

Tempus User Guide

Product Version 21.10

May 2021

© 2011-2021 Cadence Design Systems, Inc. All rights reserved.

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

The publication may be used solely for personal, informational, and noncommercial purposes;

The publication may not be modified in any way;

Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and

Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

<u>About This Manual</u>	9
<u>Audience</u>	9
<u>How This Manual Is Organized</u>	9
<u>Conventions Used in This Manual</u>	10
<u>Related Documents</u>	11

1

<u>Product and Licensing Information</u>	13
<u>1.1 Tempus Product and Licensing Overview</u>	14
<u>1.2 About Licensing</u>	14
<u>1.3 Tempus Timing Signoff Solution Products</u>	15
<u>1.4 Tempus Timing Signoff Solution Product Options</u>	17
<u>1.5 Tempus License Options for Single View Distributed STA (DSTA)</u>	19
<u>1.6 Tempus License Options for Distributed MMMC (DMMMC)</u>	20
<u>1.7 Tempus License Options for Concurrent MMMC (CMMMC)</u>	21
<u>1.8 Tempus License Options for CMMMC in DSTA Mode</u>	22
<u>1.9 Tempus Paradime Flow Licensing for Interactive ECOs in CMMMC Mode</u>	23
<u>1.10 About Tempus Timing Signoff Solution Licenses</u>	24
<u>1.11 Checking Out Tempus Licenses for Product Options</u>	25

2

<u>Design Import</u>	27
<u>2.1 Design Import Overview</u>	28
<u>2.1.1 Input Requirements</u>	28
<u>2.2 Design Import Flow</u>	30
<u>2.3 Performing Design Sanity Checks in Tempus</u>	31
<u>2.4 Tempus File Management</u>	32

3

<u>Installation and Startup</u>	34
<u>3.1 Tempus Product and Installation Information</u>	35
<u>3.2 Setting Up the Tempus Run Time Environment</u>	35
<u>3.2.1 Temporary File Locations</u>	36
<u>3.3 Launching the Tempus Console</u>	36
<u>3.3.1 Completing Command Names</u>	37
<u>3.3.2 Command-Line Editing</u>	37
<u>3.3.3 Setting Preferences</u>	39
<u>3.4 Starting the Tempus Software</u>	39
<u>3.4.1 Using the Log File Viewer</u>	40
<u>3.5 Accessing Tempus Documentation and Help</u>	41
<u>3.6 Accessing Documentation and Help from Tempus GUI</u>	41
<u>3.7 Using the Tempus man and help Commands on the Command-Line</u>	43
<u>3.8 Other Sources of Cadence Product Information</u>	44

4

<u>Analysis and Reporting</u>	45
<u>4.1 Base Delay Analysis</u>	46
<u>4.1.1 Overview</u>	47
<u>4.1.2 Base Delay Analysis Flow</u>	47
<u>4.1.3 Limitations of Traditional Delay Calculators</u>	49
<u>4.1.4 Performing Base Delay Analysis</u>	51
<u>4.1.5 Base Delay Reporting</u>	56
<u>4.1.6 Cycle-to-Cycle Jitter using Native Delay Calculation</u>	58
<u>4.2 Signal Integrity Delay and Glitch Analysis</u>	60
<u>4.2.1 Overview</u>	61
<u>4.2.2 Understanding Attacker and Victim Nets</u>	61
<u>4.2.3 SI Delay Analysis - An Overview</u>	61
<u>4.2.4 Signal Integrity Delay Analysis Flow</u>	63
<u>4.2.5 Performing Signal Integrity Delay Analysis</u>	64
<u>4.2.6 SI Delay Reporting</u>	89
<u>4.2.7 SI Glitch Analysis - An Overview</u>	95
<u>4.2.8 SI Glitch Analysis Flow</u>	97

Tempus User Guide

<u>4.2.9 Understanding SI Glitch Analysis</u>	98
<u>4.2.10 Running Detailed SI Glitch Analysis</u>	104
<u>4.2.11 SI Glitch Reporting</u>	105
<u>4.2.12 Overshoot-Undershoot Glitch Analysis - An Overview</u>	112
<u>4.2.13 Running Overshoot-Undershoot Glitch Analysis - Sample Script</u>	116
<u>4.3 Timing Analysis Modes</u>	118
<u>4.3.1 Overview</u>	119
<u>4.3.2 Specifying Timing Analysis Mode Types</u>	120
<u>4.3.3 Advanced On-Chip Variation (AOCV) Analysis</u>	138
<u>4.3.4 Statistical On-Chip Variation (SOCV) Analysis</u>	165
<u>4.4 Path-Based Analysis</u>	182
<u>4.4.1 Overview</u>	183
<u>4.4.2 GBA vs PBA Comparison</u>	183
<u>4.4.3 Reporting in Path-Based Analysis</u>	183
<u>4.4.4 Path-Based Analysis Reporting Models</u>	185
<u>4.4.5 Path-Based Analysis Essence</u>	186
<u>4.5 Analysis Of Simultaneous Switching Inputs -SSI</u>	189
<u>4.5.1 Overview</u>	190
<u>4.5.2 Implementation</u>	191
<u>4.5.3 Reporting</u>	193
<u>4.5.4 Considerations for Path-Based Analysis (PBA)</u>	194
<u>4.6 Waveform Aware and Rail Swing Checks</u>	195
<u>4.6.1 Overview</u>	196
<u>4.6.2 Rail Swing Checks - An Overview</u>	196
<u>4.6.3 Rail Swing Reporting</u>	197
<u>4.6.4 Waveform Aware Pulse Width Checks - An Overview</u>	197
<u>4.6.5 Waveform Aware Pulse Width Checks Reporting</u>	198
<u>4.7 Set Instance Library Flow</u>	200
<u>4.7.1 Overview</u>	201
<u>4.7.2 MMMC set_instance_library Flow</u>	201
<u>4.7.3 MMMC Flow Script</u>	202
<u>4.7.4 Save and Restore MMMC set_instance_library Flow</u>	204

5

Concurrent Multi-Mode Multi-Corner Timing Analysis 1

<u>5.1 Overview</u>	2
<u>5.2 Multi-Mode Multi-Corner Licensing</u>	2
<u>5.3 Configuring Setup for Multi-Mode Multi-Corner Analysis</u>	2
<u>5.4 Library Sets</u>	3
<u>5.5 Virtual Operating Conditions</u>	5
<u>5.6 RC Corner Objects</u>	7
<u>5.7 Delay Corner Objects</u>	7
<u>5.8 Power Domain Definitions</u>	9
<u>5.9 Constraint Mode Objects</u>	11
<u>5.10 Constraint Support in Multi-Mode and MMMC Analysis</u>	13
<u>5.11 Analysis Views</u>	14
<u>5.12 Design Loading and MMMC Configuration Script</u>	17
<u>5.13 Saving Multi-Mode Multi-Corner Configuration</u>	19
<u>5.14 Controlling Multi-Mode Multi-Corner Analysis through the Flow</u>	19
<u>5.15 Performing Timing Analysis</u>	19
<u>5.16 Generating Timing Reports</u>	20

6

Distributed Multi-Mode Multi-Corner Timing Analysis 21

<u>6.1 Overview</u>	22
<u>6.2 Performing Distributed Multi-Mode Multi-Corner Analysis</u>	23
<u>6.2.1 Design Loading and MMMC Configuration Script</u>	28
<u>6.3 Performing Concurrent MMMMC Analysis in DMMMC Mode</u>	34
<u>6.4 Setting Up MSV/CPF-Based Design in DMMMC Mode</u>	34
<u>6.5 Performing Setup/Hold View Analysis and Reporting</u>	36
<u>6.6 Running Single Interactive User Interface in DMMMC Mode</u>	37
<u>6.7 Running Interactive ECO on Multiple Views in DMMMC Mode</u>	38
<u>6.8 Important Guidelines and Troubleshooting DMMMC</u>	41

7

Distributed Static Timing Analysis 44

<u>7.1 Tempus Timing Analysis Overview</u>	45
--	----

<u>7.1.1 Static Timing Analysis (STA)</u>	45
<u>7.1.2 Distributed Static Timing Analysis (DSTA)</u>	45
<u>7.2 Converting STA to DSTA</u>	47
<u>7.3 Hardware Recommendations for DSTA</u>	48
<u>7.4 Accessing Specific Machines using LSF</u>	49
<u>7.5 DSTA Log Files</u>	50
<u>7.6 DSTA Operational Tips</u>	51
<u>7.7 Distributed Static Timing Analysis Flow</u>	54
<u>7.8 Additional Commands Supported in DSTA</u>	55
<u>7.9 Commands Not Supported in DSTA Mode</u>	58

8

<u>Signoff ECO</u>	65
<u>8.1 Signoff ECO Overview</u>	67
<u>8.2 Key Features</u>	67
<u>8.3 Signoff ECO Flow</u>	69
<u>8.4 Setting Up Signoff ECO Environment</u>	70
<u>8.5 Signoff Optimization Use Models</u>	71
<u>8.6 Basic Signoff ECO Optimization Techniques</u>	83
<u>8.7 Fixing SI Violations</u>	86
<u>8.8 Fixing IR Drop in Tempus Signoff ECO</u>	89
<u>8.9 Path-Based Analysis (PBA) Mode Optimization</u>	89
<u>8.10 Total Power Optimization</u>	90
<u>8.11 Setup Timing Recovery After Large Leakage or Total Power Optimization</u>	91
<u>8.12 Recommendations for Getting Best Total Power Optimization</u>	91
<u>8.13 Hierarchical ECO Optimization</u>	92
<u>8.14 Top Down Block ECO Flow</u>	96
<u>8.15 Generating Node Locations in Parasitic Data</u>	98
<u>8.16 Optimization Along Wire Topologies</u>	99
<u>8.17 Optimization using Endpoint Control</u>	99
<u>8.18 Metal ECO Flow</u>	102
<u>8.19 Handling Large Number of Active Timing Views Using SmartMMMC</u>	104

9

Timing Model Generation	106
 9.1 Extracted Timing Models	107
 9.1.1 Overview	108
 9.1.2 ETM Generation	109
 9.1.3 ETM Generation for MMMC Designs	111
 9.1.4 Slew Propagation Modes in Model Extraction	111
 9.1.5 Basic Elements of Timing Model Extraction	112
 9.1.6 Secondary Load Dependent Networks	125
 9.1.7 Characterization Point Selection	126
 9.1.8 Constraint Generation during Model Extraction	128
 9.1.9 Parallel Arcs in ETM	132
 9.1.10 Latency Arcs Modeling	133
 9.1.11 Latch-Based Model Extraction	133
 9.1.12 Model Extraction in AOCV Mode	134
 9.1.13 Stage Weight Modeling in ETM	134
 9.1.14 AOCV Derating Mode	135
 9.1.15 PG Pin Modeling During Extraction	136
 9.1.16 Extracted Timing Models with Noise (SI) Effect	137
 9.1.17 Extracted Timing Models with SOCV	138
 9.1.18 Merging Timing Models	138
 9.1.19 Limitations of ETM	140
 9.1.20 Validation of Generated ETM	144
 9.1.21 Auto-Validation of ETM	147
 9.1.22 ETM Extremity Validation	148
 9.1.23 Limitation/Implications of EV-ETM	150
 9.1.24 Ability to Check Timing Models	150
 9.2 Boundary Models	151
 9.2.1 Overview	152
 9.2.2 Boundary Model Flow	152
 9.2.3 Hierarchical Flow with Boundary Model	153
 9.2.4 Clock Mapping in Top Level Run with Boundary Models	157
 9.2.5 Recommendations for Generating Accurate Boundary Models	158
 9.2.6 Glitch-Specific Boundary Models	159
 9.2.7 Boundary Models in MMMC Mode	163

Tempus User Guide

<u>9.2.8 Save-Restore of Hierarchical Analysis</u>	163
<u>9.3 SmartScope Hierarchical Analysis</u>	164
<u>9.3.1 Overview</u>	165
<u>9.3.2 Why Use SmartScope Analysis</u>	166
<u>9.3.3 Choosing Blocks to Use for SmartScope Analysis</u>	166
<u>9.3.4 SmartScope Analysis Flow</u>	167
<u>9.3.5 Running SmartScope Hierarchical Analysis</u>	169
<u>9.3.6 Scope Generation from Distributed STA (DSTA)</u>	171
<u>9.4 Prototype Timing Modeling</u>	172
<u>9.4.1 Overview</u>	173
<u>9.4.2 Inputs and Outputs</u>	173
<u>9.4.3 Using the PTM Flow to Generate Dotlib Model</u>	174
<u>9.4.4 Handling of Bus Bits in PTM</u>	178
<u>9.5 Interface Logic Models</u>	182
<u>9.5.1 Overview</u>	183
<u>9.5.2 Creating ILMs</u>	183
<u>9.5.3 Specifying ILM Directories at the Top Level</u>	186
<u>9.5.4 ILMs Supported in MMMC Analysis</u>	187
<u>9.5.5 ILM Validation Flow</u>	188
<u>9.5.6 Use of SDF, SDC, and XTWF in ILM Flow</u>	190
<u>9.5.7 Creating XILM</u>	193
<u>9.5.8 Using ILM/XILM for Hierarchical Analysis</u>	194
 <u>10 Tempus Power Integrity Analysis</u>	196
<u>10.1 Tempus Power Integrity Analysis Overview</u>	197
<u>10.2 Licensing Requirements</u>	200
<u>10.3 Tempus Power Integrity Flow</u>	201
<u>10.4 Performing Tempus PI Analysis</u>	202
<u>10.5 Tempus PI Result Analysis and Reporting</u>	208
<u>10.6 User-Defined Critical Paths Support</u>	211
<u>10.7 Enabling Proximity Aggressors in Tempus PI</u>	212
<u>10.8 Sequential Activity and Simulation Period</u>	215
<u>10.9 Tempus PI Related STA Commands</u>	216

11

<u>Low Power Flows</u>	223
<u>11.1 Low Power Overview</u>	224
<u>11.2 Low Power Flow</u>	224
<u>11.3 PGV Flows</u>	229
<u>11.4 Importing the Design in Low Power Flow</u>	235
<u>11.5 Non-MMMC Flow Settings in Low Power Flow</u>	235

12

<u>Appendix</u>	236
<u>Appendix - Timing Debug GUI</u>	237
<u>Appendix - Multi-CPU Usage</u>	263
<u>Appendix - Base Delay, SI Delay, and SI Glitch Correlation with SPICE</u>	269
<u>Appendix - Supported CPF Commands</u>	279

13

<u>Glossary</u>	338
------------------------------	-----

About This Manual

The Tempus Timing Signoff Solution software, also known as Tempus, provides a sign-off timing and signal integrity solution for a design flow. This manual describes how to install, configure, and use Tempus to implement digital integrated circuits.

Audience

This manual is written for experienced designers of digital integrated circuits. Such designers must be familiar with design planning, placement and routing, block implementation, chip assembly, and design verification. Designers must also have a solid understanding of UNIX and Tcl/Tk programming.

How This Manual Is Organized

The chapters in this manual are organized to follow the flow of tasks through the design process. Because of variations in design implementations and methodologies, the order of the chapters will not correspond to any specific design flow.

Each chapter focuses on the concepts and tasks related to the particular design phase or topic being discussed.

In addition, the following sections provide prerequisite information for using the Tempus software:

- **"Installation and Startup"**

Describes how to install, set up, and run the Tempus software, and use the online Help system.

- **"Design Import"**

Describes how to prepare data for analysis in the Tempus software. Tempus Timing Signoff Solution

Conventions Used in This Manual

This section describes the typographic and syntax conventions used in this manual.

text Indicates text that you must type exactly as shown. For example:

```
report_annotated_check -missing_setup
```

text Indicates information for which you must substitute a name or value.

In the following example, you must substitute the name of a specific file for *worst_entries*:

```
report_clock_timing -type skew  
-nworst worst_entries
```

text Indicates the following:

- Text found in the graphical user interface (GUI), including form names, button labels, and field names
- Terms that are new to the manual, are the subject of discussion, or need special emphasis
- Titles of manuals

[]

Indicates optional arguments.

In the following example, you can specify none, one, or both of the bracketed arguments:

```
command [-arg1] [arg2 value]
```

[|]

Indicates an optional choice from a mutually exclusive list.

In the following example, you can specify any of the arguments or none of the arguments, but you cannot specify more than one:

```
command [arg1 | arg2 | arg3 | arg4]
```

{ | }

Indicates a required choice from a mutually exclusive list.

In the following example, you must specify one, and only one, of the arguments:

```
command {arg1 | arg2 | arg3}
```

Tempus User Guide

About This Manual

{ [] [] }	Indicates a required choice of one or more items in a list. In the following example, you must choose one argument from the list, but you can choose more than one: command { [arg1] [arg2] [arg3]}
{ }	Indicates curly braces that must be entered with the command syntax. In the following example, you must type the curly braces: command arg1 {x y}
...	Indicates that you can repeat the previous argument.
.	Indicates an omission in an example of computer output or input.
.	
<i>Command – Subcommand</i>	Indicates a command sequence, which shows the order in which you choose commands and subcommands from the GUI menu. In the following example, you choose <i>Analysis</i> from the menu, then <i>Specify Analysis Mode</i> from the submenu, and then <i>Advanced</i> from the tabs: <i>Analysis – Specify Analysis Mode – Advanced</i> This sequence opens the Specify Analysis Mode Advanced form.

Related Documents

For more information about Tempus, see the following documents. You can access these and other Cadence documents with the Cadence Help online documentation system.

- [Tempus Known Problems and Solutions](#)

Describes important Cadence Change Requests (CCRs) for Tempus, including solutions for working around known problems.

- [Tempus Text Command Reference](#)

Describes the Tempus text commands, including syntax and examples.

- [Tempus Menu Reference](#)

Describes Tempus graphical user interface.

Tempus User Guide

About This Manual

- **New Features in Tempus**

Provides information about new features in this release of Tempus.

- **Tempus Foundation Flows User Guide**

Describes the Cadence-recommended procedures for performing basic STA and SI signoff, MMMC signoff, and CPF-based MMMC signoff using Tempus Timing Signoff Solution.

- **README file**

Contains installation, compatibility, and other prerequisite information, including a list of Cadence change requests (CCRs) that were resolved in this release. You can read this file online at downloads.cadence.com.

Product and Licensing Information

- 1.1 [Tempus Product and Licensing Overview](#) on page 14
- 1.2 [About Licensing](#) on page 14
- 1.3 [Tempus Timing Signoff Solution Products](#) on page 15
- 1.4 [Tempus Timing Signoff Solution Product Options](#) on page 17
- 1.5 [Tempus License Options for Single View Distributed STA \(DSTA\)](#) on page 19
- 1.6 [Tempus License Options for Distributed MMMC \(DMMMC\)](#) on page 20
- 1.7 [Tempus License Options for Concurrent MMMC \(CMMMC\)](#) on page 21
- 1.8 [Tempus License Options for CMMMC in DSTA Mode](#) on page 22
- 1.9 [Tempus Paradime Flow Licensing for Interactive ECOs in CMMMC Mode](#) on page 23
- 1.10 [About Tempus Timing Signoff Solution Licenses](#) on page 24
- 1.11 [Checking Out Tempus Licenses for Product Options](#) on page 25

1.1 Tempus Product and Licensing Overview

Each Cadence Tempus Timing Signoff Solution product is sold as part of a product package. Product packages might also include product options. The options provide advanced features and capabilities, such as support for enabling distributed client servers and/or accelerating performance through increased CPU multi-threading.

Each product and product option has a corresponding license. The software uses licenses to determine the features that are available when the software runs.

1.2 About Licensing

The following terminology is useful in understanding licenses.

■ **Base license**

The license that is checked out when the software starts. Only a full-fledged product license can be used as a base license. You cannot use a product option license as a base license to start the software.

■ **Dynamic license**

A license for a product option that is not checked out until a feature provided by the product option is needed. You can check out more than one dynamic license per base license. For more information on dynamic licenses, see [Checking Out Tempus Licenses for Product Options](#).

■ **Multi-CPU license**

A license that enables additional CPUs for multi-threading, Superthreading, or distributed processing. Multi-CPU licenses must be product licenses, and can be checked out after the base license is checked out. You can check out more than one multi-CPU license per base license.

For more information on multi-CPU licenses, For more information on multi-CPU licenses, see Innovus System Licensing and Packaging on SourceLink ®.

■ **Licensing Server license**

Tempus running on Linux x86 platforms is integrated with FLEXnet version 11.16.4.0. The Tempus software using this version also requires the upgraded license server version, which is integrated with FLEXnet 11.16.4.0 or higher versions. If the license server is not updated, the Tempus software will not be able to run.

You can check the version of the licensing server by running the following commands in the terminal where the license server is installed:

```
lmgrd -v  
cdslmd -v
```

1.3 Tempus Timing Signoff Solution Products

The Tempus Timing Signoff solution package includes the products listed in the table below. To start any of these products, type the following UNIX/Linux command:

tempus

Table 1-1 Tempus Timing Signoff Solution Products

Name	Abbreviation	Product Number	Description
Tempus Timing Signoff Solution L	Tempus L	TPS100	Tempus L is one of the two base licenses in the Tempus line. Tempus L provides better timing analysis performance for both Graph-Based Analysis (GBA) and Path-Based Analysis (PBA) versus Tempus XL and also allows for multi-threaded PBA, which is unavailable in Tempus. Tempus L is functionally equivalent to Tempus XL with the exception of concurrent MMMC operation. Concurrent MMMC is available only in Tempus XL.

Tempus User Guide
Product and Licensing Information

Name	Abbreviation	Product Number	Description
			<p>Tempus L can be paired with Tempus ECO for Signoff ECO and it can be paired with Tempus MP to enable more CPUs for multi-threading.</p> <p>Tempus L can be used for acceleration and enables 8 CPUs.</p> <p>Tempus L can checkout Tempus ECO, or Voltus L/Voltus XL to load the concurrent MMMC view definitions by default. If any of these licenses are not available, the software will load the design with a SMSC (single-mode single-corner) definition.</p>
Tempus Timing Signoff Solution XL	Tempus XL	TPS200	<p>Tempus XL provides all the functionality of Tempus L and includes the primary innovation of distributed design. Distributed design leverages massive parallel computation by leveraging multiple compute servers to perform static timing analysis for a single view for designs up to 100s of millions of placeable cell instances.</p> <p>In addition, Tempus XL provides eight additional CPUs over Tempus L - no compromise on hierarchical/incremental analysis, or concurrent MMMC analysis in a single session. Tempus XL can be used for acceleration and enables 16 CPUs.</p>

Tempus User Guide
Product and Licensing Information

Name	Abbreviation	Product Number	Description
<u>Virtuoso Digital Signoff Timing Solution</u>		<u>VDS100</u>	This is a single option of integrated solution for STA in Virtuoso for Mixed-Signal designs. This option provides a design capacity limit of 50K instances and can be executed either from Virtuoso or run as standalone. It supports GBA/PBA signoff engine, noise glitch analysis, OCV/AOCV, DMMMC, CMMMC, and SOCV for advanced node designs.
<u>Tempus Library Validation Solution</u>		<u>TPS500</u>	This is a single option for library validation for timing, noise, and power. This option allows comprehensive checking across Genus, Innovus, Tempus, and Voltus and can handle NLDM, CCS, and ECSM library formats.

1.4 Tempus Timing Signoff Solution Product Options

The Tempus Timing Signoff solution product options provide the extendibility and cost-effective access to additional advanced technologies, such as support for enabling distributed client servers and/or accelerating performance through increased CPU multi-threading. These options are listed in the table below.

Tempus User Guide
Product and Licensing Information

Table 1-2 Tempus Timing Signoff Solution Product Options

Name	Abbreviation	Product Number	Description
Tempus Timing Signoff Solution ECO	Tempus ECO	TPS300	<p>The Tempus ECO option works with either Tempus L or Tempus XL to enable MMMC signoff ECO functionality. This enables physically-aware MMMC timing optimization for setup/hold and DRV violations. It also supports leakage power optimization.</p> <p>Tempus ECO is only an additional license, which is required to run physically-aware timing signoff optimization. Tempus ECO can also be used to enable concurrent MMMC operation when used with the base Tempus L license.</p>
Tempus Massively Parallel	Tempus MP	TPS400	<p>Tempus MP is a dual use accelerator for Tempus. In non-distributed analysis, Tempus MP adds sixteen CPUs to the base license of either Tempus L or Tempus XL for enhanced multi-threaded performance.</p>
			<p>While performing design distribution, Tempus MP is required for each distributed process. Distributed processes can be on separate hardware clients or can be packed on a single large compute server. In either case, it is one Tempus MP per process and each process enables up to sixteen CPUs. Tempus MP licenses can be stacked to provide more CPUs in blocks of sixteen. Tempus MP can be used for acceleration and enables 16 CPUs.</p>

Tempus User Guide
Product and Licensing Information

Name	Abbreviation	Product Number	Description
Tempus Timing Signoff Solution Power Integrity	Tempus PI	TPS600	The Tempus Power Integrity license enables identification of IR-sensitive critical paths. It is an option to the baseline licenses, Tempus-XL, or Voltus-XL.
<u>Tempus Advanced Analysis Option</u>		<u>TPS210</u>	<u>This is a single option required for all the designs using any of the functionality in this analysis package. This license is not tied to any specific technology node or foundry.</u>
<u>Tempus Timing Signoff Solution 3nm Option</u>		<u>TPS230</u>	<u>This is a single option required for all the 3nm designs independent of the foundry. This includes all the modeling features that are required by foundries to enable and support 3nm designs.</u>

Note: The default multi-CPU order is Tempus MP, Tempus L, and Tempus XL.

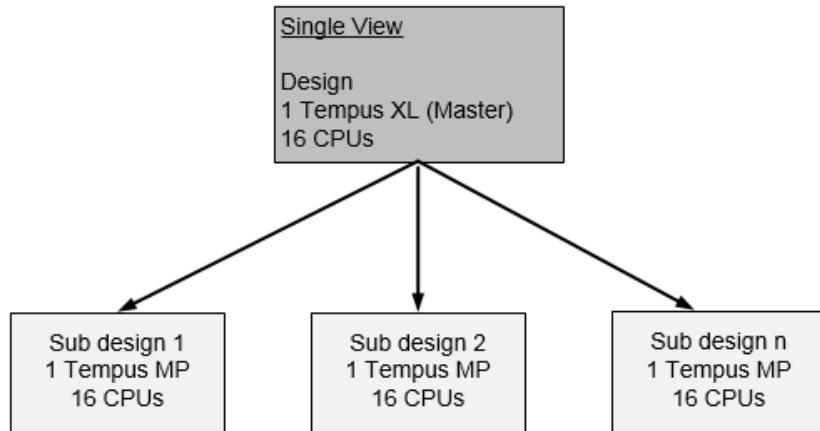
1.5 Tempus License Options for Single View Distributed STA (DSTA)

The following are the Tempus license requirements for single view DSTA mode:

- Master requires 1 Tempus XL (up to 16 CPUs).
- Clients require 1 Tempus MP each (up to 16 CPUs per client).
- Additional Tempus MP keys unlock additional CPUs for the master and client (up to 16 more per license).
- Each slave checks out a Tempus-MP license and also allows Tempus-XL functionality.

The order of licenses is shown in [Figure 1-1](#) on page 20:

Figure 1-1 DSTA license requirements

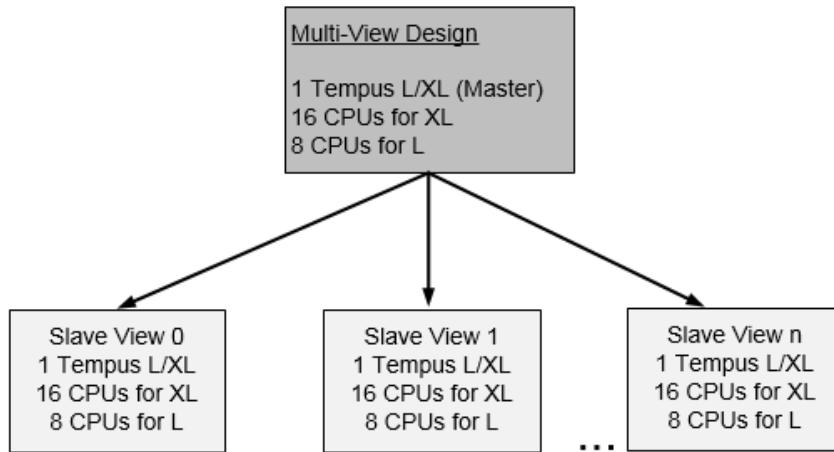


1.6 Tempus License Options for Distributed MMMC (DMMMC)

The following are the Tempus license requirements in DMMMC mode, see [Figure 1-2](#) on page 21:

- 1 Tempus L or XL to start the master session
- 1 Tempus L or XL per view or slave (1:1 view/slave ratio)
- Tempus XL enables 16 CPUs, Tempus L enables 8 CPUs
- The license order for slaves can be controlled with the `distributed_child_license_checkout_list` global variable, which defaults to "tpsl tpsxl vdsl".

Figure 1-2 DMMMC license requirements



1.7 Tempus License Options for Concurrent MMMC (CMMMC)

The following are the Tempus license requirements in CMMMC mode:

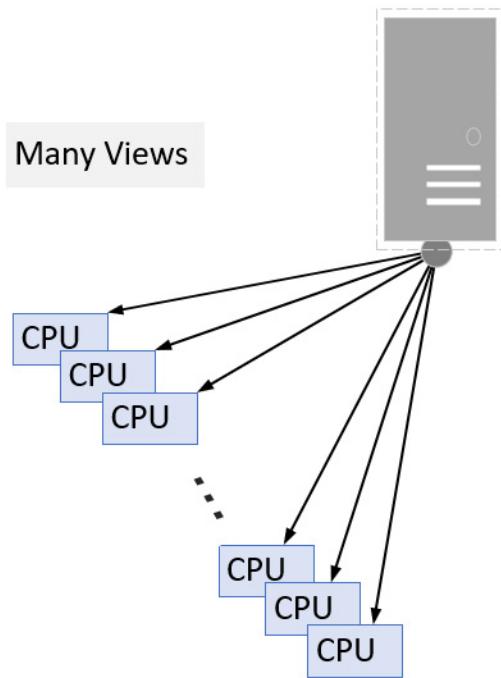
- When 'N' views are combined into a single STA job on a single machine, the following licenses are required:
 - 1 to 8 views: 1 XL (16 CPUs) or [1 L + 1 ECO] (8 CPUs)
 - 9 to 16 views: 2 XL (16 CPUs) (the L + ECO combo not accepted)
 - Unlimited views: 3 XL (16 CPUs) (the L + ECO combo not accepted)
- More CPUs enabled by checking out more L (8), XL (16), or MP (16) licenses
- Minimum configuration: 1/2/3 XL or 1(L + ECO)

Examples:

- CMMMC with 6 views on 12 CPUs = 1 XL
- CMMMC with 15 views on 18 CPUs = 2 XL + 1 XL or = 2 XL + 1 MP
- CMMMC with 20 views on 32 CPUs = 3 XL + 1 XL or = 3 XL + 1 MP

Note: The CMMMC STA licensing is different from Tempus ECO, which is also multi-mode/multi-corner.

Figure 1-3 CMMMC license requirements

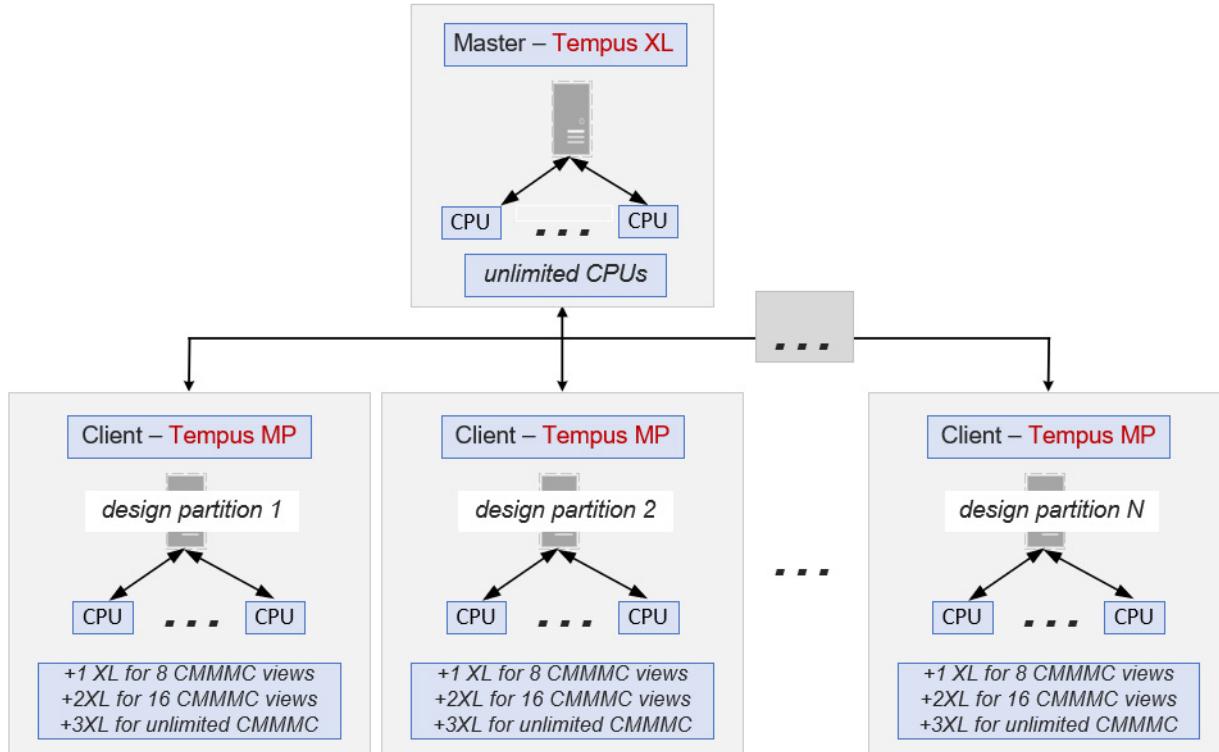


1.8 Tempus License Options for CMMMC in DSTA Mode

Tempus DSTA can distribute a CMMMC job across multiple computers. Each computer performs local multi-threading. The licensing requirements are given below:

- The master process requires an XL license:
 - unlimited views (master not affected by any number of views)
 - unlimited CPUs
- Each client requires a Tempus MP license - and licenses CMMMC just like non-DSTA jobs:
 - 1XL (16 CPU) or 1{L+ECO} (8 CPU) for <= 8 views
 - 2 XL for 9 to 16 views
 - 3 XL for unlimited views
 - Add extra CPUs with additional MP licenses
- The required minimum configuration for this setup is 3 XL + 2 MP.

Figure 1-4 CMMMC license requirements in DSTA mode



Examples:

- Run CMMMC with 6 views across 5 computers = 1XL + 5MP +5 XL = 80 CPUs (5 clients@16)
- Run CMMMC with 10 views across 3 computers = 1XL + 3MP + 6XL= 48 CPUs (3 clients@16)

1.9 Tempus Paradime Flow Licensing for Interactive ECOs in CMMMC Mode

Tempus Paradime keeps all the STA sessions alive so that ECO changes can be committed and timing can be immediately updated incrementally across all the views. The Paradime licensing model is based on a master-client architecture (DMMMC), where:

- The Master session requires one Tempus XL and one Tempus ECO.
- Each client requires a Tempus XL license.

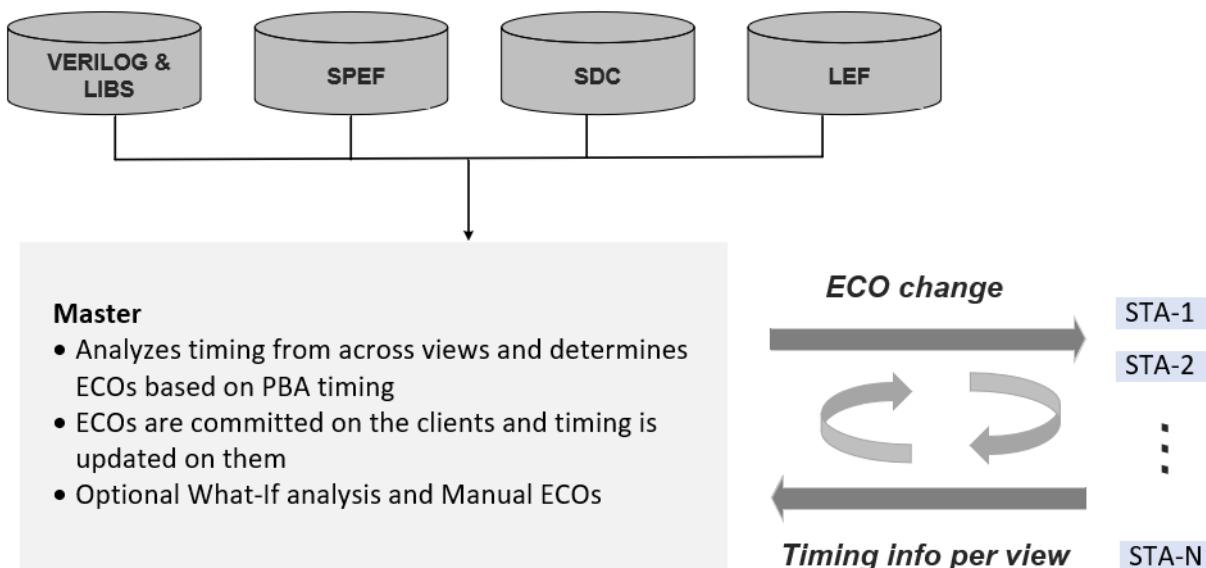
Tempus User Guide

Product and Licensing Information

You can stack extra Tempus MP, Tempus L, and Tempus XL licenses to enable more CPUs for each client session. All the licenses are checked out upon start up and are valid throughout the session.

The license flow is illustrated in the diagram below.

Figure 1-5 Tempus Paradime Flow Licensing in CMMMC Mode



1.10 About Tempus Timing Signoff Solution Licenses

When you run a command to invoke a product or product option, a license is checked out. Each product and product option has a unique license string (also called a license key). The following table lists the product and product option names, and provides the corresponding license strings.

Table 1-3 Product and product options and corresponding license strings

Product or Option	License String
Tempus Timing Signoff Solution L	TEMPUS_TIMING_SIGNOFF_L
Tempus Timing Signoff Solution XL	TEMPUS_TIMING_SIGNOFF_XL
Tempus Timing Signoff Solution ECO	TEMPUS_TIMING_SIGNOFF_TSO
Tempus Timing Signoff Solution MP	TEMPUS_TIMING_SIGNOFF_MP

The version of vendor daemon (cdslmd) and license daemon (lmgrd) should be 11.11.1.1 or higher. If the daemon versions have not been updated, Tempus may fail to checkout a license:

```
$ tempus  
Checking out Tempus Timing Signoff Solution license ...  
Fail to find any Tempus Timing Signoff Solution license...  
Check your Tempus Timing Signoff Solution license status.
```

To fix the above error, restart the license server using the latest cdslmd and lmgrd versions included in the release tree.

1.11 Checking Out Tempus Licenses for Product Options

The `tempus` command specifies the product options to be checked out when you invoke the software. The product options specified with the `-lic_startup_options` parameter are checked out immediately and the product options specified with the `-lic_options` parameter are checked out dynamically. The use model is as follows:

```
tempus -lic_startup_options "lic1 lic2 ..."
```

Note: If you do not want any product options to be checked out dynamically, use empty quotes with the `-lic_options` parameter, as follows:

```
tempus -lic_options "lic1 lic2 ..."
```

The following command can be used to check out product options after you have invoked the software:

```
set license check -checkout license -optionList "license1 license2 ..."
```

The product option specified with the `-checkoutList` parameter is checked out immediately and the product options specified with the `-optionList` parameter are checked out dynamically. With the `-checkOut` parameter, you can specify only one product option.

Note: If you do not want any product options to be checked out dynamically, use empty quotes with the `-optionList` parameter, as follows:

```
set license check -checkout option -optionList " "
```



You cannot check out a license for a product option if you have not checked out a base license.

Tempus User Guide
Product and Licensing Information

Design Import

- 2.1 [Design Import Overview](#) on page 28
- 2.2 [Input Requirements](#) on page 28
- 2.3 [Design Import Flow](#) on page 30
- 2.4 [Performing Design Sanity Checks in Tempus](#) on page 31
- 2.5 [Tempus File Management](#) on page 32

2.1 Design Import Overview

This topic describes the tasks that you perform to import the data and prepare for analysis in Tempus. The Tempus software supports both MMMC (multi-mode multi-corner) and non-MMMC methods of importing design data. However, Tempus database has MMMC configuration settings by default. Tempus works in logical and physical modes. Physical information is available in the form of LEF/DEF format, Innovus, or Open Access (OA) database.

2.1.1 Input Requirements

Required Design Data

- Timing Libraries (Liberty dotlib files)
- Verilog Netlist
- SDC Constraints
- Parasitic Data (SPEF/RCDB)

Optional Design Data

- Delay Data (SDF file)
- Physical Data

Timing Libraries

Timing library files contain timing information in ASCII format for all standard cells, blocks, and I/O pad cells. Tempus reads timing library format files (.lib). You can read in the library files using the read lib command.

Verilog Netlist

Tempus reads synthesized netlist in Verilog. You can read in the netlist files using the read_verilog command.

SDC Constraints

- To ensure that your design meets the timing requirements, you must first specify the requirements by setting the constraints. You can use the timing constraints to set:

- Timing context for constraint assertions
- Boundary conditions, such as input and output delays
- Slew rates
- Path exceptions, such as false paths, path delays, and cycle additions
- Disable certain paths in the design

You can read the timing constraints using the `read_sdc` command.

Parasitic Data (SPEF/RCDB)

You can specify the parasitic data for more accurate timing analysis using a SPEF or RCDB file.

You can use the `read_spef` command to specify the SPEF file. The `read_parasitics` command is used for reading SPEF and RCDB-based RC parasitics information, and already created RCDBs in Tempus.

Delay Data (SDF file)

This is an optional input requirement.

An SDF file contains details about the absolute delays for all cells and interconnects, including IR drop and crosstalk impact. It annotates user-specified delay information from the SDF file into the timing system. You can use the `read_sdf` command to supply pre-calculated delays and timing check values from an external delay calculator, or from a previous Tempus session.

In Tempus, you can use the `read_sdf` command to read in a SDF file and then the timing analysis engine will use the delays annotated from this SDF file.

Physical Data

This is an optional input requirement.

In Tempus, you can read physical information, such as placement, floorplan, and routing that you created using LEF and DEF in Innovus.

To read the physical data, use one of the following commands:

- The `-physical_data` parameter of the `read_design` command with either configuration file or Innovus file that you generate using Innovus.
- The `read_def` command after reading the LEF files, netlist, and .lib files.

Note: To read LEF files, use the `-lef` parameter of the `read_lib` command.

2.2 Design Import Flow

The design import flow involves the following steps:

- Loading the design
- Saving the design
- Restoring the design

These are summarized in the table below:

Table 2-1 MMMC Design Import Flow

Task	Logical Data	Physical Data	OA Physical Data
Load Design	<code>read_view_definition</code> <code>read_verilog</code> <code>set_top_module</code>	<code>read_lib -lef</code> <code>read_view_definition</code> <code>read_verilog</code> <code>set_top_module</code> <code>read_def</code>	<code>read_lib -oaRef</code> <code>read_view_definition</code> <code>read_verilog</code> <code>set_top_module</code>
Save Design	<code>save_design</code> <code>session</code>	<code>save_design</code> <code>session</code>	<code>save_design -cellview</code>
Restore Design	<code>read_design</code> <code>session topcell</code>	<code>read_design</code> <code>session topcell -physical_data</code>	<code>read_design</code> <code>-physical_data</code> <code>-cellview</code>

The following is an example script to import MMMC data:

```
read_lib -lef <>
read_view_definition <>
read_verilog <>
set_top_module top
read_spf -rc_corner <> <spf file>
read_def <>
```

Tempus User Guide

Design Import

Note: When you use the `read lib -lef`, `read view definition`, or `read verilog` command, the software schedules these files for reading. The files are actually read when you use the `set top module` command.

To load design in the DMMMC mode, refer to the “[Distributed Multi-Mode Multi-Corner Timing Analysis](#)” chapter of the *Tempus User Guide*.

Table 2-2 Non-MMMC Design Import Flow

Task	Logical Data	Physical Data	OA Physical Data
Load Design	<code>read_lib #timing / cdb</code> <code>read_verilog</code> <code>set_top_module</code>	<code>read_lib -lef</code> <code>read_lib #timing / cdb</code> <code>read_verilog</code> <code>set_top_module</code> <code>read_def</code>	<code>read_lib -oaRef</code> <code>read_lib #timing / cdb</code> <code>read_verilog</code> <code>set_top_module</code>
Save Design	<code>save_design session</code>	<code>save_design session</code>	<code>save_design -cellview</code>
Restore Design	<code>read_design session topcell</code>	<code>read_design session</code> <code>topcell -physical_data</code>	<code>read_design</code> <code>-physical_data</code> <code>-cellview</code>

The following is an example script to import Non-MMMC data:

```
read_lib -lef <>
read_lib <>
read_verilog <>
set_top_module <>
read_sdc <>
read_spef <>
read_def <>
```

When the data is imported using non-MMMC commands, Tempus internally stores it in MMMC configuration as the following MMMC objects:

```
default_emulate_view
default_emulate_delay_corner
default_emulate_rc_corner
default_emulate_libset_max
default_emulate_libset_min
default_emulate_constraint_mode
```

2.3 Performing Design Sanity Checks in Tempus

At various stages of the design process, you can check for missing or inconsistent library and design data.

To perform these checks, use the following commands:

- check_design
- check_library
- check_timing

2.4 Tempus File Management

Tempus allows you to manage temporary and auto-generated files. These are described in the sections below:

Temporary Files Management

You can specify the directory for writing out temporary files during a Tempus session.

The Tempus software creates a unique temporary sub-directory for a session, specified by `setenv TMPDIR <dir-name>` at the command prompt, where:

- Default value of TMPDIR is “/tmp”.
- All temporary files will be created under directory:
“\$TMPDIR/tempus_<user>_<pid>_<xxxxx>/”
- If write-permission is not provided in \$TMPDIR, then TMPDIR will internally reset to “/tmp”.
- If the specified TMPDIR does not have enough space, the software will not launch.
- All temporary files/directories will be deleted, once a session ends.
- No temporary file is expected to be created in the run directory.

This support is also available for DMMMC and DSTA flows.

Auto-Generated File Management

You can control the location and name of auto-generated files in a session. These files are created by Tempus automatically and are retained after the run completes. This includes report files, or other output files with default names that are auto-generated in the current run directory. The software allows you to control the directory that the files should be redirected to, and also add unique prefixes to file names. You can use the following global variables to control this behavior:

Tempus User Guide

Design Import

- auto file dir

Represents the top-level directory to store files/sub-directories that are auto-generated by the software.

- auto file prefix

Represents the prefix to be applied to all files/sub-directories that are auto-generated by the software.

Installation and Startup

- 3.1 [Tempus Product and Installation Information](#) on page 35
- 3.2 [Setting Up the Tempus Run Time Environment](#) on page 35
- 3.3 [Launching the Tempus Console](#) on page 36
- 3.4 [Completing Command Names](#) on page 37
- 3.5 [Command-Line Editing](#) on page 37
- 3.6 [Setting Preferences](#) on page 39
- 3.7 [Starting the Tempus Software](#) on page 39
- 3.8 [Accessing Tempus Documentation and Help](#) on page 41
- 3.9 [Accessing Documentation and Help from Tempus GUI](#) on page 41
- 3.10 [Using the Tempus man and help Commands on the Command-Line](#) on page 43
- 3.11 [Other Sources of Cadence Product Information](#) on page 44

3.1 Tempus Product and Installation Information

For product, release, and installation information, see the `README` file at any of the following locations:

- <http://downloads.cadence.com>
- In the software installation, where it is also available when you are using or running the software

For information about the Tempus licenses, see “[Product and Licensing Information](#)” chapter.

3.2 Setting Up the Tempus Run Time Environment

If `install_dir` is the location of the Tempus installation, then you should setup the run time environment as follows:

- Add the `install_dir/bin` directory to the path. The `bin` directory has links to all the public executable files in the install hierarchy.
- If you want the Tempus man pages to be available with the Unix `man` command, then you can add the `install_dir/share/tempus/man` to the `MANPATH` envvar.
- If you want the Stylus Common UI Tempus man pages to be available with the Unix `man` command, then you can add `install_dir/share/tempus/stylus_man` to the `MANPATH` envvar.
- If you want the Tcl man pages to be available with the Unix `man` command, then you can add the `install_dir/share/tcltools/man` to the `MANPATH` envvar.

For example, you can add the following to the startup shell script:

```
set install_dir = /tools/tempus16.2/lnx86
set path = ($install_dir/bin $path)
setenv MANPATH $install_dir/share/tempus/man:$install_dir/share/tcltools/
man:$MANPATH
```

Note: When Tempus launches, it automatically adds the legacy or the Stylus CUI man pages (if the `-stylus` parameter is specified), and the Tcl man pages at the beginning of the current `MANPATH` inside Tempus. Therefore, from within Tempus the `man` command will show both the sets of man pages before any other man pages.

3.2.1 Temporary File Locations

Each Tempus session creates its own temporary directory to store temporary files at the beginning of the run.

By default, the `tmp_dir` is created in the `/tmp` directory. If the Unix envvar `TMPDIR` is set, then the `tmp_dir` is created inside `$TMPDIR`.

The name of the `tmp_dir` will appear as given below:

`tempus_temp_[pid]_[hostname]_[user]_xxxxxx`

Where `_xxxxxx` is a string that is added to make the directory unique. For example:

`tempus_temp_10233_farm254_bob_nfp9ez`

The temporary directory is automatically removed on exit or if the run is terminated with a catchable signal (for example, `SIGSEGV`).

Supported and Compatible Platforms

The `README` file lists the supported and compatible platforms for this release.

3.3 Launching the Tempus Console

When you start Tempus in the non-GUI environment using the `-no_gui` parameter, the UNIX window (shell tool, xterm, and so on) where you start the Tempus session is called the Tempus console. This is where you enter all Tempus text commands and where the software displays messages. When a session is active, this console displays the following prompt:

`tempus>`

When you start the Tempus in the GUI environment, the Tempus console is displayed within the main Tempus window.

If you use this console for other actions, for example, to use the vi editor, the session suspends until you finish the action.

If you suspend the session by typing `Control-z`, the `tempus>` prompt is no longer displayed. To return to the Tempus session, type `fg`, which brings the session to the foreground.

3.3.1 Completing Command Names

Use the `Tab` key within the software console to complete text command names.

After you type a partial text command name and press the `Tab` key, the software displays the exact command name that completes or matches the text you typed (if the string is unique to one text command) or all the commands that match the text you typed.

For example, if you type the following text and press the `Tab` key

```
report_ti
```

The software displays the following command:

```
report_timing
```

If you type the following text and press the `Tab` key

```
report_t
```

The software displays the following commands:

```
report_table
report_tclist
report_thermal
report_thresh
report_timing
report_timing_derate
report_timing_lib
```

3.3.2 Command-Line Editing

The Tempus software provides a GNU Emacs-like editing interface. You can edit a line before it is sent to the calling program by typing control characters. A control character, shown below as a caret followed by a letter, is typed by holding down the `Control` key when typing the character.

Most editing commands can be given a repeat count, n , where n is a number. To enter a repeat count, press the `Esc` key, the number, and then the command to execute. For example, `Esc 4 ^f` moves forward four characters. If a command can be given a repeat count, the text `[n]` is shown at the end of its description.

You can type an editing command anywhere on the line, not just at the beginning. You can press `Return` anywhere on the line, not just at the end.

Note: Editing commands are case sensitive: `Esc F` is not the same as `Esc f`.

Table 3-1 Control (^) Characters

^A	Move to the beginning of the line
^B	Move left (backwards) [n]
^C	Exits from editing mode, returning the console to normal Tempus System mode
^D	Delete character [n]
^E	Move to end of line
^F	Move right (forwards) [n]
^G	Ring the bell
^H	Delete character before cursor (backspace key) [n]
^I	Complete filename (Tab key); see below
^J	Done with line (Return key)
^K	Kill to end of line (or column [n])
^L	Redisplay line
^M	Done with line (alternate Return key)
^N	Get next line from history [n]
^P	Get previous line from history [n]
^R	Search backward (forward if [n]) through history for text; must start line if text begins with an up arrow
^T	Transpose characters
^V	Insert next character, even if it is an edit command
^W	Wipe to the mark
^X^X	Exchange current location and mark
^Y	Yank back last killed text
^ [Start an escape sequence (Esc key)
^] c	Move forward to next character c
^?	Delete character before cursor (Delete key) [n]

3.3.3 Setting Preferences

You set preferences at the beginning of a new design import. You can assign special characters for the design import parser for Verilog®, DEF, and PDEF files, and control the display of the Physical view window. You can also change the hierarchical delimiter character in the netlist before importing the design, and change the DEF hierarchical default character and the PDEF bus default delimiter before loading the file.

Note: If you change the default values for the DEF delimiter or PDEF bus delimiter, these changes become the default delimiters for the DEF and PDEF writers.

For information on setting design preferences, see *Tools - Preferences* in the *Tempus Menu Reference*.

Initialization Files

By default, various initialization files are loaded up at startup, if they exist. They can be used to configure the GUI, load utility Tcl files, or configure Tempus settings.

For a list of the files and the order in which they are loaded, refer to [tempus](#) in the *Tempus Text Command Reference*.

3.4 Starting the Tempus Software

To start a Tempus session, type the following command with the appropriate parameters on the UNIX/Linux command line.

```
tempus
```

For information on using this command, see [tempus](#) in the *Tempus Text Command Reference*.

This command starts one of the following products:

- Tempus L
- Tempus XL

For an overview of the products and product licensing, see "[Product and Licensing Information](#)".

3.4.1 Using the Log File Viewer

The following methods are available to view the log file:

- [Integrated Log File Viewer](#) on page 40
- [Standalone Log File Viewer](#) on page 40

Integrated Log File Viewer

You can use the integrated log file viewer when the software is running. It has the following features:

- Ability to expand and collapse command information.
- Ability to view multiple log files in separate console windows simultaneously.
- Color coding of error, warning, and information messages.
- Access to the documentation in the *Tempus Text Command Reference*. When a log file is displayed, click on any of the underlined commands to open an HTML window that displays the documentation for that command.

Use one of the following methods to use the viewer:

→ Select *Tools - Log Viewer* on the main menu.

The *Log File* window is displayed. Select the log file to view. The software opens a separate console window and displays the log file.

For more information, see Tools - Log Viewer in the "Tools Menu" chapter of the *Tempus Menu Reference*.

Standalone Log File Viewer

You can use the standalone viewer even if the software is not running. It provides most of the same functionality as the viewer that is run within the software but does not provide access to the documentation.

To use the standalone viewer, type the following UNIX/Linux command in the console window:

```
viewlog [-file logFileName]
```

The viewer opens the most recently created log file unless you specify a different file with the *-file* parameter.

3.5 Accessing Tempus Documentation and Help

You can access the Tempus documentation and help system by using the following methods:

- [Launching Cadence Help From the Command Prompt](#) on page 41
- [Accessing Documentation and Help from Tempus GUI](#) on page 41
- [Using the Tempus man and help Commands on the Command-Line](#) on page 43
- [Other Sources of Cadence Product Information](#) on page 44

Launching Cadence Help From the Command Prompt

1. Change to the following directory:

```
installation_dir/tools/bin
```

2. Enter the following command:

```
./cdnshelp
```

After launching Cadence® Help, press F1 or choose *Help - Contents* to display the help page for Cadence Help.

For more information see the Cadence Help documentation.

3.6 Accessing Documentation and Help from Tempus GUI

The Tempus software provides the following two methods to access documentation and help from the GUI:

- [Select Help on the Main Tempus Menu](#) on page 41
- [Select Help from a Tempus Form](#) on page 42

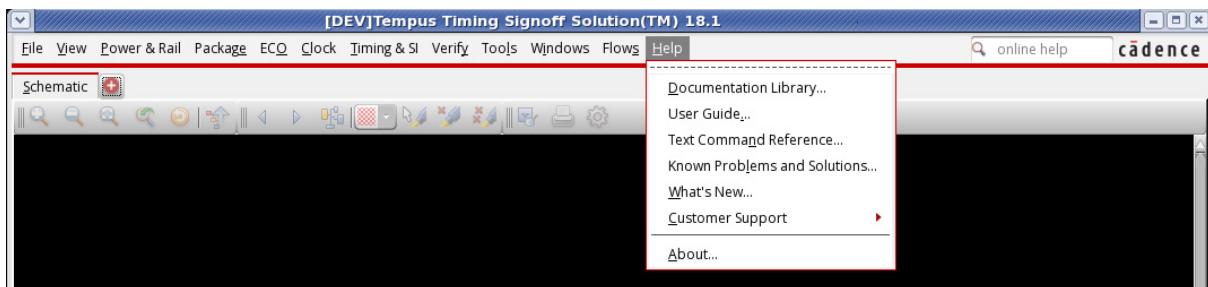
Select Help on the Main Tempus Menu

Follow the steps given below:

Tempus User Guide

Installation and Startup

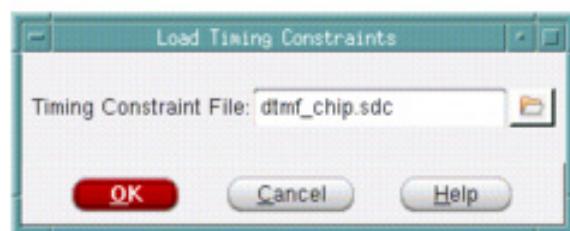
Figure 3-1 Help Menu



- Select *Help*, and click any of the following options:
- *Text Command Reference*
Opens the Table of Contents page of the text command reference.
 - *User Guide*
Opens the Table of Contents page of the user guide.
 - *Known Problems and Solutions*
Opens the Table of Contents page of the known problems and solutions document.

Select Help from a Tempus Form

- Click the *Help* button in the bottom right corner of a form.



Clicking the *Help* button opens the entry for the form in the Cadence Help window.

3.7 Using the Tempus man and help Commands on the Command-Line

Using the help Command to View the Command Syntax

- To see syntax information for a command, type the following command in the software console:

```
help command_name
```

For example, to see syntax information for the `read_lib` command, type the following command:

```
help read_lib
```

The software displays the following text:

```
Usage: read_lib <dotlib_file> [-min <min_lib_list>]  
[-max <max_lib_list>] [-cdb <cdb_list>]  
[-lef <file_names>] [-pgv <power_grid_list>]
```

- To see the entire list of Tempus commands and their syntax, type the following command in the software console:

```
help
```

Using the man Command to View the Command Description

To see the complete set of information for a Tempus command, type the following command in the software console:

```
man command_name
```

For example, to see the complete set of information for the `read_lib` command, type the following command:

```
man read_lib
```

Using the help Command to View Message Summary

To see the message summary of a particular message ID, type the following command in the software console:

```
help msg_id
```

For example, to see the message summary for the TECHLIB-1002 message ID, type the following command:

```
help TECHLIB-1002
```

Using the man Command to View Message Detail

To see the message detail of a particular message ID, type the following command in the software console:

```
man msg_id
```

For example, to see the message summary for the SI-2253 message ID, type the following command:

```
man SI-2253
```

The detailed description is not available for all active message IDs.

3.8 Other Sources of Cadence Product Information

You can also get help on Cadence products by selecting *Customer Support* on the *Help* menu. The *Customer Support* sub menu provides access to the following Cadence resources:

- Cadence Online Support
Opens Cadence Online Support in your browser.
- Web Collaboration
Opens your default browser and explains the current methodology followed by Cadence to support web based collaboration.

Analysis and Reporting

- 4.1 [Base Delay Analysis](#) on page 46
- 4.2 [Signal Integrity Delay and Glitch Analysis](#) on page 60
- 4.3 [Timing Analysis Modes](#) on page 118
 - 4.3.1 [Advanced On-Chip Variation \(AOCV\) Analysis](#) on page 138
 - 4.3.2 [Statistical On-Chip Variation \(SOCV\) Analysis](#) on page 165
- 4.4 [Path-Based Analysis](#) on page 182
- 4.5 [Analysis Of Simultaneous Switching Inputs - SSI](#) on page 189
- 4.6 [Waveform Aware and Rail Swing Checks](#) on page 195
- 4.7 [Set Instance Library Flow](#) on page 200

4.1 Base Delay Analysis

- 4.1.1 [Overview](#) on page 47
- 4.1.2 [Base Delay Analysis Flow](#) on page 47
- 4.1.3 [Limitations of Traditional Delay Calculators](#) on page 49
- 4.1.4 [Performing Base Delay Analysis](#) on page 51
- 4.1.5 [Base Delay Reporting](#) on page 56
- 4.1.6 [Cycle-to-Cycle Jitter using Native Delay Calculation](#) on page 58

4.1.1 Overview

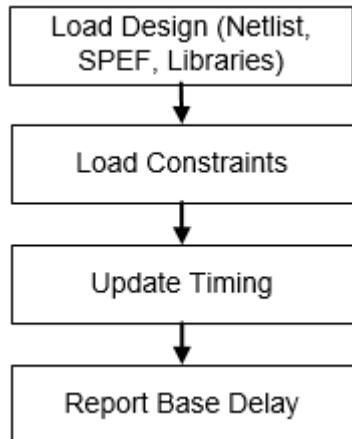
Tempus enables you to perform fast and precise signal integrity-aware delay calculations for cell-based designs. The software combines signal integrity (SI) analysis with timing analysis to check for functional failures due to SI glitch and performs accurate timing calculations (that account for both the SI and IR-drop effects). Tempus utilizes the multi-threaded circuit simulation methods to deliver accuracy, capacity, and performance needed for nano meter designs.

This chapter describes the delay analysis flow and reporting.

4.1.2 Base Delay Analysis Flow

The base delay analysis flow is described below:

Figure 4-1 Base Delay Analysis Flow



Sample Base Delay Calculation Script

You can use the following script to calculate base delay:

1. Specify the design size:

```
set_design_mode -process 28
```

2. Load the design data. Follow the steps given below.

- Load the timing libraries (.lib):

```
read_lib
```

- Schedule reading of structural gate-level verilog netlist:

```
read_verilog dma_mac.v
```

- ❑ Set the top module name, and check for consistency between the Verilog netlist and timing libraries:

```
set_top_module dma_mac
```

- ❑ Load the timing constraints:

```
read_sdc constraints.sdc
```

- ❑ Load the cell parasitics:

```
read_spef cell.spef.gz
```

3. Generate the timing report:

```
report_timing
```

4. Report base delay calculation details for the arcs:

```
report_delay_calculation -from inst1/A -to inst1/Y
```

Base Delay Analysis - Inputs and Outputs

Inputs

- Timing Library (CCS or ECSM preferred): Contains the timing data.
- Netlist: Specifies a circuit netlist in Verilog.
- SPEF: Specifies the RC parasitics information in SPEF format.
- Command File (Tcl): Contains a series of Tcl commands.
- Noise Library (CCS-N, ECSM-SI, or cdB) (Optional input): Contains characterized noise data.
- Timing Constraints: Specifies the timing constraints, including clock propagation, in a SDC file.

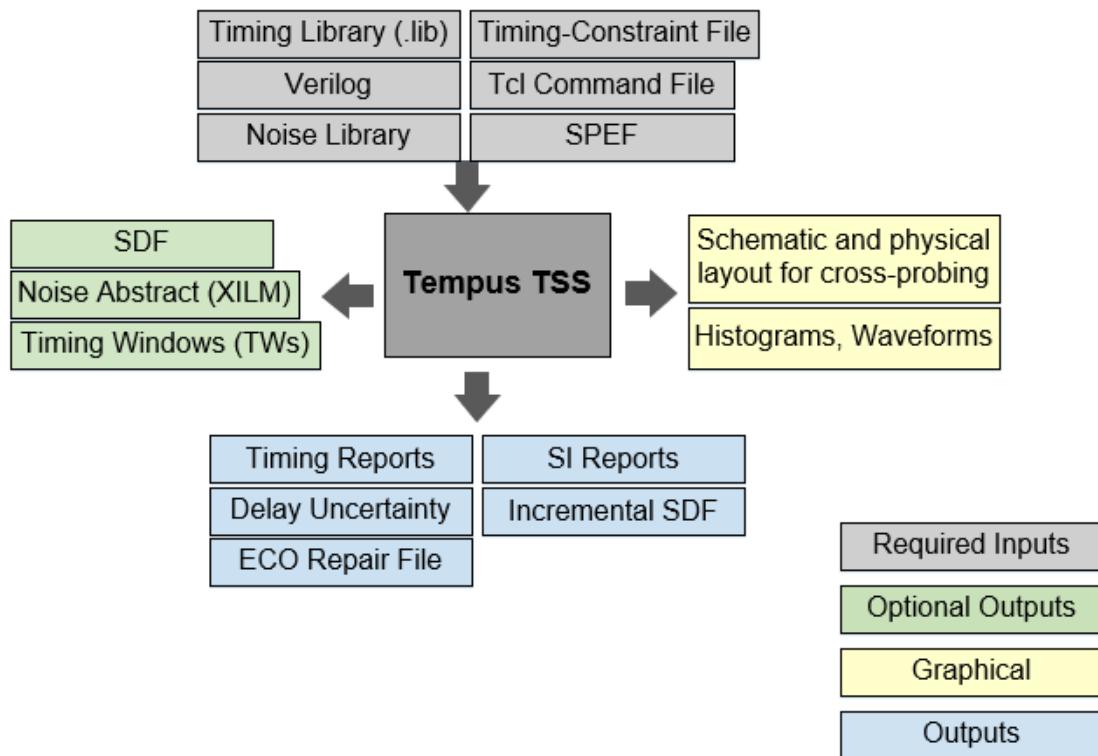
Outputs

- Timing Reports: Contains details about the critical paths in the design.
- Delay Report: Contains details about delay changes - with or without crosstalk.
- SDF File: Contains the negative and positive delay changes caused by crosstalk. The full SDF file contains details about the absolute delays for all cells and interconnects, including crosstalk impact.

- Noise Abstracts: Specifies an XILM abstract noise model. An XILM model is a detailed noise abstracts that can be used to perform hierarchical timing and noise analysis. This model contains cells that belong to block interface logic, and internal cells/nets that are coupled to the interface logic.
- Noise Report: Contains details about the glitch noise on nodes.

The inputs and outputs are shown in the diagram below.

Figure 4-2 Inputs and Outputs required for Base and SI Analysis

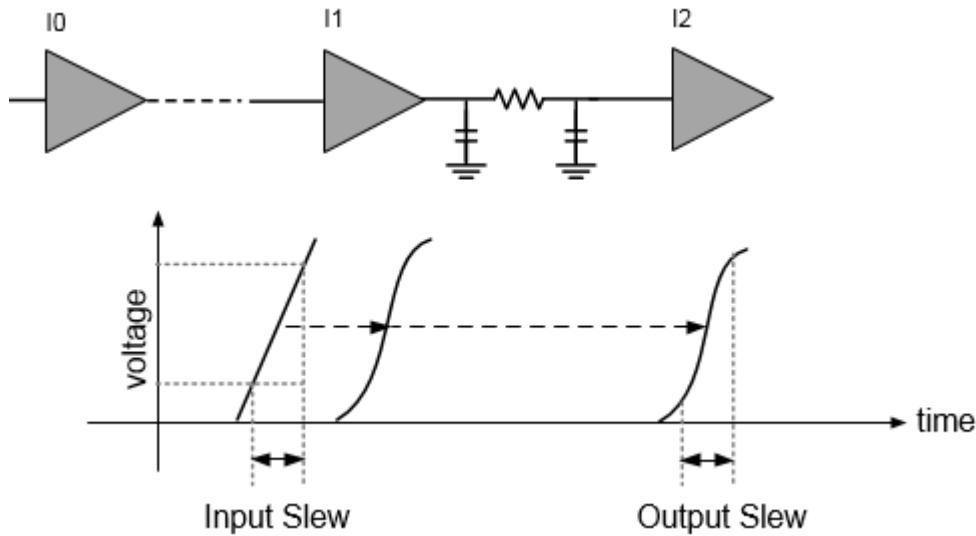


4.1.3 Limitations of Traditional Delay Calculators

Traditional delay calculators use delay as a function of the input slew and output load. With traditional delay calculators, a single linear slew value is used as the input to analyze a stage. This methodology cannot produce the desired accuracy that new technologies demand.

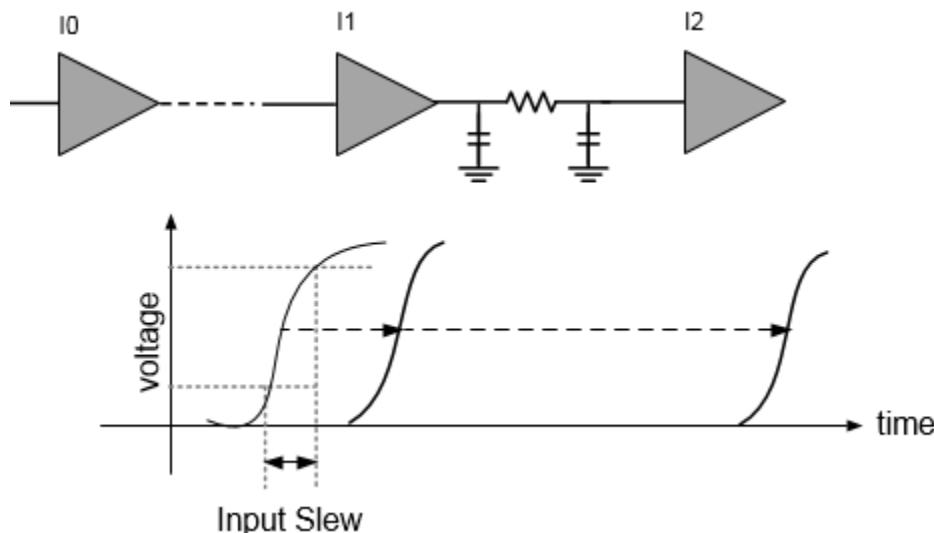
This is illustrated in the figure below.

Figure 4-3 Traditional Delay Calculation using Delay as function of slew and load



The advanced technologies (28nm and below) require waveform-based delay calculators to accurately calculate the delays based on waveforms. The waveform-based delay calculators use real waveforms as input to analyze a stage, as shown in the figure below.

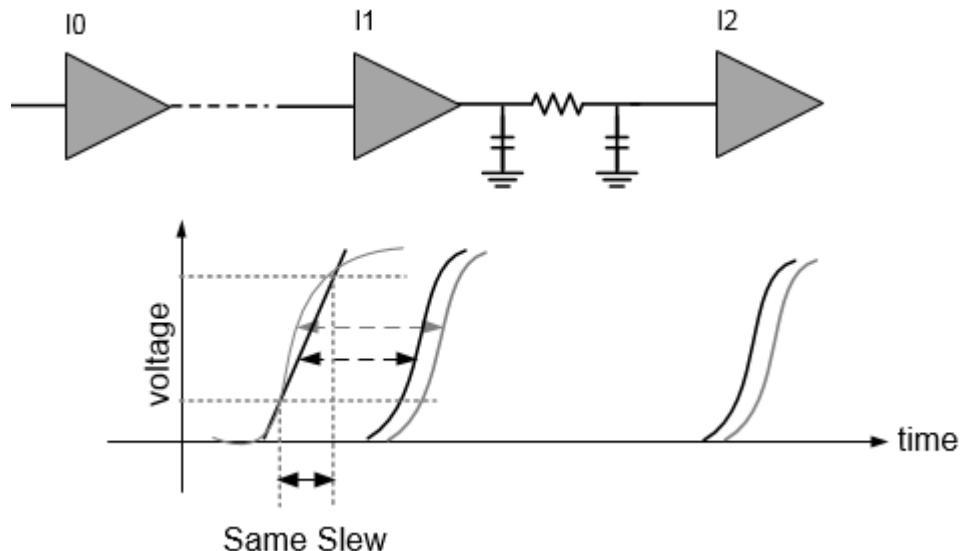
Figure 4-4 Ideal Waveform based Delay Calculation



There are several shortcomings when you use traditional delay calculators. Some of these are:

- Traditional delay calculators use single slew to calculate stage delays. This may not produce accurate results.
- Different waveforms with the same slew value produce the same delays.

Figure 4-5 Shortcomings of Traditional Delay Calculation



In the above figure, the grey waveforms indicate the input waveforms in an ideal case. The black lines indicate the linear input slew values and slew waveforms, respectively.

Here, the linear input slew value and the input waveforms use the same slew, however, the delay numbers are different. The long curve of the input waveform (grey) can produce larger delays as compared to the one produced with the linear input slew (black). Also, the measurement point shift can contribute to delay inaccuracy.

4.1.4 Performing Base Delay Analysis

To overcome the shortcomings of Traditional Delay Calculators, Tempus provides two different approaches for performing delay calculations. These are described below:

- [Equivalent Waveform Model \(EWM\)](#) on page 52
- [Waveform Propagation](#) on page 53

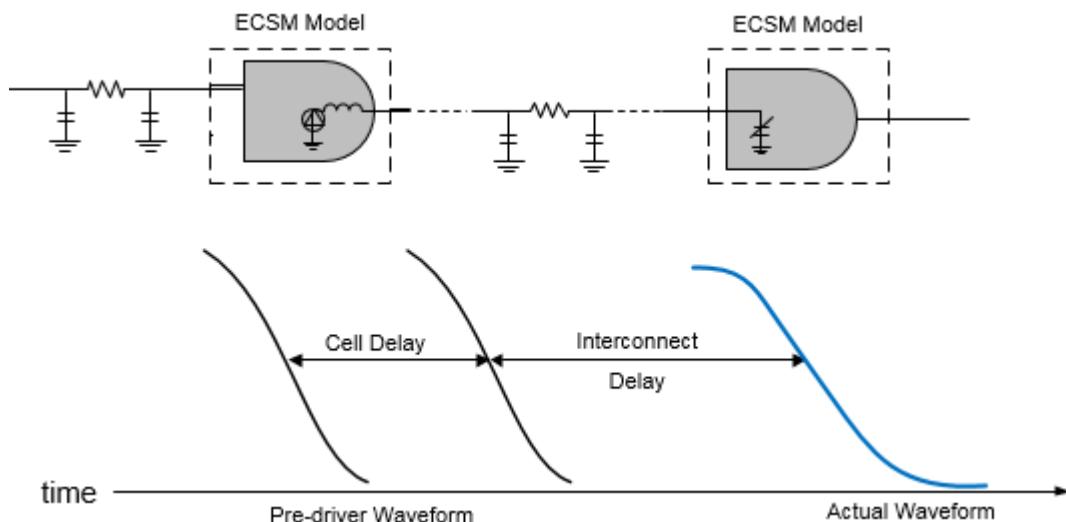
Equivalent Waveform Model (EWM)

To achieve accuracy, the waveform shapes are required to be included during delay calculations. The equivalent waveform model (EWM) computes equivalent receiver output based on the input waveform shapes and adjusts the interconnect delay accordingly. The adjustment in delay compensates for any inaccuracies that delay calculation might cause in the next stage due to lack of waveform shape information. The EWM approach provides a technique for producing higher accuracy results when compared to Spice.

When a stage is analyzed during delay calculation, a pre-defined waveform from the library (based on single input slew value) is used as a stimulus. When the EWM mode is not enabled, the input slew is measured from the actual waveform computed during the previous stage analysis. This may have entirely different characteristics compared to the pre-defined waveform used in the current stage. As a result, the delay impact due to waveform shape differences may be affected.

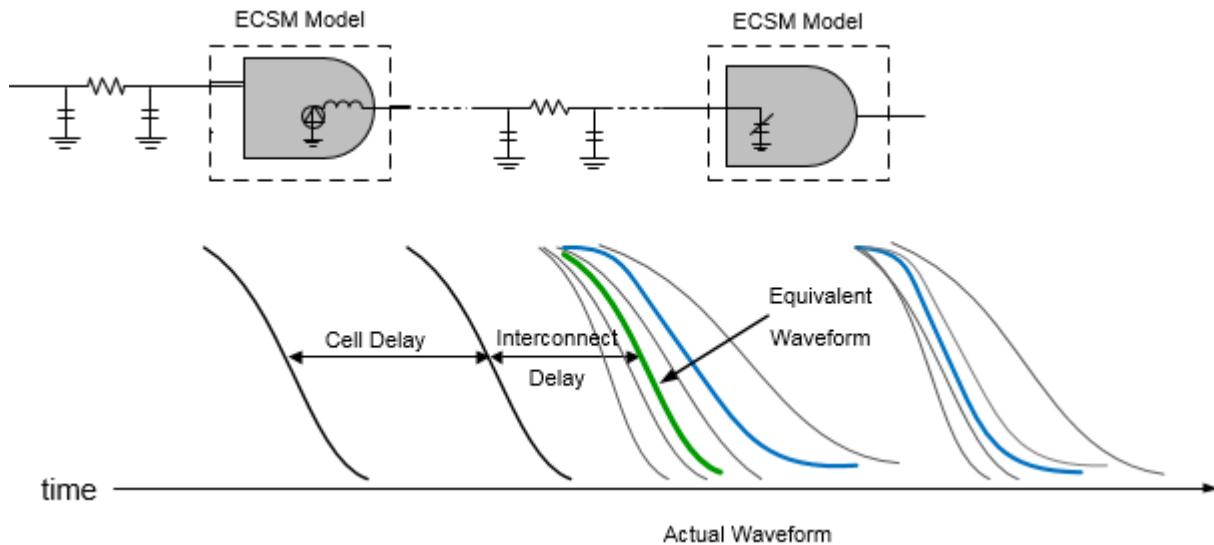
The following figure illustrates delay calculation without EWM.

Figure 4-6 Delay Calculation without EWM



The following figure illustrates delay calculation with EWM.

Figure 4-7 Delay Calculation with EWM



When EWM is enabled, the software computes the delay impact of waveform shapes on receiver cells, and computes the delay impact - thus providing an overall improvement in path delay accuracy. When EWM is enabled in SI analysis, the software provides delay adjustments based on the receiver noise response to a noisy transition. This helps to reduce the SI pessimism that will be reported if the total delay is measured on noisy waveforms at the receiver input.

