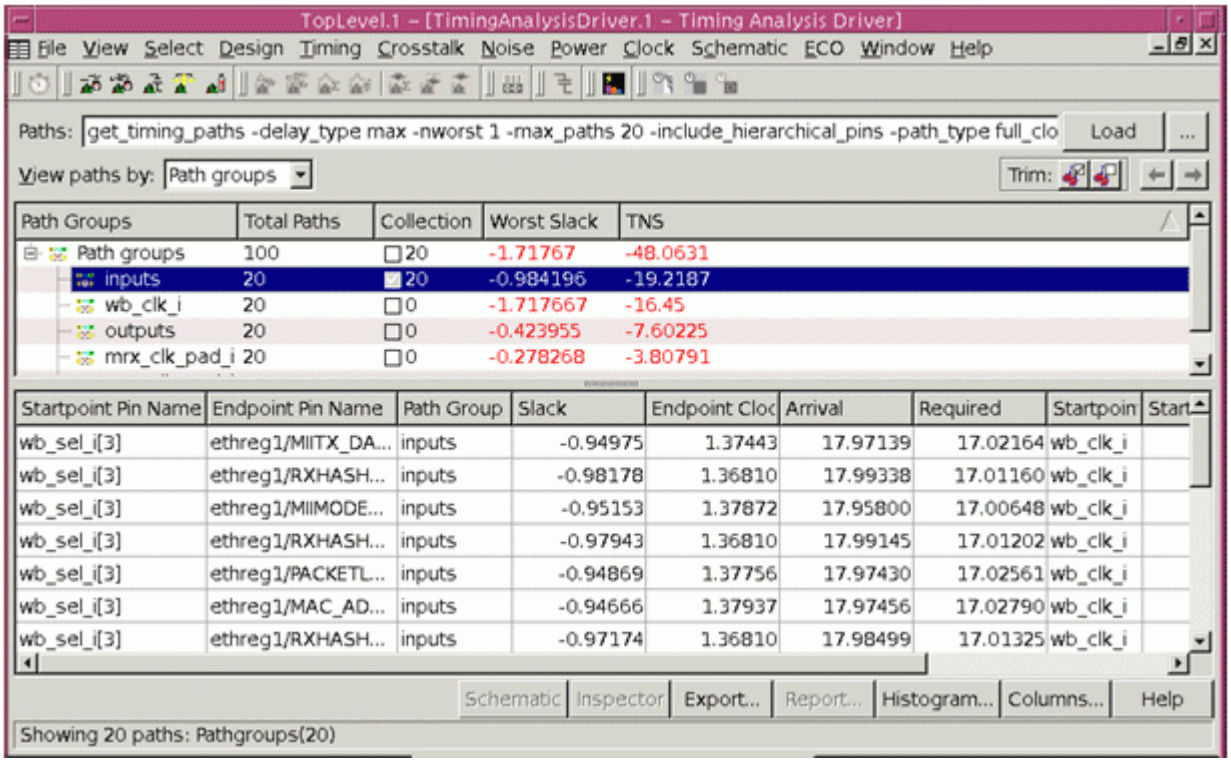
You can use the graphical user interface (GUI) to display the crosstalk analysis results in histogram form. For example, a typical analysis session might use the following steps:

1. Read in the design, enable crosstalk analysis, read in the parasitic data, and set the crosstalk analysis parameters.
2. Perform a timing analysis with `report_timing` or `report_noise`.
3. Generate histograms for endpoint slack, delta delay, bump voltage, noise slack, and noise bump.
4. In the timing path table, select the worst-case path and click [Inspector] to open the path inspector window.
5. In the path inspector window, click the Data Path tab to view the path element table. The table has columns showing delta transition times and delta delays along the path.
6. Select the victim net and perform a coupling analysis using Crosstalk > Coupling Analysis for Selected Nets.
7. Generate noise histograms using Noise > Noise Slack Histogram.
8. Check noise bumps against noise immunity curves using Noise > Noise Immunity Curve.
9. Perform crosstalk analysis using Crosstalk > Coupling Analysis for Selected Nets or Crosstalk > Coupling Analysis for SI Bottleneck Nets.

Note:

Some reports and histogram windows use the term “effective” to describe a victim net, aggressor net, or capacitor. This means an object that has not been removed by filtering and has been selected or reselected for crosstalk analysis.

Figure 3-1 Top-Level PrimeTime SI GUI Window



Crosstalk GUI Analysis

These are the commands in the Crosstalk menu of the top-level GUI window:

- Crosstalk > Delta Delay Histogram: displays a histogram of delta delay values induced on victim nets in the design
- Crosstalk > Path Delta Delay Histogram: displays a histogram of delta delay values induced on victim nets in one or more selected paths
- Crosstalk > Bump Voltage Histogram: displays a histogram of individual voltage bumps induced on one victim net by multiple aggressor nets
- Crosstalk > Accumulated Bump Voltage Histogram: displays a histogram of the accumulated voltage bumps induced on victim nets by multiple aggressor nets
- Crosstalk > Coupling Analysis For Selected Paths: displays a table of victim nets, cross-coupled aggressor nets, and crosstalk information for nets in one or more selected paths

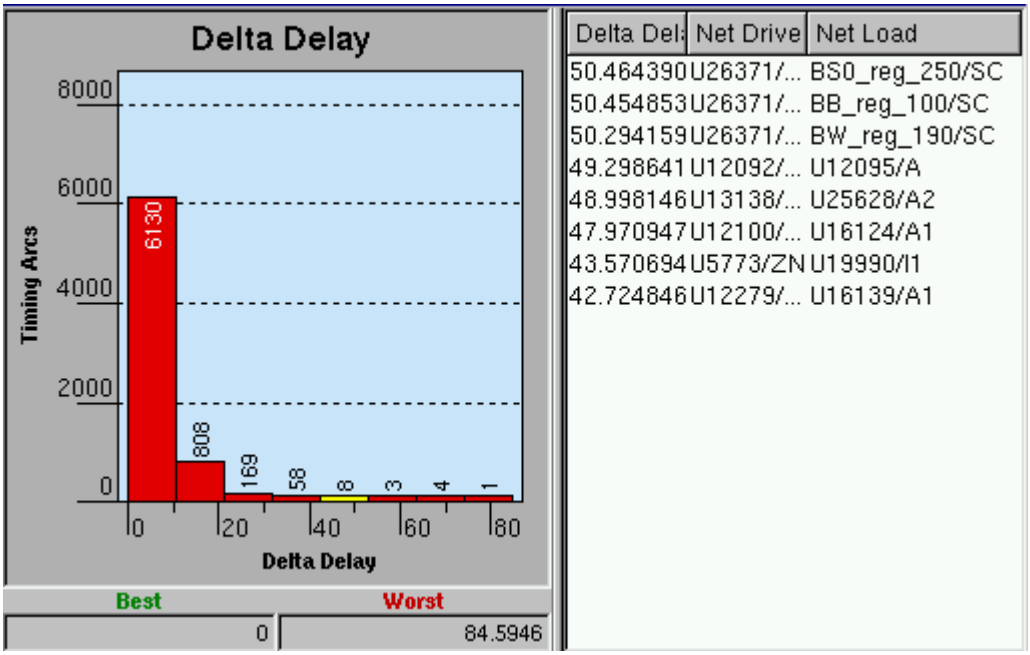
- Crosstalk > Coupling Analysis For Selected Nets: displays a table of victim nets, cross-coupled aggressor nets, and crosstalk information for nets in one or more selected nets
- Crosstalk > Coupling Analysis For SI Bottleneck Nets: displays a bottleneck report in table format

Delta Delay Histogram

A delta delay histogram shows the distribution delay change induced on victim nets by aggressor nets. Figure 3-2 is a typical delta delay histogram. In this example, the highest histogram bar is selected and highlighted. The net arcs contained in that bin are listed on the right, along with the corresponding delta delay values.

To generate a delta delay histogram, choose Crosstalk > Delta Delay Histogram or click the equivalent button in the histogram toolbar. In the Delta Delay dialog box, specify the number of bins and the types of delta delay to be included in the histogram plot. Then click OK to generate the plot.

Figure 3-2 Delta Delay Histogram



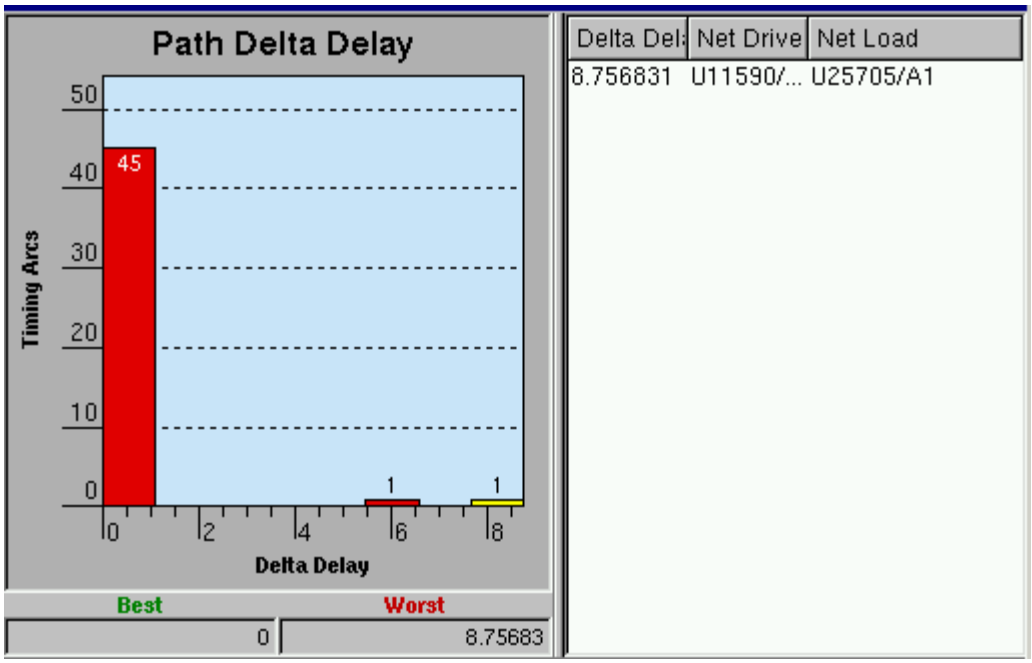
By default, all delta delays are included in the histogram plot. You can restrict the types of data included in the plot by specifying the desired delta delay range, slack range, delay type (minimum or maximum), and transition type (rising or falling).

Path Delta Delay Histogram

From a slack histogram, you can select one or more paths and view a histogram of the delta delays of nets along those paths. See Figure 3-3 for an example of a path delta delay histogram.

To generate a path delta delay histogram, you must first select a path. Starting from a path slack histogram, you can simply select the path from the list on the right. Another method is to specify the path by using from/through/to or slack criteria with the Select Paths dialog box (choose Select > Paths From/Through/To).

Figure 3-3 Path Delta Delay Histogram



If you are starting from an endpoint slack histogram, first select the endpoint of interest. Then choose Select > Paths. In the Select Paths dialog box, fill in the To field by selecting “port” or “pin” and then clicking Selection[3], and specify the other path selection criteria such as Nworst paths. Then click OK to select the path.

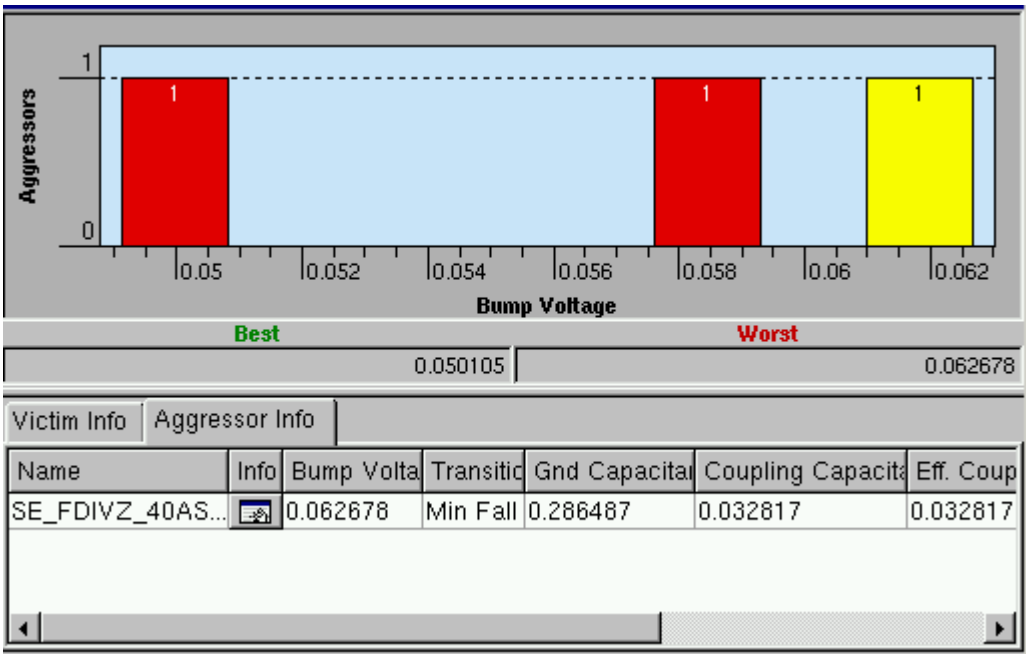
After you select the path, to generate the path delta delay histogram, choose Crosstalk > Delta Delay Histogram or click the equivalent toolbar button. In the Path Delta Delay dialog box, specify the number of bins and the types of delta delay to be included in the histogram plot. You can optionally restrict the range of delta delays and timing arc pin slacks to be included in the plot. Then click OK to generate the plot.

In the path delta delay histogram, to examine the cause of a delta delay on a particular net, select the histogram bar, select the net of interest from the list on the right, and then choose Crosstalk > Bump Voltage Histogram.

Bump Voltage Histogram

A plain (not “accumulated”) bump voltage histogram shows the distribution of individual voltage bumps induced on one victim net by multiple aggressor nets. Figure 3-4 shows an example.

Figure 3-4 Bump Voltage Histogram With Victim Info Spreadsheet



These bumps are used to calculate delta delay effects, not static noise effects. To display histograms of static noise bump data, use a command from the Noise menu.

To generate a bump voltage histogram for a net, first select the net. You can use Select > By Name or one of the Select > Nets commands, or you can select the net in a schematic window, path profile window, or histogram window. Then choose Crosstalk > Bump Voltage Histogram or click the equivalent toolbar button. In the dialog box, specify the number of bins and the types of voltage bumps to include in the plot: max_rise, max_fall, min_rise, and min_fall. Then click OK to generate the plot.

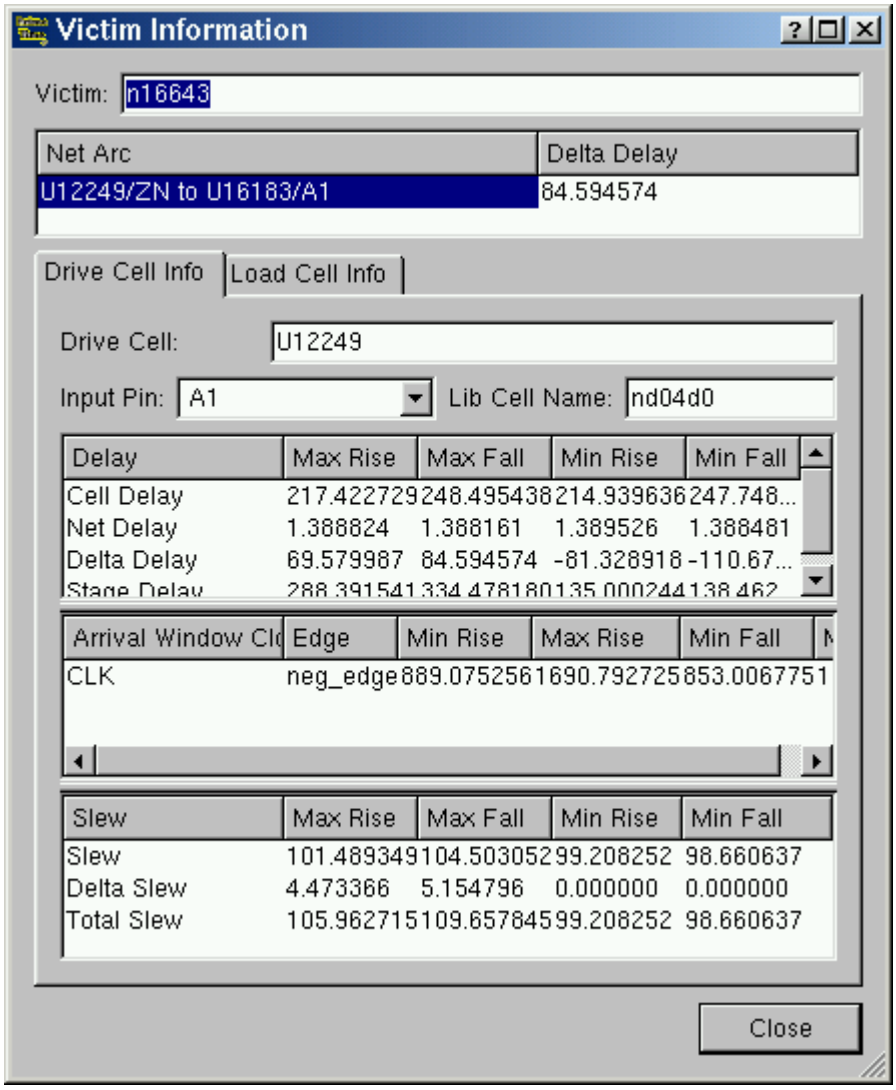
The histogram window has two tabbed pages at the bottom, labeled Victim Info and Aggressor Info. Clicking the Victim Info tab displays a spreadsheet containing detailed information on the victim net. Clicking the Aggressor Info tab displays a spreadsheet with information on all aggressor nets in the currently selected bin.

The Victim Info spreadsheet shows detailed information about the victim net, including the net name, ground capacitance, coupling capacitance, aggressor information, delta delay, and delta slew.

The Aggressor Info spreadsheet shows information about each aggressor net in the currently selected bin, including the net name, bump voltage contribution, ground capacitance, coupling capacitance, and filtering status. All bump voltages are shown as a fraction of Vdd, not an absolute number of volts.

To open a dialog box showing detailed information on the victim net, in the Victim Info spreadsheet, click the Info button next to the victim net name. Figure 3-5 shows an example of a Victim Information dialog box.

Figure 3-5 Victim Information Dialog Box



Similarly, to open a dialog box showing detailed information on an aggressor net, click the Info button next to the aggressor net name in the Aggressor Info spreadsheet.

You must close the Victim Information or Aggressor Information dialog box before you can go back to using the main GUI window.

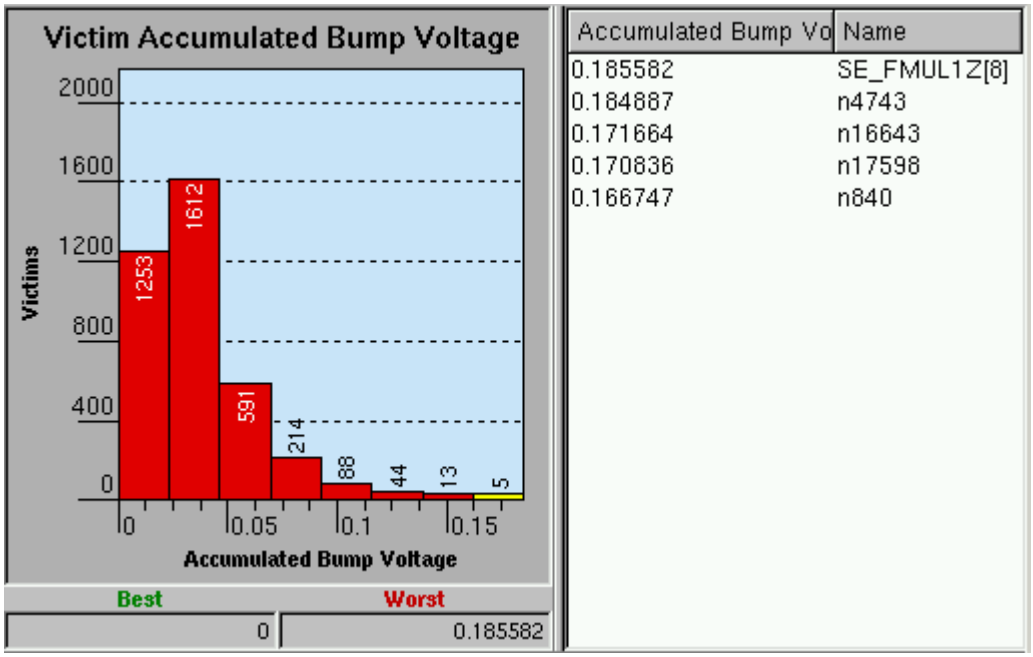
Accumulated Bump Voltage Histogram

An accumulated bump voltage histogram shows the distribution of the total voltage bumps induced on victim net arcs. The word “accumulated” means the total of all voltage bumps induced on a victim net by multiple aggressor nets.

These bumps are used to calculate delta delay effects, not static noise effects. To display histograms of static noise bump data, use a histogram menu command from the Noise menu.

Figure 3-6 shows a typical accumulated voltage bump histogram. The nets contained in the selected bin are listed on the right, along with the corresponding accumulated bump voltage values.

Figure 3-6 Accumulated Bump Voltage Histogram

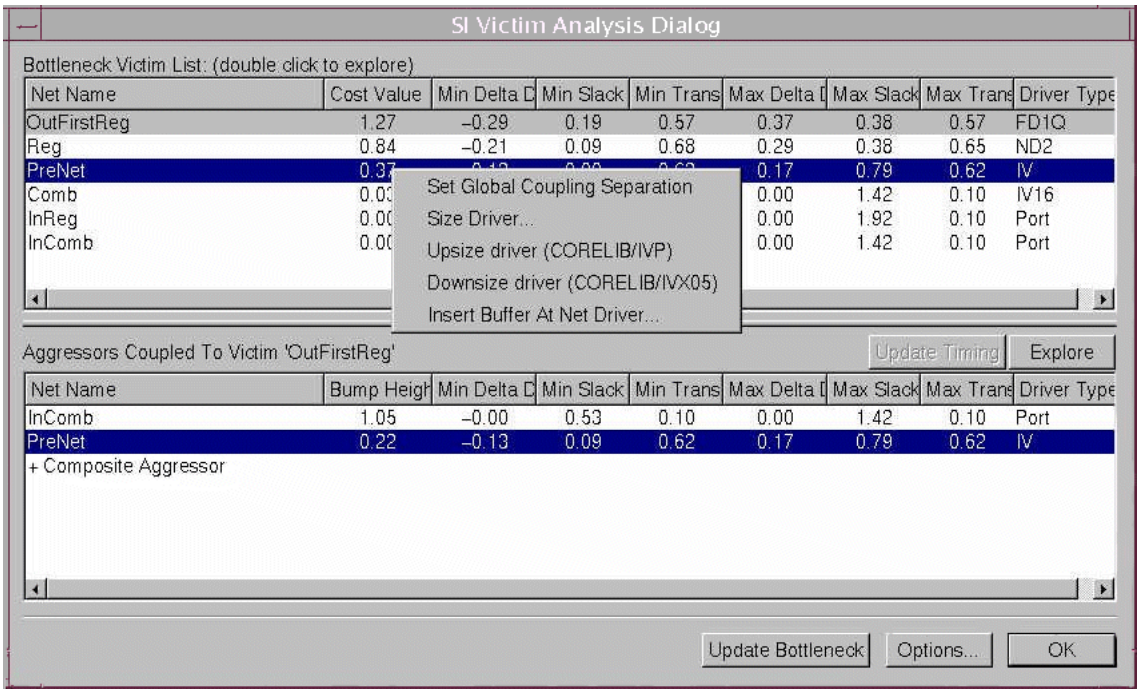


To generate an accumulated bump voltage bump histogram, choose Crosstalk > Accumulated Bump Voltage Histogram or click the equivalent toolbar button. In the dialog box, specify the number of bins and the types of accumulated voltage bumps to include in the plot: max_rise, max_fall, min_rise, and min_fall. Then click OK to generate the plot.

Crosstalk Coupling Analysis

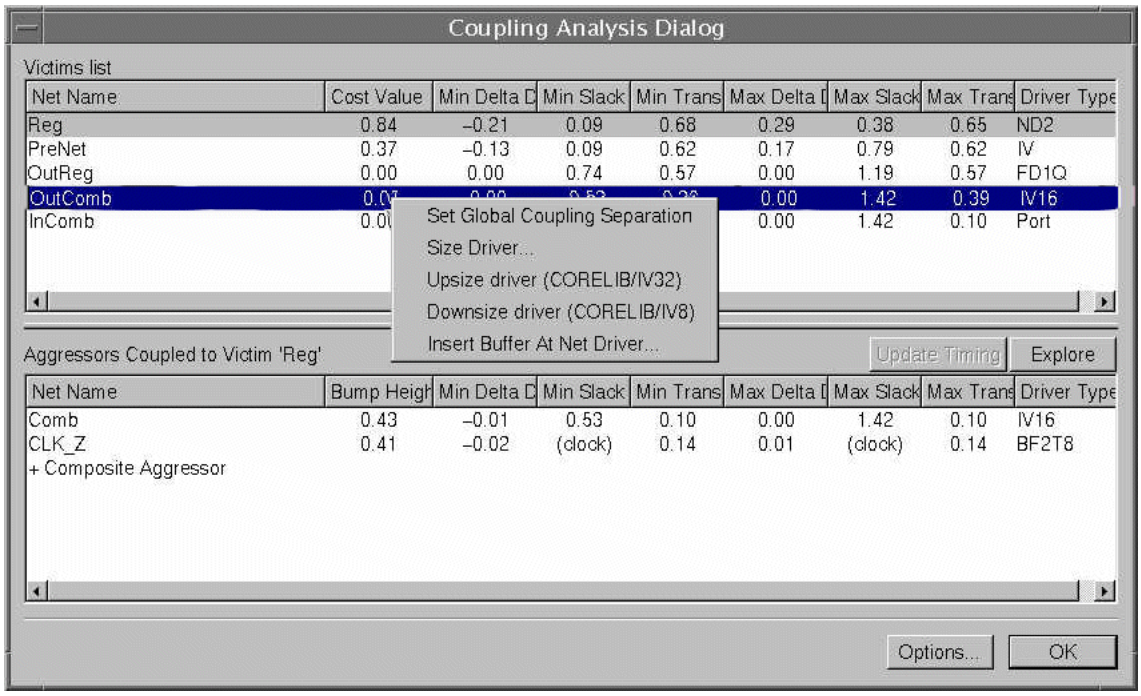
You can perform crosstalk coupling analysis on SI bottlenecks, where given nets contribute to multiple crosstalk violations. To find crosstalk bottlenecks, choose Crosstalk > Coupling Analysis for SI Bottleneck Nets. A dialog box lets you set the analysis options. This is the same a running the `report_si_bottleneck` command. The results appear in a new window, the SI Victim Analysis Dialog. See Figure 3-7.

Figure 3-7 SI Victim Analysis Table



For an analysis on a specified collection of paths or nets, first select the paths or nets, using any of various methods such Select > By Name or by selecting the paths or nets from existing windows. Then choose Crosstalk > Coupling Analysis for Selected Paths or Crosstalk > Coupling Analysis for Selected Nets. The Coupling Analysis Dialog is then filled with the information for the nets you have selected. See Figure 3-8.

Figure 3-8 Coupling Analysis Table



The SI Victim Analysis or Coupling Analysis window consists of two sections. The nets of interest are shown in the top section. When you double-click a net in the top section, the coupled nets are shown in the bottom section.

For all nets, min/max delta, slack, and transition information is shown. You can right-click any net in either section to obtain a context menu with the following ECO operations:

- Set coupling separation (either global or pairwise)
- Net driver resizing (upsizing, downsizing, or specifying a replacement cell)
- Buffer insertion

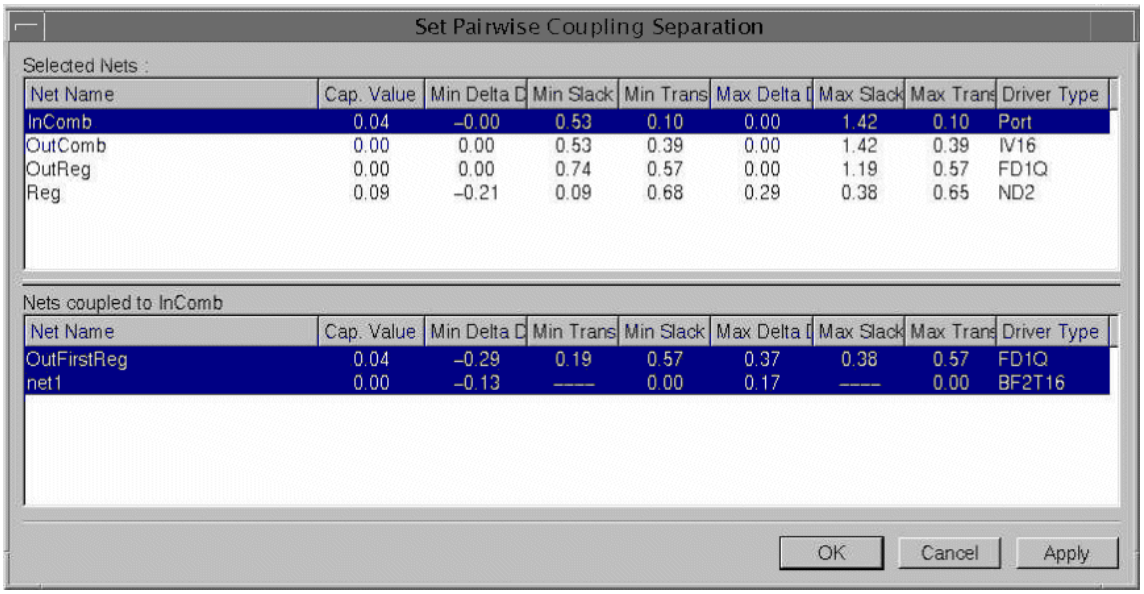
By considering the net information, you can make informed decisions about potential fixes. For example, if an aggressor net has sufficient setup slack, you could downsize its driver. If a victim net has a slow transition time, you can upsize its driver to make it more immune to crosstalk.