Use Model

Equivalent waveform model can be enabled by using the following command:

```
set delay cal mode -equivalent waveform model no_propagation
```

Waveform Propagation

The waveform shapes have a significant impact on delay calculations. Traditionally, delay calculation uses a single pre-driver waveform for specific slew value at the cell input to compute the response on the output of a cell.

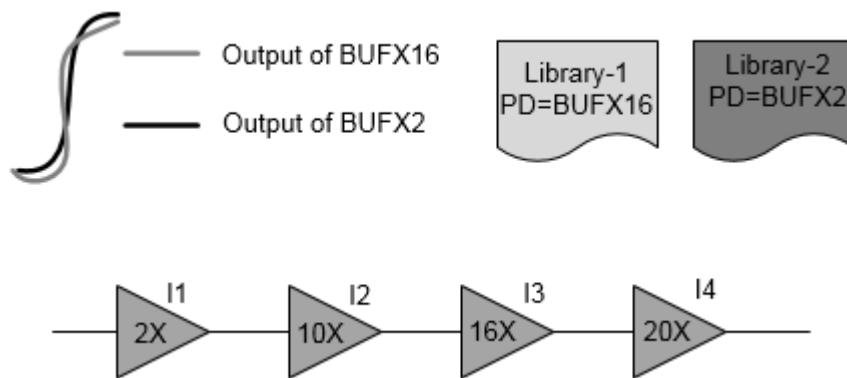
Consider two libraries characterized with pre-driver cells, BUFX16 and BUFX2. An accurate delay on all the instances driven by BUFX16 will be reported, when the library uses BUFX16 as a pre-driver cell. In this case, the instance I4 produces more accurate results as the input waveform matches with the pre-driver waveform. The accuracy of other cells is impacted due to a difference in results for the pre-driver cell versus the actual driving cell. To facilitate actual waveform propagation through a path, the waveform propagation feature stores the actual waveform instead of a single slew value at the input of each cell.

As shown in the figure below, both the BUFX16 slew (in grey) and BUFX2 slew (in black) have the same value but their waveform shapes are different. The same slew having different waveform shapes can produce different delays. The delay accuracy is a function of input waveform shapes, even if the slew values are same. If the input waveform shape is different from the input waveform used for cell characterization, the delay accuracy will be significantly affected.

The waveform propagation feature is supported in both graph- and path-based analysis modes.

The following figure illustrates waveform impact on delay.

Figure 4-8 Waveform impact on delay



Requirements for Waveform Propagation

The waveform propagation using ECSM models will be more accurate with additional receiver pincap points. Cadence recommends at least eight points. The last three points can be the lower slew threshold, delay measurement point, and upper slew threshold. The rest of the five points must be selected to represent the tail waveform accurately. It is also recommended to have actual pre-driver waveform in the ECSM libraries.

Use Model

Waveform propagation can be enabled by using the following command:

```
set delay cal mode -ewm_type moments
```

EWM-Only vs Waveform Propagation

EWM-Only

- Real waveform tail impact on the next stage is predicted and added to the current wire delay.
- The receiver cell is assumed to be the driver lumped load.

Waveform Propagation

- Real waveforms are stored and used as input for the next stage. The input waveform tail impact is used at the appropriate point.
- Unlike EWM-Only, the waveform propagation computes accurate impact of the tail as it uses distributed parasitics of wires.

EWM and Waveform Propagation - Impact on the Flow

The base delay flow is impacted in the following ways, when you use the equivalent waveform model or waveform propagation methodology:

- Enabling equivalent waveform modeling increases runtime by 10% - with no significant increase in the memory.
- Enabling waveform propagation increases runtime by 15%. There is an 8% increase in the memory in graph-based analysis (GBA) mode, and no significant memory changes take place in the path-based analysis (PBA) mode.

Normalized Driver Waveform in Library

The normalized driver waveform (pre-driver waveform) should be added to a library while characterizing so that delay can be computed accurately (as there can be different waveforms with the same slew). There are two types of pre-drivers – active and analytical. Cadence recommends that you use the active pre-driver. The active pre-driver waveform has a longer tail than the analytical pre-driver, and thus represents the real design scenario. In the absence of NDWs (normalized driver waveform) in a library, Tempus auto-generates analytic pre-driver waveforms.

An example of a normalized driver waveform in a library is given below:

```
normalized_driver_waveform (waveform_template_name) {  
    driver_waveform_name : "PreDriver20.25:rise";  
    index_1 ("0.00233, 0.01301, 0.05092, 0.1233, 0.2361, 0.3943, 0.6026");  
    index_2 ("0, 0.083, 0.166, 0.25, 0.333, 0.417, 0.5, 0.583, 0.666, 0.75, 0.833,  
0.917,1");  
    values ( \  
        "0, 0.00037, 0.0005041, 0.0006424, 0.0008009, 0.0009783, 0.001179, 0.00140,  
0.00166, 0.001995 0.002375, 0.002833, 0.003389", \  
        "0, 0.00220 0.0029643 0.003777, 0.004709, 0.005752, 0.006935, 0.00828374,  
0.00988784, 0.0117336, 0.013969, 0.0166759, 0.0199283", \  
        "0, 0.00862494, 0.0116002, 0.01473, 0.0184297, 0.0225117, 0.0271421,
```

```
0.0324164, 0.0386937, 0.0459165, 0.05466, 0.06571, 0.0779846", \
"0, 0.0208867, 0.0280917, 0.0357977, 0.0446304, 0.0545157, 0.065729,
0.0785015, 0.0937029, 0.111194, 0.132378, 0.15803, 0.188852", \
"0, 0.0399897, 0.0537844, 0.0685384, 0.0854496, 0.104376, 0.125845,
0.150299, 0.179404, 0.212893, 0.253452, 0.302566, 0.361577", \
```

Timing Library Requirement for Accurate Analysis for 16nm and Below

The ECSM/CCS library characterization for 16nm static timing analysis (STA) signoff meets the challenges of accurate timing analysis in 20nm. In order to meet accuracy demands for process nodes 16nm and below, the following are recommended:

- 8-piece pin capacitances in ECSM timing libraries
- 2-piece pin capacitance in CCS Timing libraries
- N -piece pin capacitance in CCS Timing libraries
- 2% - 98% ECSM waveform range

ECSM Libraries with 8-Piece Pin Capacitances

The 8-piece pin capacitance in the ECSM timing libraries are required to accurately model back miller current. Traditionally, the receiver pin capacitance in an ECSM library characterization is measured at the slew thresholds - that may be 30% to 70% of the VDD. As a result, the use of such thresholds in the ECSM libraries may result in some missing important data at the tail of the waveform. The 3-piece capacitance tables are extended to 8-piece tables for 20nm nodes to better capture the waveform distortions due to back miller current at the waveform tail. The selection of 8-piece pin capacitance is made such that the required 20nm waveform distortion information can be captured correctly. Since the waveform distortion happens mostly at the tail of waveforms, the pin-cap thresholds are selected so that there are more points on the tail.

4.1.5 Base Delay Reporting

The following commands can be used to generate reports on base delay:

- report_timing
- report_delay_calculation

Using the report_timing Command

You can use the `report_timing` command to report the base delays on a timing path. It is recommended to use the `report_timing -net` parameter to produce a comprehensive report.

```
tempus > set_global report_timing_format {hpin cell slew delay arrival}
tempus > report_timing -net
Path 1: VIOLATED Setup Check with Pin seg3/u9/CK
Endpoint: seg3/u9/D (v) checked with leading edge of 'CLK_W_3'
Beginpoint: seg3/u3/Q (v) triggered by leading edge of 'CLK_W_3'
Path Groups: {CLK_W_3}
Other End Arrival Time 1.104
- Setup 0.152
+ Phase Shift 2.000
- Uncertainty 0.050
= Required Time 2.902
- Arrival Time 3.151
= Slack Time -0.249
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 1.135
= Beginpoint Arrival Time 1.135
-----
Pin          Cell      Slew     Delay    Arrival
                           Time
-----
seg3/u3/CK      -        0.091    -        1.135
seg3/u3/Q ->    DFF      0.318    0.303    1.438
seg3/u4/A      BUF      0.318    0.008    1.446
seg3/u4/Y      BUF      0.003    0.158    1.604
seg3/u5/A      INV      0.005    0.003    1.607
seg3/u5/Y      INV      0.499    0.140    1.748
seg3/u6/A      INV      0.549    0.152    1.900
seg3/u6/Y      INV      0.793    0.528    2.427
seg3/u7/A      BUF      0.794    0.003    2.430
seg3/u7/Y      BUF      0.596    0.404    2.835
seg3/u7_a/A    BUF      0.596    0.060    2.895
seg3/u7_a/Y    BUF      0.003    0.250    3.145
seg3/u87A     BUF      0.003    0.001    3.146
seg3/u8/Y      BUF      0.042   -0.023    3.123
seg3/u9/D      DFF      0.072    0.029    3.151
```

Using the report_delay_calculation Command

You can use the `report_delay_calculation` command to report the delay calculation information for a cell or net timing arc.

```
tempus > report_delay_calculation -from seg3/u5/Y -to seg3/u6/A
```

```
From pin      : seg3/u5/Y
Cell          : INV
Library       : cell_w
To pin        : seg3/u6/A
```

```
Cell : INV
Library : cell_w
Delay type: net
-----
RC Summary for net seg3/n5
-----
Number of capacitance      : 17
Net capacitance            : 0.293534 pF
Total rise capacitance    : 0.320849 pF
Total fall capacitance    : 0.320780 pF
Number of resistance       : 17
Total resistance           : 567.671387 Ohm
-----
          Rise          Fall
-----
Net delay                  : 0.151900 ns  0.119000 ns
From pin transition time  : 0.499000 ns  0.211500 ns
To pin transition time    : 0.549100 ns  0.248000 ns
-----
```

4.1.6 Cycle-to-Cycle Jitter using Native Delay Calculation

The IR drop variations between two consecutive cycles can effectively reduce the available clock period for data to be correctly captured. With lower technology nodes, the IR drop variations can have a significant impact because the optimistically applied clock jitter can cause chip failures. Also, a pessimistically applied clock jitter can cause over-fixing and may increase time-to-market. These factors make it necessary to compute the IR drop induced cycle-to-cycle jitter more accurately, because the existing SPICE-based solutions can be highly run time intensive.

The clock jitter flow provides a capability to calculate the cycle-to-cycle jitter due to the IR impact on a clock network. The Tempus software runs an optimized static timing analysis using the (already generated) EIV values (effective instance voltage) for each of the cycles to compute the clock latencies of all end points corresponding to every cycle. The end point cycle-to-cycle jitter is calculated using the latencies from all the cycles.

The SPICE validation feature is available that allows validation of Tempus computed jitter against the SPICE simulated value for a given set of endpoints.

The IR drop induced jitter can be modeled as a component of setup uncertainty or by using the set clock latency -jitter command.

To run clock jitter analysis, you can use the analyze_jitter command.

The following command generates a jitter output report, which reports the clock jitter summary and details of the end points cycle-to-cycle jitter:

```
analyze_jitter -eiv_file VDD_VSS.win_avg.rpt -native true
-----
```

Tempus User Guide

Analysis and Reporting

JITTER SUMMARY

Clock Name	Cycle-To-Cycle Jitter (Type)	Total End Points
clk1	-9.99e-11 - 8.89e-11 (R)	2282
clk1	-9.87e-11 - 8.91e-11 (F)	2282
clk2	-9.93e-11 - 1.19e-10 (R)	2076
clk2	-9.39e-11 - 1.23e-10 (F)	2076

JITTER PER ENDPOINT

Clock	End Points	Adj Cycles for Max -ve/+ve	Cycle-To-Cycle Jitter Change in Jitter
clk1	FF1/CP (R	(2, 3) - (1, 2)	-6.91e-11 - 8.89e-11
clk1	FF1/CP (F	(1, 2) - (4, 5)	-6.87e-11 - 8.88e-11
clk1	FF2/CP	(2, 3) - (1, 2)	-6.81e-11 - 8.79e-11
clk1	FF2/CP (F	(1, 2) - (4, 5)	-6.77e-11 - 8.78e-11
...			

4.2 Signal Integrity Delay and Glitch Analysis

- 4.2.1 [Overview](#) on page 61
- 4.2.2 [Understanding Attacker and Victim Nets](#) on page 61
- 4.2.3 [SI Delay Analysis - An Overview](#) on page 61
- 4.2.4 [Signal Integrity Delay Analysis Flow](#) on page 63
- 4.2.5 [Performing Signal Integrity Delay Analysis](#) on page 64
- 4.2.6 [SI Delay Reporting](#) on page 89
- 4.2.7 [SI Glitch Analysis - An Overview](#) on page 95
- 4.2.8 [SI Glitch Analysis Flow](#) on page 97
- 4.2.9 [Understanding SI Glitch Analysis](#) on page 98
- 4.2.10 [Running Detailed SI Glitch Analysis](#) on page 104
- 4.2.11 [SI Glitch Reporting](#) on page 105
- 4.2.12 [Overshoot-Undershoot Glitch Analysis - An Overview](#) on page 112
- 4.2.13 [Running Overshoot-Undershoot Glitch Analysis - Sample Script](#) on page 116

4.2.1 Overview

Signal Integrity (SI) can impact the delay or introduce a glitch on a given net due to the switching of nets that may be lying in close proximity. Tempus performs SI analysis by analyzing the delay and slew changes of each switching signal in the presence of crosstalk noise. The delay and slew changes, due to crosstalk noise, are then used to determine the worst-case minimum or worst-case maximum path delays in the design. The software also checks for functional failures due to crosstalk glitch noise and reports the failing nets.

Tempus performs the following signal integrity analysis operations:

- [SI Delay Analysis - An Overview](#)
- [SI Glitch Analysis Flow](#)

To perform signal integrity analysis, it is important to understand the role of attackers and victim nets. These are discussed in the following section.

4.2.2 Understanding Attacker and Victim Nets

A victim net receives undesirable cross-coupling effects from any neighboring net. An attacker is a net that causes undesirable cross-coupling effects on a victim net. An **attacker** net can be a victim net and a victim net can also be an attacker net. The net which is analyzed for SI effects is called a **victim net**.

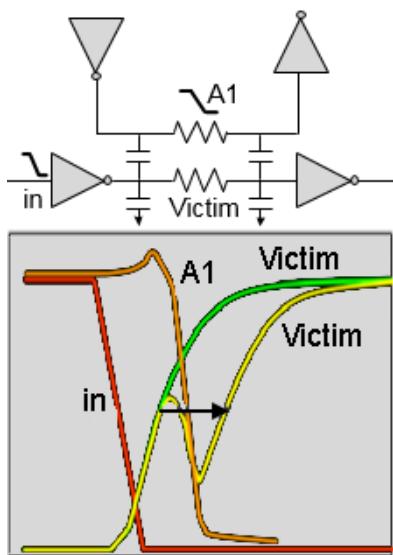
- The impact of an attacker net on a victim net depends on several factors:
- The amount of cross-coupled capacitance
- The relative time and the transition rate of the signal transitions
- The switching directions (rising, falling)
- The combination of effects from multiple attacker nets on a single victim net

4.2.3 SI Delay Analysis - An Overview

SI can increase or decrease signal delay, which can, in turn cause setup or hold failures.

Consider that attacker A1 switches in the opposite direction to the victim, as shown in the figure below, then there can be a potential increase in the victim delay. Also, notice the non-linear waveform shape that is caused by the glitch. This is shown in the figure below.

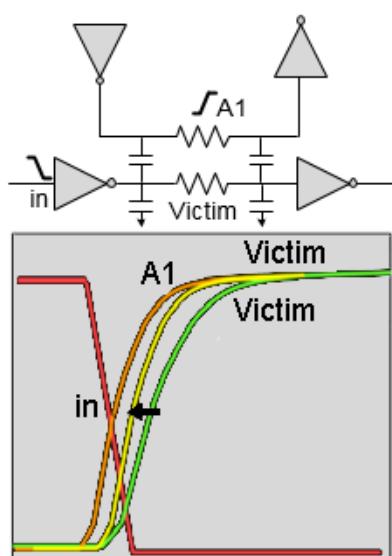
Figure 4-9 SI increases delay



SI can also decrease the delay and cause hold time failures. If both the attacker and victim are switching in the same direction, as shown in the figure below, then there is a decrease in the victim net delay. If this occurs on a critical minimum delay path, it can lead to hold violations. In other words, if a victim net transitions sooner than it was intended to, then it may cause hold violations

The following figure illustrates decreased delay.

Figure 4-10 SI decreases delay

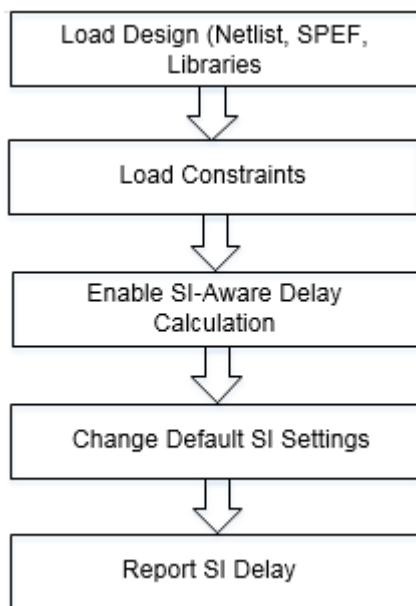


Tempus calculates the worst-case change in delay value due to cross-coupled attackers and reports the change in delay in the timing and SI reports.

4.2.4 Signal Integrity Delay Analysis Flow

The SI delay analysis flow is given below:

Figure 4-11 Tempus SI Delay Analysis Flow



Sample SI Delay Calculation Script

You can use the following script to calculate base and SI delay:

1. Specify the design size:

```
set_design_mode -process 28
```

2. Load the design data. Follow the steps given below:

- Load the timing libraries (.lib) and characterized data (.cdb):

```
read_lib typ.lib -cdb typ.cdb
```
- Schedule reading of a structural, gate-level verilog netlist:

```
read_verilog dma_mac.v
```
- Set the top module name, and check for consistency between Verilog netlist and timing libraries:

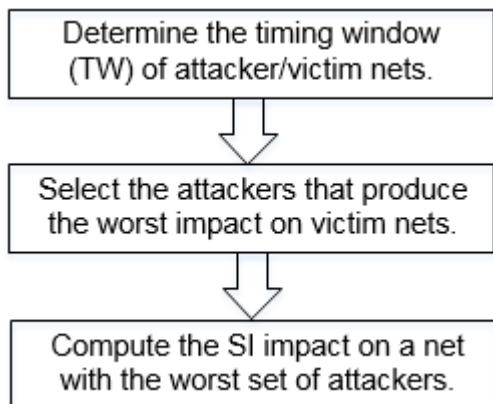
- ```
set_top_module dma_mac dma_mac
```
- ❑ Load the timing constraints:  
read\_sdc constraints.sdc
  - ❑ Load the cell parasitics:  
read\_spief cell.spief.gz
3. Set the individual attacker threshold:  
set\_si\_mode -individual\_attacker\_threshold 0.015
4. Perform 3 SI iterations:  
set\_si\_mode -num\_si\_iteration 3
5. Separate delta delay on data:  
set\_si\_mode -separate\_delta\_delay\_on\_data true -enable\_delay\_report true
6. Enable SI-aware delay calculation:  
set\_delay\_cal\_mode -siAware true
7. Report the timing:  
report\_timing
8. Report details of SI delay calculation for an arc:  
report\_delay\_calculation -si -from inst1/A -to inst1/Y
9. Report delay uncertainty due to SI:  
report\_noise -delay {min | max}

#### 4.2.5 Performing Signal Integrity Delay Analysis

Tempus performs SI analysis in conjunction with base timing analysis. The SI delay analysis calculates the change in delay due to coupling noise acting both with (min) and against (max) a transitioning signal and outputs a delay uncertainty report in text format. SI delay analysis uses a fast transistor-level transient **Timing Windows (TW) Illustration** simulation engine to ensure the highest accuracy. This is done by accounting for non-linear waveforms created due to the coupling noise and attributing delay changes to the victim.

Tempus performs the following operations during SI delay analysis:

**Figure 4-12 Tempus SI Delay Analysis Operations**



These are explained in the sections below.

- [Determining Timing Windows](#) on page 65
- [Selection of Attackers](#) on page 70

### **Determining Timing Windows**

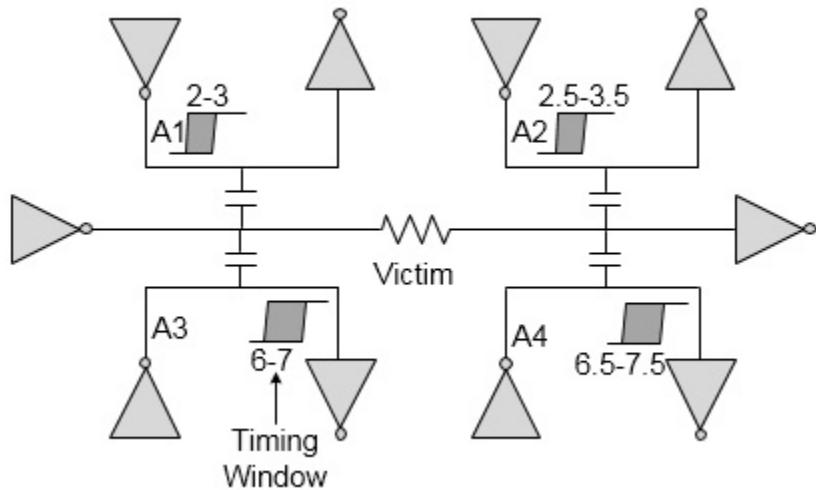
Tempus can generate or read in the timing window information (using the `read_twf` command) to determine which signals can (or cannot) switch simultaneously. The slews are used to switch attacking signals. Timing windows represent the range of time during which a net can transition. They are used to decide when it is possible for multiple attackers to combine.

Attacker timing windows are exhaustively checked for the worst-case allowable combination for the final combined simulation.

- The worst case is determined by each attacker's glitch noise contribution.
- The victim's timing windows are also checked for SI delay analysis.
- The attacker slew affects the magnitude of the noise peak/incremental delay .

A more accurate slew reduces the noise peak, if the transition time is greater than the default fast slew.

**Figure 4-13 Timing Windows (TW) Illustration**



In the above figure, the ranges represent the TWs for respective nets. For attacker A1, TW range is 2-3, that is, the time interval in which net A1 can switch. The attackers A1 and A2 can be combined to produce a bigger SI impact since their TWs are overlapping. Similarly, attackers A3 and A4 can be combined since TWs are overlapping.

Now, depending on which one of the two combinations (A1-A2, or A3-A4) produces worst SI impact, one of the combinations will be selected by the software for SI analysis.

### **Timing Windows Iterations**

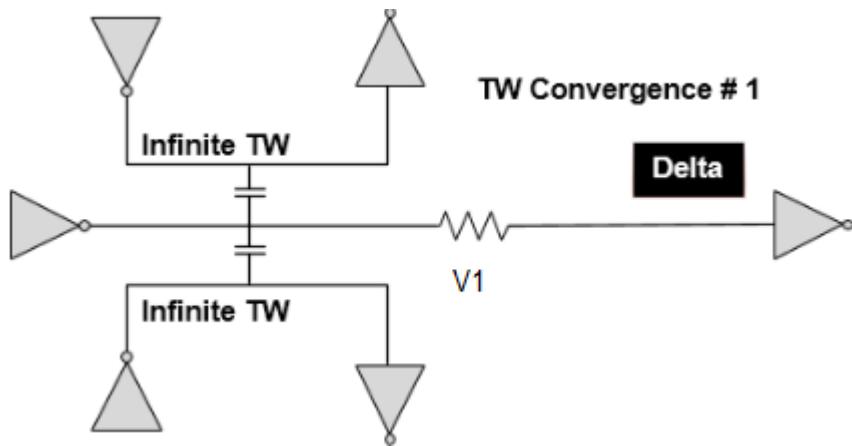
The SI effect on delay depends on timing windows and slew, whereas the timing window and slew depends on the SI effect of delay. Therefore, the timing windows impact SI delay and SI delay impacts timing windows.

Hence, an iterative approach is required to accurately compute the timing windows affected due to SI so that Tempus can accurately calculate the SI delay and glitch.

There are two primary approaches to timing window iteration. The default approach starts with an infinite timing window and iterates while shrinking the timing window width. The second approach starts with a nominal or noiseless timing window and iterates while growing the timing window width.

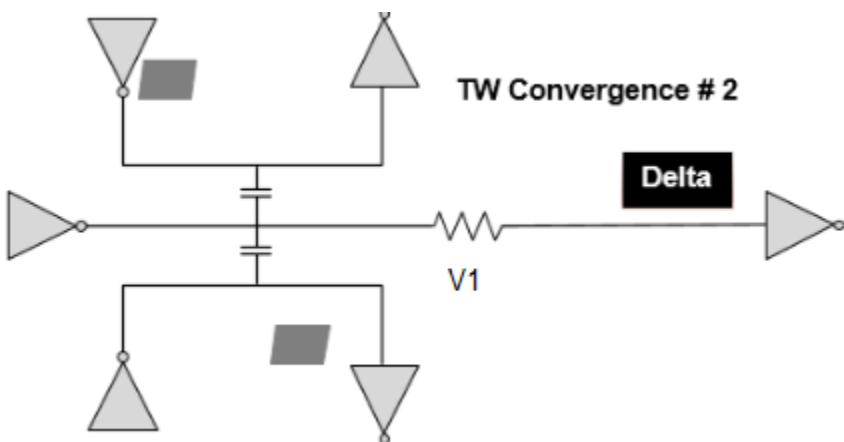
The following figure shows the default approach to start with infinite timing windows and calculates the initial delta delays.

**Figure 4-14 Timing Windows (TW) 1st Iteration**



In the second iteration, as shown in figure below, the delta delays are used to calculate new arrival times, and therefore new timing windows. These new timing windows are used to calculate new delta delays. The new delta delays are smaller because the timing windows are more precise (smaller) compared to infinite timing windows.

**Figure 4-15 Timing Windows (TW) 2nd Iteration**



By default, Tempus performs two iterations. You can change the number of Iterations by using the following command:

```
set_si_mode -num_si_iteration Number
```

All the nets are not reselected for analysis in each timing window iteration. Tempus supports the following methods of re-selection.

- Slack based
- Delta delay based

■ Xcap ratio based

These reselection criteria can be specified by using the command:

```
set_si_mode -si_reselection {delta_delay | slack | xcap_ratio} (default is slack)
```

The default approach of starting with infinite TW will incur certain pessimism as in the initial iteration, the nets will be having much bigger computed noise as compared to the actual scenario. This pessimism will be reduced in the subsequent iterations as the nets are reselected and their timing windows are shrunk. But reselecting all the nets will have a huge run time impact. Therefore, the above mentioned criteria determines which nets are to be reselected for further iterations. Understanding the trade-off between the runtime and pessimism reduction, you can appropriately decide how the above criteria needs to be deployed in the design.

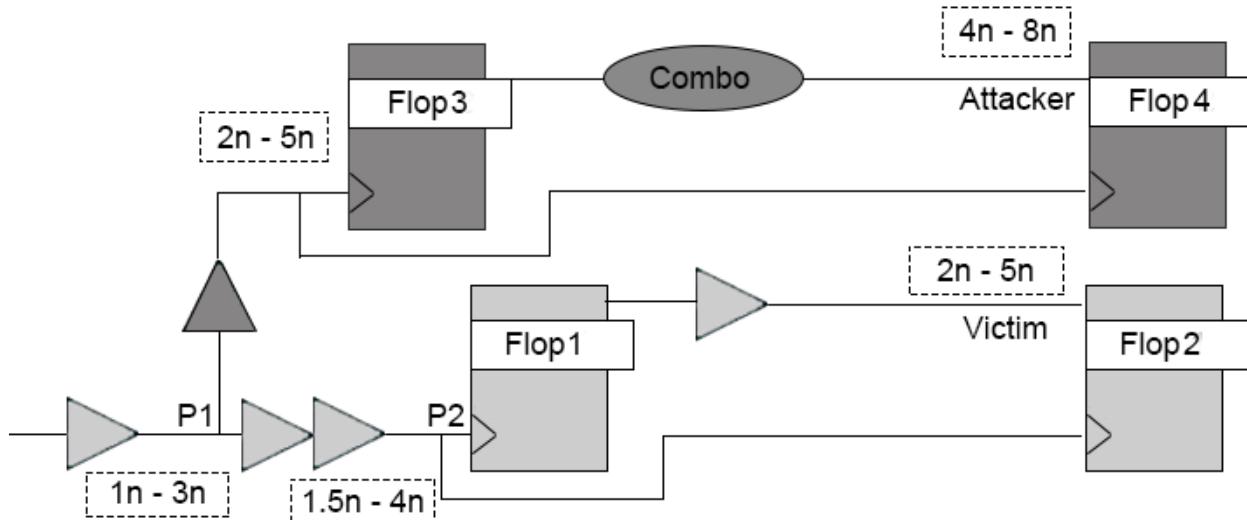
Also, it is important to understand that reselecting a particular net does not ensure that all pessimism is removed from SI analysis on that net. For instance, if the timing critical nets are being reselected, the attackers of the critical victim nets may not necessarily be critical, so the timing windows on the attackers may be pessimistic.

### ***Timing Window CPPR***

Timing windows of nets are computed irrespective of their relative positioning. There can be some pessimism in analysis when the victim/attacker share common clock path.

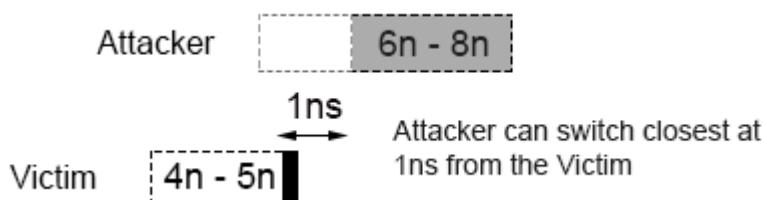
Attacker attacks a victim with its full strength if it switches at the same time as the victim, and the impact of the attacker on the victim declines as the distance between switching times increases. The computation of closest switching time based on these arrival times can be pessimistic if both attacker and victim share the same clock path.

**Figure 4-16 Timing Windows on CPPR Mode**



In the example above, for path from Flop1 to Flop2, Attacker's TW (4n – 8n) is overlapping with Victim's TW (2n – 5n). The clock arrival (at point P1), which is causing victim to reach at 5ns is 3ns. As both victim and attacker are sharing the same launch clock path till P1, they both will be launched with the same clock arrival time till the point. So corresponding to the arrival time of 3ns (at P1) the attacker can switch earliest by 6ns (and not 4ns). Therefore, the actual positioning of the attacker and victim will be, as shown in the figure below. The attacker can only attack from a shortest distance of 1ns from the victim switching time.

**Figure 4-17 Timing window of Attacker and Victim not Overlapping**



You can remove pessimism by enabling the following global variable:

```
set timing enable timing window pessimism removal true
```

The adjustment in attacker TWs accounts for the difference in min-max delay on the top common clock path due to the following factors:

Difference in base arrival of the top clock path due to early/late derates.

Min/Max SI impact on common clock path (adjusted only during hold analysis).

## Selection of Attackers

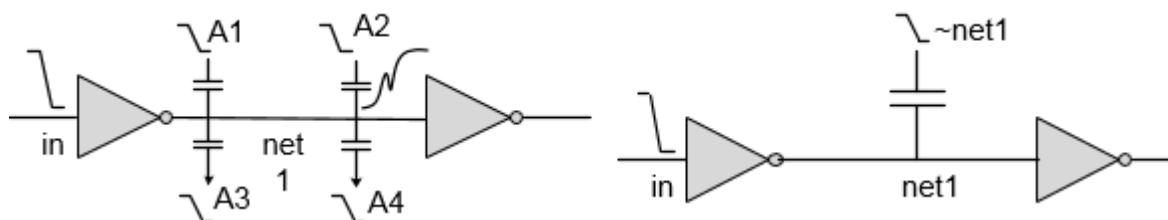
After determining the timing windows of attacker/victim nets, Tempus selects the attackers that produce the worst impact on the victim net. Tempus controls the attacker's selection based on the following factors:

- [Strength of Attacker](#) on page 70
- [Attacker Selection Based on Alignment](#) on page 72
- [Attacker Selection Based on Logical Correlation](#) on page 82

### Strength of Attacker

The strength of an attacker is the amount of glitch it produces at victim's receiver input and it depends on the coupling capacitance between attackers and victims. With new processes (smaller geometries), the ratio of  $C_{cpl}/C_{tot}$  is increasing. Some victims can have 100s or 1000s of attackers and many of these attackers can be small ones. SI analysis run time is a strong function of the number of attackers. Higher numbers of attackers can result in high analysis run time. Tempus provides a better way of handling small attackers by combining these small attackers together. Tempus models many small attackers on a victim net and replaces them with an imaginary net, called a virtual attacker.

**Figure 4-18 Virtual Attackers Modeling**



In the figure above, the small attackers, namely A1, A2, a3, and a4, are combined together to form a virtual attacker ~net1.

Tempus identifies an attacker as an individual attacker if the glitch contribution from this attacker is more than the threshold value specified with the `set_si_mode -individual_attacker_threshold_value` command. If the peak of this glitch on the victim net exceeds the specified threshold value, then the attacker is significant and is individually reported in the constituents section of the SI report. You can use the `-individual_attacker_clock_threshold` parameter to vary small attacker threshold for victim nets lying on a clock path.

If the glitch contribution of an attacker is less than the threshold value defined with the `-individual_attacker_threshold` of the `set_si_mode` command, then this attacker becomes eligible for the virtual attacker.

The two main characteristics of a virtual attacker are: coupling capacitance and the voltage source waveform used as a transition on the virtual attacker. You can use the `set_si_mode -accumulated_small_attacker_mode` command to create and control the characteristics of the virtual attacker. Tempus provides the following two methodologies for creating virtual attackers:

- Current Matching-Based Methodology
- Coupling Capacitance Matching-Based Methodology

**Note:** Both the methodologies account for the low probability of all small attackers switching at once in order to reduce the pessimism on noise and SI delay analysis.

### **Current Matching-Based Methodology**

To create virtual attackers using this methodology, you can set the following command:

```
set_si_mode -accumulated_small_attacker_mode current
```

Tempus creates virtual attackers to match both the coupling capacitance and the noise current. The PWL waveforms are generated in such a way that current induced by the transition matches the total current induced by all virtual attacker components.

This methodology differs from the coupling capacitance methodology as follows:

- Keeps several of the top virtual attacker components and calculates a PWL waveform for each analysis type using TW filtering.
- Improves on the statistical method (3-sigma method) by accounting for the probability of switching.

### **Coupling Capacitance Matching-Based Methodology**

To create virtual attackers using this methodology, you can set the following command:

```
set_si_mode -accumulated_small_attacker_mode cap
```

**Note:** This is the default methodology for all process nodes above 45nm. The 45nm and below default mode is current.

Tempus creates virtual attackers that model the combined coupling capacitance effects of small attackers on a victim net. Every victim net can potentially have a virtual attacker. Tempus assigns a name to each virtual attacker based on the name of the victim net. For example, if the victim net name is ABC, then the virtual attacker net name will be ~ABC. The

pessimism of using the virtual attacker can be reduced by adjusting the value of the `accumulated_small_attacker_factor` between 0 and 1.

```
set_si_mode accumulated_small_attacker_factor <value>
```

To recognize those attackers that may have a significant impact on a victim net, Tempus performs a quick and conservative estimation that takes into account the resistance and capacitance (RC) characteristics of the attacker and the victim net, as well as the worst-case drive strength and the load of the victim net. If the estimated coupling capacitance effect from an attacker is greater than five percent of VDD, the attacker is explicitly reported as an attacker constituent of that particular victim net. It is typical for a victim net to have as many as three or four attackers of this type. In general, attackers are usually very small and fall under this five percent threshold.

You can change the virtual attacker glitch threshold using the `set_si_mode - accumulated_small_attacker_threshold` command. The default value of this parameter is 10.01 percent of VDD. For example, to disable virtual attacker you can use a threshold value greater than 1 (default), by using the following command:

```
set_si_mode -accumulated_small_attacker_threshold 1.01
```

The virtual attacker provides time and memory savings, but sacrifices some accuracy, reporting detail, and control on slew and timing window information. Because the virtual attacker is used only for small attackers, these sacrifices should have a minimal impact on results.

## Attacker Selection Based on Alignment

Tempus supports three attacker alignment modes that allow you to select attackers based on their alignment with victim nets.

- Path Mode
- Path Overlap Mode
- Timing-Aware Edge Mode

The attacker alignment mode can be set by using the following command:

```
set_si_mode [-attacker_alignment {path | path_overlap | timing_aware_edge}]
```

The default mode is `timing_aware_edge`.

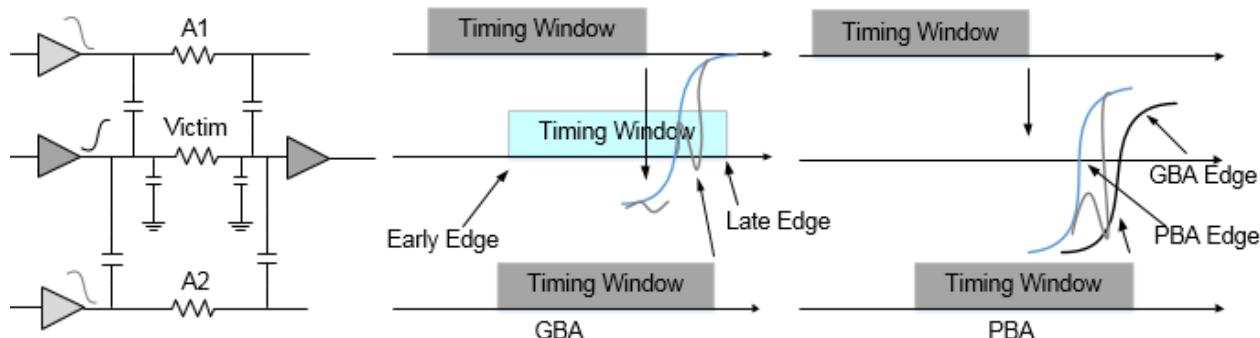
### ***Path Mode***

In path alignment mode:

- Tempus uses the victim's worst arrival edge (as shown in the figure) when determining which attackers can align with it.
- SI impact is computed according to the latest/earliest arrival edge of the victim for maximum/minimum analysis.
- This mode yields a more realistic analysis.

Path-based alignment annotates each attacker's impact on victim transition where attacker timing window edge aligns. The following example shows that in GBA mode Attacker 2 timing window edge is close to the delay measurement point, thus resulting in maximum impact.

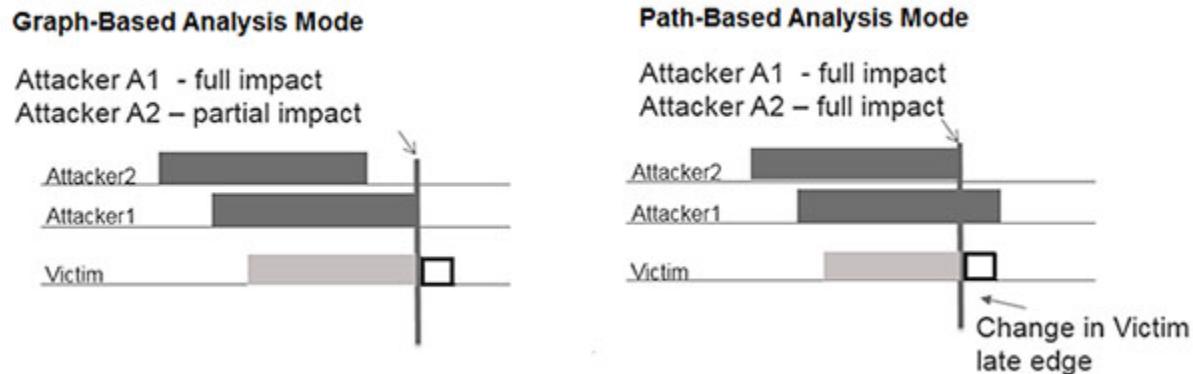
**Figure 4-19 Path Alignment Mode**



The path-based alignment is more realistic analysis and is less pessimistic than the other two modes. However, GBA is not guaranteed to bound PBA delays. In this example, the victim arrival time gets faster (move left) due to retiming of victim timing in PBA. With retimed arrival time of victim, both the attacker timing window edges now align at the delay measurement point, thus PBA sees a larger cross-talk impact than GBA.

The flow in path mode is illustrated in the figure below.

**Figure 4-20 Attackers alignment in path mode**



In the above figure, for paths that are based on the last arrival edge of victim net, only A1 will be selected for alignment in path mode.

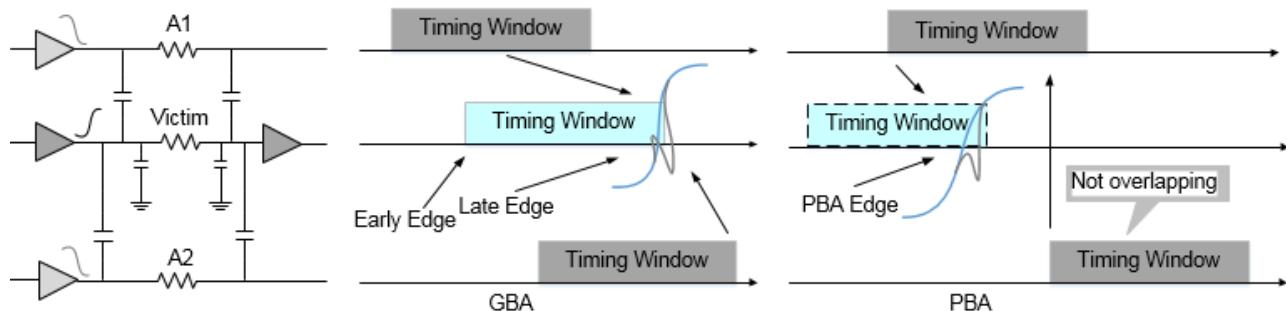
### **Path Overlap Mode**

In path overlap mode, Tempus uses the full timing window of the victim (as shown in the figure), when determining which attackers can align with it.

SI impact is calculated according to the victim edge that is lying anywhere within the timing window where the maximum impact occurs.

The path overlap mode considers all the attackers which overlap anywhere on victim transition and annotate their combined impact at delay measurement point in both GBA and PBA. Due to this methodology, the computed SI impact based on path overlap is pessimistic. However, GBA is guaranteed to bound PBA analysis. The following example illustrates path overlap alignment mode:

**Figure 4-21 Path Overlap Alignment Mode**

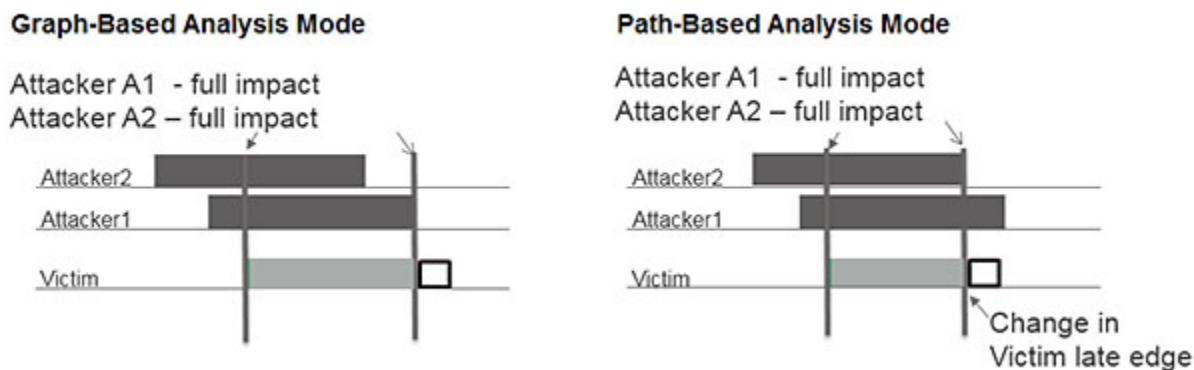


This mode is more conservative in cross-talk delay computation among the other available modes.

PBA analysis does not change based on attacker alignment modes, that is, PBA analysis always annotates individual attacker impact on victim transition where ever attacker timing windows overlap in all the three modes.

The flow in path overlap mode is illustrated in the diagram below.

**Figure 4-22 Attackers alignment in path overlap mode**



Even if victim timing window changes, the attackers' selection will remain the same. This mode ensures that GBA bounds PBA delays.

In the figure above, consider a full timing window of a victim net - both attackers A1 and A2 will be considered for alignment with victim and hence both A1 and A2 are selected in the path overlap mode. This mode is more pessimistic.

In path-based analysis (PBA) mode, Tempus uses the specific edge for the victim that occurs for a specific path when determining the alignment of the attackers to the victim. It is possible that the worst SI impact on a net occurs at a victim edge that is not the worst arrival edge for the victim, thus the PBA victim edge used inside the timing window for the victim can yield a worst-case SI result for a net. For more information, refer to section on [Path-Based Analysis](#) on page 182.

To ensure that GBA results bound PBA, you should use the path overlap alignment mode.

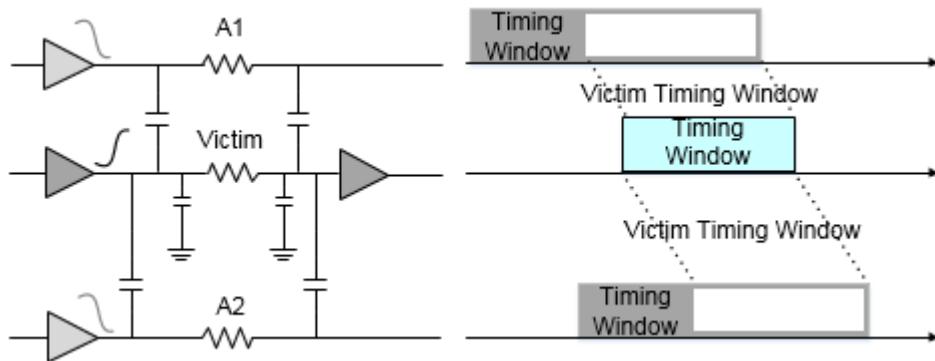
### **Timing-Aware Edge Mode**

Timing-aware edge mode performs realistic analysis similar to path-based analysis. Each attacker's impact is annotated on victim transition where attacker timing window edge aligns

with the victim. However, to ensure GBA bounds PBA analysis, the timing-aware edge mode handles some nets based on their timing properties in GBA analysis.

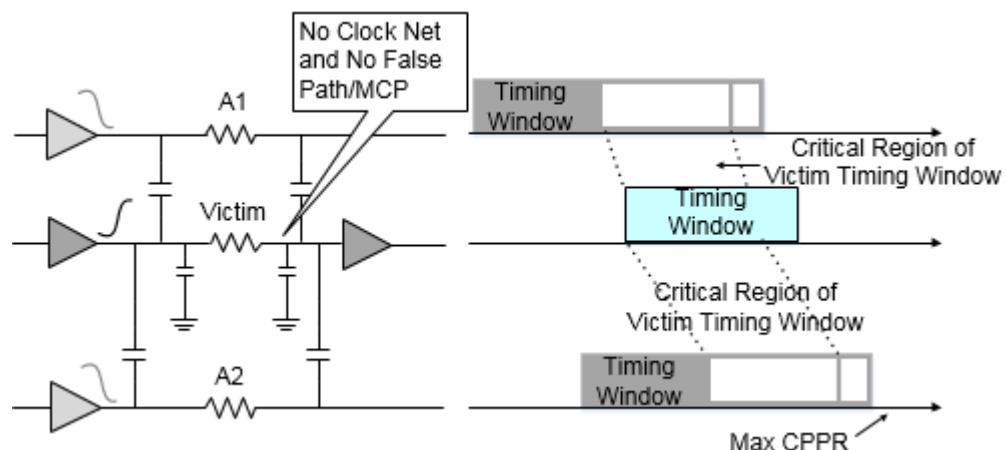
In the following example, if the victim net is a clock, or with exceptions like false paths or MCP, the attacker timing windows are extended by width of the victim timing window.

**Figure 4-23 Timing-Aware Edge Alignment Mode - Illustration 1**



If the victim net is not a clock, and has no exceptions like false path or MCP, then all the attacker timing windows are extended by a critical region of the victim timing window, in addition to the maximum CPPR, as shown below.

**Figure 4-24 Timing-Aware Edge Alignment Mode - Illustration 2**



During PBA analysis, the attacker timing windows are extended by the actual CPPR of the path, and no additional padding is required.

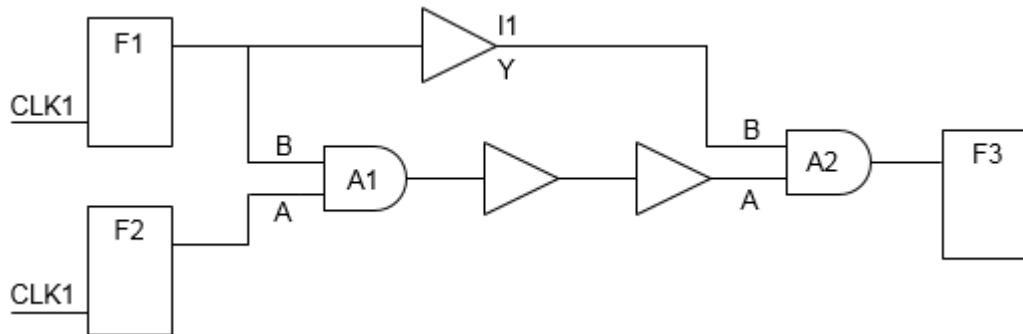
## Clock and Exception Handling Timing-Aware Edge Mode

If the victim net is a clock net, and the attacker timing windows are extended by width of the victim timing window to ensure that an attacker overlaps with the victim arrival time. Any optimism on a clock net in GBA can result in large TNS optimism (in GBA) as compared to PBA, because many paths share the same clock path. To eliminate this situation, the clock nets are handled specially in timing-aware edge mode. The exceptions in certain scenarios can cause optimism in GBA in path-based alignment mode.

Consider the following example:

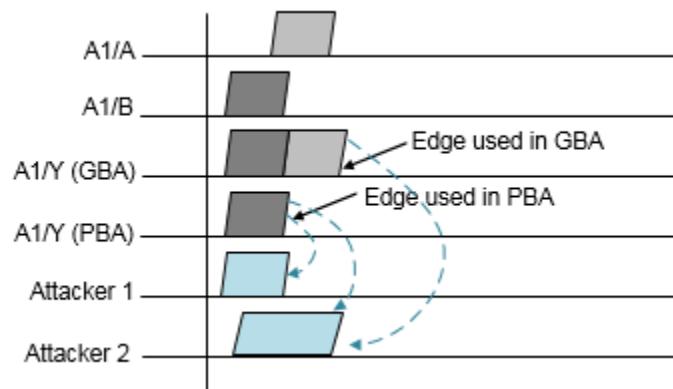
```
set_false_path -through A1/A -through A2/A
```

**Figure 4-25 Timing-Aware Edge Mode - Example 1**



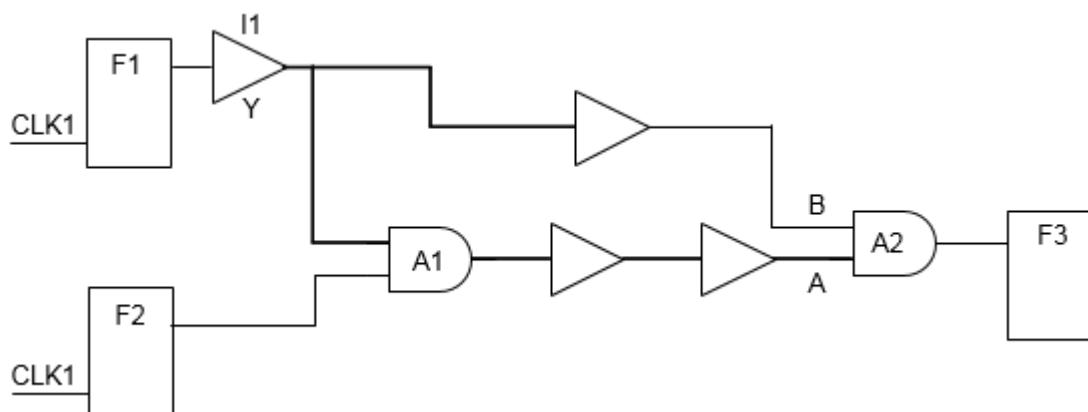
In this example, a path going through A1/B->Y in GBA analysis uses late edge, which overlaps with attacker 2 only. However in PBA mode, a path going through A1/B->Y overlaps with attackers 1 and 2 as well, this results in a larger delay than GBA. If the path ending at F3 is a critical path, then the late arrival time used in GBA analysis at A1/Y is not an optimal edge as it produces optimism in GBA. In fact, late arrival used in GBA is coming from a pin that has a false path. To overcome this caveat, timing-aware edge mode extends the attacker timing windows by width of victim timing windows, such that GBA does not report optimistic delay on critical path.

**Figure 4-26 Timing-Aware Edge Mode**



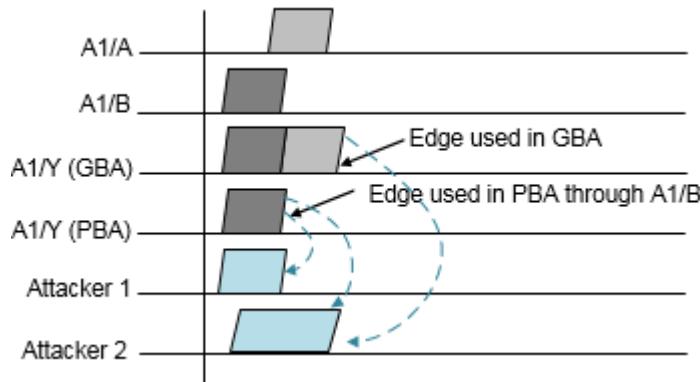
Similarly, consider the following multi-cycle path example:

**Figure 4-27 Timing-Aware Edge Mode - Example 2**



If the late arriving edge at A1/Y is based on arrival time from A1/A, then the SI delay computed based on that edge will see only one attacker, thus resulting in a smaller cross-talk delay compared to the cross-talk delay computed based on the arrival time from A1/B. If a path is reported from F2 in GBA and PBA, then PBA reports a larger delay at stage A1 due to this optimism.

**Figure 4-28**

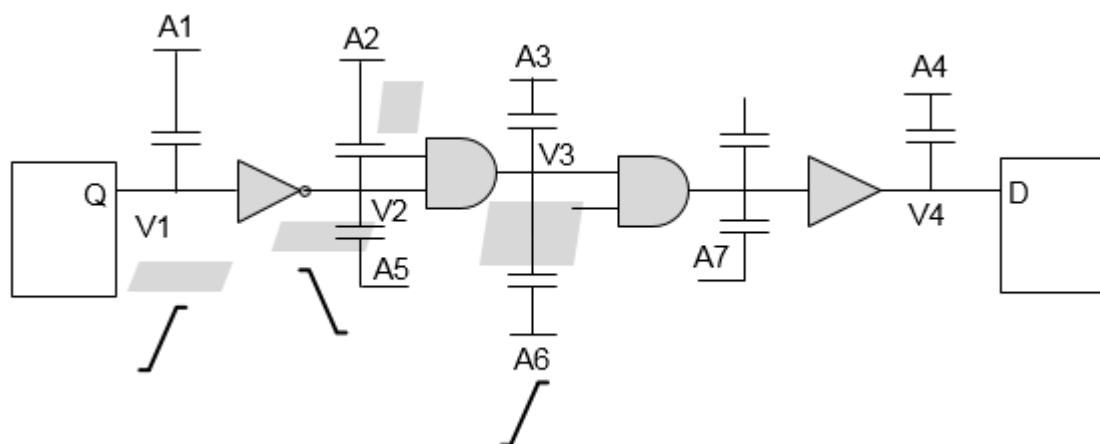


To address this optimism in GBA analysis in path-based alignment, the `timing_aware_edge` mode expands the attacker timing windows for stages with exceptions, such that GBA analysis will be pessimistic than PBA.

#### ***Path-Based Analysis with Signal Integrity***

Path-based SI analysis reduces the pessimism introduced during graph-based SI analysis by considering path-based slews and arrival for victims. This helps in reducing design closure ECO cycles, and saves area and power of design chips.

**Figure 4-29 Analysis performed using actual path slew and arrival times of victim**



Running PBA-SI on a complete set of paths in the design is computation intensive. It is recommended to run PBA-SI on critical paths identified from GBA-SI analysis. So, while using PBA-SI based signoff ensure that GBA-SI results are always pessimistic to PBA-SI.

This can be ensured by running GBA-SI in path overlap mode, by using the following command:

```
set_si_mode -attacker_alignment path_overlap (default is path)
```

### ***Attacker Selection Based on Relationship Between Clocks***

The selection of attackers also depends upon the relationships between the clocks. Presence of multiple clock groups - physically exclusive, asynchronous, and synchronous - in the design will influence attacker selection behavior of Tempus differently. Tempus by default considers all the clocks as synchronous clocks if no clock group is mentioned. You can change the behavior by using below command below:

```
set_si_mode -clocks {asynchronous | synchronous} (default is synchronous)
```

Synchronous clocks have defined phase relationships amongst each other. Timing windows are considered while evaluating the attacker/victim relationship.

The default behavior can be overridden by using the following command. This provides an option to specify the timing relationship between specific clock domains.

```
set_clock_groups
[-group <clock_list>]
[-name <name_list>]
[-logically_exclusive] | [-physically_exclusive] | [[-asynchronous]]
```

The preferred order of clock relationships is physically exclusive (highest) and then asynchronous.

The table below explains the relationships and impact of different clock grouping on STA and SI analysis:

**Table 4-1 Relationships and impact of different clock grouping**

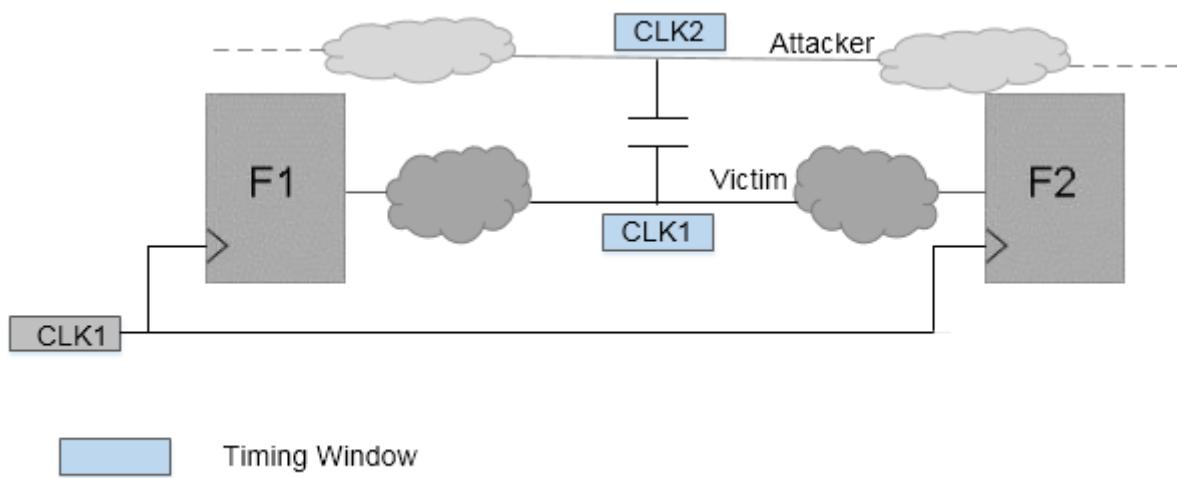
| <b>Relationship</b> | <b>Description</b>                                                                 | <b>Impact on STA or Timing Paths</b>  | <b>Impact on SI Analysis</b>                                     |
|---------------------|------------------------------------------------------------------------------------|---------------------------------------|------------------------------------------------------------------|
| Logically Exclusive | Specifies that there is no logical path between the clocks even though they exist. | Makes false path between such clocks. | No impact on SI Analysis. Crosstalk impact will remain the same. |

|                      |                                                                                                                                                         |                                       |                                                                                                                                                                                                                                                                                  |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Asynchronous         | Indicates that there is no phase relationship between these clocks.                                                                                     | Makes false path between such clocks. | Attackers from asynchronous clock to victim clock are considered to have infinite timing windows.                                                                                                                                                                                |
| Physically Exclusive | Specifies that these clocks will not exist physically at the same time in the chip.<br><br>Such clocks mainly exist in constraints during mode merging. | Makes false path between such clocks. | Attackers from physically exclusive clock to victim clock are considered non-switching during SI analysis of the victim (that is, they do not attack the victim).<br><br><b>Note:</b> When multiple clocks on a victim net exist, then they are not marked physically exclusive. |

Consider a few scenarios for clock grouping, as follows:

**Scenario1:** In the following figure, the attacker is receiving timing window (TW) with respect to the physically exclusive clock.

**Figure 4-30 Attackers receiving TW with respect to Physically exclusive clock**

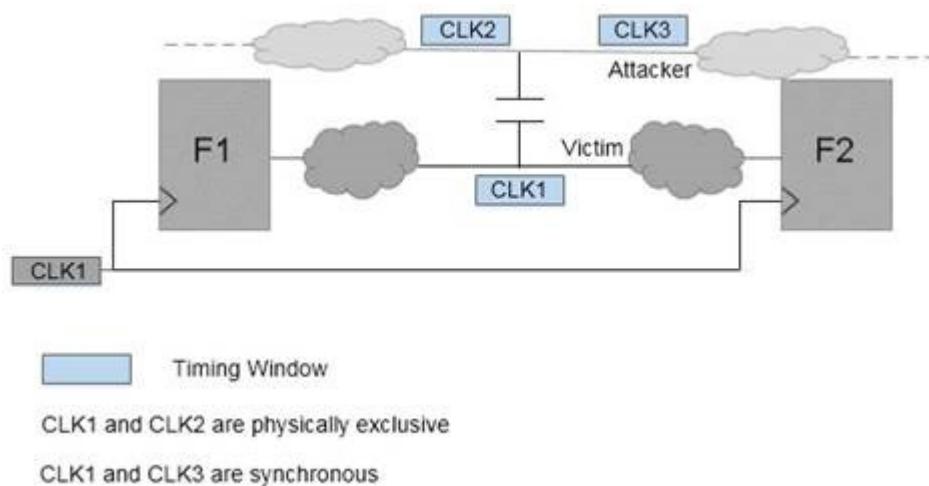


In this case, the attackers will be dropped even if the timing window overlaps with that of the victim.

No impact of attacker will be considered on the victim.

**Scenario 2:** In the following figure, the attacker has a timing window with respect to Synchronous as well as Physically Exclusive clocks to the victim.

**Figure 4-31 Attackers having TW with respect to Synchronous as well as Physically Exclusive clock**



In this case:

- Timing windows with respect to physically exclusive clocks to victim will be dropped (CLK2) and victim-attacker relationship is determined using the attacker timing window with respect to only synchronous clock (CLK3).
- Victim timing window (with respect to CLK1) overlapping is checked with attacker timing window (with respect to CLK3). As they do not overlap, attacker will not impact the victim.

#### ***Attacker Selection Based on Logical Correlation***

During SI analysis Tempus introduces some pessimism when attackers that cannot logically attack together are allowed to attack. Tempus by default assumes that all attackers can switch together in a direction to cause a worst case delay (causing a slowdown) or speed up of transition on the victim. But in some cases, there could be logical relationship between the signals, that is, attackers might not actually switch together in the same direction.

Tempus performs logical correlation for inverters and buffers, except complex gates that are not included in logical correlation.

You can use the following command to enable logical correlation:

```
set_si_mode -enable_logical_correlation true (Default: false)
```

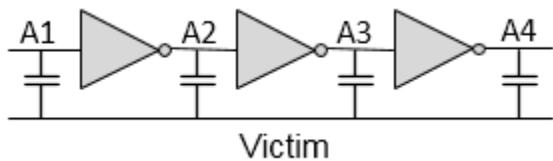
The following examples show some of the scenarios that logical correlation can handle.

### **Scenario 1: Relationship between Attackers**

Exclusive sets are subsets of attackers that can switch simultaneously.

Exclusive sets can be {A1, A3} and {A2, A4}. Here, attackers A1 and A3 will be switching in one direction, whereas A2 and A4 will switch in the opposite direction, as shown in the figure below.

**Figure 4-32 Logical Relationship Between Attackers**

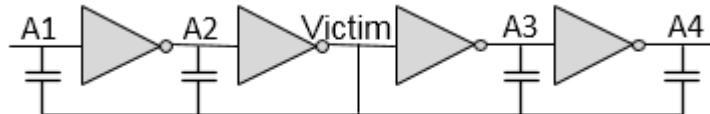


### **Scenario 2: Relationship between Victim and Attackers**

In max analysis mode, attackers A1 and A4 can be excluded. Because, these nets cannot switch in the opposite direction to the victim to increase the delay.

In min analysis mode, attackers A2 and A3 can be excluded. Because, these nets cannot switch in the same direction as the victim to decrease the delay.

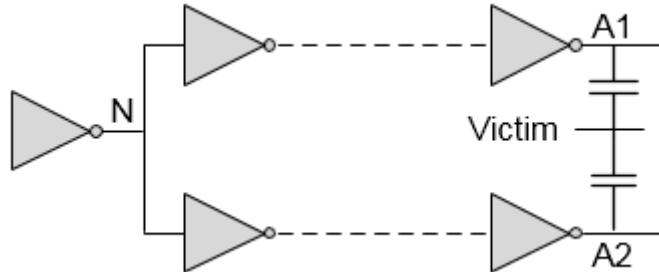
**Figure 4-33 Logical Relationship Between Victim And Attackers**



### **Scenario 3: Remote Attackers**

If the number of inverters on each branch does not differ by a multiple of 2, then either A1 or A2 will be used as an attacker, as shown in the figure below.

**Figure 4-34 Logical Relationship Between Remote Attackers**



### Waveform Computation

SI impact on delay (or glitch for glitch analysis) depends primarily on the following factors:

- Victim Slew
- Attackers' Slew
- Victim coupling cap to ground cap ratio
- Relative switching direction for victim/attacker
- Arrival windows of attackers/victim nets

SI impact on delay is performed in four steps. The first three steps will be applicable for SI Glitch analysis as well.

- Determine the slew of possible attackers and victim nets in their respective directions.
- Determine the alignments of overlapping attackers to compute the worst impact.
- Align all attackers at their respective positions.
- Compute the stage delay.

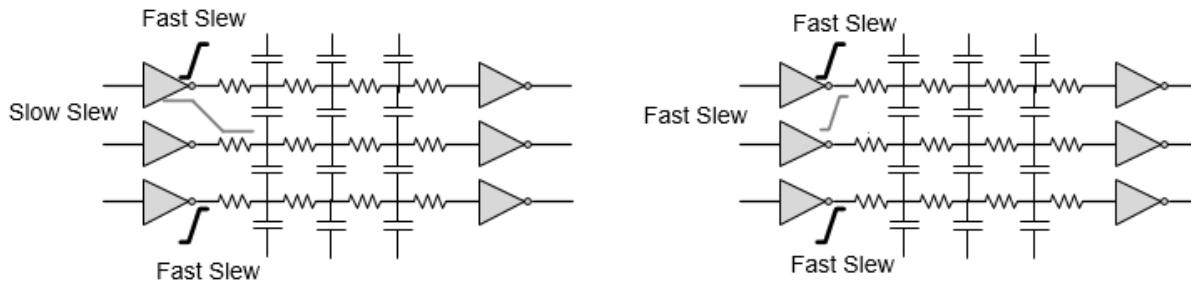
These are explained below:

#### **Step 1: Determine Slew of possible Attackers and Victim Nets in respective directions.**

Tempus implicitly considers attacker slew degradation by including the attacker's distributed parasitic network in the simulation.

Tempus determines the fastest and slowest slew possible in each direction (rise/fall) at every node in the design. For max delay analysis, the slowest slew for victim and the fastest slew (in opposite direction of victim slew) for attackers will be used, as shown in the figure below.

**Figure 4-35 Determine slew for max delay and min delay**



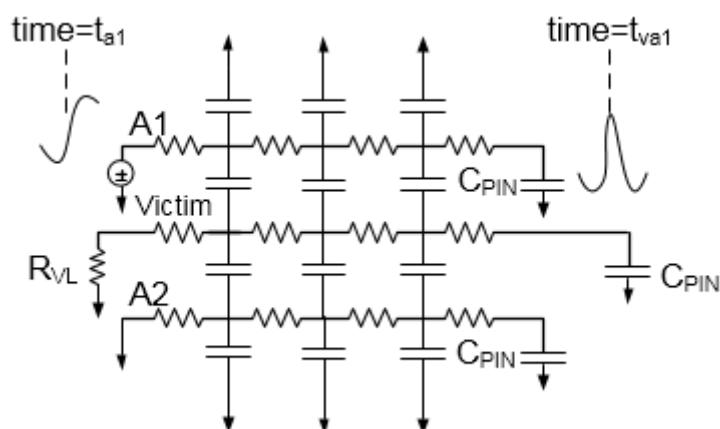
For min delay analysis, the fastest slew for victim and the fastest slew (in the same direction of victim slew) for attackers will be used, as shown in the figure above.

### **Step 2: Determine the alignments of overlapping attackers to compute the worst impact**

Tempus aligns fully overlapping attacker's switching time in a way that creates worst impact of each attacker on the victim receiver at the same time. Tempus computes the time of flight for each attacker by individually switching each of them. The time of flight is the time between the attacker switching time at the driver output and the corresponding glitch peak time at the victim receiver input.

Tempus computes the delay from each attacker to the victim for peak alignment, as illustrated in the figure below.

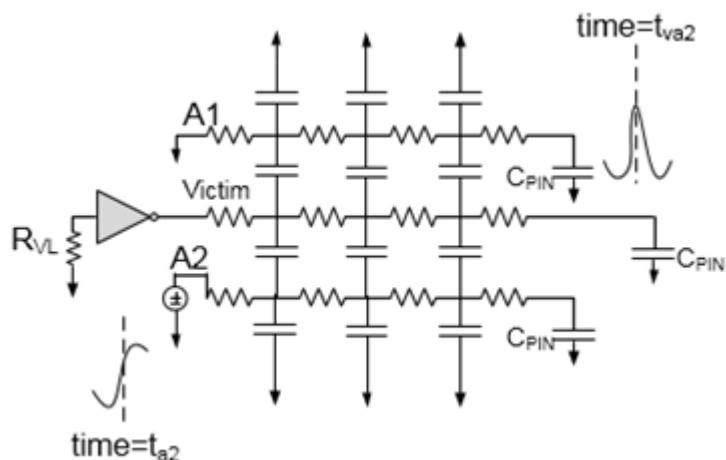
**Figure 4-36 Compute Attacker A1 To Victim Delay ( $t_{vA1} - t_{A1}$ )**



To compute time of flight for attacker A1, a voltage source with a slew rate, that is either calculated internally or annotated from static timing analysis, is applied at attacker A1 driver output while keeping all other attackers and victims silent.

Tempus calculates the delay from the 50 percent point of the switching attacker A1 at the driving pin ( $t_{a1}$ ), to the time the glitch effect on the victim receiver is maximum ( $t_{va1}$ ). For attacker A1, the delay is  $t_{va1}-t_{a1}$ .

**Figure 4-37 Compute Attacker A2 To Victim Delay ( $t_{va2} - t_{a2}$ )**



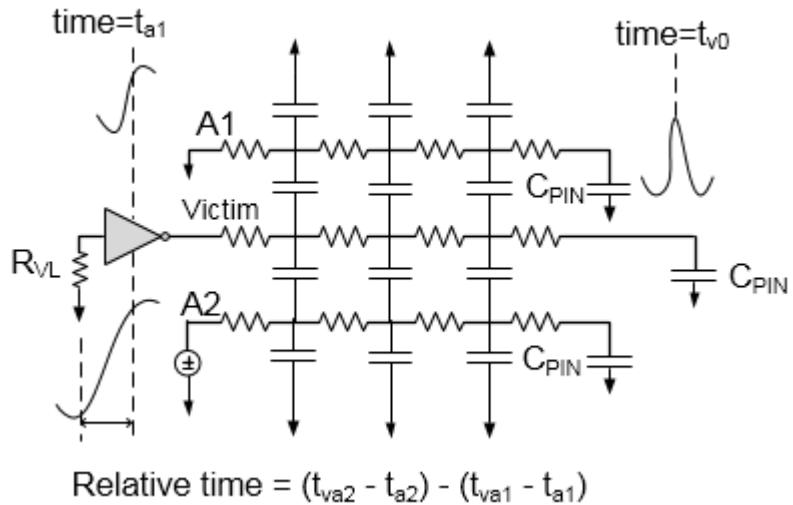
Similarly, Tempus calculates the delay from the 50% point of the switching attacker A2 at the driving pin ( $t_{a2}$ ), to the time the glitch effect on the victim is maximum ( $t_{va2}$ ). For this case, attacker A1 will be grounded. So, for attacker A2 the delay is  $t_{va2}-t_{a2}$ . These two delays can be different if one attacker is closely coupled to the near end of the victim wire, and the other attacker is closely coupled to the far end of the victim wire, and the victim wire is long.

### Step 3: Aligning all attackers at their respective position

Tempus aligns the relative switch times of the fully overlapping attackers according to their time of flights, such that their individual peak at victim receiver input occurs at the same time.

The attacker relative alignment will be  $(t_{va2}-t_{a2}) - (t_{va1}-t_{a1})$  such that the peak due to attackers A1 and A2 at the victim receiver input occurs at the same time, as shown in the figure below.

**Figure 4-38 Aligning Attackers to Have Worst Impact at Victim at Same Time**

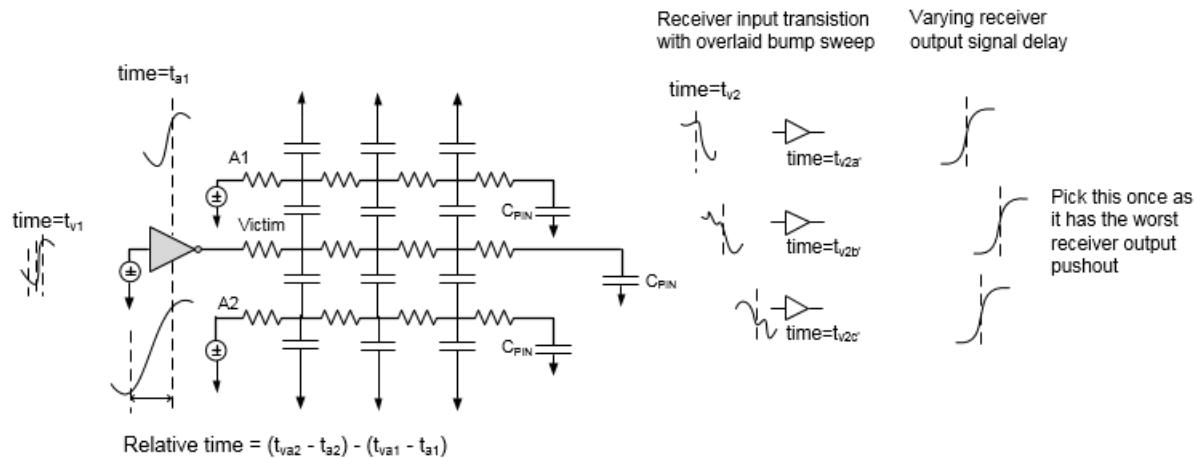


#### Step 4: Compute the stage delay

As a final step to compute the stage delay with SI, a transition is applied at the victim receiver, and the attacker's impact is evaluated across the victim slew to compute the worst-case stage delay.

In the figure below, the delay at the victim's receiver output is measured by considering the attackers across the victim slew. This will result in a change in the delay at the victim receiver's input. Tempus computes different delays as  $t_{v2a}'$ ,  $t_{v2b}'$ ,  $t_{vc2}'$ , and so on for different victim alignment, with respect to attackers. The one resulting in the worst delay will be considered for the worst-case stage delay computation as  $t_{v2}'$ . Assuming that  $t_{v2}$  is the base delay for this stage, that is when all the overlapping attackers of this victim are silent. The delta delay will be  $(t_{v2}' - t_{v2})$ . This difference becomes the SI effect on the delay results (either slow down or speed up).

**Figure 4-39 Compute stage delay with SI ( $t_{v2}' - t_{v1}$ )**



The SI impact on delay depends on the following factors:

#### ***Handling of Unconstrained Nets***

If an attacker net is unconstrained, then Tempus does not consider timing windows of such nets for attacker selection. That means an unconstrained attacker is always selected in SI analysis, that is, timing window is infinite for these unconstrained nets. This makes SI analysis more pessimistic, but bounding.

To turn this off and use a constrained timing window, you can set `set_si_mode -unconstrained_net_use_inf_tw` to `false`. The default is `true`.

#### ***Constraining SI Analysis***

SI analysis adheres to SDC constraints. However, you can additionally specify constraints on SI analysis to control whether a net can be an attacker or not. To do this, you can use the `set_quiet_attacker` command. For example,

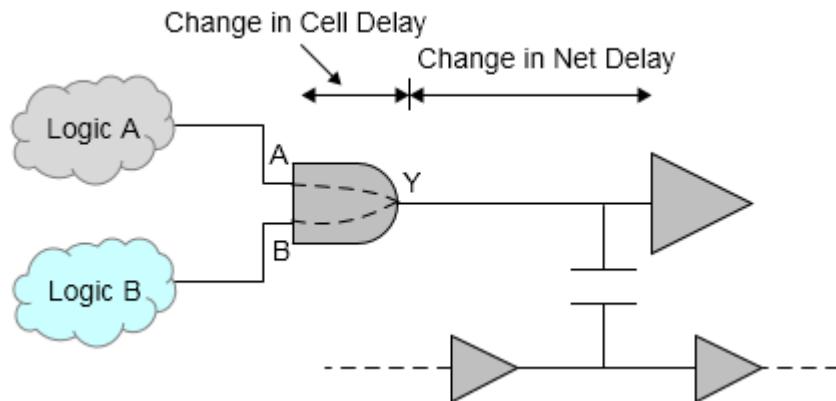
```
set_quiet_attacker -victim vic -attacker {att1 att2} -transition rise
```

This software ignores an attacker for certain edges when in reference to the victim.