You can set coupling separation between nets from the ECO menu.

For coupling separation, select ECO > Set Coupling Separation.

For pairwise coupling separation, select ECO > Set Pairwise Coupling Separation. This opens the Set Pairwise Coupling Separation window, as shown Figure 3-9. For more information, see the man pages for `set_si_delay_analysis` and `set_coupling_separation`.

Figure 3-9 Set Pairwise Coupling Separation Table



Noise GUI Analysis

The GUI supports the display of static noise analysis results in histogram form. Upon completion of static noise analysis with the `update_noise` or `report_noise` command, you can display the results with the following commands:

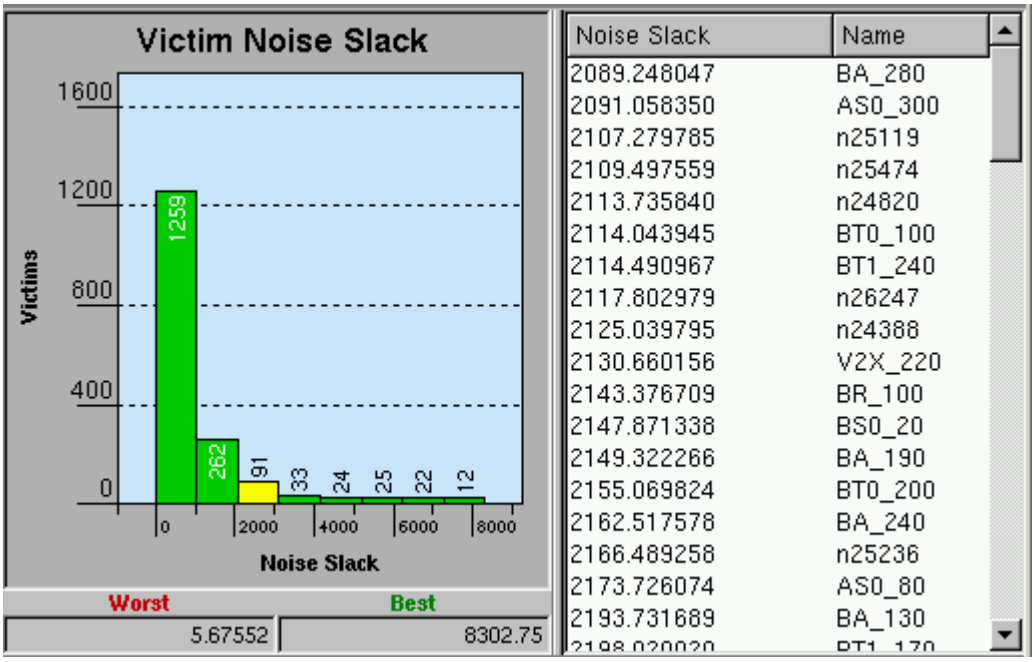
- Noise > Noise Slack Histogram
- Noise > Noise Bump Histogram
- Noise > Accumulated Noise Bump Histogram
- Noise > Noise Immunity Curve

Noise Slack Histogram

A noise slack histogram shows the distribution of noise slack values for nets in the design, considering the worst load pin in each net. Figure 3-10 shows an example. Noise slack is the amount by which a logic failure is avoided at a cell input with the worst-case composite noise

bump on that pin. The slack value is the failure threshold voltage minus the bump height, multiplied by the bump width. The units are in library voltage units multiplied by library time units.

Figure 3-10 Noise Slack Histogram



To generate a noise slack histogram, choose Noise > Noise Slack Histogram or click the equivalent toolbar button. In the dialog box, specify the bump type, the types of load pins to be included in the report, and the number of histogram bins. Then click OK to generate the plot.

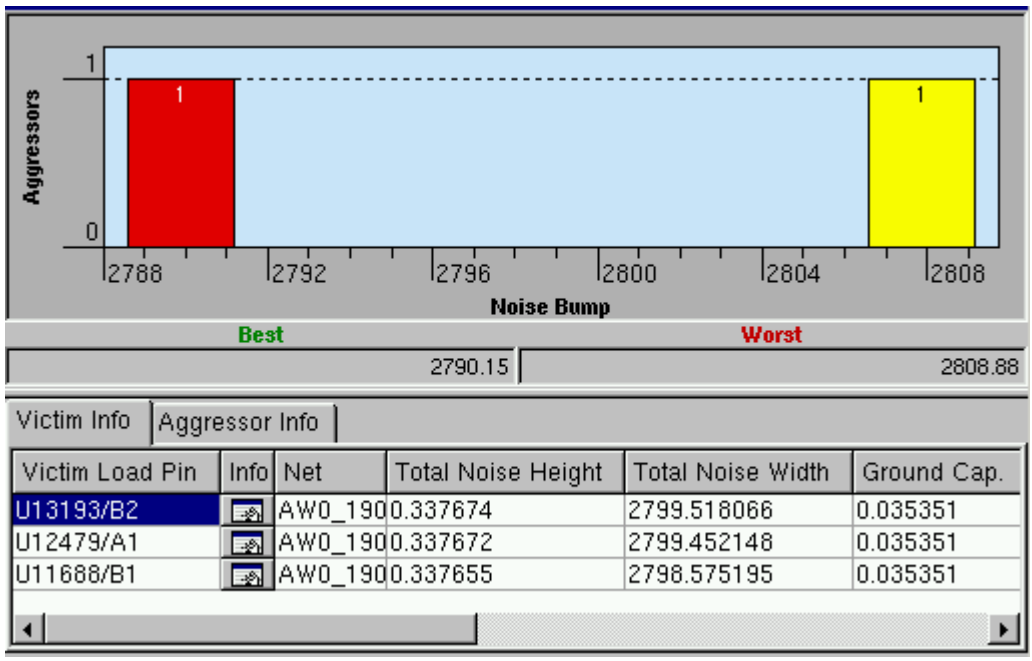
Click on any histogram bin to display a list of the nets in that bin. You can click on a net in the list to select that net and display its victim noise bump histogram, which then shows the load pins for that net.

Noise Bump Histogram

A victim noise bump histogram shows the distribution of noise bump heights on a load pin resulting from individual aggressor nets. Figure 3-11 shows an example. The aggressor nets are distributed in the histogram according to their individual contribution to the total bump height at the load pin, expressed as a fraction of the total rail-to-rail voltage. Two spreadsheets show relevant information about the victim net and aggressor nets.

To generate a bump voltage histogram, first select the net or load pin for the report. You can use Select > By Name, one of the Select > Nets commands, or Select > Pins; or you can select the net or pin in a schematic window, path profile window, histogram window, or path inspector window. Then choose Noise > Noise Bump Histogram or click the equivalent toolbar button. In the dialog box, specify the noise bump type and the number of bins. Click OK to generate the plot.

Figure 3-11 Victim Noise Bump Histogram With Victim Info Spreadsheet



If you select a net (not a specific load pin) before you generate the histogram, the histogram displays information on the load pin in the net that has the least noise slack, and the Victim Info spreadsheet lists all load pins in the net. You can select a particular load pin from the list to get the histogram for that load pin.

The histogram window has two tabbed pages at the bottom, labeled Victim Info and Aggressor Info. Clicking the Victim Info tab displays detailed information on the victim net. Clicking the Aggressor Info tab displays information on all aggressor nets in the currently selected bin.

The Victim Info spreadsheet shows detailed information about the victim net, including the load pin, noise height and width, ground capacitance, coupling capacitance, noise bump information, and noise slack information.

To open a dialog box showing detailed information on the victim net, click the Info button next to the victim name in the spreadsheet.

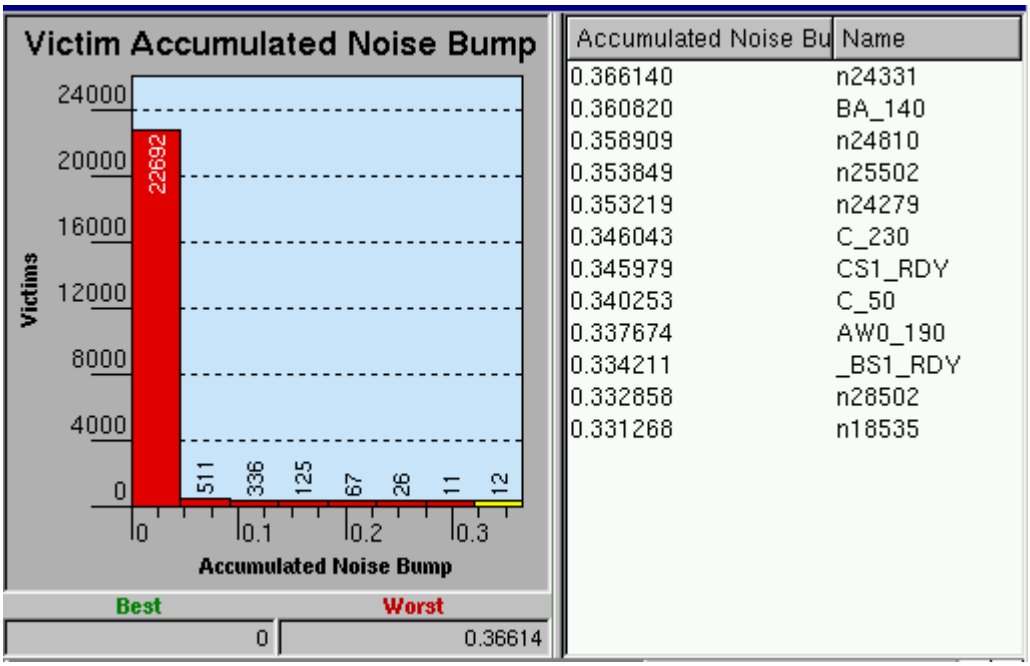
The Aggressor Info spreadsheet shows information about each aggressor net in the currently selected bin, including the net name, bump width and height, ground capacitance, coupling capacitance, and filtering status.

To open a dialog box showing more information on an aggressor net, click the Info button next to the aggressor net name in the spreadsheet.

Accumulated Noise Bump Histogram

An accumulated noise bump histogram shows the distribution of total crosstalk noise bump heights (not including propagated noise) for nets in the design, considering the worst load pin per net. Figure 3-12 shows an example. The load pins are distributed in the histogram according to their accumulated bump voltage values, expressed as a fraction of the total rail-to-rail voltage. The accumulated bump voltage is the height of the highest noise bump at the load pin resulting from all aggressor nets switching at the same time within their respective timing windows.

Figure 3-12 Accumulated Noise Bump Histogram



To generate an accumulated noise bump histogram, choose Noise > Accumulated Noise Bump Histogram or click the equivalent toolbar button. In the dialog box, specify the bump type, the types of load pins to be included in the report, and the number of histogram bins. Then click OK to generate the plot.

If necessary, PrimeTime SI runs `update_noise` in the background if to generate the noise data for the histogram.

Click on any histogram bin to display a list of the nets in that bin. You can click on a net in the list to select that net and display its victim noise bump histogram, which then shows the load pins for that net.

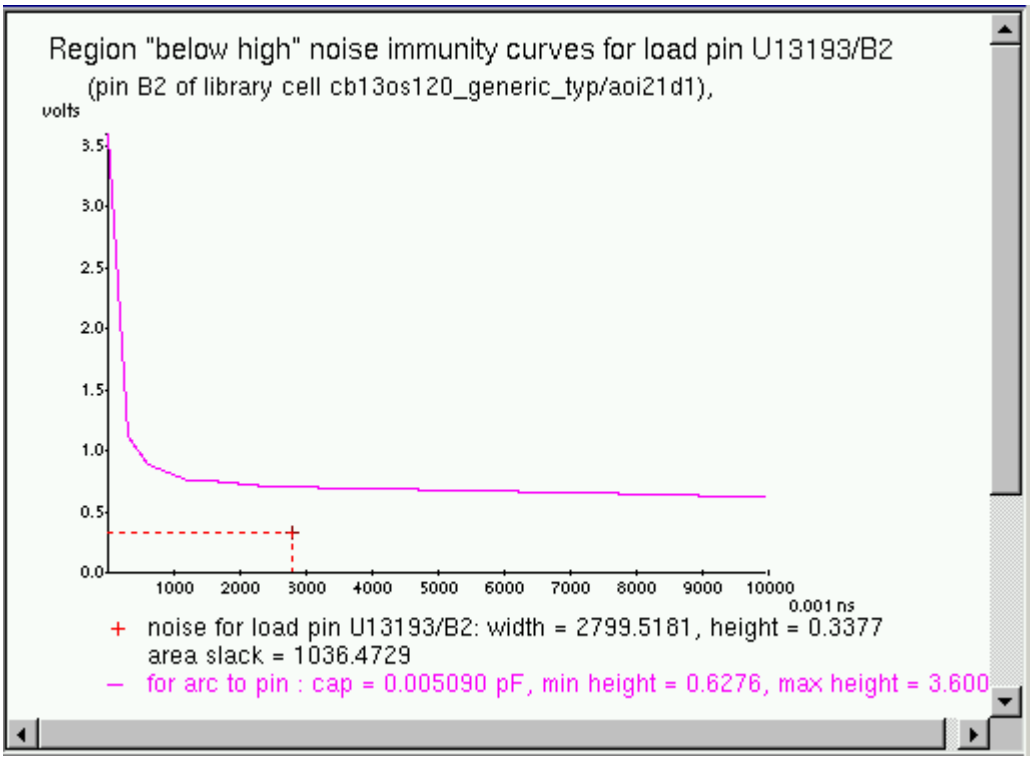
Noise Immunity Curves

The PrimeTime GUI can display a plot of the noise immunity curve at a cell input pin, showing a plot of the data point corresponding to the worst-case noise bump calculated at that pin.

To use this feature, first select the cell input pin, using any of several methods. For example, if you know the pin name, you can use Select > By Name; or from the path element table in the path inspector window, you can select the input pin showing the largest delta delay.

After you select the cell input pin, go to the main console window and choose Noise > Noise Immunity Curve. Fill in the dialog box as desired, including the noise bump type: below high, above low, above high, or below low. Then click OK. The GUI displays the noise immunity curve and the worst-case noise bump data, as shown in Figure 3-13.

Figure 3-13 Noise Detection Display



The curve shows the pass/fail input threshold voltage as a function of noise bump width. A small cross indicates the worst-case noise bump width and height at the pin. You can modify the width range and bump type of an existing plot by right-clicking and choosing Modify Graph.

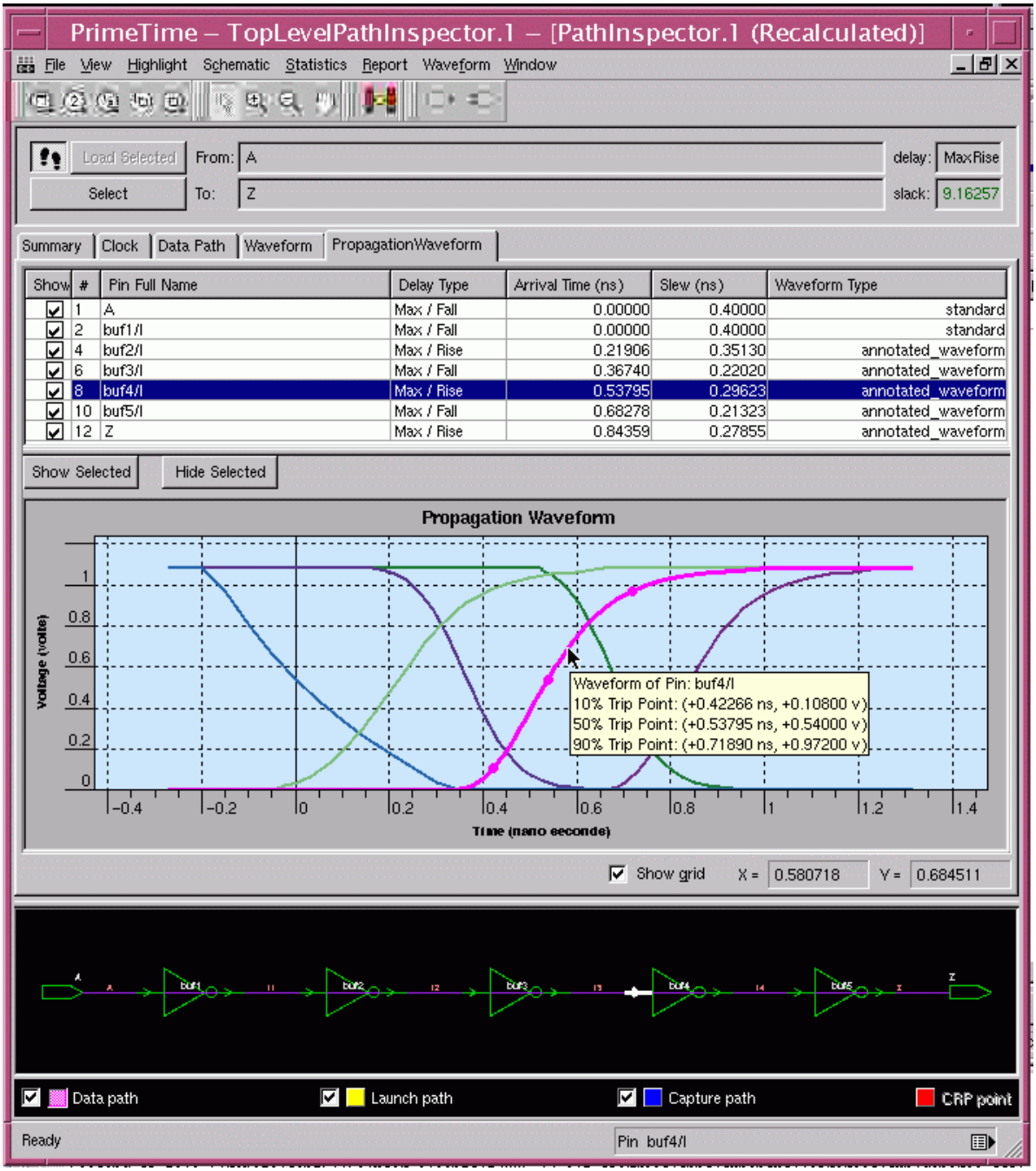
Waveform Display

For libraries that have both CCS timing and CCS noise models, PrimeTime SI can apply an advanced, gate-level delay calculation mode that uses the CCS timing and noise models, together with propagated piecewise-linear waveforms in place of simplified equivalent waveforms, for path-based analysis. This feature is invoked by setting the following variable:

```
pt_shell> set pba_enable_ccs_waveform_propagation true
```

This analysis mode is described in the section “Advanced Delay Calculation Using CCS Models” on page 2-29. When this mode is enabled, you can display the piecewise-linear waveforms in the GUI. For example, with a single recalculated timing path selected, use Timing > Inspect Recalculated Path; or in the Path Analyzer or Path Comparison Table, use the pop-up menu and select Inspect Recalculated Path. Figure 3-14 shows an example of a waveform display window.

Figure 3-14 Path-Based Waveform Display



4

Net Selection and Analysis Exit

To reduce the analysis runtime, PrimeTime SI selects only certain nets for the initial delay calculation iteration, and reselects only certain nets for subsequent iterations. To balance accuracy against speed, you can specify the selection and reselection criteria, as well as the criteria for terminating analysis iterations, as described in the following sections:

- Net Selection and Reselection
- Including or Excluding Specific Nets
- Delta Delay and Slack Reselection
- Reselecting Specific Nets
- Iteration Count and Exit

Net Selection and Reselection

Crosstalk analysis is an iterative process. For the initial delay calculation, PrimeTime SI uses a conservative model that does not consider transition timing windows, so it can quickly obtain approximate crosstalk delay values. In the second and subsequent delay calculation iterations, PrimeTime SI considers the timing windows and calculates crosstalk effects only when the aggressor and victim windows overlap. This gives a more accurate, less pessimistic analysis of worst-case crosstalk effects.

For the initial delay calculation iteration, PrimeTime SI selects all cross-coupled nets for analysis. You can also exclude specific nets as aggressors or as victims, or specific aggressor-victim net pairs, by using the `set_si_delay_analysis -exclude` command (see "Including or Excluding Specific Nets" on page 4-3).

For subsequent delay calculation iterations, several variables control the net reselection method, as indicated in Table 4-1.

Table 4-1 Net Reselection Variables

Variable	Default
`si_xtalk_reselect_clock_network`	true
`si_xtalk_reselect_delta_delay`	5
`si_xtalk_reselect_delta_delay_ratio`	0.95
`si_xtalk_reselect_max_mode_slack`	0
`si_xtalk_reselect_min_mode_slack`	0
`si_xtalk_reselect_time_borrowing_path`	true

When a net is reselected, PrimeTime SI recalculates the delays of that net, and also the delays of the nets in the fanout cone of the reselected net. You can also have PrimeTime SI reselect specific nets by using the `set_si_delay_analysis -reselect` command.

In each iteration after the first iteration, PrimeTime SI reports the total number of nets reselected and the number of nets reselected because of each possible reason. For example:

```
Number of nets reselected for next iteration: 1868. (XTALK-004)
Number of reselected nets by criteria:          all  exclusively
  si_xtalk_reselect_min_mode_slack              245       76
  si_xtalk_reselect_max_mode_slack             1783      418
```

```
  si_xtalk_reselect_delta_delay                128        0
  si_xtalk_reselect_delta_delay_ratio            4        2
  set_si_delay_analysis -reselect                0        0
  si_xtalk_reselect_time_borrowing_path       1370        7
  incremental_changes                            0        0
  si_xtalk_reselect_clock_network                0        0
Information: Starting crosstalk aware timing iteration 2.
```

Some nets are reselected for more than one reason. The “all” column shows how many nets were reselected for each reason. The “exclusively” column shows how many nets were reselected for one specific reason and no other reason. This part of the report gives an approximate indication of the amount of CPU time being spent on each type of net reselection.

Including or Excluding Specific Nets

PrimeTime SI has four commands you can use to specify particular nets that are to be included or not included in the analysis:

- `set_si_delay_analysis` includes or excludes specified nets for crosstalk delay analysis
- `set_si_noise_analysis` includes or excludes specified nets for crosstalk noise analysis
- `set_analysis_agressor_exclusion_mode` excludes aggressor to aggressor nets while switching in the same direction
- `set_coupling_separation` excludes nets or net pairs from crosstalk delay and crosstalk noise analysis

The `set_si_delay_analysis` command lets you include or exclude nets for crosstalk delay analysis in the following ways:

- Reselect specific nets in all subsequent iterations, even if they do not meet the usual reselection criteria
- Set specific nets to use infinite arrival windows
- Exclude specific nets as aggressors
- Exclude specific nets as victims
- Exclude specific aggressor-victim relationships between net pairs

If there is a conflict between the settings of the `set_coupling_separation` command and the `set_si_delay_analysis` command or `set_si_noise_analysis` command, the `set_coupling_separation` command has precedence.

Excluding a Clock Net

Suppose that you know that the clock signal delays in your design are not significantly affected by crosstalk. To exclude the clock net CLK1 from consideration as a victim net:

```
pt_shell> set_si_delay_analysis -exclude -victims \
            [get_nets CLK1]
```

PrimeTime SI will not consider the net CLK1 to be a potential victim net for crosstalk delay analysis, thereby reducing the analysis runtime. CLK1 can still be considered as an aggressor net, however.

Excluding Aggressor-Victim Pairs

To exclude specific aggressor-victim net pair relationships, use the `-exclude` option with both the `-victims` and `-aggressors` options. For example, suppose that you know that the scan clock signals (SCN_CLK_*) and clock network signals (CLK_NET_*) in your design do not affect each other for timing.

To eliminate these crosstalk effects from consideration:

```
pt_shell> set_si_delay_analysis -exclude -aggressors \
    [get_nets SCN_CLK_*] -victims [get_nets CLK_NET*]

pt_shell> set_si_delay_analysis -exclude -victims \
     [get_nets SCN_CLK_*] -aggressors [get_nets CLK_NET*]
```

The first of these two commands removes from consideration any SCN_CLK_* signal as an aggressor to any CLK_NET* signal as a victim during crosstalk delay analysis. The second command removes from consideration the aggressor-victim relationships of these signals in the opposite direction. However, note that any of these signals can still be considered aggressors or victims relative to signals not mentioned in the commands. Also, CLK_NET1 can still be considered an aggressor to CLK_NET2 as a victim, for example.

Excluding Aggressor-to-Aggressor Nets

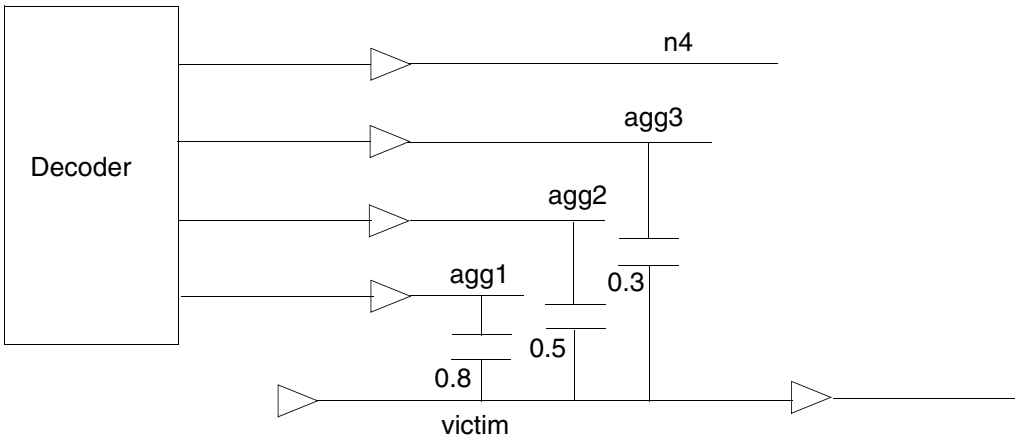
By default, when multiple aggressor arrival windows overlap with a victim transition window, PrimeTime SI considers the worst case of multiple aggressor transitions occurring in the same direction at the same time. In some cases, however, the logical relationship between the aggressor signals prevent simultaneous switching of the aggressors in the same direction at the same time.

You can reduce pessimism in crosstalk analysis by specifying groups of aggressor nets as exclusive during worst-case alignment. Exclusive aggressor nets are nets among which only a specified number of aggressors can switch simultaneously in the same direction. The nets

can be exclusive for rise, fall, or both. Exclusive for rise means only specified maximum number of aggressors within the exclusive set can rise to impact a victim and all the other aggressors within the exclusive set are considered quiet (not switching).

PrimeTime SI considers these exclusive aggressor groups during crosstalk delay and noise analysis. Figure 4-1 is an example showing one-hot signal nets where only one of the nets is active at a given instance; therefore, only one net can have a value of 1; the remaining ones have a value of 0. Here, the aggressors agg1 agg2 agg3 can be defined as an exclusive group for both rising and falling directions.

Figure 4-1 One-Hot Decoder Multiple Aggressor Example



In the initial crosstalk analysis iteration, the analysis considers the combined effect three aggressor transitions have on the victim net. In the second and higher iterations, however, the analysis considers the arrival windows of the aggressor and victim transitions. To analyze the slowdown of a falling transition on the victim net, suppose that the analysis finds the arrival windows of rising transitions on the aggressor nets as shown in Figure 4-2, then Table 4-2 shows how the aggressors are considered during crosstalk analysis.

Figure 4-2 Multiple Aggressor Arrival Windows at Decoder Outputs

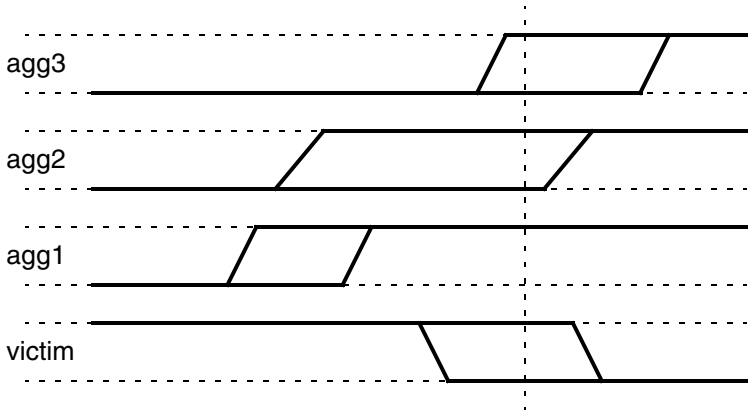


Table 4-2 Crosstalk Analysis With and Without Exclusive Group

	agg1	agg2	agg3
No exclusive group specified	Quiet	Active	Active
With exclusive group {agg1 agg2 agg3}	Quiet	Active	Quiet

Here agg1 is considered quiet in both cases because it does not overlap with the victim window. When no exclusive group is specified, both agg2 and agg3 would be considered to be active because the arrival window of the victim overlaps with the arrival windows of both agg2 and agg3. When the exclusive group is specified, only the aggressor inducing the largest bump, agg2, is considered to be active.

You can set aggressors to be exclusive while switching in the same direction for crosstalk delay and noise analysis by setting the `set_si_aggressor_exclusion` command. For example, the following command sets a group of aggressors to be exclusive in the rise direction.

```
pt_shell> set_si_aggressor_exclusion \
          [get_nets {A1 A2 A3 A4}] -rise
```

The following example instructs that only two of the aggressors in the given exclusive group can switch simultaneously.

```
pt_shell> set_si_aggressor_exclusion \
          [get_nets {DECODER*}] \
          -number_of_active_aggressors 2
```

The -rise option sets aggressor nets to be exclusive in the rise direction; -fall fall sets nets as exclusive in the fall direction. If you don't specify either of these options, the nets are assumed to be exclusive in both the rise and the fall direction. To specify the maximum number of aggressors that can be active, use the -number_of_active_aggressors option. If the number of active aggressors is not specified, only one of the exclusive aggressors is assumed to be active.

The application of commands is order independent. The most restrictive number of active aggressors is chosen. Those aggressors you have already excluded using the -exclude option to either the set_si_delay_analysis or set_si_noise_analysis commands will remain in their excluded state. Note that the next update_timing triggered after the application of these commands would be a full update.

To remove exclusive groups set in your design, use the remove_si_aggressor_exclusion command. When using this command, only those exclusive groups set using set_si_aggressor_exclusion are removed.

For example, the following will remove an exclusive group set in the rise direction.

```
pt_shell> remove_si_aggressor_exclusion \
          [get_nets {A1 A2 A3}] -rise
```

Use the -all option to remove all exclusive groups in the design.

Excluding Rising/Falling Edges or Setup/Hold

To exclude only rising or only falling edges at the victim net, use the -rise or -fall option of the set_si_delay_analysis command. To exclude only maximum (setup) or only minimum (hold) path analysis, use the -max or -min option of the set_si_delay_analysis command.

Excluding Nets from Crosstalk Noise Analysis

The set_si_noise_analysis command also causes nets to be included or not included for crosstalk noise analysis. For example, to exclude the clock net CLK1 from consideration as a victim net for crosstalk noise analysis:

```
pt_shell> set_si_noise_analysis -exclude -victims \
            [get_nets CLK1]
```

Excluding Analysis of Noise Bumps

To exclude the analysis of above-high and below-low noise bumps for the CLK1 net, use the following commands:

```
pt_shell> set_si_noise_analysis -exclude [get_nets CLK1] \
            -above -high

pt_shell> set_si_noise_analysis -exclude [get_nets CLK1] \
            -below -low
```

Coupling Separation

The `set_coupling_separation` command creates a separation constraint on nets, like using both the `set_si_delay_analysis` and `set_si_noise_analysis` commands with the `-exclude` option. For example, to prevent crosstalk analysis (for both delay and noise) between the net CLK1 and all other nets:

```
pt_shell> set_coupling_separation [get_nets CLK1]
```

To ignore the cross-coupling capacitance between two particular nets, use the `-pairwise` option. For example:

```
pt_shell> set_coupling_separation -pairwise \
            [get_nets CLK1] [get_nets NET1]
```

Removing Exclusions

To reverse the effects of `set_si_delay_analysis`, `set_si_noise_analysis`, or `set_coupling_separation`, use the command `remove_si_delay_analysis`, `remove_si_noise_analysis`, or `remove_coupling_separation`.

Only those separations or exclusions that you set directly can be removed by these commands. Any coupling separations or exclusions which were implicitly set on the nets due to the coupling between nets will not be removed.

For example, suppose your design had a victim net, V, which has three aggressors, A1, A2, and A3. To see a list of the aggressors for net V, you would use the following command:

```
pt_shell> get_attribute -class net [get_net V] aggressors
A1 A2 A3
```

If you then exclude the net V from consideration as victim for crosstalk with the following command, the excluded list for net V would be { A1 A2 A3 }:

```
pt_shell> set_si_delay_analysis -exclude -victims V
```

Even though the exclusion on net V implies that A1 is excluded as an aggressor for net V, exclusion between nets V and A1 cannot be removed by using the `remove_si_delay_analysis -aggressors` command, because no exclusion was directly set on the net A1. Therefore, after the following command, the excluded list for net V would still be { A1 A2 A3 }:

```
pt_shell> remove_si_delay_analysis -aggressors A1
Warning: Cannot remove global separation or exclusion that
was not set on net(s) A1. (XTALK-107)
```

Similarly, even though the exclusion on net V implies that A2 is excluded as an aggressor for V, this exclusion cannot be removed by using the victim-aggressor option because no exclusion was directly set for the victim-aggressor pair V and A2. Therefore, after the following command, the excluded list for net V would still be { A1 A2 A3 }:

```
pt_shell> remove_si_delay_analysis -victims V \
          -aggressors A2
Warning: Cannot remove global separation or exclusion that
was not set on net(s) V A2. (XTALK-107)
```

Only exclusions that were applied directly can be removed, so if you issued the following command, the effect of the `set_si_delay_analysis -exclude -victim V` command would be reversed, and the excluded list would be empty:

```
pt_shell> remove_si_delay_analysis -victims V
```

Delta Delay and Slack Reselection

PrimeTime SI reselects a victim net that satisfies any of the following three conditions:

- The absolute change in stage delay (either positive or negative) caused by crosstalk analysis in the previous iteration exceeds the amount specified by the `si_xtalk_reselect_delta_delay` variable.
- The relative change in stage delay (either positive or negative) caused by crosstalk analysis in the previous iteration is a ratio that exceeds the amount specified by the `si_xtalk_reselect_delta_delay_ratio` variable. The ratio is calculated by dividing the absolute change in delay by the total stage delay.
- The net slack calculated in the minimum (hold) analysis of the previous iteration is less than the threshold set by the `si_xtalk_reselect_min_mode_slack` variable, or the net slack calculated in the maximum (setup) analysis of the previous iteration is less than the threshold set by the `si_xtalk_reselect_max_mode_slack` variable.

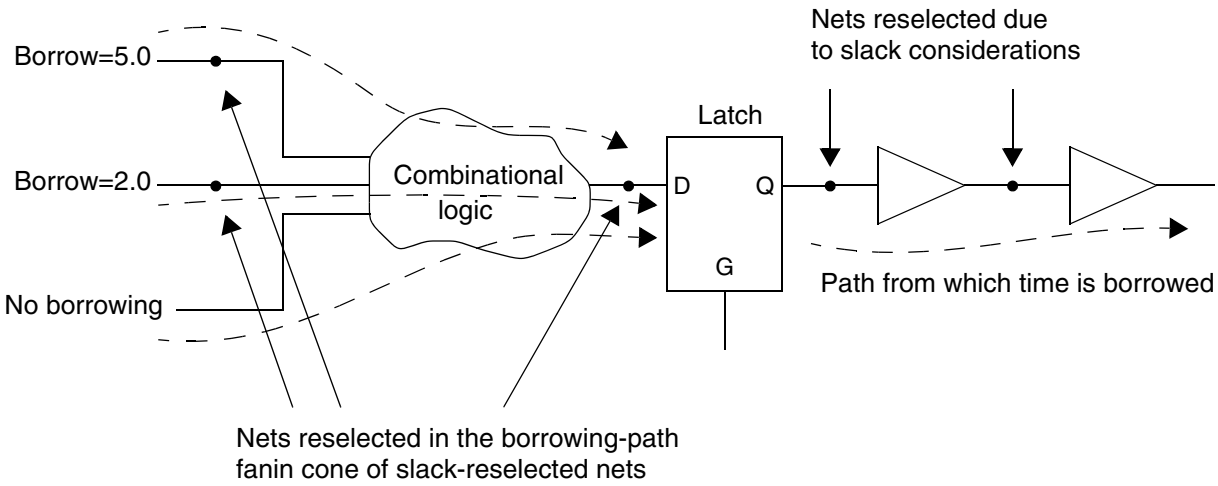
Stage delay is the delay of a stage (one cell and its fanout net). Crosstalk analysis in the previous iteration may have caused a change in the calculated worst-case delay: a decrease in delay for a minimum analysis or an increase in delay for a maximum analysis. If the amount of change is large enough, the net in that stage is reselected for further analysis.

Typically, the change in calculated delay becomes smaller and smaller for successive analysis iterations, as the analysis becomes more and more accurate. Reselecting nets based on the amount of delay change is a way to ensure accurate results.

Net slack is the worst slack value (smallest or most negative slack value) among all paths through the net. Reselecting nets based on the amount of slack is a way to ensure accurate crosstalk analysis of nets with critical timing.

For nets reselected due to slack considerations, PrimeTime SI also reselects nets in the fanin cone of each reselected net if a borrowing latch is involved, as shown in Figure 4-3. This is because a timing change in the borrowing path can affect the slack in the path from which time is being borrowed.

Figure 4-3 Reselection of Nets in Borrowing-Path Fanin Cone



PrimeTime SI traces chains of borrowing paths backward through the design, choosing all borrowing paths that borrow from each path startpoint, until it reaches a nonborrowing latch, flip-flop, or primary input. All nets in the borrowing paths are reselected.

To prevent reselection of nets in the borrowing-path fanin cone of nets reselected for slack considerations, set the variable `si_xtalk_reselect_time_borrowing_path` to false. In that case, the analysis might be faster because PrimeTime SI reselects fewer nets for analysis, but the results might be less accurate for the borrowing paths.

By default, a net satisfying any one (or more) of the following conditions is reselected for analysis:

- absolute delta delay change (..._delta_delay variable)
- relative delta delay change (..._delta_delay_ratio variable)
- hold slack (..._min_mode_slack variable)
- setup slack (..._max_mode_slack variable)
- time-borrowing slack (..._time_borrowing_paths variable)

In the “delta and slack” mode, nets that satisfy only one or two conditions are not counted in the XTALK-004 reselection report. For example:

```
Information:
Number of nets reselected for next iteration:  350. (XTALK-004)
Number of reselected nets by criteria:          all  exclusively
  si_xtalk_reselect_min_mode_slack             350        0
  si_xtalk_reselect_max_mode_slack             350        0
  si_xtalk_reselect_delta_delay                350        0
  si_xtalk_reselect_delta_delay_ratio            4        2
  set_si_delay_analysis -reselect                0        0
  si_xtalk_reselect_time_borrowing_path          0        0
  incremental_changes                            0        0
  si_xtalk_reselect_clock_network                0        0
```

Reselecting Specific Nets

The set_si_delay_analysis and set_si_noise_analysis commands let you reselect specific nets in all subsequent iterations for crosstalk analysis, even if those nets do not meet the usual reselection criteria.

For example, suppose that you want PrimeTime SI to analyze multiple reset signals (RESET1, RESET2, and so on) for crosstalk effects in all analysis iterations, even though they might not meet the usual reselection criteria. To ensure that these nets are reselected:

```
pt_shell> set_si_delay_analysis -reselect \
            [get_nets RESET*]

pt_shell> set_si_noise_analysis [get_nets RESET*]
```

Note that forced reselection of nets can cause a significant increase in runtime.

To reverse the effects of set_si_delay_analysis or set_si_noise_analysis, use remove_si_delay_analysis or remove_si_noise_analysis.

Iteration Count and Exit

PrimeTime SI calculates crosstalk effects using an iterative loop. The first iteration uses a fast, conservative analysis mode that does not consider the timing windows between aggressor and victim nets. In successive iterations, the results become increasingly accurate (less pessimistic) because PrimeTime SI can further narrow down the transition windows, allowing more and more potential crosstalk effects to be eliminated because they are spaced apart in time.

By default, PrimeTime SI exits from the loop upon completion of the second iteration, which typically provides good results in a reasonable amount of time. You can optionally specify a larger number to possibly get more accurate results at the cost of more runtime.

The loop is always terminated if no nets are reselected for the next iteration. This condition can occur if no cross-coupled nets meet the delta delay and slack reselection criteria. (For details, see "Delta Delay and Slack Reselection" on page 4-9.)

You specify the exit criteria by setting the variables listed in Table 4-3.

Table 4-3 Exit Criteria Variables

Variable	Default
`si_xtalk_exit_on_max_iteration_count`	2
`si_xtalk_exit_on_max_iteration_count_incr`	2

The `si_xtalk_exit_on_max_iteration_count` variable sets the maximum iteration count, irrespective of the analysis results. The default setting is 2, which usually causes a crosstalk analysis to run for two iterations. (An analysis of just one iteration is possible, depending on the design and the exit criteria.)

To use a specific number of analysis iterations, increase this variable setting without changing any of the other variables. For example:

```
pt_shell> set si_xtalk_exit_on_max_iteration_count 3
```

Sometimes PrimeTime SI can do a timing update incrementally, which takes less time than a full analysis. An incremental analysis can be done only after minor changes such as `size_cell`, `insert_buffer`, or `set_coupling_separation`. The number of iterations for incremental analysis is controlled by a separate variable, `si_xtalk_exit_on_max_iteration_count_incr`.

You can terminate an analysis in progress by pressing Control-c once. PrimeTime SI completes the current analysis iteration before exiting from the loop. Note that pressing Control-c multiple times will cause an exit from PrimeTime SI upon completion of the loop.

5

Multiple Supply Voltage Analysis

PrimeTime SI can analyze designs with different power supply voltages for different cells. This capability is described in the following sections:

- Multivoltage Analysis
- IR Drop Annotation

Multivoltage Analysis

PrimeTime SI has the ability to analyze designs with different power supply voltages for different cells. It accurately determines the effects of delay, slew, and noise in the presence of differences in supply voltage, taking advantage of cell delay models specified in the technology library.

The multivoltage infrastructure lets you do the following:

- Calculate uncoupled signal net delay, constraints, and timing violations while considering exact driver and load power rail voltages.
- Calculate coupled delay, noise bumps, timing violations, and noise violations while considering exact aggressor rail voltages.
- Perform signal level mismatch checking between drivers and loads.
- Perform the foregoing types of analysis with instance-specific annotated voltage (IR) drop on power rails, and with block-specific or instance-specific temperature.

Most PrimeTime analysis features are capable of producing results that are fairly close to physical (SPICE simulation) analysis. For accurate analysis in PrimeTime SI, the following types of information must be available:

- Power and ground (PG) connectivity, as described in the “Low-Power Flow Support” chapter in the *PrimeTime Advanced Timing Analysis User Guide*.
- Voltage values on PG nets specified with the `set_voltage` command, as described in the “Low-Power Flow Support” chapter in the *PrimeTime Advanced Timing Analysis User Guide*.
- IR drop values specified for PG nets with the `set_voltage` command using the `-cell` and `-pg_pin_name` options, as described in the section “IR Drop Annotation” on page 5-3. Specifying IR drop is optional. A PrimeTime SI license is required.
- A set of CCS libraries with delay and constraint data (used with the `define_scaling_lib_group` command) or library delay calculation data for each voltage (used with the `link_path_per_instance` variable). CCS-based library models are recommended for best accuracy. The `define_scaling_lib_group` command invokes delay and constraint scaling between a set of libraries that have been characterized at different voltages and temperatures. The `link_path_per_instance` variable links different cell instances to different library models. For more information, see the *PrimeTime Advanced Timing Analysis User Guide*.
- Settings that define how to use the library data to build delay calculation models. For more information, see the descriptions of `link_path_per_instance` and `define_scaling_lib_group` in the *PrimeTime Advanced Timing Analysis User Guide*.

In previous releases of PrimeTime SI, multivoltage analysis was supported by connectivity based on power domain syntax (release Z-2007.06) or voltages annotated directly on cell instances (before release Z-2007.06). These methods are still supported for backward compatibility. A PrimeTime SI license is no longer required for multivoltage analysis, except for instance-specific IR drop annotation with the `set_voltage` command.

For more information on multivoltage analysis methods used in previous releases, see the sections “Power Domain Mode of Release Z-2007.06” and “Multivoltage Method Prior to Release Z-2007.06” in the *PrimeTime Advanced Timing Analysis User Guide*.

IR Drop Annotation

You can annotate supply voltages on the PG pins of cells in the design. and thereby model the effects of IR drop on timing and noise, by using the `-cell` and `-pg_pin_name` options of the `set_voltage` command. This is the command syntax:

```
set_voltage voltage -cell cell_list -pg_pin_name pg_pin_list
```

For example,

```
pt_shell> set_voltage 1.10 -cell [get_cells U5*] -pg_pin_name VDD1
```

This command sets the supply voltage to 1.10 volts on the VDD1 pins of all the U5* cells in the design.

The `-cell` and `-pg_pin_name` options, if used in the `set_voltage` command, must be used together. These two options require a PrimeTime SI license. No PrimeTime SI license is required for the other `set_voltage` options, such as the `-object_list` option used for setting voltages on power supply nets.

Signal Level Checking With IR Drop

PrimeTime checks for mismatching voltage levels between cells that use different supply voltages when you use the following `check_timing` command:

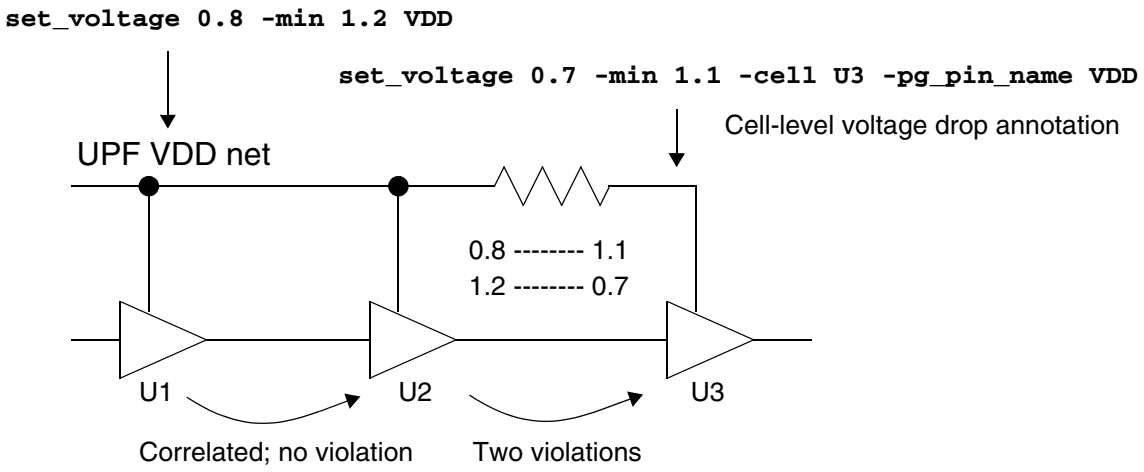
```
pt_shell> check_timing -include signal_level
```

The signal level check correctly considers the case where minimum and maximum voltages are set on voltage rails that supply both of the cells being compared. Suppose, however, that IR drop is annotated on a cell using commands similar to the following:

```
pt_shell> set_voltage 0.8 -min 1.2 VDD   # corner voltages
pt_shell> set_voltage 0.7 -min 1.1 -cell U3 -pg_pin_name VDD  # IR drop
```

This corresponds to the circuit shown in Figure 5-1.

Figure 5-1 Signal Level Checking With Cell-Level Voltage Annotation



In this case, PrimeTime SI assumes that the supply voltage annotated at the cell level is not correlated with the main supply voltage. It considers the full worst-case differences between the driver and receiver, which is to compare 0.8 to 1.1 volts and 1.2 to 0.7 volts, thus triggering two mismatch violations.

6

Static Noise Analysis

PrimeTime SI can analyze designs for logic failure due to crosstalk noise on steady-state nets. Static noise analysis is described the following sections:

- Static Noise Analysis Overview
- Noise Analysis Commands
- CCS Noise Modeling
- NLDM Noise Modeling

Static Noise Analysis Overview

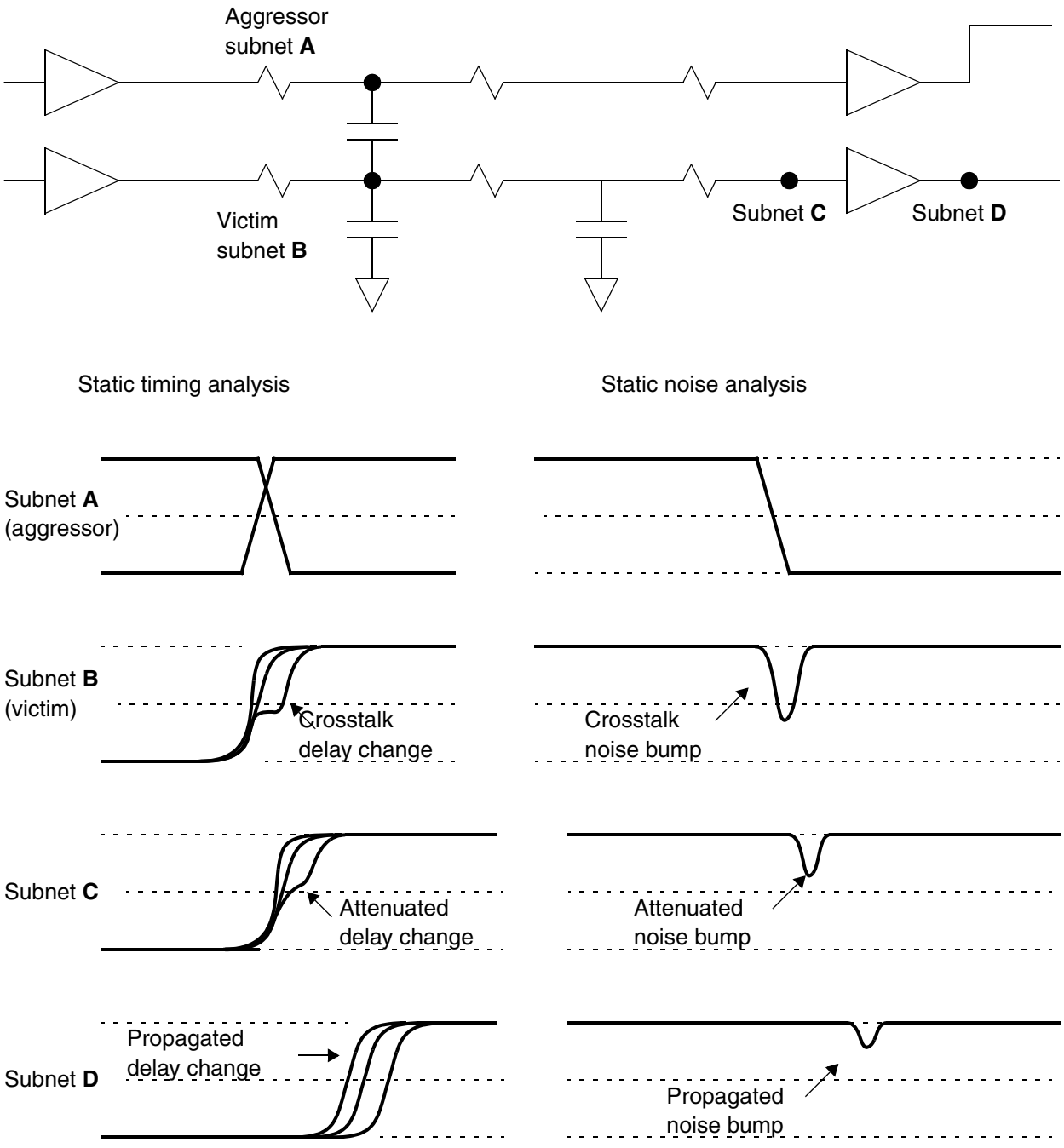
Static noise analysis is a technique for finding the worst noise effects in a design so that they can be reduced or eliminated, thereby maximizing the reliability of the finished product.

Static noise analysis, like static timing analysis, does not rely on test vectors or circuit simulation to determine circuit behavior. Instead, it considers the cross-coupling capacitance between aggressor nets and the victim net, the arrival windows of aggressor transitions, the drive characteristics of the aggressor nets, the steady-state load characteristics of the victim net driver, and the propagation of noise through cells. Using this information, PrimeTime SI determines the worst-case noise effects and reports conditions that could lead to logic failure.

Figure 6-1 compares static timing analysis and static noise analysis done by PrimeTime SI. For either type of analysis, PrimeTime SI considers the cross-coupling between aggressor nets and victim nets. For static timing analysis, PrimeTime SI determines the worst-case change in delay on the victim net caused by crosstalk. For static noise analysis, it determines the worst-case noise bump or glitch on a steady-state victim net. (Steady-state means that the net is constant at logic one or logic zero.) A noise bump on a steady-state net can be propagated down the timing path and could be captured as incorrect data at the path endpoint.

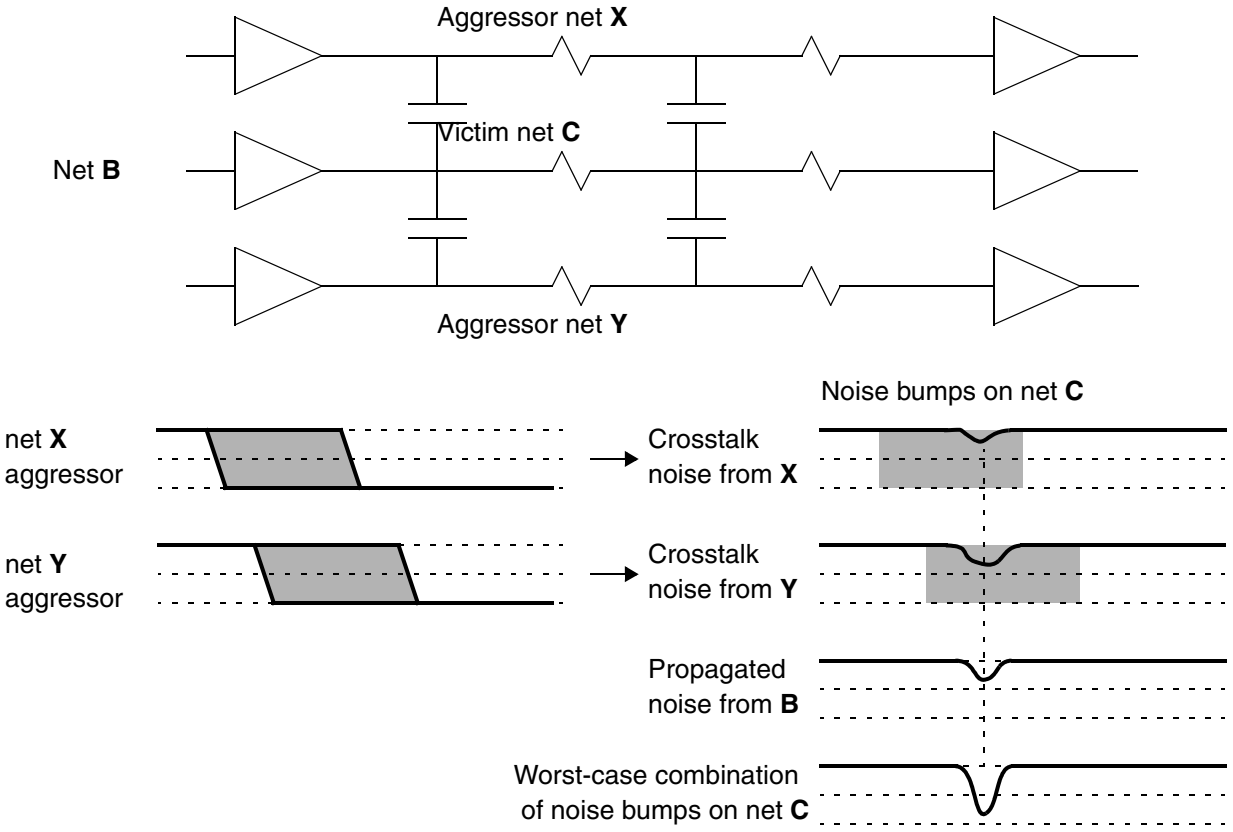
Figure 6-2 on page 6-4 shows how PrimeTime SI combines the effects from multiple aggressors and propagated noise, taking the aggressor timing windows into account, to determine the worst-case noise bump on the victim net. It propagates the worst-case noise bump through the subnets of the victim net, resulting in a composite noise bump on each load pin of the victim net.

Figure 6-1 Crosstalk Effects on Timing and Steady-State Voltage



The main commands for noise analysis are `check_noise`, `update_noise`, and `report_noise`, which operate in a manner similar to `check_timing`, `update_timing`, and `report_timing` for static timing analysis. The `check_noise` command checks the design for the presence and validity of noise models at driver and load pins. The `update_noise` command performs a noise analysis and updates the design with noise bump information. The `report_noise` command generates a report on worst-case noise effects, including width, height, and noise slack.

Figure 6-2 Combined Effects of Crosstalk and Propagated Noise



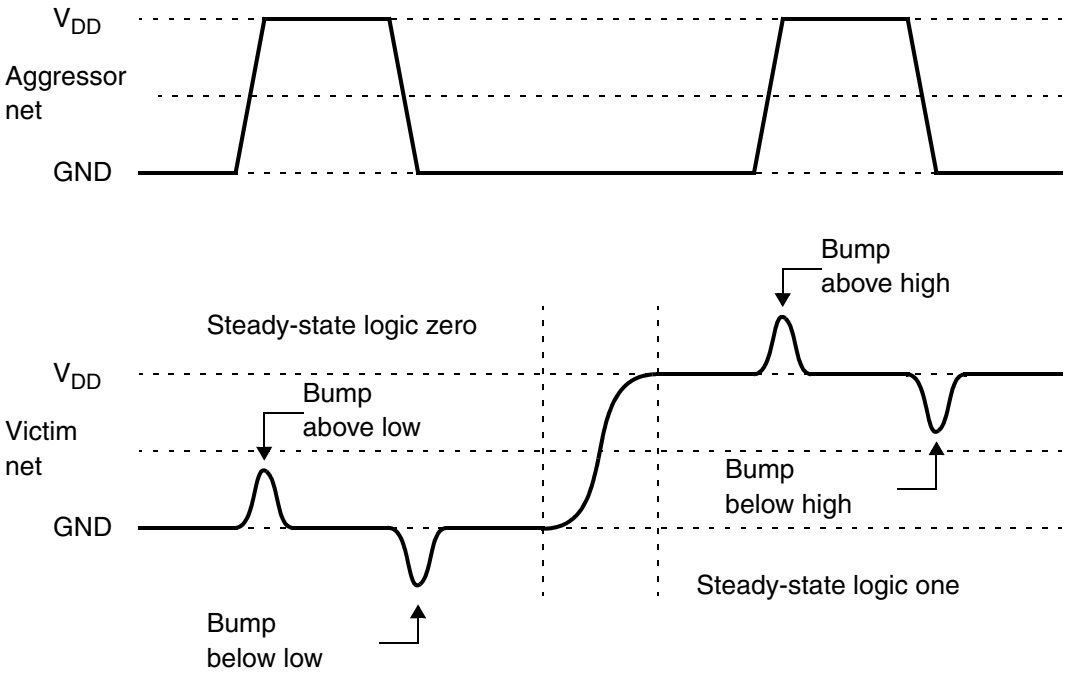
Noise Bump Characteristics

The term “noise” in electronic design generally means any undesirable deviation in voltage of a net that ought to have a constant voltage, such as a power supply or ground line. In CMOS circuits, this includes data signals being held constant at logic one or logic zero.

There are many different causes of noise such as charge storage effects at P-N junctions, power supply noise, and substrate noise. However, the dominant noise effect in deep-submicron CMOS circuits is crosstalk noise between physically adjacent logic nets. For this reason, PrimeTime SI concentrates on the analysis of crosstalk noise between these nets and the propagation of the resulting crosstalk bumps from cell input to cell output.

Noise effects, when plotted as voltage versus time, can have many different forms: bumps, ripples, random noise, and so on. PrimeTime SI concentrates on the four types of noise bumps typically caused by aggressor net transitions: above low, below low, above high, and below high, as shown in Figure 6-3.