#### ***Computing Timing Arc-Aware SI***

The SI impact on delay is computed based on the timing arc through which the path is traversing.

**Figure 4-40 SI Impact on Delay Computed on Timing Arc**



By default, Tempus calculates the delay and SI for each arc. If you choose to view the SI delay separately, it will show annotated to each arc. This helps in reducing pessimism on paths that do not come through the worst arc.

This can be controlled by using the following command:

```
set_si_mode -delta_delay_annotation_mode {lumpedOnNet | arc} (Default: arc)
```

In lumped-on-net mode, Tempus annotates the worst arc delay on the net delay.

#### 4.2.6 SI Delay Reporting

There are several commands for generating reports on SI effects on delay:

- `report_timing`
- `report_delay_calculation -si`
- `report_noise -delay {max | min}`
- `get_property`

These are explained below.

##### Using `report_timing` without Separating Delta Delay on Data

The `report_timing` command, by default, does not separate the delta delay on data (delta delay is always separated for clock). You can set the `incr_delay` option of the `report_timing_format` global variable to display the delta delays of nets. It is recommended to use the `report_timing -net` parameter for generating comprehensive reports. For example,

# Tempus User Guide

## Analysis and Reporting

```
tempus > set_global report_timing_format {hpin cell slew incr_delay delay arrival}
tempus > report_timing -net

Path 1: VIOLATED Setup Check with Pin seg3/u9/CK
Endpoint: seg3/u9/D (v) checked with leading edge of 'CLK_W_3'
Beginpoint: seg3/u3/Q (v) triggered by leading edge of 'CLK_W_3'
Path Groups: {CLK_W_3}
Other End Arrival Time 1.029
- Setup 0.153
+ Phase Shift 2.000
- Uncertainty 1000.000
= Required Time -997.124
- Arrival Time 3.839
= Slack Time -1000.963
 Clock Rise Edge 0.000
 + Clock Network Latency (Prop) 1.202
 = Beginpoint Arrival Time 1.202

Pin Cell Slew Incr Delay Arrival Time

```

| Pin          | Cell | Slew  | Incr Delay | Arrival Time |
|--------------|------|-------|------------|--------------|
| seg3/u3/CK   | -    | 0.094 | -          | 1.203        |
| seg3/u3/Q    | DFF  | 0.340 | 0.000      | 0.368        |
| seg3/u4/A -> | BUF  | 0.341 | 0.000      | 0.011        |
| seg3/u4/Y    | BUF  | 0.003 | 0.000      | 0.165        |
| seg3/u5/A    | INV  | 0.005 | 0.000      | 0.003        |
| seg3/u5/Y    | INV  | 0.516 | 0.000      | 0.225        |
| seg3/u6/A    | INV  | 0.627 | 0.000      | 0.317        |
| seg3/u6/Y    | INV  | 0.913 | 0.000      | 0.634        |
| seg3/u7/A    | BUF  | 0.914 | 0.000      | 0.008        |
| seg3/u7/Y    | BUF  | 0.655 | 0.000      | 0.492        |
| seg3/u7_a/A  | BUF  | 0.659 | 0.000      | 0.121        |
| seg3/u7_a/Y  | BUF  | 0.003 | 0.000      | 0.269        |
| seg3/u87A    | BUF  | 0.003 | 0.000      | 0.001        |
| seg3/u8/Y    | BUF  | 0.042 | 0.000      | -0.009       |
| seg3/u9/D    | DFF  | 0.071 | 0.000      | 0.031        |

```

```

## Using report\_timing with Separating Delta Delay on Data

You can use the following commands to report delta delay on data separately under the `Incr Delay` column in the timing report:

```
tempus > set_si_mode -separate_delta_delay_on_data true
tempus > report_timing -net

Path 1: VIOLATED Setup Check with Pin seg3/u9/CK
Endpoint: seg3/u9/D (v) checked with leading edge of 'CLK_W_3'
Beginpoint: seg3/u3/Q (v) triggered by leading edge of 'CLK_W_3'
Path Groups: {CLK_W_3}
Other End Arrival Time 1.029
- Setup 0.153
+ Phase Shift 2.000
- Uncertainty 1000.000

```

# Tempus User Guide

## Analysis and Reporting

```
= Required Time -997.124
- Arrival Time 3.839
= Slack Time -1000.963
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 1.202
= Beginpoint Arrival Time 1.202

Pin Cell Slew Incr Delay Arrival Time
seg3/u3/CK - 0.094 - - 1.203
seg3/u3/Q DFF 0.340 0.073 0.368 1.571
seg3/u4/A -> BUF 0.341 0.002 0.011 1.581
seg3/u4/Y BUF 0.003 0.000 0.165 1.746
seg3/u5/A INV 0.005 0.000 0.003 1.749
seg3/u5/Y INV 0.516 0.120 0.225 1.975
seg3/u6/A INV 0.627 0.160 0.317 2.292
seg3/u6/Y INV 0.913 0.025 0.634 2.926
seg3/u7/A BUF 0.914 0.005 0.008 2.934
seg3/u7/Y BUF 0.655 0.051 0.492 3.426
seg3/u7_a/A BUF 0.659 0.061 0.121 3.547
seg3/u7_a/Y BUF 0.003 0.000 0.269 3.817
seg3/u8_A BUF 0.003 0.000 0.001 3.818
seg3/u8/Y BUF 0.042 0.008 -0.009 3.809
seg3/u9/D DFF 0.071 0.009 0.031 3.839

```

## Using report\_timing with lumpedOnNet Option

By default, Tempus calculates the delay, including SI for each arc, and if you choose to view the SI delay separately, it will show as annotated to each arc. Alternatively, you can choose to annotate the SI delay solely to the net. You can use the following command to lump the delta delay all onto the net:

```
tempus > set_si_mode -delta_delay_annotation_mode lumpedOnNet
tempus > report_timing

```

```
Path 1: VIOLATED Setup Check with Pin seg3/u9/CK
Endpoint: seg3/u9/D (v) checked with leading edge of 'CLK_W_3'
Beginpoint: seg3/u3/Q (v) triggered by leading edge of 'CLK_W_3'
Path Groups: {CLK_W_3}
Other End Arrival Time 1.029
- Setup 0.153
+ Phase Shift 2.000
- Uncertainty 1000.000
= Required Time -997.124
- Arrival Time 3.839
= Slack Time -1000.963
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 1.202
= Beginpoint Arrival Time 1.202

```

# Tempus User Guide

## Analysis and Reporting

---

| Pin          | Cell | Slew  | Incr Delay | Delay  | Arrival Time |
|--------------|------|-------|------------|--------|--------------|
| seg3/u3/CK   | -    | 0.094 | -          | -      | 1.202        |
| seg3/u3/Q    | DFF  | 0.340 | 0.000      | 0.295  | 1.498        |
| seg3/u4/A -> | BUF  | 0.341 | 0.075      | 0.084  | 1.581        |
| seg3/u4/Y    | BUF  | 0.003 | 0.000      | 0.165  | 1.746        |
| seg3/u5/A    | INV  | 0.005 | 0.000      | 0.003  | 1.749        |
| seg3/u5/Y    | INV  | 0.516 | 0.000      | 0.106  | 1.855        |
| seg3/u6/A    | INV  | 0.627 | 0.279      | 0.437  | 2.292        |
| seg3/u6/Y    | INV  | 0.913 | 0.000      | 0.609  | 2.901        |
| seg3/u7/A    | BUF  | 0.914 | 0.030      | 0.033  | 2.934        |
| seg3/u7/Y    | BUF  | 0.655 | 0.000      | 0.441  | 3.375        |
| seg3/u7_a/A  | BUF  | 0.659 | 0.113      | 0.172  | 3.547        |
| seg3/u7_a/Y  | BUF  | 0.003 | 0.000      | 0.269  | 3.816        |
| seg3/u8/A    | BUF  | 0.003 | 0.000      | 0.001  | 3.818        |
| seg3/u8/Y    | BUF  | 0.042 | 0.000      | -0.017 | 3.801        |
| seg3/u9/D    | DFF  | 0.071 | 0.017      | 0.038  | 3.839        |

### Using report\_delay\_calculation -si Command

You can use the `report_delay_calculation` command to report the delay calculation information for a cell or net timing arc. When `-si` parameter is specified, the software will report the SI components, including incremental delay. For example,

```
tempus > report_delay_calculation -from seg3/u5/Y -to seg3/u6/A -si
```

```

From pin : seg3/u5/Y
Cell : INV
Library : cell_w
To pin : seg3/u6/A
Cell : INV
Library : cell_w
Base or SI: SI delay
Delay type: net
```

```

RC Summary for net seg3/n5
```

```

Number of capacitance : 17
Net capacitance : 0.293534 pF
Total rise capacitance: 0.320849 pF
Total fall capacitance: 0.320780 pF
Number of resistance : 17
Total resistance : 567.671387 Ohm
```

```

SI information for rise
```

```

Victim timing window : 1.3501 ns 2.3798 ns CLK_W_3
```

```

Attacker Status Direction Slew Coupling cap Fanout Voltage Bump
Timing window Clock
```

# Tempus User Guide

## Analysis and Reporting

```
seg1/n5 PE Fall 0.331500 ns 0.138787 pF 1 1.08 v 0.183600
v 1.6194 ns 2.7454 ns CLK_W_1
seg2/n5 SY Fall 0.358000 ns 0.138787 pF 1 1.08 v 0.183600
v 1.4860 ns 2.5635 ns CLK_W_2

SI information for fall

Victim timing window : 1.7978 ns 3.0247 ns CLK_W_3

Attacker Status Direction Slew Coupling cap Fanout Voltage Bump
Timing window Clock

seg1/n5 PE Rise 0.834500 ns 0.138787 pF 1 1.08 v 0.108000
v 1.4145 ns 2.2281 ns CLK_W_1
seg2/n5 SY Rise 0.589833 ns 0.138787 pF 1 1.08 v 0.140400
v 1.2541 ns 1.9973 ns CLK_W_2

Model types

Net Term Cell Model

seg3/n5 seg3/u5/Y INV NLDM
seg3/n5 seg3/u6/A INV NLDM
seg1/n5 Y INV NLDM
seg2/n5 Y INV NLDM

Rise Fall

Incr Net Delay : 0.279000 ns 0.053500 ns
Net delay : 0.436700 ns 0.155600 ns
From pin transition time: 0.515600 ns 0.215500 ns
To pin transition time : 0.626900 ns 0.272200 ns

Abbreviative attacker status and filtered reason:

SY: Synchronous
INF: Attacker with infinite timing window
PE : Filtered due to physical exclusion
UF : Filtered by user
SB : Filtered due to small bump
TA : Filtered because of timing window does not overlap
LC : Filtered due to logical correlation
ACT: Attacker is Active.
```

**Note:** To access attacker information, you need to make the `set_si_mode -enable_delay_report to true` setting.

### Using report\_noise -delay {max | min} Command

You can use the `report_noise -delay max -nets netName` command to report incremental delay on nets. For example,

```
tempus > report_noise -delay max -nets seg3/n5

Noise Delay Report (for view default_emulate_view)
```

# Tempus User Guide

## Analysis and Reporting

```
Generated: Mon Jul 13 09:46:16 2015
```

```

Report Options:

-type delay
-net seg3/n5
-threshold 10 (ps)
-delay max
[-style default]
[-sortBy IncrDelay]

Type IncrDelay(ps) Delay(NomMax/Max) VictimNet
R 279.000 157.700/436.700 seg3/u6/A {seg3/n5}
Constituents:
 Offset(ps) Slew(ps) Xcap(fF) Edge Status Net
Cpl: - 331.500 138.787 F PE seg1/n5
Cpl: -65.000 358.000 138.759 F SY seg2/n5

Type IncrDelay(ps) Delay(NomMax/Max) VictimNet
F 53.500 102.100/155.600 seg3/u6/A {seg3/n5}
Constituents:
 Offset(ps) Slew(ps) Xcap(fF) Edge Status Net
Cpl: - 834.500 138.787 R PE seg1/n5
Cpl: 395.000 589.833 138.759 R SY seg2/n5

```

## Using Signal Integrity Attributes with `get_property` Command

Signal Integrity information can be reported using SI attributes for both graph-based as well as path-based analysis. These attributes can be queried using the `get_property` command.

You need to enable SI attributes by setting the `timing_report_enable_si_debug` global variable to `true`.

The use model for using SI attributes for reporting is given below:

For path-based analysis, the SI attributes can be reported as follows:

The `report_timing -retime option -collection` command creates a property called `timing_points` for each corresponding stage of the path. The `timing_points` property is a pointer that contains information for each stage. Each timing point has a property called `si_object`, which contains the victim information.

The following example shows a query on signal integrity attributes for timing points:

```
set path [report_timing -through $pin -retime path_slew_propagation -collection]
set tps [get_property $path timing_points]
foreach in_collection point $tps {
 set x [get_property $point si_object]
 set vic_name [get_property $x hierarchical_name]
 puts "victim: $vic_name"
 set num_agg [get_property $x num_attackers]
```

```
puts "Number of attackers: $num_agg"
set aggs [get_property $x attackers]
puts "attackers info:"
foreach_in_collection current_agg $aggs {
 set active [get_property $current_agg is_active]
 set state [get_property $current_agg state]
 set agg_name [get_property $current_agg hierarchical_name]
 puts "$agg_name $active $state"
}
}
```

The above use model is valid for graph-based analysis also. Additionally, for GBA the SI attributes can be reported on the receiver input pin directly instead of timing points. Each pin has a property `si_victim`, which contains the victim SI information. The following example shows the usage of `si_victim`, where the number of active attackers are being printed for both rise and fall transitions in maximum delay analysis:

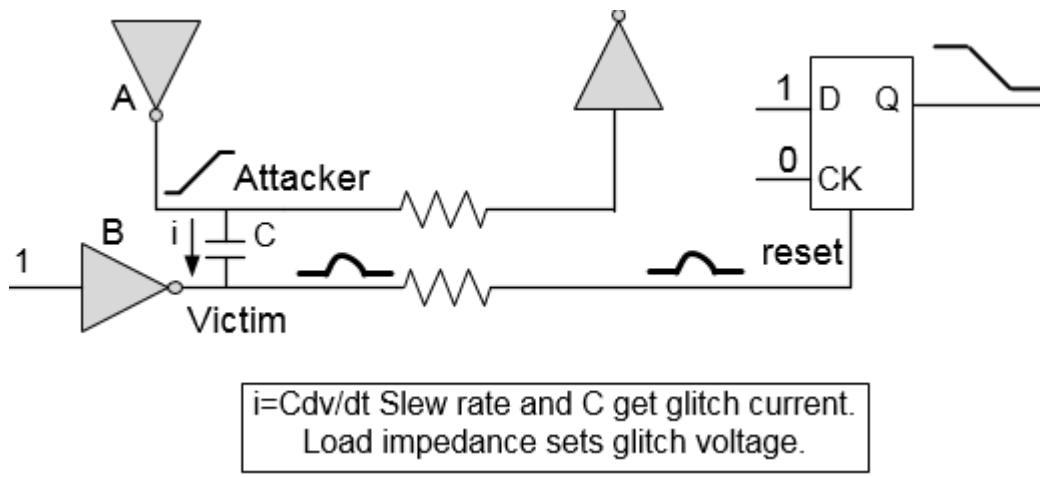
```
set vicArray [get_property [get_pins $pin] si_victim -view $view]
foreach_in_collection vic $vicArray {
 set view_type [get_property $vic view_type]
 set tran [get_property $vic transition]
 if {$view_type == "Late"} {
 if {$tran == "Rise"} {
 set num_att [get_property $vic num_active_attackers]
 puts "number of active attackers for rise transition: $num_att"
 }
 if {$tran == "Fall"} {
 set num_att [get_property $vic num_active_attackers]
 puts "number of active attackers for fall transition: $num_att"
 }
 }
}
```

For more information on signal integrity (SI) attributes/properties, refer to [get\\_property](#) command in the *Tempus Text Command Reference Guide*.

#### 4.2.7 SI Glitch Analysis - An Overview

Glitch analysis is based on calculating the worst-case glitch waveforms on each net in the design. SI can result in functional failures due to glitch. This is illustrated in the diagram below:

**Figure 4-41 Glitch due to SI resulting in functional failure**



In this example, coupling from attacker A causes a significant glitch on the reset signal so that it resets the flip-flop and destroys the stored logic state. This problem does not show up in logic simulation, formal verification, or timing analysis. Isolating the source of a noise-induced functional problem can be very time consuming and may take a few months of debugging. Tempus calculates and reports the glitch induced in the design due to these scenarios.

Tempus performs glitch analysis by analyzing each potential victim net from primary inputs to primary outputs. The analysis of each net is based on calculating the worst-case glitch waveforms that can occur at the input pins of each of the victim net's receiving cells. The receiver output peak of each receiving cell instance is computed by analyzing the propagation of the glitch waveform from the receiver's input through the first stage in the receiver. Tempus assumes all worst-case crosstalk scenarios are possible unless timing windows are used.

Tempus checks for glitch receiver peak and propagates the glitch at a single stage and checks to see if the noise glitch exceeds the failure criteria. This greatly reduces the number of potential false alarms as it utilizes the inherent glitch filtering properties of CMOS logic.

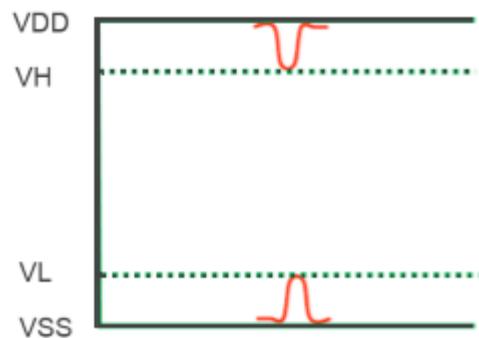
### Glitch Type

There are two types of glitch depending on victim state and attacker switching direction.

| Glitch Type                  | Victim State | Attacker Switching |
|------------------------------|--------------|--------------------|
| Glitch over ground rail (VL) | Low          | Rising             |
| Glitch under power rail (VH) | High         | Falling            |

These are illustrated in the figure below.

**Figure 4-42 VL and VH Glitch type**

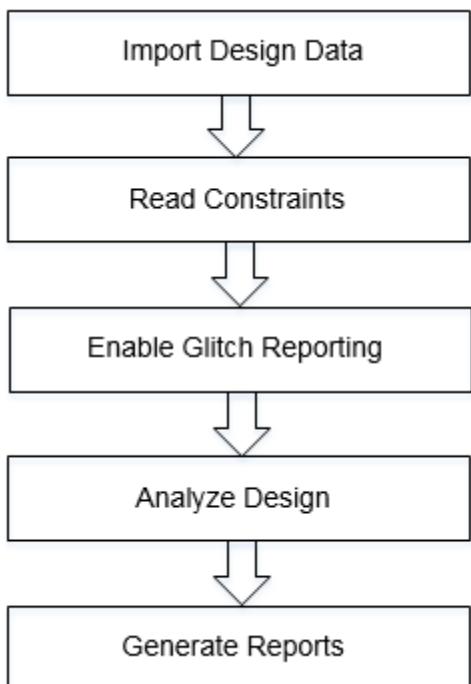


Glitch between ground and power rail voltages can cause logic failure, if they exceed logic thresholds of the technology.

#### 4.2.8 SI Glitch Analysis Flow

The following figure shows the glitch noise analysis flow:

**Figure 4-43 SI Glitch Analysis Flow**



## Sample Glitch Analysis Script

The following sample script shows the steps to perform glitch analysis:

- Load the timing libraries (.lib) and noise data (cdB, ECSV-Noise, or CCS-Noise):  
`read lib typ.lib -cdb typ.cdB`
- Schedule reading of a structural, gate-level verilog netlist:  
`read verilog top.v`
- Set the top module name, and check for consistency between Verilog netlist and timing libraries:  
`set top module dma_mac top`
- Set the proper technology node:  
`set design mode -process 16`
- Load the timing constraints:  
`read sdc constraints.sdc`
- Load the cell parasitics:  
`read spef top.spef.gz`
- Enable signal integrity (SI) analysis.  
`set delay cal mode -siAware true`
- Perform glitch analysis and enable the glitch reports to be generated  
`set si mode -enable_glitch_report true`
- Enable glitch propagation:  
`set si mode -enable_glitch_propagation true`
- Perform glitch analysis:  
`update glitch`
- Generate SI glitch report:  
`report noise-txtfile glitch.txt`

### 4.2.9 Understanding SI Glitch Analysis

In order to perform crosstalk functional failure analysis, Tempus first calculates the worst-case glitch that occurs on each victim net when it is not transitioning. This worst-case glitch is created from the worst-case combination of all allowable attackers where the attackers are nets that are coupled to the victim. The set of allowable attackers is determined from both timing and logic relationships between the victim and attacker nets. For example, attackers

whose timing is such that they do not overlap are not combined. A unique worst-case glitch is calculated for each receiver (fanout) of the victim. Once the worst-case glitch is determined, Tempus performs a number of additional steps to determine if this glitch is harmful to the functionality of the design being analyzed.

There are three key components of SI Glitch analysis:

- Glitch Waveform Computation
- Glitch Failure Detection
- Glitch Propagation

### **Glitch Waveform Computation**

For more details, refer to section on Waveform Computation.

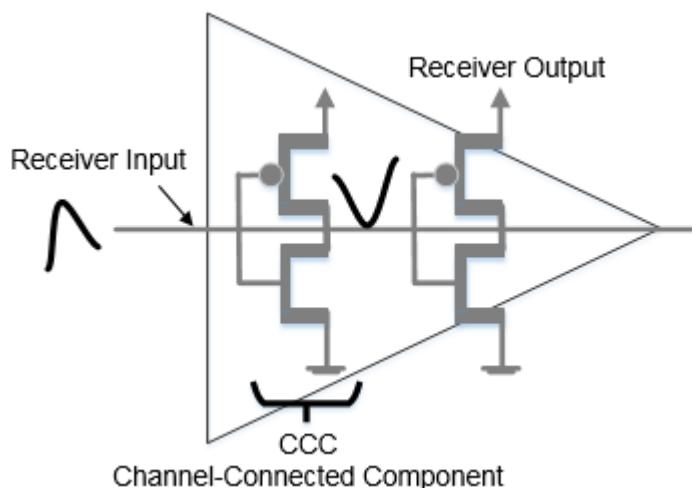
### **Glitch Failure Detection**

When worst glitch is computed, Tempus checks if a glitch can result in failure. Tempus uses two criteria for glitch failure by measuring the glitch at input and output of receiver cell, as shown in the figure below.

- Receiver Input Peak (RIP) Check
- Receiver Output Peak (ROP) Check

These are illustrated in the diagram below.

**Figure 4-44 Glitch representation at input and output of receiver**



### ***Receiver Input Peak (RIP) Check***

If a net has a receiver without noise data, then the receiver input peak (RIP) is compared against:

```
set_si_mode -input_glitch_threshold (or -input_glitch_vh_threshold -
input_glitch_vl_threshold) value
```

or

```
set_glitch_threshold -failure_point input -threshold_type failure -value
float
```

In this case, failure can occur if RIP is greater than the tolerance threshold value.

### ***Receiver Output Peak (ROP) Check***

Tempus has the capability to compute glitch impact one step inside the cell, that is, at the output of the first Channel Connected Component (CCC) in the cell. This is known as ROP (Receiver Output Peak) analysis. Since CMOS logic acts as a low pass filter, it attenuates the glitch, so that the glitch checked at the output of the first CCC may be less than the glitch at the input of the cell. Using the ROP to determine failure may reduce the number of violation counts.

**Note:** In some cases, the ROP may be more than the RIP. The ROP can be as high as VDD in case of a full rail RIP, so any result close to the VDD ROP can be considered reasonable.

If a net has receiver with noise data (cdB or ECSV-Noise or CCS-Noise), then Tempus propagates the glitch one stage and calculates the Receiver Output Peak (ROP). If the arc is from input to output of the cell, then failure will occur if ROP is greater than the values specified using the following command:

```
set_si_mode -receiver_peak_limit (or -receiver_clk_peak_limit, -
receiver_latch_peak_limit) double
```

or

```
set_glitch_threshold -failure_point last -pin_type {clock | data | latch | all}
-threshold_type failure -value float
```

If the arc is from an input to an internal node of a combinational cell, by default failure will occur if ROP is greater than 30% of the vdd value or if it is greater values specified using the following command:

```
set_glitch_threshold -failure_point input -pin_type {clock | data | latch | all} -
threshold_type failure -value float
```

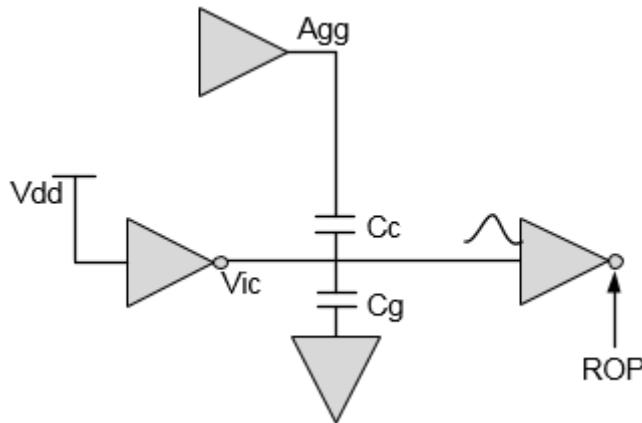
If the arc is from an input to an internal node of a sequential cell, then failure will occur if ROP is greater than the value specified using the following command:

```
set_si_mode -receiver_latch_peak_limit double
```

or

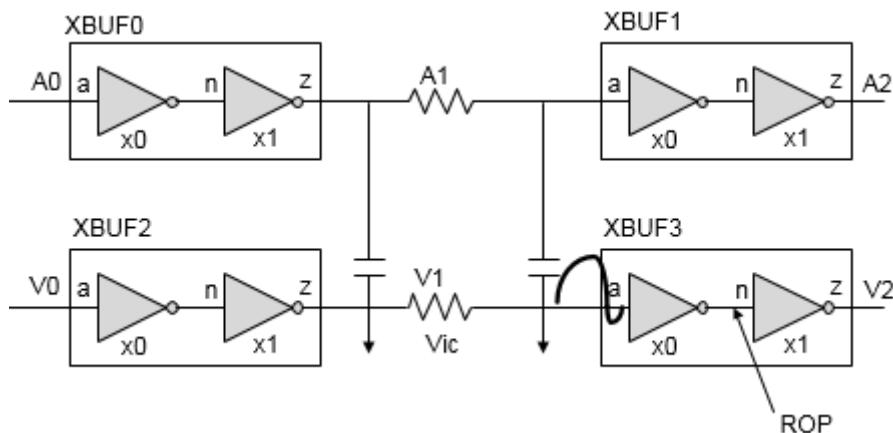
```
set_glitch_threshold -failure_point last -pin_type latch -threshold_type failure
-value float
```

**Figure 4-45 ROP (Receiver Output Peak) for single stage cell**



In case of single stage receiver cell, as shown in the figure above for Victim Vic, ROP will be performed at the output of receiver cell.

**Figure 4-46 ROP (Receiver Output Peak) for multi-stage cell**



While in case of multi-stage cell, as shown in the figure above for Victim V1, ROP will be performed at the output of the first stage of receiver cell XBUF3.

ROP is computed using the VIVO (Voltage In Voltage Out) current tables from the noise library (cdB), ECSM-N library, or CCS-N library. VIVO tables are characterized for single CCC. Tempus uses VIVO information, stored for the corresponding input pins, for ROP computation.

**Figure 4-47 VIVO model**

```

.bbox A2LVLD_X1N_A9TS_C16
+ Y VDD VNW VFN VSS A EN
bb_is_cell
bb_index_vector -val {0.000000 0.080000 0.160000 0.240000 0.320000 0.400000 0.480000 0.560000 0.640000 0.720000 } VI_INDEX0
bb_index_vector -val {0.000000 0.064444 0.128889 0.193333 0.257778 0.322222 0.386667 0.451111 0.515556 0.580000 } VI_INDEX1
bb_vivo_template -inVdd 0.72 -outVdd 0.58 -vi VI_INDEX0 -vo VI_INDEX1 \
 -current {
 4.222e-05 4.104e-05 3.981e-05 3.849e-05 3.7e-05 3.52e-05 3.272e-05 2.853e-05 1.965e-05 -1.432e-08
 1.382e-05 1.329e-05 1.276e-05 1.222e-05 1.147e-05 1.107e-05 1.038e-05 9.413e-06 7.305e-06 -1.37e-08
 2.155e-06 2.016e-06 1.893e-06 1.773e-06 1.657e-06 1.543e-06 1.426e-06 1.296e-06 1.085e-06 -1.234e-08
 1.304e-07 1.088e-07 9.935e-08 9.053e-08 8.219e-08 7.423e-08 6.649e-08 5.84e-08 4.703e-08 -1.373e-08
 5.563e-09 -3.271e-08 -3.782e-08 -4.156e-08 -4.529e-08 -4.92e-08 -5.34e-08 -5.792e-08 -6.293e-08 -7.04e-08
 2.355e-10 -5.674e-07 -6.621e-07 -7.199e-07 -7.731e-07 -8.266e-07 -8.815e-07 -9.384e-07 -9.975e-07 -1.059e-06
 1.825e-11 -4.061e-06 -4.973e-06 -5.32e-06 -5.567e-06 -5.788e-06 -5.998e-06 -6.205e-06 -6.41e-06 -6.615e-06
 9.258e-12 -9.602e-06 -1.346e-05 -1.474e-05 -1.533e-05 -1.573e-05 -1.607e-05 -1.638e-05 -1.668e-05 -1.696e-05
 -9.466e-11 -1.364e-05 -2.105e-05 -2.422e-05 -2.545e-05 -2.604e-05 -2.643e-05 -2.674e-05 -2.701e-05 -2.725e-05
 -9.321e-11 -1.598e-05 -2.344e-05 -3.013e-05 -3.217e-05 -3.3e-05 -3.339e-05 -3.364e-05 -3.392e-05 -3.398e-05
 } A2LVLD_X1N_A9TS_C16_VIVO1
bb_set_arc -arctype min_rise -from (A) -to (ny) -dir in -type inv -loadcap { 1.62937e-15 } -millercap { 3.93743e-16 }
-stimuli { A 1 EN 1 ln_85 0 } A2LVLD_X1N_A9TS_C16_VIVO1

```

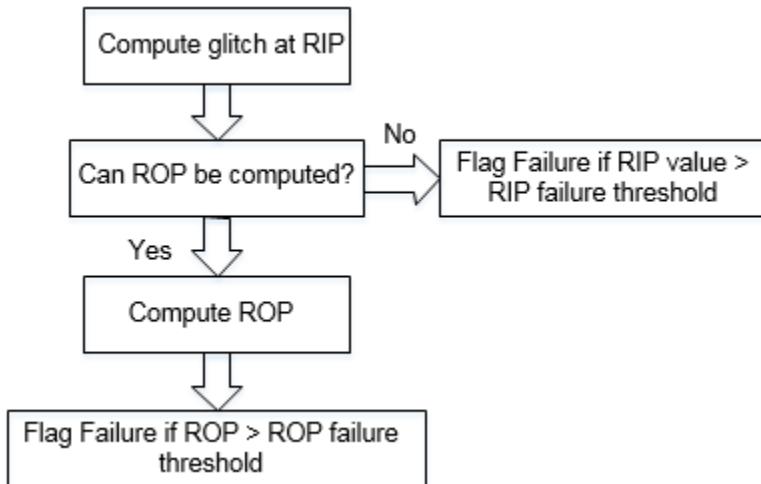
Each VIVO table is given a name

VIVO index values  
for input/output  
voltage

## Glitch Failure Flow

The Glitch Failure Flow diagram is given below.

**Figure 4-48 Glitch Failure Flow**



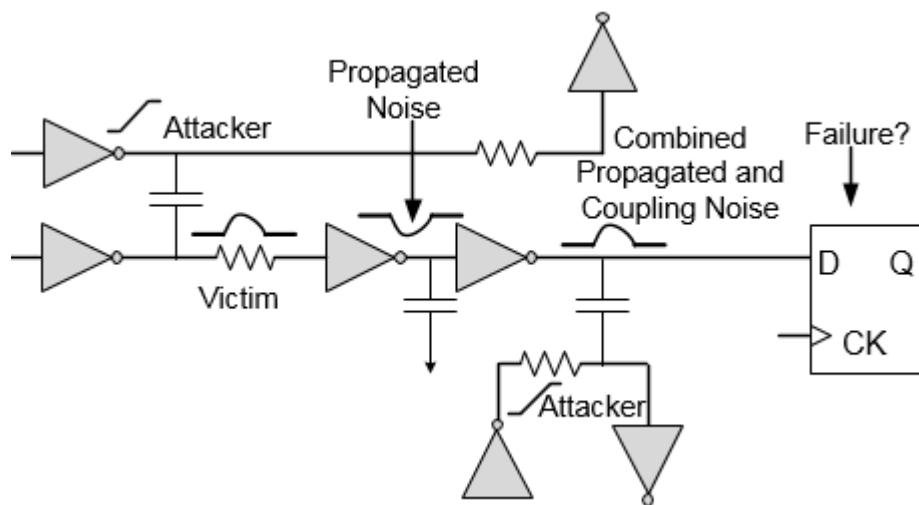
## Glitch Propagation

Tempus performs glitch propagation that:

- Allows glitch to propagate and combine with coupling glitch noise along the path.
- Models driver weakening due to glitch coming at the input pin. Due to glitch at driver input, the current capability of the driver reduces leading to larger glitch at the output.

The following figure illustrates the glitch propagation feature:

**Figure 4-49 Glitch Propagation**



Driver weakening is needed for single-stage cells, not for multi-stage cells. In single-stage cells, when ROP is not failing, driver weakening may affect the next stage if the coupling on the next stage is big enough to be kept. In multi-stage cells, there is no coupling internal to the cell so glitch propagation does not need to be taken into account. This is because a non-failing glitch is not likely to propagate to the output of the cell.

### ***Single-Stage Glitch Propagation and Driver Weakening***

If glitch propagation is enabled, Tempus will propagate the glitch through single-stage cells as long as it is not failing. A glitch is deemed failing when the receiver output peak is greater than the threshold value specified using the following command:

```
set_si_mode -receiver_peak_limit (or -receiver_clk_peak_limit
-receiver_latch_peak_limit) -double
```

The failing receiver output peaks are not propagated. Even if the input glitch is not large enough to propagate to the output of the single-stage cell, it weakens the driver such that when the output of the cell is attacked, the coupling noise is much larger. Tempus models driver weakening when glitch propagation is enabled.

Single-stage glitch propagation and driver weakening is enabled automatically when using the `update_glitch` command or by specifying the `set_si_mode -enable_glitch_propagation true` command before using the `report_timing` or `update_timing` commands.

### ***Failure Detection with Enabled Noise Propagation***

Glitch impact computation starts with the first net in the path and traced downstream. A stage is analyzed taking into account the propagated glitch from last stage and the coupling of current stage. You can compare the total glitch seen at the receiver with the failure threshold. If the glitch exceeds failure threshold, then failure is flagged at the receiver and glitch impact from this stage is not propagated further to downstream logic. If the glitch is less than failure threshold, then the glitch impact at this stage is propagated to downstream logic.

#### **4.2.10 Running Detailed SI Glitch Analysis**

To run detailed glitch analysis, you can use the `update_glitch` command. This command performs the following steps in multiple iterations:

##### **Iteration 0**

This iteration is performed to estimate slews on all the nets.

##### **Iteration 1**

- Identify the attackers which are coupled to the victim net.
- Analyze each attacker individually for glitch peak on victim.
- Determine the attackers which can have glitch peak greater than a threshold of victim voltage.
- Estimate the combined impact of all such attackers, assuming infinite timing window, on victim net for glitch.
- Update the arrival times and compute slew values for all the nets in the design.

##### **Iteration 2:**

- Select the victim nets. A net will be reselected if any of the following criteria is met:
  - the net is violating
  - the net has a glitch greater than 30% VDD
  - the net has a glitch smaller than 30%VDD, but is capable of propagating downstream.

Only those nets that are non-violating, have a glitch smaller than 30%VDD and are incapable of propagating, will not be reselected in the 2nd iteration.

- Analyze each attacker individually for glitch peak using slew/timing windows computed above. If the `set_si_mode -use_infinite_TW true` setting has been specified, then infinite timing windows is used in this iteration also.
- Determine the attackers which can produce glitch peak greater than the threshold of victim voltage.
- Identify the attackers which can produce worst impact on the victim net depending on timing window overlap, attacker slew, and other constraints.
- Perform combined simulation with identified attackers and peak aligned to compute the worst glitch at receiver input of victim net.

### Failure Criteria Determination

- If a net has a receiver without noise data, then the receiver input peak (RIP) check is done.
- If a net has receiver with noise data, then the software propagates the noise one stage and receiver output peak (ROP) check is done.

#### 4.2.11 SI Glitch Reporting

You can use the `report_noise` command to generate glitch reports. The `-format` parameter can be used to control the fields to be displayed in a glitch report.

Given below is a sample report generated using the `report_noise` command.

```

Glitch Violations Summary

Number of DC tolerance violations (VH + VL) = 0
Number of Receiver Output Peak violations (VH + VL) = 5
Number of total problem noise nets = 3

Peak(mV) Level TotalCap(fF) TotalArea Width(ps) Vdd (mV) Library
VictimNet
1071.972 VH 232.84 134.35 250.659 1320 ecsmt xv5/A {vic3}
Receiver output peak:
Value ReceiverNet cell_type
1226.940 (396.000) xv5/A (NOR4_F4) [data]
Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd (mV) Edge Status Net
Cpl: 404.669 30.000 22.600 81.242 1320 F ACT attc4
Cpl: 323.200 25.000 19.800 62.494 1320 F ACT attc3
Cpl: 203.782 21.000 16.000 37.496 1320 F ACT attc2
Cpl: 140.000 18.000 14.000 24.998 1320 F ACT attc1

```

## Tempus User Guide

### Analysis and Reporting

---

The report sections and columns are explained below.

The top section Glitch Violations Summary shows a summary of each type of violation count.

- **Peak** value 1071.972 mV is the glitch at the receiver input pin (RIP).
- **Level** can be VH or VL, which shows the type of glitch at receiver input pin. VH is glitch under power rail and VL is glitch over ground rail.
- The pin name xv5/A below **VictimNet** is the receiver input pin having glitch.
- **Receiver output peak** (ROP) value 1226.940 is the glitch at the receiver output and 396.000 is the failure threshold for ROP. NOR4\_F4 is the receiver library cell.
- **Constituents** provide the details of attackers.
  - **Source** column: Reports propagated noise as “Prp”.
  - **Status** column: The Status column shows the abbreviated attacker status and filtered reason, as given below:
    - SY: Synchronous
    - INF: Attacker with infinite timing window
    - PE: Filtered due to physical exclusion
    - UF: Filtered by user
    - SB: Filtered due to small bump
    - TA: Filtered because of timing window does not overlap
    - LC: Filtered due to logical correlation
    - ACT: Attacker is Active.

- The following sample report shows the DC tolerance violations:

```

Glitch Violations Summary :

Number of DC tolerance violations (VH + VL) = 1
Number of Receiver Output Peak violations (VH + VL) = 0
Number of total problem noise nets = 1

Peak(mV) Level TotalCap(fF) TotalArea Width(ps) Vdd(mV) Library
VictimNet
530.496 VH 144.06 83.88 316.247 1080 cell_w u9/D {vic2}
Receiver Input Threshold:
Value (Threshold) ReceiverNet Receiver failure_type
530.496 (433.296) u9/D (DFF) [bbox]
Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd(mV) Edge Status Net
Cpl: 284.770 38.000 24.167 62.007 1080 F ACT u1/n8
Cpl: 245.673 59.000 76.333 68.138 1080 F ACT u3/n8

```

In this report glitch at receiver input pin u9/D is more than the threshold.

## Tempus User Guide

### Analysis and Reporting

- The `report noise`-style extended option can be used to report additional information, such as, timing windows of attackers. A sample report is shown below:

```
tempus > report_noise -style extended

Glitch Violations Summary :

Number of DC tolerance violations (VH + VL) = 0
Number of Receiver Output Peak violations (VH + VL) = 5
Number of total problem noise nets = 3

Peak(mV) Level TotalCap(fF) TotalArea Width(ps) Vdd(mV) Library VictimNet
1071.972 VH 232.84 134.35 250.659 1320 ecsmt xv5/A {vic3}
Receiver output peak:
Value ReceiverNet cell_type
1226.940 (396.000) xv5/A (NOR4_F4) [data]
Stage information:
Cell Pin
Driver : NOR4_F4 xv4/Y
Receiver : NOR4_F4 xv5/A
Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd(mV) Edge Status Net
Cpl: 404.669 30.000 22.600 81.242 1320 F ACT attc4
Cpl: 323.200 25.000 19.800 62.494 1320 F ACT attc3
Cpl: 203.782 21.000 16.000 37.496 1320 F ACT attc2
Cpl: 140.000 18.000 14.000 24.998 1320 F ACT attc1
Timing Windows: (NanoSecs)
MinTime MaxTime Edge Clock Net
0.180 0.184 R NULL vic3
0.023 0.023 F NULL attc4
0.021 0.021 F NULL attc3
0.019 0.019 F NULL attc2
0.018 0.018 F NULL attc1

```

- You can use the `report noise -histogram` parameter to print a violation summary and histogram of RIP/ROP. A detailed glitch report is not printed.

```
tempus > report_noise -histogram -txtfile rip_rop.hist

Glitch Violations Summary :

Number of DC tolerance violations (VH + VL) = 0
Number of Receiver Output Peak violations (VH + VL) = 5
Number of total problem noise nets = 3

Receiver Input Peak Histogram for View : default_emulate_view
Percentage of Receiver Vdd No. of Receivers
[0, 10) 0
[10, 20) 0
[20, 30) 0
[30, 40) 0
[40, 50) 1
[50, 60) 1
[60, 70) 2
[70, 80) 1
```

## Tempus User Guide

### Analysis and Reporting

---

```
[80, 90) 1
[90, 100) 0

Receiver Output Peak Histogram for View : default_emulate_view
Percentage of Receiver Vdd No. of Receivers
[0, 10) 0
[10, 20) 1
[20, 30) 0
[30, 40) 1
[40, 50) 0
[50, 60) 0
[60, 70) 1
[70, 80) 0
[80, 90) 0
[90, 100) 3
```

---

### **Exceptions**

Here are some exceptions:

The following command helps in specifying certain attackers as quiet for the victim net. This implies that the net mentioned in this command will not be considered as an attacker for a particular victim net, or all victim nets.

- set\_quiet\_attacker

You can use the following parameter to specify a list of victim/attacker pairs to be made quiet:

```
set_quiet_attacker -victim string
```

If the `-attacker` parameter is not used, then all the attackers are made silent for the specified victim.

You can use the following parameter to specify the victim/attacker pair to be made quiet:

```
set_quiet_attacker -attacker string
```

If the `-victim` parameter is not used, then the given nets are made silent (as attackers) for all the victim nets in the design.

```
tempus > set_quiet_attacker -attacker attc4 -victim vic3
tempus > update_glitch
tempus > report_noise -nets vic3

Glitch Violations Summary:

Number of DC tolerance violations (VH + VL) = 0
Number of Receiver Output Peak violations (VH + VL) = 0
Number of total problem noise nets = 0

Peak(mV) Level TotalCap(fF) TotalArea Width(ps) Vdd(mV) Library VictimNet
667.128 VH 232.84 81.45 244.167 1320 ectsmt xv5/A {vic3}
Receiver output peak:
```

## Tempus User Guide

### Analysis and Reporting

```
Value ReceiverNet cell type
93.456 (396.000) xv5/Ā (NOR4_F4) [data]
Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd(mV) Edge Status Net
Cpl: 323.224 25.000 19.800 62.494 1320 F ACT attc3
Cpl: 203.808 21.000 16.000 37.496 1320 F ACT attc2
Cpl: 140.030 18.000 14.000 24.998 1320 F ACT attc1
Cpl: 0.000 0.000 22.600 81.242 1320 F UF attc4

```

In the above noise report, status of attacker `attc4` is changed to UF (User Filtered).

■ `set_annotated_glitch`

- The following parameter allows you to specify the port/pin name at which glitch needs to be annotated:

```
set_annotated_glitch -port
```

If a glitch is annotated at the driver output/input port, then it is combined with the coupling of the stage. If a glitch is annotated at the receiver input, then it overrides the glitch at receiver input with the annotated glitch.

- The following parameters allow you to specify the type of glitch waveform (vl or vh) to be annotated:

```
set_annotated_glitch -vl_glitch/-vh_glitch {waveform with time stamp
and voltage value}
```

The `vh_glitch` waveform should also be specified starting from “0”. The software automatically inverts the waveform depending on receiver voltage.

The waveform is specified as PWL with multiple time and voltage points, such as,  
`{time-1 voltage-1 time-2 voltage-2....}`

The `set_annotated_glitch -vl_glitch {0 0 110e-12 0.6 310e-12 0}` command defines glitch as shown below:



For VH, the wave should be defined starting from 0V, the software automatically inverts it when applying according to the victim voltage.

## Tempus User Guide

### Analysis and Reporting

For victim voltage 1V, the `set annotated glitch -vh_glitch {0 0 100e-12 0.6 320e-12 0}` command defines glitch, as shown below:



Here's a sample report:

```
tempus > set_annotated_glitch -vl_glitch {0 0 100e-12 0.1 200e-12 0.4 300e-12 0.3
600e-12 0} -port u9/D
tempus > update_glitch
tempus > report_noise -nets n8

Peak(mV) Level TotalCap(fF) TotalArea Width(ps) Vdd(mV) Library
VictimNet
399.924 VL 144.11 110.00 550.104 1080 cell_w u9/D {n8}
Receiver Input Threshold:
Value (Threshold) ReceiverNet Receiver failure_type
399.924 (431.244) u9/D (DFF) [bbox]
Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd(mV) Edge Status Net
Prp: 399.924 -- -- -- -- -- u8/Y

```

The software overrides the glitch value showing up at RIP with the annotated glitch, as glitch is annotated at the receiver input.

#### set noise lib pin

This command helps in mapping noise properties of a library cell pin to other library cell pins. The software copies the noise properties of a buffer type cell for a macro cell pin, for which a noise property has not been defined.

- `set_noise_lib_pin -from <>`

Specify the `from_library/from_cell/from_pin`.

- `set_noise_lib_pin -to <>`

Specify the `to_library/to_cell/to_pin`.

For example,

```
set_noise_lib_pin -from cell_w_1.6.lib/CLKBUF/A -to memory.lib/MEM_256x32/
A
```

The software will map the noise properties from cdB linked to the specified library, even if there is a difference in voltage values. You must ensure that the voltage values are similar while mapping the properties.

**Table 4-2 Impact of Timing Exceptions on Glitch Analysis.**

| Timing Exceptions            | Impact on Glitch in Infinite Timing Window Flow                                                                                 |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <u>set_case_analysis</u>     | Attacker will be marked as silent, if constant value is input. The net will be analyzed as victim even with the constant value. |
| <u>set_disable_timing</u>    | Impact due to change in attackers slew, if attacker is lying in the fanout path of disable timing.                              |
| <u>set_false_path</u>        | No impact                                                                                                                       |
| <u>set_multicycle_path</u>   | No impact                                                                                                                       |
| <u>set_clock_uncertainty</u> | No impact                                                                                                                       |
| <u>set_clock_groups</u>      | Impact on noise results due to filtering of some attackers.                                                                     |

#### ***Impact of Timing Constraints on Glitch Analysis in Infinite TW Mode***

- set\_input\_transition

This input transition command defined at the input ports, serves as a starting point to compute at slew at all nets. If its not defined, the software uses a fast slew of 5ps at input ports.

- set\_driving\_cell

This command is required to model holding resistance accurately, if there is coupling at the net connected to the input port.

- set\_load

This command sets the load at the output load. This impacts glitch computation on the net connected at the output port.

## **Checking Noise Models**

You can use the check\_noise command before the update\_glitch command to check consistency and completeness of noise models specified for a design. These checks help to ensure that the right set of models are available to perform SI analysis.

## Library Requirement for Glitch Analysis

Tempus supports cdB, ECSM-N, and CCS-N library for glitch analysis. The key advantage of moving to ECSM-N/CCS-N is both use the same library for timing and glitch analysis. The cdB library is used only for glitch analysis and a separate timing library is required for timing analysis.

### 4.2.12 Overshoot-Undershoot Glitch Analysis - An Overview

The Tempus software allows you to perform overshoot and undershoot analysis. The life span of transistors can be described by the overshoot and undershoot values. If a transistor is consistently exposed to a certain percentage of overshoot/undershoot, then it may suffer a reduced life span. Tempus analyzes and generates reports for such cases.

Due to cross-coupled capacitance between interconnects, switching activity of one net (attacker) can induce undesirable noise (glitch) on a static "victim" net. This glitch can be classified into four categories based on the victim state and attackers' switching direction, as shown in the table below.

**Table 4-3 Classification of glitch**

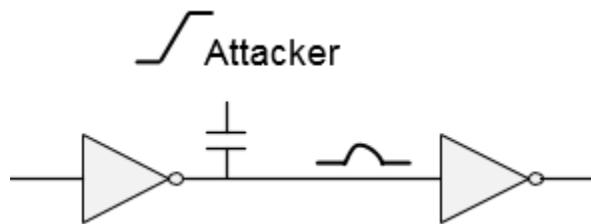
| Glitch Type                    | Victim State | Attacker Switching |
|--------------------------------|--------------|--------------------|
| Glitch over ground rail (VL)   | Low          | Rising             |
| Glitch under power rail (VH)   | High         | Falling            |
| Glitch under ground rail (VLU) | Low          | Falling            |
| Glitch over power rail (VHO)   | High         | Rising             |

The glitch types VL and VH, mentioned in Table 1, can result in functional failures, if the glitch exceeds the logic threshold.

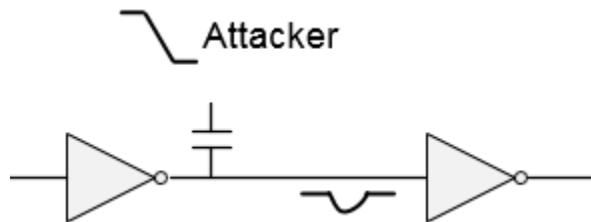
The glitch types, VLU and VHO, are known as undershoot and overshoot, respectively. While VLU (undershoot) and VHO (overshoot) do not cause any immediate silicon failure, exposure to VLU/VHO glitch over a period of time causes degradation of the gate-oxide, which increases the threshold voltage ( $V_{th}$ ), and in turn can cause silicon failure.

The following figures indicate the type of glitch waveform:

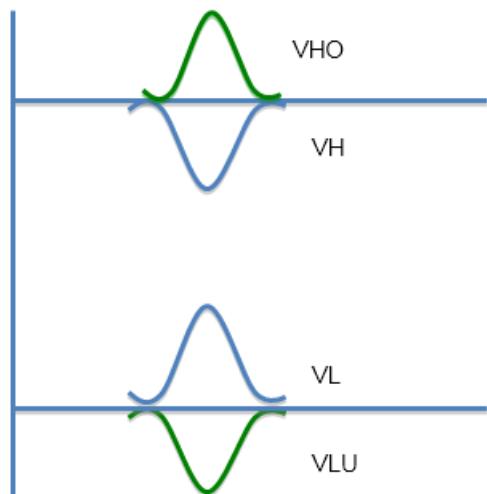
**Figure 4-50 VHO Glitch**



**Figure 4-51 VLU Glitch**



**Figure 4-52 VL, VH, VH, and VHO glitch waveforms**



Tempus performs overshoot-undershoot glitch analysis by analyzing each potential victim net in the design. The analysis of each net is based on calculating the worst-case glitch waveform that can occur at the input pins of each of the victim nets' receivers.

The VLU/VHO type of noise cannot propagate through a transistor, and therefore, Tempus does not perform the glitch propagation and ROP (Receiver Output Peak) computation in overshoot-undershoot analysis.

## Overshoot-Undershoot Analysis Flow

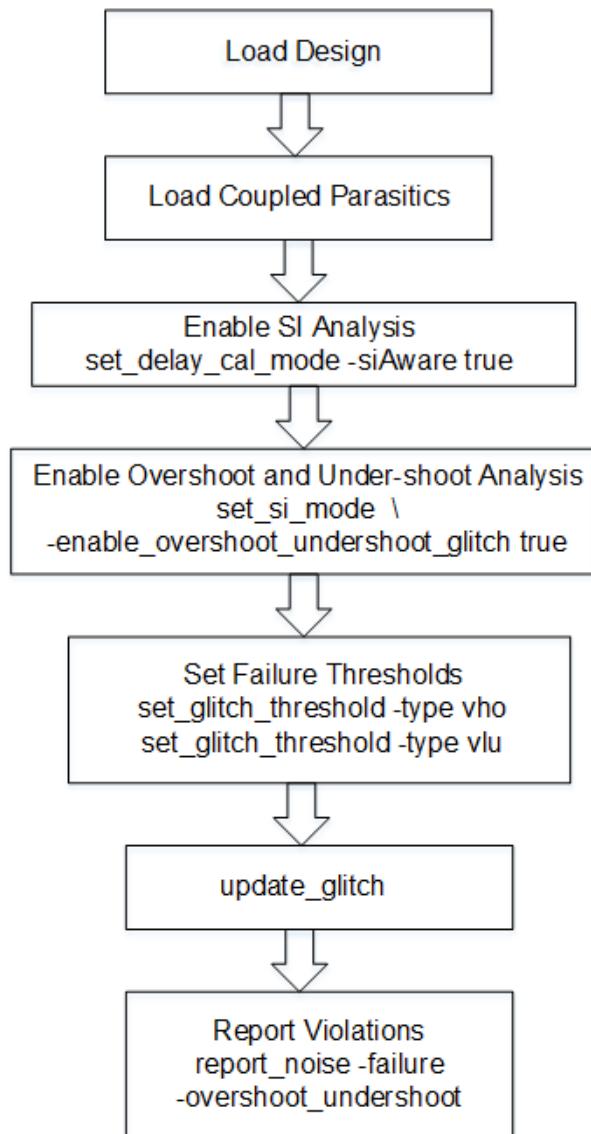
To run overshoot-undershoot analysis, perform the following steps:

- Setup the design for SI analysis.
- Enable overshoot-undershoot analysis:  
`set_si_mode -enable_overshoot_undershoot_glitch`
- Set failure thresholds:  
`set_glitch_threshold -type vho/vlu`
- Analyze overshoot and undershoot glitches:  
`update_glitch`

You can also use the `update_timing` and `report_timing` commands for overshoot/undershoot glitch analysis and reporting.

The following figure shows the overshoot-undershoot analysis flow.

**Figure 4-53 Overshoot-Undershoot Analysis Flow**



## Reporting

The reporting structure of the overshoot-undershoot analysis is similar to that of traditional glitch analysis.

### Example VLU report

---

| Peak(mV) | Level | TotalCap(fF) | TotalArea | Width(ps) | Vdd(mV) | Library | VictimNet           |
|----------|-------|--------------|-----------|-----------|---------|---------|---------------------|
| 381.132  | VLU   | 320.31       | 197.13    | 1034.459  | 1080    | cell_w  | seg1/u6/A {seg1/n5} |

## Tempus User Guide

### Analysis and Reporting

```
Receiver Input Threshold:
Value (Threshold) ReceiverNet Receiver failure_type
381.132 (431.892) seg1/u6/A (INV) [bbox]

Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd(mV) Edge Status Net
Cpl: 215.187 190.000 341.167 139.280 1080 F ACT w3
Cpl: 165.907 223.000 385.667 138.807 1080 F ACT seg3/n5
```

### Example VHO report

```
Peak(mV) Level TotalCap(fF) TotalArea Width(ps) Vdd(mV) Library VictimNet
300.348 VHO 320.31 140.66 936.680 1080 cell_w seg1/u6/A {seg1/n5}

Receiver Input Threshold:
Value (Threshold) ReceiverNet Receiver failure_type
300.348 (431.028) seg1/u6/A (INV) [bbox]

Constituents:
Source Peak(mV) Offset(ps) Slew(ps) Xcap(fF) Vdd(mV) Edge Status Net
Cpl: 196.067 185.000 281.833 139.280 1080 R ACT w3
Cpl: 104.251 197.000 842.500 138.807 1080 R ACT seg3/n5
```

The `report_noise` command parameters, such as, `-failure`, `-format`, `-histogram`, `-nets`, `-style`, `-threshold`, `-txtfile`, and `-view` work with the overshoot-undershoot reporting as well.

For more information, refer to *Tempus Text Command Reference Guide*.

### 4.2.13 Running Overshoot-Undershoot Glitch Analysis - Sample Script

The following is a sample script to run overshoot-undershoot glitch analysis:

- Load the timing libraries (.lib) and characterized data (.cdb):  
`read lib typ.lib -cdb typ.cdb`  
**Note:** This sample script shows the usage of cdb noise data, however, this feature supports CCS-N and ECSV noise models as well.
- Schedule reading of a structural, gate-level verilog netlist:  
`read verilog top.v`
- Set the top module name, and check for consistency between Verilog netlist and timing libraries:  
`set top module top`
- Load the timing constraints:  
`read sdc constraints.sdc`
- Load the cell parasitics:

read\_spef top.spef.gz

- Enable signal integrity (SI) analysis:

set\_delay\_cal\_mode -siAware true

- Enable the Overshoot-Undershoot Analysis:

set\_si\_mode -enable\_overshoot\_undershoot\_glitch true -enable\_glitch\_report true

- Set the failure threshold for the VLU and VHO type glitch:

set\_glitch\_threshold -glitch\_type vlu -value 0.3

set\_glitch\_threshold -glitch\_type vho -value 0.3

Or

- Set the failure threshold globally for both VLU and VHO by using the following command:

set\_si\_mode -input\_glitch\_threshold 0.3

In the above commands, the value specified is the ratio to VDD, that is, failure will occur if the overshoot/undershoot glitch height is greater than the mentioned threshold value times VDD. The default value is 0.4.

If both set\_glitch\_threshold and set\_si\_mode -input\_glitch\_threshold commands have been specified, then the set\_glitch\_threshold command setting will take the precedence.

- Perform overshoot-undershoot glitch analysis:

update\_glitch

- Generate the overshoot/undershoot glitch report:

report\_noise -failure -txtfile vlu\_vho.rpt -overshoot\_undershoot

Here, only the VLU/VHO type of glitches will be printed in the report. For VL/VH type of glitch numbers, you can re-run the report\_noise command without the -overshoot\_undershoot parameter.

## 4.3 Timing Analysis Modes

- 4.3.1 [Overview](#) on page 119
- 4.3.2 [Specifying Timing Analysis Mode Types](#) on page 120
- 4.3.3 [Advanced On-Chip Variation \(AOCV\) Analysis](#) on page 138
- 4.3.4 [Statistical On-Chip Variation \(SOCV\) Analysis](#) on page 165

### 4.3.1 Overview

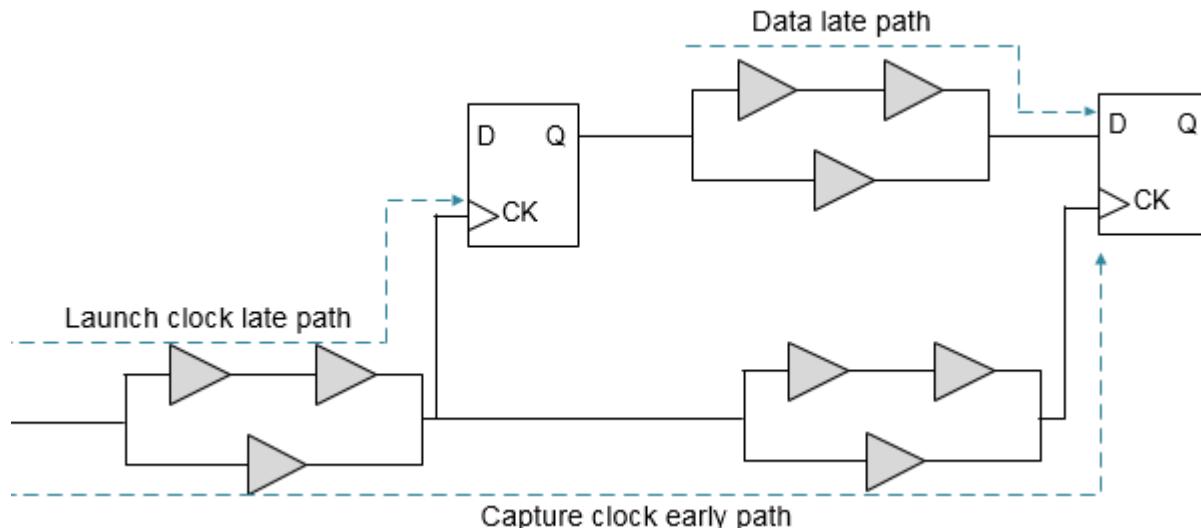
The Tempus software provides different timing analysis modes and accordingly performs calculations for setup and hold checks for each mode. For a better understanding of these modes, it is important to understand the impact of early and late paths in a design, as explained in the following section.

### Definition of Early and Late Paths

Early and late paths are referred to as the shortest and longest paths, respectively, and are used for delay calculations. The early or minimum path delay is the minimum delay through cells and nets in the path. The late or maximum path delay is the maximum delay through cells and nets in the path.

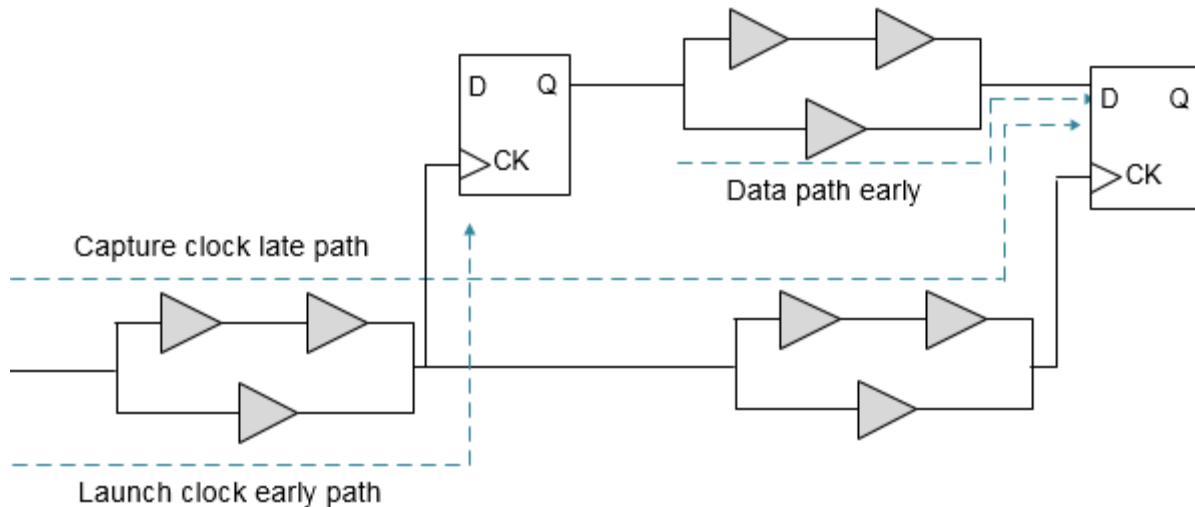
The following figure shows a setup check with late launch clock and early capture clock:

**Figure 4-54 Illustration of Setup Check- Setup check with early and late paths**



The following figure shows a hold check with early launch clock and late capture clock.

**Figure 4-55 Illustration of Hold Check - Hold check with early and late paths**



#### 4.3.2 Specifying Timing Analysis Mode Types

The timing analysis mode types are:

- [Single Timing Analysis Type Overview](#)
- [Best-Case Worst-Case \(BC-WC\) Timing Analysis Mode Overview](#)
- [On-Chip Variation \(OCV\) Timing Analysis Mode Type Overview](#)

##### Single Timing Analysis Type Overview

In this mode, the Tempus software uses a single set of delays (using one library group) based on one set of process, temperature, and voltage conditions.

To set the timing analysis mode as single, you can use the following command:

```
set analysis mode -analysisType single
```

In single analysis mode, only maximum delay values are used for delay calculation. The `-max` options of commands in constraints are used for both minimum and maximum analysis in single analysis mode. For example, the `set annotated delay -max 10` command will be used for both minimum and maximum analysis.

Even with single library group, cell delay can have variation in maximum delay based on `sdf_cond`, timing derates, and other inputs. In single analysis mode, early and late path delays are the minimum and maximum, respectively, of this range of maximum delay.

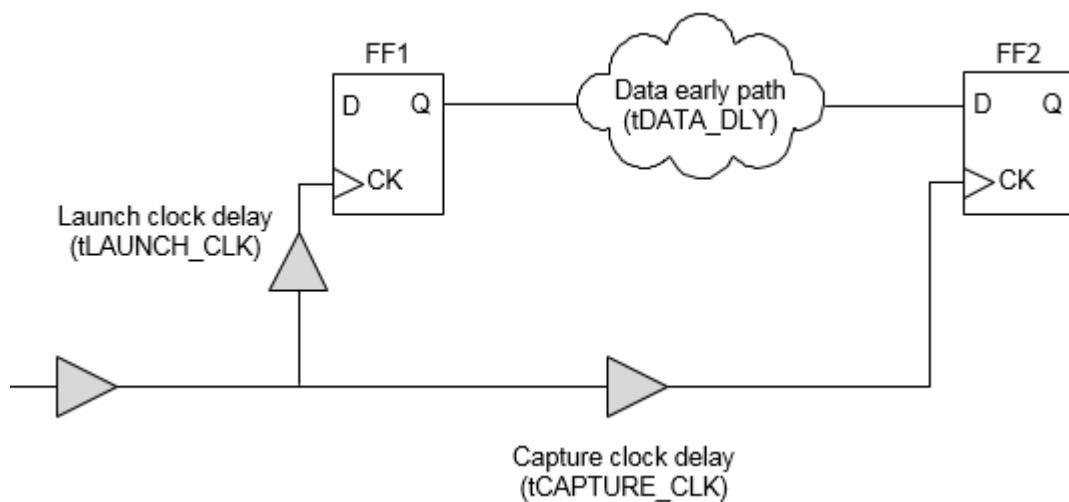
## Single Timing Analysis - Setup and Hold Checks

These are explained in the sections below.

### Setup Check in Single Timing Analysis Mode

For setup check, the software checks the late launch clock and late data paths against early capture clock path.

**Figure 4-56 Setup Check in Single Timing Analysis Mode**

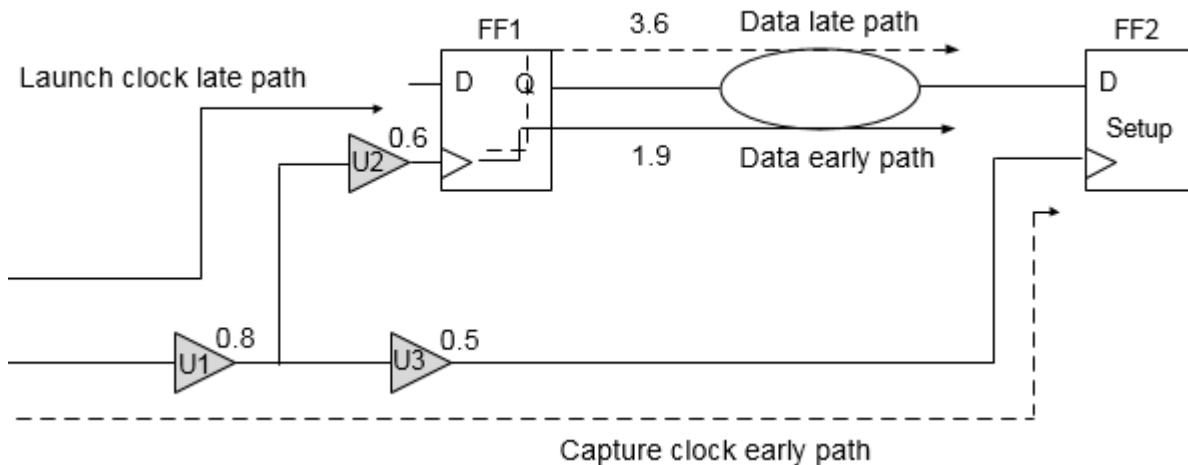


For zero slack value in a setup check, the following condition should be met:

$$\text{launch clock late path } (\text{tLAUNCH\_CLK}) + \text{data clock late path } (\text{tDATA\_DLY}) \leq \text{capture clock early path } (\text{tCAPTURE\_CLK}) + \text{clock period} - \text{setup}$$

[Figure 4-57](#) on page 122 shows the setup check on a path from FF1 to FF2.

**Figure 4-57 Setup Check in Single Timing Analysis Mode**



The software uses a library to calculate the maximum delay. For setup check, the software considers two paths between the two registers, FF1 and FF2 - the late path delay is used to calculate slack during setup check.

The following values are assumed in this example:

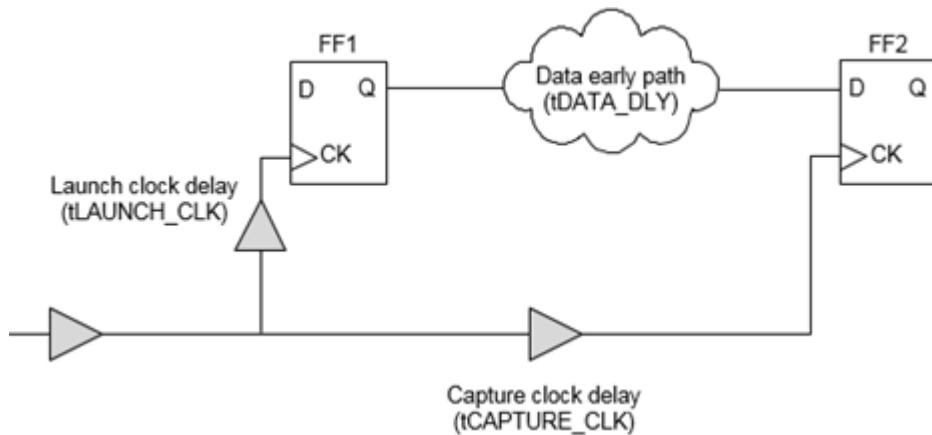
Clock period = 4; Clock source latency = none; Clock mode = Propagated

|                                |                       |
|--------------------------------|-----------------------|
| Data late path delay           | 3.6                   |
| Data early path delay          | 1.9                   |
| Launch clock late path delay   | $0.8 + 0.6 = 1.4$     |
| Capture clock early path delay | $0.8 + 0.5 = 1.3$     |
| Setup                          | 0.2                   |
| Data arrival time              | $1.4 + 3.6 = 5$       |
| Data required time             | $4 + 1.3 - 0.2 = 5.1$ |
| Setup Slack                    | $5.1 - 5 = 0.1$       |

### Hold Check in Single Timing Analysis Mode

For hold check the software compares the early arriving data against the late arriving clock at the endpoint.

**Figure 4-58 Hold Check in Single Timing Analysis Mode**

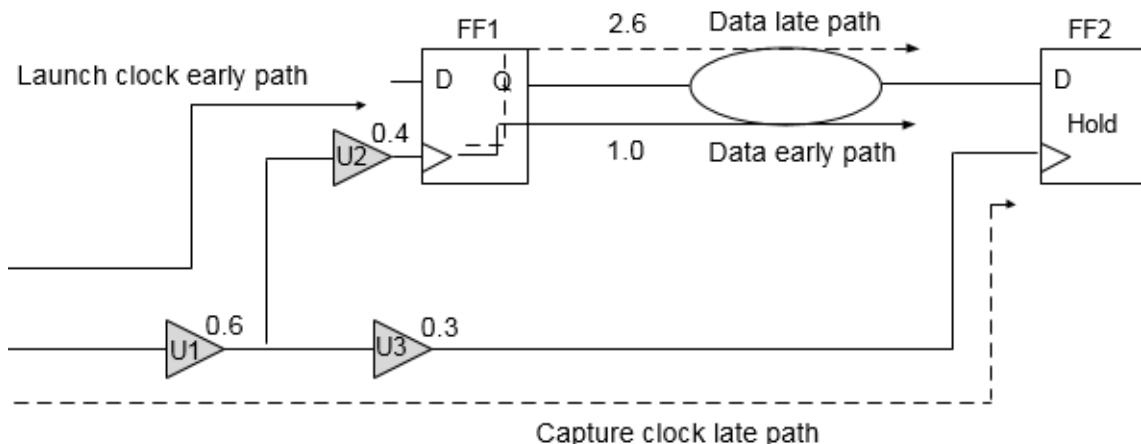


For zero slack value in a hold check, the following condition should be met:

$$\text{launch clock early path } (\text{tLAUNCH\_CLK}) + \text{data early path } (\text{tDATA\_DLY}) \Rightarrow \text{capture clock late path } (\text{tCAPTURE\_CLK}) + \text{hold}$$

The following example shows the hold check on a path from FF1 to FF2.

**Figure 4-59 Hold Check in Single Timing Analysis Mode**



The following values are assumed in this example:

Clock period = 4 ; Clock source latency = none ; Clock mode = Propagated

|                       |     |
|-----------------------|-----|
| Data late path delay  | 2.6 |
| Data early path delay | 1.0 |

|                               |                   |
|-------------------------------|-------------------|
| Launch clock early path delay | $0.6 + 0.4 = 1.0$ |
| Capture clock late path delay | $0.6 + 0.3 = 0.9$ |
| Hold                          | 0.2               |
| Data arrival time             | $1 + 1 = 2$       |
| Data Required Time            | $0.2 + 0.9 = 1.1$ |
| Hold Slack                    | $2 - 1.1 = 0.9$   |

## Performing Timing Analysis in Single Analysis Mode

To perform timing analysis in single analysis mode, you need to perform the following steps:

- Set the analysis mode to single, setup and propagated clock mode:  
`set_analysis_mode -analysisType single -checkType setup`
- Generate the timing reports for setup:  
`report_timing`
- Set the analysis mode to hold:  
`set_analysis_mode -checkType hold`
- Generate the timing reports for hold analysis:  
`report_timing`

Alternatively, you can use the `report_timing -early` command with the `set_analysis_mode -checkType setup` parameter to report hold paths.

## Best-Case Worst-Case (BC-WC) Timing Analysis Mode Overview

In best-case worst-case (BC-WC) analysis mode, the software uses delays from the maximum library group for all the paths during setup checks and minimum library group for all the paths during hold checks. In this mode, the Tempus software considers two operating conditions and checks both operating conditions in one timing analysis run. To set the timing analysis mode as BC-WC, you can use the following command:

`set_analysis_mode -analysisType bcwc`

You can use the `set_clock_latency` constraint to set the source latency for a clock in both ideal and propagated mode for setup and hold checks. You can also use this constraint to set the network latency for an ideal clock. The specified source or network latency affects the

early and late clock paths for both capture and launch clocks for both minimum and maximum operating conditions.

**Note:** The software considers the network latency, specified using the `set_clock_latency -max` (or `-min`) command, for ideal clocks only. In propagated mode, actual clock propagated latency values will be used.

### **BC-WC Timing Analysis Mode - Setup and Hold Checks**

These are explained in the sections below.

#### **Setup Check in BC-WC Mode**

For setup check, the software calculates delay values from the maximum library group for data arrival time, and network delay of both launch and capture clocks (in propagated mode).

The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

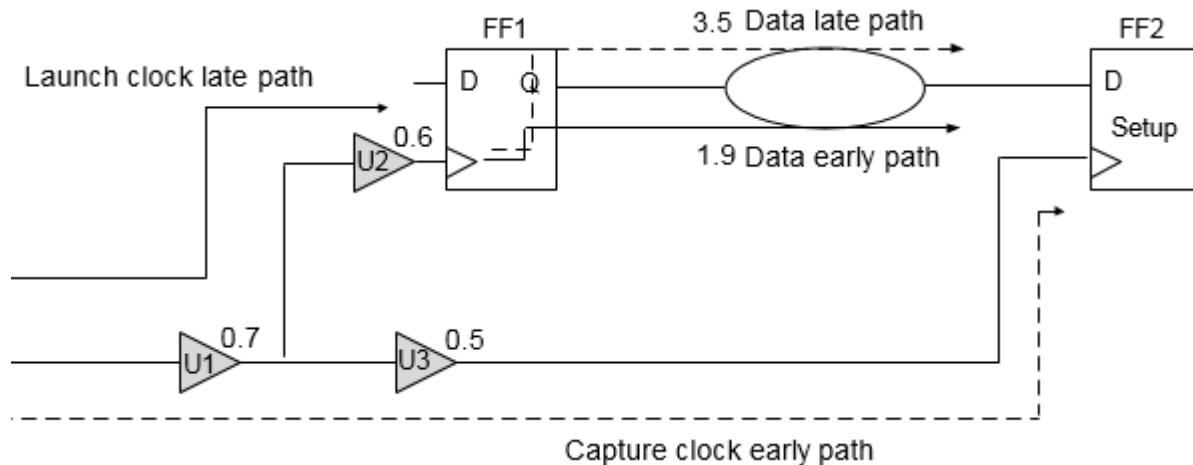
| <b>Clock Path (Operating Condition)</b> | <b>Constraint Used</b>                                  |
|-----------------------------------------|---------------------------------------------------------|
| Launch clock late path (max)            | <code>set_clock_latency -source -late -max value</code> |
| Capture clock early path (max)          | <code>set_clock_latency -source-early -max value</code> |

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

| <b>Clock Path (Operating Condition)</b> | <b>Constraint Used</b>                    |
|-----------------------------------------|-------------------------------------------|
| Launch clock late path (max)            | <code>set_clock_latency -max value</code> |
| Capture clock early path (max)          | <code>set_clock_latency -max value</code> |

The following example shows the setup check on a path from FF1 to FF2.