Figure 6-3 Noise Bump Types

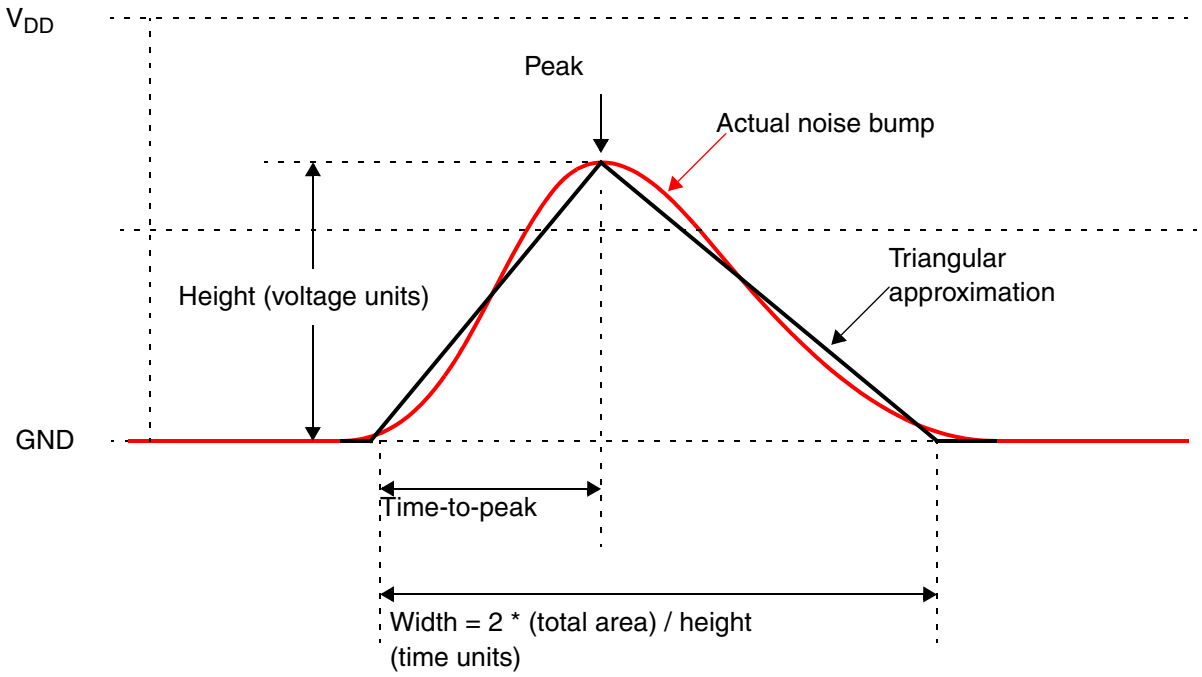


Bumps between the two rail voltages (above low and below high) can cause logic failure due if they exceed the logic thresholds of the technology. Bumps outside of the range between the two rail voltages (below low and above high) are also important because they can forward-bias pass gates at the inputs of flip-flops and latches, allowing incorrect values to be latched.

Three numbers specify the characteristics of a noise bump: the height in voltage units, the total width in time units, and the time-peak-ratio, which is the time-to-peak divided by the total bump width. A time-peak-ratio of 0.5 indicates a symmetrical bump with equal rising and falling times.

To determine the width and time-to-peak value of a noise bump, PrimeTime SI uses a triangular approximation based on the location of the peak and the area under the curve. See the example in Figure 6-4.

Figure 6-4 Noise Bump Characteristics



The width of a noise bump is defined to be two times the area under the whole curve divided by the height. The time-to-peak value is defined to be two times the area under the curve to the left of the peak, divided by the height.

Noise Bump Calculation

PrimeTime SI calculates the cumulative effect of noise from different sources: capacitive cross-coupling, propagation through logic cells and through subnets, and user-defined noise bumps.

Crosstalk Noise

To calculate noise bumps caused by signal crosstalk, PrimeTime SI considers the cross-coupling capacitance between the aggressor nets and the victim net, the arrival windows of aggressor transitions, the drive characteristics of the aggressor nets, and the steady-state

resistance characteristics of the victim net. This calculation is similar to what is done for crosstalk delay analysis, except that it uses the steady-state load characteristics (not the driver characteristics) of the driver on the victim net.

Aggressor nets that contribute to the worst-case noise bump on the victim net are called active aggressors. The remaining aggressor nets do not contribute to the worst-case bump because their timing windows are outside of the worst-case region, their bumps are too small to be significant, or they have been eliminated by the user or by logical correlation.

Propagated Noise

Propagated noise on a victim net is caused by noise at an input of the cell that is driving the victim net. PrimeTime SI can calculate propagated noise at a cell output, given the noise bump at the cell input and the load on the cell output.

The propagation of a noise bump from input to output can either amplify or attenuate the noise bump. In other words, the output noise bump can be either larger or smaller than the input noise bump, depending on the size of the input noise bump and the operating characteristics of the cell.

User-Defined Noise

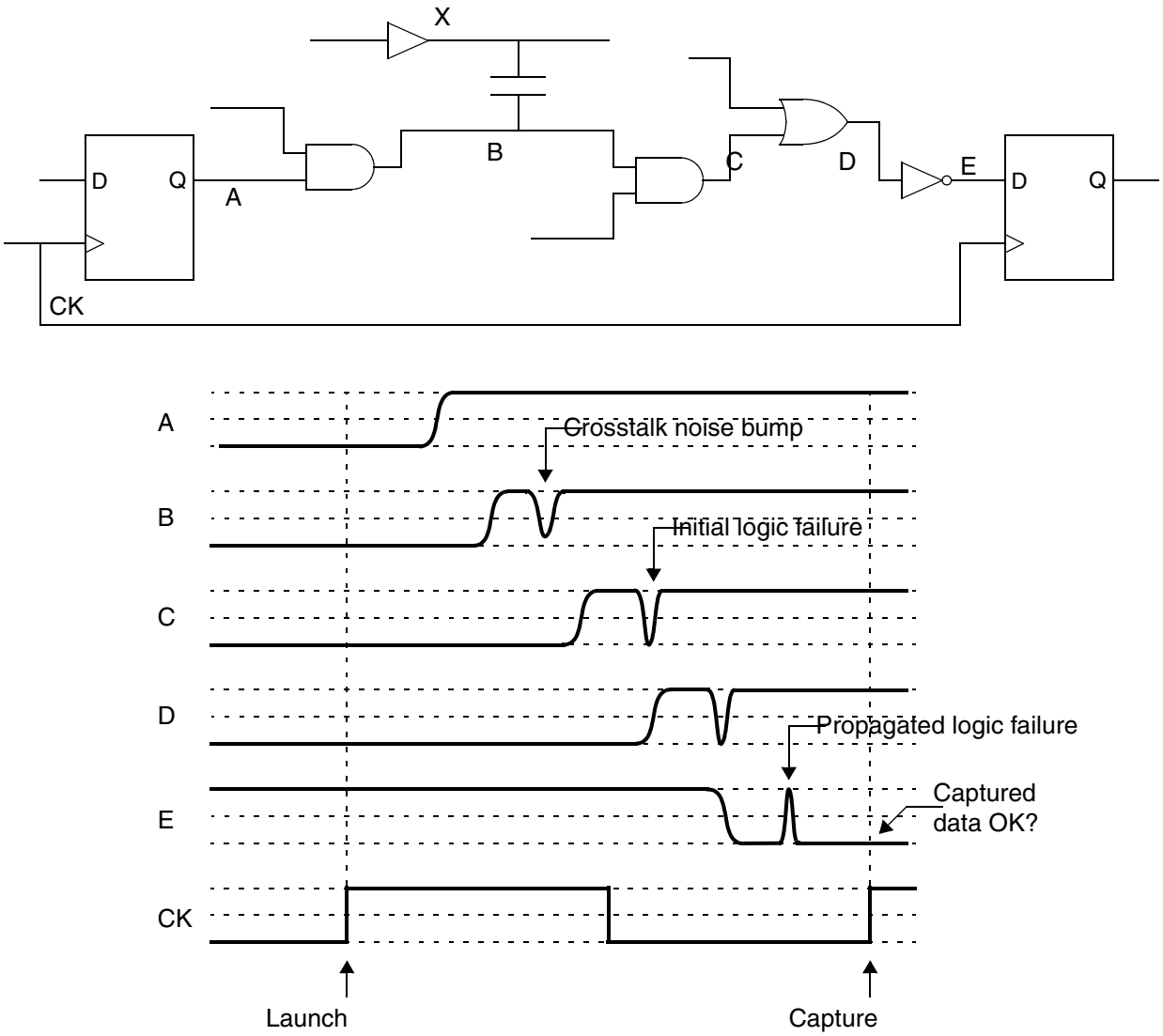
In some cases, propagated noise exists in the design but cannot be calculated. For example if the cell is an extracted timing model or a black-box model, you can specify a noise bump explicitly on any pin or port in the design by using a PrimeTime SI command, `set_input_noise`. This is called user-defined noise. User-defined noise can either override or add to any propagated noise for the applicable pin or port.

Noise-Related Logic Failures

In static noise analysis, a logic failure is an incorrect logic value on a net resulting from the propagation of a noise bump from an input to an output of a cell.

A logic failure does not necessarily result in functional failure of the device. For example, consider the circuit shown in Figure 6-5. A rising edge of clock CK launches data through a combinational path from net A to net E. Crosstalk from net X causes a large noise bump on net B, causing a logic failure on net C. The logic failure is propagated down the path to the endpoint at net E.

Figure 6-5 Propagated Logic Failure



If the logic failure is present on net E when the capture edge occurs, an error is latched, resulting in functional failure of the device. On the other hand, if the incorrect value becomes correct again before the capture edge occurs, the device will work properly.

There are two possible strategies for repairing logic failures:

- Fix logic failures that are latched and ignore logic glitches that are not latched. The latched violations are reported in the “report at endpoint” mode.

- Fix all logic failures at the source, whether or not they appear to be latched. All logic violations caused by noise are reported in the “report at source” mode. This is the default mode.

The reporting mode is controlled by the `set_noise_parameters` command. In the default (report at source) mode, when PrimeTime SI detects a logic failure, it does not propagate that failure forward through the path, based on the assumption that the failure will be fixed at the source. Instead, it reduces the size of the input noise bump to a fraction of the failure level (0.75 by default) so that noise analysis can continue for cells and nets in the fanout of the failure.

There are several ways to fix a crosstalk noise problem, such as changing the physical routing to avoid cross-coupling, inserting a buffer at the victim net, or increasing the strength of the victim net driver. The fixed design should be analyzed again to confirm the fix and to check for any new crosstalk problems.

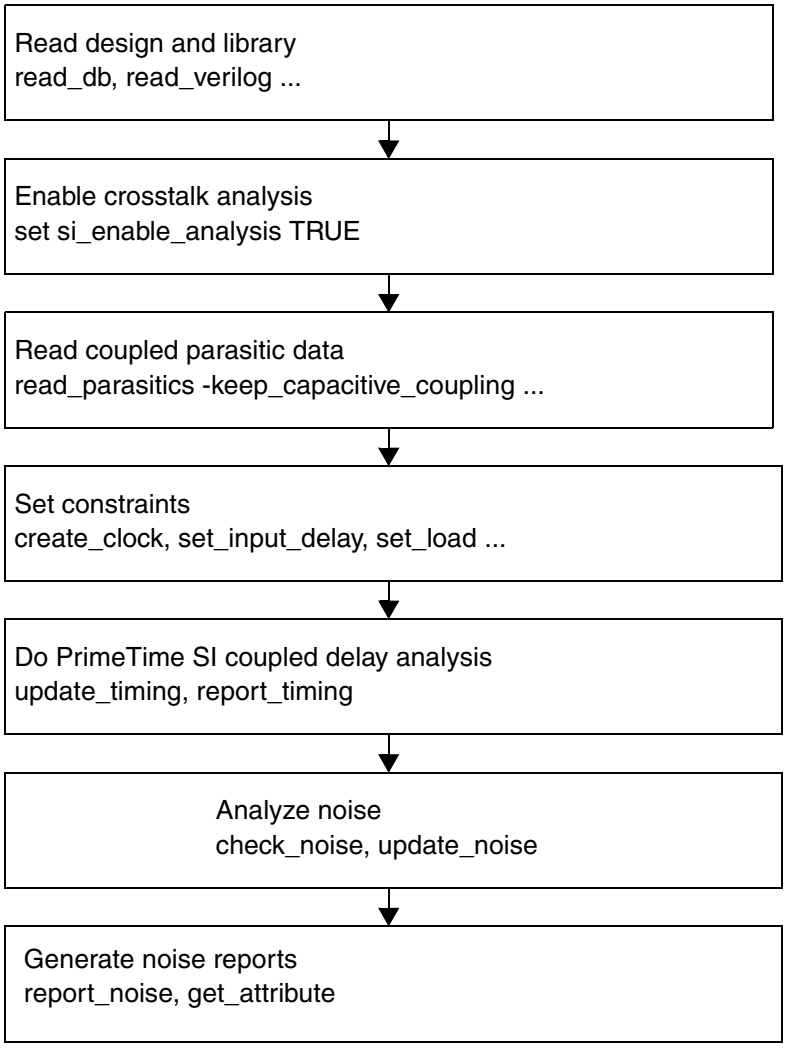
PrimeTime SI Noise Analysis Flow

Static noise analysis with PrimeTime requires a PrimeTime SI license. Crosstalk analysis must be enabled by setting the `si_enable_analysis` variable to `true`.

The typical analysis flow is the same as for an PrimeTime SI static timing analysis, with the addition of the `check_noise`, `update_noise`, and `report_noise` commands at the end of the flow. The `check_noise` command checks for the presence of noise models at the driver and load pins the design. The `update_noise` command performs static noise analysis using the aggressor timing windows previously determined by timing analysis. The `report_noise` command generates a noise report showing the locations and values of the worst-case noise bumps.

The noise analysis flow is summarized in Figure 6-6.

Figure 6-6 Noise Analysis Flows



PrimeTime SI supports two types of library noise models, called CCS (composite current source) noise and NLDM (nonlinear delay model). CCS noise is the newer, more advanced technology that provides greater accuracy by calculating the actual response for each cell and each noise bump with SPICE-like accuracy, but with the speed of static timing analysis. CCS noise also offers very fast library characterization. NLDM is the older technology that provides good results for technologies above 65 nm.

Noise Analysis Commands

PrimeTime SI has commands to invoke noise analysis, generate noise reports, specify noise bumps, and specify cell noise response characteristics. The commands are summarized in Table 6-1 and described in the following subsections.

Table 6-1 Noise Analysis Commands

Command	Purpose
`check_noise`	Checks for the presence and validity of noise models on the driver and load pins in the design, and reports any pins that are not constrained for noise.
`update_noise`	Performs a noise analysis using the parameters set with the `set_noise_parameters` command. Performs a timing update (`update_timing`) if needed to get arrival window data.
`set_noise_parameters`	Enables or disables the noise analysis options for subsequent analysis: effort level, aggressor arrival times, beyond-rail analysis, noise propagation, and source/endpoint noise reporting.
`report_noise_parameters`	Reports the settings made with the `set_noise_parameters` command.
`reset_noise_parameters`	Resets all the `set_noise_parameters` parameters to their default settings.
`set_si_noise_analysis`	Includes or excludes specified nets for crosstalk noise analysis.
`remove_si_noise_analysis`	Reverses the effects of the `set_si_noise_analysis` command.
`report_noise`	Generates a report on noise effects, including the width, height, and slack of worst-case noise bumps.
`report_noise_calculation`	Generates a detailed report on the calculation of noise effects for a specified net or pin and noise bump type. The report shows the reasons for that individual aggressors are active or inactive, the noise contribution from individual sources (aggressors, propagated noise, user-specified noise), and other information used for noise bump calculation.

Table 6-1 Noise Analysis Commands (Continued)

Command	Purpose
`report_noise_violation_sources`	Reports the noise violation sources that are propagated to failing noise endpoints; used only in the “report at endpoint” analysis mode.
`get_noise_violation_sources`	Creates a collection of noise violation sources that are propagated to failing noise endpoints; used only in the “report at endpoint” analysis mode.
`get_attribute si_noise_*`	For a specified pin in the design, returns a collection of active aggressor nets or returns a string that lists the aggressor nets and their corresponding coupled bump height and width values.
`set_noise_derate`	Derates (modifies) the calculated noise height or width by a specified fraction or absolute amount.
`set_input_noise`	Annotates a noise bump with specified width and height characteristics on a port or pin in the design, either overriding or adding to any propagated noise at that location.
`remove_input_noise`	Removes noise bump annotations previously set on ports or pins with the `set_input_noise` command.
`set_noise_lib_pin`	Sets the noise characteristics of an input pin or output pin in the design to match the noise characteristics of a specified library pin.
`remove_noise_lib_pin`	Reverses the effects of the `set_noise_lib_pin` command.
`set_noise_immunity_curve`	Specifies the three coefficient values that determine the noise immunity curve for an input port of the design or an input pin of a library cell. PrimeTime SI uses this information to determine whether a noise bump of a given height and width causes a logical failure.
`remove_noise_immunity_curve`	Removes noise immunity information previously set on ports or pins with the `set_noise_immunity_curve` command.
`set_noise_margin`	Specifies the bump-height noise margins for an input port of the design or an input pin of a library cell. PrimeTime SI uses this information to determine whether a noise bump of a given height at a cell input causes a logical failure at the cell output.

Table 6-1 Noise Analysis Commands (Continued)

Command	Purpose
`remove_noise_margin`	Removes noise margins previously set on ports or pins with the `set_noise_margin` command.
`set_steady_state_resistance`	Specifies the steady-state drive resistance for an output port of the design or an output pin of a library cell. PrimeTime SI uses this information to calculate crosstalk noise bump characteristics.
`remove_steady_state_resistance`	Removes steady-state resistance information previously set on cells with the `set_noise_margin` command.
`set_si_noise_disable_statistical`	Disables statistical analysis for the specified nets when using composite aggressor analysis.
`remove_si_noise_disable_statistical`	Reverses the effects of the `set_si_noise_disable_statistical` command.

You can get detailed information on the commands by looking at the man pages for the commands. For information on noise attributes, see "Noise Attributes" on page 6-25.

Invoking Noise Analysis

You invoke noise analysis by using the `check_noise`, `update_noise`, and `report_noise` commands, which operate like `check_timing`, `update_timing` and `report_timing`, but for noise analysis rather than timing analysis. The `si_enable_analysis` variable must be set to true.

The `update_noise` command performs crosstalk noise analysis of the current design. It invokes a full timing update (like using `update_timing`) if a timing update has not already been done. After completion of the noise analysis, you can report the results with the `report_noise` command.

You can set the parameters for noise analysis with the `set_noise_parameters` command. To view the current settings, use the `report_noise_parameters` command. To return all noise parameters to their default settings, use the `reset_noise_parameters` command.

To explicitly exclude specific nets from crosstalk noise analysis, use the `set_si_noise_analysis` command. This command works very much like the `set_si_delay_analysis` command, except that it applies to crosstalk noise analysis rather than crosstalk delay analysis. To reverse the effects of the command, use the `remove_si_noise_analysis` command.

check_noise Command

The `check_noise` command can be executed before `update_noise` to validate the correctness of a design with respect to noise analysis. It checks for the presence and validity of noise models on all load pins and driver pins in the design, and reports any pins that are not constrained for noise analysis. Running `check_noise` can quickly detect many types of noise modeling problems before you spend time using the `update_noise` command.

This is the syntax of the `check_noise` command:

```
check_noise
  [-include { noise_immunity | noise_driver } ]
  [-beyond_rail]
  [-verbose]
  [-nosplit]
  [-si_noise_immunity_default_height_ratio]
```

The `-include` option lets you specify which types of noise model checking to perform, noise immunity on load pins or noise models on driver pins, or both. By default, only noise immunity checking is performed.

The `-beyond_rail` option causes checking for beyond-rail as well as between-rail noise analysis. By default, only between-rail noise checking is performed.

The `-verbose` option reports individual pins as well as generating a summary report.

The `-nosplit` option prevents the splitting of long lines in the report.

The -`si_noise_immunity_default_height_ratio` variable controls the default margin value. The default value of this variable is 0.4 (40% of VDD). If the variable is set to 1.0, no default noise immunity will be used (backward compatibility).

By default, a `check_noise` report summarizes the number of driver and load pins with each type of noise constraint, as shown in the following example:

```
Noise driver pin check:

  Noise driver type              above_low  below_high
  ---------------------------------------------------
  CCS noise                             11          11
  library IV curve                       0           0
  library resistance                     0           0
```

```
Noise load pin immunity check:

  Noise immunity type            above_low  below_high
  ---------------------------------------------------
  user hyperbolic curve                  1           1
  user margin                            2           2
  library immunity table                 0           0
  library hyperbolic curve               0           0
  library CCS noise immunity            14          14
  library DC noise margin                0           0
  default margin                         5           5
  none                                   0           0
```

To ensure detection of noise violations throughout the design, verify that all pins are constrained for noise analysis. The pins reported in the "default margin" are not constrained, so the default margin value will be used. The number of pins reported in the “none” row of the report must be zero. For information about specifying noise immunity, see “Noise Immunity” on page 6-38.

The `check_noise` command is typically used as follows:

1. After crosstalk timing analysis, but before noise analysis, run `check_noise` to check all driver and load pins for noise constraints. If any pins are found without noise constraints, run `check_noise` with the `-verbose` option to get a detailed list of pins that lack constraints.
2. Constrain the pins for noise as needed, using commands such as `set_noise_lib_pin`, `set_noise_margin`, and `set_noise_immunity_curve`.
3. Run `check_noise` again to verify that all pins are constrained for noise analysis.

When `check_noise` confirms that all pins are constrained for noise, you can proceed to noise analysis with the `update_noise` command.

Noise Analysis Effort

The `-analysis_effort` option of the `set_noise_parameters` command lets you set the noise analysis effort to either `low` or `high` to trade off accuracy and performance. The default setting is `high`.

In the `low` mode, PrimeTime SI uses a simpler but faster noise waveform calculator for all noise bump calculations. This mode is suitable for an initial analysis when you want to quickly find the worst noise violations or when you expect no violations.

In the `high` mode, PrimeTime SI still uses the simpler and faster algorithm to calculate all noise bumps initially, which is accurate for smaller noise bumps. However, for noise bumps found to exceed a certain threshold, it passes the calculation to a more complex calculator to get better accuracy for these more significant noise bumps. This adaptive mode should be used for final sign-off analysis and for circuit simulator correlation.

Aggressor Arrival Times

If you use the `-ignore_arrival` option of the `set_noise_ parameters` command, PrimeTime SI ignores aggressor arrival windows and assumes that all aggressor nets switch at the same time for each victim net throughout the design, resulting in a conservative analysis. If noise violations are reported, you can run another analysis using arrival windows to see if the violations are eliminated.

Beyond-Rail Analysis

By default, an `update_noise` analysis only considers between-the-rails noise bumps, which are typically the more significant ones to consider. However, beyond-rail noise bumps (above high and below low) can also cause incorrect data to be latched due to forward-biasing of pass transistors at flip-flop and latch inputs. To have PrimeTime SI consider beyond-rail noise conditions at the cost of additional runtime, use the `-include_beyond_rails` option of the `set_noise_parameters` command.

Noise Propagation

By default, PrimeTime SI considers only crosstalk noise and user-specified noise, not propagated noise. This default mode is suitable for an initial analysis, when you want to quickly find the worst violations. For a more accurate noise analysis at the cost of additional runtime, enable noise propagation analysis by using the `-enable_propagation` option of the `set_noise_parameters` command.

Reporting Noise at Source or Endpoint

PrimeTime SI offers two analysis reporting modes, called “report at source” and “report at endpoint.” In the “report at source” mode, PrimeTime SI reports violations at the source of each noise violation. In the “report at endpoint” mode, PrimeTime SI reports violations only at endpoints where noise propagation stops, such as sequential cells.

Figure 6-7 and Figure 6-8 illustrate the two noise reporting modes.

Figure 6-7 Report at Source

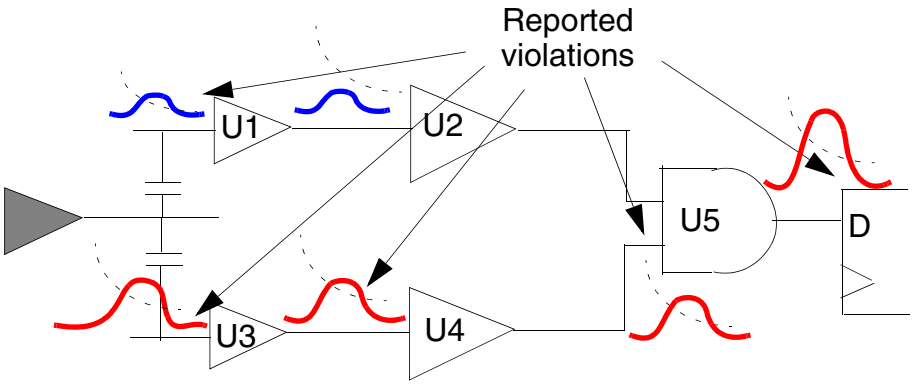
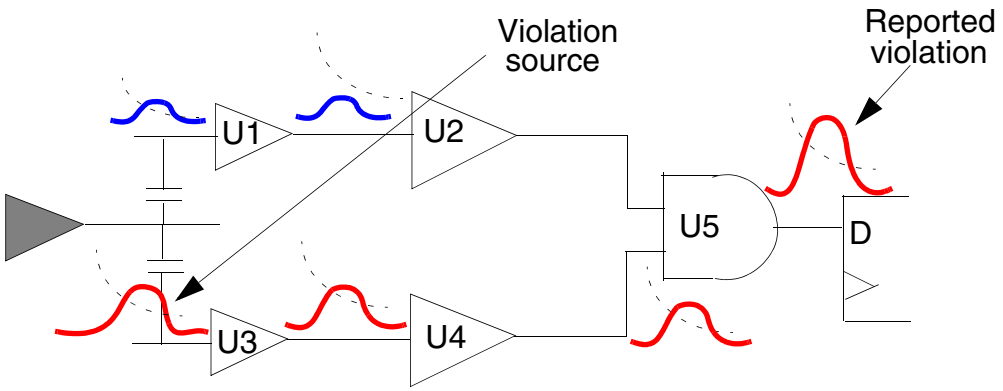


Figure 6-8 Report at Endpoint



In both figures, a violation occurs when the noise bump crosses the dotted noise immunity curve. The noise injected at the input of U3 is propagated through U4 and U5 and reaches D. The noise injected at U1, however, is not propagated beyond U2 because of the noise immunity of U2.

As shown in Figure 6-7 the input pins of U3, U4, U5, and D in the "report at source" mode are reported as violations because they each have negative slack. In the “report at endpoint” mode, as shown in Figure 6-8, only the input pin of D, the noise propagation endpoint, is reported as a violation.

A noise startpoint can be:

- An output pin of a flip-flop
- An output pin of a level-sensitive latch
- An input port

- An output pin of a multistage cell, which is a cell with multiple levels of transistor stages, such as a flip-flop or macro

A noise endpoint can be:

- A data, clock, or asynchronous pin of a flip-flop
- An input pin of a level-sensitive latch
- An output port
- Any combinational logic pin where the noise exceeds a threshold of 75 percent of Vdd, which can be controlled by setting the `si_noise_endpoint_height_threshold_ratio` variable
- An input pin of a multi-stage cell

The “report at endpoint” mode produces a more concise report because it includes only noise violations that are latched as incorrect data. The “report at source” mode produces a more complete report that includes all noise immunity violations, whether or not they are latched. The default mode is “report at source.”

To set the reporting mode to “report at endpoint,” use the following command:

```
pt_shell> set_noise_parameters \
          -analysis_mode report_at_endpoint
```

To set the reporting mode back to the default, “report at source”, use the following command:

```
pt_shell> set_noise_parameters \
          -analysis_mode report_at_source
```

Changing the mode requires a full timing update, so to save time, set the desired mode before you use the `update_noise` command.

In the “report at endpoint” mode, if you get a report of endpoint violations and you want identify the sources that caused the violations, you can use the `report_noise_violation_sources` command or the `get_noise_violation_sources` command. For details, see the man pages for these commands.

Derating Noise Results

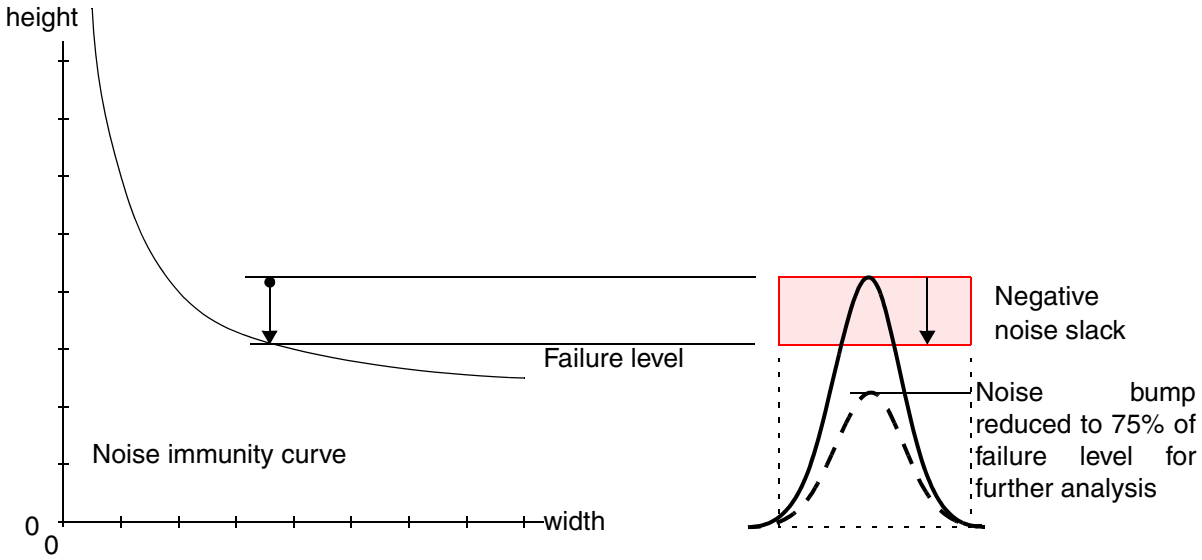
The `set_noise_derate` command modifies the calculated noise bump sizes by a specified factor or absolute amount. It works like the `set_timing_derate` command, except that it modifies noise bump sizes rather than path delays. The derating factors affect the reported bump sizes and noise slacks. In the command, you specify the types of noise bumps

(above/below high/low), the parameters to be modified (height offset, height factor, and width factor), and the factor or absolute amount of modification. For details, see the man page for the command.

Noise Failure Propagation Limit

In the "report at source" mode, when PrimeTime SI detects a functional failure caused by a noise bump, it does not propagate the incorrect logic value forward through the path. Instead, it reduces the size of the input noise bump to a fraction of the failure level so that noise analysis can continue for cells and nets in the fanout of the failure. The default factor is 0.75, as shown in Figure 6-9.

Figure 6-9 Input Noise Bump Reduction for Propagation Analysis



To specify a different reduction factor, set the `si_noise_limit_propagation_ratio` variable to the desired ratio, which must be a value between 0.0 and 1.0. Note that in the "report at endpoint" mode, the entire noise bump is propagated, so this variable has no effect.

Irrespective of this variable setting, the height of any propagated noise bump is limited to Vdd (the rail-to-rail voltage swing).

Noise Analysis with Composite Aggressors

Some complex designs require an efficient method of analyzing multiple aggressors to a single victim net. In general, such designs have the following characteristics:

- A large number of aggressors per victim net
- Little or no filtering of aggressors

- Noise calculations are performed in high effort mode

If your design has these characteristics, you may wish to use composite aggressor analysis. Composite aggressor analysis aggregates the effects of multiple aggressors into a single “virtual” aggressor, thereby reducing the computational complexity cost and improving the runtime and memory utilization without impacting accuracy.

To enable this mode, set the `si_noise_composite_aggr_mode` variable to `normal` or `statistical`. The statistical mode applies statistical analysis to the composite aggressor to reduce its overall impact on the victim net. By default, the variable is set to `disabled`, which disables composite aggressor analysis.

If you would like to use statistical mode, but have certain nets in your design for which you do not wish to statistically reduce the aggressor impact, you can disable statistical analysis for just those nets by using the following commands. These commands take a list of nets as an argument, and have no effect if a net is not part of the composite aggressor group:

- `set_si_noise_disable_statistical`
- `remove_si_noise_disable_statistical`

To see which aggressors are part of a composite aggressor, use the `report_noise_calculation` command. Aggressors that are part of a composite aggressor are labeled with a “C” in the report.

Incremental Noise Analysis

After a full noise update with the `update_noise` command, PrimeTime SI can sometimes perform a fast “incremental” noise update to analyze the effects of certain design changes. An incremental update takes just a fraction of the time needed for a full noise update, because only the portions of the design affected by the changes are analyzed.

Depending on previous usage of `update_noise` and the types of changes performed on the design, an `update_noise` command might invoke an incremental noise update, a full noise update, an incremental timing update with a full noise update, or a full timing update with a full noise update. PrimeTime SI generally uses the fastest possible type of update that gives accurate results.

You can force a complete noise update, irrespective of the changes made to the design, by using the `-full` option of `update_noise`. In that case, if a timing update is necessary due to a change such as `size_cell`, PrimeTime SI performs a full (not incremental) timing update as well.

Reporting Noise Analysis Results

The main command for reporting analysis results is `report_noise`. Detailed noise analysis information is available by using the `report_noise_calculation` command and by reporting pin attributes with the `get_attribute` command.

report_noise

The `report_noise` command provides information on the worst-case noise bumps on one or more nets. The command options let you specify the types of noise reported (above/below high/low); the pins, ports, or nets of the design to be reported; and the type of information included in the report.

If a noise update has not been done already, using `report_noise` invokes `update_noise` to generate the timing window information needed for noise analysis.

By default, the `report_noise` command by itself, without any options, reports the bump characteristics and noise slack for the pin with the smallest (or most negative) slack for each type of noise bump. For example:

```
pt_shell> report_noise
 ...
 analysis_mode report_at_source
 slack type: area

 noise_region: above_low
 pin name (net name)       width    height     slack
 ---------------------------------------------------
 tmp2f_reg/D (buf2_1f)      0.07      0.12      0.12

 noise_region: below_high
 pin name (net name)       width    height     slack
 ---------------------------------------------------
 U3/A (buf0_2f)             0.22      0.03      0.37
 ...
```

Width values are in library time units such as nanoseconds, height values are in library voltage units such as volts, and noise slack area values are in library voltage-time units such as volt-nsec.

To restrict the scope of the report to certain types of noise bumps, use `-above` or `-below` and `-high` or `-low`. For example:

```
pt_shell> report_noise -above -low
 ...

noise_region: above_low
 pin name (net name)       width    height     slack
```

```
 ---------------------------------------------------
 tmp2f_reg/D (buf2_1f)      0.07      0.12      0.12
```

To report on multiple pins having the worst noise slack, use the `-nworst n` option. For example:

```
pt_shell> report_noise -above -low -nworst 4
 ...

 noise_region: above_low
 pin name (net name)        width    height     slack
 ---------------------------------------------------
 tmp2f_reg/D (buf2_1f)       0.07      0.12      0.12
 U1/A (tmp2f)                0.24      0.01      0.39
 U2/A (tmp2f)                0.24      0.01      0.39
 U3/A (buf0_2f)              0.23      0.01      0.40
```

The `-slack_type` option specifies the method used for measuring noise slack: `height`, `area`, or `area_percent`, as described in the section "Noise Slack" on page 6-45. The default is `area`. The `-slack_lesser_than` option restricts the report to pins that have noise slack worse than a specified amount. The `-all_violators` option restricts the report to pins that have negative noise slack.

To restrict the scope of the report to specific pins, ports, or nets, specify a list of objects in the command. If the specified object is a pin, the command reports the noise on that pin. The specified pin should be a load pin (a cell input pin or bidirectional pin, not an output pin). If the specified object is a net, the command reports the noise on all load pins in the net.

For example, to restrict the scope of the report to load pins in net1 and net2:

```
pt_shell> report_noise -above -low {net1 net2}
```

To restrict the scope of the report to just the data pins, clock pins, or asynchronous pins of all registers, use the option `-data_pins`, `-clock_pins`, or `-asynch_pins`. For example:

```
pt_shell> report_noise -data_pins
```

To get a more detailed report showing separately the noise contribution of different aggressor nets and noise propagation, use the `-verbose` option. For example:

```
pt_shell> report_noise -verbose -above -low
 ...

 noise_region: above_low
 pin name (net name)       width    height     slack
 -----------------------------------------------------
 tmp2f_reg/D (buf2_1f)
 Aggressors:
 CK                         0.07      0.12
 net2                       0.04      0.02
 Total:                     0.07      0.12      0.12
```

Information on the calculation of these noise bumps is available by using the `report_noise_calculation` command.

Noise bump on the data pin of the flop is reported only if the noise bump overlaps with the setup-hold window of the data pin. If the noise bump is induced away from the setup-hold window, then the noise bump is not reported. In such cases, the `report_noise_calculation` or `report_noise -verbose` commands will report the aggressors and their contribution, but the total noise bump will be reported as 0. This not only applies to the data pin of the flop, but also to the data pin of any latch device that has setup-hold window.

When CCS noise models are used, the `report_noise` command reports slack as “positive” for small bumps where the exact slack number is not important. If you want to see exact numbers or noise information on non-endpoint pins under these conditions, use the `report_noise_calculation` command.

report_noise_calculation

The `report_noise_calculation` command generates a detailed report on the calculation of the noise bump on a net arc. In the command, you specify the type of noise bump to be reported (above/below high/low), the startpoint of the net arc, and endpoint of the net arc. The startpoint is the driver pin or driver port of a victim net and the endpoint is a load pin or load port on the same net.

Here is an example:

```
pt_shell> report_noise_calculation -below -high \
          -from buf2/ZN -to buf5/I
```

The command generates a report similar to the following:

```
Units:  1ns  1pF  1kOhm

Analysis mode                        : report_at_source
Region                               : below_high
Victim driver pin                    : buf2/ZN
Victim driver library cell           : mylib/INVD3
```

```
Victim net                              : I2
Victim driver effective capacitance     : 0.200827

Steady state resistance source          : library set CCS
                                          noise iv curve
Driver voltage swing                    : 1.080000
Noise derate height offset              : 0.000000
Noise derate height scale factor        : 1.000000
Noise derate width scale factor         : 1.000000
Noise effort threshold                  : 0.000000
Noise composite aggressor mode          : disabled

Noise calculations:

Attributes:
  A - aggressor is active
  C - aggressor is a composite aggressor
  D - aggressor is analyzed with detailed engine
  E - aggressor is screened due to user exclusion
  G - aggressor is analyzed with gate level simulator
  I - aggressor has infinite window
  L - aggressor is screened due to logical correlation
  S - aggressor is screened due to small bump height
  X - aggressor is screened due to aggressor exclusion

             Height    Width     Area      Aggressor Attr.
-------------------------------------------------------------
 Aggressors:
 I1          0.300604  0.786466  0.118207  A I D
 I3          0.300604  0.786466  0.118207  A I D
 Total:      0.497124  0.919737  0.228612  G

Noise slack calculation:

Constraint type: library CCS noise immunity

             Height              Area
-------------------------------------------------------------
 Required    0.581570            (0.581570 * 0.919737)
 Actual      0.497124            (0.497124 * 0.919737)
-------------------------------------------------------------
 Slack       0.084445            0.077668
```

The command uses the noise analysis settings set by the `set_noise_parameters` command. It invokes `report_noise` if a noise update has not been performed already.

The report first identifies the victim driver net and pin. It also shows any derating factors set by the `set_noise_derate` command. Then it shows the noise bumps resulting from each aggressor net and from propagation of noise from the previous stage of the driver.

A column labeled “Aggressor Attributes” shows additional information about each effective aggressor. (Aggressors that have been removed by filtering are not included in this report.) The letters in this column indicate the following conditions:

- A: The aggressor net is active. It contributes to the worst-case noise bump, taking into consideration the possible overlap times of all aggressor nets.
- C: The aggressor is a composite aggressor composed of two or more smaller aggressors working together. See “Crosstalk Analysis with Composite Aggressors” on page 2-24.
- D: The effects of the aggressor net were calculated with the detailed (high-effort) algorithm. See “Noise Analysis Effort” on page 6-15.
- E: The aggressor net is not active because it has been excluded from consideration by `set_case_analysis` or `set_si_noise_analysis` (see “Invoking Noise Analysis” on page 6-13).
- G: The noise was analyzed by gate-level simulation using CCS noise models. See “CCS Noise Modeling” on page 6-30.
- I: The aggressor net uses an infinite timing window; it has no fixed timing relationship with the victim net. This happens when the `-ignore_arrival` mode has been set with the `set_noise_ parameters` command or when the victim net and aggressor nets are in different clock domains.
- L: The aggressor net is not active due to logical correlation with the victim net or with other aggressors. For details, see “Logical Correlation” on page 2-6.”
- S: The aggressor net is not active because it has been screened out. An internal pre-screening analysis has determined that the aggressor bump is too small to be significant.
- X: The aggressor net is not active because it has been excluded from consideration by `set_si_aggressor_exclusion`. For details, see “Excluding Aggressor-to-Aggressor Nets” on page 4-4.

Noise Attributes

Crosstalk information is stored in attributes associated with pins, nets, ports, timing points, timing arcs, library pins, and library timing arcs. To view this information, you can use the `get_attribute` command. The attributes that carry noise-related information are listed and described in Appendix A, “Crosstalk Attributes”.

If a noise update has not been done already, requesting a noise attribute might invoke `update_noise` to generate the analysis results needed to determine the attribute value.

For information on attributes and how to extract attribute information from the design, see the chapter called “Object Attributes” in the *PrimeTime Advanced Timing Analysis User Guide*.

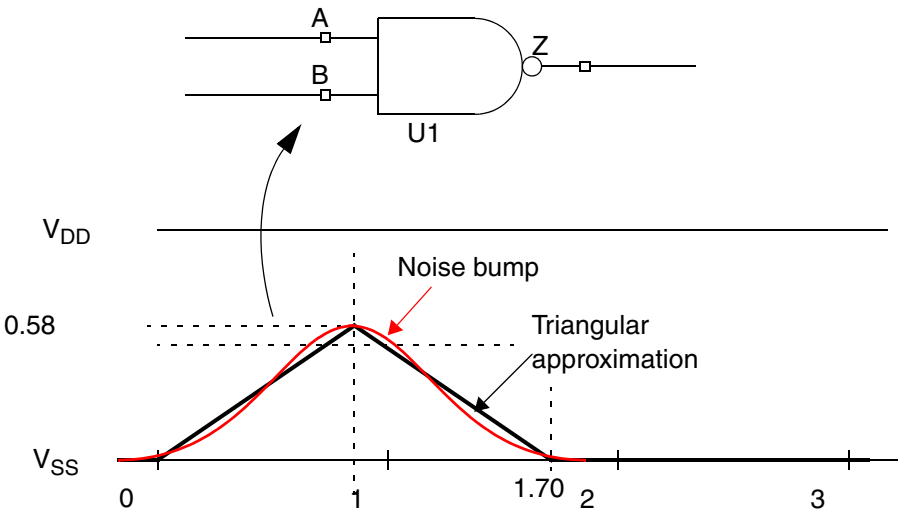
Setting Noise Bumps

Rather than allow PrimeTime SI to calculate propagated noise bumps, you can explicitly specify a noise bump at any port or pin. To do so, use the `set_input_noise` command. In the command, you specify the port or pin in the design on which to apply the noise and the characteristics of the noise bump: the type (above/below high/low), the width in library time units, and the height in library voltage units (always entered as a positive number). For example:

```
pt_shell> set_input_noise -above -low \
            -width 1.70 -height 0.58 [get_pins U1/B]
```

This creates a noise bump on pin U1/B with the waveform characteristics shown in Figure 6-10.

Figure 6-10 Noise Bump Created With the set_input_noise Command



PrimeTime SI treats this injected noise as propagated noise from the driver of the net connected to the pin. By default, this noise bump overrides any propagated noise that would otherwise be calculated from the driver of the net. However, if you use the `-add_noise` option in the `set_input_noise` command, the noise is added to any existing propagated noise or previously added noise.

You can specify propagated noise on an output pin rather than a load pin. In that case, unless you use the `-add_noise` option, the specified noise bump overrides any propagated noise that would otherwise be calculated from the driver. The specified noise bump is propagated through the subnets and attenuated by the parasitic capacitance and resistance specified for the net, until the propagated noise reaches each of load pin on the net.

When extracted timing models or other hierarchical timing models are used in a design, the specified noise bump can be used to model the worst-case noise that can occur at each output port of the timing model. The `extract_model` command includes a set of `set_input_noise` commands in the output constraint script to model the propagated noise at the output pins of the model.

Noise Analysis with Incomplete Library Data

To specify cell noise response characteristics in the absence of library-specified characteristics, or to override the library-specified characteristics at specified points in the design, you can use the following commands:

- `set_noise_lib_pin`
- `set_noise_immunity_curve`
- `set_noise_margin`
- `set_steady_state_resistance`

These are the corresponding commands for removing previous command-specified noise response characteristics:

- `remove_noise_lib_pin`
- `remove_noise_immunity_curve`
- `remove_noise_margin`
- `remove_steady_state_resistance`

set_noise_lib_pin

The `set_noise_lib_pin` command specifies that the noise characteristics of a pin in the design is same as the noise characteristics of a library pin. This command is useful when some of the library cells have noise information, but some cells do not. For example, with the following command, the input pin A of the `macro_cell` cell inherits the noise characteristics of the input pin I of the library cell `ccs_noise_lib/inv`. This information is used to calculate the noise immunity of the pin `macro_cell/A`.

```
pt_shell > set_noise_lib_pin [get_pins macro_cell/A] ccs_noise_lib/inv/I
```

Similarly, with the following command, the output pin `macro_cell/Z` inherits the noise characteristics of the output pin Z of the library cell `ccs_noise_lib/buf`. This information is used to calculate the noise bump induced on the net driven by `macro_cell/Z`.

```
pt_shell> set_noise_lib_pin [get_pins macro_cell/Z] ccs_noise_lib/buf/Z
```

set_noise_immunity_curve

The `set_noise_immunity_curve` command specifies the noise immunity characteristics at an input of a library cell or at an input port of the design. PrimeTime SI uses this information to determine whether a noise bump at the input will cause a logic failure. For a description of what is considered a logic failure, see "Noise Immunity" on page 6-38.

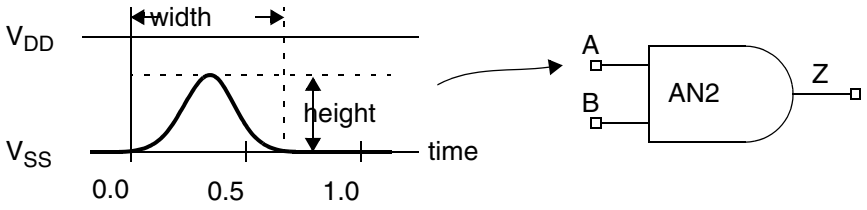
A noise immunity curve specifies the largest allowable noise bump that can occur at the input in terms of width and height. The curve shows the maximum allowable bump height as a function of the bump width. This function is established by three coefficients, as described in the section "Noise Immunity Curves" on page 6-40.

In the `set_noise_immunity_curve` command, you specify the type of noise bump (above/below high/low), the three coefficient values that define the curve, and the pin of a library cell or the input port the design to which the curve applies. For example:

```
pt_shell> set_noise_immunity_curve -above -low \
            -width 0.00 -height 0.58 -area 0.0064 \
            lib_name/AN2/A
```

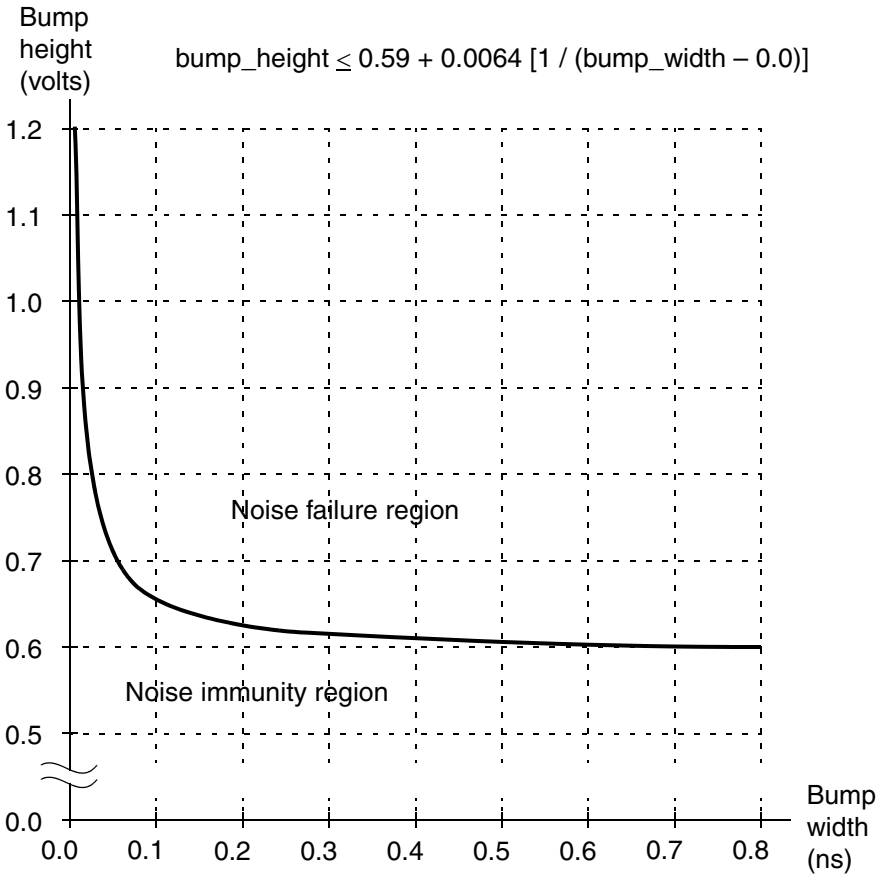
This defines the above-low noise immunity curve for input A of library cell AN2, as indicated in Figure 6-11. The resulting noise immunity curve is plotted in Figure 6-12.

Figure 6-11 Command-Specified Noise Immunity Curve




```
set_noise_immunity_curve -above -low \
  -width 0.0 -height 0.58 -area 0.0064 lib_name/AN2/A
```

Figure 6-12 Plot of Command-Specified Noise Immunity Curve



In order to fully specify the noise immunity characteristics of a cell or design, you need four commands per input: one each for above-low, below-high, above-high, and below-low noise bumps at the input. All coefficients are entered as positive numbers.

For more information, see “Noise Immunity Curves” on page 6-40.

set_noise_margin

Where there is no noise immunity curve specified by the library or by the `set_noise_immunity_curve` command, PrimeTime SI uses noise margins based on bump heights, as described in the section “Bump Height Noise Margins” on page 6-44.

The `set_noise_margin` command specifies the noise margin at an input of a library cell or at an input port of the design. For a library cell, this information overrides the library-specified voltage noise margins.

In the `set_noise_margin` command, you specify the type of noise bump (above/below high/low), the bump height failure threshold for the input, and the input pin of a library cell or input port the design to which the setting applies. For example:

```
pt_shell> set_noise_margin -above -low 0.4 lib_name/AN2/A
```

All noise margin values are entered as positive numbers, including those for below-high and below-low noise bumps.

set_steady_state_resistance

The `set_steady_state_resistance` command specifies the drive resistance at an output of a library cell or at an output port of design. PrimeTime SI uses this information to determine the size of voltage bumps in the presence of crosstalk noise.

In the `set_steady_state_resistance` command, you specify the type of noise bump (above/below high/low), the drive resistance in library resistance units such as ohms or Kohms, and the output pin of a library cell or the output port the design to which the setting applies.

For example:

```
pt_shell> set_steady_state_resistance -above -low \
            resistance_value 0.042 \
            lib_name/AN2/Z
```

All resistance values are entered as positive numbers.

CCS Noise Modeling

CCS noise uses an advanced, current-based driver model that performs noise analysis with SPICE-like accuracy. Using an adaptive algorithm for analysis, PrimeTime SI precisely calculates not only injected crosstalk noise bumps, but also propagated noise bumps and driver weakening effects. The current-based CCS noise model provides the accuracy needed for process technologies at 65 nm and below.

With CCS noise models, PrimeTime SI computes nonlinear noise bump waveforms on-the-fly with excellent correlation with SPICE simulations at speeds capable of handling designs containing millions of gates. Unlike NLDM models that use static noise immunity curves, CCS noise models allow PrimeTime SI to calculate noise immunity dynamically, including the effects of nonmonotonic noise waveforms and noise propagation.

When CCS noise models are present, a victim net is first analyzed with a fast macro model engine. The noise bumps calculated with this engine are generally conservative. If the magnitude of the calculated noise bump is smaller than the transistor threshold voltage of

the receiver pins, no further analysis is done because such a small noise bump does not affect the output of the receiver cell. In such cases, noise slack is simply reported as "POSITIVE". On the other hand, if the noise bump calculated with the macro model engine exceeds the receiver transistor threshold voltage, the noise bumps are further calculated using a more accurate CCS noise gate-level simulation engine. This tiered approach provides a good tradeoff between accuracy and performance. The nets that have potential noise violations are analyzed with an accurate engine while the nets with smaller noise bumps are analyzed with a fast engine.

Noise Immunity

PrimeTime SI uses the following order of precedence when choosing which noise immunity information to use:

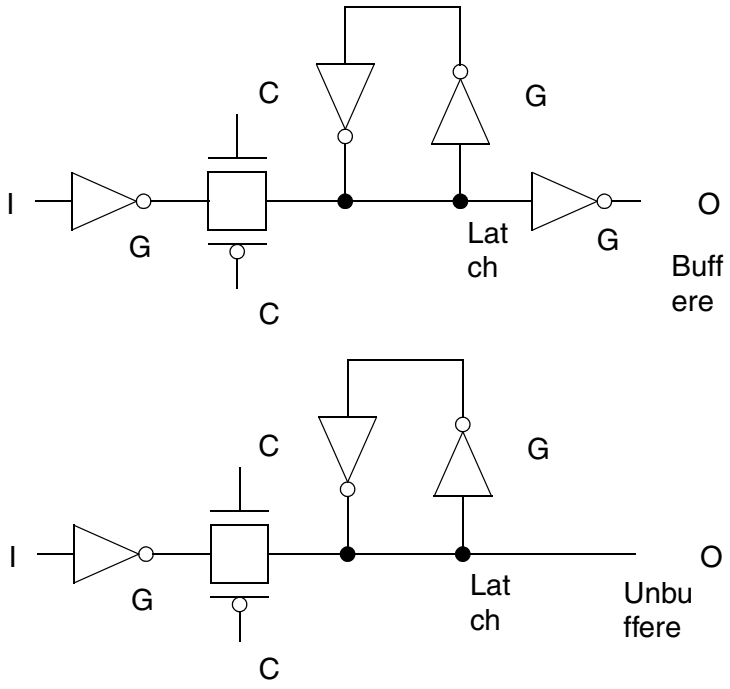
1. Static noise immunity curve annotated by the user with the `set_noise_immunity_curve` command
2. DC noise margin annotated by the user with the `set_noise_margin` command
3. Arc-specific noise immunity curve from library
4. Pin-specific noise immunity curve from library
5. CCS noise model from library
6. DC noise margin from library

For example, if you specify noise immunity curve information using the `set_noise_immunity_curve` or `set_noise_margin` command, PrimeTime SI uses that information for noise slack calculation, even if the library has CCS noise information.

CCS Noise Analysis for Unbuffered-Output Latches

Level-sensitive latches are often used in high-performance designs because they have smaller data-to-output and clock-to-output delays than flip-flops. Figure 6-13 shows two possible ways to build a level-sensitive latch, with and without an output buffer. Both of these examples use an input buffer, a pass gate, and an inverter loop that holds the latched value on a latch node.

Figure 6-13 Latch Circuits With Buffered and Unbuffered Outputs



The latch with an output buffer is resistant to noise at the output because the buffer isolates the latch node from the output node. The latch circuit without an output buffer is faster and uses less power but is very sensitive to noise at the output.

Noise analysis is performed for an output pin if its attribute is_unbuffered is set to true in the library and the pin is characterized for noise. The is_unbuffered pin attribute has been supported by Liberty CCS modeling syntax and Library Compiler since the B-2007.12 release, as described in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

You can specify the name of an unbuffered cell output pin in the `report_noise` command, producing a report similar to what you get when you specify an input pin or bidirectional pin. For example,

```
pt_shell> report_noise [get_pins I2/Z]
...

 analysis mode: report_at_source
 slack type: area

 noise_region: above_low
 pin name (net name)       width    height     slack
 -----------------------------------------------------
```

```
I2/Z (P4)                    1.03      2.03     -1.99

noise_region: below_high
pin name (net name)        width    height     slack
-------------------------------------------------------
I2/Z (P4)                    1.04      2.07     -2.04
```

NLDM Noise Modeling

When NLDM noise modeling is used, the noise response characteristics of the cells must be specified either in the Synopsys .lib technology library or by using PrimeTime SI commands. These are the types of noise response information used in noise analysis:

- The steady-state I-V (current-voltage) characteristics of the cell outputs. This information is needed to determine the size of noise bumps resulting from crosstalk
- The noise immunity characteristics of the cell inputs, either in terms of allowable noise bump heights and widths (noise immunity curves) or in terms of bump heights alone. This information is needed to determine whether a noise bump of a given size at the input will cause a logic failure at the output
- The width and height of propagated noise bumps at the cell outputs, given the width and height of the noise bumps at the cell inputs and the load on the output. This information needed to determine the size of noise bumps resulting from propagation

It is better to specify the noise response characteristics in the library. However, if the library lacks noise response information, or if you want to override the library-specified information, you can use PrimeTime SI commands to specify the steady-state I-V characteristics, noise immunity characteristics, or both.

If noise propagation information is not available in the library, you can use PrimeTime SI commands to specify explicitly the noise bumps at any ports or pins in the design.

The library syntax offers more specification features and flexibility than PrimeTime SI commands. In libraries you can specify piecewise-linear models, polynomial models, and noise propagation models of various types, in addition to the methods supported by PrimeTime SI commands.

In the absence of cell characterization data obtained in the laboratory, you can use a circuit simulator such as SPICE to get theoretical noise response characteristics, and use that information in the library or in PrimeTime SI commands that specify noise response characteristics. This process can provide detailed static noise analysis results with PrimeTime SI.

If cell noise characteristics are not specified in the library and not specified by PrimeTime SI commands, PrimeTime SI still performs noise analysis by estimating noise characteristics from the library-specified cell timing and slew characteristics. However, noise analysis under these conditions cannot detect logic failures and cannot calculate noise propagation effects.

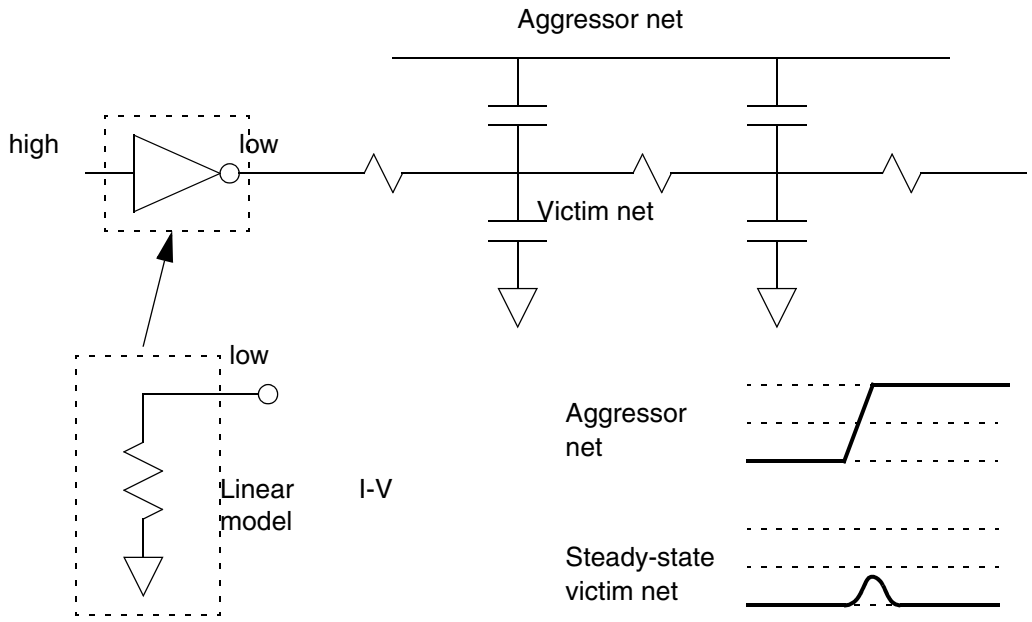
For detailed information on the library syntax, see the Synopsys Library Compiler documentation. The following subsections provide an overview of cell noise response characteristics and how to specify these characteristics in the library.