**Figure 4-60 Setup Check in BC-WC Timing Analysis Mode**



The software uses the max library to scale all delays at WC conditions. The following values are assumed in this example:

Clock period = 4; Clock source latency = none ; Clock mode = Propagated

|                                |                     |
|--------------------------------|---------------------|
| Data late path delay           | 3.5                 |
| Data early path delay          | 1.9                 |
| Launch clock late path delay   | $0.7 + 0.6 = 1.3$   |
| Capture clock early path delay | $0.7 + 0.5 = 1.2$   |
| Setup                          | 0.2                 |
| Data arrival time              | $1.3 + 3.5 = 4.8$   |
| Data Required Time             | $4 + 1.2 - 0.2 = 5$ |
| Setup Slack                    | $5 - 4.8 = 0.2$     |

### HOLD Check in BC-WC Mode

For hold check, the software uses the delay values from the min library for the data arrival time, and network delay of both launch and capture clocks (in propagated mode).

The source latency in both ideal and propagated modes for hold checks is defined in the constraints used by various clock paths as follows:

| Clock Path (Operating Condition) | Constraint Used |
|----------------------------------|-----------------|
|                                  |                 |

|                               |                                                             |
|-------------------------------|-------------------------------------------------------------|
| Launch clock early path (min) | <code>set_clock_latency</code><br>-source -early -min value |
| Capture clock late path (min) | <code>set_clock_latency</code><br>-source -late -min value  |

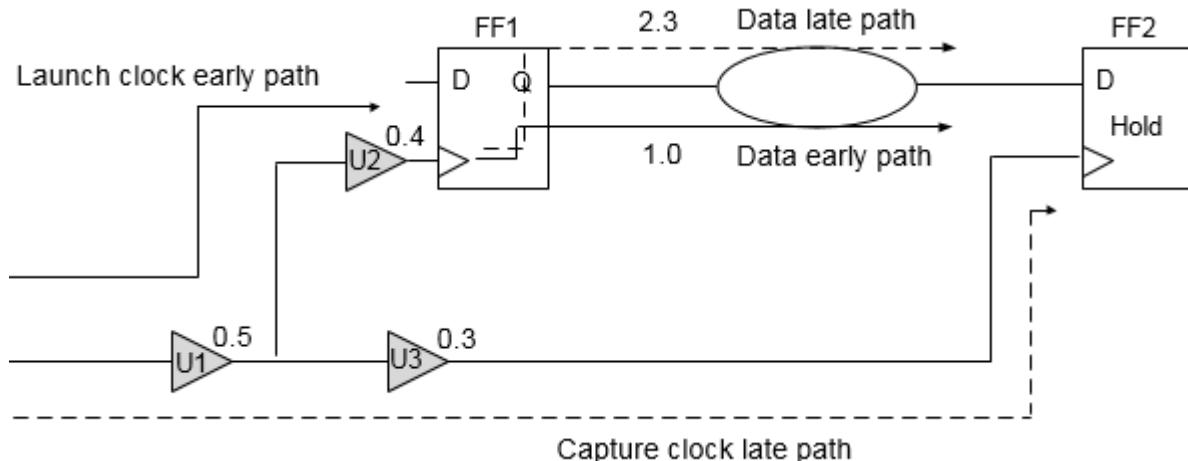
The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

| Clock Path (Operating Condition) | Constraint Used                              |
|----------------------------------|----------------------------------------------|
| Launch clock early path (min)    | <code>set_clock_latency</code><br>-min value |
| Capture clock late path (min)    | <code>set_clock_latency</code><br>-min value |

**Note:** You can also use one library containing two operating conditions in this mode.

The following example shows the hold check on a path from FF1 to FF2.

**Figure 4-61 Hold Check in BC-WC Timing Analysis Mode**



The software uses the min library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock period = 4; Clock source latency = none; Clock mode = Propagated

|                               |                   |
|-------------------------------|-------------------|
| Data late path delay          | 2.3               |
| Data early path delay         | 1.0               |
| Launch clock early path delay | $0.5 + 0.4 = 0.9$ |
| Capture clock late path delay | $0.3 + 0.5 = 0.8$ |
| Hold                          | 0.1               |
| Data arrival time             | $0.9 + 1 = 1.9$   |
| Data required time            | $0.1 + 0.8 = 0.9$ |
| Hold Slack                    | $1.9 - 0.9 = 1$   |

## Performing Timing Analysis in BC-WC Analysis Mode

To perform timing analysis in BC-WC analysis mode, complete the following steps:

- Set the analysis mode to BC-WC:  
`set_analysis_mode -analysisType bcwc -checkType setup`
- Generate the timing reports for setup:  
`report_timing`
- Set the analysis mode to hold and propagated clock mode:  
`set_analysis_mode -checkType hold`
- Generate the timing reports for hold:  
`report_timing`

Alternatively, you can use the `report_timing -early` command with `set_analysis_mode -checkType setup` option to report hold paths.

## On-Chip Variation (OCV) Timing Analysis Mode Type Overview

In on-chip variation (OCV) mode, the software calculates clock and data path delays based on minimum and maximum operating conditions for setup analysis and vice-versa for hold analysis. These delays are used together in the analysis of each check.

The OCV is the small difference in the operating parameter value across the chip. Each timing arc in the design can have an early and a late delay to account for the on-chip process, voltage, and temperature variation.

## OCV Timing Analysis Mode - Setup and Hold Checks

These are explained in the sections below.

### Setup Check in OCV Mode

In OCV mode setup check, the software uses the timing delay values from the late library set and operating conditions for the data and the launch clock network delay. The software uses the delay values from the early library set and operating conditions for the capturing clock network delay assuming that the clocks are set in propagated mode.

**Note:** You can use one library instead of both maximum and minimum libraries, and apply timing derates for performing min/max analysis, respectively.

The source latency in both ideal and propagated modes for setup checks is defined in the constraints used by various clock paths as follows:

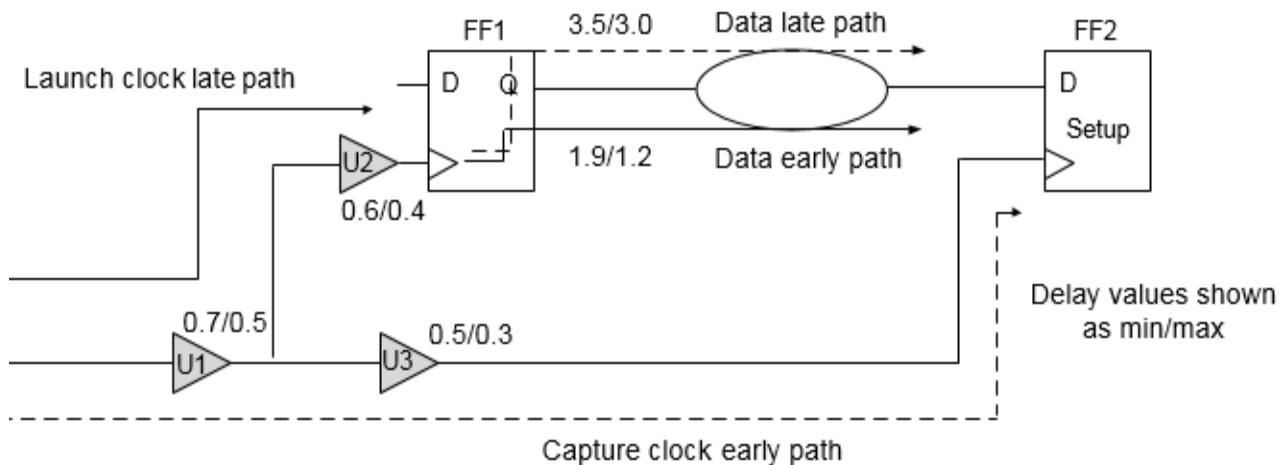
| Clock Path (Operating Condition) | Constraint Used                                                                                                                          |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Launch clock late path (max)     | <code>set_clock_latency</code><br><br><code>-source -late -max value</code><br><br>or <code>set_clock_latency -source -late value</code> |
| Capture clock early path (min)   | <code>set_clock_latency -source -early -min value</code><br><br>or <code>set_clock_latency -source -early value</code>                   |

The network latency in ideal mode for setup checks is defined in the constraints used by various clock paths as follows:

| Clock Path (Operating Condition) | Constraint Used                           |
|----------------------------------|-------------------------------------------|
| Launch clock late path           | <code>set_clock_latency -max value</code> |
| Capture clock early path         | <code>set_clock_latency -min value</code> |

The following example shows the setup check on a path from FF1 to FF2.

**Figure 4-62 Setup Check in OCV Mode**



The software uses the max library for all late path delays and min library for all early path delays.

The following values are assumed in this example:

Clock period = 4; Clock source latency = none; Clock mode = Propagated

|                                      |                       |
|--------------------------------------|-----------------------|
| Data late path delay (max)           | 3.5                   |
| Data late path delay (min)           | 3.0                   |
| Launch clock late path delay (max)   | $0.7 + 0.6 = 1.3$     |
| Capture clock early path delay (min) | $0.5 + 0.3 = 0.8$     |
| Setup                                | 0.2                   |
| Data arrival time                    | $1.3 + 3.5 = 4.8$     |
| Data Required Time                   | $4 + 0.8 - 0.2 = 4.6$ |
| Setup Slack                          | $4.6 - 4.8 = -0.2$    |

### Hold Check in OCV Mode

For OCV hold check, the software uses the timing delay values from the early library set and operating conditions for the data arrival time and launch clock network delay. The software uses delay values from the late library set and operating conditions for the capturing clock network delay assuming that the clocks are set in propagated mode.

The source latency in both ideal and propagated modes for hold checks is defined in the constraints used by various clock paths as follows:

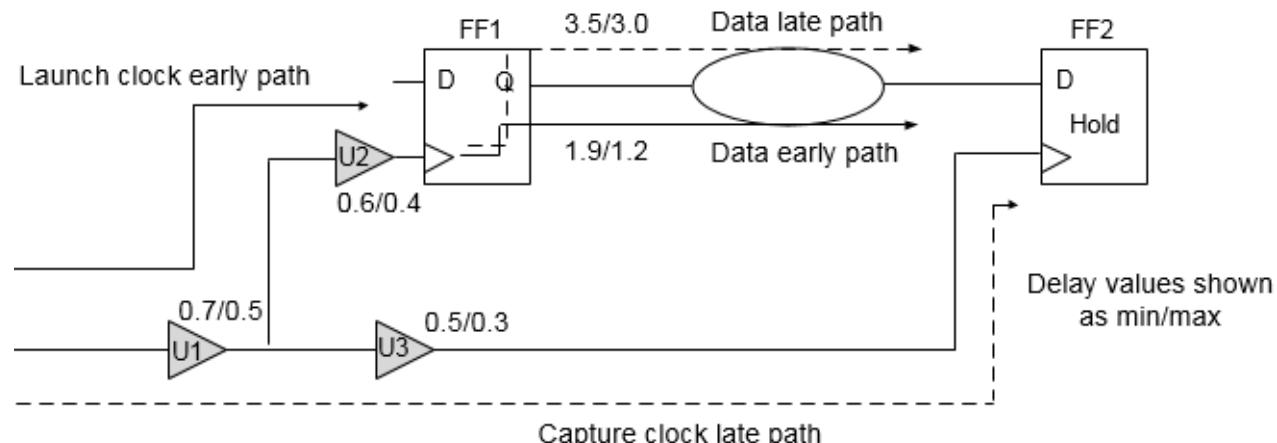
| Clock Path (Operating Condition) | Constraint Used                                                                                                   |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Launch clock early path (min)    | <code>set_clock_latency-source -early -min value</code><br>or <code>set_clock_latency -source -early value</code> |
| Capture clock late path (max)    | <code>set_clock_latency -source -late -max value</code><br>or <code>set_clock_latency -source -late value</code>  |

The network latency in ideal mode for hold checks is defined in the constraints used by various clock paths as follows:

| Clock Path (Operating Condition) | Constraint Used                           |
|----------------------------------|-------------------------------------------|
| Launch clock early path          | <code>set_clock_latency -min value</code> |
| Capture clock late path          | <code>set_clock_latency -max value</code> |

The following example shows the hold check on the path from FF1 to FF2.

**Figure 4-63 Hold Check in OCV Mode**



The software uses the max library to scale all delays at WC conditions and minimum library to scale all delays at BC conditions.

The following values are assumed in this example:

Clock period = 4; Clock source latency = none; Clock mode = Propagated

|                                     |                   |
|-------------------------------------|-------------------|
| Data early path delay (max)         |                   |
| Data early path delay(min)          | 1.2               |
| Launch clock early path delay (min) | $0.5 + 0.4 = 0.9$ |
| Capture clock late path delay (max) | $0.7 + 0.5 = 1.2$ |
| Hold                                | 0.1               |
| Data arrival time                   | $0.9 + 1.2 = 2.1$ |
| Data required time                  | $0.1 + 1.2 = 1.3$ |
| Hold Slack                          | $2.1 - 1.3 = 0.8$ |

## Performing Timing Analysis in OCV Mode

To perform timing analysis in OCV mode, complete the following steps:

- Set the analysis mode to OCV and propagated clock mode:

```
set analysis mode -analysisType onChipVariation
```

- Generate the timing reports for setup:

```
report timing -late
```

- Generate the timing reports for hold:

```
report_timing -early
```

## Using Timing Derates in OCV Analysis Mode

When the set timing derate command is used, the following paths in OCV mode are affected:

| Violations | Data         | Launch Clock  | Capture Clock |
|------------|--------------|---------------|---------------|
| Setup      | -late -data  | -late -clock  | -early -clock |
| Hold       | -early -data | -early -clock | -late -clock  |

## Clock Path Pessimism Removal

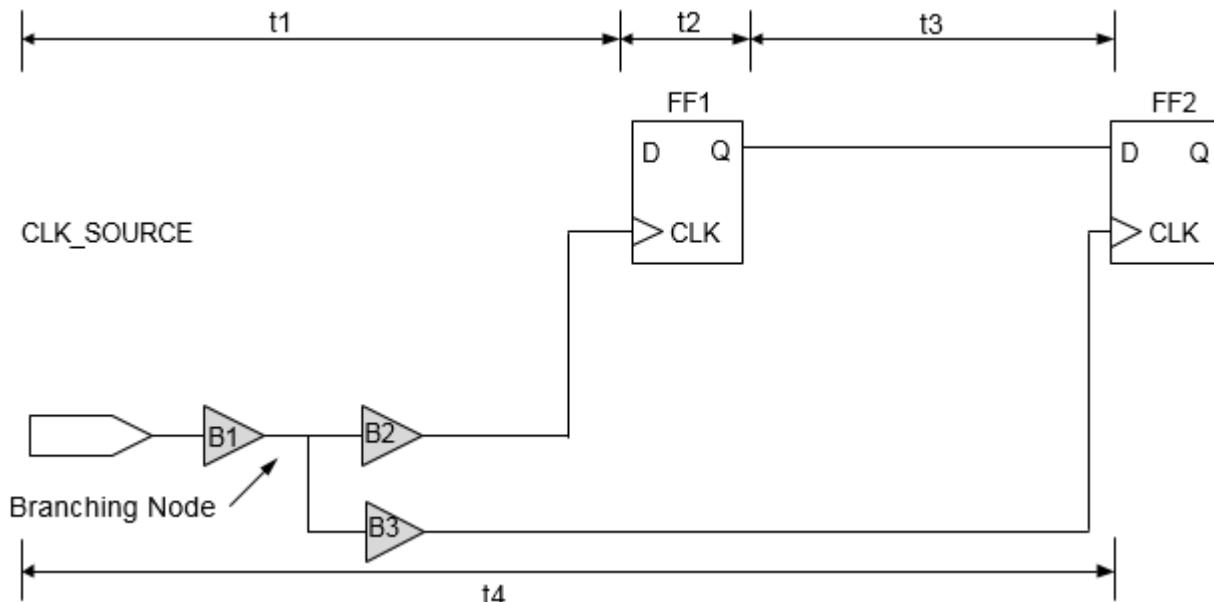
Clock Path Pessimism Removal (CPPR) is the process of identifying and removing pessimism introduced in slack reports for clock paths when launch and capture clock paths have a segment in common.

You can introduce early or late delay variations by using the `set_analysis_mode -analysisType onChipVariation` command or by using the `set_timing_determinate` command for early and late clock paths. In CPPR calculations, the difference between late and early delays (for the common clock segment between launch and capture clock path) is calculated first and then this number is adjusted in the slack calculations to remove the pessimism, which existed because of considering common clock path to be both late and early at the same time. To remove this pessimism in propagated clock mode, you can use the following command:

```
set_analysis_mode -cppr true
```

Consider the following figure for setup check between flops FF1 and FF2.

**Figure 4-64 Signal Path**



The setup check equation for the example in the figure above with pessimism (without CPPR) is as follows:

$$t_{1\max} + t_{2\max} + t_{3\max} \leq t_{4\min} + t_{cp} - t_{su}$$

where  $t_{cp}$  is the clock period and  $t_{su}$  is the setup requirement at D pin of flip-flop FF2.

The above setup check equation incorrectly implies that the common clock network, B1, can simultaneously use maximum delay for the launch clock late path (clock source to FF1/CLK) and minimum delay for the capture clock early path (clock source to FF2/CLK). Using the CPPR to remove this pessimism, the setup check equation is as follows:

$$t_{1\max} + t_{2\max} + t_{3\max} \leq t_{4\min} + t_{\text{cp}} - t_{\text{su}} + t_{\text{cppr}}$$

where  $t_{\text{cppr}}$  is the difference in the maximum and minimum delay from the clock source to the branching node.

Similarly, hold check equation using CPPR is as follows:

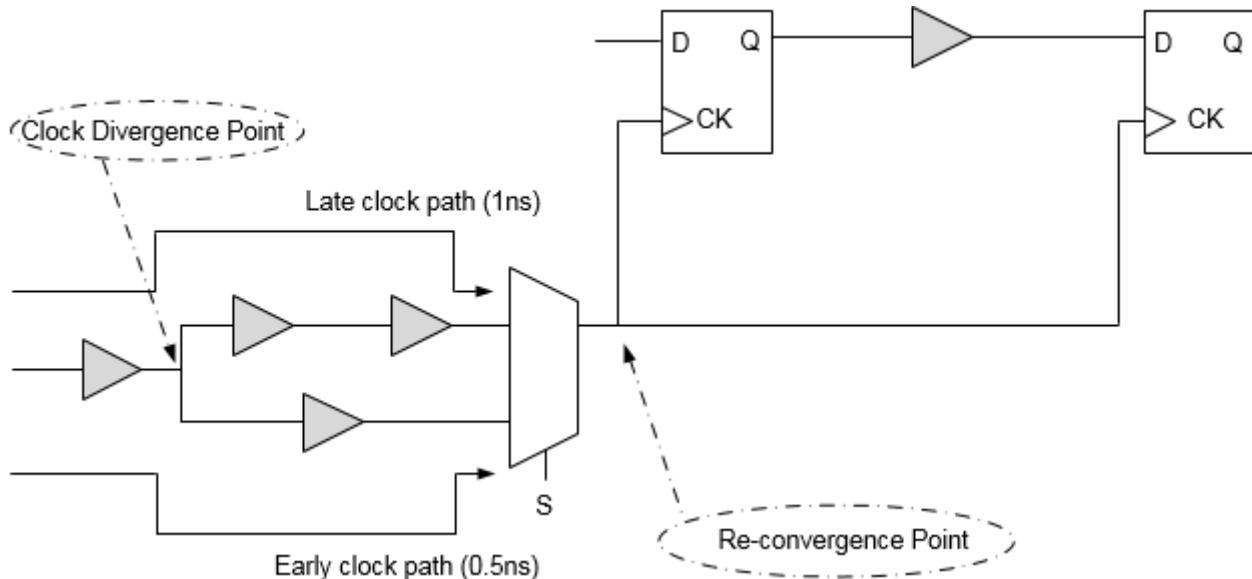
$$t_{1\min} + t_{2\min} + t_{3\min} + t_{\text{cppr}} \leq t_{4\max} + t_H$$

where  $t_H$  is the hold requirement at D pin of flop FF2.

### Clock Reconvergence and CPPR

If a design contains re-convergent logic on the clock path, then the timing analysis software might assume certain pessimism while calculating slack. The following figure shows a circuit in which timing analysis is performed in single analysis mode.

**Figure 4-65 Illustration of Clock Reconvergence**



In this case, if the `set_case_analysis` command has not been set at point S of the multiplexer, the timing analysis software will assume different delay values for early and late

paths. For example, if common early clock path from the clock source to the common clock point has a delay of 0.5ns, and the same common late clock path has delay of 1ns, then a pessimism equal to 0.5ns is introduced in the design. The above pessimism is not specific to single analysis mode only; it also applies to best-case/worst-case and on-chip variation methodologies. You can use CPPR to remove pessimism introduced due to reconvergence.

## Supported CPPR Global Variables

The Tempus software uses multiple global variables to control CPPR behavior. These can be changed based on the user requirements. When these global variables are changed, there can be changes in common clock path pessimism removal in terms of accuracy, common point selection, SI behavior, and so on.

Some of these global variables include:

- `timing_cppr_remove_clock_to_data_crp`
- `timing_cppr_self_loop_mode`
- `timing_cppr_skip_clock_reconvergence`
- `timing_cppr_threshold_ps`
- `timing_cppr_transition_sense`
- `timing_enable_pessimistic_cppr_for_reconvergent_clock_paths`
- `timing_enable_si_cppr`
- `timing_cppr_opposite_edge_mean_scale_factor`
- `timing_cppr_opposite_edge_sigma_scale_factor`

Tempus uses a default threshold of 20ps during pessimism removal. This means that 20ps of uncertainty remains in the analysis. To set the threshold value you can use the `timing_cppr_threshold_ps` global variable. When you set this global variable to a specified value, all the paths may be reported without having pessimism removed by the given value of the global variable. During SI analysis, the CPPR behavior for setup and hold checks can be controlled by setting the `timing_enable_si_cppr` global variable.

## Timing Analysis Results Before and After CPPR

The following example shows a full clock path timing report generated before CPPR analysis. The timing slack is 7.388 without any CPPR adjustments:

```
Path 1: MET Setup Check with Pin ff2/CK
Endpoint: ff2/D (v) checked with leading edge of 'clk'
```

# Tempus User Guide

## Analysis and Reporting

---

```

Beginpoint: ff1/Q (v) triggered by leading edge of 'clk'
Path Groups: {clk}
Other End Arrival Time 0.972
- Setup 0.335
+ Phase Shift 10.000
= Required Time 10.637
- Arrival Time 3.249
= Slack Time 7.388
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Timing Path:

Instance Cell Arc Delay Arrival Time Required Time

c1 CLKBUFX1 clk ^ 0.000 7.388
 CLKBUFX1 A ^ -> Y ^ 0.312 0.312 7.700
c2 CLKBUFX1 A ^ -> Y ^ 0.343 0.655 8.042
c3 CLKBUFX1 A ^ -> Y ^ 0.175 0.829 8.217
c4 CLKBUFX1 A ^ -> Y ^ 0.126 0.956 8.343
c5 CLKBUFX1 A ^ -> Y ^ 0.152 1.108 8.496
mux1 MX2X1 B ^ -> Y ^ 0.211 1.319 8.706
cp11 CLKBUFX1 A ^ -> Y ^ 0.146 1.465 8.852
cp12 CLKBUFX1 A ^ -> Y ^ 0.131 1.596 8.983
cp13 CLKBUFX1 A ^ -> Y ^ 0.157 1.752 9.140
ff1 DFFHQX1 CK ^ -> Q v 0.422 2.175 9.562
u2 BUFX1 A v -> Y v 0.309 2.483 9.871
u4 BUFX1 A v -> Y v 0.279 2.762 10.150
u5 BUFX1 A v -> Y v 0.258 3.020 10.408
u6 BUFX1 A v -> Y v 0.228 3.248 10.636
ff2 DFFHQX1 D v 0.001 3.249 10.637

Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Other End Path:

Instance Cell Arc Delay Arrival Time Required Time

 CLKBUFX1 clk ^ 0.000 -7.388
c1 CLKBUFX1 A ^ -> Y ^ 0.124 0.124 -7.264
c2 CLKBUFX1 A ^ -> Y ^ 0.160 0.284 -7.103
cp21 CLKBUFX1 A ^ -> Y ^ 0.150 0.434 -6.954
cp22 CLKBUFX1 A ^ -> Y ^ 0.133 0.566 -6.821
cp23 CLKBUFX1 A ^ -> Y ^ 0.120 0.687 -6.701
cp24 CLKBUFX1 A ^ -> Y ^ 0.112 0.799 -6.589
cp25 CLKBUFX1 A ^ -> Y ^ 0.093 0.891 -6.496
cp26 CLKBUFX1 A ^ -> Y ^ 0.081 0.972 -6.416
ff2 DFFHQX1 CK ^ 0.000 0.972 -6.416

```

When CPPR adjustment is made, the timing slack improvement is seen as pessimism and is displayed as “CPPR Adjustment” in the timing report.

In the example given below, “c2” is a common clock point after which the clock diverges:

```

Path 1: MET Setup Check with Pin ff2/CK
Endpoint: ff2/D (v) checked with leading edge of 'clk'
Beginpoint: ff1/Q (v) triggered by leading edge of 'clk'
```

# Tempus User Guide

## Analysis and Reporting

---

Path Groups: {clk}

|                           |        |
|---------------------------|--------|
| Other End Arrival Time    | 0.972  |
| - Setup                   | 0.335  |
| + Phase Shift             | 10.000 |
| + CPPR Adjustment         | 0.370  |
| = Required Time           | 11.008 |
| - Arrival Time            | 3.249  |
| = Slack Time              | 7.758  |
| Clock Rise Edge           | 0.000  |
| = Beginpoint Arrival Time | 0.000  |

Timing Path:

| Instance | Cell     | Arc                 | Delay | Arrival Time | Required Time |
|----------|----------|---------------------|-------|--------------|---------------|
| <hr/>    |          |                     |       |              |               |
| c1       | CLKBUFX1 | clk ^<br>A ^ -> Y ^ | 0.312 | 0.312        | 8.070         |
| c2       | CLKBUFX1 | A ^ -> Y ^          | 0.343 | 0.655        | 8.413         |
| c3       | CLKBUFX1 | A ^ -> Y ^          | 0.175 | 0.829        | 8.588         |
| c4       | CLKBUFX1 | A ^ -> Y ^          | 0.126 | 0.956        | 8.714         |
| c5       | CLKBUFX1 | A ^ -> Y ^          | 0.152 | 1.108        | 8.866         |
| mux1     | MX2X1    | B ^ -> Y ^          | 0.211 | 1.319        | 9.077         |
| cp11     | CLKBUFX1 | A ^ -> Y ^          | 0.146 | 1.465        | 9.223         |
| cp12     | CLKBUFX1 | A ^ -> Y ^          | 0.131 | 1.596        | 9.354         |
| cp13     | CLKBUFX1 | A ^ -> Y ^          | 0.157 | 1.752        | 9.511         |
| ff1      | DFFHQX1  | CK ^ -> Q v         | 0.422 | 2.175        | 9.933         |
| u2       | BUFX1    | A v -> Y v          | 0.309 | 2.483        | 10.242        |
| u4       | BUFX1    | A v -> Y v          | 0.279 | 2.762        | 10.520        |
| u5       | BUFX1    | A v -> Y v          | 0.258 | 3.020        | 10.778        |
| u6       | BUFX1    | A v -> Y v          | 0.228 | 3.248        | 11.007        |
| ff2      | DFFHQX1  | D v                 | 0.001 | 3.249        | 11.008        |

Clock Rise Edge 0.000

= Beginpoint Arrival Time 0.000

Other End Path:

| Instance | Cell     | Arc                 | Delay | Arrival Time | Required Time |
|----------|----------|---------------------|-------|--------------|---------------|
| <hr/>    |          |                     |       |              |               |
| c1       | CLKBUFX1 | clk ^<br>A ^ -> Y ^ | 0.124 | 0.124        | -7.758        |
| c2       | CLKBUFX1 | A ^ -> Y ^          | 0.160 | 0.284        | -7.634        |
| cp21     | CLKBUFX1 | A ^ -> Y ^          | 0.150 | 0.434        | -7.474        |
| cp22     | CLKBUFX1 | A ^ -> Y ^          | 0.133 | 0.566        | -7.325        |
| cp23     | CLKBUFX1 | A ^ -> Y ^          | 0.120 | 0.687        | -7.192        |
| cp24     | CLKBUFX1 | A ^ -> Y ^          | 0.112 | 0.799        | -7.072        |
| cp25     | CLKBUFX1 | A ^ -> Y ^          | 0.093 | 0.891        | -6.960        |
| cp26     | CLKBUFX1 | A ^ -> Y ^          | 0.081 | 0.972        | -6.867        |
| ff2      | DFFHQX1  | CK ^                | 0.000 | 0.972        | -6.786        |

### **4.3.3 Advanced On-Chip Variation (AOCV) Analysis**

- 4.3.3.1 [Overview](#) on page 139
- 4.3.3.2 [Using AOCV Derating Library Formats](#) on page 139
- 4.3.3.3 [Enabling AOCV Analysis](#) on page 144
- 4.3.3.4 [Reading AOCV Libraries](#) on page 144
- 4.3.3.5 [Stage Counts and their Properties for Launch and Capture Paths](#) on page 145
- 4.3.3.6 [Handling OCV Derating with AOCV](#) on page 150
- 4.3.3.7 [Configuring AOCV Modes](#) on page 152
- 4.3.3.8 [Reporting in AOCV Modes](#) on page 152
- 4.3.3.9 [AOCV Reporting](#) on page 162

#### 4.3.3.1 Overview

In on-chip variation (OCV) mode, the software uses the timing delay values from the late or early library sets and operating conditions for the data, capture, and launch clocks. OCV analysis uses a single constant derating factor. However, in actual Silicon, there can be lot of sources of variation. For example, variation of transistor length, distance, width, doping amount, and so on. To model these, advanced on-chip variation (AOCV) analysis can be used during timing analysis and optimization. AOCV analysis uses variable derating factors that are based on the distance of the cell and logic depth. Depending on the number of logic levels and placement locations, different derating factors can be used.

AOCV derating libraries are provided by library vendors, or can be characterized using Cadence's tool "Variety". An AOCV factor is chosen so that when multiplied by the lib nominal delay value, you get close to the mean  $\pm 3$  sigma value for that many stages in the path. As number of stages in the path increases, the sigma value (the spread) relative to the mean (nominal value) decreases on the order  $1/\sqrt{n}$ . Due to which AOCV derate factor required decreases.

#### 4.3.3.2 Using AOCV Derating Library Formats

There are two different AOCV derating library formats:

- External AOCV Format
- Cadence-Specific AOCV Derating Format

The characterization data within either format is essentially the same - the differences are mainly syntactic.

## External AOCV Format

The external AOCV format is currently supported by third parties. For example,

```
version: 2.0
voltage: 1.5
object_type: lib_cell
object_spec: LIB/BUF1X
delay_type: [net cell]+
path_type: [data clock]+
derate_type: [early | late]
rf_type: [rise fall]+
depth: 1 2 3 4 5
distance: 500 1000 1500 2000
table:
1.123 1.090 1.075 1.067 1.062
1.124 ...
```

3rd Party AOCV Format

## Cadence-Specific AOCV Derating Format

For example,

```
[Apply-[Net | Cell]]
Vdd 1.5
[Early | Late][-Rise | Fall][Clock | Data]
[Cell | Net | Instance] LIB/BUFX1
Stage 1 2 3 4 5
Distance 500 1000 1500 2000
1.123 1.090 1.075 1.067 1.062
1.124 ...
```

Cadence AOCV Format

## Cadence-Specific AOCV Derating Format: Library Syntax

The Cadence specific AOCV derating format is made of several sections allowing you to customize how each table is applied:

| Sections               | Tags        | Notes    |
|------------------------|-------------|----------|
| [Apply- [Net   Cell] ] | Table Group | Optional |

|                                                |                |                                        |
|------------------------------------------------|----------------|----------------------------------------|
| [Voltage 1.5]                                  | Table Voltage  | Optional                               |
| [Early   Late] -[Rise   Fall]? -[Clock   Data] | Table ID       | Early or Late is required              |
| [Cell   Net   Instance ] LIB/BUFS1             | Object ID      | Optional                               |
| Depth 1 2 3 4 5                                | Derating Table | It is possible to have a 1-D distance. |
| [Distance 500 1000 1500 2000]?                 |                |                                        |
| 1.123 1.090 1.075 1.067 1.062                  |                |                                        |
| 1.124 1.091 1.076 1.068 1.063                  |                |                                        |
| 1.125 1.092 1.077 1.070 1.065                  |                |                                        |
| 1.126 1.094 1.079 1.072 1.067                  |                |                                        |

The table sections are described below.

### **Table Group**

The Apply- tag can be used with either a net or cell descriptor to designate the table for use in derating net delays or cell delays, respectively. If the Apply- tag is not specified, then the table will be used for both net and cell delay derating. The Apply- tags are described below:  
To designate a table for only net-based derating, you should specify:

Apply-Net

To designate a table for only cell-based derating, use the following:

Apply-Cell

If you use the Object ID field to specify unique tables for specified libraries and/or library cells, you should also specify Apply-Cell for these tables.

### **Table ID**

The Table ID is a tag made up of up to three sub tags, which are used to further indicate to the derating table to control early versus late, rise versus fall, and clock path versus data path.  
{Early | Late}

Each table specifies Early or Late tags. Each object related to AOCV derating has both Early and Late derating tables. AOCV derating of only early or late paths is not allowed. It is, however, legal to use derating values of 1.0 for the tables.

By default, an Early or Late derating table is used to supply derating for both rising and falling delays. If a unique derating for each transition is required, the -Rise or -Fall tag can be added to the Table ID for more specific results.

{Early-Rise}

To configure a table for only rising delays on early paths, you should use Early-Rise sub tag.  
{Late-Fall}

To configure a table for falling delays on late timing paths, you can specify Late-Fall.

[Clock | Data]

Derating tables can also be configured so that they are specific to clock path delays or data path delays. This can be achieved by adding a final -Clock or -Data tag to the Table ID string.

The complete set of possible Table IDs is:

|             |                  |                  |                 |
|-------------|------------------|------------------|-----------------|
| Early       | Early-Rise-Clock | Early-Rise-Clock | Late-Rise-Clock |
| Early-Rise  | Early-Rise-Data  | Late-Rise        | Late-Rise-Data  |
| Early-Fall  | Early-Fall-Clock | Late-Fall        | Late-Fall-Clock |
| Early-Clock | Early-Fall-Data  | Late-Clock       | Late-Fall-Data  |
| Early-Data  |                  | Late-Data        |                 |

In the case where there is an overlap in the table specification, the derating table with the tightest specification is used.

## Object ID

You can further control the AOCV derating tables to be applied to specific timing libraries or library cell groups. The syntax also supports specifying design-level cell instances or nets, however, these types of design-level references are not currently supported.

**Note:** The AOCV libraries are configured and stored in the system the same way as Liberty timing libraries, however, only library or library-cell level derate factors are supported.

In the AOCV flow you can use the Cell Object ID to designate particular derating tables to be used for specific library-level groups. An object expression using simple wildcard rules can be used for a variety of selection criteria. For example, the following expression specifies a derating table to be used for a specific library:

Cell stdcell/\*

The following expression specifies derating for a specific library cell:

```
Cell stdcell/BUF1S
```

The following expression specifies a common derate table for a group of cells with similar drive strength:

```
Cell stdcell/*1S
Cell stdcell/*4S
```

The following expression specifies a common derate factor for a group of libraries at the same corner:

```
Cell *-P1V15T120/*
```

## Table Voltage

In the future, an optional voltage designation for the table can be specified by adding the Voltage specifier to the model description. You can specify several tables with different voltage specifications within the same .aocv library file. When performing derating, the system uses the table that matches the current operating voltage of the cell. If the operating voltage of the cell does not exactly match one of the specified voltage points, then the following are considered:

- If the operating voltage is between the Voltage points of two derating tables, the system will use linear interpolation to derive the proper derating factor.
- If the requested voltage is large than the maximum Voltage table, then that table data will be used without extrapolation.
- If the requested voltage is lower than the minimum Voltage table, then the minimum Voltage data is used without extrapolation.
- To specify a voltage of 1.5Volts for the derating table, use a specification of:  
`Voltage 1.5`

## Derating Table

The AOCV format allows derating table specifications that are either one or two-dimensional – the possible axes being Distance, the bounding box diagonal of that path, and Stage - the stage count depth of the path. Although AOCV graph-based analysis supports stage-based derating only, you may specify either a 1-D stage-based table, or a 2-D Stage x Distance table.

When using a 2-D table in conjunction with AOCV graph-based stage-based only analysis, the system will attempt to pick the proper row of the table based on the estimated chip and/or core dimensions. If the software includes physical information, the chip and core size

are automatically determined from the design. The `aocv_chip_size` and `aocv_core_size` global variables can be used in the software to specify the physical dimensions when this information is not available. If this value falls off the table, then the final row of the table will be used without extrapolation.

The Stage and Distance keywords are added to the model to provide the axis points for the table. The table rows represent “distance” variance and the columns “stage count”, irrespective of the order in which stage and distance keywords are specified for the model. These are explained below:

**Distance:** List of distance (real) values, specified in microns, increasing in magnitude.

**Stage:** List of integer values representing stage depth, increasing in magnitude.

The following example shows a derating table with 5 Stage indices and 4 Distance indices:

```
Stage 1 2 3 4 5
Distance 500 1000 1500 2000
1.123 1.090 1.075 1.067 1.062
1.124 1.091 1.076 1.068 1.063
1.125 1.092 1.077 1.070 1.065
1.126 1.094 1.079 1.072 1.067
```

In case the stage or distance lookup value goes beyond the range of the table - either smaller or larger, the system will use the last bound entry point. Extrapolation will not be performed on either end of the axis.

#### 4.3.3.3 Enabling AOCV Analysis

AOCV analysis can be enabled by using the following command variables.

```
set analysis mode -aocv true
```

#### 4.3.3.4 Reading AOCV Libraries

##### SMSC Environment

In SMSC environment, the AOCV libraries can be read using the command below:

```
read lib -aocv "myDerateLib.aocv.lib"
```

##### MMMC Environment

In MMMC environment, AOCV libraries can be read using the `create_library_set` command.

For example:

```
create_library_set -name slow -timing slow.lib -aocv test.aocv.lib
```

In case, the library set is already created , it can be updated to read the new aocv library or update the existing library using the update\_library\_set command.

For example:

```
update_library_set -name slow -aocv test.aocv.lib
```

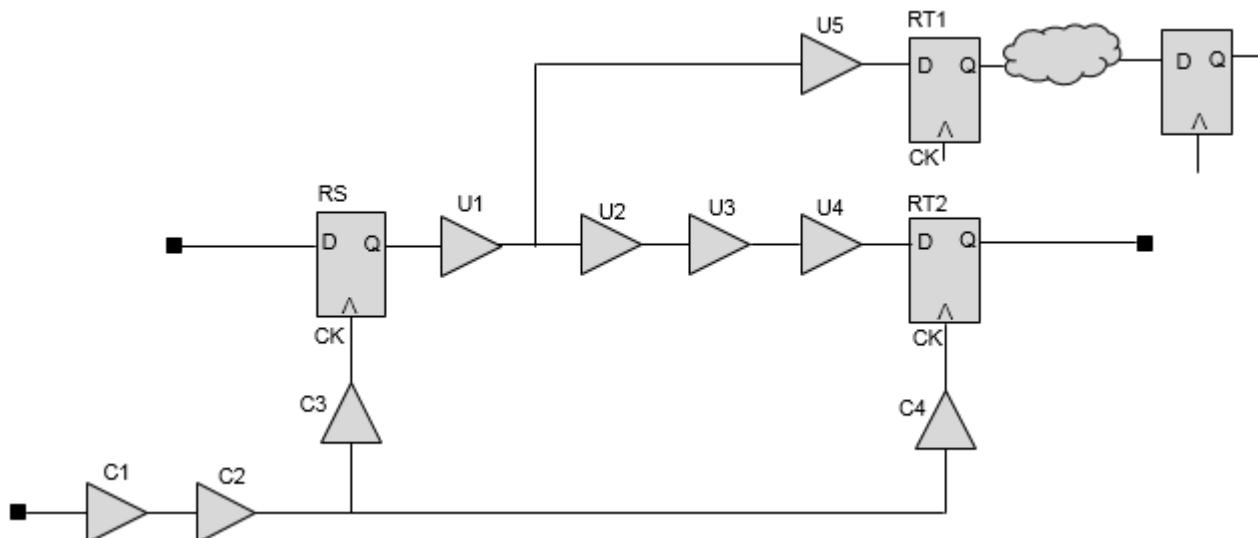
#### 4.3.3.5 Stage Counts and their Properties for Launch and Capture Paths

Most cells in a clock path have both launch stage counts and capture stage counts. This means a clock buffer exists in a launch path and a capture path simultaneously so the buffer has its own launch stage count property and capture stage count property. This section covers a comprehensive explanation for the launch stage count and capture stage count.

The following four timing paths in the figure below explain launch and capture stage counts:

- The timing path from Input to RS/D (input to reg path)
- The timing path from RS/CK to RT1/D (reg to reg path)
- The timing path from RS/CK to RT2/D (reg to reg path)
- The timing path from RT2/CK to output1 (reg to output path)

**Figure 4-66 Timing Paths**



The following default settings are assumed:

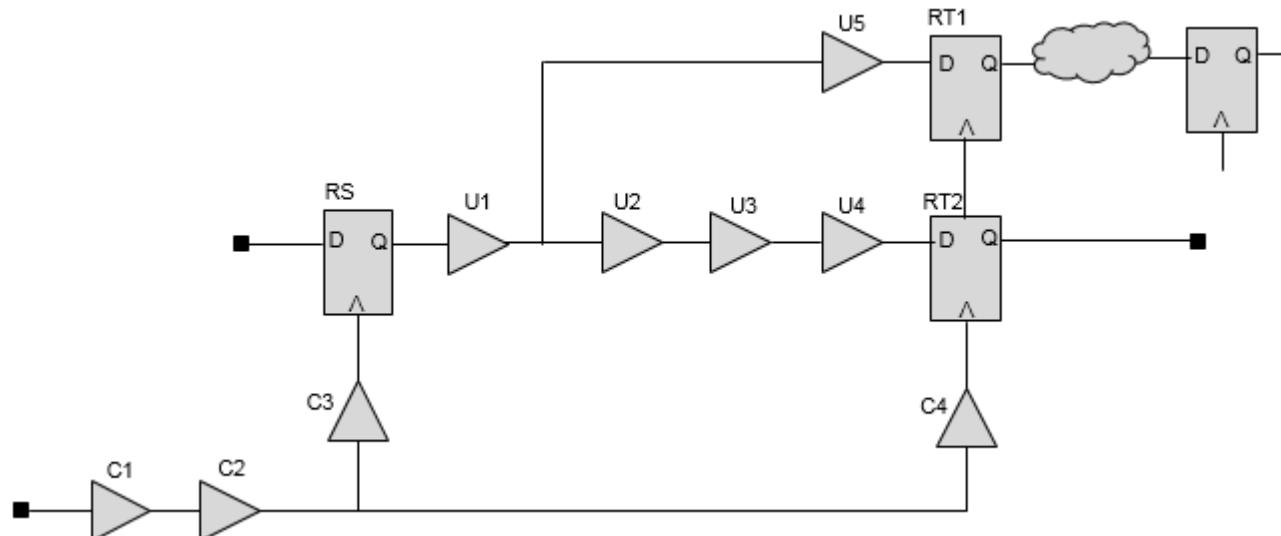
```
set timing aocv analysis mode launch_capture; #(default)
set timing aocv use cell depth for net false; #(default: false)
```

### Stage Counting Graph Based Analysis (GBA)

Graph-based AOCV analysis does choose a pessimistic situation so each cell has its own pessimistic property (only one value, not two or more values like PBA) such as the late launch stage count for cell, early capture stage count for cell, and so on.

The following figure includes the launch cell and net stage counts:

**Figure 4-67 Launch cell and net stage counts**



The stage counts of C1 and C2 for launch paths in the above example are as follows:

The clock port is connected to input pin of C1.

1. From the clock port, there are 4 launch paths which include C1 and C2. that is, path from the clock port through C1/C2/C3/RS/U1/U5 to RT1.
2. The launch path from the clock port through C1/C2/C3/RS/U1/U2/U3/U4 to RT2.
3. The launch path from the clock port through C1/C2/C4/RT1 through some combination logic to the FF on the top-right side.
4. The launch path from the clock port through C1/C2/C4/RT2 to the output port.

The last path is the shortest launch path that includes C1 and C2 and each launch cell stage count is 4 (C1, C2, C4, and RT2).

Next, stage count is reset to 0 at the common point C2/Y.

In case of the launch path, which includes C3/RS/U1, each stage count of C3/RS/U1 is 4/4/4 for the end point RT1 and stage count of 6/6/6 for the end point RT2.

Now, the problem of choosing the stage count occurs on C3/RS/U1 because the cells exist in the same launch path in the case of the two timings paths (RS to RT1 and RS to RT2) and have different stage count based in path-based AOCV analysis.

Therefore, to maintain pessimism, the smaller stage count 4 is chosen. So, each launch cell stage count of C3/RS/U1 is 4 in case of graph-based AOCV analysis. U5 has a stage count of 4 and U2/U3/U4 has a stage count of 6, respectively. Smaller stage count means more variation, so it is mapped to the worse AOCV derate for late and early.

**Note:** In GBA, the algorithm requires resetting of stage count at clock branch points- a major source of pessimism.

In the given figure, another common point occurs at the pin C4/Y. which is connected to RT1/CK and RT2/CK. So, state count is reset to 0 at the C4/Y pin. The stage count of the path from the C4/Y through RT1 is longer than that of the path from the C4/Y through RT2 to the output port. RT2 has stage count of 1 and RT1 has a stage count of a value greater than 1 and C4 has a stage count of 2.

The following table summarizes the graph-based launch cell stage counts of the above example:

| Segment                                                                        | Members  | Stage Count | Stage Path        |
|--------------------------------------------------------------------------------|----------|-------------|-------------------|
| Clock root -> CPPR Common Point (C2/Y)                                         | C1-C2    | 4           | C1-C2-C4-RT2      |
| CPPR Common Point (C2/Y) -> Launch Clock Branch Point (RS) -> Data Path Branch | C3-RS-U1 | 4           | C3-RS-U1-U5       |
| Data Path Branch -> Data Path Endpoint                                         | U5       | 4           | C3-RS-U1-U5       |
| Data Path Branch -> Data Path Endpoint                                         | U2-U3-U4 | 6           | C3-RS-U1-U2-U3-U4 |
| CRPR Common Point (C2/Y) -> CRPR Common Point (C4/Y)                           | C4       | 2           | C4-RT2            |

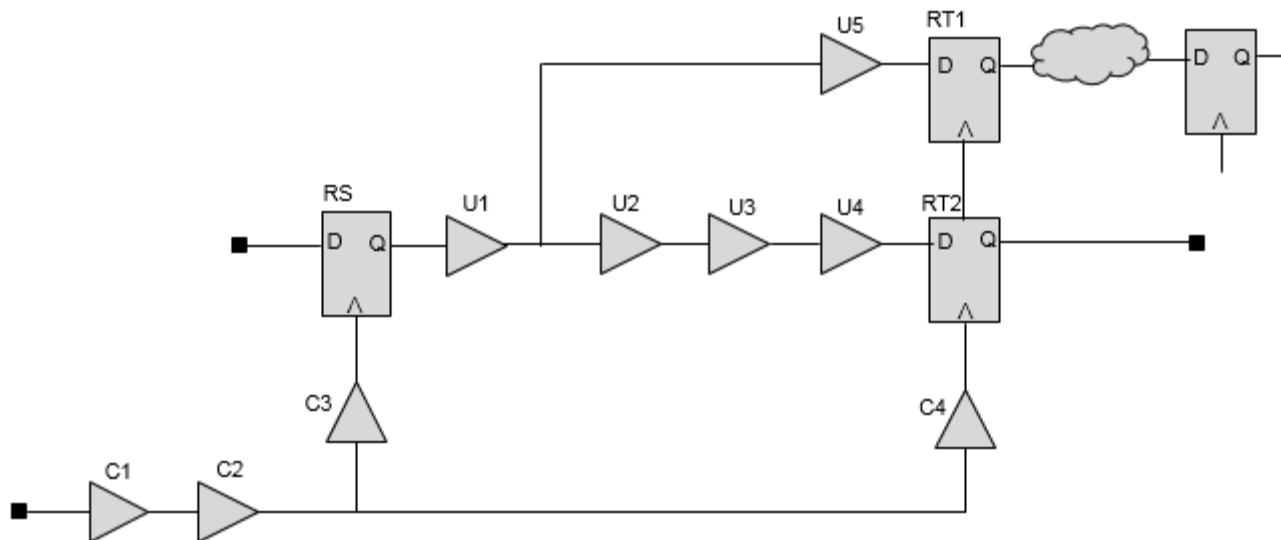
|                                                                                         |     |   |     |
|-----------------------------------------------------------------------------------------|-----|---|-----|
| CRPR Common Point (C4/Y) -> Launch<br>Clock Branch Point (RT2) -> data Path<br>Endpoint | RT2 | 1 | RT2 |
|-----------------------------------------------------------------------------------------|-----|---|-----|

### Stage Counting Path-Based Analysis (PBA)

The method to calculate the stage count under path-based AOCV mode is intuitive.

The following figure includes the launch cell and net stage counts:

**Figure 4-68 Launch cell and net stage counts**



Firstly, look for the CPPR common point on the clock launch path and the capture path of a timing path.

To derive the common point from RS to RT1:-

- The launch path from RS to RT1:  
clock->C1->C2->C3->RS->U1->U5->RT1/D
- The capture path from RS to RT1:  
clock->C1->C2->C4->RT1/CK.

So, the common point (pin) of the above timing path is the C2/Y pin.

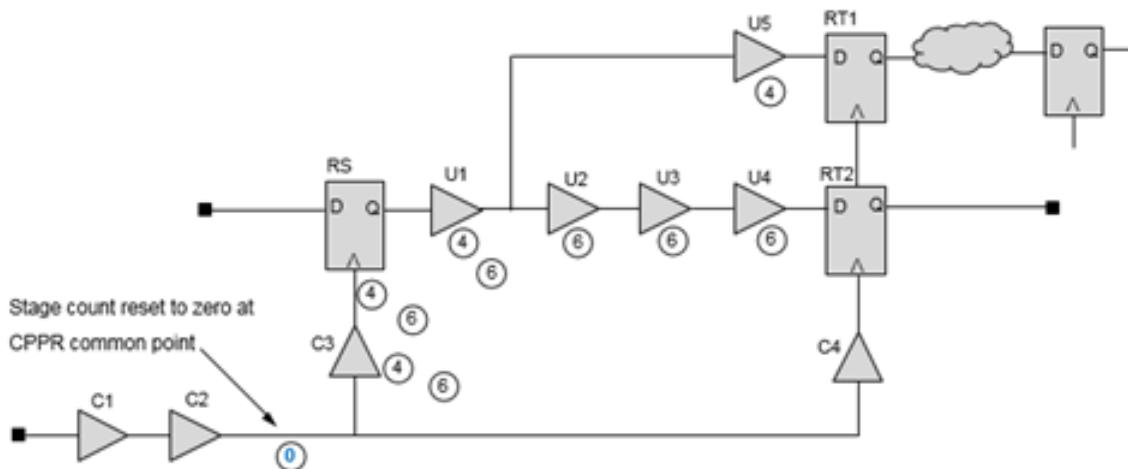
To derive the common point from RS to RT2:

- The launch path from RS to RT2:  
clock->C1->C2->C3->RS->U1->U2->U3->U4->RT2/D.
- The capture path from RS to RT2:  
clock->C1->C2->C4->RT2/CK.

So, the common point (pin) of the above timing path is the C2/Y pin.

Now, for calculation purpose, the stage count at C2/Y is reset to 0, as shown in below figure.

**Figure 4-69 Stage count at C2/Y reset to 0**



Therefore, the launch path from the pin C2/Y to RT1/D has a stage count of 4, that is,  
C3(1)->RS(2)->U1(3)->U5(4)->RT1/D

The launch path from the pin RS to RT2/D has a stage count of 6, that is,  
C3(1)->RS(2)->U1(3)->U2(4)->U3(5)->U4(6)->RT2/D

| Segment                                                         | Members           | Stage Count | Stage Path        |
|-----------------------------------------------------------------|-------------------|-------------|-------------------|
| Launch Clock Branch Point - ><br>Data Path Endpoint (RS to RT1) | C3-RS-U1-U5       | 4           | C3-RS-U1-U5       |
| Launch Clock Branch Point - ><br>Data Path Endpoint (RS to RT2) | C3-RS-U1-U2-U3-U4 | 6           | C3-RS-U1-U2-U3-U4 |

The table above summarizes the stage count of the 2 launch paths in PBA.

Based on above derived stage count, the tool calculates the diagonal distance for the timing path and then gets the AOCV derate values from the AOCV derating library with the stage count and the distance.

**Note:** The `report_timing` command can be used to report each cell's AOCV derate and each net's AOCV derate.

So, think over which AOCV derates of the cells/nets on the common path are used for the common parts. In the above example, you have to check the stage counts of the cells C1 and C2 on the common path and AOCV derates of the cells. The stage count (SC) calculation on the common path cells is relating to graph-based AOCV analysis.

#### 4.3.3.6 Handling OCV Derating with AOCV

The timing system offers a set of controls that enable you to flexibly configure the derating environment. You can perform the following functions:

- Control whether AOCV and OCV derating factors are multiplied or added.
- Specify whether the reference point for the AOCV and OCV factors are nominal around 0.0, or around 1.0 - and adjust the calculation appropriately.
- Allow the final OCV factor to be built up by successive multiplication or addition of sub-factors by using the `set_timing_derate` command.

These are described in the subsequent sections.

#### Using AOCV Addition or Multiplication with OCV

By default, when AOCV analysis is enabled, any OCV style derating specified with the `set_timing_derate` command is multiplied by the calculated AOCV factor to get the final derating value. You can use the following setting to add or multiply AOCV and OCV factors:

```
set_timing aocv_derate_mode aocv_multiplicative (default) | aocv_additive
```

#### Using Derating Reference Points

The AOCV and OCV derating factors may be added or multiplied, so it is important to understand whether each factor is nominal at zero or one. For example, if there is an AOCV +3% factor modeled as 1.03, and an OCV factor having a plus two percent factor modeled as 1.02, simple multiplication of factors yields a final derate of around 1.051. If the same factors are added up, a sum of 2.05 is obtained, which is not the expected value. The timing system can understand that both of these factors are nominally referenced to 1.0, and it can then automatically adjust the final result. In this case the timer will calculate:

(OCV) (AOCV) (Adjust)  
 $1.02 + 1.03 - 1.0 = 1.05$

The following global variables can be used to allow the specification of the reference point for both OCV and AOCV factors:

```
set_global timing derate aocv reference point 0 | 1 (default)
set_global timing derate ocv reference point 0 | 1 (default)
```

According to current AOCV library definitions, the AOCV reference point is always expected to be set at 1.0.

### Using Addition and Multiplication of OCV Sub-Factors

The software provides the ability to perform incremental adjustments to the defined timing derate factors. The software also allows addition and multiplication (using `set_timing_derate -add` and `-multiply` parameters) to function on an instance-specific basis as well.

When `aocv_additive` mode is selected, the adjustments are performed as shown in the table below.

| AOCV Factor | AOCV Reference Pt | OCV Factor | OCV Reference Pt | Add or Multiply     | Final Derate Calculation |
|-------------|-------------------|------------|------------------|---------------------|--------------------------|
| X           | 0/1               | Y          | 0/1              | aocv_multiplicative | $X * Y$                  |
| X           | 1.0               | Y          | 1.0              | aocv_additive       | $X + Y - 1$              |
| X           | 1.0               | Y          | 0.0              | aocv_additive       | $X + Y$                  |
| X           | 0.0               | Y          | 1.0              | aocv_additive       | $X + Y$                  |
| X           | 0.0               | Y          | 0.0              | aocv_additive       | $X + Y + 1$              |

The table shows the hierarchy of derating factors in increasing order of precedence. No adjustments are performed for `aocv_multiplicative` mode.

**Note:** It is expected that AOCV derating libraries will be referenced to 1.0. So the combinations above are not expected to be actively used.

#### 4.3.3.7 Configuring AOCV Modes

The Tempus software allows you can control the handling of AOCV analysis modes by using the `timing_aocv_analysis_mode` setting. You can specify one of the following modes for both PBA and GBA:

- `combine_launch_capture`
- `launch_capture` (default)
- `clock_only`
- `separate_data_clock`

These are described below.

- **`combine_launch_capture` (default)**

When set to `combine_launch_capture`, the software calculates the AOCV stage count using the combined depth for launch and capture paths. This is achieved by the summation of the path depth on the launch and capture paths.

- **`launch_capture`**

When set to `launch_capture`, AOCV analysis will be performed on both the data and clock paths, and the AOCV derating factor will be calculated based for both clock and data path stages.

- **`clock_only`**

When set to `clock_only`, AOCV analysis will be performed for clock paths only, and AOCV derating factor will be calculated based on only the clock path stage depth.

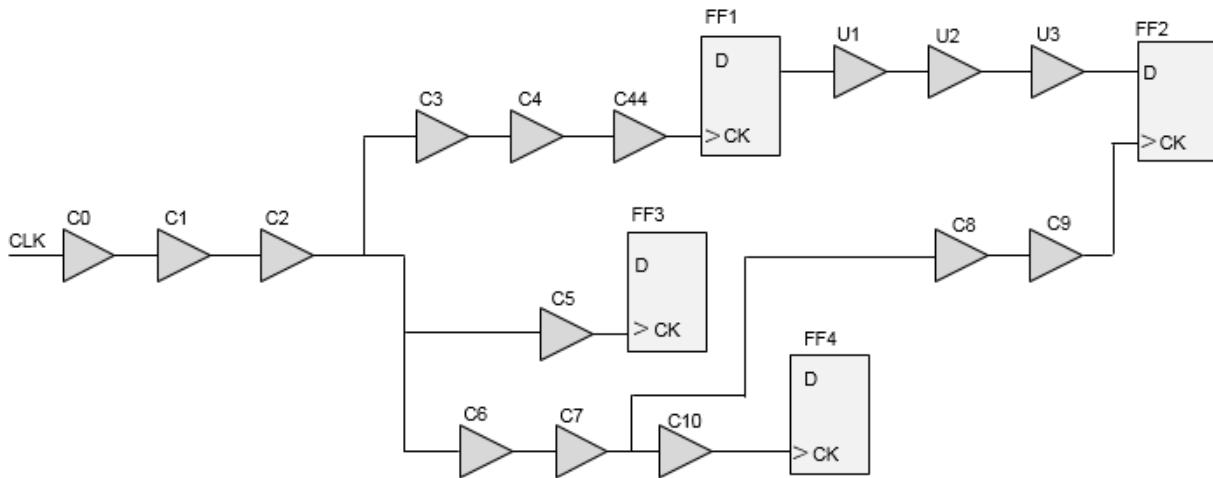
- **`separate_data_clock`**

When set to `separate_data_clock`, the software calculates the AOCV stage count for clock paths and data paths separately. Both clock and data paths have related AOCV derating factors.

#### 4.3.3.8 Reporting in AOCV Modes

Consider the following diagram:

**Figure 4-70 AOCV stage count**



The following examples show AOCV stage count in GBA and PBA mode:

*timing\_aocv\_analysis\_mode* *clock\_only mode (GBA)*

```
Path 1: MET Setup Check with Pin FF2/CK
Endpoint: FF2/D (^) checked with leading edge of 'clk'
Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'
Path Groups: {clk}
Other End Arrival Time 0.000
- Setup 0.018
+ Phase Shift 10.000
= Required Time 9.982
- Arrival Time 0.079
= Slack Time 9.903
 Clock Rise Edge 0.000
 = Beginpoint Arrival Time 0.000
 Timing Path:
```

| Delay | Arrival Time | Pin      | Arc         | Aocv Derate | Aocv Stage Count |
|-------|--------------|----------|-------------|-------------|------------------|
| -     | 0.000        | clk      | clk ^       | -           | -                |
| 0.000 | 0.000        | C0/A     | -           | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y     | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A     | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C1/Y     | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C2/A     | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C2/Y     | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C3/A     | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C3/Y     | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C4/A     | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C4/Y     | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C44/A    | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C44/Y    | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | FF1/CK-> | -           | 1.000       | 3.000            |
| 0.047 | 0.047        | FF1/O    | CK ^ -> O ^ | 1.000       | -                |

# Tempus User Guide

## Analysis and Reporting

---

|       |       |       |            |       |   |
|-------|-------|-------|------------|-------|---|
| 0.001 | 0.048 | U1/A  | -          | 1.000 | - |
| 0.012 | 0.059 | U1/Y  | A ^ -> Y ^ | 1.000 | - |
| 0.001 | 0.060 | U2/A  | -          | 1.000 | - |
| 0.009 | 0.069 | U2/Y  | A ^ -> Y ^ | 1.000 | - |
| 0.001 | 0.070 | U3/A  | -          | 1.000 | - |
| 0.008 | 0.078 | U3/Y  | A ^ -> Y ^ | 1.000 | - |
| 0.001 | 0.079 | FF2/D | ->         | 1.000 | - |

Clock Rise Edge 0.000  
= Beginpoint Arrival Time 0.000  
Other End Path:

| Delay | Arrival Time | Pin    | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A   | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C2/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A   | -          | 1.000       | 3.000            |
| 0.000 | 0.000        | C6/Y   | A ^ -> Y ^ | 0.803       | 3.000            |
| 0.000 | 0.000        | C7/A   | -          | 1.000       | 3.000            |
| 0.000 | 0.000        | C7/Y   | A ^ -> Y ^ | 0.803       | 3.000            |
| 0.000 | 0.000        | C8/A   | -          | 1.000       | 2.000            |
| 0.000 | 0.000        | C8/Y   | A ^ -> Y ^ | 0.793       | 2.000            |
| 0.000 | 0.000        | C9/A   | -          | 1.000       | 2.000            |
| 0.000 | 0.000        | C9/Y   | A ^ -> Y ^ | 0.793       | 2.000            |
| 0.000 | 0.000        | FF2/CK | -          | 1.000       | 2.000            |

### ***timing\_aocv\_analysis\_mode\_clock\_only mode (PBA)***

Path 1: MET Setup Check with Pin FF2/CK  
Endpoint: FF2/D (^) checked with leading edge of 'clk'  
Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'  
Path Groups: {clk}  
Retime Analysis { AOCV(Clock) }  
Other End Arrival Time 0.000  
- Setup 0.018  
+ Phase Shift 10.000  
= Required Time 9.982  
- Arrival Time 0.079  
= Slack Time 9.903  
Clock Rise Edge 0.000  
= Beginpoint Arrival Time 0.000  
Timing Path:

| Delay | Arrival Time | Pin  | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|------|------------|-------------|------------------|
| -     | 0.000        | clk  | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y | A ^ -> Y ^ | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A | -          | 1.000       | 6.000            |
| 0.000 | 0.000        | C1/Y | A ^ -> Y ^ | 1.186       | 5.000            |
| 0.000 | 0.000        | C2/A | -          | 1.000       | 6.000            |
| 0.000 | 0.000        | C2/Y | A ^ -> Y ^ | 1.186       | 5.000            |
| 0.000 | 0.000        | C3/A | -          | 1.000       | 6.000            |

# Tempus User Guide

## Analysis and Reporting

---

|       |       |           |             |       |       |
|-------|-------|-----------|-------------|-------|-------|
| 0.000 | 0.000 | C3/Y      | A ^ -> Y ^  | 1.186 | 5.000 |
| 0.000 | 0.000 | C4/A      | -           | 1.000 | 6.000 |
| 0.000 | 0.000 | C4/Y      | A ^ -> Y ^  | 1.186 | 5.000 |
| 0.000 | 0.000 | C44/A     | -           | 1.000 | 6.000 |
| 0.000 | 0.000 | C44/Y     | A ^ -> Y ^  | 1.186 | 5.000 |
| 0.000 | 0.000 | FF1/CK -> | -           | 1.000 | 6.000 |
| 0.047 | 0.047 | FF1/Q     | CK ^ -> Q ^ | 1.000 | -     |
| 0.001 | 0.048 | U1/A      | -           | 1.000 | -     |
| 0.012 | 0.059 | U1/Y      | A ^ -> Y ^  | 1.000 | -     |
| 0.001 | 0.060 | U2/A      | -           | 1.000 | -     |
| 0.009 | 0.069 | U2/Y      | A ^ -> Y ^  | 1.000 | -     |
| 0.001 | 0.070 | U3/A      | -           | 1.000 | -     |
| 0.008 | 0.078 | U3/Y      | A ^ -> Y ^  | 1.000 | -     |
| 0.001 | 0.079 | FF2/D ->  | -           | 1.000 | -     |

Clock Rise Edge 0.000  
= Beginpoint Arrival Time 0.000  
Other End Path:

| Delay | Arrival Time | Pin    | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A   | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C6/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C7/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C7/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C8/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C8/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C9/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C9/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | FF2/CK | -          | 1.000       | 5.000            |

### ***timing\_aocv\_analysis\_mode\_separate\_data\_clock mode (GBA)***

Path 1: MET Setup Check with Pin FF2/CK  
Endpoint: FF2/D (^) checked with leading edge of 'clk'  
Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'  
Path Groups: {clk}  
Other End Arrival Time 0.000  
- Setup 0.018  
+ Phase Shift 10.000  
= Required Time 9.982  
- Arrival Time 0.093  
= Slack Time 9.889  
Clock Rise Edge 0.000  
= Beginpoint Arrival Time 0.000  
Timing Path:

| Delay | Arrival Time | Pin  | Arc   | Aocv Derate | Aocv Stage Count |
|-------|--------------|------|-------|-------------|------------------|
| -     | 0.000        | clk  | clk ^ | -           | -                |
| 0.000 | 0.000        | C0/A | -     | 1.000       | 4.000            |

# Tempus User Guide

## Analysis and Reporting

---

|       |       |           |             |       |       |
|-------|-------|-----------|-------------|-------|-------|
| 0.000 | 0.000 | C0/Y      | A ^ -> Y ^  | 1.190 | 4.000 |
| 0.000 | 0.000 | C1/A      | -           | 1.000 | 3.000 |
| 0.000 | 0.000 | C1/Y      | A ^ -> Y ^  | 1.197 | 3.000 |
| 0.000 | 0.000 | C2/A      | -           | 1.000 | 3.000 |
| 0.000 | 0.000 | C2/Y      | A ^ -> Y ^  | 1.197 | 3.000 |
| 0.000 | 0.000 | C3/A      | -           | 1.000 | 3.000 |
| 0.000 | 0.000 | C3/Y      | A ^ -> Y ^  | 1.197 | 3.000 |
| 0.000 | 0.000 | C4/A      | -           | 1.000 | 3.000 |
| 0.000 | 0.000 | C4/Y      | A ^ -> Y ^  | 1.197 | 3.000 |
| 0.000 | 0.000 | C44/A     | -           | 1.000 | 3.000 |
| 0.000 | 0.000 | C44/Y     | A ^ -> Y ^  | 1.197 | 3.000 |
| 0.000 | 0.000 | FF1/CK -> | -           | 1.000 | 3.000 |
| 0.056 | 0.056 | FF1/Q     | CK ^ -> Q ^ | 1.190 | 4.000 |
| 0.001 | 0.056 | U1/A      | -           | 1.000 | 4.000 |
| 0.014 | 0.070 | U1/Y      | A ^ -> Y ^  | 1.190 | 4.000 |
| 0.001 | 0.072 | U2/A      | -           | 1.000 | 4.000 |
| 0.010 | 0.082 | U2/Y      | A ^ -> Y ^  | 1.190 | 4.000 |
| 0.001 | 0.083 | U3/A      | -           | 1.000 | 4.000 |
| 0.009 | 0.092 | U3/Y      | A ^ -> Y ^  | 1.190 | 4.000 |
| 0.001 | 0.093 | FF2/D ->  | -           | 1.000 | 4.000 |

Clock Rise Edge 0.000

= Beginpoint Arrival Time 0.000

Other End Path:

| Delay | Arrival Time | Pin    | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A   | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A   | -          | 1.000       | 3.000            |
| 0.000 | 0.000        | C6/Y   | A ^ -> Y ^ | 0.803       | 3.000            |
| 0.000 | 0.000        | C7/A   | -          | 1.000       | 3.000            |
| 0.000 | 0.000        | C7/Y   | A ^ -> Y ^ | 0.803       | 3.000            |
| 0.000 | 0.000        | C8/A   | -          | 1.000       | 2.000            |
| 0.000 | 0.000        | C8/Y   | A ^ -> Y ^ | 0.793       | 2.000            |
| 0.000 | 0.000        | C9/A   | -          | 1.000       | 2.000            |
| 0.000 | 0.000        | C9/Y   | A ^ -> Y ^ | 0.793       | 2.000            |
| 0.000 | 0.000        | FF2/CK | -          | 1.000       | 2.000            |

### ***timing\_aocv\_analysis\_mode\_separate\_data\_clock mode (PBA)***

Path 1: MET Setup Check with Pin FF2/CK

Endpoint: FF2/D (^) checked with leading edge of 'clk'

Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'

Path Groups: {clk}

Retime Analysis { AOCV(Separate) }

Other End Arrival Time 0.000

- Setup 0.018

+ Phase Shift 10.000

= Required Time 9.982

- Arrival Time 0.093

= Slack Time 9.889

Clock Rise Edge 0.000

= Beginpoint Arrival Time 0.000

Timing Path:

# Tempus User Guide

## Analysis and Reporting

---

| Delay | Arrival Time | Pin       | Arc         | Aocv Derate | Aocv Stage Count |
|-------|--------------|-----------|-------------|-------------|------------------|
| -     | 0.000        | clk       | clk ^       | -           | -                |
| 0.000 | 0.000        | C0/A      | -           | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A      | -           | 1.000       | 6.000            |
| 0.000 | 0.000        | C1/Y      | A ^ -> Y ^  | 1.186       | 5.000            |
| 0.000 | 0.000        | C2/A      | -           | 1.000       | 6.000            |
| 0.000 | 0.000        | C2/Y      | A ^ -> Y ^  | 1.186       | 5.000            |
| 0.000 | 0.000        | C3/A      | -           | 1.000       | 6.000            |
| 0.000 | 0.000        | C3/Y      | A ^ -> Y ^  | 1.186       | 5.000            |
| 0.000 | 0.000        | C4/A      | -           | 1.000       | 6.000            |
| 0.000 | 0.000        | C4/Y      | A ^ -> Y ^  | 1.186       | 5.000            |
| 0.000 | 0.000        | C44/A     | -           | 1.000       | 6.000            |
| 0.000 | 0.000        | C44/Y     | A ^ -> Y ^  | 1.186       | 5.000            |
| 0.000 | 0.000        | FF1/CK -> | -           | 1.000       | 6.000            |
| 0.056 | 0.056        | FF1/Q     | CK ^ -> Q ^ | 1.190       | 4.000            |
| 0.001 | 0.056        | U1/A      | -           | 1.000       | 4.000            |
| 0.014 | 0.070        | U1/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.001 | 0.072        | U2/A      | -           | 1.000       | 4.000            |
| 0.010 | 0.082        | U2/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.001 | 0.083        | U3/A      | -           | 1.000       | 4.000            |
| 0.009 | 0.092        | U3/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.001 | 0.093        | FF2/D ->  | -           | 1.000       | 4.000            |

Clock Rise Edge 0.000

= Beginpoint Arrival Time 0.000

Other End Path:

| Delay | Arrival Time | Pin    | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A   | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C6/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C7/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C7/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C8/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C8/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C9/A   | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C9/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | FF2/CK | -          | 1.000       | 5.000            |

### ***timing\_aocv\_analysis\_mode launch\_capture (GBA)***

Path 1: MET Setup Check with Pin FF2/CK

Endpoint: FF2/D (^) checked with leading edge of 'clk'

Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'

Path Groups: {clk}

Other End Arrival Time 0.000

- Setup 0.018

+ Phase Shift 10.000

# Tempus User Guide

## Analysis and Reporting

---

```
= Required Time 9.982
- Arrival Time 0.093
= Slack Time 9.889
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Timing Path:
```

| Delay | Arrival Time | Pin       | Arc         | Aocv Derate | Aocv Stage Count |
|-------|--------------|-----------|-------------|-------------|------------------|
| -     | 0.000        | clk       | clk ^       | -           | -                |
| 0.000 | 0.000        | C0/A      | -           | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A      | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C1/Y      | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C2/A      | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C2/Y      | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C3/A      | -           | 1.000       | 7.000            |
| 0.000 | 0.000        | C3/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.000 | 0.000        | C4/A      | -           | 1.000       | 7.000            |
| 0.000 | 0.000        | C4/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.000 | 0.000        | C44/A     | -           | 1.000       | 7.000            |
| 0.000 | 0.000        | C44/Y     | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.000 | 0.000        | FF1/CK -> | -           | 1.000       | 7.000            |
| 0.055 | 0.055        | FF1/Q     | CK ^ -> Q ^ | 1.181       | 7.000            |
| 0.001 | 0.056        | U1/A      | -           | 1.000       | 7.000            |
| 0.014 | 0.070        | U1/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.001 | 0.071        | U2/A      | -           | 1.000       | 7.000            |
| 0.010 | 0.081        | U2/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.001 | 0.082        | U3/A      | -           | 1.000       | 7.000            |
| 0.009 | 0.092        | U3/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.001 | 0.093        | FF2/D ->  | -           | 1.000       | 7.000            |

```
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Other End Path:
```

| Delay | Arrival Time | Pin    | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A   | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A   | -          | 1.000       | 3.000            |
| 0.000 | 0.000        | C6/Y   | A ^ -> Y ^ | 0.803       | 3.000            |
| 0.000 | 0.000        | C7/A   | -          | 1.000       | 3.000            |
| 0.000 | 0.000        | C7/Y   | A ^ -> Y ^ | 0.803       | 3.000            |
| 0.000 | 0.000        | C8/A   | -          | 1.000       | 2.000            |
| 0.000 | 0.000        | C8/Y   | A ^ -> Y ^ | 0.793       | 2.000            |
| 0.000 | 0.000        | C9/A   | -          | 1.000       | 2.000            |
| 0.000 | 0.000        | C9/Y   | A ^ -> Y ^ | 0.793       | 2.000            |
| 0.000 | 0.000        | FF2/CK | -          | 1.000       | 2.000            |

# Tempus User Guide

## Analysis and Reporting

---

### ***timing\_aocv\_analysis\_mode launch\_capture (PBA):***

```

Path 1: MET Setup Check with Pin FF2/CK
Endpoint: FF2/D (^) checked with leading edge of 'clk'
Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'
Path Groups: {clk}
Retime Analysis { AOCV(Default) }
Other End Arrival Time 0.000
- Setup 0.018
+ Phase Shift 10.000
= Required Time 9.982
- Arrival Time 0.092
= Slack Time 9.890
Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Timing Path:

```

| Delay | Arrival Time | Pin    | Arc         | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|-------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^       | -           | -                |
| 0.000 | 0.000        | C0/A   | -           | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y   | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A   | -           | 1.000       | 10.000           |
| 0.000 | 0.000        | C1/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.000 | 0.000        | C2/A   | -           | 1.000       | 10.000           |
| 0.000 | 0.000        | C2/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.000 | 0.000        | C3/A   | -           | 1.000       | 10.000           |
| 0.000 | 0.000        | C3/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.000 | 0.000        | C4/A   | -           | 1.000       | 10.000           |
| 0.000 | 0.000        | C4/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.000 | 0.000        | C44/A  | -           | 1.000       | 10.000           |
| 0.000 | 0.000        | C44/Y  | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.000 | 0.000        | FF1/CK | ->          | 1.000       | 10.000           |
| 0.055 | 0.055        | FF1/Q  | CK ^ -> Q ^ | 1.177       | 9.000            |
| 0.001 | 0.056        | U1/A   | -           | 1.000       | 10.000           |
| 0.014 | 0.070        | U1/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.001 | 0.071        | U2/A   | -           | 1.000       | 10.000           |
| 0.010 | 0.081        | U2/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.001 | 0.082        | U3/A   | -           | 1.000       | 10.000           |
| 0.009 | 0.091        | U3/Y   | A ^ -> Y ^  | 1.177       | 9.000            |
| 0.001 | 0.092        | FF2/D  | ->          | 1.000       | 10.000           |

```

Clock Rise Edge 0.000
= Beginpoint Arrival Time 0.000
Other End Path:

```

| Delay | Arrival Time | Pin  | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|------|------------|-------------|------------------|
| -     | 0.000        | clk  | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C6/Y | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C7/A | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C7/Y | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C8/A | -          | 1.000       | 5.000            |
| 0.000 | 0.000        | C8/Y | A ^ -> Y ^ | 0.810       | 4.000            |

# Tempus User Guide

## Analysis and Reporting

---

|       |       |        |            |       |       |
|-------|-------|--------|------------|-------|-------|
| 0.000 | 0.000 | C9/A   | -          | 1.000 | 5.000 |
| 0.000 | 0.000 | C9/Y   | A ^ -> Y ^ | 0.810 | 4.000 |
| 0.000 | 0.000 | FF2/CK | -          | 1.000 | 5.000 |

---

### ***timing\_aocv\_analysis\_mode\_combine\_launch\_capture (GBA)***

Path 1: MET Setup Check with Pin FF2/CK  
 Endpoint: FF2/D (^) checked with leading edge of 'clk'  
 Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'  
 Path Groups: {clk}  
 Other End Arrival Time 0.000  
 - Setup 0.018  
 + Phase Shift 10.000  
 = Required Time 9.982  
 - Arrival Time 0.093  
 = Slack Time 9.889  
 Clock Rise Edge 0.000  
 = Beginpoint Arrival Time 0.000  
 Timing Path:

| Delay | Arrival Time | Pin       | Arc         | Aocv Derate | Aocv Stage Count |
|-------|--------------|-----------|-------------|-------------|------------------|
| -     | 0.000        | clk       | clk ^       | -           | -                |
| 0.000 | 0.000        | C0/A      | -           | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A      | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C1/Y      | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C2/A      | -           | 1.000       | 3.000            |
| 0.000 | 0.000        | C2/Y      | A ^ -> Y ^  | 1.197       | 3.000            |
| 0.000 | 0.000        | C3/A      | -           | 1.000       | 7.000            |
| 0.000 | 0.000        | C3/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.000 | 0.000        | C4/A      | -           | 1.000       | 7.000            |
| 0.000 | 0.000        | C4/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.000 | 0.000        | C44/A     | -           | 1.000       | 7.000            |
| 0.000 | 0.000        | C44/Y     | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.000 | 0.000        | FF1/CK -> | -           | 1.000       | 7.000            |
| 0.055 | 0.055        | FF1/Q     | CK ^ -> Q ^ | 1.181       | 7.000            |
| 0.001 | 0.056        | U1/A      | -           | 1.000       | 7.000            |
| 0.014 | 0.070        | U1/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.001 | 0.071        | U2/A      | -           | 1.000       | 7.000            |
| 0.010 | 0.081        | U2/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.001 | 0.082        | U3/A      | -           | 1.000       | 7.000            |
| 0.009 | 0.092        | U3/Y      | A ^ -> Y ^  | 1.181       | 7.000            |
| 0.001 | 0.093        | FF2/D ->  | -           | 1.000       | 7.000            |

Clock Rise Edge 0.000  
 = Beginpoint Arrival Time 0.000  
 Other End Path:

| Delay | Arrival Time | Pin  | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|------|------------|-------------|------------------|
| -     | 0.000        | clk  | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y | A ^ -> Y ^ | 0.810       | 4.000            |

# Tempus User Guide

## Analysis and Reporting

---

|       |       |        |            |       |       |
|-------|-------|--------|------------|-------|-------|
| 0.000 | 0.000 | C6/A   | -          | 1.000 | 3.000 |
| 0.000 | 0.000 | C6/Y   | A ^ -> Y ^ | 0.803 | 3.000 |
| 0.000 | 0.000 | C7/A   | -          | 1.000 | 3.000 |
| 0.000 | 0.000 | C7/Y   | A ^ -> Y ^ | 0.803 | 3.000 |
| 0.000 | 0.000 | C8/A   | -          | 1.000 | 2.000 |
| 0.000 | 0.000 | C8/Y   | A ^ -> Y ^ | 0.793 | 2.000 |
| 0.000 | 0.000 | C9/A   | -          | 1.000 | 2.000 |
| 0.000 | 0.000 | C9/Y   | A ^ -> Y ^ | 0.793 | 2.000 |
| 0.000 | 0.000 | FF2/CK | -          | 1.000 | 2.000 |

---

### ***timing\_aocv\_analysis\_mode\_combine\_launch\_capture (PBA)***

Path 1: MET Setup Check with Pin FF2/CK  
 Endpoint: FF2/D (^) checked with leading edge of 'clk'  
 Beginpoint: FF1/Q (^) triggered by leading edge of 'clk'  
 Path Groups: {clk}  
 Retime Analysis { AOCV(Combine) }  
 Other End Arrival Time 0.000  
 - Setup 0.018  
 + Phase Shift 10.000  
 = Required Time 9.982  
 - Arrival Time 0.092  
 = Slack Time 9.890  
 Clock Rise Edge 0.000  
 = Beginpoint Arrival Time 0.000  
 Timing Path:

| Delay | Arrival Time | Pin       | Arc         | Aocv Derate | Aocv Stage Count |
|-------|--------------|-----------|-------------|-------------|------------------|
| -     | 0.000        | clk       | clk ^       | -           | -                |
| 0.000 | 0.000        | C0/A      | -           | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y      | A ^ -> Y ^  | 1.190       | 4.000            |
| 0.000 | 0.000        | C1/A      | -           | 1.000       | 15.000           |
| 0.000 | 0.000        | C1/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.000 | 0.000        | C2/A      | -           | 1.000       | 15.000           |
| 0.000 | 0.000        | C2/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.000 | 0.000        | C3/A      | -           | 1.000       | 15.000           |
| 0.000 | 0.000        | C3/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.000 | 0.000        | C4/A      | -           | 1.000       | 15.000           |
| 0.000 | 0.000        | C4/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.000 | 0.000        | C44/A     | -           | 1.000       | 15.000           |
| 0.000 | 0.000        | C44/Y     | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.000 | 0.000        | FF1/CK -> | -           | 1.000       | 15.000           |
| 0.055 | 0.055        | FF1/Q     | CK ^ -> Q ^ | 1.173       | 13.000           |
| 0.001 | 0.056        | U1/A      | -           | 1.000       | 15.000           |
| 0.014 | 0.069        | U1/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.001 | 0.071        | U2/A      | -           | 1.000       | 15.000           |
| 0.010 | 0.081        | U2/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.001 | 0.082        | U3/A      | -           | 1.000       | 15.000           |
| 0.009 | 0.091        | U3/Y      | A ^ -> Y ^  | 1.173       | 13.000           |
| 0.001 | 0.092        | FF2/D ->  | -           | 1.000       | 15.000           |

---

Clock Rise Edge 0.000  
 = Beginpoint Arrival Time 0.000  
 Other End Path:

| Delay | Arrival Time | Pin    | Arc        | Aocv Derate | Aocv Stage Count |
|-------|--------------|--------|------------|-------------|------------------|
| -     | 0.000        | clk    | clk ^      | -           | -                |
| 0.000 | 0.000        | C0/A   | -          | 1.000       | 4.000            |
| 0.000 | 0.000        | C0/Y   | A ^ -> Y ^ | 0.810       | 4.000            |
| 0.000 | 0.000        | C6/A   | -          | 1.000       | 15.000           |
| 0.000 | 0.000        | C6/Y   | A ^ -> Y ^ | 0.827       | 13.000           |
| 0.000 | 0.000        | C7/A   | -          | 1.000       | 15.000           |
| 0.000 | 0.000        | C7/Y   | A ^ -> Y ^ | 0.827       | 13.000           |
| 0.000 | 0.000        | C8/A   | -          | 1.000       | 15.000           |
| 0.000 | 0.000        | C8/Y   | A ^ -> Y ^ | 0.827       | 13.000           |
| 0.000 | 0.000        | C9/A   | -          | 1.000       | 15.000           |
| 0.000 | 0.000        | C9/Y   | A ^ -> Y ^ | 0.827       | 13.000           |
| 0.000 | 0.000        | FF2/CK | -          | 1.000       | 15.000           |

#### 4.3.3.9 AOCV Reporting

The Tempus software allows you to generate AOCV reports in both GBA and PBA modes.