Steady-State I-V Characteristics

A steady-state driver of a net affects the size of crosstalk bumps on the net due to its loading effects on the net. PrimeTime SI needs the steady-state I-V characteristics of the driver output in order to accurately calculate the characteristics of crosstalk noise bumps.

For example, consider the crosstalk noise analysis in Figure 6-14. The driver of the victim net is steady at logic zero. When a rising transition occurs on the aggressor net, it causes an above-low noise bump on the victim net. The steady-state I-V characteristics of the driver affect the size of the size bump. The simplest model that can be used is a linear I-V curve, which is the same as a resistor, like the model shown in the diagram.

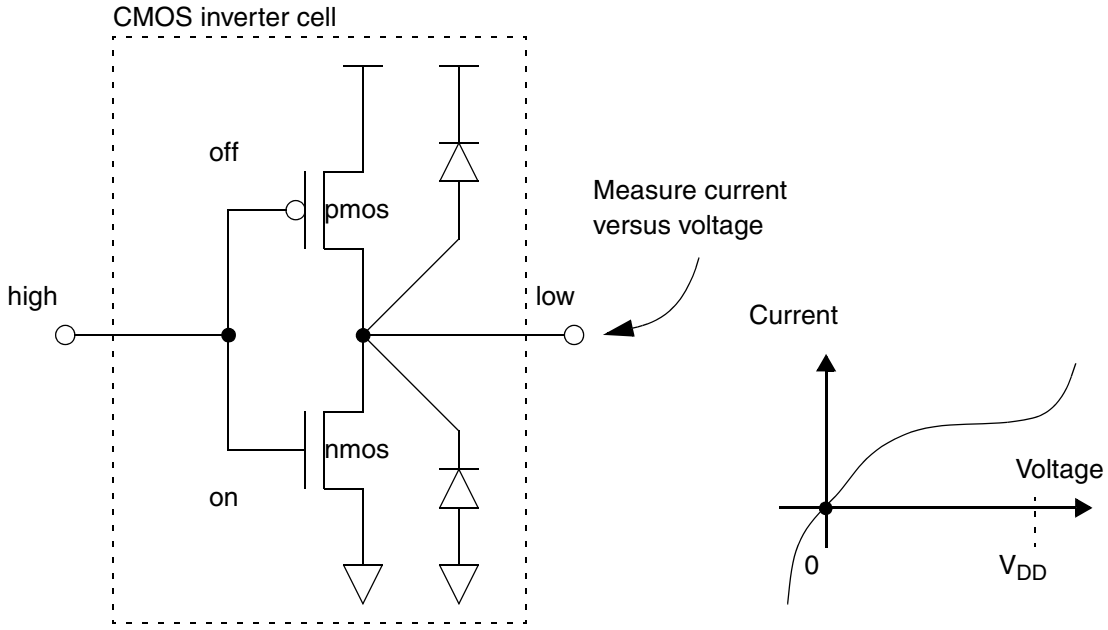
Figure 6-14 Linear I-V Model for a Cell With Steady-State Low Output



The I-V characteristics of a cell can be measured in the laboratory by placing the cell output into either the low or high state, and then measuring the current at different voltages at the output, as shown for the CMOS inverter in Figure 6-15. The circuit diagram includes the diodes at the output that exist because of the p-n isolation junctions of the pulldown and pullup transistors. These diodes affect the I-V characteristics at voltages below zero and above Vdd.

A typical CMOS output at logic zero has a steady-state operating point at (0.0, 0.0). In the small region surrounding this operating point, the output behaves like a resistor and the I-V plot looks like a straight line. However, at larger positive voltages, the I-V plot becomes nonlinear due to the NMOS transistor shutting off and because of forward-biasing of the PMOS junction diode. At voltages well below zero, the plot become nonlinear due to the forward-biasing of the NMOS junction diode.

Figure 6-15 I-V Characterization of a Steady-State Low Output

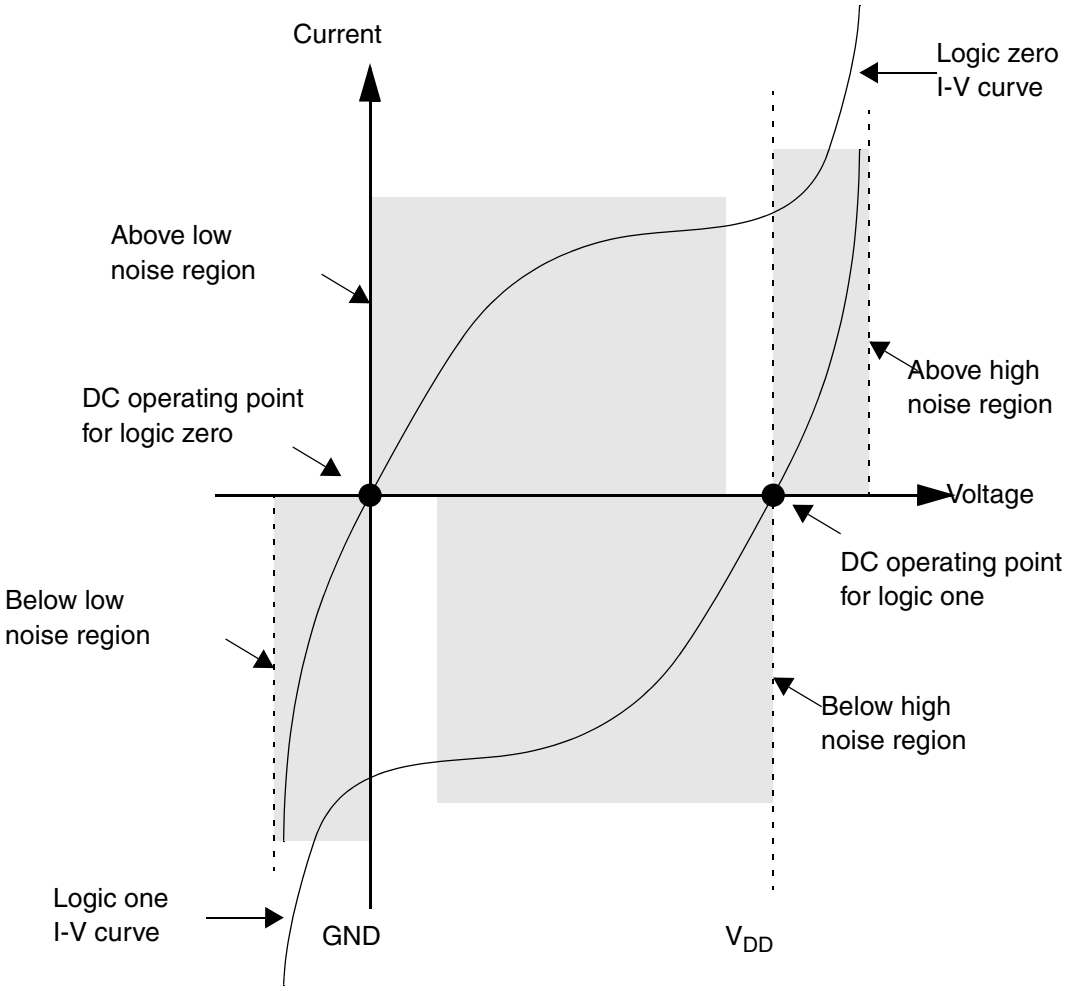


A similar I-V plot can be obtained by placing the CMOS output at logic one. Its steady-state operating point is at (Vdd, 0.0).

Figure 6-16 shows a typical plot of both the logic zero and logic one I-V curves on the same graph. For a process technology that is symmetrical between NMOS and PMOS transistor characteristics, the logic-one I-V curve is the same as the logic-zero curve rotated 180 degrees and shifted to the right by the amount Vdd.

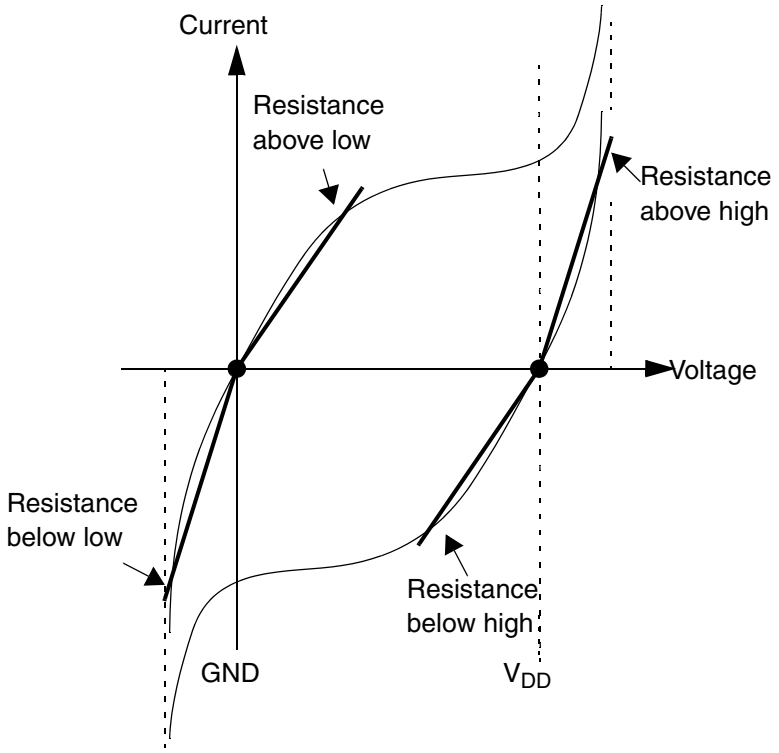
The four shaded portions indicate the operating regions when noise bumps occur: below low, above low, below high, and above high. The library syntax allows the I-V characteristics to be specified as two resistors (one each for above low and below low), as a piecewise-linear model, or as a polynomial function.

Figure 6-16 Steady-State I-V Characteristics at a CMOS Output



For example, to approximate the I-V curve using two resistors, you could draw the two lines shown in Figure 6-17. Each resistance value is the inverse of the slope of the line (voltage divided by current). You can enter the four steady-state resistance values into the cell library description or specify them with the PrimeTime SI command `set_steady_state_resistance`.

Figure 6-17 Resistance Approximation of Steady-State I-V Characteristics



For even better accuracy, you can specify the I-V characteristics using a piecewise-linear model or a polynomial model. These more accurate models can be specified only in the library, not with PrimeTime SI commands.

In the absence of library-specified or command-specified I-V characteristics, PrimeTime SI uses an estimated linear resistance calculated from the output delay, output slew, and NMOS/PMOS transistor threshold voltages. The output delay and slew are specified in the library. PrimeTime SI assumes an NMOS and PMOS threshold voltage of 0.2 times the rail-to-rail voltage.

In case of conflict between different methods, PrimeTime SI uses steady-state I-V specifications in the following order:

- PrimeTime SI command-specified steady-state resistance (`set_steady_state_resistance` command)
- Library-specified per-arc I-V polynomials or tables
- Library-specified per-pin steady-state resistance
- PrimeTime SI estimation from output delay, output slew, and NMOS/PMOS transistor threshold voltages

Table 6-2 lists and briefly describes the methods that can be used to specify cell output steady-state I-V characteristics in the Synopsys .lib technology library. For more information on library types and the Library Compiler syntax, see the Library Compiler documentation.

Table 6-2 Library Specification Methods for Output I-V Characteristics

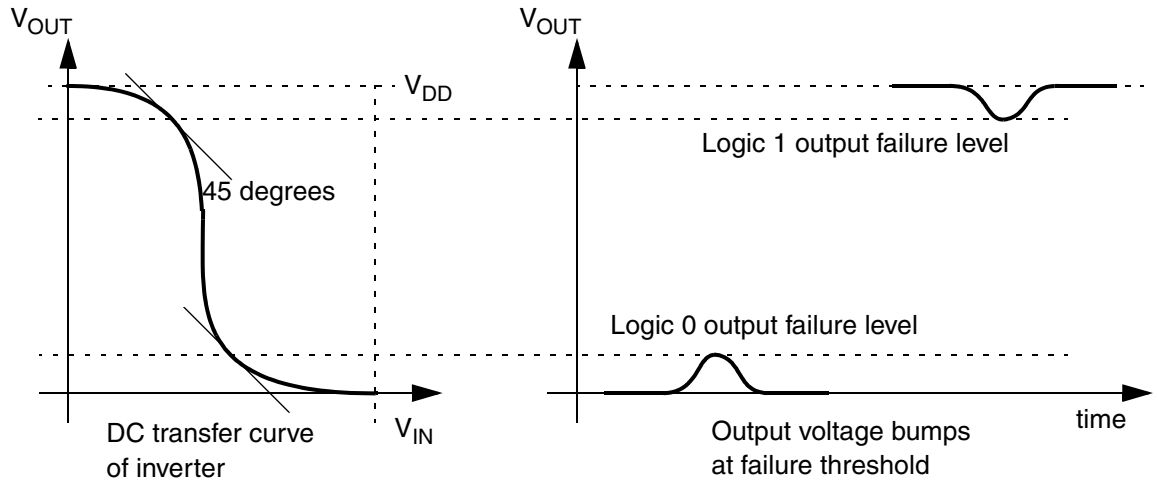
Specification method	Library type	How specified in library
Polynomials	SPDM	Two polynomials per timing arc for steady-state low and steady-state high. Polynomials describe current as a function of voltage.
Lookup tables	NLDM	Two lookup tables per timing arc for steady-state low and steady-state high. Tables describe current as a function of voltage.
Steady-state resistance	NLDM or SPDM	Four floating-point resistance values per timing arc for above low, below low, above high, and below high. Output is modeled as a resistor connected to ground for logic zero or connected to Vdd for logic one. Can also be specified per output with the `set_steady_state_ resistance` command.

Noise Immunity

Each cell input can tolerate a certain amount of noise without causing a failure at the cell output. This characteristic is called noise immunity. PrimeTime SI uses the library-specified or command-specified noise immunity at cell inputs to determine whether noise failures occur and the amount of noise slack at each cell input.

The recommended definition of “logic failure” is established by the points in the DC transfer curve at which the slope of the curve reaches 45 degrees. This definition is illustrated for the case of an inverter in Figure 6-18. However, the creator of the library might use some other failure criteria.

Figure 6-18 Noise Immunity Failure Definition



Noise immunity can be specified either in terms of allowable noise bump heights and widths at the cell inputs (specified as noise immunity curves, polynomials, or tables) or in terms of noise margins that consider only the bump heights at the cell inputs.

In case of conflict between different methods, PrimeTime SI uses noise immunity specifications in the following order:

- PrimeTime SI command-specified noise immunity curves (`set_noise_immunity_curve` command)
- PrimeTime SI command-specified bump height noise margins (`set_noise_margin` command)
- Library-specified per-arc noise immunity polynomials or tables
- Library-specified per-pin noise immunity curves
- Library-specified DC noise margins (V_{OL}, V_{OH}, V_{IL}, V_{IH})

Table 6-3 lists and briefly describes the methods that can be used to specify cell noise immunity characteristics in the Synopsys .lib technology library. For more information on library types and the Library Compiler syntax, see the Library Compiler documentation.

Table 6-3 Library Specification Methods for Noise Immunity

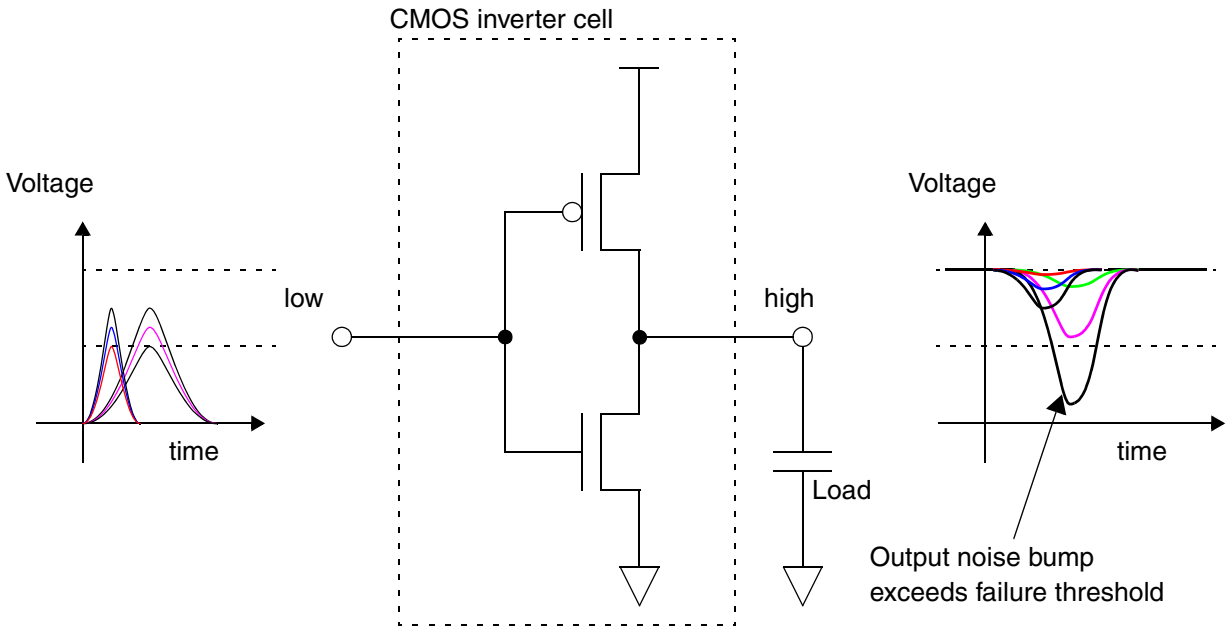
Specification method	Library type	How specified in library
Hyperbolic noise immunity curves (per input)	NLDM or SPDM	Four hyperbolas per input specify the maximum noise bump height as a function of noise bump width: one hyperbola each for input noise bumps above low, below low, above high, and below high. Each hyperbola is specified with three floating-point coefficients. Can also be specified with the `set_noise_immunity_curve` command.
Noise immunity polynomials (per timing arc)	SPDM	Four polynomials per timing arc specify maximum noise height as a function of noise width and output load: one polynomial each for input noise bumps above low, below low, above high, and below high.
Noise immunity lookup tables (per timing arc)	NLDM	Four lookup tables per timing arc specify maximum noise height as a function of noise width and output load: one table each for input noise bumps above low, below low, above high, and below high.
Noise margins based on bump height only (per input)	NLDM or SPDM	Four floating-point values specify the maximum and minimum allowable voltages for logic zero and logic one for each input pin. Can also be specified with the `set_noise_margin` command.

Noise Immunity Curves

When you use a noise immunity curve, PrimeTime SI considers the width and height of noise bumps occurring at cell inputs. Because of the capacitance at the cell input, a very brief noise bump can be tolerated, even if it is relatively high.

The noise immunity of a cell input can be measured in the laboratory by applying noise bumps of varying heights and widths to the input, as shown for the CMOS inverter in Figure 6-19. Bumps at the input that are small in terms of width and height will not cause any change at the output. However, bumps that are wide and high will cause the output to have a bump or even change logic state entirely.

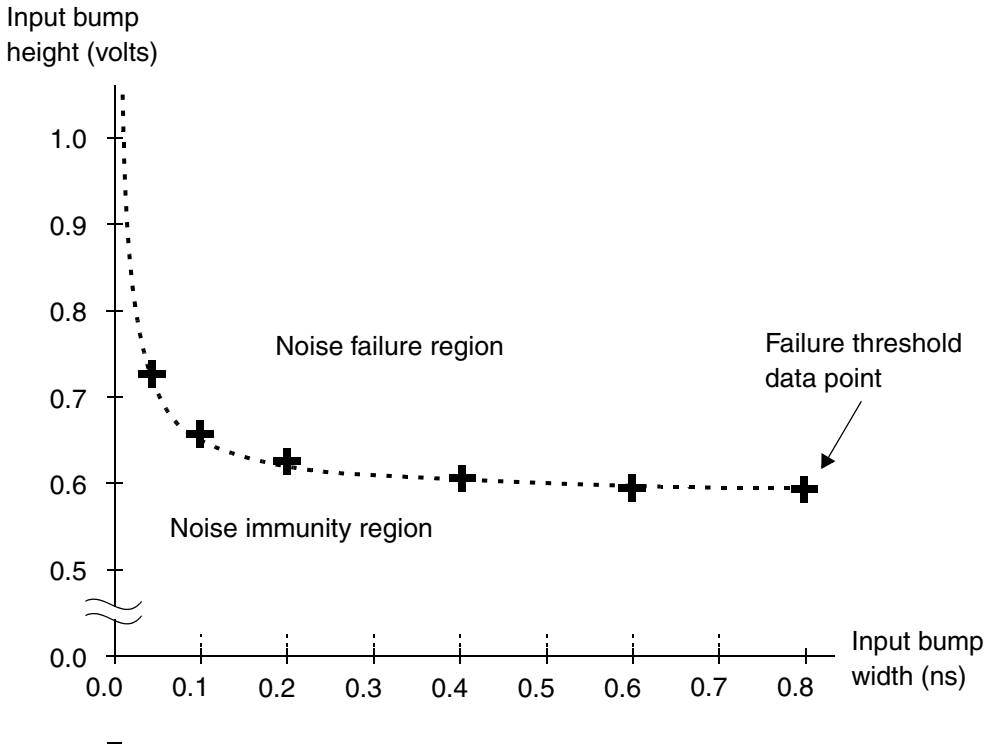
Figure 6-19 Noise Immunity Characterization of Input



If the cell has multiple outputs, there can be multiple timing arcs with different immunity characteristics for the same input. For each input, the values obtained from the worst-case input-to-output timing arc should be used in the library. Because noise immunity is sensitive to output load, characterization should be done with the worst-case (smallest) capacitive load.

Under the worst-case conditions, if you select several combinations of height and width at which logic failures just begin to appear and plot them on a graph, you will get a curve similar to the one shown in Figure 6-20. Given this information for each cell input and the size of the input noise bump, PrimeTime SI can determine whether a logic failure will occur at the cell output.

Figure 6-20 Input Bump Width Versus Height at Failure Thresholds



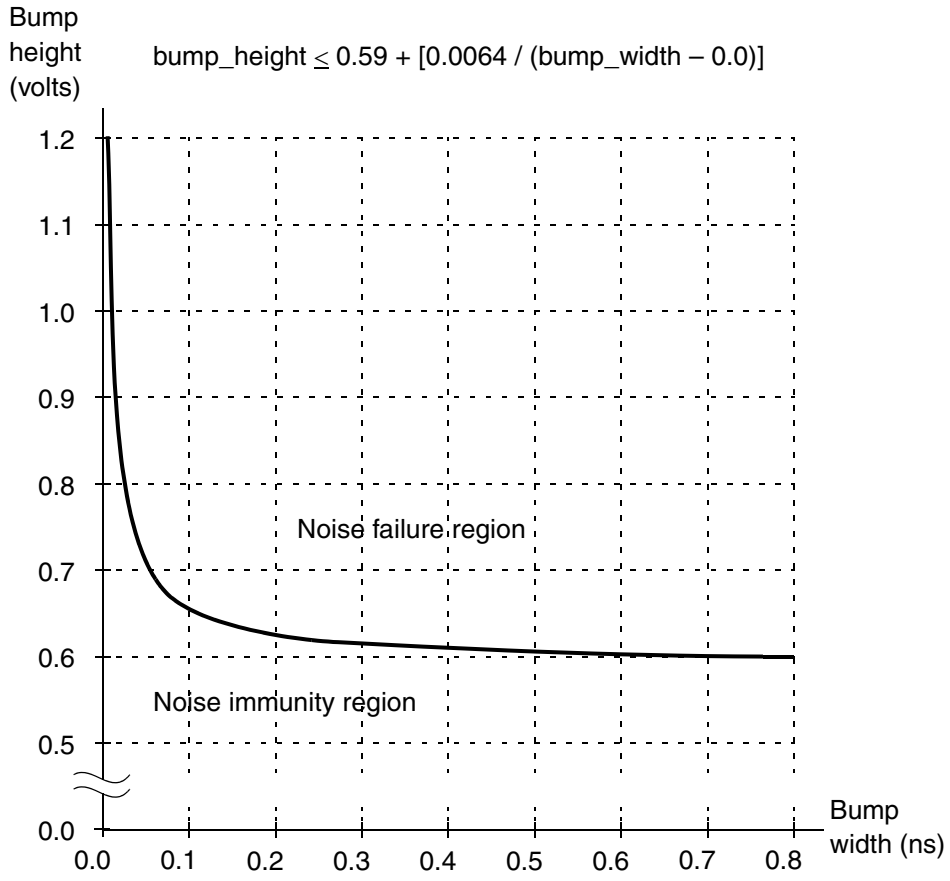
The library syntax allows a noise immunity curve to be specified as a hyperbolic curve, a piecewise-linear model, or a polynomial. For a hyperbolic curve, three coefficients fully specify the hyperbola:

$$y = c_1 + \frac{c_2}{(x - c_3)}$$

where *y* is the height and *x* is the width of the input voltage bump at the threshold of logic failure, *c1* is a voltage offset equal to the DC noise immunity value, *c2* is a size parameter for the hyperbolic curve, and *c3* is a time-value offset. The three coefficients should be chosen to match the hyperbolic curve to the data points obtained by laboratory characterization.

For example, Figure 6-21 shows a plot of a noise immunity curve with *c1* = 0.59, *c2* = 0.0064, and *c3* = 0.0. Combinations of bump height and width below the curve are in the region of noise immunity, while those above the curve are in the region of noise failure.

Figure 6-21 Hyperbolic Noise Immunity Curve



In Library Compiler syntax, the coefficients *c1*, *c2*, and *c3* are called height_coefficient, area_coefficient, and width_coefficient.

Noise immunity characteristics can vary for different noise bump types, so there can be four different noise immunity curves associated with each input: below low, above low, below high, and above high. All coefficients are specified as positive numbers for all four types of noise bumps.

In the absence of library-specified noise immunity characteristics, or to override the library-specified characteristics, you can use the PrimeTime SI command `set_noise_immunity_curve`, which lets you set the three hyperbolic coefficients for specified ports or cell input pins. The coefficients *c1*, *c2*, and *c3* are set with the options `-height`, `-area`, and `-width`.

You can display noise immunity curves in the PrimeTime GUI. For details, see “Noise Immunity Curves” on page 3-16.

Noise Immunity Polynomials and Tables

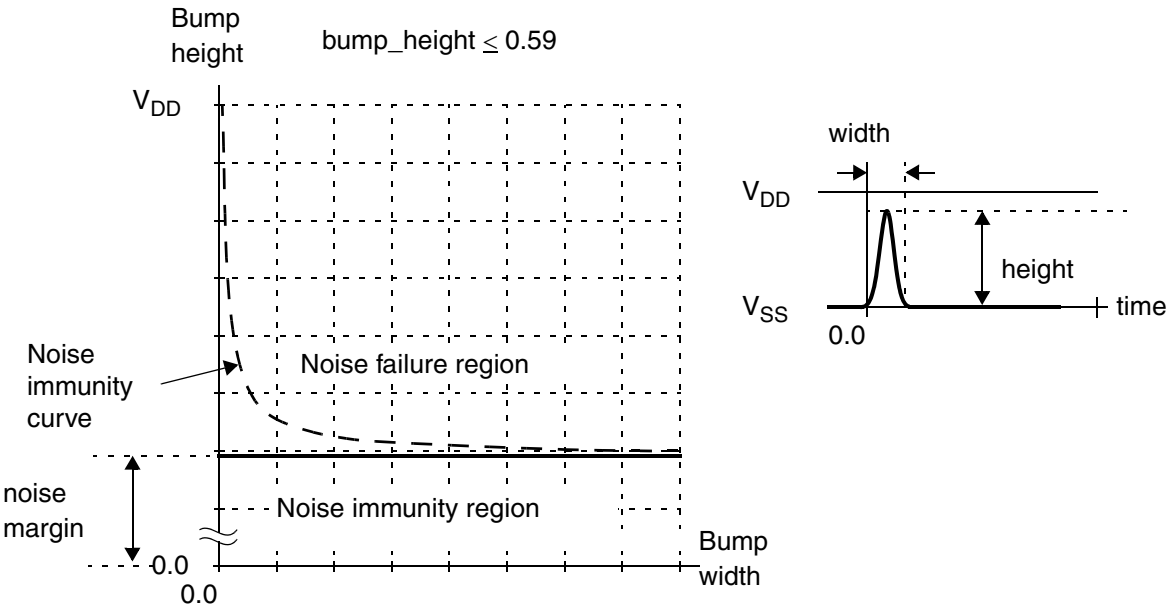
In cases where the noise immunity characteristics vary significantly due to different output loads or different paths through the cell, the library can use polynomials or lookup tables instead of hyperbolic noise immunity curves. Using polynomials or lookup tables, the library specifies the maximum input bump height as a function of input bump width and output capacitive load. Each noise immunity specification applies to an individual input-to-output timing arc rather than all arcs through a given input.

Per-arc specification allows for state-dependent variation in noise immunity. For example, for an XOR gate with inputs A and B and output Z, the A-to-Z and B-to-Z arcs might have different noise immunity due to the different paths taken through the transistors in the cell.

Bump Height Noise Margins

Instead of using noise immunity specifications that consider input bump width, input bump height, and output load, you can use noise margins that consider only the input bump height. Using height-only noise margins is simpler, faster, and more conservative than the other methods. Figure 6-22 compares noise immunity curves and bump height noise margins.

Figure 6-22 Height-only Noise Margin Versus Noise Immunity Curve



For high, narrow noise bumps, using height-only noise margins is pessimistic because it treats some bumps as noise failures that would otherwise pass with the immunity curve model. However, for wide noise bumps, using noise margins gives the same results as using noise immunity curves.

There are four different noise margin values associated with each input: below low, above low, below high, and above high. These values are specified as positive numbers for all four types of noise bumps.

In the absence of library-specified noise immunity specifications, or to override the library-specified specifications, you can use the PrimeTime SI command `set_noise_margin`, which lets you set the height-only noise margins for specified ports or cell input pins.

In the absence of command-specified or library-specified noise immunity data, PrimeTime SI calculates the maximum allowable noise bump heights based on DC noise margins of the driver and receiver, as defined in the .lib technology library by the input/output logic-level parameters V_{IL}, V_{IH}, V_{OL}, and V_{OH}.

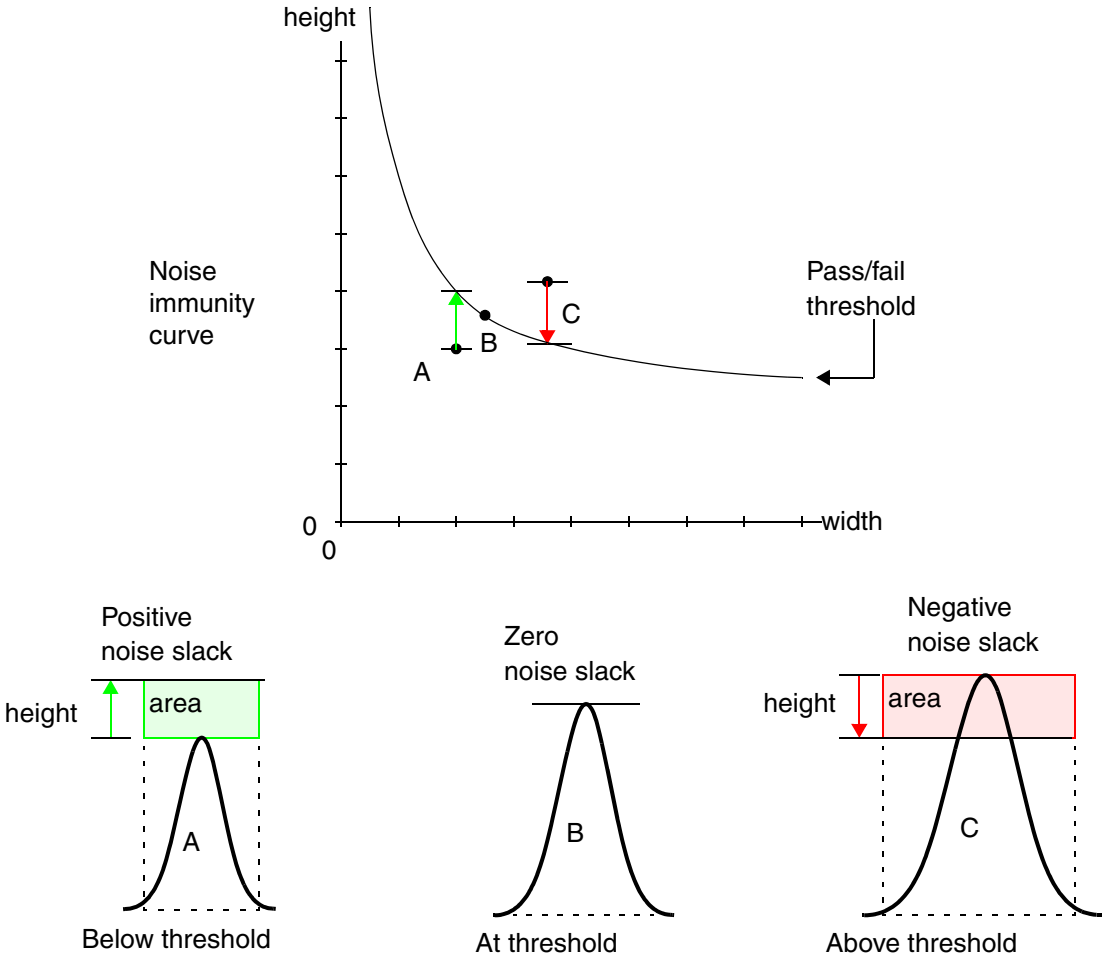
Noise Slack

In static timing analysis, “slack” is the amount of time by which a timing constraint is met. It is the difference between the required time and the arrival time of a signal transition. It is in library units of time, such as nanoseconds. Negative slack indicates a timing failure.

In static noise analysis, there is a choice of noise slack reporting methods, called the “area,” “height,” and “area_percent” methods. The default method is “area.” In that case, noise slack is determined by the amount of *voltage and time* by which a noise constraint is met.

The calculation of noise slack is illustrated in Figure 6-23. The figure shows three noise bumps: one below the threshold of noise failure, one just at the threshold of failure, and one above the threshold of failure. The width and height of each noise bump is plotted as a black dot on the noise immunity curve for the cell input.

Figure 6-23 Noise Slack Calculation

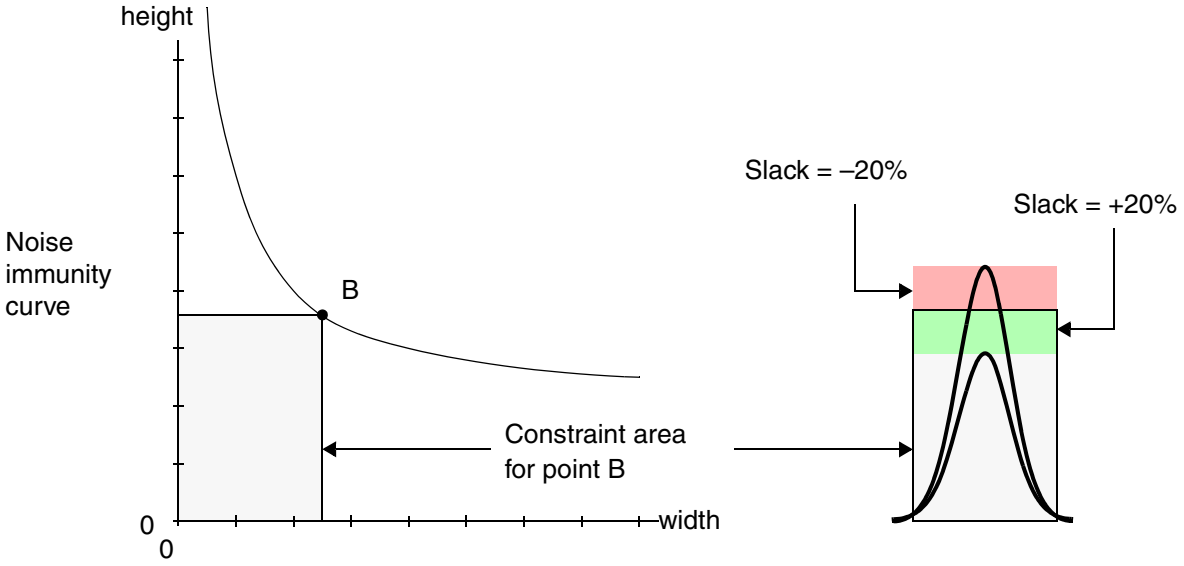


The “area” type noise slack is the voltage margin (height of the curve above the data point) multiplied by the noise bump width, as indicated by the shaded rectangles in the figure. For a noise bump below the failure threshold, the slack is positive, or for a noise bump above the failure threshold, the slack is negative. The units for “area” noise slack are library units of voltage multiplied by library units of time, such as millivolt-nanoseconds.

Using the “height” method, the noise slack is defined as the voltage margin alone, in library voltage units.

Using the “area_percent” method, the noise slack is defined as the area slack divided by the total “constraint area.” The constraint area is the bump width multiplied by the allowed height, equal to the area of the rectangle bounded by the data point in the noise immunity curve, as shown in Figure 6-24.

Figure 6-24 Area Percent Slack Calculation

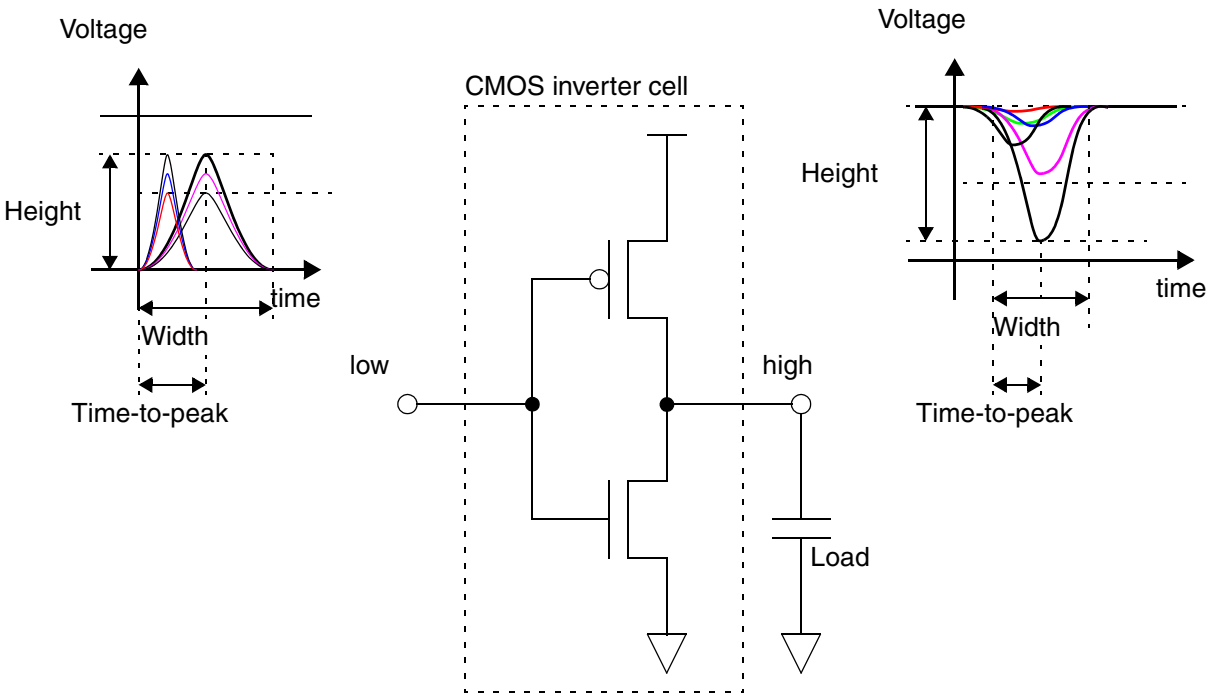


Propagated Noise Characteristics

A noise bump at a cell input, if large enough in terms of height and width, will cause a noise bump at the cell output. This effect is called noise propagation.

The noise propagation for an input-to-output timing arc can be measured in the laboratory by applying noise bumps of varying heights and widths to the input and measuring the resulting noise bumps at the output, as shown for the CMOS inverter in Figure 6-25. The output noise bump characteristics depend on the width and height of the input bump and the capacitive load on the output, and to a lesser extent, the time-peak-ratio of the input bump.

Figure 6-25 Noise Propagation Characterization



After the noise propagation effects have been characterized, the information can be entered into the library. The library syntax supports two ways to specify propagation effects: polynomials and lookup tables.

With polynomials, the library specifies the height, width, and time-peak-ratio of the output bump as a function of the input bump height, input bump width, input bump time-peak-ratio, and output load. There are 12 such polynomials for each timing arc because there are three functions (height, width, and time-peak-ratio) for each of four output bump types: below low, above low, below high, and above high.

The time-peak-ratio is the time-to-peak value divided by the width of the noise bump. The time-peak-ratio can be floating-point value between 0.0 and 1.0. For a symmetrical noise bump, the time-peak-ratio is 0.5. Including time-peak-ratio characteristics makes a more accurate noise propagation model, but requires more work for characterization.

With lookup tables, the library specifies the height and width of the output bump as a function of the input bump height, input bump width, and output load. There are eight such lookup tables for each timing arc because there are two functions (height and width) for each of four output bump types: below low, above low, below high, and above high. The lookup tables do not include time-peak-ratio information, but PrimeTime SI still calculates the time-peak-ratio characteristics of propagated noise bumps based on the cell delay and slew information in the library.

Table 6-4 lists and briefly describes the methods that can be used to specify propagated noise characteristics in the Synopsys .lib technology library. For more information on library types and the Library Compiler syntax, see the Library Compiler documentation.

Table 6-4 Library Specification Methods for Propagated Noise

Specification method	Library type	How specified in library
Polynomials	SPDM	Four sets of three polynomials (12 polynomials) per timing arc that specify the output bump height, output bump width, and output peak-time-ratio as a function of input bump height, input bump width, input peak-time-ratio, and output load; one set of three polynomials each for output noise bumps above low, below low, above high, and below high.
Lookup tables	NLDM	Four pairs of lookup tables (8 tables) per timing arc that specify the output bump height and output bump width as a function of input bump height, input bump width, and output load; one pair of tables each for output noise bumps above low, below low, above high, and below high.

A

Crosstalk Attributes

You can use the `get_attribute` command to obtain data from the design, including crosstalk data. The attributes are listed and described in the following sections of this appendix:

Crosstalk Attributes by Class

The term “effective,” when used to describe an aggressor net, means an object selected for analysis and not removed by filtering.

PrimeTime SI attributes are read-only unless otherwise specified.

Find the attribute descriptions by class in the following tables:

- `net` object type, Table A-1 on page A-3
- `lib_timing_arc` object class, Table A-2 on page A-6
- `timing_arc` object class, Table A-3 on page A-7
- `timing_point` object class, Table A-4 on page A-8
- `pin` object class, Table A-5 on page A-9
- `lib_pin` object class, Table A-6 on page A-12
- `port` object class, Table A-7 on page A-12

Net Attributes

Table A-1 Attributes of the net Object Type

Attribute	Type	Description
aggressors	string	This attribute shows the aggressor nets that impact the victim net.

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}`

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.

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`

Table A-1 Attributes of the net Object Type (Continued)

Attribute	Type	Description
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.
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.
si_xtalk_composite_aggr_ min_rise si_xtalk_composite_aggr_ min_fall si_xtalk_composite_aggr_ max_rise si_xtalk_composite_aggr_ max_fall	string	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_coupling_ capacitance	float	This attribute lists the total cross-coupling capacitance a victim net has in pF.

Table A-1 Attributes of the net Object Type (Continued)

Attribute	Type	Description
`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.
`user_pairwise_coupling_ separated`	`string`	The collection of nets which have been pairwise-separated with the `set_coupling_separation` command.

Lib Timing Arc Attributes

Table A-2 Attributes of the lib_timing_arc Object Type Class

Attribute	Type	Description
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.
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.

Timing Arc Attributes

Table A-3 Attributes of the timing_arc Object Type Class

Attribute	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.
has_noise_immunity_ above_low has_noise_immunity_ below_high	boolean	This attribute indicates whether the library timing arc has a noise immunity curve for above-low or below-high noise bumps.
si_has_immunity_above_low si_has_immunity_below_high si_has_immunity_below_low si_has_immunity_above_high	boolean	This attribute indicates whether the noise immunity information is present in the timing arc object in a library.

Timing Point Attributes

Table A-4 Attributes of the timing_point Object Type Class

Attribute	Type	Description
annotated_delay_delta	float	When using PrimeTime SI, calculates the effect of crosstalk.
annotated_delta_ transition	float	When using PrimeTime SI, calculates the effect of crosstalk
si_xtalk_bumps	float	Lists each aggressor net and the voltage bumps that rising and falling aggressor transitions induce on the victim net (worst of rising and falling min or max bumps, each expressed as a decimal fraction of the rail-to-rail voltage), or gives a reason why the aggressor net has no effect on the victim net.

Pin Attributes

Table A-5 Attributes of the pin Object Type Class

Attribute	Type	Description
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.
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.
si_noise_active_aggressors_ above_high si_noise_active_aggressors_ below_high si_noise_active_aggressors_ above_low si_noise_active_aggressors_ below_low	collec-tion	This attribute returns a collection of active aggressor nets for the input pin, 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.

Table A-5 Attributes of the pin Object Type Class (Continued)

Attribute	Type	Description

`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.

`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 offset 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.

Table A-5 Attributes of the pin Object Type Class (Continued)

Attribute	Type	Description
si_noise_total_bump_above_high si_noise_total_bump_below_high si_noise_total_bump_above_low 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.
si_has_immunity_above_low si_has_immunity_below_high si_has_immunity_below_low si_has_immunity_above_high	boolean	This attribute indicates whether noise immunity information is present for a pin.

Lib Pin Attributes

Table A-6 Attributes of the lib_pin Object Type Class

Attribute	Type	Description
driver_waveform_rise driver_waveform_fall	string	This attribute indicates the type of driver waveform for the library pin, either ramp or standard.
has_ccs_noise_above_low has_ccs_noise_below_high has_ccs_noise_below_low has_ccs_noise_above_high	boolean	This attribute indicates whether CCS noise information is present for a pin in a library.
si_has_immunity_above_low si_has_immunity_below_high si_has_immunity_below_low si_has_immunity_above_high	boolean	This attribute indicates whether NLDM noise immunity information is present for a pin in a library.

Port Attributes

Table A-7 Attributes of the port Object Type Class

Attribute	Type	Description
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.
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.

Table A-7 Attributes of the port Object Type Class (Continued)

Attribute	Type	Description
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.
si_noise_active_aggressors_ above_high si_noise_active_aggressors_ below_high si_noise_active_aggressors_ above_low si_noise_active_aggressors_ below_low	collec -tion	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_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 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.

Table A-7 Attributes of the port Object Type Class (Continued)

Attribute	Type	Description
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 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.
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 port in the above-high, below-high, above-low, or below-low region.
si_noise_total_bump_above_high si_noise_total_bump_below_high si_noise_total_bump_above_low 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.

B

SPICE Analysis

PrimeTime SI lets you generate a SPICE deck from the design database so that you can verify the crosstalk analysis results with a SPICE simulator. Generating and using the SPICE deck are described in the following sections:

SPICE Deck Usage

PrimeTime SI has the ability to generate a SPICE deck template from the design database for a particular path of interest, including the capacitive cross-coupling structure. The purpose of this capability is to help you simulate the path of interest and examine a particular circuit topology using a SPICE simulator.

The command to generate a SPICE deck is `write_spice_deck`. Using the command requires a PrimeTime SI license. The command generates a SPICE netlist that includes the cells of a specified path or arc, together with the resistors, ground capacitors, and coupling capacitors to aggressor nets related to the nets in the path or arc. It also generates the stimuli to sensitize the victim path and aggressors according to the options specified in the `write_spice_deck` command.

The SPICE deck generator is useful for creating a structural representation of the victim path and its aggressor nets. However, you cannot expect to use a single SPICE deck simulation run and do a direct, one-to-one comparison with the PrimeTime SI results. There are several reasons for this:

- The path that you select might be so long, with so many cross-coupling capacitors, that even a single SPICE simulation run might take too long to be practical.
- PrimeTime SI does the calculation using multiple iterations, taking into consideration the whole environment of the victim path, using multiple switching cycles. The SPICE deck generator can only generate a single victim path (with its aggressor nets) and can only consider one switching cycle of the path.
- Other factors such as the slew propagation scheme and accuracy of the library can affect comparison between the PrimeTime SI analysis and SPICE simulation results.

Requirements

The following requirements apply to generating and using the SPICE deck from PrimeTime SI:

- The SPICE subcircuit definitions must be available for all gates used in the SPICE deck. You can use the include statement.
- The SPICE models for all transistors and other devices used in the gates must be available.
- You must define all power and ground nets in the SPICE deck output. All the generated ground capacitors are connected to net 0. All generated piecewise linear (PWL) delay models are relative to net 0. The header in the generated SPICE deck contains comment lines that demonstrate how to define Vdd as power and Vss as ground.

- Check all SPICE comment lines in the SPICE deck output for notes, warnings, and error messages. Some typical errors are: missing subcircuit file, missing definition in the subcircuit file, incompatible pin order, or unconstrained path that prevents PrimeTime SI from determining the arrival window.
- PrimeTime SI produces all the required stimuli for the file. However, it might not be able to do so in certain cases, such as for a RAM lacking a definition of the appropriate data.

Generating the SPICE Deck

To generate a SPICE deck for a path in the design, use the `write_spice_deck` command. In the command, you specify the output file name and the paths (a collection of paths or arcs) in the design for which you are generating the SPICE deck.

This is the `write_spice_deck` command syntax:

```
write_spice_deck
  [-align_aggressors]
  [-analysis_type type]
  [-header header_file_name]
  [-initial_delay delay]
  [-logic_one_name v1name]
  [-logic_one_voltage v1]
  [-logic_zero_name v0name]
  [-logic_zero_voltage v0]
  [-minimum_transition_time trans]
  [-output file_name]
  [-sub_circuit_file spice_subckt]
  [-sweep_size num]
  [-sweep_step num]
  [-time_precision precision]
  [-transient_size tran_size]
  [-transient_step tran_step]
  [-use_probe]
  [-user_measures user_measure_list]
  [-sample_size number_of_samples]
  paths_arcs_list
```

The command must specify the paths or arcs for which to generate the SPICE deck. The paths are specified in collection form and obtained by the `get_timing_paths` or `get_timing_arcs` command. Use `get_timing_paths` to get a collection of one or more full paths. Each path starts from an input port or clock pin and ends at an output port or data pin. Use `get_timing_arcs` to get a collection of one or more timing arcs (such as one stage of a timing path). Each arc starts at one pin and ends on another.

The write_spice_deck command options specify the subcircuit file used, the scope of clock circuit generation, and other details. For a timing arc, the -analysis_type option specifies the type of noise analysis for which to generate the circuit stimulus waveforms.

The -output option specifies the file name for the SPICE deck written for the first timing path, and the base name used for generating other output file names.

The -user_measures option lets you specify the .measure statements generated in the SPICE deck, for example, to measure the delay, slew, noise peak values for specified arcs, ports, or pins.

For detailed information about the many options of the write_spice_deck command, refer to its man page.

Writing a SPICE Deck for a Path

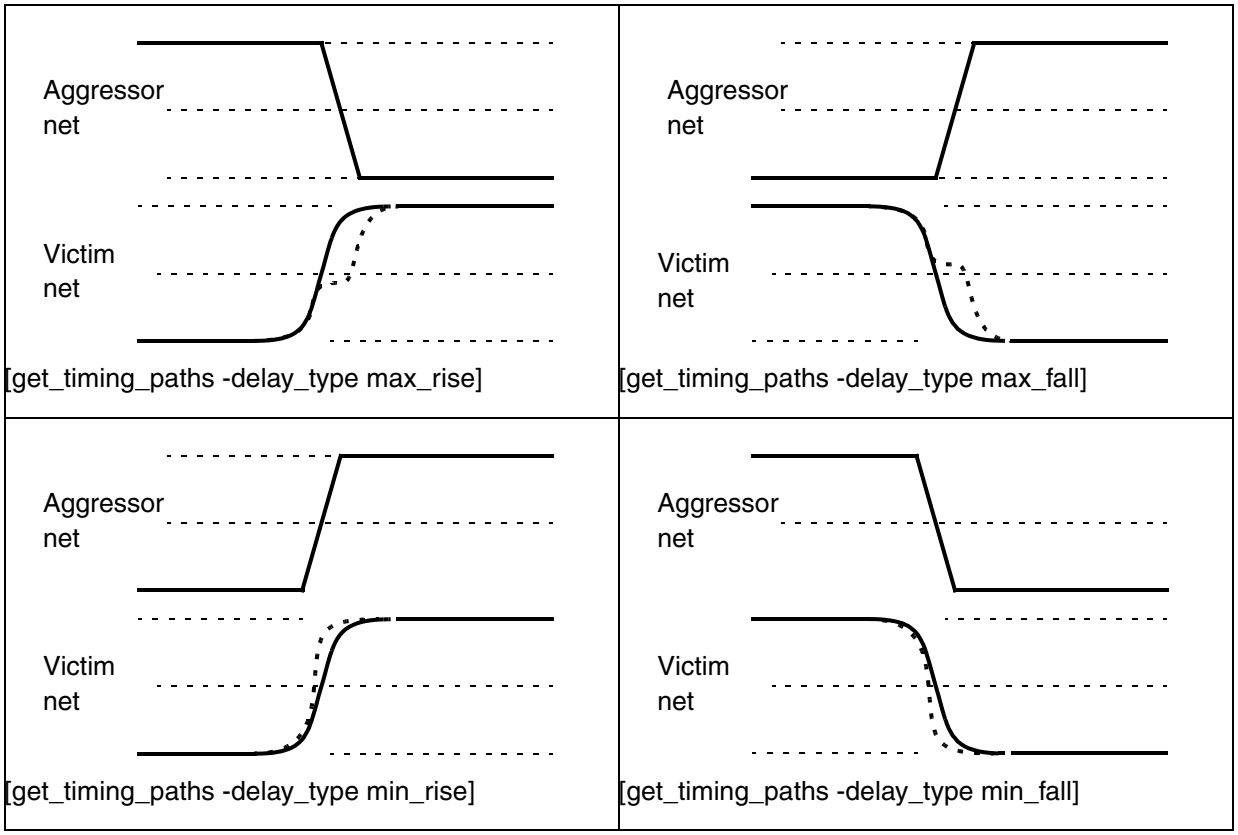
To write a SPICE deck for a full path, use the get_timing_paths command to create a collection containing the path of interest. You can use the generated SPICE deck to analyze crosstalk delay effects (but not static noise effects) for the path. Here is an example:

```
pt_shell> write_spice_deck -header header.spi \
            -output testcase.spi \
            -logic_one_voltage 1.5 \
            -logic_zero_voltage 0.0 \
            -sub_circuit_file ./SPICE/subckt.spi \
            [get_timing_paths -from A2 -to buf5/A]
```

The get_timing_paths command specifies the path for which to generate a SPICE deck. You can specify particular paths or transitions by using options such as -from, -to, or -delay_type in the get_timing_paths command. When used without options, it gets the single path with the worst timing slack, so that the write_spice_deck command generates the circuit and stimulus associated with that path and timing conditions.

To generate the circuit stimulus waveforms for a particular set of victim and aggressor transitions, use the -delay_type option of the get_timing_paths command. The option, when set to max_rise, max_fall, min_rise, or min_fall, specifies the type of delay (maximum or minimum) and the type of transition considered at the path endpoint (rising or falling). For crosstalk occurring at the path endpoint, this option setting specifies the crosstalk conditions illustrated in Figure B-1. For each victim net along the path, the write_spice_deck command sensitizes the aggressor nets appropriately in order to test the specified maximum or minimum delay transition.

Figure B-1 How to Specify Crosstalk Delay Transitions for Paths



The generated SPICE deck includes all cross-coupled aggressor nets and capacitors along the full path. A long or complex path with a lot of coupling capacitors can result in a very large data file, too large to be practical for SPICE simulation. In that case, try using `get_timing_arcs` to select just a portion of the path that has the victim net and the immediate surrounding circuitry.

Writing a SPICE Deck for an Arc

To write a SPICE deck for an arc from one point in the design to another, use the `get_timing_arcs` command . You can use the generated SPICE deck to analyze crosstalk delay effects or static noise effects for the arc.

Here is an example:

```
pt_shell> write_spice_deck -header header.spi \
        -analysis_type above_high \
        -output ../SIMUL_BEYOND_HIGH/new_general.spi \
        -logic_one_voltage 1.5 -logic_zero_voltage 0.0 \
```

```
-sub_circuit_file ./SPICE/subckt.spi \
[get_timing_arc -to buf5/A]
```

The `get_timing_arcs` command specifies the arc for which to generate the SPICE deck. Use a combination of the options `-from`, `-to`, and `-of_objects` in the `get_timing_arcs` command to specify the portion of the design to be analyzed.

The `-analysis_type` option of the `write_spice_deck` command specifies the type of noise analysis for which to generate the SPICE circuit stimulus. For crosstalk delay analysis, use `max_rise`, `max_fall`, `min_rise`, or `min_fall`, as indicated in Figure B-2. For static noise analysis, use `above_high`, `below_high`, `above_low`, or `below_low`, as indicated in Figure B-3.