If the `set_analysis_mode -aocv true` is set, the impact of each reporting command is as follows:

- `report_timing -retime aocv`

PBA stage counting will be used for calculating AOCV derate factor for the given path. GBA slew will be used for slew propagation.

- `report_timing -retime aocv_path_slew_propagation`

In this case, PBA stage counting will be used for calculating the AOCV derate factor for the given path. PBA slew will be used for slew propagation.

- `report_timing`

In this case, GBA stage counting will be used for calculating the AOCV derate factor for the given path. GBA slew will be used for slew propagation.

#### 4.3.3.10 Reporting with OCV/AOCV Derating

In a timing report there may be a combination of OCV and AOCV derating factors - `aocv_derate` and `user_derate` (OCV) fields of the report. An additional `stage_count` field is available to aid in better understanding the derivation of the `aocv_derate` factor.

## Tempus User Guide

### Analysis and Reporting

In the previous example of an additive combination of OCV and AOCV, one of the reference points was 0 and the other was 1. In such cases, the software does not make any adjustments to normalize the final derate factor so the timing report columns are quite intuitive.

In the timing run below, the nominal delay for all arcs has been set to 1.0ns so that the value in the Delay field indicates the total derate factor applied.

```
set_global report_timing_format {instance cell arc stage_count aocv_derate
user_derate delay arrival}
report_timing -net ...
```

| Instance | Cell   | Arc         | Aocv  | Aocv   | User   | Delay | Arrival |
|----------|--------|-------------|-------|--------|--------|-------|---------|
|          |        |             | Stage | Derate | Derate |       | Time    |
|          |        |             | Count |        |        |       |         |
| RS       | DFFX1  |             | 7.000 | 1.030  | 1.000  | 2.030 | 20.450  |
| RS       | DFFX1  | CK ^ -> Q v | 6.000 | 1.030  | 0.010  | 1.040 | 21.490  |
| U1       | BUFXXX |             | 7.000 | 1.030  | 1.000  | 2.030 | 23.520  |
| U1       | BUFXXX | A v -> Y v  | 6.000 | 1.030  | 0.030  | 1.060 | 24.580  |

If the derating factors are changed such that both AOCV and OCV are referenced to 1.0, then an implicit -1 will be used since the mode is still aocv\_additive. In this case, the effect of the '-1' will be seen in the aocv\_derate value.

For example, set both the reference points to 0, and modify the AOCV library so that it outputs a value of 0.030ns.

```
set_global timing_derate_aocv_reference_point 0
set_global timing_derate_ocv_reference_point 0
set_timing_derate -delay_corner dcMax -late -cell_delay 1.01
set_timing_derate -delay_corner dcMax -late -cell_delay 0.02 -add [get_cells U1]
```

| Instance | Cell   | Arc         | Aocv  | Aocv   | User   | Delay | Arrival |
|----------|--------|-------------|-------|--------|--------|-------|---------|
|          |        |             | Stage | Derate | Derate |       | Time    |
|          |        |             | Count |        |        |       |         |
| RS       | DFFX1  |             | 7.000 | 1.030  | 1.000  | 2.030 | 20.450  |
| RS       | DFQX1  | CK ^ -> Q v | 6.000 | 1.030  | 0.010  | 1.040 | 21.490  |
| U1       | BUFXXX |             | 7.000 | 1.030  | 1.000  | 2.030 | 23.520  |
| U1       | BUFXXX | A v -> Y v  | 6.000 | 1.030  | 0.030  | 1.060 | 24.580  |

In this case, the +1.0 adjustment to normalize the final derating factor is again accounted for in the aocv\_derate column.

## AOCV Derate and Stage Count Properties of Timing Arcs

You can use stage count and AOCV derate properties to report and query on AOCV data. As an example consider the following path:

```
report_timing -format {arc hpin aocv_derate stage_count} -path_type full_clock
```

| Endpoint:       | lat3/D   |             |                  |
|-----------------|----------|-------------|------------------|
| Beginpoint:     | in2      |             |                  |
| <hr/>           |          |             |                  |
| Arc             | Pin      | Aocv Derate | Aocv Stage Count |
| in2 v           | in2      |             |                  |
| A v -> Y v      | buf133/Y | 1.217       | 3.000            |
| A v -> Y v      | buf5/Y   | 1.217       | 3.000            |
| A v -> Y v      | buf0/Y   | 1.217       | 3.000            |
| D v             | lat3/D   | 1.217       | 3.000            |
| <hr/>           |          |             |                  |
| Other End Path: |          |             |                  |
| Arc             | Pin      | Aocv Derate | Aocv Stage Count |
| clk^            | clk      |             |                  |
| A ^ -> Y ^      | buf11/Y  | 0.888       | 2.000            |
| A ^ -> Y v      | invl2/Y  | 0.888       | 2.000            |
| GN v            | lat3/GN  | 0.888       | 2.000            |
| <hr/>           |          |             |                  |

For example, obtaining the various stage counts corresponding to the above path is shown below:

```
get_property [get_arcs -from buf133/A -to buf133/Y] aocv_stage_count_data_late
3.000
get_property [get_arcs -from buf11/A -to buf11/Y]
aocv_stage_count_capture_clock_early
```

To obtain derate corresponding to the above path:

```
get_property [get_arcs -from buf11/A -to buf11/Y]
aocv_derate_capture_clock_early_rise 0.888
```

For more information on AOCV derate properties, refer to [report\\_property](#) and [get\\_property](#) commands in the *Tempus Text Command Reference*.

## 4.3.4 Statistical On-Chip Variation (SOCV) Analysis

- 4.3.4.1 [Overview](#) on page 166
- 4.3.4.2 [Introduction to SOCV](#) on page 166
- 4.3.4.3 [SOCV Data Flow](#) on page 167
- 4.3.4.4 [SOCV Analysis](#) on page 169
- 4.3.4.5 [Enabling SOCV Analysis](#) on page 169
- 4.3.4.6 [SOCV Analysis Reporting](#) on page 170
- 4.3.4.7 [Sample Variation on Delay Template from Liberty 2013.12](#) on page 177
- 4.3.4.8 [Timing Behaviors in SOCV Analysis Mode](#) on page 179
- 4.3.4.9 [Advanced Timing Tcl Scripting in SOCV Analysis Mode](#) on page 179

#### 4.3.4.1 Overview

Industrial demand for higher performing digital semiconductor products has highlighted the clear need to reduce design guard-band without sacrificing time-to-market. Improvements to on-chip variation modeling are a driving component of the guard band reduction strategy. In this section we review the current industry approaches to on-chip variation (OCV) modeling and introduce the Statistical OCV (SOCV) analysis, which utilizes a single statistical parameter representing sigma variation. The SOCV extends existing single variable statistical approaches by adding slew and load-dependent sigma per timing arc.

For many years, modeling of the on-chip process variation has been achieved through scaling of the nominal delays of all instances with a single derating factor. Such a uniform derating, often referred to as OCV derate, often causes excessive pessimism for some instances while optimism for others. In order to guarantee conservative analysis, OCV derates were chosen pessimistically, resulting in over-design and increase of turnaround times for timing closure.

To overcome these setbacks, an advanced OCV (AOCV) approach was introduced. With AOCV the derating factor is instance specific and depends on the number of logic levels in a path and/or location of the instance. The AOCV derating factors are pre-characterized for each cell typically using Monte-Carlo simulation. This methodology reduced pessimism and optimism of the traditional OCV methodology. However, AOCV presented challenges for computing of the stage count and distance of the path on a die for a specific path in graph-based analysis (GBA). In addition, AOCV assumes that the ratio of delay variation to nominal delay does not depend on the input slew and load.

On the opposite spectrum of the methods modeling variability in STA is statistical static timing analysis (SSTA). It models delay distributions due to local and global random parameter variations accurately and is, therefore, much more accurate than other mentioned methods. However, SSTA is much more expensive in terms of run time and memory, and requires special timing libraries, which is often too high cost to pay for design houses and foundries.

#### 4.3.4.2 Introduction to SOCV

Statistical OCV (SOCV) is a new analysis technique that provides for a good compromise between the run time and accuracy for modeling variation.

SOCV computes the impact of local process variations on the delay and slew of each instance in the design at a given global variation corner. SOCV uses one parameter in SSTA mode. This single parameter models local cell variations. Similar to SSTA, SOCV propagates the sigma of arrival and required times through the timing graph, and then computes statistical characteristics of slack at all timing pins. Modeling on-chip variations in terms of a single variable is fast, yet it yields better accuracy than modeling the variations as derating factors.

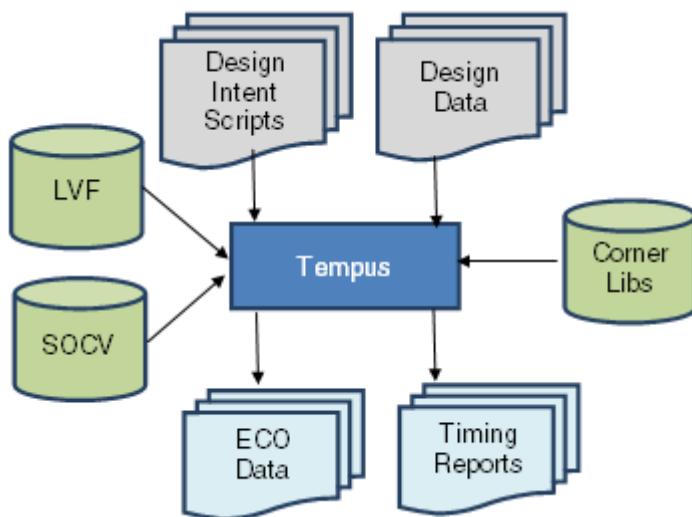
For accurate SOCV analysis, a special library data is required. It includes arc-level absolute variations of cell delays, output transitions, and timing checks as functions of input slew and load. Since SOCV is able to accurately compute the delay and slew variation on each instance, it solves the two main limitations of simpler methods: (i) cases where delay is close to zero while delay variation is not, and (ii) impact of input slew variations on delay.

SOCV analysis is currently available in the Cadence Tempus Timing Signoff Solution for sign-off timing analysis and timing closure.

#### 4.3.4.3 SOCV Data Flow

SOCV analysis requires only the addition of variation libraries to the typical timing sign-off and closure configuration. Variation data can be provided either as a Cadence SOCV library or as Liberty Variation Format (LVF) library. Tempus fully supports both library formats including modeling for variation on delay, transition, and constraints.

**Figure 4-71 SOCV Data Flow**



#### Reading Variation Library Data

Tempus provides both a single-corner and MMMC use model, which can accommodate SOCV or LVF format variation libraries.

In single-corner mode, you can use the following parameter to read Cadence SOCV format libraries:

```
read_lib -socv [list libA.socv libB.socv ...]
```

To read Liberty libraries with integrated LVF data, you should read the library in the same manner as a non-LVF Liberty timing library.

```
read_lib [list libA-LVF.lib libB-LVF.lib ...]
```

When working in an MMMC environment, you should specify the variation libraries as part of the create\_library\_set command in your MMMC configuration file. To incorporate Cadence SOCV format libraries:

```
create_library_set -name libsetMax
-timing [list
${library_path_1}/lib1.lib
${library_path_1}/lib2.lib
]
-socv [list
${socv_lib_path}/lib1.socv
${socv_lib_path}/lib2.socv
]
```

To use Liberty libraries with merged LVF data, simply load the Liberty libraries with the create\_library\_set -timing parameter.

## Using AOCV Derating Libraries with SOCV Analysis

Tempus allows the option of using AOCV libraries to provide variation modeling data for the SOCV flow. This method can be used for initial experimentation and testing of the SOCV flow, but lacks the accuracy required for signoff. Using AOCV as a source will provide only variation on delay; no variation on transition or constraints.

To utilize the AOCV bootstrap method, you must set the timing global variables to control transformation of AOCV data. These settings must be made before loading any timing library data.

To enable the transformation process, use the following command:

```
set timing library infer socv from aocv true
```

The AOCV derating libraries are characterized to a particular sigma value - without additional guidance the transformation process will consider the AOCV library to be at 3-sigma. To define the characterization sigma of the AOCV libraries, you can make the following setting:

```
set timing library scale aocv to socv to n sigma 4.5
```

You should specify reading the AOCV libraries in the same way as for normal AOCV analysis.

For single-corner mode, use the following command:

```
read lib -aocv {myLib.aocv ...}
```

For a MMMC environment, you can make the following settings:

```
create_library_set -name libsetMax
-timing [list
 ${library_path_1}/lib1.lib
 ${library_path_1}/lib2.lib
]
-aocv [list
 ${aocv_lib_path}/lib1.aocv
 ${aocv_lib_path}/lib2.aocv
]
```

#### 4.3.4.4 SOCV Analysis

The SOCV analysis is essentially an optimized, single parameter statistical analysis, that is built on the underlying Tempus SSTA analysis engine. From a user-interface point of view, the software is able to present results in terms of discrete values for delays, arrivals, and slacks. The interface can also show statistical representation of data.

##### Enabling SOCV Analysis

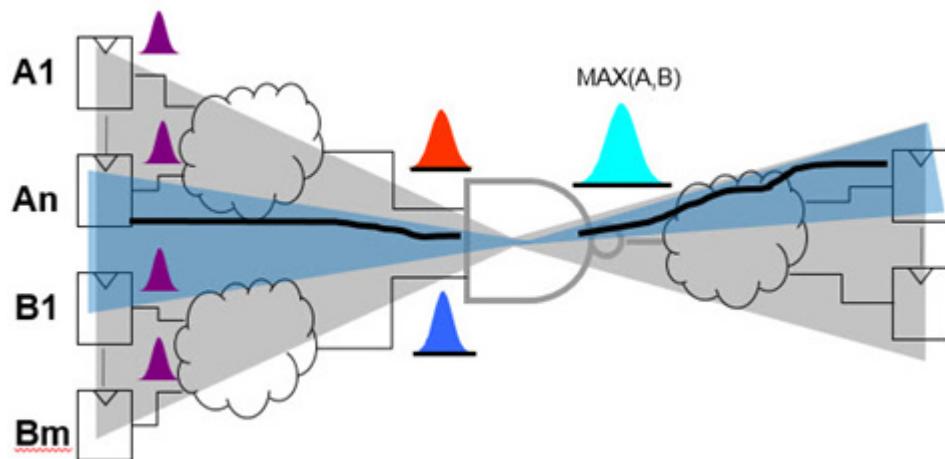
The SOCV analysis can be enabled using the set\_analysis\_mode -socv parameter, as shown below:

```
set_analysis_mode -socv true -analysisType onChipVariation -cppr both
```

##### Configuring SOCV Max Operation

During arrival and required timing propagation, timing data values are represented concurrently by statistical mean and sigma values. At the output of multi-input gates, the statistical arrival times must be merged in a conservative operation. There are several options for controlling the statistical Max operation.

**Figure 4-72 SOCV Max Operation**



Selection of the Max operation can be controlled by making the following setting:

```
set timing_socv_statistical_min_max_mode mean_and_three_sigma_bound
```

In this mode, the worst-case calculations of mean value and standard deviation are performed separately.

#### 4.3.4.5 SOCV Analysis Reporting

Reports in SOCV analysis mode will, by default, present timing data in discrete numbers rather than as statistical quantities. If desired, the timing report can be configured to report the statistical mean and sigma of the various timing data. The format of the timing report can be customized by a combination of global settings and `report_timing -format` options.

All the timing calculations are performed in the statistical domain. For example, adding delays along a path to determine an arrival time calculation of slack-based on subtraction of arrival and required times. Discrete representations of these values are computed as mean + n\*sigma – where n is typically 3.0.

##### Path Reporting With Discrete Values

The default timing report is configured to report timing quantities as discrete numbers rather than as statistical quantities. The `report_timing` fields: slew, delay, and arrival will still report discrete numeric values.

As a result of calculating arrival times in the statistical realm, adding the current delay to the previous arrival will generally not add up exactly to the next arrival time in the report:

| Timing Point | Edge | Delay   | Arrival Time       |
|--------------|------|---------|--------------------|
| in_4         | v    | 0.50000 |                    |
| buff_4_1/Z   | v    | 0.02784 | 0.52784            |
| buff_4_2/Z   | v    | 0.02807 | 0.55383 != 0.55591 |
| buff_4_3/Z   | v    | 0.02807 | 0.57947 != 0.58190 |
| buff_4_4/Z   | v    | 0.02912 | 0.60590            |
| out_4 ->     | v    | 0.00000 | 0.60590            |

The statistical mean and sigma representation of timing data is converted to discrete values by the formula: mean + n-sigma, where n is a user-specifiable value. To control the sigma multiplier, you can use the `set_socv_reporting_nsigma_multiplier` command. This command supports n-sigma specification per view per setup/hold combination.

To apply sigma multiplier to the specified view in setup mode, use the following command:

```
set_socv_reporting_nsigma_multiplier -setup 2.5 -view std_max_setup
```

## Path Reporting with Statistical Data

The `report_timing` command now supports additional statistical report fields when operating in SOCV mode.

- The delay\_mean and delay\_sigma fields provide statistical data for each timing arc
- The slew\_mean and slew\_sigma fields provide data for transitions
- The arrival\_mean and arrival\_sigma fields provide the statistical information for arrival timing

The conversion of statistical quantities in the report to discrete values is computed by:

- New mean is the sum of the means:  $\mu_3 = \mu_1 + \mu_2$

## Tempus User Guide

### Analysis and Reporting

---

- New sigma is the root-sum-squared of sigmas:  $\sigma_3 = \sigma_1^2 + \sigma_2^2$

| report_timing Output With Statistical Fields and Discrete Conversion at 3-sigma |      |            |             |                           |              |               |                             |
|---------------------------------------------------------------------------------|------|------------|-------------|---------------------------|--------------|---------------|-----------------------------|
| Timing Point                                                                    | Edge | Delay Mean | Delay Sigma | Delay ( $\mu + 3\sigma$ ) | Arrival Mean | Arrival Sigma | Arrival ( $\mu + 3\sigma$ ) |
| in 4                                                                            | v    |            |             |                           | 0.50000      | 0.00000       | 0.50000                     |
| buff 4 1/Z                                                                      | v    | 0.02430    | 0.00118     | 0.02784                   | 0.52430      | 0.00118       | 0.52784                     |
| buff 4 2/Z                                                                      | v    | 0.02450    | 0.00119     | 0.02807                   | 0.54880      | 0.00168       | 0.55383                     |
| buff 4 3/Z                                                                      | v    | 0.02450    | 0.00119     | 0.02807                   | 0.57330      | 0.00206       | 0.57947                     |
| buff 4 4/Z                                                                      | v    | 0.02540    | 0.00124     | 0.02912                   | 0.59870      | 0.00240       | 0.60590                     |
| out 4 ->                                                                        | v    | 0.00000    | 0.00000     | 0.00000                   | 0.59870      | 0.00240       | 0.60590                     |

|                                                                                              |                                                                                           |                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\mu_1 \quad \sigma_1$ $(0.52430, 0.00118)$ <p style="color: red;">Previous Arrival Time</p> | $\mu_2 \quad \sigma_2$ $(0.02450, 0.00119)$ <p style="color: green;">Next Stage Delay</p> | $(\text{sum of mean's})$ $\mu_1 + \mu_2$ $= (0.52430 + 0.02450), (\sqrt(0.00118^2 + 0.00119^2))$ $= (0.54880, 0.00168)$ <p style="color: blue;">New Arrival Time</p> |
|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Summary Section Reporting

The summary section of the `report_timing` report is where the final arrival and required times are calculated, and the path slack is determined. Other quantities such as CPPR, Uncertainty, and Setup/Hold checks values are also accounted in this section's calculations. Like the detailed path report, all calculations are initially performed using statistical arithmetic of the mean and sigma values. The discrete representations of these values are reported based on the via mean +n-sigma conversion.

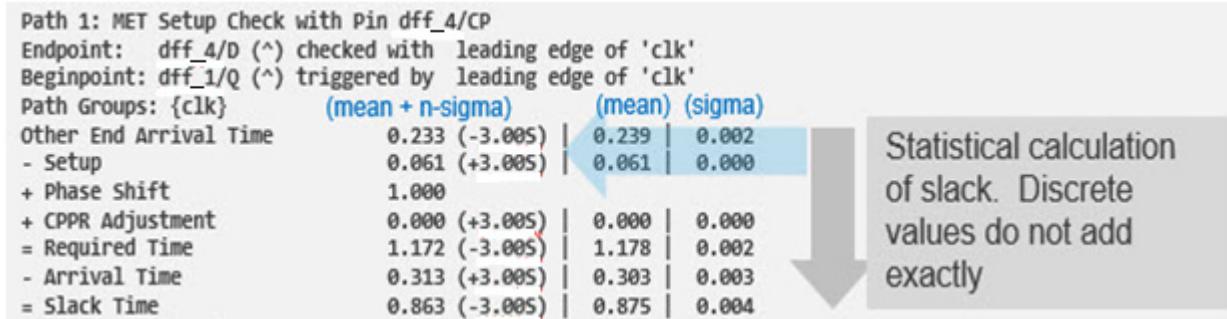
By default, the `report_timing` command output will display discrete representations of the data in the summary section of the report.

```
Path 1: MET Setup Check with Pin dff 4/CP
Endpoint: dff 4/D (^) checked with leading edge of 'clk'
Beginpoint: dff 1/Q (^) triggered by leading edge of 'clk'
Path Groups: {clk}
Other End Arrival Time 0.233
- Setup 0.061
+ Phase Shift 1.000
+ CPPR Adjustment 0.000
= Required Time 1.172
- Arrival Time 0.313
= Slack Time 0.863
```

To enable the display of related statistical representation of data in the summary section, you can set the following global variable to `true`:

`timing_report_enable_verbose_ssta_mode`

With this setting, additional details will be added to the `report_timing` summary section. The current “n” value for n-sigma conversion is shown, along with the mean and sigma components for each data value.



| Path Groups: {clk}     | (mean + n-sigma) | (mean) | (sigma) |
|------------------------|------------------|--------|---------|
| Other End Arrival Time | 0.233 (-3.005)   | 0.239  | 0.002   |
| - Setup                | 0.061 (+3.005)   | 0.061  | 0.000   |
| + Phase Shift          | 1.000            |        |         |
| + CPPR Adjustment      | 0.000 (+3.005)   | 0.000  | 0.000   |
| = Required Time        | 1.172 (-3.005)   | 1.178  | 0.002   |
| - Arrival Time         | 0.313 (+3.005)   | 0.303  | 0.003   |
| = Slack Time           | 0.863 (-3.005)   | 0.875  | 0.004   |

## Sample Initialization and Configuration Scripts

### ***Single-Corner Use Model (non-MMMC) - SOCV Native Libraries***

```
set timing_socv_statistical_min_max_mode three sigma_bound
set_socv_reporting_nsigma_multiplier -setup 2.5
set_timing_report_enable_verbose_ssta_mode true

set report_timing_format {timing_point edge cell slew load \
delay_mean delay_sigma delay \
arrival_mean arrival_sigma arrival}

read_lib ${LIB_DIR}/standard.lib
read_lib -socv [list ${LIB_DIR}/socv/standard.socv]

read_verilog ./test.v
set_top_module test
read_sdc ./test.sdc

set_analysis_mode -socv true -cppr both
report_timing
```

### ***Single-Corner Use Model (non-MMMC) - LVF Native Libraries***

```
set timing_socv_statistical_min_max_mode three sigma_bound
set_socv_reporting_nsigma_multiplier -setup 2.5
set_timing_report_enable_verbose_ssta_mode true

set report_timing_format {timing_point edge cell slew load \
delay_mean delay_sigma delay \
arrival_mean arrival_sigma arrival}

read_lib ${LIB_DIR}/standard-lvf.lib
read_verilog ./test.v
set_top_module test
read_sdc ./test.sdc

set_analysis_mode -socv true -cppr both
report_timing
```

### ***Single-Corner Use Model (non-MMMC) - AOCV Bootstrap Mode***

When running in AOCV bootstrap mode, load the AOCV libraries. Before loading any library information, you must ensure that the required timing library global variables have been set.

```
set timing_library_infer_socv_from_aocv true
set timing_library_scale_aocv_to_socv_to_n_sigma 3
set timing_socv_statistical_min_max_mode three_sigma_bounded
set_socv_reporting_nsigma_multiplier -setup 2.5
set timing_report_enable_verbose_ssta_mode true
set report_timing_format \
{timing_point edge cell slew load \
delay_mean delay_sigma delay \
arrival_mean arrival_sigma arrival}
read_lib ${LIB_DIR}/standard.lib
read_lib -aocv [list ${LIB_DIR}/aocv/standard.aocv]
read_verilog ./test.v
set_top_module test
read_sdc ./test.sdc
set_analysis_mode -socv true -cppr both
report_timing
```

### ***MMMC Use Model Using Native SOCV or LVF Libraries***

In MMMC configurations, library data is provided through a MMMC library set object, using the create\_library\_set command.

In the following examples native SOCV or LVF format library is used to make changes in the MMMC configuration file.

```
set timing_socv_statistical_min_max_mode three_sigma_bounded
set_socv_reporting_nsigma_multiplier -setup 2.5
set timing_report_enable_verbose_ssta_mode true
set report_timing_format {timing_point edge cell slew load \
delay_mean delay_sigma delay \
arrival_mean arrival_sigma arrival}
read_view_definition ./viewDef.tcl
read_verilog ./test.v
set_top_module test
set_analysis_mode -socv true -cppr both
report_timing
```

### ***Common MMMC Initialization Script With Native SOCV or LVF***

```
create_library_set -name libsetMax \
-timing [list ${library_path_1}/lib1.lib ${library_path_1}/lib2.lib] \
-socv [list \
 ${socv_lib_path}/lib1.socv \
 ${socv_lib_path}/lib2.socv \
]
```

### **Portion of MMMC Configuration Script (viewDef.tcl) for Loading Native SOCV Libraries**

```
create_library_set -name libsetMax \
-timing [list \
 ${library_path_1}/lib1-lvf.lib \
 ${library_path_1}/lib2-lvf.lib \
] \
```

### **Portion of MMMC Configuration Script (viewDef.tcl) For Loading AOCV Libraries**

#### **MMMC Use Model– AOCV Bootstrap Mode**

When running in AOCV bootstrap mode, load the AOCV libraries. You must ensure that the required timing library global variables are set before loading any library information.

```
set timing_library_infer_socv_from_aocv true
set timing_library_scale_aocv_to_socv_to_n_sigma 3
set timing_socv_statistical_min_max_mode three_sigma_bounded
set_socv_reporting_nsigma_multiplier -setup 2.5
set_timing_report_enable_verbose_ssta_mode true
set report_timing_format {timing_point edge cell slew load \
delay_mean delay_sigma delay \
arrival_mean arrival_sigma arrival}
read_view_definition ./viewDef.tcl
read_verilog ./test.v
set_top_module test
set_analysis_mode -socv true -cppr both
report_timing
```

#### **MMMC Initialization Script for AOCV Bootstrap Mode**

```
create_library_set -name libsetMax \
-timing [list \
 ${library_path_1}/lib1.lib \
 ${library_path_1}/lib2.lib \
] \
-aocv [list \
 ${aocv_lib_path}/lib1.aocv \
 ${aocv_lib_path}/lib2.aocv \
]
```

## Variation Library Formats

The Cadence SOCV library format and the standardized Liberty Variation Format (LVF) provide essentially the same information: variation sigma for delay, transition, and constraints per arc, indexed by load and slew.

### ***Liberty Variation Format (LVF)***

The detailed specifications of Liberty LVF for variation on delay and transition are available from OpenSource Liberty Version 2013.12.

```
cell (cell_name) {
 ocv_derate_distance_group: ocv_derate_group_name;
 ...
 pin | bus | bundle (name) {
 direction: input | output;
 timing() {
 ...
 ocv_sigma_cell_rise(delay_lu_template_name) {
 sigma_type: early | late | early_and_late;
 index_1 ("float, ..., float");
 index_2 ("float, ..., float");
 values ("float, ..., float", \
 ...
 "float, ..., float");
 }
 ocv_sigma_cell_fall(delay_lu_template_name) {
 sigma_type: early | late | early_and_late;
 index_1 ("float, ..., float");
 index_2 ("float, ..., float");
 values ("float, ..., float", \
 ...
 "float, ..., float");
 }
 ...
 } /* end of timing */
 ...
 } /* end of pin */
 ...
}
```

#### 4.3.4.6 Sample Variation on Delay Template from Liberty 2013.12

##### Cadence SOCV Library Format (.socv)

The SOCV library format is as follows:

```
object_spec: libXYZ/NAND2
pin: Y
pin_direction: rise
related_pin: A
related_direction: rise
type: delay
when:
related_transition: 0.0002, 0.0050, 0.0125, 0.0500
output_loading: 0.0002, 0.0015, 0.0035, 0.0090
sigma:
 0.0029, 0.0034, 0.0042, 0.0066
 0.0035, 0.0036, 0.0044, 0.0065
 0.0039, 0.0043, 0.0050, 0.0072
 0.0061, 0.0066, 0.0071, 0.0088
```

Variation  
On Delay

```
object_spec: libXYZ/NAND2
pin: Y
pin_direction: rise
related_pin: A
related_direction: rise
type: slew
when:
related_transition: 0.0002, 0.0050, 0.0125, 0.0500
output_loading: 0.0002, 0.0015, 0.0035, 0.0090
distance:
sigma:
 0.0006, 0.0024, 0.0037, 0.0072
...
```

Variation  
on Slew

```
object_spec: libXYZ/DFF
pin: D
pin_direction: rise
related_pin: CK
related_direction: rise
type: hold
when:
related_transition: 0.0002, 0.0200, 0.0500, 0.1200
constraint_transition: 0.0002, 0.0200, 0.0400, 0.0900
distance:
sigma:
 0.0762, 0.0756, 0.0749, 0.0732
...
```

Variation  
on Checks

**Tempus User Guide**  
Analysis and Reporting

---

The following table describes the SOCV Library Attributes:

| SOCV Library Attribute | Attribute Type                                   | Description                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| object_type            | library   design                                 | Use library to indicate that the SOCV data is with respect to library cells. Use design, for design-specific data such as spatial derate multipliers.                                                                                                                                                                                                                                                    |
| object_spec            | libName/<br>libCellName                          | <p>Specifies the internal library name and library cell name that is consistent with a Liberty timing library that has been loaded in the session.</p> <p>The SOCV library is considered an adjunct library file which inherits certain attributes – such as PVT from the master Liberty library.</p> <p>Simple wildcard expressions are allowed for both the <i>libName</i> and <i>libCellName</i>.</p> |
| pin                    | pinName                                          | The output or constrained pin name of the arc being represented by this section.                                                                                                                                                                                                                                                                                                                         |
| pin_direction          | rise   fall                                      | The direction of the transition at the output pin.                                                                                                                                                                                                                                                                                                                                                       |
| related_pin            | pinName                                          | The related pin name of the arc being represented by this section.                                                                                                                                                                                                                                                                                                                                       |
| related_direction      | rise   fall                                      | The direction of the transition at the related pin.                                                                                                                                                                                                                                                                                                                                                      |
| type                   | delay   slew   setup   hold   recovery   removal | Indicates whether the current arc section is modeling variation on delay, transition, or constraints.                                                                                                                                                                                                                                                                                                    |
| when                   | Liberty compliant ‘when’ attribute string        | The Liberty ‘when’ attribute string consistent with the same timing arc in the Liberty library.                                                                                                                                                                                                                                                                                                          |
| related_transition     | float+                                           | Comma-separated list of transition indices on the arc’s related_pin.                                                                                                                                                                                                                                                                                                                                     |
| output_loading         | float+                                           | Comma-separated list of output load indices.                                                                                                                                                                                                                                                                                                                                                             |
| constraint_transition  | float+                                           | For constraint data, a comma-separated list of transitions on the arc’s constrained pin.                                                                                                                                                                                                                                                                                                                 |

|                       |                    |                                                                      |
|-----------------------|--------------------|----------------------------------------------------------------------|
| sigma                 | 2-D table of float | Table of variation data indexed by transition and load.              |
| late_distance         | float+             | Comma-separated list of distance values.                             |
| late_distance_derate  | float+             | 1-D table of spatial derating factors for application to late paths. |
| early_distance        | float+             | Comma-separated list of distance values.                             |
| early_distance_derate | float+             | 1-D table of spatial derating factors for application to late paths. |

#### 4.3.4.7 Timing Behaviors in SOCV Analysis Mode

- All timing constraint assertions (SDCs) are interpreted as a specification of the mean value with a sigma value of 0.0.
- By default, the set\_timing\_derate factors are applied to both the mean and sigma values. You can use the -mean/-sigma parameters to apply separate derating to the mean/sigma components of the delay.
- The read\_sdf values are interpreted as annotations to the mean value – the sigma value is 0.0.
- The delay values written out by write\_sdf are the mean + n-sigma representation. If the selected analysis type is SOCV, you can use the -no\_variation parameter to export the SDF delay and check values without including the variation component.
- All timing reports are in terms of the mean + n-sigma discrete transformation of the value.
- The report\_delay\_calculation command by default reports the mean component of delay.

#### 4.3.4.8 Advanced Timing Tcl Scripting in SOCV Analysis Mode

The following properties are added to the timing\_arc object and can be specified using the get\_property command:

- delay\_mean\_max\_fall
- delay\_mean\_max\_rise
- delay\_mean\_min\_fall
- delay\_mean\_min\_rise
- delay\_sigma\_max\_fall

- delay\_sigma\_max\_rise
- delay\_sigma\_min\_fall
- delay\_sigma\_min\_rise

The following properties are added to the pin object and can be queried using the [get\\_property](#) command:

- arrival\_mean\_max\_fall
- arrival\_mean\_max\_rise
- arrival\_mean\_min\_fall
- arrival\_mean\_min\_rise
- arrival\_sigma\_max\_fall
- arrival\_sigma\_max\_rise
- arrival\_sigma\_min\_fall
- arrival\_sigma\_min\_rise
- slack\_mean\_max
- slack\_mean\_max\_fall
- slack\_mean\_max\_rise
- slack\_mean\_min
- slack\_mean\_min\_fall
- slack\_mean\_min\_rise
- slack\_sigma\_max
- slack\_sigma\_max\_fall
- slack\_sigma\_max\_rise
- slack\_sigma\_min
- slack\_sigma\_min\_fall
- slack\_sigma\_min\_rise
- slew\_mean\_max\_fall
- slew\_mean\_max\_rise
- slew\_mean\_min\_fall
- slew\_mean\_min\_rise

- slew\_sigma\_max\_fall
- slew\_sigma\_max\_rise
- slew\_sigma\_min\_fall
- slew\_sigma\_min\_rise

## 4.4 Path-Based Analysis

- 4.4.1 [Overview](#) on page 183
- 4.4.2 [GBA vs PBA Comparison](#) on page 183
- 4.4.3 [Reporting in Path-Based Analysis](#) on page 183
- 4.4.4 [Path-Based Analysis Essence](#) on page 186

#### 4.4.1 Overview

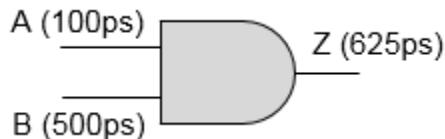
Static Timing Analysis (STA) software by default uses a pessimistic algorithm for timing, which is based on the worst slew propagation (slew merging), referred to as graph-based analysis (GBA). In GBA mode, the software considers both the worst arrival and the worst slew in a path during timing analysis, even if the worst slew is corresponding to a input pin different than the relevant pin for the current path. This gives a pessimistic result.

Path-Based Analysis (PBA) involves re-timing the components of a timing path based on the actual slew that is propagated in this path. This technique primarily aims to remove the pessimism that is introduced due to slew merging at various nodes in the design when graph-based analysis is run (represented in timing graph as nodes).

#### 4.4.2 GBA vs PBA Comparison

When path-based analysis is run, the slew merging is not done and the actual slew on the timing arc of path is propagated. Thus, path-based analysis re-calculates timing in a timing path without considering side path slews that might otherwise affect the arrival times and slew values used in the calculation.

To understand slew merging in GBA, consider an example of a AND gate below.



Here, it is assumed that for any input slew, the output slew is 25% more than the input slew. So for a slew of 100ps at input A, corresponding output slew at Z is 125ps. Similarly, if slew at B is 500ps, then slew at Z is 625ps. In GBA for calculating delay and slew propagation through this AND gate, the worse input slew (through B) is always considered, irrespective of the fact that the path is through pin A or B. But in PBA, the actual slew through the concerned pin is accounted for in delay calculation and slew propagation.

#### 4.4.3 Reporting in Path-Based Analysis

##### Commands for PBA Reporting

You can use the following commands to report violations in PBA mode:

```
report_constraint -retime <option>
report_timing -retime <option>
```

**Note:** To generate the analysis summary file, you can use the following options with the `report_timing` command - to give high value of nworst and max\_path to get good coverage. The report summary will be stored in the pba.summ file.

```
report_timing -retiming path_slew_propagation -max_paths <> -nworst <> -path_type
summary_slack_only -<late/early> -analysis_summary_file pba.summ
```

### Sample Report Showing GBA vs PBA Timing Results

The following example demonstrates GBA vs PBA timing results. Given below is the GBA report through “A” pin of a AND gate “ua1”:

```
Path 1: VIOLATED Setup Check with Pin u5/CK
Endpoint: u5/D (^) checked with leading edge of 'CLK_W_4'
Beginpoint: u2/Q (^) triggered by leading edge of 'CLK_W_4'
Other End Arrival Time 0.453
- Setup 0.210
+ Phase Shift 10.000
+ CPPR Adjustment 0.000
- Uncertainty 10.000
= Required Time 0.243
- Arrival Time 1.587
= Slack Time -1.344
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 0.455
= Beginpoint Arrival Time 0.455

Load Slew Delay Arrival Cell Arc Pin Required
Time Time

0.205 0.178 - 0.455 - CK ^ u2/CK -0.889
0.020 0.107 0.268 0.723 DFF CK ^ -> Q ^ u2/Q -0.621
0.020 0.107 0.002 0.725 BUF - u3/A -0.619
0.011 0.093 0.075 0.800 BUF A ^ -> Y ^ u3/Y -0.544
0.011 0.093 0.000 0.800 AND - ua1/A -> -0.544
0.011 2.002 0.071 0.871 AND A ^ -> Y ^ ua1/Y -0.473
0.011 2.002 0.000 0.871 BUF - u4/A -0.473
0.006 1.951 0.716 1.587 BUF A ^ -> Y ^ u4/Y 0.243
0.006 1.951 0.000 1.587 DFF - u5/D 0.243
```

The slew through “A” pin of the AND gate “ua1” is 0.093ns and through B pin is 1.068ns (not shown in the above report). During GBA, the software performs slew merging and uses 1.068ns as the slew through “A” pin. Hence, a lot of pessimism is added in this analysis, which can be removed if PBA is used.

To enable PBA mode, you can use the following command:

```
report_timing -retiming path_slew_propagation
```

Given below is the result of PBA analysis of the above path. The highlighted red numbers show how PBA numbers remove pessimism induced due to slew merging in GBA.

**Note:** To report re-timed numbers, you can add `retime_slew`, `retime_delay`, and `retime_incr_delay` options to the `report_timing_format` global variable or use the `with the report_timing -format parameter`.

```
Path 1: VIOLATED Setup Check with Pin u5/CK
Endpoint: u5/D (^) checked with leading edge of 'CLK_W_4'
Beginpoint: u2/Q (^) triggered by leading edge of 'CLK_W_4'
Retime Analysis { Path-Slew }
Other End Arrival Time 0.453
- Setup 0.068
+ Phase Shift 10.000
+ CPPR Adjustment 0.000
- Uncertainty 10.000
= Required Time 0.385
- Arrival Time 0.934
= Slack Time -0.549 --> PBA Slack
= Slack Time(original) -1.344 --> GBA Slack
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 0.455
= Beginpoint Arrival Time 0.455

Load Slew Retime Delay Retime Arrival Cell Arc Pin Required
 Slew Delay Time
----- Time
0.205 0.178 0.178 - - 0.455 - CK ^ u2/CK -0.094
0.020 0.107 0.107 0.268 0.268 0.723 DFF CK ^ -> Q^ u2/Q 0.174
0.020 0.107 0.107 0.002 0.002 0.725 BUF - u3/A 0.176
0.011 0.093 0.093 0.075 0.075 0.800 BUF A ^ -> Y ^ u3/Y 0.251
0.011 0.093 0.093 0.000 0.000 0.800 AND - ua1/A -> 0.251
0.011 2.002 0.079 0.071 0.071 0.871 AND A ^ -> Y ^ ua1/Y 0.322
0.011 2.002 0.079 0.000 0.000 0.871 BUF - u4/A 0.322
0.006 1.951 0.062 0.716 0.064 0.934 BUF A ^ -> Y ^ u4/Y 0.385
0.006 1.951 0.062 0.000 0.000 0.934 DFF - u5/D 0.385

```

**Note:** The `timing_disable_retime_clock_path_slew_propagation` global variable controls whether retiming is done for both data as well as clock paths.

#### 4.4.4 Path-Based Analysis Reporting Models

There are two types of path-based analysis reporting models:

- Regular
- Exhaustive

##### Regular

In the regular model of PBA reporting, you can use the `report_timing` command, as shown below:

```
report_timing -retime <retime type> -max_paths 'N' -nworst 'M' -max_slack 'S'
```

Tempus performs the following steps:

1. Collect top ‘N’ critical paths with GBA slack less than ‘S’ with maximum of ‘M’ paths per endpoint picked.
2. Perform PBA on the paths from the above step.
3. Report paths with PBA\_Slack < ‘S’, sorted based on GBA slacks.

### **Exhaustive**

Due to the impact of path-based analysis, slack ordering of paths can be changed due to path re-calculation. The most critical path before re-calculation may not be the most critical path after re-calculation. This is the reason why exhaustive path-based analysis is performed, which aims at reporting the true worst path after PBA is performed by examining all the violating paths in the design.

You can perform exhaustive reporting by using the `report_timing -retime_mode exhaustive` parameter.

To ensure optimal path search runtime, you can use the following setting to limit the worst paths:

```
set_global timing pba_exhaustive_path_nworst_limit 'L' (default: 10K)
```

Tempus performs the steps given below, when you use the following command:

```
report_timing -retime <retime type> -retime_mode exhaustive -max_paths 'N' -nworst 'M' -max_slack 0
```

1. Collect top ‘N’ GBA violating endpoints (max slack less than zero).
2. Perform PBA on top ‘L’ nworst paths for each endpoint in the above step. Whenever the Lth path on an endpoint is a PBA candidate (GBA violating), Tempus will issue a warning indicating the lack of full coverage on that end point.
3. Report top ‘N’ paths from step 2 above with PBA\_Slack less than 0 - sorted by PBA slack with maximum of ‘M’ paths per endpoint are picked.

#### **4.4.5 Path-Based Analysis Essence**

The following sections describe the impact of PBA when used with SI and AOCV.

## AOCV with PBA

The stage count in AOCV (advanced on-chip variation) is calculated differently in GBA and PBA, and hence AOCV derates differ in GBA and PBA mode. Further, the `timing_aocv_analysis_mode` global variable control in GBA and PBA differs based on the selected mode. The two methods of AOCV analysis with PBA are:

`report_timing -retime aocv`

and

`report_timing -retime aocv_path_slew_propagation`

For more information on AOCV, refer to the [Timing Analysis Modes](#) chapter.

## Signal Integrity with PBA

PBA has significant impact when performing noise analysis. When PBA is enabled, the actual timing window of the victim is computed. The attacker timing window is always taken from GBA. In this way pessimism in SI can be reduced.

Below is sample report which shows change in `incr_delay` (shown under `retime_incr_delay` column) in PBA. For more information, refer to the [Signal Integrity Delay and Glitch Analysis](#).

```
Path 1: VIOLATED Setup Check with Pin u14/CK
Endpoint: u14/D (^) checked with leading edge of 'CLK_W_1'
Beginpoint: u9/Q (^) triggered by leading edge of 'CLK_W_1'
Path Groups: {CLK_W_1}
Retime Analysis {`Path-Slew SI WP EWM }
Other End Arrival Time 0.849
- Setup 0.093
+ Phase Shift 20.000
+ CPPR Adjustment 0.001
- Uncertainty 1000.000
= Required Time -979.243
- Arrival Time 1.922
= Slack Time -981.170
= Slack Time(original) -981.435
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 1.059
= Beginpoint Arrival Time 1.059

Pin Cell Slew Retime Delay Retime Arrival Incr Retime
 Slew Delay Time Delay Incr Delay

u9/CK - 0.096 0.096 - - 1.059 - 0.000
u9/Q -> DFF 0.380 0.380 0.318 0.318 1.377 0.000 0.000
u10/A BUF 0.383 0.383 0.008 0.008 1.385 0.000 0.000
u10/Y BUF 0.143 0.143 0.108 0.108 1.493 0.000 0.000
u11/A INV 0.143 0.143 0.005 0.005 1.497 0.000 0.000
u11/Y INV 0.212 0.212 0.090 0.090 1.587 0.000 0.000
u12/A INV 0.336 0.336 0.433 0.168 1.755 0.336 0.071
u12/Y INV 0.092 0.092 0.052 0.052 1.807 0.000 0.000
u13/A BUF 0.092 0.092 0.009 0.009 1.816 0.008 0.008
u13/Y BUF 0.136 0.136 0.104 0.104 1.920 0.000 0.000
```

## **Tempus User Guide**

### Analysis and Reporting

---

u14/D -> DFF 0.136 0.136 0.001 0.001 1.922 0.000 0.000

---

## 4.5 Analysis Of Simultaneous Switching Inputs - SSI

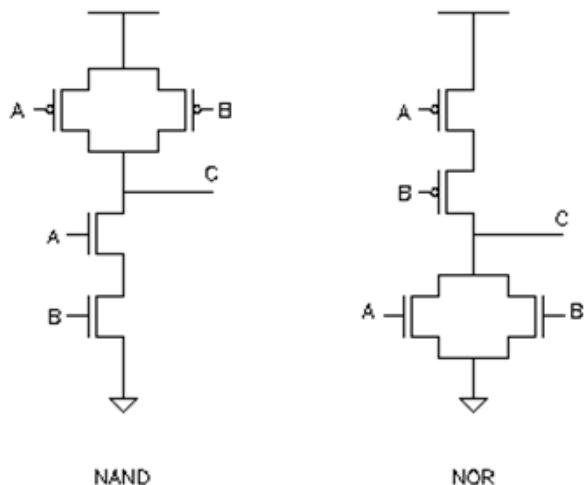
- 4.5.1 [Overview](#) on page 190
- 4.5.2 [Implementation](#) on page 191
- 4.5.3 [Reporting](#) on page 193
- 4.5.4 [Considerations for Path-Based Analysis \(PBA\)](#) on page 194

## 4.5.1 Overview

Multiple input cells that have an underlying transistor topology with parallel P or N device pulling the output up or down respectively, can exhibit accelerated delays when multiple inputs are switching in the same direction. If a secondary input switches within a specified delay window with respect to the primary transition, the delay from the initial transition will be smaller.

The example below shows a NAND gate implementation with parallel PMOS pull-up transistors, and a NOR gate with parallel NMOS pull-down devices:

**Figure 4-73 NAND gate implementation**



Simultaneous falling transitions at the NAND gate's A and B inputs will accelerate the rising output transition. Conversely, rising transitions of the NOR gate's A and B inputs within a specified proximity window will accelerate the output falling delay.

Current characterization approaches typically consider a single-input switching. In real-world scenarios where multiple inputs of a cell can be switching at the same time, the delay of a particular arc can be substantially less.

Faster delays on early paths increase pessimism. Guard-banding approaches can be used to derate the early paths – but doing this globally can lead to overall increased pessimism over the whole design.

With SSI enabled, simultaneous switching effects can be detected by the software and specific SSI derating factors provided by the user can be applied in those cases. When these timing-guided derating assertions are applied, the overall guard-band derating can be reduced – resulting in an overall less pessimistic analysis.

## 4.5.2 Implementation

The implementation of the simultaneous switching input (SSI) derating functionality involves additional syntax provided to the `set_timing_derate` constraint, library properties for defining the switching sensitivity delay window, and a set of heuristics of determining which input phase arrival times should be considered as interacting, and therefore, subject to possible SSI derating.

### SSI Derating Command Syntax

The SSI derating is controlled by the `set_timing_derate -input_switching` parameter. You can use this parameter to configure SSI derating on specific library cells when they are analyzed on early data paths, typically, the data paths for Hold analysis. For example,

```
set_timing_derate 0.5 -fall -data -early -input_switching [get_lib_cells NOR2X]
```

By using additional `set_timing_derate` options, you can configure the derate behavior for different delay corners or operating voltage domains:

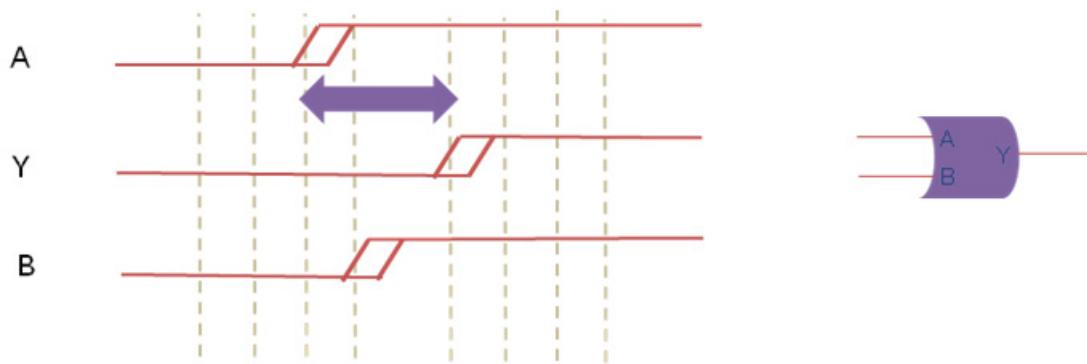
```
set_timing_derate 0.5 -fall -data -early
-delay_corner tt_800 -power_domain pdDefault
-input_switching [get_lib_cells NOR2*]
```

### Controlling the SSI Proximity Window

In order for a secondary signal transition to affect the delay from the initial input switching, it must occur within a certain proximity of the first input's transition. By default, the proximity window is equivalent to the early input-to-output delay from the first transitioning input pin.

In the diagram below, pin A is the first input to switch. For a transition at B to affect the delay from A to Y, it must transition after A has transitioned, but before Y has completed its transition to the new state.

**Figure 4-74 SSI Proximity Window**



The switching proximity window can be configured by utilizing library-level user-defined properties.

The `k_input_switching_window_rise` and `k_input_switching_window_fall` properties need to be first defined using the following syntax:

```
define_property -type float -object_type lib k_input_switching_window_rise
define_property -type float -object_type lib k_input_switching_window_fall
```

You can then set the proximity factor on a per library basis:

```
set_property [get_libs slow] k_input_switching_window_rise 1.1
set_property [get_libs slow] k_input_switching_window_fall 1.1
```

The property definitions and settings are preserved persistently in the SDC file if the design is saved.

### Heuristics for Selecting Interacting Clock Phases

Based on the clocking relationships of the signals arriving on the input pins of a cell configured for SSI, the software will determine if the SSI derate should always be applied, never be applied, or applied based on the evaluation of the proximity window. Conservative analysis requires applying the SSI derate, unless it can be shown that the inputs are not interacting. The following table summarizes the situations where SSI should be applied or not applied:

| Condition                                                                      | SSI Behavior                  | Notes |
|--------------------------------------------------------------------------------|-------------------------------|-------|
| Data are derived from synchronous clock phases of the same frequency and phase | Evaluate SSI switching window |       |

|                                                                                                     |                                                                                                                                                                       |                                                          |
|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| Data are derived from synchronous clock phases of different frequency or different phase            | Apply SSI derate always                                                                                                                                               |                                                          |
| Data are derived from physically-exclusive clocks                                                   | Do not evaluate SSI                                                                                                                                                   |                                                          |
| Data are derived from logically-exclusive clocks                                                    | Evaluate switching window if clocks are same period, phase, and are synchronous                                                                                       |                                                          |
| Data are derived from asynchronous clocks                                                           | Apply SSI derate always                                                                                                                                               | Applied if an asynchronous clock is present on any input |
| Data arrival on same pin from multiple clocks that are synchronous, have the same period, and phase | Evaluate switching window based on worst-casing of arrival times                                                                                                      |                                                          |
| Data arrival from two physically exclusive clocks                                                   | Checks for each clock are done separately between the inputs to see if a non-exclusive relationship exists between clocks of one input pin versus clocks of the other |                                                          |

#### 4.5.3 Reporting

All the deratings applied via the `set_timing_derate` command are reflected in the “User Derate” column of the detailed `report_timing` report output.

In the example below, a nominal derate on early data paths is set as 0.90. An additional simultaneous switching derate of 0.50 is also specified. The User Derate column will show a cumulative derate of 0.45 if the simultaneous input switching condition has been met.

```
set_timing_derate 0.90 -early -data -fall
set_timing_derate 0.50 -early -data -fall \
 -input_switching [get_lib_cells NOR2X1]
```

| Timing Point | Edge | Cell   | Load  | Slew  | User Derate | Delay | Arrival Time |
|--------------|------|--------|-------|-------|-------------|-------|--------------|
| in_test      | ^    |        |       | 0.004 |             |       | 4.000        |
| in_test      |      | (net)  | 0.023 |       |             |       |              |
| U_W/A ->     | ^    | NOR2X1 |       | 0.004 | 1.000       | 0.000 | 4.000        |
| U_W/Y        | v    | NOR2X1 |       | 0.066 | 0.450       | 0.450 | 4.450        |
| out_win_2    |      | (net)  | 0.006 |       |             |       |              |
| U1/A         | v    | BUFX1  |       | 0.066 | 1.000       | 0.000 | 4.450        |
| U1/Y         | v    | BUFX1  |       | 0.090 | 1.000       | 0.157 | 4.607        |
| out_win_1    |      | (net)  | 0.006 |       |             |       |              |
| U2/A         | v    | BUFX1  |       | 0.090 | 1.000       | 0.000 | 4.607        |
| U2/Y         | v    | BUFX1  |       | 0.076 | 1.000       | 0.154 | 4.761        |
| out_win      |      | (net)  | 0.004 |       |             |       |              |
| out_win      | v    |        |       | 0.076 | 1.000       | 0.000 | 4.761        |

In the timing report shown above, the delay from U\_W/A to U\_W/Y is subject to SSI derating; it received the full, cumulative derate of 0.45.

#### 4.5.4 Considerations for Path-Based Analysis (PBA)

In path-based analysis (PBA), a timing path that is found by graph-based analysis (GBA), is re-timed for increased accuracy. Re-timing can remove pessimism due to slew merging effects, AOCV stage counts, and so on. During re-timing, only the specific pin-to-pin path of interest gets a new, focused analysis. If the timing path to one pin of an SSI sensitive gate is retimed, its arrival time (for an early data path) may come later than that in GBA analysis. The arrival times of other inputs, which are still based on GBA timing can result in SSI derates being applied in PBA mode where they are not in GBA, or vice-versa. To preserve GBA bounding of PBA results, and to ensure conservative analysis, SSI derating will be applied to a given instance based on the GBA handling. If SSI derates are applied in GBA mode, then they will also be applied in PBA.

## 4.6 Waveform Aware and Rail Swing Checks

- 4.5.1 [Overview](#) on page 196
- 4.5.2 [Rail Swing Checks - An Overview](#) on page 196
- 4.5.3 [Rail Swing Reporting](#) on page 197
- 4.5.4 [Waveform Aware Pulse Width Checks - An Overview](#) on page 197
- 4.5.5 [Waveform Aware Pulse Width Checks Reporting](#) on page 198

## 4.6.1 Overview

In high-frequency designs, clock pulses may get distorted or die down during propagation through the clock tree network. This may be due to slow slews or long waveform tails causing rise/fall waveforms to not reach the VDD/VSS levels before the opposing transition arrives.

The minimum pulse width checks may not capture such failures, as these checks work based on the arrivals at delay threshold, so the waveform shapes are not considered.

The Tempus software has a capability to perform the following checks to identify and report such potential clock distortions:

- Rail Swing Checks
- Waveform aware Pulse Width Checks

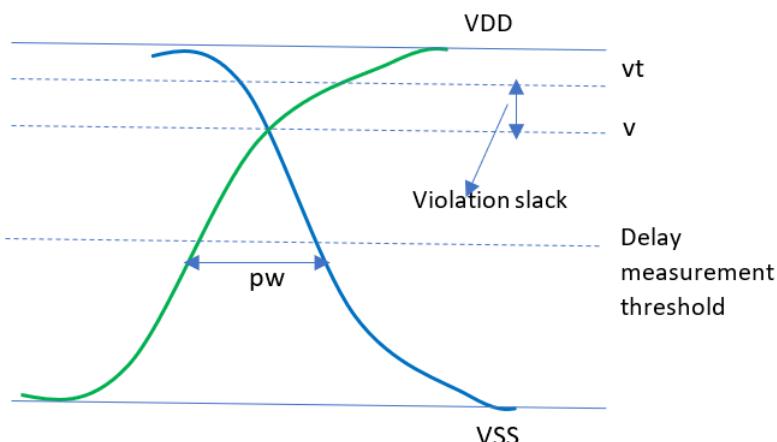
These are described below.

## 4.6.2 Rail Swing Checks - An Overview

The Tempus software provides detailed transition waveforms at each pin, which can be used for performing rail swing checks. This data allows the software to report violations on all the clock network pins, where the transition does not reach the ~VDD/VSS levels.

The pulse at any pin is formed by aligning the actual rise and fall transition waveforms over the pulse width (pw), that is computed using regular minimum pulse width checks. The intersection of these rise and fall transition waveforms defines the rail voltage achieved (v). The achieved rail voltage (v) is then checked against the user-defined threshold (vt) to check the violations.

This is illustrated in the diagram below.



## Use Model

The following global variables can be used to specify the voltage thresholds as a ratio of the VDD:

- timing\_rail\_swing\_checks\_high\_voltage\_threshold
- timing\_rail\_swing\_checks\_low\_voltage\_threshold

For example,

```
set timing_rail_swing_checks_high_voltage_threshold 0.95
set timing_rail_swing_checks_low_voltage_threshold 0.05
```

### 4.6.3 Rail Swing Reporting

The rail swing violations can be reported using the following command:

```
report constraint -check_type rail_swing -all_violators
```

The following example shows a sample output report, where all the values are specified as ratio of [VDD-VSS] supply voltages:

---

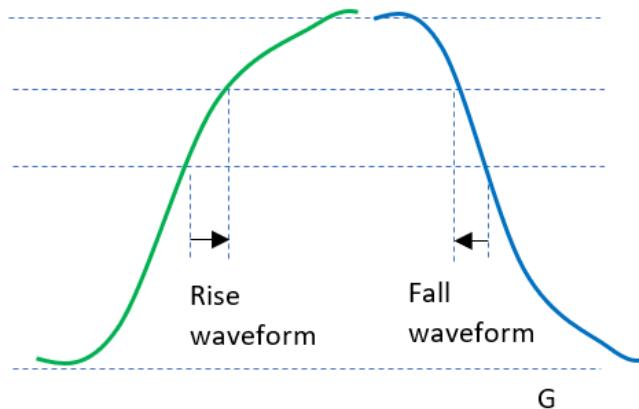
| RAIL SWING VIOLATIONS |                          |                        |        |       |       |
|-----------------------|--------------------------|------------------------|--------|-------|-------|
| Pin                   | Required<br>Signal Level | Actual<br>Signal Level | Slack  | Clock | View  |
|                       |                          |                        |        |       | Name  |
| inst8/A (low)         | 0.05                     | 0.072                  | -0.022 | CLK1  | view1 |
| f4/CP (low)           | 0.05                     | 0.065                  | -0.015 | CLK2  | view1 |
| inst1/A (low)         | 0.05                     | 0.056                  | -0.006 | CLK2  | view2 |
| ff2/CP (high)         | 0.95                     | 0.938                  | -0.012 | CLK1  | view2 |
| inst1/A (high)        | 0.95                     | 0.946                  | -0.004 | CLK1  | view1 |

---

### 4.6.4 Waveform Aware Pulse Width Checks - An Overview

Traditional pulse width checks only check the pulse widths at delay measurement thresholds and therefore do not provide any information on the waveform shape. To control waveform shape degradation on clock networks, the Tempus software provides waveform aware pulse

width checks to report minimum pulse width violations on user-specified voltage levels, and not just delay measurement thresholds. These checks can be applied on clock network pins.



## Use Model

You can use the following global variables to specify voltage levels for waveform aware pulse width checks:

- timing\_waveform\_aware\_pulse\_width\_checks\_high\_voltage\_level
- timing\_waveform\_aware\_pulse\_width\_checks\_low\_voltage\_level

For example,

```
set timing_waveform_aware_pulse_width_checks_high_voltage_level 0.75
set timing_waveform_aware_pulse_width_checks_low_voltage_level 0.25
```

You can use the following command to specify the required pulse width at a user-defined voltage level:

```
set_min_pulse_width -waveform_aware_type {absolute | source_width_ratio}
```

For example,

```
set_min_pulse_width 0.1 [get_clocks clk1] -waveform_aware_type absolute
```

To reset these settings, you can use the reset\_min\_pulse\_width command.

### 4.6.5 Waveform Aware Pulse Width Checks Reporting

To report waveform-aware pulse width checks, you can use the following command:

```
report_min_pulse_width -waveform_aware_type {regular | waveform_aware | both}
```

By default, the regular pulse width checks are reported.

## Tempus User Guide

### Analysis and Reporting

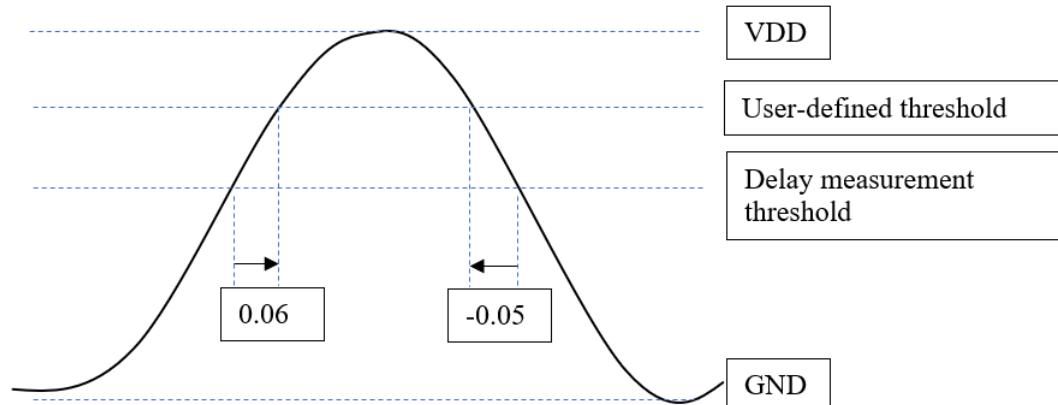
The following command generates a report as shown below:

```
report_min_pulse_width -waveform_aware_type waveform_aware -pins FF1/CP -verbose
```

```

Path 1: MET PulseWidth Check with Pin FF1/CP
Ending Clock Edge: FF1/CP (v) checked with trailing edge of 'clk'
Beginning Clock Edge: FF1/CP (^) triggered by leading edge of 'clk'
Other End Arrival Time 1.485000
+ Other End Arrival Time Adjustment -0.05
- Arrival Time Adjustment 0.06
- Pulse Width 0.047190
+ Phase Shift 0.000000
+ CPPR Adjustment 0.032300
= Required Time 1.420110
- Arrival Time 1.013900
= Slack Time 0.346210
```

The "Arrival Time Adjustment" and "Other End Arrival Time Adjustment" details in the above report denote the offsets measured over the actual launching transition waveforms and capturing transition waveforms respectively, as shown below:



These offsets are the time differences between the delay measurement threshold and user-specified voltage level measured over the actual transition waveforms. The traditional pulse widths are adjusted by the computed offsets to obtain a width at the user-specified voltage threshold.

## 4.7 Set Instance Library Flow

- 4.5.1 [Overview](#) on page 201
- 4.5.2 [MMMC set\\_instance\\_library Flow](#) on page 201
- 4.5.3 [MMMC Flow Script](#) on page 202
- 4.5.4 [Save and Restore MMMC set\\_instance\\_library Flow](#) on page 204

## 4.7.1 Overview

In MSV designs, if the power intent information is not available in the form of PGV, CPF, or UPF, the Tempus software does not have the mechanism to bind the correct PVT library. For this you can use the `set_instance_library` command to specify the library that you want to bind to an instance.

In case of macros where different PVT libraries are available and none of those libraries match with the PVT, the software will attempt to interpolate. You can use the `set_instance_library` command to bind it with the closest PVT library.

In MSV designs that have PGV netlist, CPF, or UPF, where power intent information is present, the `set_instance_library` command can be used on a handful of instances where power intent is not correct and can bind it correctly. This feature is useful during the initial design cycle.

## 4.7.2 MMMC set\_instance\_library Flow

Tempus requires timing libraries, netlist as input along with the other essential data required for static timing analysis. In addition to all these, the MMMC flow requires ViewDefinition template file. In this template file, you can define all the analysis views required, corresponding RC corner, constraint mode and delay corner. When you specify the `set_instance_library` command in the ViewDefinition file, it overrides the default binding mechanism for a cell from the library set.

### ViewDefinition Template

To apply the `set_instance_library` command in the MMMC flow, you need to create a file with all the `set_instance_library` commands. For this you can use the `create_delay_corner -instance_library_file` parameter.

For example,

```
create_delay_corner -instance_library_file <file1>
create_library_set -name complete_library_set \
-timing [list \
/ics/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st ao_buffer_cpf2_6_DSH_MP_CP/savedDB.dat/libs/mmmc/
C28SOI_SC_12_CORI_LL_ss28_0.90V_0.00V_m0.60V_0.60V_m40C.lib \
/ics/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st ao_buffer_cpf2_6_DSH_MP_CP/savedDB.dat/libs/mmmc/
C28SOI_SC_12_CORR_LL_ss28_0.90V_0.00V_m0.60V_0.60V_m40C.lib \
/ics/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st ao_buffer_cpf2_6_DSH_MP_CP/savedDB.dat/libs/mmmc/
C28SOI_SC_12_SHIFTFBP10_LR_ss28_0.90V_0.95V_m40C.lib \
/ics/etsproject3/mgarg/aayhis/myur/MSV_DBS/
```

## Tempus User Guide

### Analysis and Reporting

---

```
NW_PGV_paradime_st_ao_buffer_cpf2_6_DSH_MP_CP/savedDB.dat/libs/mmmc/
C32_EPM_EPOD_D12SWSG_LR_ss28_0.90V_m40C.lib \
/icd/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st_ao_buffer_cpf2_6_DSH_MP_CP/savedDB.dat/libs/mmmc/
C32_SC_12_CORE_LL_ss28_0.90V_0.00V_m0.60V_0.60V_m40C.lib \
/icd/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st_ao_buffer_cpf2_6_DSH_MP_CP/savedDB.dat/libs/mmmc/new.lib \
my.lib \
newcell.lib \]
create_op_cond -name opcond1 -library_file
/icd/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st_ao_buffer_cpf1_4_DSH_MP_CP/savedDB.dat/libs/mmmc/
C28SOI_SC_12_CORI_LL_ss28_0.90V_0.00V_m0.60V_0.60V_m40C.lib -P 1.228 -V 0.9 -T -40
create_rc_corner -name min_corner
create_rc_corner -name max_corner
create_delay_corner -name slow_max \
-pg_net_voltages {VDD2@0.9 VDD1@0.9 vPD2@-0.6 vPD1@-0.6} \
-library_set complete_library_set \
-opcond_library C28SOI_SC_12_CORI_LL \
-opcond_opcond1 \
-rc_corner max_corner \
-instance_library_file sil.tcl
create_delay_corner -name fast_min \
-pg_net_voltages {VDD2@0.9 VDD1@0.9 vPD2@-0.6 vPD1@-0.6} \
-library_set complete_library_set \
-opcond_library C28SOI_SC_12_CORI_LL \
-opcond_opcond1 \ -rc_corner min_corner
create_constraint_mode -name func_mode \
-sdc_files \
[list /icd/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st_ao_buffer_cpf2_6_DSH_MP_CPG/pgv_data/mmmc/modes/func_mode/
func_mode.sdc]
create_analysis_view -name setup_func_m40 -constraint_mode func_mode -delay_corner
slow_max -latency_file /icd/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st_ao_buffer_cpf2_6_DSH_MP_CP/pgv_data/mmmc/views/setup_func_m40/
latency.sdc
create_analysis_view -name hold_func_m40 -constraint_mode func_mode -delay_corner
fast_min -latency_file /icd/etsproject3/mgarg/aayhis/myur/MSV_DBS/
NW_PGV_paradime_st_ao_buffer_cpf2_6_DSH_MP_CP/pgv_data/mmmc/views/hold_func_m40/
latency.sdc
set_analysis_view -setup [list setup_func_m40] -hold [list hold_func_m40]
```

### 4.7.3 MMMC Flow Script

The MMMC flow sample script is shown below:

```
set_multi_cpu_usage -localCpu 16
set_timing_library_read_without_power false
read_view_definition setup_func_m40_viewDefinition.tcl
read_verilog pg-explicit.v
set_top_module top
```

## Tempus User Guide

### Analysis and Reporting

---

```
report_instance_library -inst { iP/dff1 iP/and1 } > ril_test.rpt
```

The library that is used for binding instances with the `set_instance_library` command must be an active library loaded in the session via active library set defined with the `create_library_set` command.

The following command is used in the sil.tcl:

```
set_instance_library -library newcell.lib -instance {iP/and1 iP/dff1}
```

You can use the following command to verify whether the instance of the cell is bound with the library specified with the `set_instance_library` command:

```
report_instance_library -inst {iP/and1 iP/dff1} > ril_test.rpt
```

The output of the `report_instance_library` command is given below:

```
Instance : iP/dff1
Cell : C12T32_LL_SDFPQX8
Analysis View : setup_Func_m40
Power Domain : Default
Library/Libset : new3cell_C32_SC_12_CORE_LL/complete_library_set
Library File : /it/sjclapa06p5_scratch01/aayhis/expire_2021_03_30/
scratch_dec/SIL_DOC/sil_view/doc_case/newcell.lib
User Setting : True
Op Cond : P->1.228000, V->0.900000, T->-40.000000 (new3cell_C32_SC_12_CORE_LL/
%NOM_PVT)
Cell Type : gateCell
Instance : iP/and1
Cell : C12T32_LL_AND2X8
Analysis View : setup_Func_m40
Power Domain : Default
Library/Libset : new3cell_C32_SC_12_CORE_LL/complete_library_set
Library File : /it/sjclapa06p5_scratch01/aayhis/expire_2021_03_30/
scratch_dec/SIL_DOC/sil_view/doc_case/newcell.lib
User Setting : True
Op Cond : P->1.228000, V->0.900000, T->-40.000000
(new3cell_C32_SC_12_CORE_LL/%NOM_PVT)
Cell Type : gateCell
```

The above output clearly shows the following:

1. The library binding was applied for the instance `iP/and1` and `iP/dff1` with the help of the `set_instance_library` command and the library file reported in the `report_instance_library` output is the same as that applied.
2. User setting: `True` indicates that the `set_instance_library` command has been applied.

#### 4.7.4 Save and Restore MMMC set\_instance\_library Flow

In the save and restore flow, the `set_instance_library` command used with the `create_delay_corner` command in the `viewDefinition` file will be saved in the save session directory.

##### Save Session Script

The save session sample script is shown below:

```
set_multi_cpu_usage -localCpu 16
set timing_library_read_without_power false
read_view_definition setup_func_m40_viewDefinition.tcl
read_verilog pg-explicit.v
set_top_module top
update_timing -full
save_design new_test.db
```

The `set_instance_library` command used in the above script is saved in the following format:

```
set_instance_library \
 -library \
 " ${::IMEX::sillibVar}/mmmc/newcell.lib" \
 -instance \
 " ip/and1 \
 ip/dff1 "
```

##### Restore Session Script

The restore session sample script is shown below:

```
source new_test.db
report_instance_library -inst { ip/and1 ip/dff1 } > ril_restore1_test.rpt
```

Once the design has been restored from the saved session, the library binding that was applied for the instance, using the `set_instance_library` command, in the saved session script will be the same as in the restored session.