Figure B-2 How to Specify Crosstalk Delay Transitions for Arcs

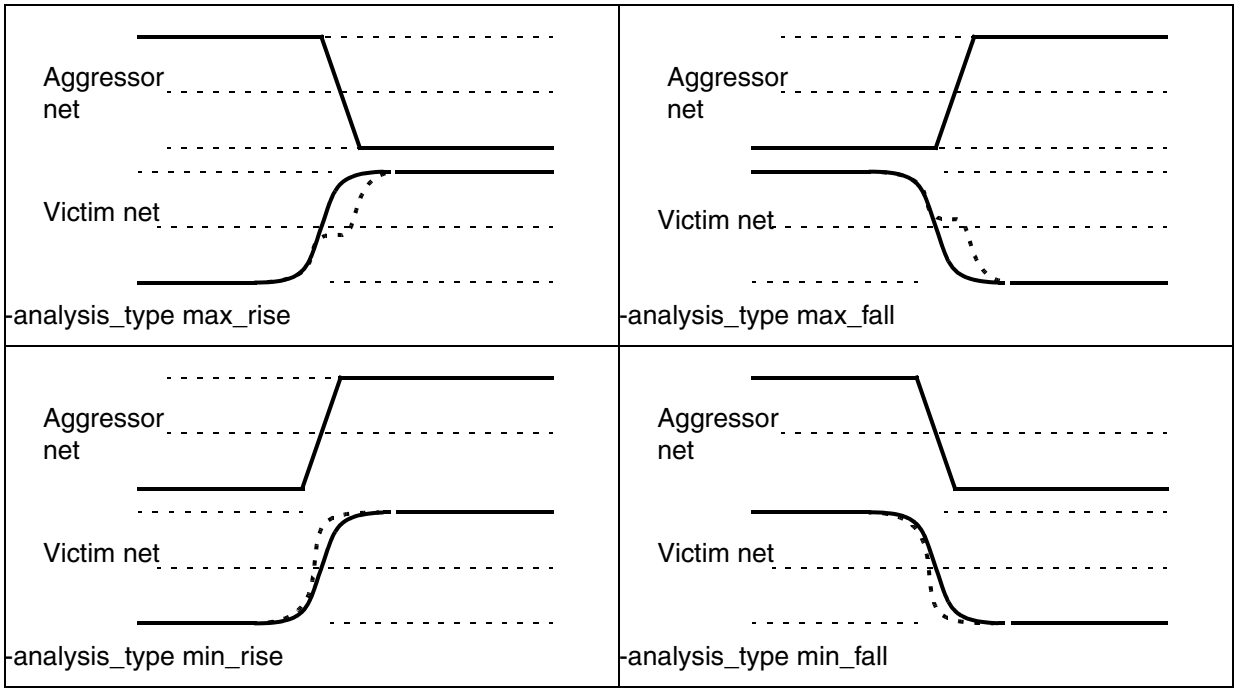
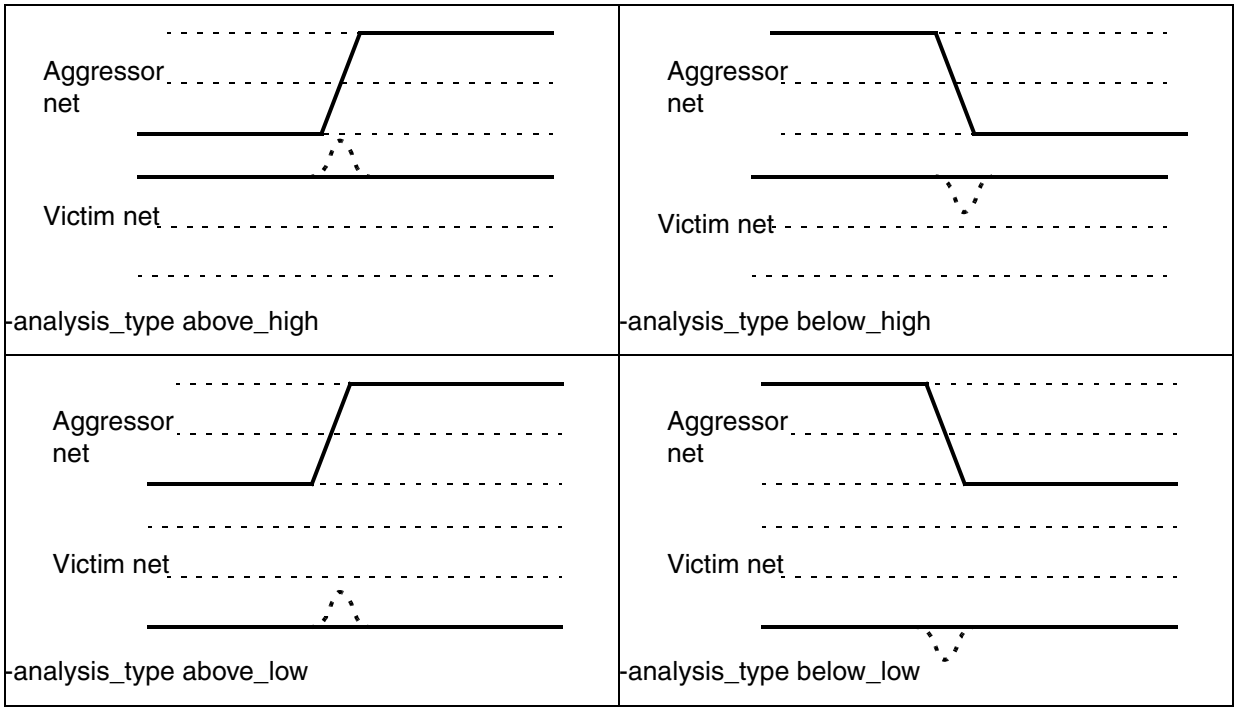


Figure B-3 How to Specify Crosstalk Noise Transitions for Arcs



By default, `write_spice_deck` places an aggressor transition in the middle of the arrival timing window, not necessarily at the time with worst-case crosstalk effects. In order to get agreement between PrimeTime SI and SPICE, it is necessary to “sweep” the transition time using a sequence of SPICE simulations, to find the worst-case transition time.

To reduce the simulation effort, use the `-align_aggressors` option of `write_spice_deck`. This places the aggressor transitions where they produce the worst-case crosstalk effects for the conditions specified by the `-analysis_type` option. The specified condition can be `max_rise`, `max_fall`, `min_rise`, or `min_fall` for crosstalk delay analysis or `above_high`, `below_high`, `above_low`, or `below_low` for crosstalk noise analysis. The aggressor alignment feature is available for timing stages obtained by the `get_timing_arcs` command, but not for timing paths obtained by the `get_timing_paths` command.

Aggressor alignment is done only for a net timing arc (an arc from the output pin of a cell, through a net, to the input pin of another cell), not for a cell timing arc (an arc from an input pin to an output pin of a cell), even though the coupled RC network of the driven net is written.

When `write_spice_deck` uses aggressor alignment for a net arc, it chooses the same driving cell arc that was used during `update_timing` in PrimeTime. Aggressor alignment does not work for an aggressor that is directly driven with the `set_driving_ cell` command.

User-Defined Sensitization in write_spice_deck

You can sensitize a cell in the SPICE deck with its specific sensitization sequence by accepting user sensitization data. This helps to correctly sensitize complex sequential cells such as a JK, toggle, or enable flip-flop when you write a SPICE deck. You can use this feature to sensitize a specific arc of a combinational cell or a complex sequential cell.

You can apply complete sensitization only to the following cell types:

- The launching sequential cell of a timing path
- The driver cell of a timing arc
- The aggressor driver cells that are not a part of the victim circuitry

The following sections provide command details, syntax, and examples for setting, reporting, and removing user-defined sensitization.

Setting User Sensitization

Using the `set_user_sensitization` command, you can sensitize the cells used by the `write_spice_deck` command to create the SPICE deck. The `set_user_sensitization` command has the following syntax:

```
set_user_sensitization
  -analysis_type [ rise | fall | high | low ]
  [-initial_condition {node_name voltage}]
  [-tie_high logic_one_input_pin_list]
  [-tie_low logic_zero_input_pin_list]
  [-use_pwl_for_clock]
  [-event_step separation_time]
  -event_pins input_pin__name_list
  -event_states list_of_r_f_1_and_0s
  arc_list
```

The `set_user_sensitization` command accepts either a library timing arc or a timing arc collection to sensitize an arc of a given cell type. When you set the sensitization, it overrides the logic values set by the command `get_timing_path -justify` to constrain cells on the timing path.

The `-initial_condition` option causes `write_spice_deck` to write the initial transient condition for the cell instances of the specified library timing arc. For example, top/block1/ff1 is a flip-flop in the design and its corresponding timing arc is sensitized with the option `-initial_condition {.n1 0.5}`. The `write_spice_deck` command will print the following .IC statement in the SPICE deck for a path that uses top/block1/ff1 as the launching cell.

```
.IC V(top/block1/ff1.n1) = 0.5
```

If you sensitize a timing arc, you need to specify the state of all input pins using the `-event_states`, `-tie_high`, or `-tie_low` option. The timing arc or library timing arc in the arc collection must be compatible with the condition of the pins in other options of this command. For example, the final states of all input pins must satisfy the “when” condition of the arc. Typically the library doesn't have enough information on arcs for the `set_user_sensitization` command to eliminate most user errors.

You must specify at least one transition using `-event_states` for the analysis type set to rise or fall. In addition to attaching the voltage sources to all the input pins, you have the option of specifying the initial transient voltage conditions on some internal nodes and the output pins of the sensitized cell to initialize it to the correct state.

In the following example, `set_user_sensitization` sets the sensitization sequence for an enable pin of a buffer that can be placed in the high-impedance state. The low state of the EN pin places the buffer in the high-impedance state. At time 0, the cells open and pin A is in low state. Therefore, the output pin Y should be near 0 after the cell is placed in the high-impedance state. After pin A changes to high and the pin EN rises, pin Y should make a rising transition.

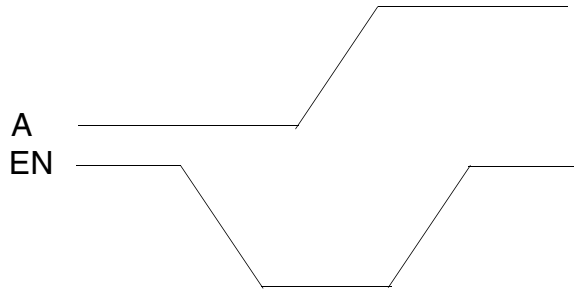
```
...
set timing_enable_preset_clear_arcs true
...
set_user_sensitization
  [get_lib_timing_arcs -from enbuf/EN -to enbuf/Y]
  -analysis_type rise \
  -event_setp 1 \
  -event_pins {A EN} \
  -event_states {0 f \ # Y in initialize to low
                 r 0 \ # Y is tristated while A is rising
                 1 r} \# Y should rise
```

Based on this, `write_spice_deck` prints the following:

```
VA  A  0  pwl 0 0.   6e-10 0.0 7e-10 1.65
VEN EN 0  pwl 0 1.65 5e-10 0.0 8e-10 0.0  9e-10 1.65
```

This example produces the following waveform:

Figure B-4 Sensitization Waveforms



Reporting User Sensitization

The `report_user_sensitization` command reports the sensitization sequence for

- A library timing arc collection
- An instance timing arc collection
- All the library timing arcs that have a user sensitization sequence annotated on them in a library
- All the instance timing arcs that have a user sensitization sequence annotated on them in a design

The `report_user_sensitization` command has the following syntax:

```
report_user_sensitization
  [-analysis_type [ rise | fall | high | low ]]
  [-arc arcs_list]
  [-library library_name]
  [-design design_name]
```

The following example reports user sensitization for the rising timing arc.

```
pt_shell> report_user_sensitization \
    [-get_lib_timing_arc -from jkffx1/CP -to jkffx1/Q] \
     -analysis_type rise

****************************************
Report : report_user_sensitization
-analysis_type_rise
Design : test
****************************************
from_lib_pin to_lib_pin sense analysis_type
EN -> Y enable_high rise
Event step : 1
Event library pins: A EN
1 0 f
2 r 0
3 1 r
```

Removing User Sensitization

Use the `remove_user_sensitization` command to remove sensitizations set using `set_user_sensitization` for a library, design, instance arc collection, or a library arc collection. The syntax for the command is

```
remove_user_sensitization
  [-analysis_type [ rise | fall | high | low ]]
  [-arc arcs_list]
  [-library library_name]
  [-design design_name]
```

Use the -analysis_type option to specify the final state of the timing or library arc's output pin. To remove a list timing or library arcs with their annotated sensitization, use the -arc option to this command. (The get_timing_arcs or get_lib_timing_arcs command generates a list of arguments.)

The following example removes user sensitization information on the rise state for arc arcs_1.

```
pt_shell> remove_user_sensitization -analysis_type rise \
          -arc $arc_1
User sensitization of timing arc inp_victim/S -> inp_victim/X
Information: 1 user sensitization removed. (SPICE-118)
```

Limitations

The following limitations apply to this release of set_user_sensitization:

- Noise analysis is not supported.
- The combination of set_driving_cell and set_user_sensitization is partially supported. The -multiply_by and -no_design_rule options are ignored by set_user_sensitization.
- Event synchronization is only performed for the launching cell of the victim path and its aggressors as long as the aggressors are not part of the victim path.

Library Sensitization in write_spice_deck

The write_spice_deck command can use of the stimulus information available in the library for sensitizing the logic cells of a timing path or a stage, thereby sensitizing the timing arc with the same stimulus used when the arc was characterized. This results in more accurate PrimeTime-to-SPICE correlation for both coupled and uncoupled analysis.

The write_spice_deck command first uses any sensitization specified explicitly in PrimeTime SI with the set_user_sensitization command, as described in the foregoing section. If there is no user-specified sensitization, it uses the sensitization information contained in the library. In the absence of both user-specified and library-specified sensitization, it applies default sensitization based on the "when" conditions of the timing arc and the logic functions for the combinational logic or the binary state table for sequential logic.

For information about the cell sensitization syntax in Liberty format, see the section called "Sensitization Support" in the "Timing Arcs" chapter of the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

The following example demonstrates how `write_spice_deck` uses library-defined sensitization information. In this example, the information is specified in the library as follows:

```
sensitization (2in_1out){
pin_names (IN1, IN2, OUT);
vector (0, "0 0 0") ;
vector (1, "0 0 1") ;
vector (2, "0 1 0") ;
vector (3, "0 1 1") ;
vector (4, "1 0 0") ;
vector (5, "1 0 1") ;
vector (6, "1 1 0") ;
vector (7, "1 1 1") ;


 cell(my_cell){
     sensitization_master : 2in_1out;
     pin_name_map (A, B, Z);
       ...
  pin(Z) {
       ...
     timing() {
          related_pin : A;
          wave_rise (0, 4, 2, 6, 3);
          wave_fall (1, 5, 3, 6);
          wave_rise_sampling_index : 4;
          wave_fall_sampling_index : 2;
     }
   }
 }
```

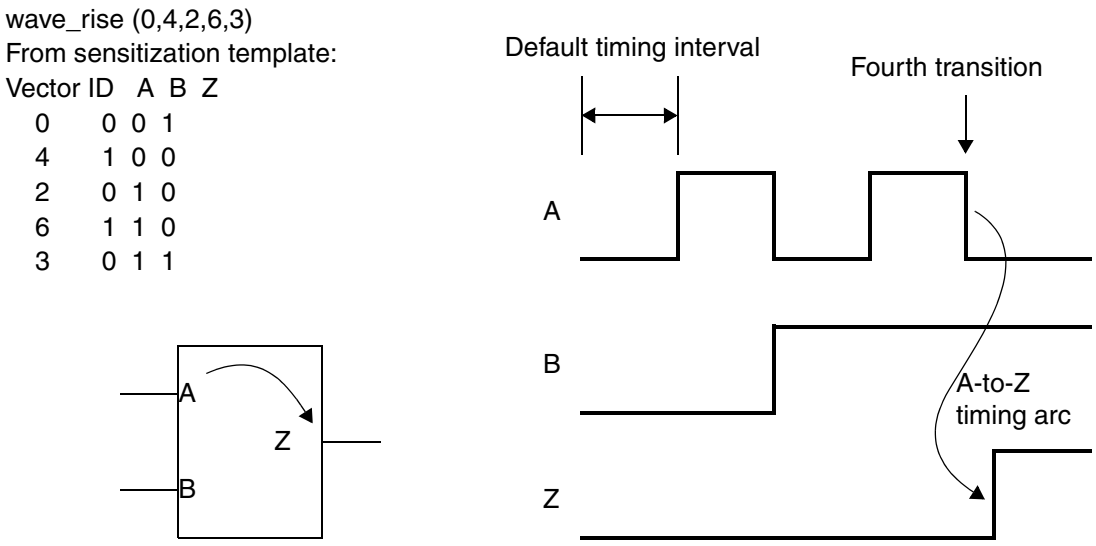
The sensitization template `2in_1out` defines eight vectors. Each vector defines a logic value for the cell pins in the order corresponding to the `pin_names` list.

The `cell(my_cell)` statement instantiates the sensitization template `2in_1out` and maps its pins A, B, and Z to IN1, IN2, and OUT, respectively.

For the timing arc from A to Z, the attribute `wave_rise` defines waveforms at pins A and B that result in a rising edge at pin Z, using a sequential list of vectors previously defined in the `2in_1out` template. The `wave_rise_sampling_index` attribute defines the transition number corresponding to `wave_rise` on which the delay and slew values are to be measured for the timing arc. In this example, it is set to the fourth transition. The sensitization for a falling transition at the output is similarly defined by the `wave_fall` and `wave_fall_sampling_index` attributes.

Figure B-5 shows the interpretation of the `wave_rise` attribute in PrimeTime SI. The vector sequence (0, 4, 2, 6, 3) represents the waveforms applied to pins A and B in order to generate a rising edge at pin Z. The delay and slew measurements of the A-to-Z timing arc occur on the fourth sequence transition (in this case, the transition between vector ID number 6 and vector ID number 3).

Figure B-5 Usage of wave_rise in Library Sensitization Definition



PrimeTime SI assigns a time interval between transitions to a fixed interval such that the associated logic cell has sufficient time to settle down and initialize itself to a proper state. A different time interval can be specified with the `wave_rise_timing_interval` attribute. For example, to set the time interval to 0.5 library time units:

```
wave_rise_timing_interval : 0.5;
```

Due to the adaptive time-stepping used by SPICE simulators, providing more simulation time than necessary for the logic to stabilize will not increase simulation runtime.

Library Driver Waveform

When `write_spice_deck` sensitizes the input of a cell, it should use the same waveform that was used for characterization of the timing data. PrimeTime SI gets the predriver modeling information from the library, if available.

If the library does not contain the predriver information, the standard Synopsys predriver is used by default. To specify the predriver type explicitly in PrimeTime SI, use the following command:

```
set_library_driver_waveform
  [-type ramp | standard ]
  [library_objects]
```

The `-type` setting specifies the predriver type, either a simple ramp or the standard Synopsys predriver. If you specify one or more library objects (a collection of libraries or library cells) in the command, it applies only to those objects. Otherwise, the command applies to the whole design.

The library driver setting affects not only `write_spice_deck`, but also the predriver used for advanced delay calculation using CCS models. For more information, see "Advanced Delay Calculation Using CCS Models" on page 2-29.

Generated SPICE Deck Example

Here is an example of a `write_spice_deck` command:

```
pt_shell> write_spice_deck -output my_output \
            -sub_circuit_file spice_subckt \
            -logic_one_voltage 1.8 \
            [get_timing_paths -justify]
```

This example produces a SPICE deck file named my_output, based on the definitions given in the SPICE subcircuit file called spice_subckt. The `-logic_one_voltage` option specifies the upper voltage swing of the gate input pins to 1.8 volts, which affects piecewise linear (PWL) model generation.

Example B-1 shows the output file generated by this command. For the applicable input SPICE subcircuit definitions, see Example B-2.

Example B-1 SPICE Deck Output

```
MAX. critical path section: (falling) SecIn -> (falling)
SecondReg/D.
*.global vdd vss
*vvdd vdd 0 1.8
*vvss vss 0 0
.include "./spice/spi_deck.subckt"
***   critical path 0 has 7 pins.
********** Arrival Window Info. for pin 'SecIn' **********
* {myclock} pos_edge {min_r_f 0.1 0.1} {max_r_f 0.1 0.1}
* --clock-- {0 2}
****************************************
* !!! INFO: PrimeTime created the following critical path falling
```

```
input port 'SecIn' waveform.
* Please verify.
* For rising pwl
* vSecIn SecIn 0 pwl(0.0ns 0 10.0995ns 0 10.1005ns 1.8)
* For falling pwl
vSecIn SecIn 0 pwl(0.0ns 1.8 10.0995ns 1.8 10.1005ns 0)
****************************************
* resistor(s) for net 'SecIn'.
r0      SecIn:4      SecIn:8      4.000000
r1      SecIn:3      SecIn:4      4.000000
r2      PreInv/A     SecIn:3      4.000000
r3      SecIn:5      SecIn:6      1.368000
r4      SecIn:6      SecIn:8      0.049228
r5      SecIn:5      SecIn        0.053363
* ground capacitors(s) for net 'SecIn'.
c0      PreInv/A     0      0.052000ff
* c1    SecIn:4      0      0.000000ff
c2      SecIn        0      0.081000ff
c3      SecIn:5      0      0.404000ff
c4      SecIn:6      0      0.009000ff
* c5    SecIn:8      0      0.000000ff
c6      SecIn:3      0      0.026000ff
* cross capacitance of net 'SecIn'.
cc0      SecIn:4      PreInv/Y      0.008000ff
cc1      SecIn:4      PreNet:14     0.026000ff
cc2      SecIn:4      PreNet:8      0.023000ff
cc3      SecIn:4      PreNet:4      0.023000ff
cc4      SecIn:8      PreInv/Y      0.009000ff
cc5      SecIn:8      PreNet:14     0.029000ff
cc6      SecIn:8      PreNet:8      0.026000ff
cc7      SecIn:8      PreNet:4      0.026000ff
cc8      SecIn:3      PreInv/Y      0.008000ff
cc9      SecIn:3      PreNet:14     0.026000ff
cc10     SecIn:3      PreNet:8      0.023000ff
cc11     SecIn:3      PreNet:4      0.023000ff
```

Example B-2 SPICE Subcircuit Input File

```
.SUBCKT buf1a3 A Y
MU11 N1N4 A VSS  VSS  n L=0.24 W=1.66
MU12 N1N4 A VDD  VDD  p L=0.24 W=2.50
MU21 Y N1N4 VSS  VSS  n L=0.24 W=1.66
MU22 Y N1N4 VDD  VDD  p L=0.24 W=2.50
.ENDS
.SUBCKT fdf1a6 CLK D Q
M1 N1N56 TP2 N1N119  VSS  n L=0.24 W=1.00
M2 N1N119 D VSS  VSS  n L=0.24 W=1.00
M3 N1N56 TP3 N1N35  VSS  n L=0.24 W=0.58
M4 N1N35 TP7 VSS  VSS  n L=0.24 W=0.58
M5 TP7 N1N56 VSS  VSS  n L=0.24 W=1.00
M6 TP7 TP3 N1N46  VSS  n L=0.24 W=0.58
M7 N1N46 TP2 N1N37  VSS  n L=0.24 W=0.58
M8 N1N37 N1N108 VSS  VSS  n L=0.24 W=0.58
```

```
M9 TP3 TP2 VDD  VDD  p L=0.24 W=0.78
.ENDS
.SUBCKT inv1a6 A Y
M1 Y A VSS  VSS  n L=0.24 W=3.32
M2 Y A VDD  VDD  p L=0.24 W=5.00
.ENDS
.SUBCKT or2c3 A B Y
M1I551 N1I551N10 B VSS  VSS  n L=0.24 W=1.66
M1I552 Y A N1I551N10  VSS  n L=0.24 W=1.66
M1I553 Y A VDD  VDD  p L=0.24 W=2.50
M1I554 Y B VDD  VDD  p L=0.24 W=2.50
.ENDS
```

C

SPEF Warning Messages

When PrimeTime SI reads a Standard Parasitic Exchange Format (SPEF) file, it parses and checks the file to ensure that it correctly conforms to the SPEF format. If there are problems with the SPEF file, a warning message is issued. To verify and resolve problems you might encounter, see the following sections:

- SPEF Warning Messages
- SPEF Attributes

SPEF Warning Messages

The following warning messages are issued if the cross-coupled capacitors are specified incorrectly in the SPEF file:

- PARA-021

Warning: SPEF aggressor in '*first_net_name:sub_node*' not found in design '*my_design*'. (PARA-021)

The message indicates that a net or net node cannot be found in the netlist. This typically means that a capacitor defined on a net, and the net it is connected to, cannot be found in the netlist.

- PARA-022

Warning: SPEF cross capacitor (*first_net_name:sub_node second_net_name:sub_node capacitor value*) is reduced due to missing aggressor subnode. (PARA-022)

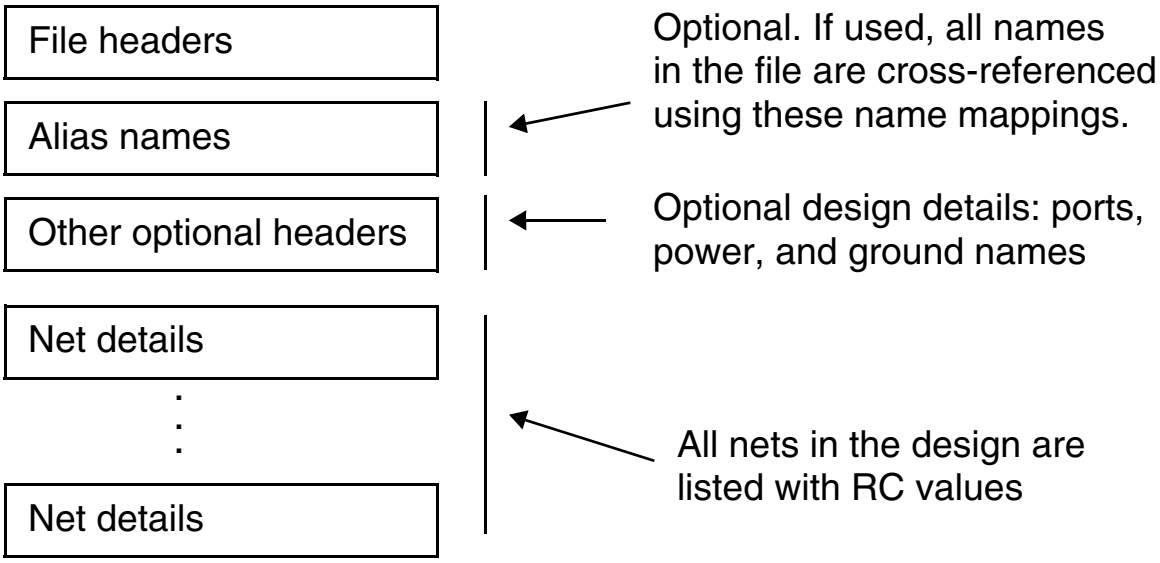
The message indicates that a cross-coupled capacitor is connected to *first_net_name:sub_node* but the other end of the capacitor, *second_net_name:sub_node*, cannot be found in the database, and therefore the capacitance is grounded.

SPEF Attributes

The SPEF standard allows many attributes of a design to be specified. One key attribute of the SPEF standard is that headers are not required in the SPEF file, but if the header is present in the file, information must follow.

Figure C-1 shows the SPEF generated from a coupled circuit. The illustration describes the key attributes related to cross-coupling capacitance.

Figure C-1 Overview of SPEF File



An example file header is shown in Figure C-2. The key attributes described in the file headers are the design name, units, hierarchical divider, and information regarding the source of the file.

Figure C-2 SPEF File Header

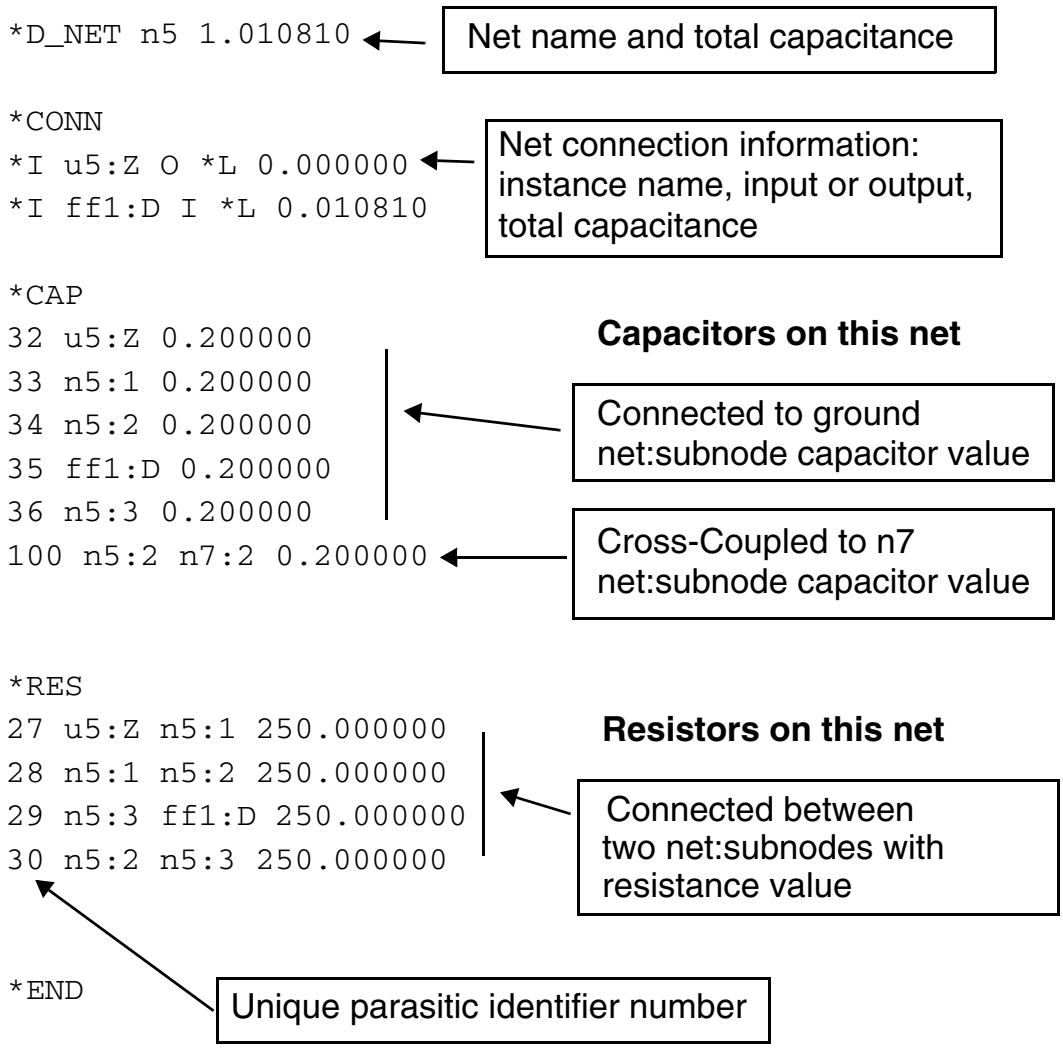
```
*SPEF "1481-1998"
*DESIGN "xtalk_2ff"
*DATE "Fri Oct 21 15:07:57 2000"
*VENDOR "Synopsys, Inc."
*PROGRAM "PrimeTime-SI"
*VERSION "2000.10-SI2"
*DESIGN_FLOW "Synopsys"
*DIVIDER /
*DELIMITER :
*T_UNIT 1 NS
*C_UNIT 1 PF
*R_UNIT 1 OHM
*L_UNIT 1 HENRY
```

Design details

Default units

As shown in Figure C-3, the key section of the SPEF file is the section that describes the net details. This section starts with the keyword *DNET and ends with keyword *END.

Figure C-3 Net Details of the SPEF File



For valid SPEF, n7 must also be coupled to n5.
The *D_NET for n7 must contain “100 n7:2 n5:2 0.200000”
Note the use of the same unique identifier and value.

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W