The output of the `report_instance_library` command after the restore session is shown below:

|               |   |                  |
|---------------|---|------------------|
| Instance      | : | ip/and1          |
| Cell          | : | C12T32_LL_AND2X8 |
| Analysis View | : | setup_func_m40   |
| Power Domain  | : | Default          |

## Tempus User Guide

### Analysis and Reporting

---

```
Library/Libset : new3cell_C32_SC_12_CORE_LL/complete_library_set
Library File : /it/sjclapa06p5_scratch01/aayhis/expire_2021_03_30/
scratch_dec/SIL_DOC/sil_view/doc_case/newcell.lib
User Setting : True
Op Cond : P->1.228000, V->0.900000, T->-40.000000
(new3cell_C32_SC_12_CORE_LL/%NOM_PVT)
Cell Type : gateCell
Instance : iP/dff1
Cell : C12T32_LL_SDFPQX8
Analysis View : setup_func_m40
Power Domain : Default
Library/Libset : new3cell_C32_SC_12_CORE_LL/complete_library_set
Library File : /it/sjclapa06p5_scratch01/aayhis/expire_2021_03_30/
scratch_dec/SIL_DOC/sil_view/doc_case/newcell.lib
User Setting : True
Op Cond : P->1.228000, V->0.900000, T->-40.000000
(new3cell_C32_SC_12_CORE_LL/%NOM_PVT)
Cell Type : gateCell
```

# Concurrent Multi-Mode Multi-Corner Timing Analysis

---

- 5.1 [Overview](#) on page 2
- 5.2 [Multi-Mode Multi-Corner Licensing](#) on page 2
- 5.3 [Configuring Setup for Multi-Mode Multi-Corner Analysis](#) on page 2
- 5.4 [Library Sets](#) on page 3
- 5.5 [Virtual Operating Conditions](#) on page 5
- 5.6 [RC Corner Objects](#) on page 7
- 5.7 [Delay Corner Objects](#) on page 7
- 5.8 [Power Domain Definitions](#) on page 9
- 5.9 [Constraint Mode Objects](#) on page 11
- 5.10 [Constraint Support in Multi-Mode and MMMC Analysis](#) on page 13
- 5.11 [Analysis Views](#) on page 14
- 5.12 [Saving Multi-Mode Multi-Corner Configuration](#) on page 19
- 5.13 [Configuring Setup for Multi-Mode Multi-Corner Analysis](#) on page 2
- 5.14 [Performing Timing Analysis](#) on page 19
- 5.15 [Generating Timing Reports](#) on page 20

## 5.1 Overview

As feature sizes decrease, it becomes difficult to determine the worst-case combinations of device corner (timing libraries and PVT operating conditions), RC corners, and constraint modes at which the designers need to validate timing requirements for a design. Multi-mode multi-corner analysis and optimization provides the ability to configure the software to support multiple combinations of modes and corners, and to evaluate them concurrently.

Multi-mode multi-corner analysis uses a tiered approach for defining the required data. It allows top-level definitions (analysis views) that share common information to be specified by referencing the same lower-level objects. The active analysis views defined in the software represent the different design variations that will be analyzed.

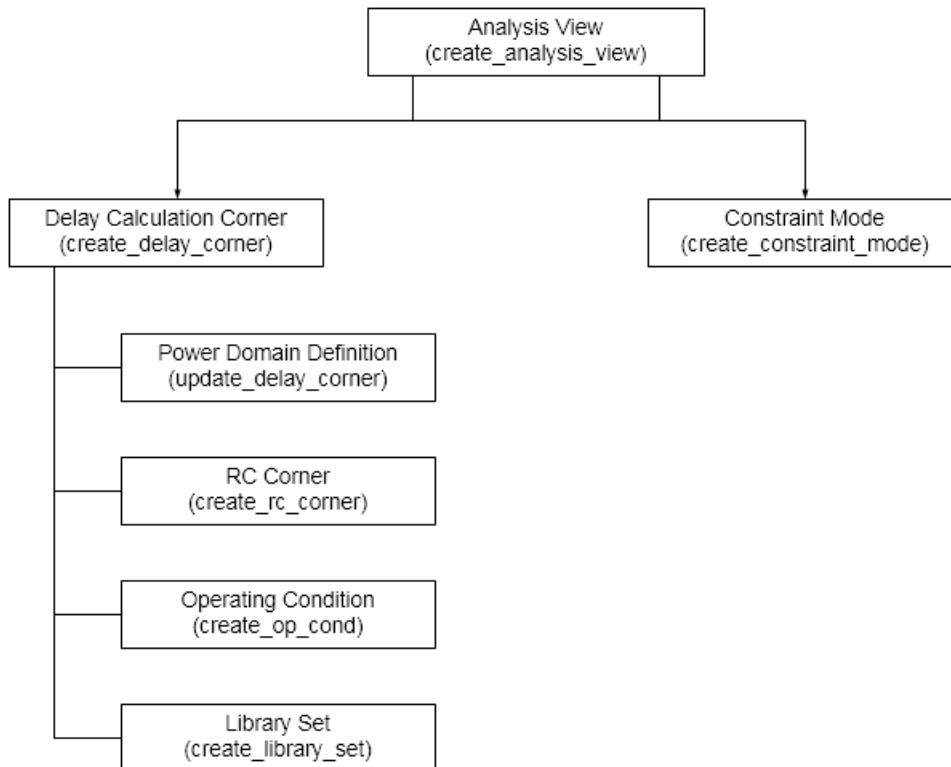
## 5.2 Multi-Mode Multi-Corner Licensing

Multi-corner analysis requires an XL license. Multi-mode analysis does not require a XL license, and is limited to only one delay corner for setup analysis, and one delay corner for hold analysis.

## 5.3 Configuring Setup for Multi-Mode Multi-Corner Analysis

Multi-mode multi-corner analysis uses a tiered approach to assemble the information necessary for timing analysis and optimization. Each top-level definition (called an analysis view) is composed of a delay corner and a constraint mode. The active analysis views defined in the software represent the different design variations that will be analyzed.

**Figure 5-1 Hierarchical Analysis View Configuration**



## 5.4 Library Sets

This section includes:

- [Creating Library Sets](#)
- [Updating Library Sets](#)

### Creating Library Sets

Complex designs might require the specification of multiple library files to define all of the standard cells, memory devices, pads, and so forth, included in the design. Different library sets can be defined to provide uniquely characterized libraries for each delay corner or power domain.

Library sets allow a group of library files to be treated as a single entity so that higher-level descriptions (delay corners) can simply refer to the library configuration by name. A library set consists of timing libraries and can also include cdB/UDN models, AOCV, SOCV libraries which are supported through various options of [create\\_library\\_set](#) command.

## Tempus User Guide

### Concurrent Multi-Mode Multi-Corner Timing Analysis

For details on UDN syntax, refer to the *cDB Noise Library Format* section in the *Noise Library Characterization* chapter of the *make\_cDB Noise Characterizer User Guide*.

For more information on UDN model, refer to the *User-Defined Noise Model - An Overview* section in the *Noise Library Characterization* chapter of the *make\_cDB Noise Characterizer User Guide*.

The same library set can be referenced multiple times by different delay corners.

**Note:** You can create as many library sets as needed to support for delay corners that will be included in the multi-mode multi-corner analysis.

To create a library set, use the following command:

```
create_library_set
```

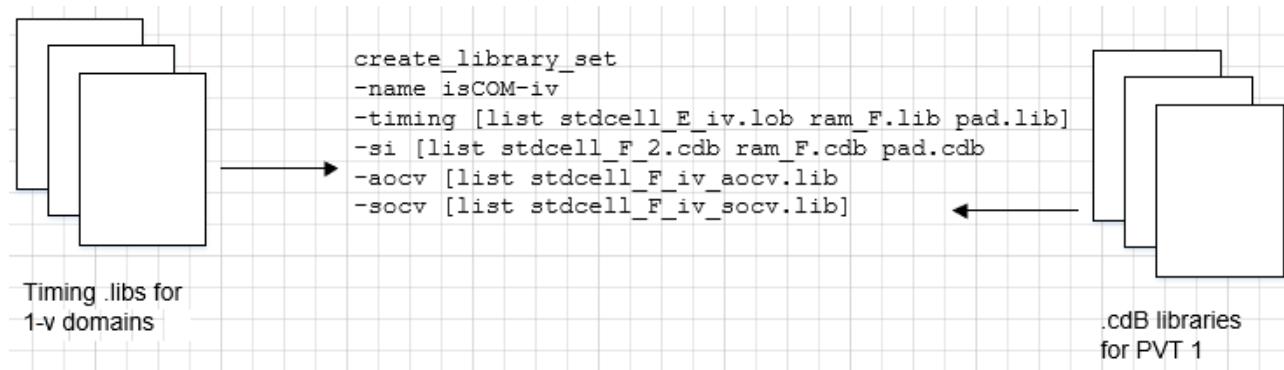
### **Flow Library Requirements**

Timing libraries must be read into the system first, using `read_lib` command. If you want to create a library set that includes cDB libraries, they also must be read into the system first, using the `read_lib -cdb` command. This same flow is adopted in single-mode single-corner (SMSC) analysis.

The order in which you define timing libraries is important. The software considers the first library in the list as master library, with successive libraries having low priority.

The library sets can be created only after the top module is set. The following figure shows the creation of a library set that associates timing libraries and cDB libraries with a nominal voltage of 1 volt with the library name is COM-1V.

**Figure 5-2 Creation of Library Sets**



\* Either AOCV or SOCV libraries can be specified at a time. The AOCV and SOCV files can also be added to the library set by either `-aocv [list aocv1.lib aocv2.lib]` or `-socv [list socv1.lib socv2.lib]` in the create library set settings.

**Note:** You can use the `create_library_set -timing` parameter and interpolate a set of timing libraries to perform IR-drop aware delay calculation.

## Updating Library Sets

To change the timing and cdB library files for an existing library set, use the following command:

- update\_library\_set

You can update a library set in the GUI mode by using the following ways:

Click the *File* tab and select Configure MMMC. A new window opens, double-click on one of the library set objects, where the timing library files need to be updated. A new terminal will open that allows you to add or delete timing library files. After updating the files, click on **Apply** and **OK** to save the changes.

Or

Click the *Timing & SI* tab and select MMMC Browser. Select the library set and double-click on one of the library sets where the timing library files need to be updated. A new terminal will open where you can add or delete timing library files. After updating the files, click on **Apply** and **OK** to save the changes.

### Note:

- It may not be necessary to update a library set, but Timing/SI libraries under a library set can be updated.
- You can use the `update_library_set` command or the *Edit Library Set* option (in GUI Mode), before (or any time later) the multi-mode multi-corner views are loaded into the design. Once the views are loaded, any change in the library set (using the `update_library_set` command (or *Edit Library Set* form)) will reset the timing, RC data, and delay corner settings for all the analysis views.

## 5.5 Virtual Operating Conditions

This section includes:

- Creating Virtual Operating Conditions

■ Editing Virtual Operating Conditions

### **Creating Virtual Operating Conditions**

In general, the process, voltage, and temperature (PVT) point is specified by referring to a predefined operating condition defined in a timing library. The library operating condition provides the system with values for P, V, and T. However, there are situations when there are no predefined operating conditions in user timing libraries, or pre-existing operating conditions are not consistent with user's operating environment. Instead of modifying timing libraries to add or adjust operating condition definitions, you can create a set of virtual operating conditions for a library, to define a PVT operating point. These virtual operating conditions can then be referenced by a delay corner, as if they actually existed in the library.

The operating conditions scale the delay numbers if there is a mismatch between the PVT condition of libraries and the virtual operating conditions. To create a virtual operating condition for a library, use the following command:

■ create op cond

For example, the following command creates a virtual operating condition called PVT1 for the library IsCOM-1V:

```
create_op_cond -name PVT1
 -library_file IsCOM-1V.lib
 -P 1.0
 -V 1.2
 -T 120
```

### **Editing Virtual Operating Conditions**

In GUI mode, you can add or change attributes for a defined virtual operating condition using the “Add OP Cond” form.

Click the *File* tab and choose *Configure MMMC*. A new window opens in which you need to double-click on the *OP Conds* tab and add the operating condition. Click **Apply** and **OK** for the changes to take affect.

or

Click the *MMMC Browser* under the *Timing & SI* tab, and the double-click on the OP Conds tab. After adding the operating condition, click **Apply** and **OK** for the changes to take affect.

You can edit a virtual operating condition before multi-mode multi-corner view definitions are loaded into the design or any time later. However, once the software is in multi-mode multi-

corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## 5.6 RC Corner Objects

This section includes:

- [Creating RC Corner Objects](#)

### **Creating RC Corner Objects**

An RC corner object provides the information necessary to properly annotate, and use the RCs for delay calculation. In Tempus, you read the RC information using the SPEF files. A SPEF file contains the parasitic information for a RC corner. In multi-mode multi-corner analysis, you can use a SPEF file for each view.

RC corner objects are referenced when creating delay calculation corner objects.

To create an RC corner, use the following command:

- [create\\_rc\\_corner](#)

For example, the following command creates an RC corner called `rc-typ`:

```
create_rc_corner -name rc-typ
```

## 5.7 Delay Corner Objects

This section includes:

- [Creating Delay Corner Objects](#)
- [Editing Delay Corner Objects](#)
- [Power Domain Definitions](#)

### **Creating Delay Corner Objects**

A delay corner provides information necessary to control delay calculation for a specific analysis view. Each corner contains information on the libraries, the operating conditions with which the libraries should be accessed, and the RC corner for loading the parasitic data. Delay corner objects can be shared by multiple top-level analysis views.

## Tempus User Guide

### Concurrent Multi-Mode Multi-Corner Timing Analysis

To create a delay calculation corner, use the following command:

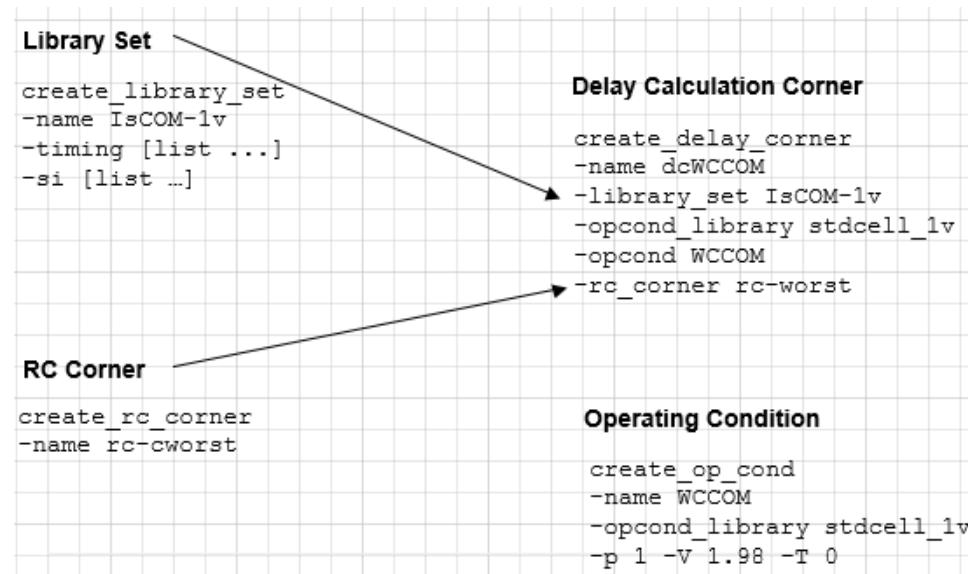
- create\_delay\_corner

Use separate delay calculation corners to define major PVT operating points (for example, Best-Case and Worst-Case).

Use the `-early_*` and `-late_*` parameters within a single delay calculation corner to control on-chip variation.

The following figure shows the creation of a delay calculation corner called `dcWCCOM`. This corner uses the libraries from `IsCOM-1v`, sets the operating condition to `WCCOM`, as defined in the `stdcell_1v` timing library, and uses the `rc-cworst` RC corner:

**Figure 5-3 Creation of a delay calculation corner**



### Editing Delay Corner Objects

To add or change attribute values of an existing delay calculation corner object, use the following command:

- update\_delay\_corner

In GUI mode, you can make changes to a delay calculation corner object by using *Add Delay Corner* form.

## Tempus User Guide

### Concurrent Multi-Mode Multi-Corner Timing Analysis

---

Click the *File* tab and choose *Configure MMMC*. A new window opens, double-click on the *Delay Corners* tab. Select the type of analysis mode and specify the rc corner, library set, OpCond Lib, OpCond, IrDrop File. Once done, click **Apply** and **OK** to save the settings.

Or,

Click *Timing & SI* tab and choose *MMMC Browser*. A new window opens, double-click on the *Delay Corners* tab. Select the type of analysis mode and specify the rc corner, Library set, OpCond Lib, OpCond, IrDrop File. Once done, click **Apply** and **OK** to save the settings.

You can use the `update_delay_corner` command or the “Add Delay Corner” form before (or any time later) multi-mode multi-corner view definitions are loaded into the design. However, once the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

The following example shows how to create two corners and update the delay corners with the appropriate information.

Create worst-case RC corner based on the worst-case RC parameters:

```
create_rc_corner -name rc_cworst
```

Best-case RC corner based on the best-case RC parameters:

```
create_rc_corner -name rc_cbest
```

To associate the best-case and worst-case RC corners with the appropriate delay corner, use the `update_delay_corner` command:

```
update_delay_corner -name AV_HL_FUNC_MIN_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_HL_FUNC_MIN_RC2_dc -rc_corner rc_cbest
update_delay_corner -name AV_HL_FUNC_MAX_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_HL_FUNC_MAX_RC2_dc -rc_corner rc_cbest
update_delay_corner -name AV_HL_SCAN_MIN_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_HL_SCAN_MAX_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_HO_FUNC_MIN_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_HO_FUNC_MAX_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_LO_FUNC_MIN_RC1_dc -rc_corner rc_cworst
update_delay_corner -name AV_LO_FUNC_MAX_RC1_dc -rc_corner rc_cworst
```

## 5.8 Power Domain Definitions

### Adding Power Domain Definition to Delay Calculation Corners

A single delay calculation corner object specifies the delay calculation rules for the entire design. If a design includes power domains, the delay calculation corner can contain domain-specific subsections that specify the required operating condition information, and any necessary timing library rebinding for the power domain.

## Tempus User Guide

### Concurrent Multi-Mode Multi-Corner Timing Analysis

---

You must use the same power domain names that you read in using the power domain file to define the physical aspects of the domain. For more information about power domain file, see [read\\_power\\_domain](#).

To create a power domain definition for a delay calculation corner, use the following command:

- [update\\_delay\\_corner](#)

For example, the following command adds definitions for the power domain `domain-3V` to the dcWCCOM delay calculation corner:

```
update_delay_corner -name dcWCCOM
-power_domain domain-3V
-library_set libs-3volt
-opcond_library delayvolt_3V
-opcond_slow_3V
```

### Editing Power Domain Definitions

To add or change attribute values for an existing power domain definition, use the following command:

- [update\\_delay\\_corner](#)

In GUI mode, you can make changes to a power domain definition using the *Edit Power Domain* form:

Click the *File* tab and choose *Configure MMMC*. A new window opens, double-click on the *Delay Corners* tab. In the “*Power Domain List*” column, click **Add**. A new *Add Power Domain* window opens, you can enter the details for library set, OpCond Lib, OpCond, IrDrop File. Once done, click **Apply** and **OK** to save the settings.

or:

Click *Timing & SI* tab and choose *MMMC Browser*. A new window opens, double-click on the *Delay Corners* tab. In the “*Power Domain List*” column, click **Add**. A new “*Add Power Domain*” window opens, you can enter the details for library set, OpCond Lib, OpCond, IrDrop File. Once done, click **Apply** and **OK** to save the settings.

You can use the `update_delay_corner` command or the *Add Delay Corner* form before (or any time later) multi-mode multi-corner view definitions are loaded into the design. However, after the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data being reset for all analysis views.

## 5.9 Constraint Mode Objects

This section includes:

- [Creating Constraint Mode Objects](#)
- [Editing Constraint Mode Objects](#)
- [Entering Constraints Interactively](#)
- [Constraint Support in Multi-Mode and MMMC Analysis](#)

### Creating Constraint Mode Objects

A constraint mode object defines one of possibly many different functional, test, or Dynamic Voltage and Frequency Scaling (DVFS) modes of a design. It consists of a list of SDC constraint files that contain timing analysis information, such as the clock specifications, case analysis constraints, I/O timings, and path exceptions that make each mode unique. SDC files can be shared by many different constraint modes, and the same constraint mode can be associated with multiple analysis views.

To create a constraint mode object, use the following command:

- [create\\_constraint\\_mode](#)

The following figure shows that the constraint mode “missionSetup” reads in 3 constraint mode files “io.sdc”, “mission1-clks.sdc” and “mission1-except.sdc”:

**Figure 5-4 Constraint Mode Objects**

```
create_constraint_mode
-name missionSetup
-sdc_files [list io.sdc mission1.clks.sdc mission1-except.sdc

[list io.sdc mission1-clks.sdc mission1-except.sdc]
SDC Files
```

### Editing Constraint Mode Objects

To change the SDC constraint file information for an existing constraint mode object, use the following command:

■ update constraint mode

In the GUI mode, you can make changes to a constraint mode object using the *Add Constraint Mode* form.

Click the *File* tab and choose *Configure MMMC*. A new window opens, double-click on the *Constraint Modes* tab and add the “SDC Constraint File(s)”. Once done, click **Apply** and **OK** to save the settings.

Or,

Click *Timing & SI* tab and choose *MMMC Browser*. A new window opens, double-click on the *Constraint Modes* tab and add the “SDC Constraint File(s)”. Once done, click **Apply** and **OK** to save the settings.

You can use the `update_constraint_mode` command or the *Add Constraint Mode* form before (or any time later) multi-mode multi-corner view definitions are loaded into the design. However, once the software is in multi-mode multi-corner analysis mode, any changes to an existing object results in the timing, delay calculation, and RC data are reset for all the analysis views.

## Entering Constraints Interactively

The Tempus software allows you to update assertions for a multi-mode multi-corner constraint mode object, and have those changes take immediate effect, use the following command:

■ set interactive constraint modes

Specifying `set_interactive_constraint_modes` puts the software into interactive mode for the specified constraint mode objects. Any constraints that you specify after this command will take effect immediately on all active analysis views that are associated with these constraint modes. By default, no constraint modes are considered active.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all views in the current session that are associated with the constraint mode func1:

```
set_interactive_constraint_modes func1
set_propagated_clock [all_clocks]
```

The software stays in interactive mode until you exit it by specifying the `set_interactive_constraint_modes` command with an empty list:

```
set_interactive_constraint_modes { }
```

All the new constraints are saved in the SDC file for the specified constraint mode when you save the design.

The `all_constraint_modes` command can be used to generate a list of constraint modes as the argument for this command.

For example, the following commands put the software into interactive constraint entry mode, and apply the `set_propagated_clock` assertion on all active analysis views in the current session.

```
set_interactive_constraint_modes [all_constraint_modes -active]
set_propagated_clock [all_clocks]
```

You can use the `get_interactive_constraint_modes` command to return a list of the constraint mode objects in interactive constraint entry mode.

**Note:** Interactive constraint mode only works when the software is in regular multi-mode multi-corner timing analysis mode. You cannot use it in distributed multi-mode multi-corner analysis. In min/max analysis mode, constraints are always accepted interactively.

## 5.10 Constraint Support in Multi-Mode and MMMC Analysis

The Tempus software isolates the following SDC constraints from conflicting with each other, in both multi-mode and multi-mode multi-corner analysis modes:

- `create_clock`
- `create_generated_clock`
- `set_annotated_check`
- `set_annotated_delay`
- `set_annotated_transition`
- `set_case_analysis`
- `set_clock_gating_check`
- `set_clock_groups`
- `set_clock_latency`
- `set_clock_sense`
- `set_clock_transition`
- `set_clock_uncertainty`

- `set_disable_timing`
- `set_drive`
- `set_driving_cell`
- `set_false_path`
- `set_fanout_load`
- `set_input_delay`
- `set_input_transition`
- `set_load`
- `set_max_delay`
- `set_max_time_borrow`
- `set_max_transition`
- `set_min_delay`
- `set_min_pulse_width`
- `set_multicycle_path`
- `set_output_delay`
- `set_propagated_clock`
- `set_resistance`

**Note:** Path groups defined with `group_path` are considered to be global definitions across all analysis views.

## 5.11 Analysis Views

This section includes:

- [Creating Analysis Views](#)
- [Editing Analysis View Objects](#)
- [Setting Active Analysis Views](#)
- [Guidelines for Setting Active Analysis Views](#)

## Creating Analysis Views

An analysis view object provides all of the information necessary to control a given multi-mode multi-corner analysis. It consists of a delay calculation corner and a constraint mode.

To create an analysis view, use the following command:

- create\_analysis\_view

The following figure shows the creation of the analysis view missionSetup:

**Figure 5-5 Creation of New Analysis View**

### Delay Calculation Corner

```
create_delay_corner
-name dcWCCOM
-library_set IsCOM-iv
-opcond_library stdcell_iv
-opcond WCCOM
-rc_corner rc-cworst
```

### Analysis View

```
create_analysis_view
-name missionSlow
-delay_corner dcWCCOX
-constraint_mode missionSetup
```

### Constraint Mode

```
create_constraint_mode
-name missionSetup
-sdc_files [list io.sdc ...]
```

## Editing Analysis View Objects

To change the attribute values for an existing analysis view, use the following command:

- update\_analysis\_view

In the GUI mode, you can make changes to an analysis view using the *Add Analysis View* form:

Click the *File* tab and choose *Configure MMMC*. A new window opens, double-click on the *Analysis Views* tab and specify the constraint mode, delay corner, and analysis view. Click **Apply** and **OK** to save the settings.

Or,

Click *Timing & SI* tab and choose *MMMC Browser*. A new window opens, double-click on the *Analysis Views* tab and specify the constraint mode, delay corner, and analysis view. Click **Apply** and **OK** to save the settings.

**Note:** You can use the `update_analysis_view` command or the *Add Analysis View* form before (or any time later) multi-mode multi-corner view definitions are loaded into the design. However, once the software is in multi-mode multi-corner analysis mode, any changes to an existing object in the timing, delay calculation, and RC data are reset for all the analysis views.

## Setting Active Analysis Views

After creating analysis views, you must set which views the software should use for setup and hold analysis. These "active" views represent the different design variations that will be analyzed. Active views can be changed throughout the flow to utilize different subsets of views. Libraries and data are loaded into the system, as required to support the selected set of active views.

To set active analysis views, use the following command:

- `set_analysis_view`

**Note:** You must define at least one setup and one hold analysis view.

For example, the following command sets `missionSlow` and `mission2Slow` as the active views for setup analysis, and `missionFast` and `testFast` as the active views for hold analysis:

```
set_analysis_view
 -setup {missionSlow mission2Slow}
 -hold {missionFast testFast}
```

## Guidelines for Setting Active Analysis Views

The order in which you specify views, using the `set_analysis_view` command, is important. By default, the first view defined in the `-setup` and `-hold` lists is the default view.

## Changing the Default Active Analysis View

For example, if the analysis view `missionSlow` is currently the default active setup view in the design, the following command temporarily changes the default view to `mission2Slow`:

```
set_default_view -setup mission2Slow
```

## Checking the Multi-Mode Multi-Corner Configuration

Use the following command to generate a hierarchical report of your current multi-mode multi-corner configuration:

- report analysis views

## 5.12 Design Loading and MMMC Configuration Script

The script to load a design must contain design loading as well as all MMMC configuration commands, such as, `create_library_set`, `create_delay_corner`, and `create_constraint_mode`.

A sample `common_init.tcl` script is shown below:

```
common_init.tcl
read_mmmc INPUT/viewDefinition.tcl
read_physical -lef "
 lib/lef/gsclib045.fixed.lef
 lib/lef/MEM1_256X32.lef
 lib/lef/MEM1_1024X32.lef
 lib/lef/MEM1_4096X32.lef
 lib/lef/MEM2_128X16.lef
 lib/lef/MEM2_128X32.lef
 lib/lef/MEM2_136X32.lef
 lib/lef/MEM2_512X32.lef
 lib/lef/MEM2_1024X32.lef
 lib/lef/MEM2_2048X32.lef
 lib/lef/MEM2_4096X32.lef
 lib/lef/pdkIO.lef
"
read_netlist INPUT/asic_entity.v.gz
#read_def aa.def
#read_power_intent
init_design
if {[get_db program_short_name] == "tempus"} {
 read_db -physical_data tempus_test.db
}
```

A sample `viewDefinition.tcl` script is shown below:

```
viewDefinition.tcl
create_library_set -name fast \
 -timing \
 [list ${::IMEX::libVar}/mmmc/fast.lib \
 ${::IMEX::libVar}/mmmc/pdkIO.lib \
 ${::IMEX::libVar}/mmmc/MEM2_4096X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM2_2048X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM2_1024X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM1_4096X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM1_1024X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM1_256X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM2_512X32_slow.lib \
 ${::IMEX::libVar}/mmmc/MEM2_136X32_slow.lib]
```

# Tempus User Guide

## Concurrent Multi-Mode Multi-Corner Timing Analysis

---

```
${{::IMEX::libVar}}/mmmc/MEM2_128X32_slow.lib\
${{::IMEX::libVar}}/mmmc/MEM2_128X16_slow.lib\
${{::IMEX::libVar}}/mmmc/leon.lib]
create_library_set -name slow \
-timing \
[list ${{::IMEX::libVar}}/mmmc/slow.lib \
${{::IMEX::libVar}}/mmmc/pdkIO.lib \
${{::IMEX::libVar}}/mmmc/MEM2_4096X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM2_2048X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM2_1024X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM1_4096X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM1_1024X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM1_256X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM2_512X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM2_136X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM2_128X32_slow.lib \
${{::IMEX::libVar}}/mmmc/MEM2_128X16_slow.lib \
${{::IMEX::libVar}}/mmmc/leon.lib]
create_timing_condition -name default_mapping_tc_2 \
-library_sets [list fast]
create_timing_condition -name default_mapping_tc_1 \
-library_sets [list slow]
create_rc_corner -name rc_min \
-cap_table ${{::IMEX::libVar}}/mmmc/capTable \
-pre_route_res 1.34236 \
-post_route_res 0.994125 \
-pre_route_cap 1.10066 \
-post_route_cap 0.960235 \
-post_route_cross_cap 1.22327 \
-pre_route_clock_res 0 \
-pre_route_clock_cap 1.105 \
-post_route_clock_cap 0.969117 \
-temperature 0 \
-qrc_tech ${{::IMEX::libVar}}/mmmc/qrcTechFile \
create_rc_corner -name rc_max \
-cap_table ${{::IMEX::libVar}}/mmmc/capTable \
-pre_route_res 1.34236 \
-post_route_res 0.994125 \
-pre_route_cap 1.10066 \
-post_route_cap 0.960234 \
-post_route_cross_cap 1.22327 \
-pre_route_clock_res 0 \
-pre_route_clock_cap 0.967898 \
-post_route_clock_cap 0.969117 \
-temperature 125 \
-qrc_tech ${{::IMEX::libVar}}/mmmc/qrcTechFile \
create_opcond -name PVT1 \
-process 1 \
-voltage 2 \
-temperature 120 \
create_delay_corner -name slow_max \
-timing_condition {default_mapping_tc_1} \
-rc_corner rc_max
create_delay_corner -name fast_min \
-timing_condition {default_mapping_tc_2} \
-rc_corner rc_min
create_constraint_mode -name functional \
-sdc_files \
[list ${{::IMEX::libVar}}/mmmc/eagle.sdc]
create_analysis_view -name func_slow_max -constraint_mode functional -delay_corner \
slow_max
```

```
create_analysis_view -name func_fast_min -constraint_mode functional -delay_corner
fast min
set_analysis_view -setup [list func_slow_max] -hold [list func_fast_min]
```

## 5.13 Saving Multi-Mode Multi-Corner Configuration

The software stores multi-mode multi-corner configuration as Tcl commands in the tempus.cmd file. This file contains information on all the library sets, RC corners, delay corners, constraint modes, and analysis view definitions that you created. You can reload the configuration information into a design session by sourcing the tempus.cmd file.

You can also save the design using the [save\\_design](#) command. If constraints are sourced interactively then a designer can save the design after interactive reading of constraints through this command.

## 5.14 Controlling Multi-Mode Multi-Corner Analysis through the Flow

The following table describes how Tempus applications behave in multi-mode multi-corner analysis mode throughout the flow:

**Table 5-1 Application Behavior in Multi-Mode Multi-Corner Analysis Mode**

| Flow Step         | Commands                                   | Application Behavior                                                                                          |
|-------------------|--------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| Delay calculation | <a href="#"><u>write_sdf</u></a> -view     | Uses the early and late delays for the specified analysis view to populate the min and max SDF triplet slots. |
| Timing Signoff    | <a href="#"><u>report_timing</u></a> -view | Analyzes all active analysis views in the design, or for a specified view (-view) concurrently.               |

## 5.15 Performing Timing Analysis

The software performs multi-mode multi-corner timing analysis concurrently on all active analysis views in the design. To run multi-mode multi-corner timing analysis, you use the following command:

- [report\\_timing](#)

By default, the `report_timing` command analyzes all active analysis views in the design concurrently. The timing report details indicate the views that are responsible for a path. By default, the software returns only one path per end point.

If you have a large number of views, you can use the `report_timing -view` command to process multiple views. The software stores results for each view, along with an aggregated report in machine-readable timing reports (.mtarpt). You can view the results graphically in the Tempus Timing Debug environment. You can use this procedure to determine the critical view subsets that should be used to drive the implementation flow.

**Note:** The SPEF data is only read after a view is created to avoid multi-corner errors.

## 5.16 Generating Timing Reports

The software can generate various reports that contain timing information for each active analysis view.

For more information on timing reports, see the `report_timing` *Sample Reports* section in the *Tempus Text Command Reference*.

For other reports, see the `report_*` commands in the “Timing Analysis Commands” chapter.

# Distributed Multi-Mode Multi-Corner Timing Analysis

---

- 6.1 [Overview](#) on page 22
- 6.2 [Performing Distributed Multi-Mode Multi-Corner Analysis](#) on page 23
- 6.3 [Performing Concurrent MMMMC Analysis in DMMMC Mode](#) on page 34
- 6.4 [Setting Up MSV/CPF-Based Design in DMMMC Mode](#) on page 34
- 6.5 [Performing Setup/Hold View Analysis and Reporting](#) on page 36
- 6.6 [Running Single Interactive User Interface in DMMMC Mode](#) on page 37
- 6.7 [Running Interactive ECO on Multiple Views in DMMMC Mode](#) on page 38
- 6.8 [Important Guidelines and Troubleshooting DMMMC](#) on page 41

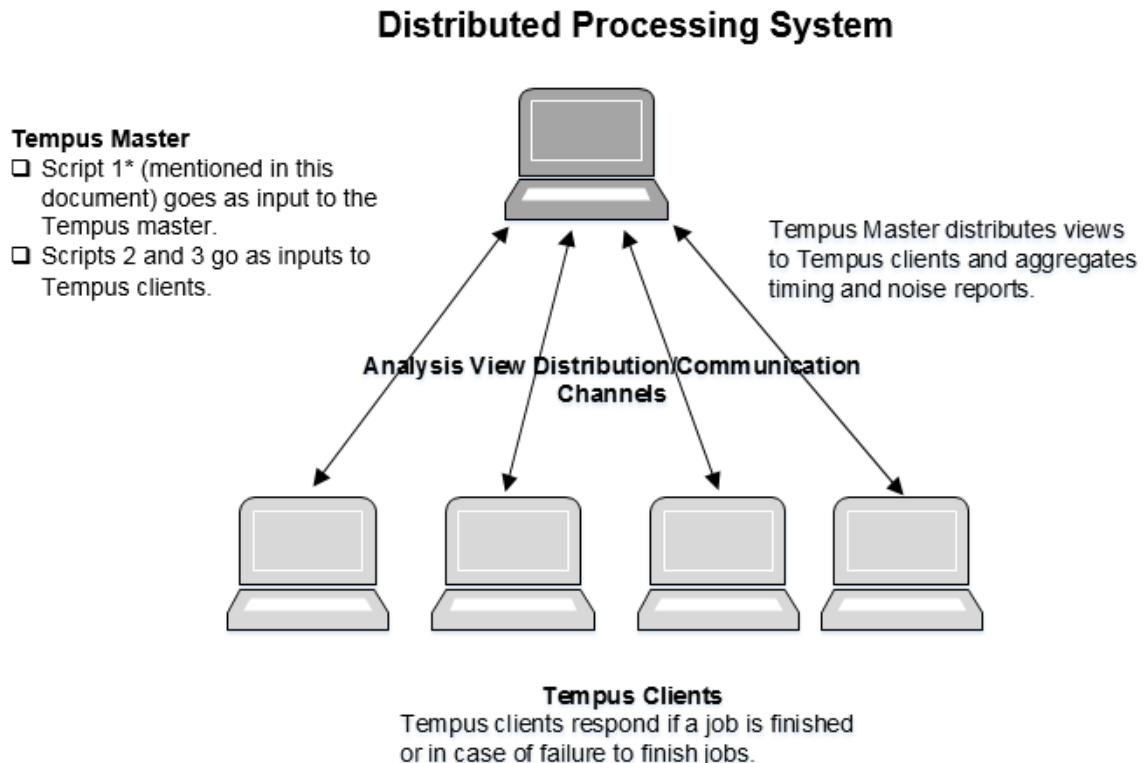
## 6.1 Overview

The multi-mode multi-corner (MMMC) timing analysis can be run in distributed mode also. Distributed multi-mode multi-corner (DMMMC) enables controlled distribution of various analysis views to a selected or shared group of machines, and then merges the reports generated for timing and signal integrity across the views.

This chapter describes how an existing MMMC setup can be transitioned to distributed MMMC (that is, to multiple machines/CPUs). These multiple machines/CPUs will be referred as Tempus clients in this chapter.

Distributed MMMC is a method of processing timing analysis views where the runs are distributed on several Tempus clients. This allows you to run Timing/SI analysis for different analysis views on multiple Tempus clients and then merge/aggregate the generated reports from various views into a single report. The flow is shown in the figure below.

**Figure 6-1 DMMMC Flow**



The Tempus master invokes a set of Tempus clients on the specified machines/CPUs, using the set distribute host command. Besides tracking the activity on Tempus clients, the

Tempus master also governs the responsibility of adjusting queued analysis jobs to active Tempus clients, in case some Tempus clients fail due to an unexpected event.

Distributed MMMC in Tempus enables:

- Parallel analysis of multiple MMMC views running simultaneously on multiple Tempus clients controlled via a Tempus Master session.
- Merging of timing reports.
- Saving a significant amount of real time as compared to sequential runs when there are a large number of views to analyze.
- Accelerated utilization of available computing power (multiple CPU cores) on local or remote machines/hardware via super threading.

#### ***Licensing Requirements***

Refer to “[Tempus License Options for Distributed MMMC \(DMMMC\)](#)” in the “[Product and Licensing Information](#)” chapter.

#### ***Restrictions***

The following restriction applies to distributed MMMC analysis:

- Distributed MMMC analysis is supported in GUI mode. However, the GUI menu and forms are not available for DMMMC commands.

## **6.2 Performing Distributed Multi-Mode Multi-Corner Analysis**

Distributed multi-mode multi-corner analysis is performed by reading a flow script into the master Tempus server, which then distributes the analysis runs to the user-specified machines. Analysis can further be multi-threaded on these machines.

The software saves all data from the distributed analysis runs in a user-specified output directory (specified with the [distribute\\_read\\_design](#) command).

- The software generates view-specific reports in directories with the corresponding view name within this directory.  
When concurrent multi-mode multi-corner timing analysis (CMMMC) is enabled, within a DMMMC framework, the software generates CMMMC session-specific reports in directories that have the corresponding CMMMC session names, under a user-defined

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

output directory (specified using the `distribute_read_design -outdir` parameter). The CMMMC session names are `cmmmc_1`, `cmmmc_2` and so on. The software creates a mapping file (`views_dir_map.rpt`) to show the mapping of CMMMC session names with a list of the combined views. The master log file contains messages that refer to the mapping file. If the mapping file already exists in the run directory, software will create the mapping file with name `views_dir_map.rpt1`, `views_dir_map.rpt2` and so on.

An example of `views_dir_map.rpt` file is given below:

```
Dir: cmmmc_1 Views: {view1 view2}
Dir: cmmmc_2 Views: {view3 view4}
Dir: cmmmc_3 Views: {view5}
```

For more information on how to enable CMMMC, you can refer to the [distribute\\_views](#) command.

- In the DMMMC analysis flow, client monitoring is enabled by default. This feature enables robust master/client resource monitoring and re-launches the clients when they fail due to resource overloading. The master monitors the health of clients at regular intervals and writes the information about per client resources (cpu/memory/tmp) in the `host-monitor.log` file. If the `host-monitor` log file already exists in the run directory, software will create the new file with name `host-monitor.log1`, `host-monitor.log2` and so on.

An example report is given below:

```
#####
Status keys: E excellent, G good, B bad, T terrible.
Memory values:
total = Real memory of host
progs = Real memory used by processes, including swappable cache
used = Real memory used by processes, excluding swappable cache
Started at: 16/10/04 01:54:14
sjfib003(0) TMPDIR is : /tmp/ssv_tmpdir_11610_OIMPtR
=====
Host CPU Memory (GB) TMPDIR (GB)
name(id) util % total progs used %used used avail
=====
01:54:14
sjfib003(0) 44 E 377 193 51 13 E 0.00 228.19 E
01:55:14
sjfib003(0) 40 E 377 193 51 13 E 0.00 228.19 E
ec1-hw025(1) 33 E 330 110 26 7 E 0.22 432.97 E
ec1-hw025(2) 33 E 330 110 26 7 E 0.22 432.97 E
#####
```

The software checks for overloading of client resources and issues appropriate warning messages for over-utilization of client resources. If client resources are over-utilized and view analysis is not able to proceed, the software will automatically terminate the view job and restart it on another client.

The DMMMC flow enables re-launch of clients using the same LSF configuration in the LSF mode, if any overloading (cpu/mem) occurs in clients during view analysis. When set\_distribute\_host -local mode is set, the re-launch feature is disabled.

## Handling Scripts

For distributed MMMC analysis the following Tcl scripts are required:

- [Master Script](#) on page 25
- [Design Loading and MMMC Configuration Script](#) on page 28
- [Parasitic Information, Analysis, and Report Generation Script](#) on page 30

## Master Script

The master script is the overall flow script that runs the entire Tempus distributed MMMC session. This script includes all generic environment variables, distribution commands, and commands for specifying CPU usage, running DMMMC analysis, and merging reports.

The master script is read into the master Tempus server, which then distributes the analysis runs to the user-specified machines.

### *Creating the Master Script*

The following steps list the order in which you should define the commands in the master script that are necessary for running a distributed MMMC analysis session.

1. Specify environment variables or shell variables to be used in the flow.
2. Specify distribution infrastructure like machine names, mode of execution (LSF or rsh), by using the set\_distribute\_host command.

For example:

```
set_distribute_host -rsh -add {hostName1 hostName2 hostName3 hostName4}
```

3. Specify the Tempus client usage by using the set\_multi\_cpu\_usage -remoteHost command. The number specified here must be less than or equal to the number of machines specified in step 2, (if specific machines have been requested using the -rsh parameter of the set\_distribute\_host command).

For example:

```
set_multi_cpu_usage -remoteHost 4
```

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

The following example shows that jobs on each remote host can further be multi-threaded:

```
set_multi_cpu_usage -remoteHost 4 -cpuPerRemoteHost 4
```

If LSF distribution is used, then you can specify the number of Tempus clients to be used. The software needs CPU usage information in order to figure out how many licenses the system can use for distribution.

4. Load the design in distributed mode using the distribute\_read\_design command. This requires providing an input script that contains all the design loading commands, except the read\_spec and read\_parasitics command. The distribute\_read\_design command opens up N number of Tempus clients (as specified with the set\_multi\_cpu\_usage -remoteHost) for the Tempus distributed processing session, and loads the design script into the Tempus clients.

For example:

```
distribute_read_design
-design_script $rootDir/design_load.tcl
-outdir $output_directory
```

5. Specify the active setup and hold analysis views.

For example:

```
set Views [list func_ss_rcmax test_ss_rcmax func_ss_rctyp]
```

6. Distribute and analyze views using the distribute\_views command. This command distributes the active setup and hold analysis views to Tempus clients, and sources the specified command script to perform timing and signal integrity analysis.

For example:

```
distribute_views
-views $Views
-script $rootDir/analysis.tcl
```

7. Merge the timing and noise reports generated from various servers using the merge\_timing\_reports command.
8. Close all the Tempus clients using the exit\_servers command to end the distributed processing session.

The following sample master script needs to be provided as input to the Tempus master.

#### **Example: Sample Master Script**

```
Script 1: Master.tcl
Setup the environment variables or shell variables to be used in the flow.
```

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

```
source env.tcl
set cwd [pwd]
set outDir "$cwd/Output"

Instructs Tempus to launch jobs in rsh. You can specify the list of machines that can be used
to launch utility servers. You can also specify LSF queue (refer to the Tempus Text Command
Reference).
set distribute host -rsh -add {hostname1 hostname2}

or

set_distribute_host -lsf -queue <queue-name> -args {-x -W 30} -resource
"OSNAME==Linux && OSREL==EE50 rusage [mem=10000]"

Instructs 3 Tempus clients to be used for this session run. Each Tempus client will use 8
CPUs for multi-threading.
set multi cpu usage -remoteHost 3 -cpuPerRemoteHost 8

Invokes the Tempus clients based on the above information and loads Script-2
(design_load.tcl) on each client. Specified "outDir" is the area where the Tempus client log
files will be generated. All the outputs relevant to this session will be created here. The master
log will reside in the directory where the run is launched.
distribute read design -design_script $rootDir/design_load.tcl -outdir
$outDir

Distributes list of specified analysis views to Tempus clients and processes the commands
specified in Script-3 (analysis.tcl) on each client.
set views [list func_ss_rcmax test_ss_rcmax func_ss_rctyp]
distribute views -views $views -script $rootDir/analysis.tcl

Merges the timing report (machine readable format) files from specified views and
generates a merged view file. The paths are sorted based on the slack value.
merge_timing_reports -view_dirs { func_ss_rcmax test_ss_rcmax func_ss_rctyp } -
base_report hold_worst.rpt -base_dir ./dist_mmmc/out > merged_hold_worst.rpt

#Close Tempus session
exit_servers

For more information on loading a MMMC configuration script, see Design Loading and MMMC Configuration Script on page 28.

For more information on parasitics, see Parasitic Information, Analysis, and Report Generation Script on page 30.
```

## 6.2.1 Design Loading and MMMC Configuration Script

This script contains the common information that needs to be shared by all Tempus clients. The script must include the design loading commands, as well as all MMMC configuration commands, for example, `create library set`, `create delay corner`, and `create constraint mode`.

The analysis views are specified with the `distribute_views` command, which distributes them internally to the Tempus clients for processing. This command is automatically issued by Tempus master via a distribution method (`distribute_views`).

**Note:** The script must not include parasitic file loading commands (`read_spef`) and analysis view setting commands (`set_analysis_view`). If specified, these commands will be ignored by the distribution mechanism.

This script is loaded using the `distribute_read_design` command.

### Example: Sample MMMC Configuration Script

```
Script 2 : "design_load.tcl"
read_design $rootDir/sample_chip.enc.dat

create_library_set -name libs_best -timing [list $rootDir/library/stdcell//fast.lib \
$rootDir/ ram_128x16A_fast_syn.lib \
$rootDir/ram_256x16A_fast_syn.lib \
$rootDir/rom_512x16A_fast_syn.lib \
$rootDir/pllclk_fast.lib] -si [list \
$rootDir/ cdb/artisan_ff.cdb]

create_library_set -name libs_typ -timing [list $rootDir/library/stdcell//typical.lib \
$rootDir/ram_128x16A_typical_syn.lib \
$rootDir/ram_256x16A_typical_syn.lib \
$rootDir/rom_512x16A_typical_syn.lib \
$rootDir/pllclk_slow.lib] -si [list \
$rootDir/cdb/artisan_ss.cdb]

create_library_set -name libs_worst -timing [list $rootDir/library/stdcell//slow.lib \
$rootDir/ram_128x16A_slow_syn.lib \
$rootDir/ram_256x16A_slow_syn.lib \
$rootDir/rom_512x16A_slow_syn.lib \
$rootDir/pllclk_slow.lib] -si [list \
$rootDir/cdb/artisan_ss.cdb]

create_rc_corner -name rcMin
create_rc_corner -name rcMax
create_rc_corner -name rcTyp

create_op_cond -name 1.98V_0C -library $rootDir/fast.lib -P 1 -V 1.98 -T 0
create_op_cond -name 1.80V_25C -library $rootDir/typical.lib -P 1 -V 1.80 -T 25
create_op_cond -name 1.62V_125C -library $rootDir/slow.lib -P 1 -V 1.62 -T 125
create_delay_corner -name typ_rcMax -library_set libs_typ -opcond_library typical \
-opcond 1.80V_25C -rc_corner rcMax
create_delay_corner -name typ_rcMin -library_set libs_typ -opcond_library typical \
\
```

# Tempus User Guide

## Distributed Multi-Mode Multi-Corner Timing Analysis

---

```
-opcond 1.80V_25C -rc_corner rcMin
create_delay_corner -name worst_rcTyp -library_set libs_worst -opcond_library slow \
\
-opcond 1.62V_125C -rc_corner rcTyp
create_delay_corner -name worst_rcMax -library_set libs_worst -opcond_library slow \
\
-opcond 1.62V_125C -rc_corner rcMax
create_delay_corner -name best_rcTyp -library_set libs_best -opcond_library fast \
-opcond 1.98V_0C -rc_corner rcTyp
create_constraint_mode -name func ${rootDir}"/sample_chip.sdc
create_constraint_mode -name test ${rootDir}"/sample_chip_testmode.sdc
create_analysis_view -name func_ss_rcmax -constraint_mode func -delay_corner
worst_rcMax
create_analysis_view -name test_ss_rcmax -constraint_mode test -delay_corner
worst_rcMax
create_analysis_view -name func_ss_rctyp -constraint_mode func -delay_corner
worst_rcTyp
create_analysis_view -name func_best_rctyp -constraint_mode func -delay_corner
best_rcTyp
```

### **Handling TCL Vars in DMMMC Setup**

Note the following points for handling Tcl variables:

- Tcl Variables defined in master.tcl are available in Tempus master.
- Tcl Variables defined in design\_load.tcl or analysis.tcl will be available in Tempus clients.
- Tempus software global variables are available in both master and clients. These global variables will have an impact on the analysis only when they are set inside the design\_load.tcl or analysis.tcl file.
- Variables set by either distribute\_command { ... } or in interactive mode (enabled with the set distribute\_mmmc\_mode -interactive true setting) will be available or changed only on Tempus clients.

**Note:** For more information on interactive mode, refer to sections “Running Interactive User Interface in DMMMC” and “Interactive ECO on Multiple Views in the *Tempus User Guide*.

- Use the get\_distribute\_variables command to fetch the value of a Tempus client Tcl variable from the Tempus master session.

### **Example: Setting Tcl vars on Tempus clients from master**

```
Create scripts to be run on client/slaves
distribute_command {
set clocks [get_clocks *]
set clock_names [get_object_name $clocks]
set totalClocks [sizeof_collection $clocks]
}
```

# Tempus User Guide

## Distributed Multi-Mode Multi-Corner Timing Analysis

---

```
Fetch client variable as array back to master
Specify name of client variables w/o '$'
get_distribute_variables "totalClocks clock_names"
foreach viewName [array names totalClocks] {
 Puts "View '$viewName' has total '$totalClocks($viewName)' clocks named
 '$clock_names($viewName)'"
}
distribute_command {report_timing}
```

### Parasitic Information, Analysis, and Report Generation Script

This script contains commands for SPEF reading, analysis, reporting, and ECO DB generation (for Tempus ECO flow, if needed).

The commands for reading parasitic information must be specified in the script before the reporting commands. You must specify parasitic files for all the RC corners. For example:

```
read_spef -rc_corner corner1 rc1.spef
read_spef -rc_corner corner2 rc2.spef
read_spef -rc_corner corner3 rc3.spef
read_spef -rc_corner corner4 rc4.spef
```

If the file includes report commands that redirect their output to a specified file (for example, report\_timing >r.rpt), then the software generates the file in the active view directory (output directory specified with the distribute\_read\_design command). For report commands with no specified redirection, the software generates the report in the corresponding log file.

This script is loaded using the distribute\_views command.

### Example: Sample Parasitic Information and Report Generation Script

```
Script 3 : "analysis.tcl"
read_spef -rc_corner rcTyp SPEF/sample_chip_typ.spef.gz
read_spef -rc_corner rcMin SPEF/sample_chip_fastspef.gz
read_spef -rc_corner rcMax SPEF/sample_chip_slow.spef.gz
set_design_mode -process 28
set_delay_cal_mode -SIAware true
set_si_mode -enable_glitch_report true
report_timing -machine_readable -max_path 10 -check_type -setup >
setup_report_max.rpt
report_timing -nworst 1 -check_type -setup > setup_worst.rpt
report_timing -max_path 10 -machine_readable -check_type hold >
hold_report_max.rpt
report_timing -nworst 1 -check_type hold > hold_worst.rpt
write_sdf sample_ets.sdf
write_sdc constraint.sdc
write_eco_opt_db
```

## Saving and Restoring DMMMC Sessions

You can save and restore distributed MMMC sessions.

The save/restore flow is as follows:

1. The Tempus software reads the distributed MMMC data using the following command:

```
distribute_read_design -design_script design_load.tcl
```

2. You can save the design using the following command:

```
distribute_views -save_design list_of_views (or you can specify all)
```

The `distribute_views -save_design` parameter creates N parallel saved design databases, where N is the number of views specified in the list of views. You can provide a list of views for which the saved session is required.

3. Start another Tempus master session to restore the above saved sessions. You can also restore individual saved sessions in a non-distributed Tempus run.

4. You can use the following command to create N Tempus clients, each of which will load one saved session:

```
distribute_read_design -restore_design list_of_views -restore_dir
pathname_of_directory_of_saved_session
```

5. The `-restore_design` parameter specifies the design views to be restored.

The following script examples show distributed MMMC save and restore flows:

### Distributed MMMC Save Flow

The following example saves sessions for N views:

```
set_distribute_host -local | -rsh ...
set_multi_cpu_usage -remoteHosts N
distribute_read_design -design_script design_load.tcl -outdir Output
set Views [list func_rmax func_rcmin test_rcmax]
distribute_views -views $Views -script analysis.tcl -save_design all
```

### Distributed MMMC Restore Flow

The following example restores saved sessions for N views:

```
set_distribute_host -local
set_multicpu_usage -remoteHosts N
set Views [list func_rmax func_rcmin test_rcmax]
```

# output is the same directory used in the above example where views were saved.

```
distribute_read_design -restore_design $Views -restore_dir Output -outdir
new_output
distribute_command {report_timing}
```

To restore a single view from DMMMC saved session on a single machine environment use the following commands:

```
read_design Output/view1.enc.dat <topcell>
report_timing
```

## The Output File Structure

The MMMC commands generate output data in directories as view names inside the specified output directory. You can specify the output directory in the `distribute_read_design` command with the `-outdir` parameter. For example, consider the following scenario:

1. A design containing three views (view1, view2, and view3).
2. Run `distribute read design -design_script design_load.tcl -outdir Output` command.
3. Specify the following commands in `analysis.tcl` (script-3):

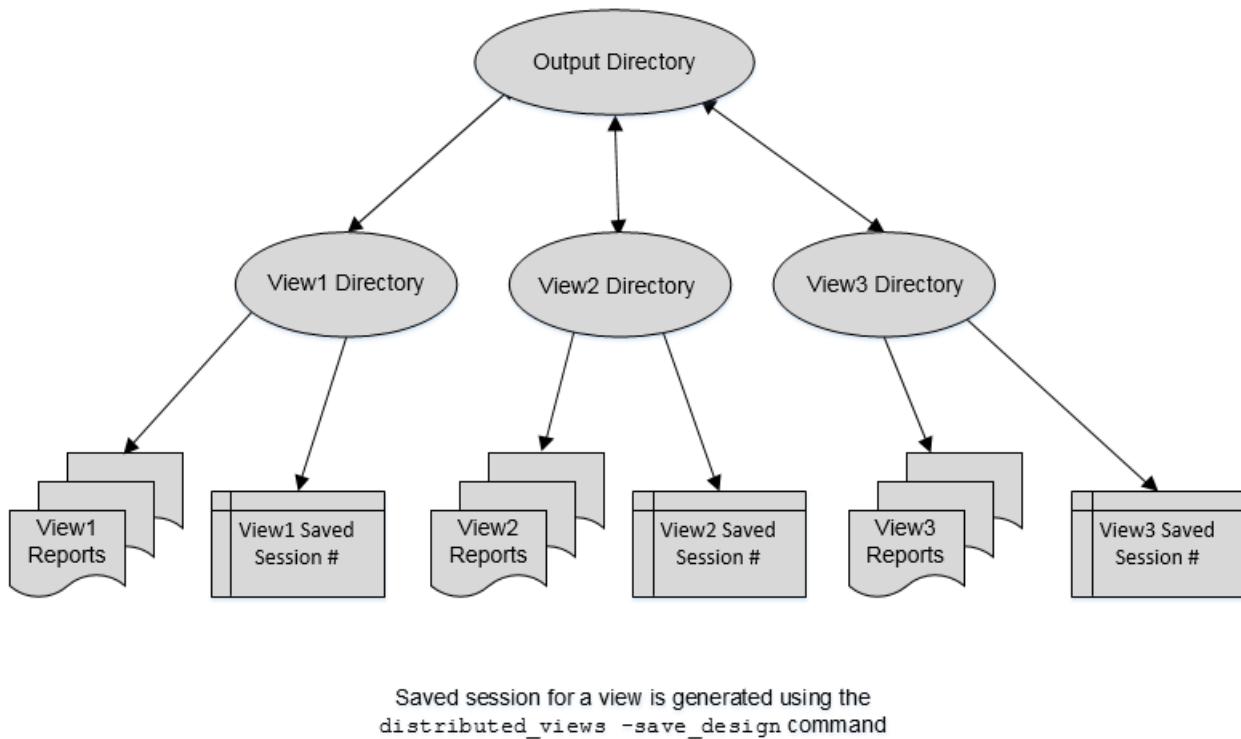
```
report_timing -max_paths 200 > setup_path.rpt
report_timing -max_paths 200 -early > hold_path.rpt
report_path_exceptions -early > rpe.rpt
```

The software will generate the following three reports inside each view's directory:

```
./Output/view1/setup_path.rpt
./Output/view1/hold_path.rpt
./Output/view1/rpe.rpt
```

All the Tempus client-specific .log/.cmd files reports will be saved in the `./Output` directory. The directory structure is shown in the figure below.

**Figure 6-2 Output directory structure in DMMMC mode**



## Aggregating Timing Reports

After processing the views in distributed mode, segregated reports will be available for each view in a specific directory that is to be analyzed. One of the prime requirements is to have a holistic look of all the analysis views for further action.

The `merge_timing_reports` command can be used to merge reports that are generated using the reporting commands, such as, `report_timing` and `report_constraint`.

You can merge view specific standard timing reports that can be generated using the `report_timing` and `report_constraints` commands.

For GUI-based timing analysis, you can merge view-specific timing reports generated in machine readable format (using `report_timing -machine_readable` command in `analysis.tcl`), to create a single merged report, which can be loaded in Tempus Global Timing Debug (GTD).

Merging of timing reports is done based on slack criticality. As each path will contain the “view” field that represents the view to which a path belongs, so analysis of critical views becomes easier.

- Output generated from the `merge_timing_reports` command will be saved in a user-specified output file.

## 6.3 Performing Concurrent MMMMC Analysis in DMMMC Mode

By default, the `distribute_views` command distributes a single view for each Tempus client. The command can also perform concurrent multi-mode multi-corner (CMMMC) timing analysis within the DMMMC framework. You can combine the analysis views for CMMMC analysis on Tempus client(s). This can be done in the following two ways:

- **Delay Corner-Based Auto Grouping:**

You can specify a simple list of view names using the `distribute_views -view` option and use the `-combine_num_views` option to specify the maximum number of views to be combined per Tempus client. For example:

```
distribute_views -combine_num_views 4 -views {setup1 setup2 setup3
setup4 setup5 setup6 setup7 setup8} -script ...
```

Assuming `setup1` to `setup4` to have delay corner `dc1` and `setup5` to `setup8` to have delay corner `dc2`, then the above example will group `setup1` to `setup4` to Tempus client1, and `setup5` to `setup8` will be grouped to Tempus client2.

- **User-Controlled View Grouping:**

You can specify a list of list formats for combining multiple concurrent views on Tempus client(s). For example,

```
distribute_views -views {{setup1 setup2 setup3 setup4} {setup5 setup6
setup7 setup8}} -script ...
```

The above command will group `setup1` to `setup4` views to Tempus client1 and will group `setup5` to `setup8` views to Tempus client2, irrespective of the delay corner.

To get optimum run time while using view grouping, it is recommended to use delay corner-based auto grouping, as mentioned in the example above.

## 6.4 Setting Up MSV/CPF-Based Design in DMMMC Mode

In this flow, the common power format (CPF) file should not contain any timing related information. All the library data is present in the `viewdefinition.tcl` file.

**Script -1 (master.tcl):**

Same as in non-MSV DMMMC flow.

**Script -2 (design\_load.tcl):**

The following points should be noted while migrating to this flow:

1. All the library related data and operating conditions are defined in the viewdefinition.tcl file.
2. Power domains are created in the power.cpf file. This file should not contain any timing or operating conditions related information.
3. Timing data and PVT conditions are linked to power domains, by using the update\_delay\_corner -power\_domain command in the `viewdefinition.tcl` file.
4. Before reading the CPF, the `update_delay_corner` command can be used to:
  - a) Create place-holder domains.
  - b) Populate domains with appropriate MSV data during CPF loading.

**Sample design\_load.tcl:**

```
Script 2 : "design_load.tcl"
source $designDir/settings.tcl
source $dataDir/global_file.tcl
read_verilog $dataDir/netlist1.final.v
read_verilog $dataDir/netlist2.final.v
set out_dir out_dir
read_view_definition viewdefinition.tcl
set_top_module xyz
read_power_domain -cpf power.cpf
set_interactive_constraint_modes [all_constraint_modes -active]
set_propagated_clock [all_clocks]
set_annotated_delay -cell -to [all_inputs] 0

viewdefinition.tcl
create_library_set -name libset_MIN -timing "$dataDir/lib/XYZ1.lib.gz $dataDir/lib/XYZ2.lib"
create_library_set -name libset_MAX -timing "$dataDir/lib/XYZ3.lib $dataDir/lib/XYZ4.lib"
create_library_set -name dummy_libset1 -timing dummy1.lib
create_op_cond -name opcond_MIN -V 1.32 -P 1 -T -40 -library_file { $dataDir/lib/ABC1.lib.gz }
create_op_cond -name opcond_MAX -V 1.08 -P 1 -T 125 -library_file { $dataDir/lib/ABC2.lib.gz }
create_op_cond -name dummy_opcond -V 1 -P 1 -T 125 -library_file { dummy.lib }
create_rc_corner -name MAX_rc_corner
create_rc_corner -name MIN_rc_corner
```

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

```
create_delay_corner -name MIN_dc \
-library_set libset_MIN \
-rc_corner MIN_rc_corner \
-opcond opcond_MIN
create_delay_corner -name MAX_dc \
-library_set libset_MAX \
-rc_corner MAX_rc_corner \
-opcond opcond_MAX
update_delay_corner -name MAX_dc -power_domain dummy_pd1 \
-library_set dummy_libset \
-opcond dummy_opcond1
create_constraint_mode -name MIN_CM -sdc_files "$dataDir/constraints/CM_MIN.tcl
$dataDir/config_sdc1.tcl"
create_constraint_mode -name MAX_CM -sdc_files "$dataDir/constraints/CM_MAX.tcl
$dataDir/config_sdc2.tcl"
create_analysis_view -name view_MIN -constraint_mode MIN_CM -delay_corner MIN_dc
create_analysis_view -name view_MAX -constraint_mode MAX_CM -delay_corner MAX_dc
power.cpf
set_cpf_version 1.1
set_design Test
create_power_domain -name dummy_pd0 -default
create_power_domain -name dummy_pd1
end_design
```

#### **Script - 3 (analysis.tcl):**

This script contains all the SPEF reading, analysis, and reporting commands.

```
Script 3: "analysis.tcl"
read_spef -rc_corner MIN_rc_corner "$dataDir/pqr1.spef.gz $dataDir/pqr2.spef.gz "
read_spef -rc_corner MAX_rc_corner "$dataDir/pqr3.spef.gz $dataDir/pqr4.spef.gz "
set_delay_cal_mode -SIAware true
report_timing -max_path 10 -check_type -setup > setup_report_max.rpt
report_timing -nworst 1 -check_type -setup > setup_worst.rpt
report_timing -max_path 10 -check_type hold > hold_report_max.rpt
report_timing -nworst 1 -check_type hold > hold_worst.rpt
```

## **6.5 Performing Setup/Hold View Analysis and Reporting**

You can perform view specific analysis and reporting in DMMMC mode.

The script example below demonstrates how to perform view-specific reporting. The following example shows 4 views. The section in the analysis.tcl script, which is provided as input to the distribute\_views command, generates setup.rpt and hold.rpt reports for two views each:

#### **Example: Performing view-specific reporting**

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

```
set mySetupViews "view1 view2"
set myHoldViews "view3 view4"
```

#### #Reports For Setup Analysis Views

```
set currentView [all_setup_analysis_views]
set reportSetup 0
foreach vName $currentView { if { [lsearch $mySetupViews $vName] != -1 } { set
reportSetup 1 ; break; } }
if { $reportSetup } {
```

#### # All Setup View reporting Commands

```
report_timing -late > setup.rpt}
```

#### #Reports for hold analysis views

```
set currentView [all_hold_analysis_views] ;
set reportHold 0
foreach vName $currentView { if { [lsearch $myHoldViews $vName] != -1 } { set
reportHold 1 ; break; } }
if { $reportHold } {
```

```
All Hold View reporting Commands.
```

## 6.6 Running Single Interactive User Interface in DMMMC Mode

The Tempus software supports distributed MMMC interactive interface mode. In this mode, some commands that are run in a master session are also implemented to all the Tempus clients.

The following are distributed MMMC interactive interface commands:

- set\_distribute\_mmmc\_mode
- distribute\_command
- get\_distribute\_variables

The DMMMC interactive interface is available only after the views have been dispatched to Tempus clients.

The DMMMC interactive mode is successfully enabled when the Tempus shell prompt changes from the standard `tempus>` to `tempus_dmmmc>`. Once in interactive mode (set using `set_distribute_mmmc_mode -interactive true` setting), the commands are no longer needed to be dispatched using the `distribute_command {}`, and can be directly run in the Tempus master session.

The DMMMC interactive mode supports reporting commands, timing constraints, file writing commands (such as `write_sdc`) and ECO commands.

The DMMMC interactive interface is available only after the `distribute_views` commands are run.

### **Example: Distributed MMMC Interactive Flow**

The following example shows the distributed MMMC interactive flow:

```
tempus> set viewnames {view01 view02 ...}
tempus> distribute_read_design -restore_design (or you can use distribute_views -views $viewnames)

To enable DMMMC interactive mode.
tempus> set_distribute_mmmc_mode -views $viewnames -interactive true

Displays timing for each selected view on master console.
tempus_dmmmc> report_timing

Run some ECO commands on interactive views.
Performs ECO in each view-server independently.
tempus_dmmmc> change_cell -inst U1 -upsized

Run some reporting commands on interactive views.
Displays updated timing for each view on the master console.
tempus_dmmmc> report_timing
tempus_dmmmc> report_timing -view view01

Will turn on all the loaded view-servers for interactive mode.
tempus> set_distribute_mmmc_mode -interactive true -views all

Apply constraint commands on interactive views.
tempus_dmmmc> set_interactive_constraint_modes [all_constraint_modes -active]
tempus_dmmmc> set_input_delay -clock CK 0.2 {EN}
tempus_dmmmc> set_case_analysis 0 SEL
```

The `distribute_command` can be run several times to create a synchronized flow with multiple commands.

## **6.7 Running Interactive ECO on Multiple Views in DMMMC Mode**

You can perform the following steps interactively by using the interactive DMMMC feature, after basic distributed analysis has been performed:

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

- Perform What-If ECO and check results from multiple views using `-evaluate_all` or `evaluate_only` parameters of ECO commands.
- Perform ECO commit operation in Tempus on multiple views in parallel environment.
- Run `get_property` query to know the slack and other property values with ECO commands from multiple views.
- Save distributed session with ECO.

The above steps can be executed on the interactive prompt, or directly from the master in the distributed environment.

#### **Example: Interactive ECO Usage in Distributed MMMC Environment on Interactive prompt**

#MultiCore Machine : Local mode (Default)

```
tempus> set_distribute_host -local
```

#RSH mode

```
tempus> set_distribute_host -rsh -add {machine1 machine2} \
```

OR

#LSF mode useful for getting a machine from a remote farm environment

```
tempus> set_distribute_host -lsf -queue normal -lsf Resource "OSNAME==Linux && OSREL==EE50"
```

```
tempus> set_multi_cpu_usage -remoteHost 2 -cpuPerRemoteHost 8
```

# Create servers with basic MMMC design

```
tempus> distribute_read_design -design_script design_load.tcl -outdir output
tempus> set views [list view01 view02]
```

# Run analysis script on select views at remote machines

```
tempus> distribute_views -views $views -script analysis.tcl
```

# Distribute Interactive Mode commands, scripting, and so on

```
tempus> set_distribute_mmmc_mode -interactive true -views all
```

# Get reports back on master console

```
tempus_dmmmc> report_timing -through [get_cells i2]
```

# Scripts examples to be run on remote hosts

```
tempus_dmmmc> set clocks [get_clocks *]
tempus_dmmmc> set clock_names [get_object_name $clocks]
tempus_dmmmc> set totalClocks [sizeof_collection $clocks]
```

# Tempus User Guide

## Distributed Multi-Mode Multi-Corner Timing Analysis

---

# Perform What-If ECO and check results from multiple views using -evaluate\_all / -evaluate\_only options of ECO commands

```
tempus_dmmmc> change_cell -inst i2 -upsize -evaluate_all
tempus_dmmmc> change_cell -inst i2 -upsize -evaluate_only
```

# Perform ECO commit operation in Tempus on multiple views in parallel environment

```
tempus_dmmmc> change_cell -inst i2 -upsize
```

# Perform get\_property query to know slack and other property values with ECO commands from multiple views

```
tempus_dmmmc> report_timing -through [get_cells i2]
tempus_dmmmc> set newSlack [get_property [get_pins i2/A] slack_max_rise]
tempus_dmmmc> set_distribute_mmmc_mode -interactive false
tempus> get_distribute_variables newSlack
```

# Save distributed session with ECO

```
tempus> distribute_views -save_design all
tempus> puts "Waiting remote servers to exit"
```

# Merge Analysis reports

```
tempus> merge_timing_reports -view_dirs {view01 view02} -base_report setup_sta.rpt
-base_dir output > merged_setup.rpt
tempus> merge_timing_report -view_dirs {view01 view2} -base_report hold_sta.rpt -
base_dir output > merged_hold.rpt
```

## LOG Snapshot from Interactive ECO

```
tempus_dmmmc 13> change_cell -inst i2 -upsize -evaluate_all
Variable ::_dmi_result(view01) on master session updated with value '1'.
Variable ::_dmi_result(view02) on master session updated with value '1'.
RESULTS FROM VIEW view02 ###
---- Cell: slow/INVX16 (view: view02) ----
Through-object Slack: 1.842
Max Tran Slack: 4.3224
---- Cell: slow/INVX12 (view: view02) ----
Through-object Slack: 1.831
Max Tran Slack: 4.3495
---- Cell: slow/INVX8 (view: view02) ----
Through-object Slack: 1.784
Max Tran Slack: 4.0948
RESULTS FROM VIEW view01 ###
---- Cell: slow/INVX16 (view: view01) ----
Through-object Slack: 0.542
Max Tran Slack: 4.3224
---- Cell: slow/INVX12 (view: view01) ----
Through-object Slack: 0.530
Max Tran Slack: 4.3495
---- Cell: slow/INVX8 (view: view01) ----
Through-object Slack: 0.484
Max Tran Slack: 4.0948
tempus_dmmmc 14> change_cell -inst i2 -upsize -evaluate_only
Variable ::_dmi_result(view01) on master session updated with value '1'.
Variable ::_dmi_result(view02) on master session updated with value '1'.
```

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

```
RESULTS FROM VIEW view02
---- Cell: slow/CLKINVX1 (view: view02) ----
Through-object Slack: 1.671
Max Tran Slack: 4.3935

RESULTS FROM VIEW view01
---- Cell: slow/CLKINVX1 (view: view01) ----
Through-object Slack: 0.371
Max Tran Slack: 4.3935
0

tempus_dmmmc 15> change_cell -inst i2 -upsized
Variable ::_dmi_result(view01) on master session updated with value '1'.
Variable ::_dmi_result(view02) on master session updated with value '1'.

RESULTS FROM VIEW view02
Resize i2 (INVX1) to CLKINVX1.

RESULTS FROM VIEW view01
Resize i2 (INVX1) to CLKINVX1.
0
tempus_dmmmc 18> set newSlack [get_property [get_pins i2/A] slack_max_rise]
Variable ::_dmi_result(view01) on master session updated with value '1.3085'.
Variable ::_dmi_result(view02) on master session updated with value '-0.9915'.
0
tempus_dmmmc 21> set_distribute_mmmc_mode -interactive false
tempus_dmmmc 22> distribute_views -save_design all
CPUs with views 'view01 view02' have been enabled for interactive distributed
mmmc mode.
Variable ::_dmi_result(view01) on master session updated with value '1'.
Variable ::_dmi_result(view02) on master session updated with value '1'.

RESULTS FROM VIEW view02
Writing Netlist "output_di_mmmc/view02/mmmc.enc.dat/mmmc.v.gz" ...
Saving timing graph ...
Saving configuration ...
Saving preference file output/view02/mmmc.enc.dat/enc.pref.tcl ...
Saving parasitic data in file 'output/view02/mmmc.enc.dat/mmmc.rcdb.d' ...
Writing RC database file using in-memory parasitic updation done by ECO commands.
Creating parasitic data file 'output/view02/mmmc.enc.dat/mmmc.rcdb.d/header.da' in
memory efficient access mode for storing RC.

RESULTS FROM VIEW view01
Writing Netlist "output_di_mmmc/view01/mmmc.enc.dat/mmmc.v.gz" ...
Saving timing graph ...
Saving configuration ...
Saving preference file output/view01/mmmc.enc.dat/enc.pref.tcl ...
Saving parasitic data in file 'output/view01/mmmc.enc.dat/mmmc.rcdb.d' ...
Writing RC database file using in-memory parasitic updation done by ECO commands.
Creating parasitic data file 'output/view01/mmmc.enc.dat/mmmc.rcdb.d/header.da' in
memory efficient access mode for storing RC.
```

## 6.8 Important Guidelines and Troubleshooting DMMMC

This section provides information on various kinds of error scenarios.

## Tempus User Guide

### Distributed Multi-Mode Multi-Corner Timing Analysis

---

Specify all the parasitic file loading commands (pertinent to all RC corners used in the design) in the analysis.tcl (Script 3) script that are accepted as an argument to the distribute\_views command.

- If distributed MMMC session is not getting launched, you need to check the following:
  - There are no errors in your design loading file (design\_load.tcl) that is used as an argument to the distribute\_read\_design command.
  - If the error message “Unable to open socket, connection refused” is displayed, then check for items listed in bullets ‘c’ to ‘g’. In all these cases, the DMMMC system will issue the appropriate error/warning messages.
  - Log on to the machines specified as resources to the DMMMC session. You must have direct login access to these machines (that is, without requiring a password at the time of login). Update your rhosts settings to avoid this issue.
  - Make sure that the specified Tempus clients in the DMMMC setup are not heavily loaded. Enough resources should be available on each Tempus client to process a view.
  - If using LSF mode, then the required LSF variable and environmental settings should be done prior to the launch. Similarly, for rsh the DMMMC system assumes you have configured the RSH access to the Tempus client properly.
  - The commands that are supported beyond timing and noise analysis flow are not used in the scripts. The MMMC parameters and options related to RC extraction must be removed from the design\_load.tcl to run the flow smoothly.
- The set\_analysis\_view command is not allowed in any of the scripts that are accepted as input to the DMMMC system. Specify a list of views using the distribute\_views command. This command will distribute these views internally to the Tempus clients for processing.
- All merging commands work on the Tempus master and the output of these commands will be placed in the specified output file. These merge commands will not follow the protocol of moving the output reports to the specific output area, as mentioned above. This applies to any command that is capable of generating an output specific to a view.
- Single machine MMMC and distributed MMMC cannot be used concurrently in the same Tempus master session.
- DMMMC is a batch mode execution feature - distributed save/restore and interactive command entry capabilities are supported in Tempus.
- Merged noise report will not specify the view name for each record of delay/glitch. You will see multiple entries of records where each record will correspond to a specific

---

**Tempus User Guide**  
Distributed Multi-Mode Multi-Corner Timing Analysis

---

Tempus client. To match the record to a view and make noise reports handy, you need to generate text reports along with the merged view text report.

---

## Distributed Static Timing Analysis

---

- 7.1 [Tempus Timing Analysis Overview](#) on page 45
- 7.2 [Static Timing Analysis \(STA\)](#) on page 45
- 7.3 [Distributed Static Timing Analysis \(DSTA\)](#) on page 45
- 7.4 [Converting STA to DSTA](#) on page 47
- 7.5 [Hardware Recommendations for DSTA](#) on page 48
- 7.6 [Accessing Specific Machines using LSF](#) on page 49
- 7.7 [DSTA Log Files](#) on page 50
- 7.8 [DSTA Operational Tips](#) on page 51
- 7.9 [Distributed Static Timing Analysis Flow](#) on page 54
- 7.10 [Additional Commands Supported in DSTA](#) on page 55
- 7.11 [Commands Not Supported in DSTA Mode](#) on page 58

## 7.1 Tempus Timing Analysis Overview

The Tempus Timing Analysis tool provides enhanced performance and capacity to real world designs by using an innovative massive parallel computer architecture. Timing Analysis has two modes of operation:

- Static Timing Analysis (STA)
- Distributed Static Timing Analysis (DSTA)

These are described below.

### 7.1.1 Static Timing Analysis (STA)

Static timing analysis (STA) refers to a traditional single-machine, multi-threaded use model. Tempus operates efficiently in the STA model with improved performance due to highly optimized multi-threaded routines.

To launch Tempus in the GUI environment, use the following command:

tempus

Use the following command to launch Tempus without the GUI:

`tempus -no_gui`

### 7.1.2 Distributed Static Timing Analysis (DSTA)

Distributed static timing analysis (DSTA) refers to the master/client architecture that fully leverages multiple threads on master/client processes allowing maximum scalability for growing and complex designs. Apart from the overall run time reduction, another significant advantage of the DSTA master/client architecture is the overall per-process memory usage reduction. The master holds the entire design while each of the clients hold a portion of the current timing graph.

You must execute the main script on the master machine - work is shared between multiple client machines.

To launch Tempus in DSTA mode, use the following command:

`tempus -distributed`

Distributed Tempus does not have a GUI mode.

To run the DSTA flow, you must specify the following additional commands:

- set distribute host
- distribute start clients
- distribute partition

**Note:** All the STA commands are not supported in the DSTA mode. For more details, see section [Commands Not Supported in DSTA Mode](#) on page 58.

## Using STA Versus DSTA

The following table describes some of the key features of STA and DSTA:

| STA                                                                                                                                                                                                                                                                                                                                 | DSTA                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>■ Utilizes a single machine and supports multi-threading.</li><li>■ Multi-threaded performance has improved over the earlier timing analysis solution - Encounter Timing System (ETS).</li><li>■ Maintains signoff accuracy.</li><li>■ Ideal for small designs (&lt;15M instances).</li></ul> | <ul style="list-style-type: none"><li>■ Utilizes multiple parallel machines and supports multi-threading.</li><li>■ Leverages large number of less powerful machines.</li><li>■ Reduces overall runtime on large designs.</li><li>■ Reduces per-process memory footprint on large designs.</li><li>■ Maintains signoff accuracy.</li><li>■ Requires minor script changes.</li><li>■ Ideal for large designs (&gt;15M instances).</li></ul> |

The choice of the static timing analysis mode depends on the design size and runtime.

- A one million instance design may not realize the benefits of DSTA.
- A 200 million instance design is not possible using a single machine STA.
- A 15 million instance design may choose STA or DTSA based on runtime expectations.

If you need to reduce the run time, and have already maximized the threading of STA, then Tempus DSTA is the best choice. Alternatively, if the run time is less than an hour and your memory requirements are not a limiting factor, then STA should be selected. Running STA on a large design will cause longer run time and memory usage. Similarly, running DSTA on a small design will consume processors with no significant gain.

## Performance Considerations for STA and DSTA

Use the following to improve performance of static timing analysis:

- LDB for libraries
- RCDB for parasitics
- Pre-defined Tcl constraints
- Local disks with fast read/write speed

## 7.2 Converting STA to DSTA

The procedure to convert a STA script to a DSTA script is given below:

- Run a single view in the STA mode (single machine, 8 threads).
- Investigate if there are any errors.
- Read the library file.
- Read the verilog file.
- Read the SPEF file.
- Read constraints.
- Update timing.
- Custom reporting.

The table below shows sample scripts for STA and DSTA flows:

| STA Script                                                                                                                                                                                                        | DSTA Script                                                                                                                                                                                                                                                                             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| # Non-distributed script<br><pre>puts "Starting script:<br/>set_multi_cpu_usage -localCpu 4<br/>read_verilog<br/>read_lib<br/>set_top_module<br/>read_spef<br/>read_sdc<br/>update_timing<br/>report_timing</pre> | # Distributed script<br><pre>puts "Starting script:<br/>set_multi_cpu_usage ...<br/>set_distribute_host ...<br/>distribute_start_clients<br/>read_verilog<br/>read_lib<br/>set_top_module<br/>read_spef<br/>distribute_partition<br/>read_sdc<br/>update_timing<br/>report_timing</pre> |

The basic scripting rules for DTSA are:

- You must have at least 2 clients.
- Use the set\_multi\_cpu\_usage command before starting a client(s).
- Load parasitics before running the distribute\_partition command.
- Load constraints after the distribute\_partition command.

The set\_multi\_cpu\_usage command can be specified as shown below:

```
set clientCnt 2
set threadCnt 4
set_multi_cpu_usage -cpuPerRemoteHost $threadCnt \
-localCpu $threadCnt -remoteHost $clientCnt
```

You can specify the LSF parameters using the set\_distribute\_host command:

```
set_distribute_host -timeout 300 -lsf -queue lnx64 \
-args {-n 4 -R "(CPUS>=4) span[hosts=1] rusage[mem=15000]"}
```

## 7.3 Hardware Recommendations for DSTA

The following are the guidelines for running DSTA:

- Start with 4 clients, 8 threads per client. Typically, 4 client DSTA will have half the run time of a 0 client STA.
- Add clients for each additional ~15 million instances.
- All clients must have similar performance architecture. The heterogeneous machines will result in dissimilar client run times – slowest client dictates overall run time.
- Memory allocation - 1.5GB peak memory per million instances (master and client).

An example of a 100 million instance design is given below:

- 5 - 10 machines
- 8 - 16 CPUs per machine
- 128 GB RAM per client
- Run time 3-5 hours

The following table shows reasonable client counts and thread counts for various sizes of designs:

| Design Instance Count | Client Count | Client Thread Count | Master Thread Count | Resulting Processor Count | Comments                              |
|-----------------------|--------------|---------------------|---------------------|---------------------------|---------------------------------------|
| A few million         | 2            | 2                   | 2                   | 6                         | For a small tutorial design           |
| 20 million            | 4            | 4                   | 4                   | 20                        | -                                     |
| 50 million            | 4            | 8                   | 8                   | 40                        | -                                     |
| 100 million           | 8            | 8                   | 8                   | 72                        | -                                     |
| 200 million           | 12           | 8                   | 12                  | 108                       | ■ Increased client count saves memory |
| 400 million           | 12           | 12                  | 16                  | 160                       | ■ Increased client count saves memory |

## 7.4 Accessing Specific Machines using LSF

You can specify the following LSF parameters to use specific machines:

- To avoid a bad machine, use:

```
bsub -q ssv -m ssvpe -n 4 \
 -R "(hname != sjfsb416) rusage[mem=3000] " "xterm"
```

- To target a specific machine, use:

```
bsub -q ssv -m ssvpe -n 4 \
 -R "hname = sjfsb416 rusage[mem=3000] " "xterm"
```

- To target a group of machines, use:

```
bsub -q ssv -m ssvpe -n 4 \
 -R "(hname == sjfsb415 || hname == sjfsb416 || \
 hname == sjfsb417 || hname == sjfsb418 || \
 hname == sjfsb419) rusage[mem=30000] " "xterm"
```

## 7.5 DSTA Log Files

### ***Master Log***

Master logs are unique to DSTA. You can name a log file using the following command:

```
tempus -log
```

The default log file name is `./tempus.log`. The file naming increments to `tempus.log1` in the next run. If you do not want to increment, but want to overwrite the existing log file, use the `tempus -overwrite` command.

The master log shows that the master machine starts the clients, and loads the design. The client machines do most of the work with constraints, timing, and reports.

### ***Sample Master Log File***

```
--- Running on machine1(x86_64 w/Linux 2.6.18-308.el5) ---
This version was compiled on Sun Sep 1 14:53:30 PDT 2013.
INFO (DSTA-1025): Sourcing /icd/flow/ETS/ETS132/13.20-b059_1/lnx86/share/
tcltools/icd8.5.9/lib/tcl8.5/history.tcl
INFO (DSTA-1025): Sourcing ../blank/run_dsta.tcl
INFO (DSTA-1017): Starting 2 client processes.
INFO (DSTA-1013): Reading verilog file: ./zondammav10gs80.final.v
INFO (DSTA-1600): Loading library file: ./GS80_W_-40_0.92_0.92_CORE.lib.gz
INFO (DSTA-1025): Sourcing ./tim_ets_settings.tcl
INFO (DSTA-1193): Reading spef file: ./zondammav10gs80.spef.maxc_maxvia_-40.gz
INFO (DSTA-1421): After generating partitions: Cpu=00:01:43 Real=00:01:40
Peak_mem=4257meg Cur_mem=4257meg
INFO (DSTA-1020): Running command: update_timing -full
INFO (DSTA-1389): Cond arc const client 1 starts at 1378131471.
INFO (DSTA-1389): Cond arc const client 0 starts at 1378131522.
INFO (DSTA-1389): Cond arc const client 0 all done at 1378131523.
INFO (DSTA-1389): Cond arc const client 1 all done at 1378131523.
```

### ***Client Log***

The client log files are the same as the Tempus STA log files. Each client runs a copy of Tempus. The client machine has its own directory and log file, as shown below:

```
partOutput_0/tempus.log
partOutput_1/tempus.log
```

The log files will increment to `tempus.log1` in the next run.

**Note:** It is possible that you have client logs that are numbered in an unordered manner, as shown below:

```
partOutput_0/tempus.log12
partOutput_1/tempus.log6
```

This is caused due to multiple runs with different number of clients. It is recommended that you check the time stamp to ensure that the logs are for the specific master/client set.

### Sample Client Log File

```
--- Starting "Tempus Timing Signoff Solution v13.20-b059_1" on Mon Sep 2 12:38:07
2013 (mem=70.9M) ---
Server is up on machine1 (PID=830)
Multi-CPU acceleration using 4 CPU(s).

Scheduling LEF file(s) ./uc_CS402LND.lef to be loaded when the set_top_module
command is issued.
Scheduling LEF file(s) ./uc_CS402LZD.lef to be loaded when the set_top_module
command is issued.
Scheduling timing library file(s) ./LIB/scIMux_f_worst.lib to be loaded when the
set_top_module command is issued.
Scheduling timing library file(s) ./LIB/scxOpt_f_worst.lib to be loaded when the
set_top_module command is issued.

Set top cell to scTop.
** info: there are 406704 modules.
** info: there are 13891666 stdCell insts.
** info: there are 53 Pad insts.
** info: there are 293926 macros.

<CMD> report_timing
Analyzing view default_emulate_view with delay corner[0]
default_emulate_delay_corner, rc corner[0] ...
All-RC-Corners-Per-Net-In-Memory is turned ON...
Analyzing view default_emulate_view with delay corner[0]
default_emulate_delay_corner, rc corner[0] ...
Analyzing view default_emulate_view with delay corner[0]
default_emulate_delay_corner, rc corner[0] ...
```

## 7.6 DSTA Operational Tips

Although the operating environment of DSTA is similar to STA, you need to manage the new operational requirements in the distributed mode to maximize the DSTA performance. This section provides operating tips for the DSTA environment:

### Name the Master and Client Logs

#### Master

- To name the master log and overwrite the old log file, use the following command:  

```
tempus -distributed -overwrite -log tempus.log
```
- Consider using 4c8t to indicate 4 clients 8 threads, use:

## Tempus User Guide

### Distributed Static Timing Analysis

---

```
tempus -distributed -overwrite -log tempus_4c8t.log
```

#### **Client**

To avoid randomly numbered log file names, delete, or rename the old directories before the next run, use the following command:

```
rm -rf partOutput*
```

#### **Save the DSTA Logs**

Saving the master and client log files allows comparison of various client/thread configurations and scripts. To save logs, use the following commands:

```
mkdir saveLog1
cp -rfp tempus*.log pathOutput* saveLog1
```

#### **Master Log Completion**

Add a print message at the end of the script to confirm completion of DSTA.

```
Master run.tcl Script
. . .
. . .
puts "All done"
```

#### **Use a Single run.tcl for both STA and DSTA**

You can use a single `run.tcl` file for both STA and DSTA. This can ensure that there are no errors when DSTA is selected over STA. This can be done by wrapping the DSTA commands with a '`if`' statement, and allowing STA to skip these commands and DSTA to use them.

For example,

```
if {[info command distribute_partition] != "" } {
 puts "You are using Tempus DSTA"
 set_multi_cpu_usage -localCpu 4 -cpuPerRemoteHost 4 -remoteHost 2
} else {
 puts "You are using Tempus STA"
 set_multi_cpu_usage -localCpu 4
}
```

When you start Tempus, it automatically triggers the '`if`' statement.

## Configure the Script for both Batch and Interactive Modes

You can run a script in both the batch and interactive modes without changing the Tcl commands, by specifying the following in the `run.tcl` file:

```
Master run.tcl Script
. . .
return ; # Stops the script if it is in the interactive mode but does not exit
exit ; # Exits Tempus if it is in the batch mode
```

## Tracking Master Runtime

You can use the following command to track the start to finish runtime:

```
Initialize at the top of the master run.tcl
set initTimeTick [clock seconds]
. . .
. . .
. . .

Calculate runtime at the bottom of the master run.tcl
set lastTimeTick [clock seconds]
set totalTime [expr $lastTimeTick - $initTimeTick]
set totalMin [format "%4.1f" [expr $totalTime / 60.0]]
puts "The total walltime minutes was: $totalMin minutes"
```

## Tracking Master Memory

You can track the master current and peak memory using the following command:

[report\\_resource](#)

This command will print the resource usage for the master. The following is an example output:

```
Cpu=03:02:02 Real=02:31:31 Peak_mem=70502meg Cur_mem=70494meg
```

## Tracking Client Runtime and Memory

To print the resource usage for each client, you can use the following command:

[distribute print client usage](#)

This command does not interrupt or wait for whatever the client is currently doing.

A sample output of this command is given below:

```
INFO (DSTA-1039): Client resource usage:
```

```
Client 0: Cpu=00:44:25 Real=01:25:57 Peak_mem=11095meg Cur_mem=9816meg
Client 1: Cpu=00:44:29 Real=01:25:58 Peak_mem=11338meg Cur_mem=10072meg
```

```
Client 2: Cpu=00:44:25 Real=01:25:57 Peak_mem=11095meg Cur_mem=9816meg
Client 3: Cpu=00:44:29 Real=01:25:58 Peak_mem=11338meg Cur_mem=10072meg
```

## Monitoring tmp Disk Space

You can measure the /tmp free disk space by inserting the following code at the beginning and at the end of the script:

```
set dir /tmp
set a [lindex [lindex [split [exec df -k $dir] \n] end] end-2]
set a [expr {$a / 2.0**20}]
set localTmpGigs [format "%0.1f" $a]
puts "The /tmp disk has $localTmpGigs gigs free"
```

## 7.7 Distributed Static Timing Analysis Flow

The DSTA flow consists of the following steps:

- Define and start DSTA clients:

```
set_multi_cpu_usage ...
set_distribute_host ...
}
distribute_start_clients
```

- Load the design. The constraints must be read after the parasitics.

```
source load_libs.tcl
source load_netlist.tcl
source load_parasitics.tcl
```

- Initiate DSTA partitioning:

```
distribute_partition
```

- Load constraints and time the design:

```
source load_constraints.tcl
update_timing -full
distribute_print_client_usage
```

- Generate PBA reports:

```
set reportName rt_max_200K.rpt.gz
report_timing -late -max_paths 200000 -nworst 1 -path_type \
full_clock -net > $reportName
set reportName rt_max_pba_200K.rpt.gz
set maxPathCnt 200000
```

```
set nWorstCnt 1
report_timing -retiming path_slew_propagation -max_paths $maxPathCnt \
-nworst $nWorstCnt -path_type full_clock > $reportName
```

If a pre-defined sorting criteria is not defined, then the timing report output is displayed in any order.

**Note:** The path numbers that are assigned to each path in the timing report are not reported in DSTA output reports.

## 7.8 Additional Commands Supported in DSTA

The following commands are supported in both non-distributed Tempus and DSTA:

- [write\\_sdf](#)
- [write\\_twf](#)
- [read\\_design](#)
- [save\\_design](#)
- [report\\_disk](#)

The use models of `write_sdf` and `write_twf` commands are the same for both DSTA and non-distributed Tempus. The use models of these commands have been modified for DSTA, as given below:

- `save_design`: Allows you to save a snapshot of the design in the distributed timing analysis mode. All the design data, partitioning information, and client sessions are saved in the directory specified using the `save_design` command. The saved data size can be 2 times the size of the flat saved session due to partition overlap.
- The command usage of `save_design` is:

```
Usage: save_design [-help] <session> [-cellview {libname cellname
viewname}] [-noRC] [-overwrite]
```

Where `-cellview` and `-noRC` parameters are not supported in DSTA.

### Example

```
DSTA setup
set_multi_cpu_usage -localCpu 4 -remoteHost 2 -cpuPerRemoteHost 4
set_distribute_host -local -timeout 300
Start clients
```

```
distribute_start_clients
read_lib
read_verilog
set_top_module
read_spef/read_rcdb
distribute_partition
read_sdc
report_timing

Save current session
save_design <session> -overwrite
```

- **read\_design:** Enables you to restore the analyzed design in the distributed mode for further reporting (or analysis).

The command usage of `read_design` is:

```
Usage: read_design [-help] [-cellview {libname cellname viewname}] [-physical_data]
```

Where `-cellview` and `-physical_data` parameters are not supported in DSTA.

### **Example**

**## DSTA setup**

```
set_multi_cpu_usage -localCpu 4 -remoteHost 2 -cpuPerRemoteHost 4
set_distribute_host -local -timeout 300
```

**## restore saved session**

```
read_design distSave -cell <cellname>
```

**## Design is now in fully analyzed state**

The number of clients must be same for the saved and restored sessions. The number of master and client threads can be changed in the `read_design` from that used in the `save_design`.

- **report\_disk:** Measures the disk metrics of the current working directory and the /tmp directory. The output of this command allows you to investigate Mb/sec throughput of the disk and disk I/O requirements. This command is supported in both distributed (DSTA) and non-distributed Tempus.

The command usage of `report_disk` is:

```
report_disk [-help] [-dir directory_name]
```

### **Example**

```
dsta> report_disk
DISK_INFO : 206.66 Mb/S 381G total 1.1G used 359G free /tmp
{206.66 381G 1.1G 359G}
```

## Tempus User Guide

### Distributed Static Timing Analysis

---

```
dsta> distribute_start_clients -count 2
INFO (DSTA-1017): Starting 2 client processes.
Connected to sjfsb439 39262 1 (PID=31660)
Connected to sjfsb439 35186 0 (PID=31658)
INFO (DSTA-1019): Received application variables on master.
0

dsta> report_disk
DISK_INFO (Master) : 182.53 Mb/S 381G total 1.1G used 359G
free /tmp
DISK_INFO (Client 0) : 94.90 Mb/S 381G total 1.1G used 359G
free /tmp
DISK_INFO (Client 1) : 104.28 Mb/S 381G total 1.1G used 359G
free /tmp
{182.53 381G 1.1G 359G} {94.90 381G 1.1G 359G} {104.28 381G 1.1G 359G}
```

## Sample Scripts

### STA

The following is a sample STA Tcl script:

#### # Non-distributed script

```
puts "Starting script:

set_multi_cpu_usage -localCpu 4
read_verilog
read_lib
read_spef
read_sdc
update_timing
report_timing
```

### DSTA

```
set clientCnt 4
set threadCnt 2
set lsfQueue ssv
set hostGroup ssvpe
set maxMem 91000

set_multi_cpu_usage -cpuPerRemoteHost $threadCnt -localCpu \
$threadCnt -remoteHost $clientCnt

set_distribute_host -timeout 300 -lsf \
-queue ${lsfQueue} -args "-m ${hostGroup} -n ${threadCnt} \
-R \"(CPUS>=${threadCnt}), span\[hosts=1\] \
rusage\[mem=${maxMem}\]\\""
}

distribute_start_clients
```

```
source load_libs.tcl
source load_netlist.tcl
source load_parasitics.tcl
distribute_partition
source load_constraints.tcl
update_timing -full
distribute_print_client_usage
set reportName rt_max_200K.rpt.gz
report_timing -late -max_paths 200000 -nworst 1 -path_type \
full_clock -net > $reportName
set reportName rt_max_pba_200K.rpt.gz
set maxPathCnt 200000
set nWorstCnt 1
report_timing -retime path_slew_propagation -max_paths $maxPathCnt \
-nworst $nWorstCnt -path_type full_clock > $reportName
```

## 7.9 Commands Not Supported in DSTA Mode

The following commands are not supported in DSTA mode:

### System Commands - Basic Tool-Tcl Commands

- start\_gui
- stop\_gui

### General Commands

- change\_inst\_name
- get\_metric
- get\_preference
- save\_testcase
- set\_preference
- write\_flow\_template

### Design Import Commands

- free\_design
- read\_celtic\_script
- read\_def
- restore\_oa\_design
- save\_config

- set\_license\_check
- write\_def
- write\_rcdb

### **Multiple-CPU Processing Commands**

- check\_multi\_cpu\_usage

### **Distributed MMMC Commands**

- distribute\_command
- distribute\_design\_views
- distribute\_read\_design
- distribute\_views
- exit\_serversreset\_servers
- get\_distribute\_variables
- remove\_distribute\_view
- set\_distribute\_variables

### **Timing Constraint Commands**

- write\_load

### **Timing Analysis Commands**

- create\_boundary\_model
- create\_path\_scope
- display\_timing\_map
- get\_constant\_for\_timing
- get\_equivalent\_cells
- get\_propagated\_clock
- merge\_scope
- merge\_timing\_reports
- read\_boundary\_model
- read\_scope
- read\_scope\_model

- `read_scope_spef`
- `read_scope_verilog`
- `report_annotated_assertions`
- `report_cell_instance_timing`
- `report_clock_sense`
- `report_critical_instance`
- `report_design_rule`
- `report_freqViolation`
- `report_lib_cells`
- `report_pba_aocv_derate`
- `report_slack_histogram`
- `report_statistical_timing_derate_factors`
- `report_wire_load`
- `set_exclude_net`
- `write_annotated_transition`
- `write_loop_break_sdc`

### **Tcl Interface Commands**

- `get_activity`
- `get_power`

### **Signal Integrity Commands**

- `report_voltage_scaling`
- `report_voltage_swing`
- `write_noise_eco`

### **Timing Model Commands**

- `compare_model_timing`
- `do_extract_model`
- `merge_model_timing`
- `write_model_timing`

## **Interface Logic Models - ILM**

- import\_ilm\_data
- read\_ilm
- set\_ilm\_mode
- specify\_ilm
- unspecify\_ilm
- write\_ilm

## **Prototype Timing Model - PTM**

- create\_ptm\_constraint\_arc
- create\_ptm\_delay\_arc
- create\_ptm\_drive\_type
- create\_ptm\_generated\_clock
- create\_ptm\_load\_type
- create\_ptm\_model
- create\_ptm\_path\_type
- create\_ptm\_port
- report\_ptm\_model
- save\_ptm\_model
- set\_ptm\_design\_rule
- set\_ptm\_global\_parameter
- set\_ptm\_port\_drive
- set\_ptm\_port\_load

## **Timing Debug Commands**

- analyze\_paths\_by\_basic\_path\_group
- analyze\_paths\_by\_bottleneck
- analyze\_paths\_by\_clock\_domain
- analyze\_paths\_by\_critical\_false\_path
- analyze\_paths\_by\_drv
- analyze\_paths\_by\_hier\_port

- analyze\_paths\_by\_hierarchy
- analyze\_paths\_by\_view
- create\_path\_category
- delete\_path\_category
- dump\_histogram\_view
- highlight\_timing\_report
- load\_path\_categories
- load\_timing\_debug\_report
- save\_path\_categories
- write\_category\_summary
- write\_text\_timing\_report

### **Manual ECO Commands**

- attach\_module\_portconnect\_hpins
- delete\_net
- detach\_module\_port

### **Signoff ECO Commands**

- eco\_opt\_design
- get\_eco\_opt\_mode
- merge\_hierarchical\_def
- read\_partition
- set\_filler\_mode
- set\_place\_mode
- specifyCellEdgeSpacing
- specifyCellEdgeType

### **Low Power Commands**

- set\_msv\_mode

### **Clock Tree Debugger Commands**

- close\_ctd\_win

- ctd\_trace
- ctd\_win
- get\_ctd\_win\_id
- get\_ctd\_win\_title
- set\_ctd\_win\_title

### **RC Extraction Commands**

- extract\_rc
- get\_qrc\_tech\_file
- rc\_out
- report\_rcdb
- report\_unit\_parasitics
- set\_qrc\_tech\_file

### **Power Calculation Commands**

- dump\_unannotated\_nets
- map\_activity\_file
- propagate\_activity
- read\_activity\_file
- report\_instance\_power
- report\_power
- report\_vector\_profile
- reset\_power\_activity
- restore\_power\_database
- set\_default\_switching\_activity
- set\_dynamic\_power\_simulation
- set\_inst\_temperature\_file
- set\_power
- set\_power\_calc\_temperature
- set\_power\_include\_file
- set\_power\_output\_dir

- `set_twf_attribute`
- `set_virtual_clock_network_parameters`
- `write_power_constraints`
- `write_tcf`

### **Unsupported Parameters**

The `-tcl_list` parameter of all the `report_*` commands, is not supported in the DSTA mode.

---

# Signoff ECO

---

- 8.1 [Signoff ECO Overview](#) on page 67
- 8.2 [Key Features](#) on page 67
- 8.3 [Signoff ECO Flow](#) on page 69
- 8.4 [Setting Up Signoff ECO Environment](#) on page 70
  - [Starting ECO](#) on page 71
  - [Terminating ECO](#) on page 71
- 8.5 [Signoff Optimization Use Models](#) on page 71
  - [Generating ECO Timing DB and Running ECO Timing Closure in Separate Sessions](#) on page 71
  - [Running Signoff Optimization with ECO Timing Generated in Batch Mode Using DMMMC](#) on page 72
  - [Running Signoff Optimization with ECO Timing Generated Using CMMMC](#) on page 73
  - [Parallel Distributed Interactive MMMC Analysis and ECO \(PARADIME\) Flow](#) on page 73
  - [Running Signoff Timing/Power Optimization from Innovus](#) on page 80
- 8.6 [Basic Signoff ECO Optimization Techniques](#) on page 83
- 8.7 [Fixing SI Violations](#) on page 86
  - [SI Glitch Violations](#) on page 86
  - [SI Slew Violations](#) on page 87
  - [SI Crosstalk Delta Delay Violations](#) on page 87
- 8.8 [Fixing IR Drop in Tempus Signoff ECO](#) on page 89

## **Tempus User Guide**

### Signoff ECO

---

- 8.9 [Path-Based Analysis \(PBA\) Mode Optimization](#) on page 89
- 8.10 [Total Power Optimization](#) on page 90
- 8.11 [Setup Timing Recovery After Large Leakage or Total Power Optimization](#) on page 91
- 8.12 [Recommendations for Getting Best Total Power Optimization](#) on page 91
- 8.13 [Hierarchical ECO Optimization](#) on page 92
  - [Sample Script for Hierarchical Flow](#) on page 93
  - [Optimization of Master/Clone Designs](#) on page 94
- 8.14 [Top Down Block ECO Flow](#) on page 96
- 8.15 [Generating Node Locations in Parasitic Data](#) on page 98
  - [Using Innovus Engines](#) on page 98
  - [Using Standalone QRC](#) on page 98
  - [Using a Third-Party Tool](#) on page 98
- 8.16 [Optimization Along Wire Topologies](#) on page 99
- 8.17 [Optimization using Endpoint Control](#) on page 99
- 8.18 [Metal ECO Flow](#) on page 102
- 8.19 [Handling Large Number of Active Timing Views Using SmartMMMC](#) on page 104

## 8.1 Signoff ECO Overview

Tempus ECO is a licensed feature available in Tempus to address certain signoff concerns. Tempus ECO is easy to plug in in any Signoff analysis flow and allows the user to clean up the remaining timing violations as well as reducing the power consumption of the netlist. Also, this feature allows fixing timing violations on large designs (multi-million instances) with many analysis views. You can run the ECO feature in Tempus by invoking Tempus using the `-tso` option.

In Signoff environment, designers always find some remaining timing violations that must be cleaned up. These violations can be due to many reasons:

- Block-level versus top-level timing analysis do not match.
- Imprecise Timing budgeting, which has generated the setup/hold/drive/cap/glitch violations.
- Timing was not optimized across all the analysis views during implementation flow to avoid over-fixing, and the designer is relying on the Signoff stage to catch final violations. The final violations are identified at the Signoff stage for which the absolute number can be slightly different in the implementation flow.
- Often additional corners and/or modes are added at the time of signoff.
- There may be timing model differences.
- Implementation tools cannot optimize timing with full Signoff STA precision.

Instead of back-annotating the timing into the implementation tool, it is much easier to stay in the Signoff environment and perform various transforms to fix the remaining violations. In the Signoff stage, the netlist changes committed are written in an ECO file, which will be used in the implementation tool to perform incremental place and route.

In addition to timing closure, Tempus ECO is also a solution to reduce overall power consumption of the netlist. At Signoff stage, some amount of timing pessimism is removed due to Path Based Analysis (PBA), also called as retiming, and that extra timing margin can be used to perform netlist change leading to overall power reduction. This is especially beneficial when AOCV or SOCV timing modes are enabled.

## 8.2 Key Features

The key features of Tempus ECO are as follows:

- Focus on timing and power optimization while being MMMC-aware, STA Signoff aware, SI-aware, and Physical-aware (optional).

- Concurrent and distributed timing analysis. Highly scalable.
- Supports PBA timing and power optimization to benefit from reduced timing pessimism
- Targets minimal netlist change for timing closure.
- Local density and routing congestion will be considered when adding buffers in Physical-aware mode.
- Supports MSV designs and hierarchical based designs (including different row height).
- Supports replicated hierarchies (also named as Master/Clone optimization).
- Honors all modules/instances/cells/nets/clocks specific attributes.
- Requires only a single ECO loop to close timing.
- Supports large capacity in netlist size (>200M gates) and number of active views timed (>100).
- Supports multi-threaded ECO optimization, on-route buffering, clock skewing, and dummy load cell insertion.

## Usage

These features can be used in the scenarios mentioned below to fix timing violations:

### Full flat assembled design in hierarchical flow

The goal is to fix the inter-partition timing violations after assembling the design on full flat DB.

In this scenario, the timing violations to be fixed can be large but their number should not be big. This means that you can have some paths with negative slack because those are inter-partitions paths, which might not have optimized correctly earlier in the flow. But the number of such violations is usually not too much because there are not many such paths.

### Block implementation flow

At the block level, there may be remaining violations at the Signoff stage. This is because one methodology is to remove any pessimism in the timing constraints used by the implementation tool, and then rely on the Signoff STA tool to fix the real remaining violations.

In this scenario, since optimization is already performed once in the flow, thus the timing violations to fix are probably small.

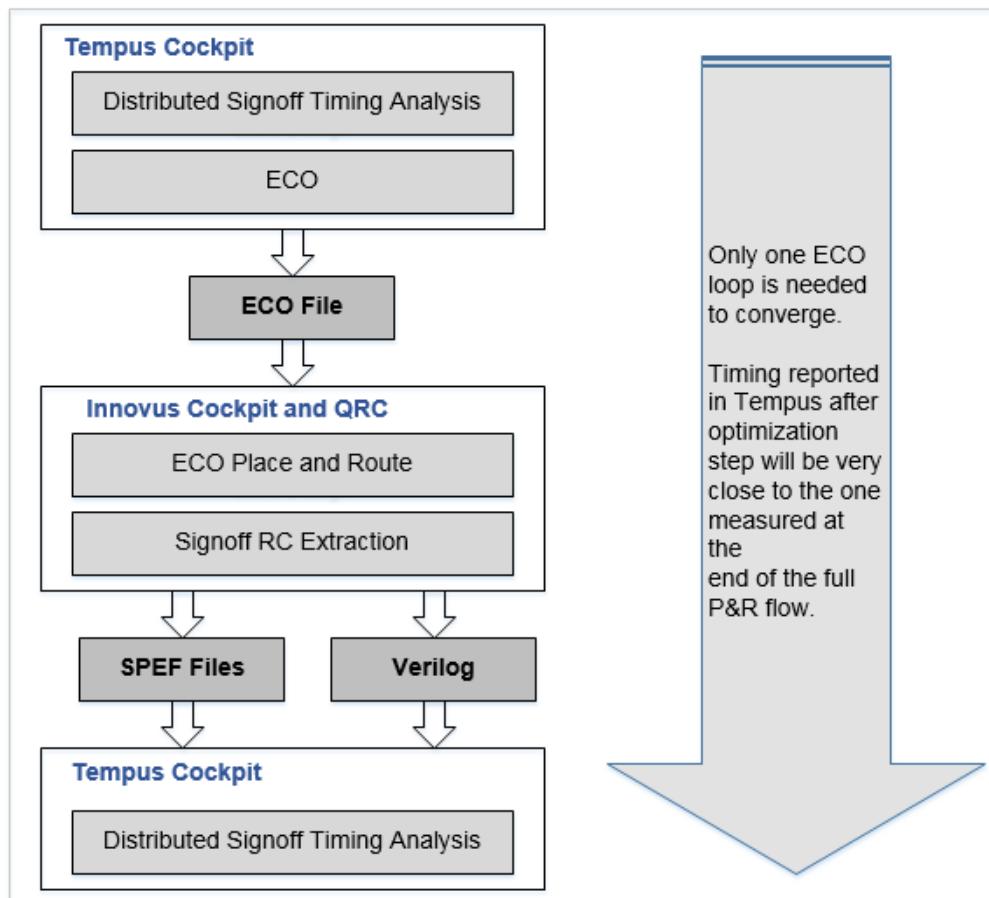
Here the challenge is to perform precise fixing across all analysis views without creating any new violations.

This is typically where Path Based Analysis mode would be applied in order to allow Power, Performance, Area push with Signoff timing analysis accuracy.

## 8.3 Signoff ECO Flow

The diagram below shows how Tempus ECO fits in the flow. The design is first loaded in Tempus for signoff timing analysis. To fix any remaining timing violations, *Tempus ECO* is run, which generates an ECO file. That is then taken to the implementation tool to implement the Tempus ECOs in the design. After the Tempus ECO is implemented, ECO place and route is done and then the design is re-extracted. The design or Verilog netlist after Tempus ECO implementation and the re-extracted SPEF files are then brought back into Tempus to run signoff timing analysis.

**Figure 8-1 ECO Flow**



## 8.4 Setting Up Signoff ECO Environment

This topic describes the tasks that you perform to set the ECO environment, and the commands to start and terminate ECO.

The input and output data, and various fixing methodologies and targets for setting up the ECO environment are as follows:

### **Input Data**

- Technology data (timing libraries)
- Design data (Verilog, MMMC, SPEF)
- Optional design data (DEF, CPF/UPF, ILM, and LEF)

### **Timing and Power Optimization Methodology and Targets**

- Uses cell swapping, cell resizing, and buffer/inverter insertion and deletion.
- Fixes timing violations, such as:
  - Hold timing
  - Setup timing
  - Design Rule (max\_cap/max\_tran)
  - SI violations (SI Slew, SI Xtalk and SI Glitch)
  - Reduces Power and Area:
    - Area reduction
    - Leakage power reduction
    - Dynamic power reduction
    - Leakage and dynamic power reduction concurrently

### **Output Data**

- Detailed reporting on all the ECOs being performed.
- Detailed diagnostic reports on the remaining violations that are not fixed.
- Standard format ECO file.
- Final timing summary reports.

## Starting ECO

To start a ECO session, type the following command with the appropriate parameters on the UNIX/Linux command line.

```
tempus > tempus -tso
```

## Terminating ECO

To end a ECO session, type "exit" in the Tempus shell.

## 8.5 Signoff Optimization Use Models

The following section describes the various signoff optimization use models:

- Generating ECO Timing DB and Running ECO Timing Closure in Separate Sessions
- Running Signoff Optimization with ECO timing generated in batch mode using DMMMC
- Running Signoff Optimization with ECO timing generated using CMMMC
- Parallel Distributed Interactive MMMC Analysis and ECO (PARADIME) Flow
- Running Signoff Optimization from Innovus

### Generating ECO Timing DB and Running ECO Timing Closure in Separate Sessions

ECO Timing DB is a lightweight timing graph built using the design input data. Since this timing graph is lightweight compared to the complete timing graph and stores only the data that is required for ECO fixing, it reduces the memory.

Within ECO fixing, Timing analysis is based on the hold and setup light-weight timing graph. Evaluation and commit time are also reduced when done on light-weight timing graphs. Therefore, it reduces both memory as well as turnaround time (TAT) significantly.

The write\_eco\_opt\_db command will save the ECO Timing DB after timing analysis, in non-distributed mode or in a distributed-MMMC session. In case several views are active at the same time, ECO Timing DB for each of them will be saved. The data will be saved in the directory pointed to by the set\_eco\_opt\_mode -save\_eco\_opt\_db option.

### Example

```
tempus> set_eco_opt_mode -save_eco_opt_db mySignOffTGDir
tempus> write_eco_opt_db
```

This will save the ECO Timing DB information in the mySignOffTGDDir directory.

The `write_eco_opt_db` command is also supported in the concurrent MMMC analysis mode. In addition, ECO Timing DB are saved in a directory that contains all active views' timing graph. So if the ECO Timing DBs are generated in a separate session, you can assemble them by simply copying into the same directory and use the following naming convention:

`<topModuleName>_<viewName>.tgd` and `<topModuleName>_<viewName>.ckd`

### Example

The below figure shows that in Session 1, the clients performing timing analysis in parallel can be launched on the local machine, but can also be distributed on LSF farm (`set_distribute_host -lsf`).

**Figure 8-2 ECO Timing Closure in Separate Sessions**

| <b>Session 1</b>                                                                                                                                                                                                                                                                                                      | <b>Session 2</b>                                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>ECO Timing DB Generation using DMMMC</b></p> <pre>set distribute host -local set_multi_cpu_usage -localcpu 8 -remoteHost 6 \     -cpuPerRemoteHost 8  distribute_read_design \     -design_script loadDesign.tcl -outdir .  distribute_views -views \$setup_and_hold_views \     -script spef_and_sta.tcl</pre> | <p><b>Timing Fix using Previously Generated ECO Timing DB</b></p> <pre>source loadDesign_physical.tcl  set_analysis_view -setup [list \$setup_views] \     -hold [list \$hold_views]  set_distribute_host -local set_multi_cpu_usage -localcpu 8  set_eco_opt_mode -load_eco_opt_db mysignoffTGDDir source spef.tcl eco_opt_design -hold</pre> |
| <p><b>Content Example for spef_and_sta.tcl</b></p> <pre>read_spef -rc_corner rc_max max.spef.gz read_spef -rc_corner rc_minl min1.spef.gz read_spef -rc_corner rc_min2 min2.spef.gz set_eco_opt_mode -save_eco_opt_db mysignoffTGDDir write_eco_opt_db</pre>                                                          | <p><b>Content Example for spef.tcl</b></p> <pre>read_spef -rc_corner rc_max max.spef .gz read_spef -rc_corner rc_minl min1.spef .gz read_spef -rc_corner rc_min2 min2.spef.gz</pre>                                                                                                                                                            |

### Running Signoff Optimization with ECO Timing Generated in Batch Mode Using DMMMC

When `eco_opt_design` is called without providing a pointer to ECO Timing DB database through `set_eco_opt_mode -load_eco_opt_db name`, it will automatically generate the ECO Timing DB in batch mode.

Example of Hold fixing with embedded Distributed timing analysis:

## Tempus User Guide

### Signoff ECO

---

```
read_lib $liberty
read lib -lef $lef
read_verilog $netlist
set_top_module my_top
source viewDefinition.tcl
read_def $def_file
source spef.tcl
set_distribute_host -local
set_multi_cpu_usage -localCpu 8 -remoteHost 2 -cpuPerRemoteHost 4
eco_opt_design -hold
```

**Note:** spef.tcl must contain the `read_spef` or `read_parasitics` commands so that every active `rc_corner` is parasitic annotated.

### Running Signoff Optimization with ECO Timing Generated Using CMMMC

For small to medium size designs where not many views are active (below 8), you can generate ECO Timing DB in CMMMC and perform ECO fixing in one single Tempus session,

Example of Hold fixing with ECO Timing DB generation and ECO fixing in one single Tempus CMMMC session:

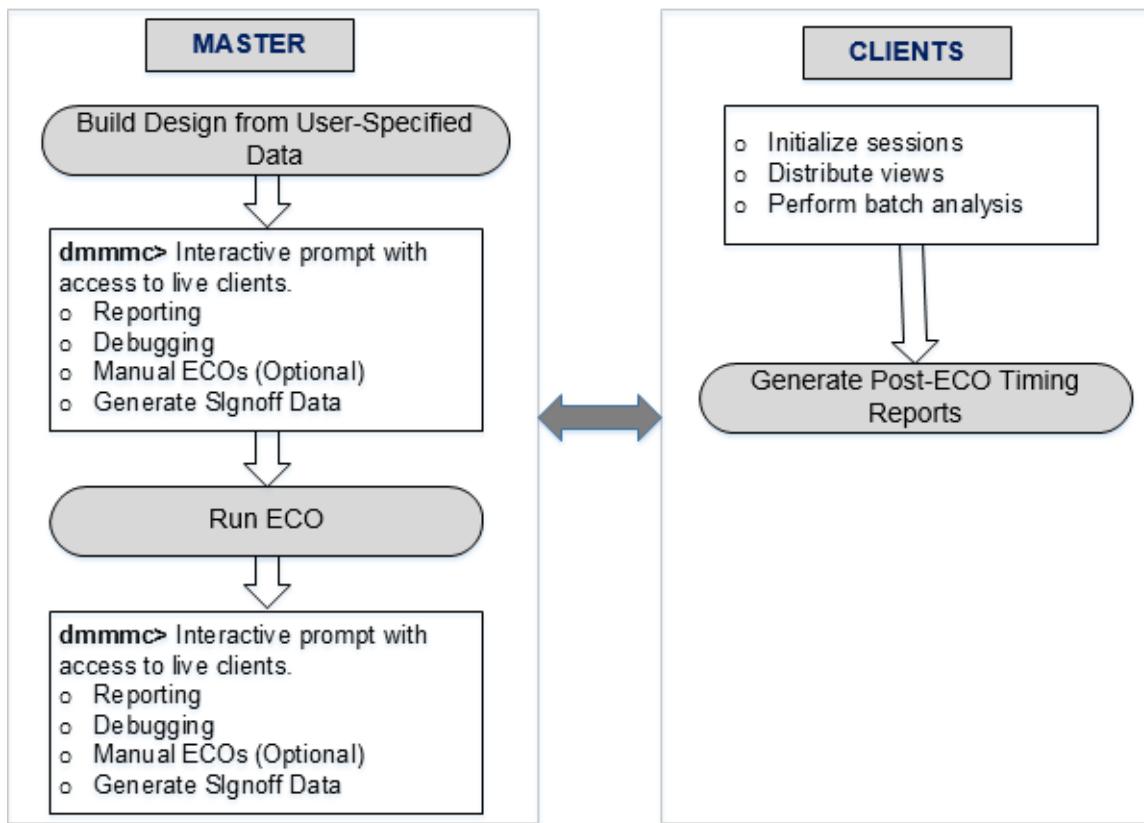
```
read_lib $liberty
read_lib -lef $lef
read_verilog $netlist
set_top_module my_top
source viewDefinition.tcl
read_def $def_file
source spef.tcl
set_distribute_host -local
set_multi_cpu_usage -localCpu 8
set_eco_opt_mode -save_eco_opt_db myEcoTimingDB
write_eco_opt_db
set_eco_opt_mode -load_eco_opt_db myEcoTimingDB
eco_opt_design -hold
```

**Note:** spef.tcl must contain the `read_spef` or `read_parasitics` commands so that every active `rc_corner` is parasitic annotated.

### Parallel Distributed Interactive MMMC Analysis and ECO (PARADIME) Flow

The PARADIME flow allow you to perform various tasks, such as distributed timing analysis, ECO, and any interactive debugging (such as, reporting, what-if analysis) in a single Tempus session leading to improved usability of the tool. The licensing requirements for this flow require Tempus to be invoked with the `-tso` license option for performing optimization in addition to the Tempus Distributed Multi-Mode Multi-Corner licensing requirements.

The following diagram shows the launch of the PARADIME flow, where Tempus master is used for ECO runs and clients are used for timing analysis runs.

**Figure 8-3 The PARADIME Flow**

### Enabling the PARADIME Flow

To enable this flow, you need to make the following setting in the script provided to the Tempus master.

```
set distribute mmmc mode -eco true
```

In this flow, the number of remoteHost should be equal to number of views.

### Running the PARADIME Flow

The Tempus master and clients load the design data for running either of the following flows:

- Full analysis
- Restored analysis

## Full Analysis Flow

To run the full analysis flow, you need to make the following settings:

```
distribute_read_design -design_script design_loading.tcl -outdir Output
distribute_views -script report.tcl -views {view1 view2}
```

Tempus master launches multiple DMMMC clients to load the design data as specified in `design_loading.tcl` and run `report.tcl` in order to perform full timing analysis runs and generate timing reports. Meanwhile, Tempus master also loads MMMC design for all active views of interest.

Once the design loading and timing analysis is done on the clients, the software is available with an interactive `dmmmc>` prompt with access to live clients. You can do the following:

- Perform merging of report after full timing analysis run or run manual commands to find out critical views of interest.
- Perform interactive querying and debugging on one, few, or all views of clients.

By default, all active views will be specified as setup and hold views for Tempus ECO run on master. If setup and hold views for Tempus ECO run are different, they can be specified using the variables below:

```
set eco_setup_view_list "<list of setup views>" # To specify setup views for
ECO run
set eco_hold_view_list "<list of hold views>" # To specify hold views for ECO
run
```

## Sample Script

```
To configure multiple CPUs
set distribute_host -local set_multi_cpu_usage -localCpu 8 -remoteHost 2 -
cpuPerRemoteHost 8
To enable the flow
set distribute_mmmc_mode -eco true
To specify different Tempus version for DMMMC client runs
set eco_client_tempus_executable "/icd/flows/142_tempus/bin/tempus"
Variables to specify list of setup and hold view for Tempus ECO (in case different
than the views restored on clients). By default, all clients' views will be in
setup/hold view list
set eco_setup_view_list "view1"
set eco_hold_view_list "view2"
Variables to specify physical data
set eco_lef_file_list "tech.lef design.lef"
set eco_def_file_list "design.def"
set data_dir /a/b/c
distribute_read_design -design_script design_loading.tcl -outdir
eco_restore_output_dmmc
distribute_views -script report.tcl -views {view1 view2}
report_timing > before_ECO.rpt
eco_opt_design -hold
report_timing > after_ECO.rpt
```

## Tempus User Guide

### Signoff ECO

---

The design\_loading.tcl script is the following:

```
read view definition <view definition file>
read verilog design.v
set top module
```

The report.tcl script is the following:

```
read_spef <all corner spef files>
update_timing -full
report* commands (optional)
```

### **Restored Analysis Flow**

In this flow, saved sessions from individual STA runs are restored. These individual SMSC STA runs have unique view names and are saved in MMMC configuration using a view definition file with a unique library set, delay-corner, rc-corner, constraint-mode, analysis-view, and so on. This flow uses the saved SMSC sessions as clients. This is done by setting up the compatible DMMMC sessions to restore the flow structure, as shown below:

```
set Views [list func_rmax func_rcmin test_rcmax]
distribute_read_design -restore_design $Views -restore_dir
<pathname_of_directory_of_saved_session> -outdir Output
```

Alternatively, a way to restore saved SMSC sessions is available if sessions are not saved in compatible DMMMC restore flow structure. You can provide a list of sessions' directories and corresponding view names as input in pairs as mentioned below:

```
set data_dir <unix path>
distribute_read_design -restore_design { {view1 $data_dir/view1/session.enc}
{view2 $data_dir/view2/session.enc} ...{viewN $data_dir/viewN/session.enc} } -
outdir Output
```

Same number of clients are required as the number of sessions to be restored. Each saved session can have data for one MMMC view.

Data from same restored sessions is loaded on the master for all views together. List of setup and hold views for Tempus ECO run can be specified using the variables below:

```
set eco_setup_view_list "<list of setup views>" # To specify setup views for Tempus
ECO run
set eco_hold_view_list "<list of hold views>" # To specify hold views for Tempus
ECO run
```

All data for master is picked from the saved sessions, including constraints and parasitic information. Only the libraries are picked from the UNIX path that are given in the view definition file because the library data is not saved in saved sessions. In the scenario where there is no common view definition file for restore sessions, you can auto-merge (or concatenate) all the view definition files, and create a single view definition file (which is the superset of all the individual sessions' view definition files) to load on the master.

If there is no common view definition file, you need to do the following settings:

```
set_distribute_mmmc_mode -merge_view_definition
```

## Sample Script

```
To configure multiple CPUs
set_distribute_host -local/-rsh/-lsf ... set_multi_cpu_usage -localCpu 8 -remoteHost
2 -cpuPerRemoteHost 8
To enable the flow
set_distribute_mmmc_mode -eco true
To specify different Tempus version for DMMMC client runs
set eco_client_tempus_executable "/icd/flows/142_tempus/bin/tempus" -
Variables to specify list of setup and hold view for Tempus ECO (in case different
than the views restored on clients). By default, all clients' views will be in
setup/hold view list.
set eco_setup_view_list "view1"

set eco_hold_view_list "view2"
Variables to specify physical data
set eco_lef_file_list "tech.lef design.lef"

set eco_def_file_list "design.def"

set data_dir /a/b/c
Restoring SMSC saved sessions (not saved in DMMMC compatible directory structure)
distribute read design -restore_design { {view1 $data_dir/setup_view1.enc}
{view2 $data_dir/hold_view2.enc} } -outdir eco_restore_output_dmmmc
 # Interactive DMMMC prompt pre-Tempus ECO
dmmmc> report timing > before_ECO.rpt
eco opt design -hold
Interactive DMMMC prompt post-Tempus ECO
dmmmc> report_timing > after_ECO.rpt
```

## Sample Script with Interactive Commands

```
To configure multiple CPUs
set_distribute_host -local/-rsh/-lsf ... set_multi_cpu_usage -localCpu 8 -remoteHost
2 -cpuPerRemoteHost 8
To enable the flow
set_distribute_mmmc_mode -eco true
To specify different Tempus version for DMMMC client runs
set eco_client_tempus_executable "/icd/flows/142_tempus/bin/tempus" -
Variables to specify list of setup and hold view for Tempus ECO (in case different
than the views restored on clients). By default, all clients' views will be in
setup/hold view list.
set eco_setup_view_list "view1"
set eco_hold_view_list "view2"
Variables to specify physical data
set eco_lef_file_list "tech.lef design.lef"
set eco_def_file_list "design.def"
set data_dir /a/b/c
Restoring SMSC saved sessions (not saved in DMMMC compatible directory structure)
```

## Tempus User Guide

### Signoff ECO

---

```
distribute_read_design -restore_design { view1 view2 } -restore_dir $data_dir -
outdir eco_restore output dmmmc
Interactive DMMMC prompt pre-Tempus ECO
dmmmc> report_timing > before_ECO.rpt
Perform some manual ECOs
dmmmc> change_cell inst1 -upsizer
dmmmc> add_repeater -term inst2/Y -cell BUFX6
dmmmc> set_eco_opt_mode -add_inst false
eco_opt_design -hold
Interactive DMMMC prompt pre-Tempus ECO
dmmmc> report_timing > after_hold_ECO.rpt
eco_opt_design -setup
```

### ***Pre-ECO Interactive Shell***

After the design loading and analysis (relevant only in full analysis flow) is done on clients, the interactive prompt is available to user. You can do report merge after full timing analysis run or run manual commands to find out critical views of interest. Any pre-requisite commands/setting for Tempus ECO run should be applied by this stage.

Interactive querying and debugging can be done on any number (one, few or all) of clients. To select few views, use:

```
distribute_command -views { <list of views to select>} { <list of commands>}
```

If any command (s) are to run on all views, use:

```
distribute_command { <list of commands>}
```

Manual ECOs (if any) must be done on Tempus master directly without distribute\_command. Netlist update is done for all clients and master database. Tempus software detects if manual ECOs are performed using distribute\_command and errors out.

### ***Running ECO***

When you run ECO file written by Tempus ECO, it is sent automatically to clients for post ECO signoff timing report generation. On completion of the Tempus ECO run, the master will return to interactive prompt with access to Tempus clients for signoff timing report generation and manual what-if analysis. You can also do successive Tempus ECO runs.

Note that running Tempus ECO as part of the paradigm flow is different from running normal ECO (when invoked from outside this flow) where all the set\_eco\_opt\_mode parameters are available in this flow except for the parameters below:

- -load\_eco\_opt\_db: ECO cannot accept pre-generated ECO Timing DB from outside.
- -save\_eco\_opt\_db: This parameter is ignored and ECO Timing DBs are stored in DMMMC output directory.

Post ECO, you can generate sign-off reports, debugging, perform manual what-if analysis and so on.

#### ***Post-ECO Interactive Shell***

The interactive prompt after ECO is available to the user. It can be used for generating sign-off reports, debugging, doing manual what-if analysis, and so on. You can also run another Tempus ECO run at this stage.

#### ***Use Model***

- Master does not have any timing information available. There is check in software to ensure that timer update is never triggered in master session.
- All timing related queries are to be done from clients using two ways
  - Use `distribute_command` in non-interactive mode. You can also make selected views active in this mode.  
`distribute_command -views { <list of views | all}`
  - Use `set_distribute_mmmc_mode` in interactive mode. You can also make selected views active in this mode.  
`set_distribute_mmmc_mode [-interactive {true | false} [-views {list_of_views | all}]]`
- If any command needs to apply to both master and clients, it needs to be specified for both master and clients using separate commands. For example, if user needs to specify timing derates, it needs to be done for master as well as clients
  - For master: Use `set_timing_derate...`
  - For clients: Use `distribute_command {set_timing_derate ...}`

Any settings/commands, which affect timing analysis should be applied to clients using  
`distribute_command { list of commands}`

Use the following command to assign variable value from master to clients

```
set_distribute_variables "list_of_variable_names" [-view <viewName>]
```

Different Tempus version can be specified for client runs. This is useful in scenarios where timing analysis and ECO need to be run with different Tempus versions.

```
set eco_client_tempus_executable "tempus executable path"
```

## Running Signoff Timing/Power Optimization from Innovus

Signoff Timing Optimization feature allows you to run timing and power optimization within Innovus on Signoff parasitic from Quantus QRC and Signoff timing from Tempus. This feature gives a complete automated solution for using the entire signoff tools through one high-level super command.

In a good timing closure methodology, at the implementation stage the timing targets that are set by the signoff Static Timing Analysis (STA) tool should be met. In order to ensure that the design state is close to sign-off quality, the timing reported by the implementation tool must correlate as much as possible to the signoff STA tool. From that stage, signoff timing optimization provides the following features:

- Allows you to run timing and power optimization on signoff timing within Innovus.
- Maximizes the usage of Cadence tools - Innovus will enable you to run extraction (Quantis QRC) and Tempus without extra effort.
- To provide signoff timing report in Innovus using Tempus, you can use the `signoffTimeDesign` command of Innovus. This command uses Quantus QRC and Tempus in standalone mode to perform signoff STA using the DMMC infrastructure and saves an ECO Timing DB per view. This signoff timing can be optimized using `signoffOptDesign`. Similarly, the `signoffOptDesign` command can be used to perform power optimization on this signoff timing.

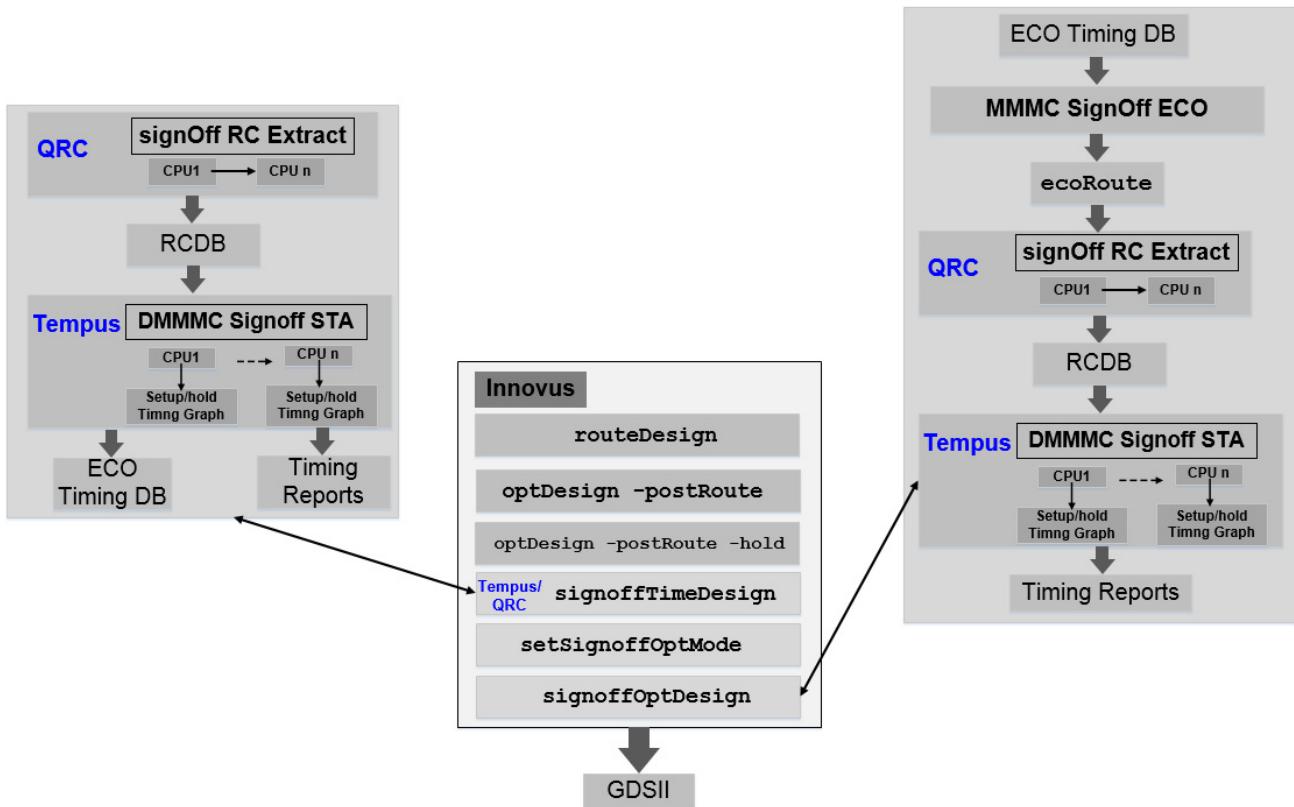
The following diagram illustrates the flow and the architecture of each signoff command:

# Tempus User Guide

## Signoff ECO

---

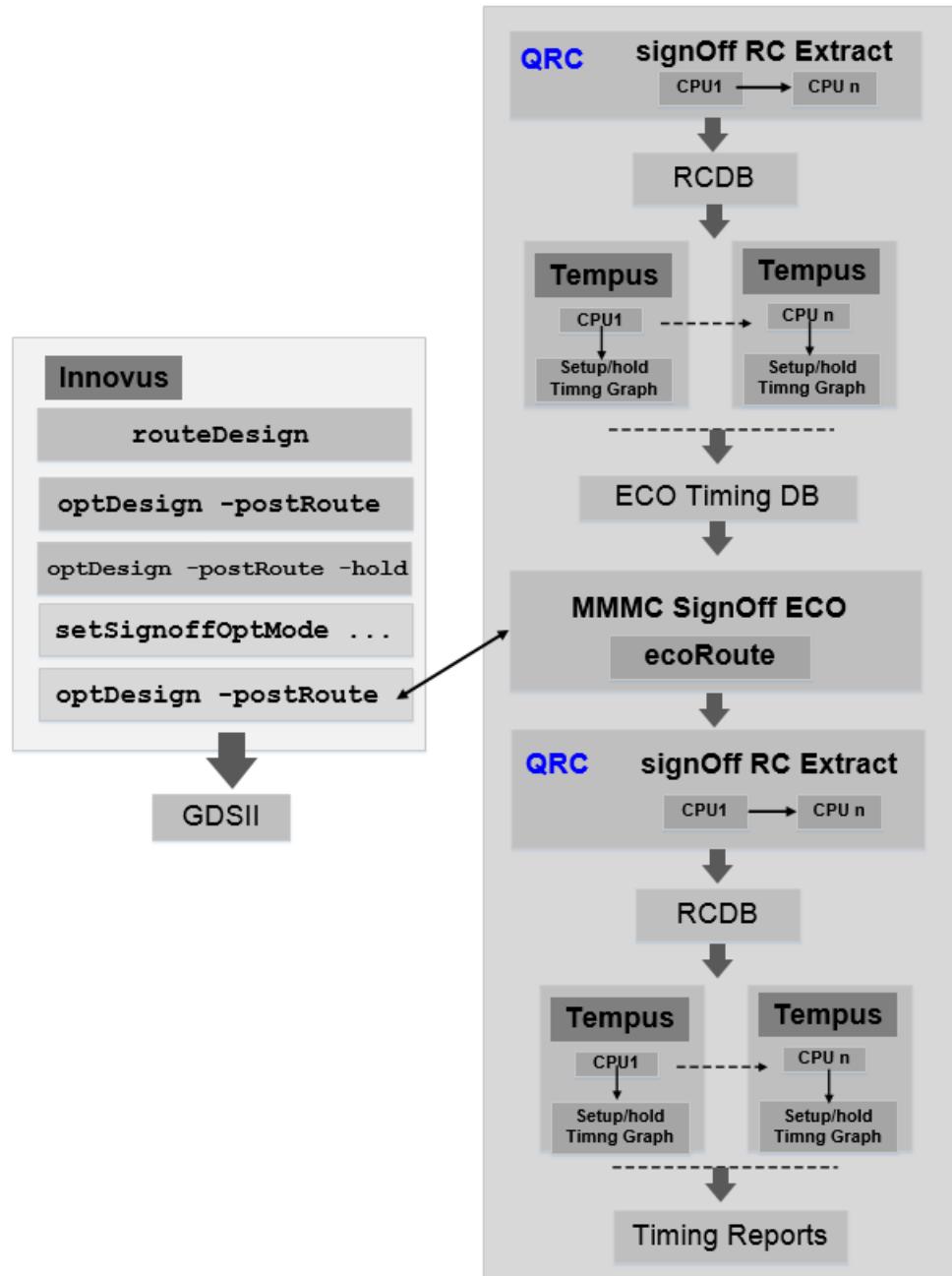
**Figure 8-4 Signoff Timing Optimization Flow**



## Tempus User Guide

### Signoff ECO

Figure 8-5 Signoff Timing Optimization Flow



Signoff Timing Optimization provides a flexible flow to accommodate any specific methodology:

If parasitics should be extracted using TQuantus or IQuantus, this can be done manually by you. Then `signoffTimeDesign` can be used with the `-reportOnly` option to reuse those parasitics and skip the Quantus QRC call.

In case specific steps/options should be performed before/during Tempus ECO routing, this step can be performed manually after running `signoffOptDesign` with the `-noEcoRoute` option.

When specific signoff STA commands/options/global variables should be applied, you can set these in a file and pass it to Tempus through `setSignoffOptMode -preStaTcl FILE` parameter.

The syntax below is the basic template flow when user runs parasitic extraction and ecoRoute manually:

```
setMultiCpuUsage -localCpu 16 -remoteHost 4 -cpuPerRemoteHost 4

setExtractRCMode -effortLevel high
extractRC
signoffTimeDesign -reportOnly -outDir RPT_initial
signoffOptDesign -hold -noEcoRoute
ecoRoute

extractRC
signoffTimeDesign -reportOnly -outDir RPT_final
```

## 8.6 Basic Signoff ECO Optimization Techniques

Tempus ECO is able to optimize timing/DRV/Leakage/Area/Hold/Power/Setup. The main command to perform those different optimizations is `eco_opt_design` and you can select the targeted optimization using the following options:

```
eco_opt_design [-area] [-drv] [-dynamic] [-hold] [-leakage] [-power] [-setup]
```

This command will generate the `eco_innovus.tcl` file, which should be used in Innovus to perform place and route. It also generates the `eco_tempus.tcl` file, which can be sourced in Tempus to perform timing analysis after ECOs. Each optimization engine always ensures no new violations. For example, when fixing Setup timing, the tool ensures no new DRV/Hold violations, or when fixing Hold timing, the tool ensures no new DRV/Setup violations. This is the key to ensure good timing convergence in a minimum of ECO loops.

Notes before running `eco_opt_design`:

- It is mandatory to provide parasitic before running `eco_opt_design`. The syntax below shows how to load the SPEF files for every active RC corner:

```
read_spef -rc_corner rc_max max.spef.gz
read_spef -rc_corner rc_min1 min1.spef.gz
read_spef -rc_corner rc_min2 min2.spef.gz
```

To perform SI analysis/fixing, SPEF must contain the coupling capacitance data.

- All timing derates, AOCV tables, delay calculation settings, CTE global variables must also be provided before running the `eco_opt_design` since those will be used during ECO fixing.
- It is important to correctly set up the timing environment and multi-CPU settings before running `eco_opt_design` to avail the benefit of DMMMC timing analysis and Multi-threading during timing/leakage/area optimization.
- In case there are filler cells in the design, `eco_opt_design` will automatically delete them at the start. To implement this, it is mandatory to specify the filler cells that need to be removed by using the command, `set_filler_mode -core {{list-of-cells1} {list-of-cells2} ...}`.

By default, `eco_opt_design` will use buffering/resizing/swapping techniques to improve timing/DRC/Leakage/Area. For Setup timing closure, `eco_opt_design` is also able to insert pairs of inverters. Additionally, sequential elements can also be sized by enabling `set_eco_opt_mode -optimize_sequential_cells true` and buffer deletion can be enabled during area recovery by enabling `set_eco_opt_mode -delete_inst true`.

## Examples

- Example 1:

```
eco_opt_design -hold
```

This will perform Hold fixing by using Vth swapping, buffering, and resizing techniques on the combinational cells.

- Example 2:

```
set_eco_opt_mode -optimize_sequential_cells true
eco_opt_design -leakage
```

This will perform Leakage recovery by using Vth swapping technique on both the combinational and sequential cells.

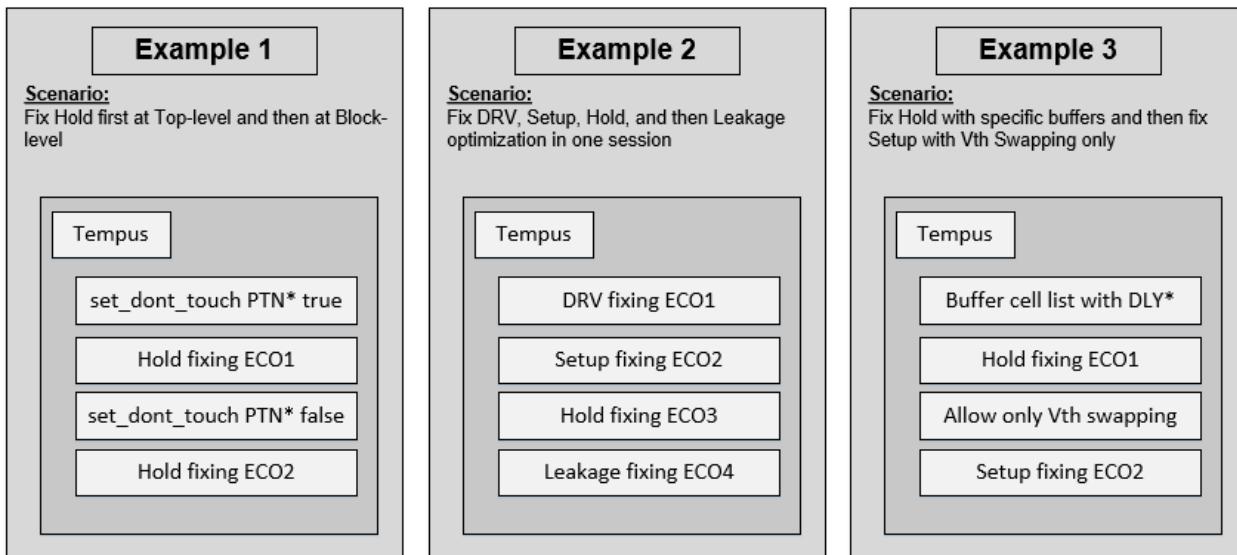
- Example 3:

```
set_eco_opt_mode -delete_inst true
eco_opt_design -area
```

This will perform Area recovery by using downsizing on both the combination and sequential cells in addition to buffer removal.

At the signoff stage, you can optimize all remaining violations, such as DRV, Setup, and Hold, but also to recover Power or Area when needed. This can be achieved in one single Tempus session using the incremental optimization feature enabled with `set_eco_opt_mode -allow_multiple_incremental true`.

**Figure 8-6 Examples**



Notes about incremental optimization are as follows:

The recommended flow is the one described in Example 2 above. In case, area optimization is needed, it should be called as the first flow step.

Note that if more Tempus ECOs are done in a session then more routing changes will be needed when implementing those ECOs and therefore larger is the risk to get timing degradation.

Each optimization step will output an ECO file so in order to avoid overwriting the previous one, you should apply a unique prefix for each with the `set_eco_opt_mode -eco_file_prefix prefixName` option.

### Example

```

set_eco_opt_mode -load_eco_opt_db ECOdb
set_eco_opt_mode -allow_multiple_incremental true
set_eco_opt_mode -eco_file_prefix DRV_FIX
eco_opt_design -drv
set_eco_opt_mode -eco_file_prefix HOLD_FIX
eco_opt_design -hold

```

This will perform DRV fixing followed by Hold fixing, while giving a different prefix for the ECO files.

## 8.7 Fixing SI Violations

Tempus ECO is able to fix different violations related to SI glitch, SI slew, and SI crosstalk delta delay.

### SI Glitch Violations

A signal integrity noise glitch, also called as voltage bump, generated by crosstalk coupling can propagate and amplify while traveling along a path. As a result, this glitch can cause incorrect signal state change. Tempus provides the ability to fix SI glitch violations in Tempus ECO. This is done using the `-fix_glitch` parameter of the `set_eco_opt_mode` command.

#### Syntax:

```
set_eco_opt_mode -fix_glitch true|false
```

When set to `true`, the software performs SI glitch fixing during DRV fixing using resizing and buffering techniques.

The default value is `false`.

#### Note:

- This parameter must also be set during ECO DB generation.
- The glitch fixing is done before `max_tran/max_cap` fixing.
- It is mandatory to set `set_si_mode -enable_glitch_report true` and use the `report_noise` command to analyse and get a signoff glitch report.
- In case remaining SI glitch could not be fixed using resizing and buffering techniques, it is recommended to select those nets and re-route them with extra SI property.

#### Example

The below example shows fixing Si glitch violations only:

```
set_eco_opt_mode -fix_max_cap false -fix_max_tran false -fix_glitch true
The below example shows fixing of SI glitch violations using DRV fixing in addition to regular
max_tran/max_cap violations:
```

```
set_eco_opt_mode -fix_glitch true
eco_opt_design -drv
```

## SI Slew Violations

Crosstalk effects caused by parasitic capacitance between adjacent nets can lead to a larger signal transition (SI slews) on the victim nets. Tempus provides the ability to consider SI slews apart from base slews while performing transition violation fixing during DRV optimizations. All other optimizers that include setup, hold, leakage, area, power, and dynamic do not consider SI slews. You must run this as the first step of optimization before proceeding for any other incremental optimization.

This is done using the `-fix_si_slew` parameter of the `set_eco_opt_mode` command.

### Syntax

```
set_eco_opt_mode -fix_si_slew true|false
```

When this parameter is set to `true`, the tool checks `max_tran` rule against SI slew and such violations will be resolved during DRV fixing using resizing and buffering techniques.

The default value is `false`.

**Note:** This parameter must also be set during ECO DB generation in addition to `set_si_mode -enable_drv_with_delta_slew true`.

To analyse and report SI slew violations:

```
set_global_timing_use_incremental_si_transition true
set_si_mode -enable_drv_with_delta_slew true
report_constraint -drvViolation_type max_transition -all_violators
```

### Examples:

The below example shows fixing SI slews violations only:

```
set_eco_opt_mode -fix_max_cap false -fix_si_slew true
```

To analyse and report SI slew violations:

```
set_global_timing_use_incremental_si_transition true
set_si_mode -enable_drv_with_delta_slew true
report_constraint -drvViolation_type max_transition -all_violators
```

## SI Crosstalk Delta Delay Violations

In Tempus when attackers are transitioning opposite the victim, it causes the arrival time to increase (positive delta delay). And, when attackers are transitioning at the same time as the victim, it causes the arrival time to reduce (negative delta delay).

Tempus provides the ability to fix crosstalk delta delay in Tempus ECO. This helps in improving the design robustness, SI-delay fixing convergence, and minimizing the Setup versus Hold timing conflicts on critical paths.

This is done using the `-fix_xtalk` parameter of the `set_eco_opt_modeset_db` command.

**Syntax:**

```
set_eco_opt_mode -fix_xtalk true|false
```

When set to `true`, the software reduces the crosstalk on the net that violates the thresholds set by you.

The default value is `false`.

**Note:**

- This parameter must also be set during ECO DB generation.
- xtalk delta delay fixing is done after `max_tran/max_cap` fixing as a separate phase.
- It is mandatory to set the below SI mode options:  
`set_si_mode -separate_delta_delay_on_data true -delta_delay_annotation_mode lumpedOnNet`

In addition, the following parameters of the `set_eco_opt_mode` command are used to select nets for xtalk fixing during optimization:

- To fix the nets with the crosstalk delta delay value equal to or greater than a specified threshold value:
  - For setup views: `set_eco_opt_mode -setup_xtalk_delta_threshold <value in ns>` (default 0.3 ns)
  - For hold views: `set_eco_opt_mode -hold_xtalk_delta_threshold <value in ns>` (default 0.3 ns)
- To fix the nets with the slack threshold value equal to or less than a specified threshold value:

For setup views:

```
set_eco_opt_mode -setup_xtalk_slack_threshold <value in ns> (default 1000 ns)
```

For hold views:

```
set_eco_opt_mode -hold_xtalk_slack_threshold <value in ns> (default 1000 ns)
```

**Example**

The following example uses DRV fixing to reduce the crosstalk delay on all nets that violated the 300ps threshold rule in setup views:

```
set_eco_opt_mode -setup_xtalk_delta_threshold 0.3
set_eco_opt_mode -fix_xtalk true
eco_opt_design -drv
```

## 8.8 Fixing IR Drop in Tempus Signoff ECO

Tempus ECO provides the ability to fix IR drop voltages for all instances in the design. The IR drop files that are generated through rail analysis in Voltus include victim and attacker data within hotspots and IR drop voltages. The IR drop hotspot information includes victim instance names, voltage drop, and the current drawn by each of the instances. These IR drop files are then read into the Tempus-specified directory.

The software picks the worst victim and its attackers from ECO-DB. If the victim is violating then the attackers with positive slack are identified and downsized to avoid any new timing violations. Else, non-violating victims are ignored.

To fix IR drop in Tempus ECO, use the `-fix_ir_drop` and `-load_irdrop_db` parameters of the `set_eco_opt_mode` command. When enabling the `-fix_ir_drop` parameter along with providing an IR Drop DB using the `-load_irdrop_db` parameter, `eco_opt_design -drv` performs an IR drop fixing by downsizing instances without creating any Setup/Hold/DRV violations.

For more information, refer to the “[Timing-aware IR Drop Fixing](#)” section in the *Voltus User Guide*.

## 8.9 Path-Based Analysis (PBA) Mode Optimization

At implementation stage, tools are using the Graph Based Analysis (GBA) mode because that allows fast turnaround time to analyze timing and to update timing incrementally. The drawback is that the GBA mode is generating pessimistic timing. At the signoff stage, where perfect accuracy is needed, you can enable the PBA mode and perform final pass of timing closure. In addition, it is recommended to run the area or power recovery engine to reach the best possible Power Performance Area (PPA) metrics. When enabling PBA, each timing path is timed in its own context so that the slews or AOCV factors are computed based only on the path being timed.

The following five options are specific to the PBA mode for Tempus ECO:

To enable PBA and select the retiming mode:

```
set eco opt mode -retimed #default "none"
```

To select whether PBA is applied to Setup or Hold or both:

```
-check_type #default "early"
```

To specify the maximum slack to be considered for retiming:

```
-max_slack #default 0
```

To specify the maximum number of paths to be retimed in total:

```
-max_paths #default -1
```

To specify the number of paths to be retimed for each endpoint:

```
-nworst #default -1
```

The following is an example of SOCV PBA based Leakage optimization:

```
read design -physical_data postroute.enc.dat topChip
<set all signoff STA views>
<apply signoff STA settings/options/globals >
<load parasitic>
set multi cpu usage -localCpu 16
set eco opt mode -retimed path_slew_propagation
set_eco_opt_mode -check_type both
set_eco_opt_mode -max_slack 10
set_eco_opt_mode -max_paths 5000000
set_eco_opt_mode -nworst 50
set_eco_opt_mode -pba_effort high
set_eco_opt_mode -save_eco_opt_db ECO-DB-PBA
write_eco_opt_db
set_eco_opt_mode -load_eco_opt_db ECO-DB-PBA
eco opt design -leakage
```

**Note:** In above example, the `-max_slack` option is set to 10 (DB timing unit) so that positive GBA slack paths are also getting retimed in order to get even more positive setup slack on those paths. That extra positive slack can be then used to perform even more power recovery.

## 8.10 Total Power Optimization

ECO performs total power optimization to reclaim total power in the design. This is done using the `-power` parameter of the `eco opt design` command. Using this parameter, the tool replaces the need to run the leakage and dynamic power optimizations in consecutive steps. During Total Power optimization, the engine will concurrently minimize leakage, switching, and internal power in order to achieve the lowest overall power consumption.

During total power optimization, `report_power` needs to be run before `eco_opt_design`. The netlist activity can either be computed using default toggling rate on the inputs or input from an external VCD/SAIF file. In addition, the software performs swapping and sizing on both the combinational and sequential cells, and also does buffer removal when enabled.

**Example:**

```
report_power
set_eco_opt_mode -delete_inst true -optimize_sequential_cells true
eco_opt_design -power
```

In order to allow sequential elements to be resized during power optimization, ensure that there is no Fixed attribute applied on them. If you measure Setup timing degradation after doing a Power optimization, see section *Setup Timing Recovery After a Large Leakage or Power Optimization*.

## 8.11 Setup Timing Recovery After Large Leakage or Total Power Optimization

It is quite common to get large number of ECOs generated when doing Power optimization. This means that 30 percent to 80 percent of the netlist has been changed. Although the tool is doing an accurate timing estimation for each of the ECOs generated; however, it will not be able to guarantee zero impact on Setup timing. In order to recover the Setup timing up to an initial value, a few ECOs might be needed having low or no power cost. This Setup timing recovery can be achieved by running a new signoff timing analysis and then calling Setup optimization with `set_eco_opt_mode -setupRecovery true`. The recovery will be done with Vth swapping only, which means that same size and same pin geometry cell sizing. Because of this there is no need to re-route the design and final timing can be generated.

## 8.12 Recommendations for Getting Best Total Power Optimization

For Total Power optimization, here are the main recommendations to achieve the best QOR:

- Timing analysis must be done in PBA mode.
- Re-timing in PBA mode must be done on positive GBA slack paths.
- The `set_eco_opt_mode -pba_effort` parameter must be set to `high`.
- Ensure that the list of usable cells is as large/rich as possible, and remove any Fixed placement attribute on the sequential elements to allow their resizing.

- Verify that all the sequential elements are not Fixed, otherwise they will not be the candidates for resizing.
- When doing aggressive Power optimization, a large amount of netlist changes is generated and the timing histogram attains a huge peak at around 0ns slack.

In this context, the software cannot completely avoid Setup timing degradation and a light Setup recovery based on the Vth cell swapping will help to get back to the initial timing. The solution is to perform a setup timing recovery optimization after generating a fresh signoff timing post-power optimization.

### **Template Script for Total Power Optimization in Tempus Cockpit**

```
read_design -physical_data postroute.enc.dat top
<set all signoff STA views>
<apply signoff STA settings/options/globals >
<load parasitics>
set_multi_cpu_usage -localCpu 16
set_eco_opt_mode -allow_multiple_incremental true
set_eco_opt_mode -retime path_slew_propagation \
-check_type both -pba_effort high \
-max_slack 10 -max_paths 10000000 -nworst 50 \
-delete_inst true \
-save_eco_opt_db ECO-DB-PBA
write_eco_opt_db
report_power
eco_opt_design -power
set_eco_opt_mode -max_slack 0 -max_paths 10000000 -nworst 50 \
-save_eco_opt_db ECO-DB-PBA2

write_eco_opt_db
set_eco_opt_mode -load_eco_opt_db ECO-DB-PBA2 -setup_recovery true
eco_opt_design -setup
update_timing -full
```

## **8.13 Hierarchical ECO Optimization**

Very large netlists are usually not implemented flat since the memory usage and runtime would be too large to be productive. In such a case, you apply a hierarchical flow methodology, either bottom-up or top-down, where the top-level netlist and the block-level netlist are implemented in parallel. Once those independent netlists are implemented, you have to assemble the design to perform signoff timing analysis in full flat mode, meaning that the entire design is timed. Most of the time this full flat signoff timing analysis will reveal timing violations on the timing path between top-level and partitions, or even only between partitions. This can be explained by the fact that the input/output timing constraints at the block level never models the required time from the top level perfectly, but also because at the top level, each partition is represented by a timing model, which is also never fully accurate. In order to fix those timing violations, the solution is to run the Tempus ECO feature to perform hierarchical chip finishing.

## Tempus User Guide

### Signoff ECO

---

On a hierarchical design, you have the flexibility to optimize timing only at the top level, or on the top-level and block-level netlists. You can apply the `dont_touch` attribute on the modules that should not be touched by ECO fixing. The output of the `eco_opt_design` command is an ECO file for the top level and for each identified partition, so that you can go back to implement those ECOs in each of the respective netlists.

On a hierarchical design, the buffering happens in logical and physical hierarchy-aware mode. This also implies that the partitions' interface remain unchanged.

In order to perform hierarchical chip finishing in Tempus, there are two main phases that need to be correctly set up:

- Assembling the design

First, the top-level and all block-level verilog must be loaded in Tempus. Then, the `merge_hierarchical_def` command will merge the top-level DEF and all the block-level DEF files. Next, the `read_spef` command loads the top-level SPEF and all block-level DEFs for every corner.

- Specifying which modules should be treated as partitions

In order for the tool to know which modules should be treated as partitions, you have to provide the list of those module cells through an external file. This is done using `set_eco_opt_mode -partition_list_file file`.

### Sample Script for Hierarchical Flow

```
read_lib $liberty
read_view_definition viewDefinition.tcl
read_verilog "my_top.v pnt1.v pnt2.v"
set_top_module my_top
source timing_settings_and_derating.tcl
merge_hierarchical_def "my_top.def pnt1.def pnt2.def"

read_spef -rc_corner rc_max "my_top_max.spef ptn1_max.spef ptn2_max.spef"
read_spef -rc_corner rc_min1 "my_top_min1.spef ptn1_min1.spef ptn2_min1.spef"
read_spef -rc_corner rc_min2 "my_top_min2.spef ptn1_min2.spef ptn2_min2.spef"
set_eco_opt_mode -buffer_cell_list "BUFX1 BUFX2 BUFX8 DLYX1 DLYX4"
set_eco_opt_mode -load_eco_opt_db myECODB
set_eco_opt_mode -partition_list_file partition.txt
eco_opt_design -hold
```

**Note:** You must provide the pointer to the ECO Timing database using `set_eco_opt_mode -load_eco_opt_db` since `eco_opt_design` cannot do it in batch mode for hierarchical designs.

## Optimization of Master/Clone Designs

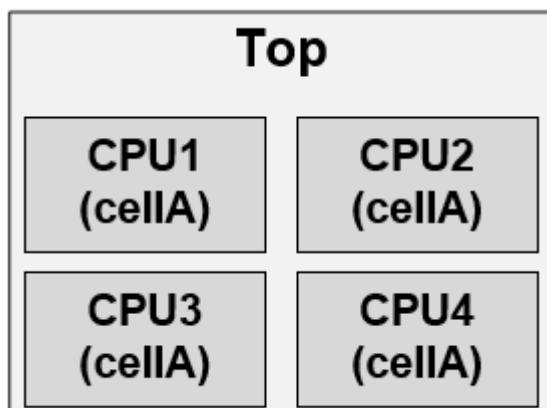
The intent of the Master/Clone design methodology is to build common blocks once and instantiate them multiple times. It is also called a hierarchical implementation with Master/Clones. This methodology allows the duplicated functionality within a chip to be built and verified once, and then instantiated multiple times, thus reducing work load and data storage. Keeping a non-uniquified netlist makes the scope and breadth of late logic changes easier. The logic changes would then be limited to a subset of the chip, reducing the critical path turnaround time and amount of re-verification required to achieve final signoff.

A cloned block is implemented so that it can operate in the worst-case corner scenario in any of its instantiation but once assembled in its top level, the full flat STA might reveal new timing violations. Master/Clone timing optimization is needed because each clone can have:

- Different physical environments (such as different IO loads )
- Different timing environments (such as change in clock slew/skew/OCV between blocks and flat)
- Different signal integrity environments

The Master/Clone timing optimization capability in Signoff ECO is able to fix timing violations and optimize timing in the replicated modules while earlier those would have been treated as `dont_touch`. Any netlist change will be checked against each clone's timing and physical context to ensure that no violations are created. Some of the benefits of Master/Clone timing optimization are:

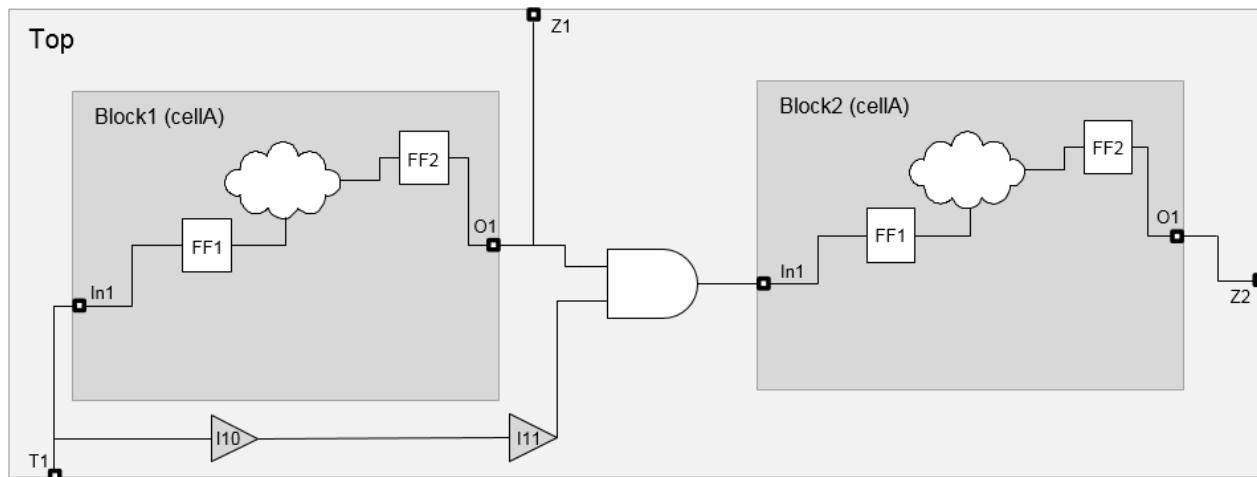
- Top and CPUs are optimized



- One ECO file for all CPUs
- Generated ECOs are valid for each CPU's timing context
- Optimal timing closure for full flat timing

The use model for running ECO timing closure on a hierarchical design with unique or replicated instances is the same.

**Figure 8-7 : Illustration of Master Clone Implementation**



- Physical/timing context is different for every clone boundary
- Timing optimization works on any kind of path:
  - from FF1 to FF2 on all clones or inter-clones paths too (if any)
  - from Top/T1 to Block1/FF1 and from Top/T1 to Block2/FF2
  - from Block/FF2 to Top/Z1 and from Block/FF2 to Top/Z2
  - Paths crossing multiple clones such as Block1/FF2 to Block2/FF1
- Buffering can happen on any of the nets, even the nets crossing the clone boundary

The Master/Clone use model for a design with 2 clones block 1 and block 2 is given below:

```
read_lib $liberty
read_lef -lef "$techno_lef $all_lef"
read_verilog "my_top.v cellA.v"
set_top module my_top
merge_hierarchical_def "my_top.def cellA.def"
```

Instead of the above commands, you can specify the command:

```
read_design -physical_data top.enc.dat my_top
source viewDefinition_and_derating.tcl
read_spef -rc_corner rc_max "my_top_max.spef cellA_max.spef"
read_spef -rc_corner rc_min1 "my_top_min1.spef cellA_min1.spef"
read_spef -rc_corner rc_min2 "my_top_min2.spef cellA_min2.spef"
```

```
set_eco_opt_mode -load_eco_opt_db myECODB
set_eco_opt_mode -partition_list_file partition.txt
Content of partition.txt is:
```

```
cellA
set_eco_opt_mode -optimize_replicated_modules true
eco_opt_design -hold
```

To run this flow, you must do the following:

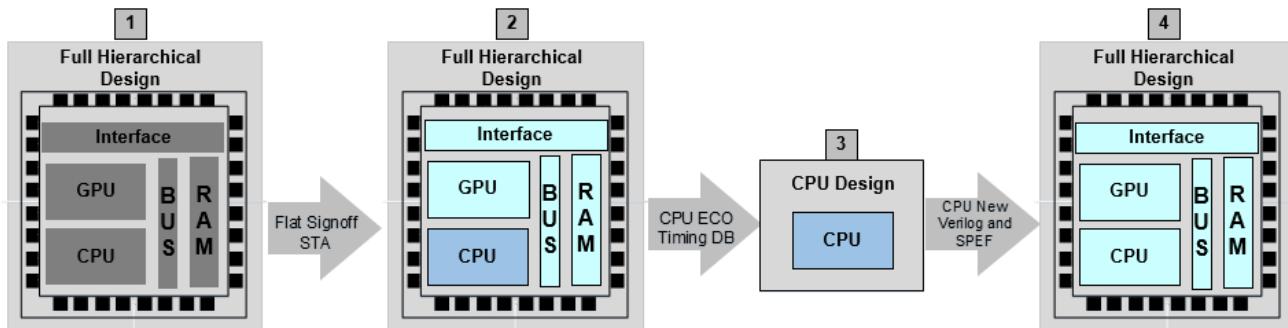
- Specify the `-optimize_replicated_modules true` parameter of the `set_eco_opt_mode` command to optimize timing in the replicated modules. This also enables leakage optimization in the Master/Clone mode. When this command is specified, a single ECO file is generated for each set of clone.
- Specify an ECO Timing database using `set_eco_opt_mode -load_eco_opt_db` since `eco_opt_design` cannot generate it automatically when the design is hierarchical.
- When replicated instances have different orientations, the design must be assembled in Tempus because loading one full flat DEF/Verilog will not work. Or you can also load an assembled DB from Innovus.

## 8.14 Top Down Block ECO Flow

The Top Down Block ECO flow is used to optimize a block level design while using ECO Timing DB generated during top level full flat STA. The software fixes the timing violations for large-scale designs with faster turnaround time and consumes lesser memory, since ECO fixing session is loading only the block level data.

Using this flow, you can run timing analysis on a hierarchical design and finds timing violations in one particular block. Instead of running ECO at the full chip level, you can generate ECO Timing DB for that block only. During the next Tempus session, you can load only the data for the identified block (including physical data) and then reuses the ECO Timing DB generated earlier and runs ECO fixing again. So once the ECOs are implemented, you can rerun timing analysis on the hierarchical design and as a result, less or no timing violations are reported, for the identified block.

**Figure 8-8 : Top Down Block ECO Flow Steps**



- **Step 1:** Design is assembled and final signoff timing constraints are applied.
- **Step 2:** Timing is met except in the CPU block. At block level, CPU design does not include the latest timing signoff constraints.
- **Step 3:** Load Verilog/DEF/Libs for CPU block only, and run Tempus ECO using ECO Timing DB generated at top level.
- **Step 4:** Timing is met for all blocks.

To automate the top down block ECO flow, use the `-pruned_block_name` parameter of the `set_eco_opt_mode` command. Using this parameter, you can provide a module name or hierarchical instance name.

The following scripts are used for running Top down block ECO flow:

### Full Chip Script

```
<load hierarchical design>
<load parasitic>
set_eco_opt_mode -pruned_block_name "top/CPU"
set_eco_opt_mode -save_eco_opt_db ECO-DB-CPU
write_eco_opt_db
```

### Block Level Script

```
<load block level>
<load parasitic>
set_eco_opt_mode -load_eco_opt_db ECO-DB-CPU
eco_opt_design -setup
```

#### Note:

- At block level, you do not need to set this parameter as the tool automatically identifies the running of ECO fixing on an ECO timing DB, which is generated in block scope context.

- There is no need to assemble the physical data for the full chip session.
- You do not have to provide SDC at block level. All timing information extracted from top level are embedded in the ECO Timing DB directory.
- This flow does not support Master/Clone module, but you can select one clone by providing a hierarchical instance name and then run the Top down Block Eco flow.

## 8.15 Generating Node Locations in Parasitic Data

To be able to use `set_eco_opt_mode -along_route_buffering true`, the parasitic must contain information about the node location. You can generate the location of nodes in the parasitic data using the following modes:

- Using Innovus Engines
- Using Standalone QRC
- Using Third-Party Tool

**Note:** Once you have the SPEF file containing the node locations, you can load them using the `-starN` option of the `read_spef` command.

### Using Innovus Engines

Generating node locations is done by default using the Innovus RC/TQuantus/IQuantus engines. The information is saved automatically inside the RCDB file.

### Using Standalone QRC

You can generate SPEF along with node locations using the following option in the QRC command file:

```
output_db -type spef -subtype standard -disable_subnodes false
```

**Note:** Use the following command to pass a command file to Quantus QRC when running it from the Innovus cockpit.

```
setExtractRCMode -qrcCmdFile <file>
```

### Using a Third-Party Tool

You can use a third-party tool to generate SPEF with node locations.

#### Example:

The syntax below shows how node locations are saved in a SPEF file. A SPEF file with the node locations has extra lines with the \*N prefix in the CONN section.

```
*CONN
 *I *15488:z O *C 71.65 237.505 *L 0 *D MUX2_X1
 *I *15465:D I *C 73.035 236.08 *L 0.00316 *D SDFFR_X1
 *N *907:2 *C 72.675 236.95 //CDNZ 1
 *N *907:3 *C 72.675 236.95 //CDNZ 2
 *N *907:4 *C 72.675 235.76 //CDNZ 2
```

## 8.16 Optimization Along Wire Topologies

By default, buffers are inserted only at the start or/and end points of a net. This limits the scope of timing optimization, especially for DRV and Setup optimizations. So to improve the delay of a net, Tempus needs to split the net/pin load and that can only be done efficiently if buffers are added along the route. In addition, this change would also allow buffers to be added at the branching point in case the net has multiple fanouts. Once the buffer is inserted somewhere along the route, the tool precisely estimates the new parasitic for the current and newly created net by using the node location information from the parasitic database.

To enable this feature, there are two prerequisites:

Use `set_eco_opt_mode -along_route_buffering true`.

To provide parasitic with node location (See section, [Generating Node Locations in Parasitic Data on page 98](#))

In addition, when enabling this feature, the setup timing optimization will be able to insert pairs of inverter along the route, if that is seen as a better choice for timing closure compared to single buffer insertion.

**Example:**

```
set_eco_opt_mode -along_route_buffering true
eco_opt_design -drv
```

This will perform DRV fixing in a mode where buffering will also be able to insert buffers along the route topology.

## 8.17 Optimization using Endpoint Control

Tempus ECO supports the following ways for path group-based fixing:

1. Endpoint-based inclusion/exclusion per view
2. Endpoint-specific slack adjustment per view (both positive and negative adjustments)
3. Option to fix only register-to-register paths

Path group support for Tempus ECO can be used for hold or setup fixing.

**Note:** This feature is currently not supported for DRV or leakage optimization.

### **1. Endpoint-based inclusion/exclusion per view**

You can specify endpoints that need to be included or excluded during the optimization for a view with the following set\_eco\_opt\_mode parameters:

- -select\_hold\_endpoints
- -select\_setup\_endpoints

#### **File format:**

```
<View> include/exclude <Endpoints>
```

<View> can be specified as either viewName or V\* (if the endpoints are to be included/excluded for all views)

<Endpoints> can be specified in three ways:

- As space-separated list of endpoint names: for example, E1 E2 E3.
- Slack range in nanosecond: <minSlack> <maxSlack>, where all endpoints with the slack greater than or equal to `minSlack` and slack less than or equal to `maxSlack` will be included/excluded, for example, -2.0 -1.0.
- E\*, if all endpoints for the view need to be included/excluded.

#### **Note:**

- When few pins are excluded from the view, those pins cannot be included back even after overriding the excluded pins, as shown in the below example:

```
> V* exclude E*
> V* include inst/FF/D
```
- Tempus default behavior refers to the inclusion of all pins for the view, as shown below:

```
> V* include E*
```
- Wildcards cannot be used for including/excluding pins, as shown in the below example:

```
> V* include inst/FF/*
> V* include inst*/FF/D
```

- You can set exclude endpoints one by one without using any wild card.

### Sample configuration files

The lines starting with # are comments and will be skipped.

```
File : hold_exclude_example
Exclude endpoints EXECUTE_INST/pc_acc_reg/SE and EXECUTE_INST/
read_prog_reg/SE from view core+typ-rcMin for hold fixing
core+typ-rcMin exclude EXECUTE_INST/pc_acc_reg/SE EXECUTE_INST/
read_prog_reg/SE

File : hold_include_example_with_range
Include only the endpoints that have slacks >= -0.5 ns and <= 0.04
ns for view core+best-rcTyp for hold fixing
core+best-rcTyp include -0.5 0.04
```

## 2. Endpoint-specific slack adjustment per view

You can specify slack adjustment (margin) for selected endpoints during optimization for a view with the following set\_eco\_opt\_mode parameters:

- -specify\_hold\_endpoints\_margin
- -specify\_setup\_endpoints\_margin

### File format:

<View> <Margin> <Endpoints>

<View> can be specified as either viewName or V\* (if the endpoints are to be included/excluded for all views)

<Margin> is specified as a float value in nanosecond. Margin value is subtracted from the endpoint slack, so a positive margin means that the endpoint slack would be degraded by the margin amount.

<Endpoints> can be specified in three ways:

- As space-separated list of endpoint names: for example, E1 E2 E3.
- Slack range in nanosecond: <minSlack> <maxSlack>, where all endpoints with the slack greater than or equal to `minSlack` and slack less than or equal to `maxSlack` would be included/excluded: for example, -2.0 -1.0.
- E\* if all endpoints for the view need to be included/excluded.

## Sample configuration files

The lines starting with # are comments and would be skipped.

```
File : hold_margin_allViews_slackRange_example
Apply margin of 0.1 nanosecond to all endpoints that have slacks between -0.07
to -0.03 for all hold views
V* 0.1 -0.07 -0.03
File : hold_margin_allEndPoints_example
Apply margin of 0.2 nanosecond to all endpoints for hold view core+typ-rcMin
core+typ-rcMin 0.2 E*
File : hold_margin_selectedEndPoints_example
Apply margin of 0.6 ns to endpoints TDSP_CORE_MACH_INST/phi_6_reg/SE and
TDSP_CORE_MACH_INST/phi_1_reg/SE for hold view core+best-rcTyp
core+typ-rcMin 0.6 TDSP_CORE_MACH_INST/phi_6_reg/SE TDSP_CORE_MACH_INST/
phi_1_reg/SE
```

### 3. Option to fix only register-to-register paths

You can choose to optimize timing/leakage/area only on register-to-register paths during eco opt design with the options specified below. Other path group configurations will be ignored in this mode. The timing for non-register-to-register paths for fixing mode (hold or setup) will be adjusted to 0, but the timing for other mode (setup during hold fixing/hold during setup fixing) is not affected.

```
set eco opt mode -optimize_core_only <true/false>
read_spef <all corner spef files>
update_timing -full
report* commands (optional)
```

In addition to timing closure, Tempus ECO is also a solution to reduce overall power consumption of the netlist. At Signoff stage, some amount of timing pessimism is removed due to Path Based Analysis, also called retiming, and that extra timing margin can be used to perform netlist change leading to lesser overall power. This is especially beneficial when AOCV or SOCV timing mode are enabled.

## 8.18 Metal ECO Flow

The *Metal ECO* flow, also known as *Post-Mask* flow supports the Gate Array filler cells. Using this flow, the timing closure engine performs the netlist change without touching the base layer mask.

Before starting metal ECO, perform the following sanity checks:

- All instances of the design must be placed (no unplaced instance allowed).
- The initial design placement density must be 100%, that is no empty spaces left.

- The list of Gate Array filler cells must contain enough granularity to ensure that any empty space created during metal ECO can be filled back.
- Do not specify cells through `setTieHiLoMode` if Tie cell insertion is not allowed.

The following command is used to run the metal ECO flow:

```
set_eco_opt_mode -post_mask true
```

The software performs the following netlist changes:

- Inserts the Gate Array buffers or an inverter pair in place of existing Gate Array filler cells.
- Resizes regular cells to Gate Array cells. It also resize Gate Array cells to Gate Array cells.
- Deletes a regular cell.

The software automatically identifies the Gate Array cells by checking which library instances have the same SITE name as the Gate Array filler cells specified by the user. Here, empty spaces will be filled up by the Gate Array filler cells to ensure 100% placement density.

By default, the `-post_mask` parameter is set to `false`.

The following command lists all the Gate Array filler cells, which can be deleted or inserted back:

```
set_eco_opt_mode -use_ga_filler_list {<list_of_Gate_Array_filler_cells_name>}
```

Note that this parameter is applicable in metal ECO mode only.

**Note:**

- The Metal ECO flow is supported by DRV, Setup, and Hold fixing.
- When `setTieHiLoMode -cell {{<tieLo_name> <tieHi_name>}}` setting is used, this flow will insert the TieLo cells to connect dangling input pins, in case the existing TieLo cells in the floorplan are extremely far or too overloaded.

**Example:**

The following command enables Hold fixing in the metal ECO mode by allowing the listed Gate Array filler cells to be deleted or inserted during the ECO process.

```
tempus> set_eco_opt_mode -post_mask true
tempus> set_eco_opt_mode -use_ga_filler_list {GFILL1BWP GFILL2BWP GFILL4BWP
GFILL10BWP}
tempus> eco_opt_design -hold
```

## 8.19 Handling Large Number of Active Timing Views Using SmartMMMC

The *SmartMMMC* feature handles a large number of active timing views. This feature is used to enable reduced memory and runtime solution when many timing views are enabled during timing or power ECO fixing stage.

To enable the *SmartMMMC* feature in the two sessions flow, use the below parameters of the `set_eco_opt_mode` command:

■ `set_eco_opt_mode`:

- `-create_smart_mmmc_db true|false`: Enables saving the additional compact timing information in the ECO DB directory during the STA session. When set to `true`, the parameter must be applied before the `write_eco_opt_db` command and has no effect on the `eco_opt_design` command.

**Example:**

The following command saves the regular timing and additional compact timing information in the ECO DB directory named `EcoTimingDB` in the STA SMSC session.

```
tempus> set_eco_opt_mode -save_eco_opt_db EcoTimingDB
tempus> set_eco_opt_mode -create_smart_mmmc_db true
tempus> write_eco_opt_db
```

- `-load_smart_mmmc_db <dirName>`: Points Tempus ECO to the ECO DB directory containing the additional compact timing information during the ECO fixing session. This parameter must be set before the design is loaded and must point to an ECO DB directory, which was previously generated through `write_eco_opt_db` using the setting, `set_eco_opt_mode -create_smart_mmmc_db true`.

**Note:** Before `eco_opt_design`, you still need to point to the ECO DB directory using the setting, `set_eco_opt_mode -load_eco_opt_db <dirName>`.

**Example:**

The following command loads the design with a minimized set of data to reduce the memory or runtime during an ECO fixing stage when many timing views are enabled, and performs Setup timing closure in the ECO MMMC session.

```
tempus > set_eco_opt_mode -load_smart_mmmc_db EcoTimingDB
tempus> <load_design>
tempus> set_eco_opt_mode -load_eco_opt_db EcoTimingDB
tempus> eco_opt_design -setup
```

## Tempus User Guide

### Signoff ECO

---

To enable the *SmartMMMC* feature in the *Paradime* flow, use the below parameters of the `set_distribute_mmmc_mode` command:

■ `set_distribute_mmmc_mode`:

- `-enable_smart_mmmc`: when this is enabled, it must be applied before the design is loaded in Paradime and `set_distribute_mmmc_mode -eco true` should be set.

**Example:**

The following commands enable design loading in Paradime with the minimized set of data to reduce the memory or runtime during ECO fixing stage when many timing views are enabled, and performs Setup timing closure.

```
tempus > set_multi_cpu_usage -localCpu 16 -remoteHost 4 -cpuPerRemoteHost 4
tempus > set_distribute_mmmc_mode -eco true
tempus > set_distribute_mmmc_mode -enable_smart_mmmc
tempus > set_eco_opt_mode -retime path_slew_propagation
tempus > set_distribute_mmmc_mode -merge_view_definitions
tempus > set eco_setup_view_list "<list_of_Setup active_views>"
tempus > set eco_setup_view_list "<list_of_Hold active_views>"
tempus > set eco_lef_file_list "<list_of_LEF_files>"
tempus > set eco_def_file_list "<list_of_DEF_files>"
tempus > distribute_read_design -restore_design all -restore_dir
PARADIME_SESSIONS/ -outdir reports_Paradime
tempus > eco_opt_design -setup
```

---

# Timing Model Generation

---

- 9.1 [Extracted Timing Models](#) on page 107
- 9.2 [Boundary Models](#) on page 151
- 9.3 [SmartScope Hierarchical Analysis](#) on page 164
- 9.4 [Prototype Timing Modeling](#) on page 172
- 9.5 [Interface Logic Models](#) on page 182

## 9.1 Extracted Timing Models

- 9.1.1 [Overview](#) on page 108
- 9.1.2 [ETM Generation](#) on page 109
- 9.1.3 [ETM Generation for MMMC Designs](#) on page 111
- 9.1.4 [Slew Propagation Modes in Model Extraction](#) on page 111
- 9.1.5 [Basic Elements of Timing Model Extraction](#) on page 112
- 9.1.6 [Secondary Load Dependent Networks](#) on page 125
- 9.1.7 [Characterization Point Selection](#) on page 126
- 9.1.8 [Constraint Generation during Model Extraction](#) on page 128
- 9.1.9 [Parallel Arcs in ETM](#) on page 132
- 9.1.10 [Latency Arcs Modeling](#) on page 133
- 9.1.11 [Latch-Based Model Extraction](#) on page 133
- 9.1.12 [Model Extraction in AOCV Mode](#) on page 134
- 9.1.13 [Stage Weight Modeling in ETM](#) on page 134
- 9.1.14 [AOCV Derating Mode](#) on page 135
- 9.1.15 [PG Pin Modeling During Extraction](#) on page 136
- 9.1.16 [Extracted Timing Models with Noise \(SI\) Effect](#) on page 137
- 9.1.17 [Extracted Timing Models with SOCV](#) on page 138
- 9.1.18 [Merging Timing Models](#) on page 138
- 9.1.19 [Limitations of ETM](#) on page 140
- 9.1.20 [Validation of Generated ETM](#) on page 144
- 9.1.21 [Auto-Validation of ETM](#) on page 147
- 9.1.22 [ETM Extremity Validation](#) on page 148
- 9.1.23 [Limitation/Implications of EV-ETM](#) on page 150
- 9.1.24 [Ability to Check Timing Models](#) on page 150

### 9.1.1 Overview

Timing model extraction is the process of extracting the interface timing of hierarchical blocks into a timing library. Typically, model extraction has the following advantages:

- Reduces the memory requirements by generation of extraction timing models (ETMs) for respective blocks, which may be huge in size.
- Reduces the static timing analysis (STA) run time.
- Hides the proprietary implementation details of the block from a third-party.

The `do_extract_model` command is used to build a timing model (.lib) of a digital block to be used with a timing analyzer. The extraction of the "actual" timing context for a lower-level module instance is required for achieving timing convergence with large hierarchical design databases. The timing analyzer has the following advantages while analyzing timing extracted models instead of a complete block design:

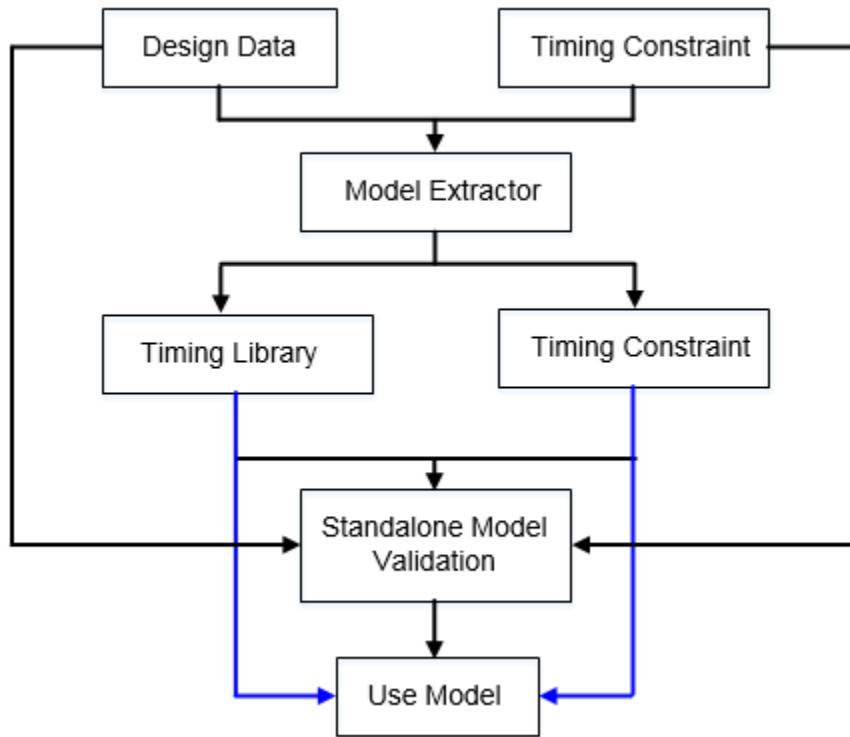
- Provides the timing engine capability to allow for quick derivation of a module's timing context.
- Extracts data correctly across multiple clock domains.
- Allows merging of various models in various modes for their respective blocks.

You can start with model extraction with the following data loaded in the software:

- Design netlist
- Timing libraries
- Block context (constraints, such as, clocks, path exceptions, operating conditions etc.)
- RC data of the nets

The following diagram illustrates this flow:

**Figure 9-1 ETM Flow**



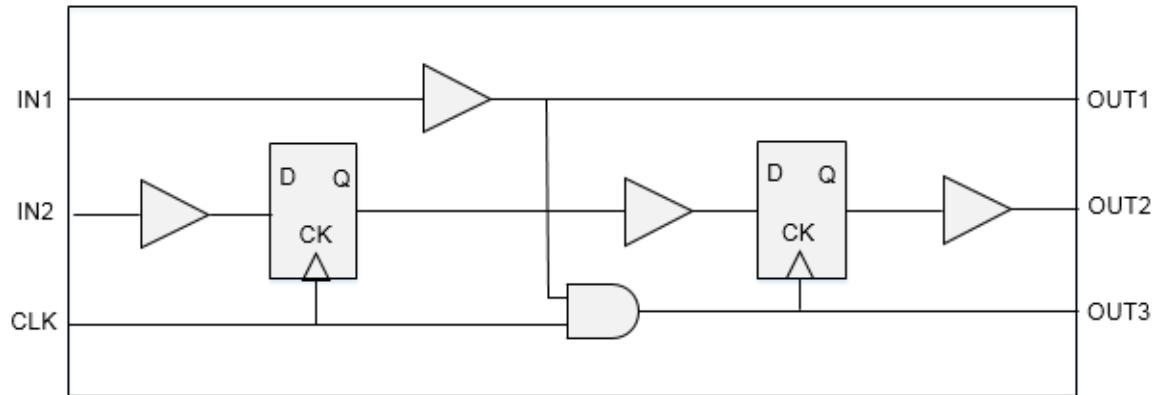
### 9.1.2 ETM Generation

ETM represents the timing for interface paths. Input to flop/gate, input to output and flop/gate to output paths of a design are preserved as timing arcs in the ETM. If there are multiple clocks capturing the data from an input port then an arc with respect to each input port is extracted.

The flop-to-flop type of paths are not written in the ETM, as these do not affect the interface paths timing.

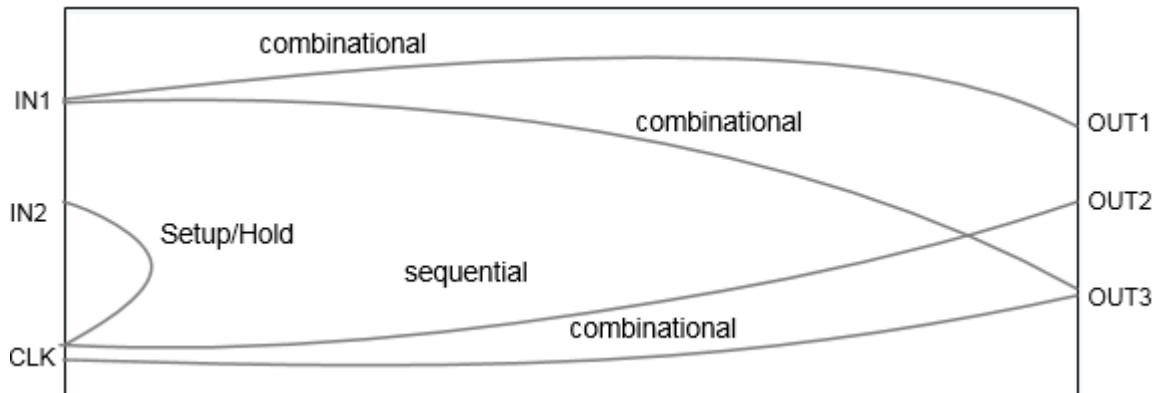
The following figure shows the equivalent ETM for a given netlist:

**Figure 9-2 Equivalent ETM for a given netlist**



The following figure shows the extracted model:

**Figure 9-3 Extracted Model**



The extracted timing model is context-independent because it gives the correct value of arcs/delays, even with varying values of input transitions, output loads, input delays, or output delays. In such cases, it is not required to re-extract the model if some of the context is changed at a later stage of development.

However, the model depends upon the operating conditions, wire load models, annotated delays/loads, and RC data present on internal nets defined in the original design. So if these elements change at the development stage of design then we need to re-extract the model for correlation with the changed scenario.

### 9.1.3 ETM Generation for MMMC Designs

In MMMC configuration, for generating ETM, only one view should be active. If you need to generate ETM for a specified view, however, the view is not active in setup as well as hold mode, then the software will show an error. It is a prerequisite that the view should be active for both setup and hold analysis. You can use the `set_analysis_view -setup {$viewName} -hold {$viewName}` command before running the `do_extract_model` command to achieve this.

After model extraction for MMMC designs, the dumped assertion files will be added with  `${viewName}` suffix, and the dumped derivative related assertion files will be added with  `${delayCornerName}` prefix. You need to choose the corresponding file while using the extracted models.

### 9.1.4 Slew Propagation Modes in Model Extraction

Model extraction supports worst slew and path-based slew propagation modes. These are described below.

#### ■ Worst Slew Propagation

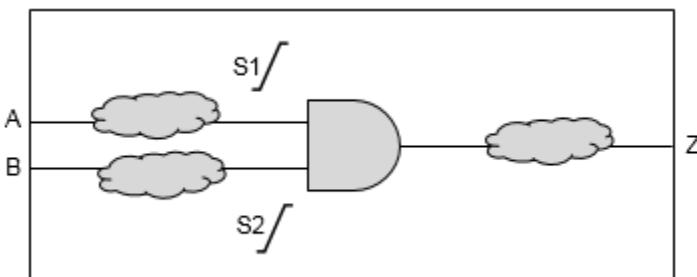
In worst slew propagation mode, the worst of all the slews coming at a convergent point is propagated for characterizing the delay of an arc.

#### ■ Path-Based Slew Propagation

In path-based slew propagation mode, the slew in the actual path is propagated.

The slew propagation mode can be controlled by using the `timing extract model slew propagation mode` global variable. The default value of this global variable is `worst_slew`.

An example of slew propagation is given below.



In worst slew propagation mode, for characterizing both A-Z and B-Z delay arcs, the worst slews S1 and S2 will be propagated to calculate the delay of logic downstream to the AND gate.

In path-based propagation mode, the slew S1 will be propagated to characterize the A-Z arc, whereas S2 will be propagated to characterize the B-Z arc.

### 9.1.5 Basic Elements of Timing Model Extraction

The following basic elements are relevant to model extraction:

- Nets – internal and boundary nets
- Timing Paths - check and delay paths – in2reg, in2out, and reg2out paths
- Minimum Pulse Width and Period Checks
- Path Exceptions – false, multi-cycle, min delay, max delay, disable timing
- Constants
- Unconstrained Paths
- Clock Gating Checks
- Annotate Delays, Slews, and Load
- Design Rules
- Clocks - created clocks and generated clocks

These are explained in the following sections.

#### Nets

Nets can mainly be classified into two types – boundary nets and internal nets. The nets which are directly connected to the input or output ports of the block are termed as the boundary nets. The nets which drive and are driven by some internal instance pin of the block are termed as the internal nets. Model extraction treats both the nets differently as boundary nets are always related and connected to the context.

##### ■ Boundary Nets

The boundaries are connected to the context. So the RC data for them may change if the context of the block changes at a later stage. To be better context independent the model should not consider SPEF or DSPEF data for their delay calculation and a separate SPEF/DSPEF file for the boundary nets can be stitched to the top level data while instantiating the extracted model.

The software by default considers the RC data while extracting the model. The software provides a command-line option that can be used to ignore the RC data for boundary nets by using the following command:

```
set_delay_cal_mode -ignoreNetLoad true
```

**Note:** Currently, the software does not output the SPEF file for boundary nets which can be used at the top level. To solve this problem, you need to create the top level SPEF file for boundary nets of this block or use the default behavior to consider the boundary nets RC data for model extraction.

## ■ Internal Nets

As the internal nets connect the pins for the internal instance of the block, they are in no way dependent on the context environment. So the RC data defined for the internal nets is used as it is for delay calculation and is added in the extracted paths. If no RC is defined for these nets, then the wire load models are used to calculate their delays.

However, if a load or resistance is annotated using the `set_load/set_resistance` command, then it will override the annotated SPEF or the applied wire load model.

If the nets are annotated with the delays using the `set_annotation_delay` command or SDF annotation, then the annotated delay is used instead of the calculated delays. In case of incremental delays the addition of calculated and incremental delays is used.

## Timing Paths

Timing paths are broadly divided in four types:

- In2reg
- reg2out
- in2out
- reg2reg: The reg2reg paths are not relevant to the model extraction process.

These are described below.

### ■ In2Reg Paths

A In2reg path is a setup/hold check that starts from an input port and is captured at a flop or gating element by a clock. So the in2reg type of paths are captured in equivalent setup or hold checks. The setup/hold values to be written in ETM are calculated using the data path delay, the setup/hold value of the library cell and the clock path delay. The equations can be written as follows.

Setup arc value = data path delay (input to flop) + setup value of flop – clock path delay (clock source to clk pin)

Hold arc value = data path delay (input to flop) - hold value of flop – clock path delay (clock source to clk pin)

The value for these arcs is the function of the transition on the input port and the transition at the clock source. If a flop is captured by multiple clocks then separate setup/hold arcs are extracted with respect to each clock source.

### ■ **Reg2out Paths**

The reg2out paths are paths starting from a register and ending up on an output port. These paths are a combination of trigger arcs (of starting register) and the combinational delay from sink of trigger arc to the output port. So for such paths an equivalent trigger/sequential arc is modeled in the extracted model. The delay for the arc will be equal to:

Sequential arc delay = delay (clock source to CK of register) + delay (register CK pin to out port).

The delay for these arcs is a function of the slew at the clock source and the capacitance at the output port. As the extracted model can be used for max as well as min analysis, two arcs are preserved in the model to represent the longest and the shortest path. Different types (rising\_edge/falling\_edge) of arcs are extracted for the different valid clock edges.

### ■ **In2Out Paths**

The in2out paths are the path starting from an input port and ending up on an output port. These paths are pure combinational paths. So for such paths an equivalent combinational arc is modeled in the extracted model. The delay for the arc will be equal to:

Combinational arc delay = delay (delay of all elements in the path)

The delay for these arcs is a function of the slew at the input port and the capacitance at the output port. As the extracted model can be used for max as well as min analysis, two arcs are preserved in the model to represent the longest and the shortest combinational path. In case if no path exists for a particular transition (rise/fall), half unate (combinational\_rise /combinational\_fall) arcs will be extracted. The timing sense for the arc will depend on the function of worst (early/late) paths.

## **Minimum Pulse Width and Period Checks**

The minimum pulse width and period constraints defined at the CK pin of the registers are transferred to the clock source pins during model extraction.

There may be several different type of registers in the fanout of a clock source. So while transferring the minimum pulse width to the clock source you can use the worst values present on the fanout registers. The pulse width is calculated as:

- pulse width = MAX(Arrival time of Launch clock Path - Arrival time of Capture Clock Path + pulse width at clock pins from library)

The minimum period for any clock pin is calculated in the same way as the minimum pulse width. The minimum period is calculated as:

- Min Period = MAX(Arrival time of Launch clock Path - Arrival time of Capture Clock Path + Min Period at clock pins from library)

These checks can be modeled in the following three ways:

- Library Attribute
- Scalar Tables
- Arc-Based Modeling

These are explained below.

### ***Library Attribute***

By default, these checks are written as library attributes in ETM.

```
pin (CLKIN2) {
 clock : true ;
 min_period : 2.0554;
 min_period : 2.6248;
 min_pulse_width_low : 0.2773;
 min_pulse_width_high : 0.7673;
}
```

If `min_period` checks corresponding to both rise and fall transitions are present in a design, then they will be modeled as duplicate entries in ETM when written as library attributes. To avoid duplicate entries we can model these checks as scalar tables or arcs, where rise and fall transition of `min_period` checks can be modeled as shown in following sections. These arcs will be honored by the timer depending on the context, hence will be more accurate.

### ***Scalar Tables***

If the `timing extract model write clock checks as scalar tables` global variable is set to `true`, then these checks will be written as scalar tables as shown below:

```
pin (CLKIN2) {
 clock : true ;
 timing() {
 timing_type : minimum_period ;
```

```

 rise_constraint (scalar){
 values(" 2.0554");
 }
 fall_constraint (scalar){
 values(" 2.6248");
 }
 related_pin :" CLKIN2 ";
 }
 timing() {
 timing_type : min_pulse_width ;
 rise_constraint (scalar){
 values(" 0.7673");
 }
 fall_constraint (scalar){
 values(" 0.2773");
 }
 related_pin :" CLKIN2 ";
 }
}

```

### Arc-Based Modeling

These checks can also be written as arcs by setting the `timing_extract_model_write_clock_checks_as_arc` global variable to `true`. In this case the resulting arc is the function of rise/fall slew of CLK port, hence delay and slew propagation of the clock network for rise and fall transitions is considered for writing these arcs. The arc-based modeling of clock-style checks is more accurate and is recommended for use.

```

pin (CLKIN2) {
 clock : true ;
 timing() {
 timing_type : minimum_period ;
 rise_constraint (lut_timing_2){
 values(" 2.000, 2.000, 2.000, 2.000, 2.000, 2.000, 2.000");
 }
 fall_constraint (lut_timing_2){
 values(" 2.000, 2.000, 2.000, 2.000, 2.000, 2.000, 2.000");
 }
 related_pin :" CLKIN2 ";
 }
 timing() {
 timing_type : min_pulse_width ;
 rise_constraint (lut_timing_2){
 values("0.767, 0.767, 0.767, 0.767, 0.742, 0.087, 0.100");
 }
 fall_constraint (lut_timing_2){
 values("0.201, 0.201, 0.201, 0.201, 0.238, 0.247, 0.2636");
 }
 related_pin :" CLKIN2 ";
 }
}

```

## Path Exceptions

Timing extraction flow honors path exceptions defined in the given constraints' file. Typically, the following path exceptions are used:

- set false path
- set multicycle path
- set min delay
- set max delay

For more information on handling of these exceptions, refer to section "[Constraint Generation during Model Extraction](#)".

## Constants

The case analysis and constants are propagated while extracting a model. When you specify the set case analysis command, you can use the following global variable to control how constants, propagated at o/p or bi-di ports, are modeled in ETM:

```
set timing extract model case analysis in library true (or false)
```

When set to `false`, the software models the propagated or applied constants to output ports in the generated constraints' file (which is generated using the do extract model - assertions command).

When set to `true`, the software models the propagated constants to output ports written as pin functions in the extracted model.

## Unconstrained Paths

The unconstrained end points exist when proper launch/capture clock phase is not propagated at the desired endpoint due to any exceptions. The unconstrained input ports due to false path exceptions are ignored and are not modeled during ETM extraction. The unconstrained combinational IO paths due to false path exceptions are ignored and are not modeled during ETM extraction. The unconstrained trigger arcs are calculated during ETM extraction.

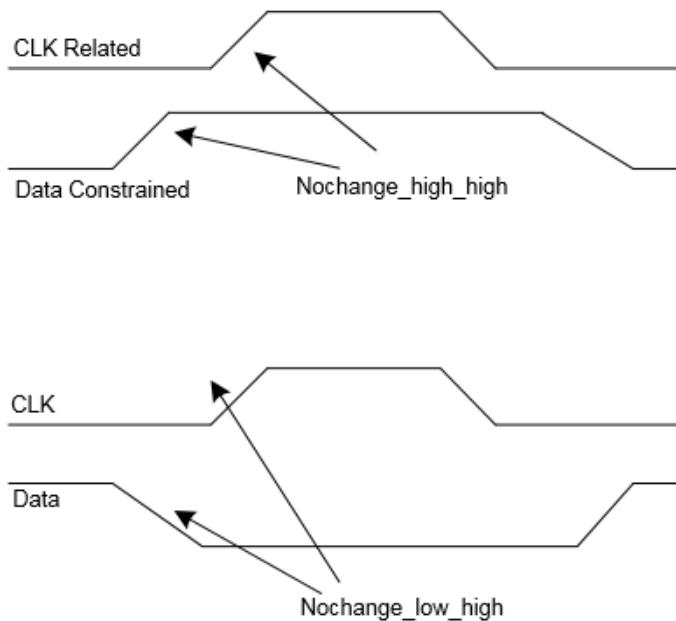
## Clock Gating Checks

If integrated clock gating (ICG) cells are used for gating the clocks, then these arcs are preserved as setup/hold arcs. If combinational cells are used for gating, then these arcs will

be preserved as no-change arcs between a clock pin and its enabling signal pin. If you wish to extract these checks as regular setup/hold checks, you can set the `timing_extract_model_gating_as_nochange_arc` global variable to `false`.

Depending on the type of clock gating situation, no change checks are inferred as follows:

**Figure 9-4 no\_change Checks Modeling**



### **Simple Clock Gating with AND or Logic**

The following diagram shows clock gating AND/OR logic:



A simple AND gate check will be as follows:

```
timing () {
 timing_type: nochange_high_high;
}
timing () {
 timing_type: nochange_low_high;
}
```

**Note:** After extracting the clock gating check arc the signal downstream the gate output is not propagated to the clock pins of the registers.

### **Clock Gating with Blackbox or Unknown Logic**

In case of clock gating with unknown logic, as shown below, no\_change checks will be checked with respect to both the rising edge and the falling edge of the clock signal.



### **No Clock Gating Logic**

There are some gates, such as multiplexers or XOR gates, where a clock signal cannot control the clock gate, as shown below.



In such cases, no checks are inferred so you need to explicitly set the gating check if you want these interface paths to be extracted in the ETM. To know where static timing analyzer will not infer the gating in such situations, you can use the following command:

```
check_timing -check_only clock_gating_controlling_edge_unknown -verbose
```

### **Annotate Delays, Slews, and Load**

Model extraction uses the back annotated delay slews and the load information during the extraction process and reflect the effect in the extracted model. Here is a small description how these are used.

#### **■ Annotated Delays**

The annotated delays using SDF or the `set_annotated_delay` command override the calculated delay (from library or RC data) value for the arc; however, the output slew for the arcs is used as calculated. In case of incremental delays the delta delay is added to the calculated arc delay.

#### **■ Annotated Slews**

Annotated slews are ignored during timing model extraction. These constraints are dumped in assertions file generated with the `do_extract_model -assertions` parameter.

## ■ **Annotated Load**

Annotated load overrides the pin capacitance defined in the library, and is used for delay calculations.

**Note:** The annotated delays are generated with a particular context and remain true for that context (transition times) only. So annotating the delays/slews makes the model context dependent. So to create a context independent model it is recommended not to annotate the SDF.

## **Design Rules**

Model extraction uses the design rules defined in the library. The worst (smaller for max design rules and vice-versa) among all the fanout pins (topologically first level) is used for the input ports and the worst of all the fanin pins (topologically first level) is used for the output ports.

If design rule limits are defined at the pin and library level, the design rule from the pin is chosen, rather than the worst of the pin and library level, because the pin level information overrides the cell level, which in turn overrides the library level information. If design rule violations (DRV) are coming from the constraints, then you can choose them based on the following setting:

```
set timing extract model consider design level drv true (or false)
```

When set to `false`, the user-asserted design level DRVs are not considered while modeling DRVs in the extracted timing model.

When set to `true`, the user-asserted design level DRVs are considered while modeling DRVs into the extracted timing model.

## **Clocks**

- Created Clocks
- Generated Clocks
- Clocks with Sequential and Combinational Arcs

### ***Created Clocks***

If the `create_clock` assertion is applied on the pin of a design (not port), then the pin is modeled as an internal pin with the same name as that of the clock asserted on it. If a clock is created on a design port, then this port will be preserved as ETM I/O pin, with the same name as that of the port itself.

```
create_clock [get_ports {CLKIN}] -name clk -period 3 -waveform {0 1.5}
create_clock [get_pins buf5/Y] -name clk3 -period 3 -waveform {0 1.5}

pin (CLKIN) {
 clock : true ;
 direction : input ;
 capacitance : 0.0042;
}
pin (clk3) {
 clock : true ;
 direction : internal ;
 capacitance : 0.0110;
}
```

### ***Generated Clocks***

The generated clocks that defined in a hierarchical block are supposed to be a part of the block itself and do not come from the top level. So ETM preserves generated clocks inside the model as internal pins using the Liberty generated\_clock construct. The name of any such internal pin is the same as that of the generated clock.

Some of the examples are described below.

#### **Example 1: Single generated clock on a single pin**

```
create_generated_clock -name gclk -source [get_ports clk] -divide_by 2
[get_pins buf1/A]
```

During the extraction process the pin “buf1/A” will be preserved as an internal pin in the library with name gclk. The model will show as follows:

```
generated_clock (gclk) {
master_pin : clk;
divided_by : 2;
clock_pin : "gclk ";
}
pin (gclk) {
clock: true ;
direction: internal ;
.
.
}
```

#### **Example 2: Multiple clocks on the same pin**

There are some design scenarios when multiple generated clocks are defined on a single pin. This asserts multiple clock definitions on the pins. In this case, during model extraction the pin is duplicated with the name of the clock. For example, in a design if there are two clock definitions, such as:

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK
-divide_by 2 [get_pins buf1/A]
create_generated_clock -name gclk2 -source [get_ports clk] -add -master_clock CLK
-divide_by 2 [get_pins buf1/A]
```

During the extraction process the pin buf1/A will be preserved as an internal pin with names gclk1 and gclk2. The model will be as follows:

```
generated_clock (gclk1) {
 master_pin : clk;
 divided_by : 2;
 clock_pin : "gclk1 ";
}
generated_clock (gclk2) {
 master_pin : clk;
 divided_by : 2;
 clock_pin : "gclk2 ";
}
pin (gclk1) {
 clock: true;
 direction: internal;
....
}
pin (gclk2) {
 clock: true;
 direction: internal;
....
}
```

### **Example 3: Generated clocks on multiple pins**

When a generated clock is defined on multiple pins, the software asserts a single clock definition on multiple pins. To handle this scenario, all the pins asserted with this clock are preserved as internal pins in the model with the name [clock\_name\_<count>] and the generated\_clock construct is written in the model with all the pin names in the clock\_pin attributes with a space between them. For example, consider a netlist having clock definitions on multiple pins.

```
create_generated_clock -name gclk1 -source [get_ports clk] -add -master_clock CLK
-divide_by 2 [get_pins {buf1/A buf2/A}]
```

During the extraction process, the pin buf1/A will be preserved as an internal pin in the library with the name gclk1\_1; and the pin buf2/A will be preserved as an internal pin with name gclk1. The model will be as follows:

```
generated_clock (gclk1) {
 master_pin : clk;
 divided_by : 2;
 clock_pin : "gclk1_1 gclk1 ";
}
pin (gclk1_1) {
 clock: true;
 direction: internal;
...
}
pin (gclk1) {
 clock: true;
 direction: internal;
...
}
```

In this case when a ETM is read, two generated clocks `gclk1_1` and `gclk1` will be created on two internal pins - `gclk1_1` and `gclk1`, respectively. Any constraint coming from the top level will match only with clock `gclk1`, and not with `gclk1_1`. To avoid such a situation, you can set `timing_library_genclk_use_group_name` global variable to `true`. When this global variable is enabled, the original clock (`gclk1`) is generated at two internal pins - `gclk1_1` and `gclk1`.

The `report_clocks` command shows the following output:

Clock Descriptions							
Clock Name	Source	Period	Lead	Trail	Gen	Prop	
clk	clkin	3.600	0.000	1.800	n	y	
gclk1	BLOCK/gclk1_1	7.200	0.000	3.600	y	y	

Clock Descriptions							
Clock Name	Source	Period	Lead	Trail	Gen	Prop	
clk	clkin	3.600	0.000	1.800	n	y	
gclk1	BLOCK/gclk1	7.200	0.000	3.600	y	y	
gclk1_1	BLOCK/gclk1_1	7.200	0.000	3.600	y	y	

#### Example 4: Multiple clocks on master clock root pin

Liberty supports the `master_pin` attribute inside a `generated_clock` group definition, which refers to the pin where the corresponding master clock is defined.

Please note that you are not allowed to define a master clock name in the `generated_clock` group in Liberty. Due to this, the software may infer an incorrect master for the generated clock `genclk` – if multiple clocks are defined on the `CLKIN` pin.

```
generated_clock (genclk) {
 divided_by : 2 ;
 clock_pin : " genclk ";
 master_pin : CLKIN ;
}
```

Hence, for the correct master clock association with the generated clocks, you need to provide the `create_generated_clock` definition with the corresponding master clock, while reading the ETM at the top level.

#### Example 5: Generated clock in case of multi-ETM instantiation at top

When the `timing_prefix module_name with library genclk` global variable is set to `true`, the software appends the instance name to the clock pin name when creating a generated clock.

For example, assume that the cell for instance a/b in the timing library contains the following generated clock group:

```
generated_clock (genclk1) {
 divided_by : 2 ;
 clock_pin : " genclk1 " ;
 master_pin : CLKIN ;
}
```

By default (`true`), the software creates a generated clock with the name - `genclk1`.

When set to `false`, the software uses only the clock pin name when creating a generated clock.

If ETM is instantiated at the top level as `BLOCK1` and `BLOCK2`, then by default the `report_clocks` command output will show the following:

Clock Descriptions						
Clock Name	Source	Period	Lead	Trail	Generated	Propagated
BLOCK1/genclk1	BLOCK1/genclk1	7.200	0.000	3.600	y	y
BLOCK2/genclk1	BLOCK2/genclk1	7.200	0.000	3.600	y	y

If you do not want to differentiate between `genclk1` generated in various ETM instantiations, you can set `timing_prefix_module_name_with_library_genclk` global variable to `false`. In this case, the `report_clocks` command will show the following output:

Clock Descriptions						
Clock Name	Source	Period	Lead	Trail	Generated	Propagated
genclk1	BLOCK2/genclk1	7.200	0.000	3.600	y	y

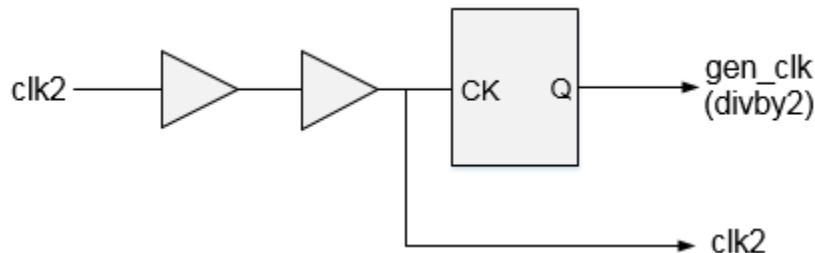
**Note:** You will need to ensure that the constraints are applied with respect to the generated clock naming. If required, you can remove the library generated clocks and then apply the specified generated clocks so that minimal changes are required in the constraints.

### ***Clocks with Sequential and Combinational Arcs***

Consider a case where both the sequential and combinational arcs exist between a clock (CLK) and an output (OUT) pin. If any of the early/late sequential/combination arcs are missing (due to a path exception) between the CLK and OUT pins, the CLK pin is duplicated

as two pins with `_SEQ_pin` and `_COMB_pin` suffixes. All the combinational arcs starting from this clock root to the duplicated pin are bound with the “`_COMB_pin`” suffix and all the sequential arcs are bound to be duplicated pin with the “`_SEQ_pin`” suffix.

**Figure 9-5 Sequential and combinational arcs between a clock**



The following will be created in the ETM:

```

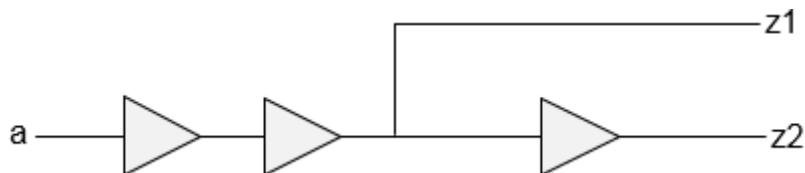
pin (CLKIN_SEQ_pin) {
 clock : true ;
 direction : internal ;
 capacitance : 0.0110;
}
pin (CLKIN_COMB_pin) {
 clock : true ;
 direction : internal ;
 capacitance : 0.0110;
}
pin (DATAOUT) {
 direction : output ;
 timing() {
 timing_type : combinational ;
 fall_transition (lut_timing_3){ ... }
 related_pin :" CLKIN_COMB_pin ";
 }
 timing() {
 timing_type : rising_edge ;
 fall_transition (lut_timing_3){ ... }
 related_pin :" CLKIN_SEQ_pin ";
 }
}

```

### 9.1.6 Secondary Load Dependent Networks

If there is a timing path from one output pin to another pin, the model extractor considers both the primary output loading and the secondary output loading when output-to-output delays are computed. In the following figure there is an output-to-output path from z1 to z2:

**Figure 9-6 Output-to-Output path from z1 to z2**



The extracted timing model for this block consists of two delay arcs - one from  $a \rightarrow z_1$  and another from  $a \rightarrow z_2$ . But the delay of arc  $a \rightarrow z_2$  also depends on the secondary loading, that is, at port  $z_1$ .

This gives rise to a delay arc from  $a$  to  $z_2$  which is dependent on two loads and one input slew. So this arc will be dumped as a three-dimensional delay table because the delay depends on the primary output loading at  $z_2$ , as well as a secondary output loading at  $z_1$ .

The table template will be as follows:

```

lu_table_template (lut_timing_1) {
variable_1: input_net_transition;
index_1 ("0...0 1.0 2.0 3.0 4.0 5.0");
variable_2: total_output_net_capacitance;
index_2 ("0...0 1.0 2.0 3.0 4.0 5.0");
variable_3: related_out_total_output_net_capacitance;
index_3 ("0...0 1.0 2.0 3.0 4.0 5.0");
}

```

By default, 3D arc modeling is disabled - the software considers 3D arcs as 2D arcs in timing modeling.

**Note:** You should always buffer the port to avoid 3D dependencies. If there is more than a couple of load dependencies, then there will be accuracy loss since model extraction cannot accurately model beyond 3D arcs accurately.

### 9.1.7 Characterization Point Selection

The selection of characterization during point extracted models depends on the selection criteria described below.

If no `input_slews`/`clock_slews`/`output_load` is specified as an argument in the `do_extract_model` command, then the software determines the characterization points from the designs' interface elements - that are supplied with the external slew/load. This can be explained as follows.

All the check arcs (e.g., `clk->in`) will be characterized for the slew points as follows:

- Reference slew points will be taken from the slew index of the first element just after the "clk" port.

- Signal slew points will be taken from the slew index of the first element just after the “in” port.

All the sequential (e.g., clk->out) arcs will be characterized as follows:

- Input slew will be taken from the slew index of the first element just after the \*clk\* port.
- Output load will be taken from the load index of the last element just before the \*out\* port.

All the combinational arcs (e.g., in->out) will be characterized as follows:

- Input slew will be taken from the slew index of the first element just after the \*in\* port.
- Output load will be taken from the load index of the last element just before the \*out\* port.

Consider the following topology in a netlist:

in -----> buf1-- -----> ff1/d ----->buf2 -----> out

clk -----> clk\_buf ---- ->ff1/clk----->buf2 -----> out

in -----> buf3 -----> buf4 -----> buf5 -----> out

A snapshot of the original timing library for the interface elements (i.e., BUF and CLK\_BUF) is as follows:

```
Cell (BUF)
{
 timing ()
 index_1 ("0.0500, 1.4000, 4.5000");
 index_2 ("1.0500, 6.5000, 10.0000");
}
Cell (CLK_BUF)
{
 timing ()
 index_1 ("1.0500, 2.4000, 3.5000");
 index_2 ("0.0500, 4.5000, 5.0000");
}
```

If the do extract model command is used, then the extracted model will be characterized as follows:

- The signal slew is taken from buf1 slew-index in the original libraries.

```
Check Arc clk->in
index_1 ("0 0.0500, 1.4000, 4.5000");
```

- The reference slew is taken from clk\_buf slew-index in the original libraries.

```
index_1 ("0 1.0500, 2.4000, 3.5000");
```

- The input slew is taken from clk\_buf slew-index in the original libraries.

```
Seq. Arc clk->out
index_1 ("0 1.0500, 2.4000, 3.5000");
```

- The output load is taken from buf2 slew-index in the original libraries.

```
index_2 ("0 1.0500, 6.5000, 10.0000");
```

- The input slew is taken from buf3 slew-index in the original libraries.

```
Combinational arc in->out
index_1 ("0 1.0500, 1.4000, 4.5000");
```

- The output load is taken from buf5 slew-index in the original libraries.

```
index_2 ("0 1.0500, 6.5000, 10.0000");
```

**Note:** A value of 0 is always added to the characterization points.

### 9.1.8 Constraint Generation during Model Extraction

The extracted models need to be validated before they are transferred to other designers to be used for a top level analysis or optimization flow. For validation purposes, we need a subset of the original timing constraints and further a subset of these constraints is needed for fitting the model in a top level netlist.

The model extraction process will output a timing library and two constraint files containing the subset of the original constraint. One of these constraint files will be used for a standalone validation of the model compared to the original netlist. By default model.asrt and model.asrt.latchInferredMCP will be written in the CWD. If the do\_extract\_model - assertions test.asrt command is specified, then three assertion files are generated, named -test.asrt, test.asrt.latchInferredMCP, and top\_model.asrt. The other constraint file (top\_model.asrt) will be used when the extracted model will be stitched to a top level netlist and test.asrt will be used for standalone model validation.

Two separate constraint files are required for the following:

- Standalone validation will need all the context parameters of the design.
- STA or optimization flow when stitched to a top level environment, the context parameter will be automatically coming from the top level. So some of the constraints (in the above file) need to be filtered.

The following sections describe constraints in a model extraction flow:

- False Path Exceptions
- Multi-Cycle Path Exceptions

### ***False Path Exceptions***

If the `set_false_path` constraint is applied through some internal pins/ports, then those paths/arcs are removed during extraction.

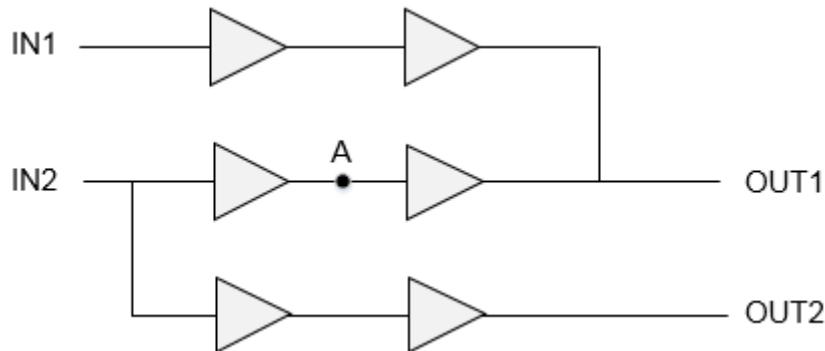
If the `set_false_path` constraint is applied between clocks or a clock and a port, then these paths are removed and these exceptions are saved in a `top_model.asrt` as well as `test.asrt` file.

### ***Multi-Cycle Path Exceptions***

If multi-cycle paths through pins/ports are applied, then during extraction the worst paths through them mapped to the IO ports are calculated and the exceptions through these IOs are dumped in the `top_model.asrt` file.

Consider the following design scenario with multi-cycle path exception applied at A.

**Figure 9-7 Multi-cycle path exception**



If there is no path from IN1/OUT1 and IN2/OUT2, the multi-cycle path will be pushed to IN2/OUT1. Since there are other paths going through these ports, multi-cycle paths cannot be pushed to IN2/OUT1 and cycle adjustment is done in delay values of the arcs. This will make the model context dependent.

When you set `timing_extract_model_disable_cycle_adjustment` global variable to `true`, the software will disable the cycle adjustment to make ETM context independent. You will have to manually apply the multi-cycle path at the top level while using this ETM.

**Note:** You should examine the constraint file ("top\_model.asrt"), as these constraints are just indicative of what is needed to be modeled at the top level. In certain situations, it may not be possible to extract all possible exceptions for top level due to limitations of liberty to model them correctly for a given path in case of conflicts.

■ set disable timing

If the `set_disable_timing` constraint is applied on any arc, then this arc gets disabled and is not used for the extraction purpose. Thus, such arcs are handled internally by the tool and are not written in any of the constraint files. Also, the path broken by them will not be extracted during ETM generation.

■ set case analysis

Constants applied/propagated by the `set_case_analysis` command can be written as a function attribute of pins in the ETM by setting the `timing_extract_model_case_analysis_in_library` global variable to `true`. This can be written in the ETM-validation constraint file generated by setting this global variable to `false`.

■ create\_clock

The pins/ports having `create_clock` definitions on them are preserved in the timing model. If a clock is created on some internal pin then the pin name needs to be modified to reflect the instance name of the extracted model. This is achieved using the same variable approach.

```
[get_pins "$ETM_CORE/pin1 $ETM_CORE/pin2 $ETM_CORE/pin3"]
```

When the extracted model is stitched to some top level netlist, then the clock definitions are supposed to come from the top level netlist itself. So that such clock definitions are not needed to be dumped in the top level constraint file, but need to exist in the assertion file for standalone validation.

■ create\_generated\_clock

The pins having the `create_generated_clock` defined on them are preserved in the extracted model as internal pins. As generated clocks are very much intended for the block itself and are not supposed to be coming from the top level, liberty provides syntax to define the generated clocks in the timing model. You can use the following use model:

```
generated_clock (gclk) {
 clock_pin : gclk ;
 master_pin : clk ;
 multiplied_by : 2 ;
}
```

While loading a model, the software names the generated clocks after their target pin names, so while dumping the generated clocks in the library the target pin name is modified to the clock name itself. This facilitates the same name of generated clock names in the original netlist and the extracted model. This name changing cuts down the need to modify other constraints (clock uncertainty/path exceptions) related to this clock. In this case it is not required to dump these generated clocks as a separate constraint.

■ set input delay/set output delay

The constraints like `set_input_delay` and `set_output_delay` lie in this category. Though these constraints can be applied on ports as well as some internal pins. Such constraints applied on some internal pins are of no use for the extracted models, as in an extracted model only interface timing is considered.

The constraints applied on the ports need not any change as the port will be preserved with the same name. So these constraints are dumped out in the assertion file for validation and do not have any impact on the generated ETM. These need not to be dumped out for the top level constraint file as in a top level the data must be coming to the port from some other top level register or port. So the input delay value is supposed to be the delay of path from top level register/port to this port.

■ `set_max_transition`/`set_max_capacitance`

If these constraints exist in the SDC file for block and the `timing_extract_model_consider_design_level_drv` global variable is set to `false`, then these `max_cap`/`max_trans` constraints (from SDC) will not have any impact on the generated model.

If the `timing_extract_model_consider_design_level_drv` global variable is set to `true`, then these constraints are considered while generating the timing model. The conservative value of `max_transition` defined at a library cell associated with a port or `set_max_transition` defined in the SDC is chosen for all the ports. Similarly, in case of `max_capacitance`, a conservative value defined at a library cell associated with a port or defined in the SDC is chosen for all the output ports.

**Note:** The `set_max_transition`/`set_max_trans` constraint defined in the SDC is always dumped in the side constraints file that is read during validation.

■ `set_load`/`set_resistance`/`set_annotated_transition`

The `set_load`, `set_resistance`, or `set_annotated_transition` constraints can be applied on pins as well as ports. The constraints applied on internal pins are handled in the extraction flow during delay calculation and are included in the model. But the constraints applied on the ports are treated as out-of-context environment and need to be written out in the constraints file to be read for model validation.

■ `set_annotated_delay`/`set_annotated_check`

The `set_annotated_delay` or `set_annotated_check` constraint is applied on the internal arcs as incremental or absolute values. The annotated delays/checks are handled in the extraction flow during delay calculation, so these are included in the model. This constraint is not written out in any of the constraint files.

■ `set_input_transition`/`set_driving_cell`

The `set_input_transition` or `set_driving_cell` constraints are applied only on ports. At the top level, the input transition at ETM pins depends upon the transition of the signal coming from the top-level circuitry. Similarly, the driving cell is needed only at the block level to replicate the actual driver of ETM at the top level. Hence, these are written only in the validation constraint file.

■ `set_clock_uncertainty`/`set_clock_latency`

By default, the clock uncertainty or clock latency constraints are written in the validation constraint file. But if the

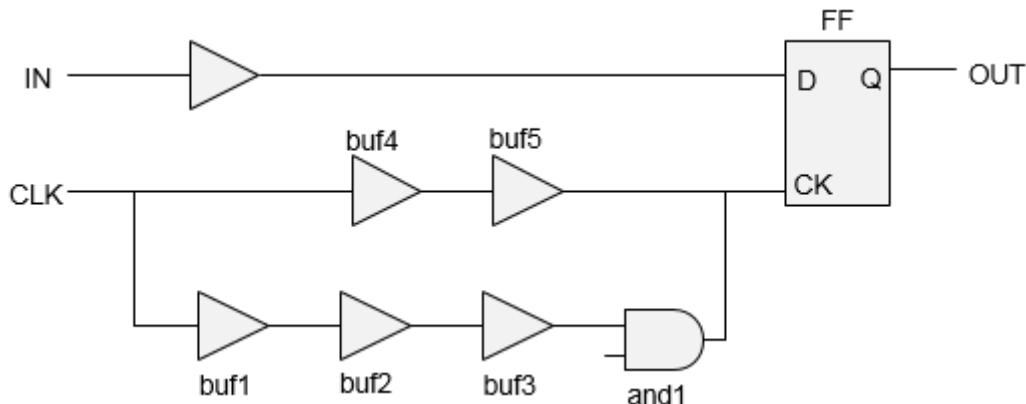
`timing_extract_model_include_latency_and_uncertainty` global variable is set to `true`, then these values will be taken care of during ETM characterization. And the delay tables in ETM are written with these values included.

### 9.1.9 Parallel Arcs in ETM

If a model is extracted in the BC-WC or OCV mode, then timing is handled using the early and late path analysis (as done with the `report_timing` command).

Consider the following design:

**Figure 9-8 Parallel arcs in ETM**



In the above design there are two paths CLK->IN (check path) and CLK->OUT (sequential path).

Here, the check (setup) path will be captured by the early path of clock (CLK->buf4->buf5->FF/CK). But the sequential path will be using the late clock path (CLK->buf1->buf2->buf3->and1->FF/CK). The extraction of early/late attributes is differentiated through the `min_delay_arc` attribute. So, in the extracted model for combinational/trigger/latency arc, both the rise/fall maximum and minimum arcs exist.

### 9.1.10 Latency Arcs Modeling

Latency arcs are between a generated clock and the respective master clock. The actual paths are considered during extraction. For example, consider a generated clock with - divide\_by 2. The edge relationship of master and generated clock will be “R->R” and “R->F”. The paths “F->F” and “R->F” arc will not be dumped, even if such paths are present in the design for data propagation.

In case of an ideal master clock, the arcs between master and generated clock source, is controlled through the following setting:

```
set timing extract model ideal_clock latency arc false (or true)
```

When set to `true`, the zero delay arc from the master clock to generated clock is considered in the extracted model. When set to `false`, this arc is not modeled.

### 9.1.11 Latch-Based Model Extraction

Extraction model handles transparent latches during model extraction based on the arrival times defined on input ports and the specified clock periods - whether a path from an input port to a latch causes borrowing or not.

The following points should be noted:

- If the path causes borrowing then path traversal should continue through the latch until either a non-borrowing latch or edge-triggered flip-flop is encountered. If the path does not cause borrowing, then path tracing should be stopped and a setup arc should be extracted relative to the closing edge of the first (interface) latch.
- The borrowing behavior of latches does not impact the extracted hold arc. So hold arcs should always be extracted relative to the close edge of the first interface register (latch or flip-flop) that is encountered while tracing paths from input ports.
- Borrowing can result in paths being traced through latches from an input port to an output port. In this case, a delay (comb) arc should be extracted from the input port to output port. In scenarios where path becomes more than 1 cycle, inferred multi-cycle paths are dumped in a separate file named as “model.asrt. latchInferredMCP”.
- When tracing paths from interface registers to output ports, transparent latches will be handled by tracing through borrowing latches backwards and generating a sequential delay arc from a clock port to an output port.

**Note:** As all such latch traversal arcs are true for setup analysis, there will be a `set false path -hold` statement in the assertion file for any such arcs. Please also note that “model.asrt. latchInferredMCP” file is indicative of what assertions you may need to be

used at the top level. The assertions should be audited before using multi-cycle paths, as they cannot be inferred in all the cases.

### **9.1.12 Model Extraction in AOCV Mode**

Tempus supports modeling of AOCV derated delays and stage weights in ETM. These stage weights and AOCV derates are used during the top level analysis with ETM by correctly computing the stage counts and derates of the paths related to ETM.

The AOCV analysis mode can be enabled by using the following command:

```
set analysis mode -aocv true
```

### **9.1.13 Stage Weight Modeling in ETM**

The stage weight modeling feature can be enabled by using the following command:

```
do extract model -include_aocv_weights
```

Using this command will enable the software to write stage weights on individual arcs.

The weights written in the ETM are not meant to derate the model. The model will be containing the derated delays. The stage weight extracted on arcs will be used to calculate the stage count of chip level paths going across the ETM.

The extraction of stage weight will be done by tracing the minimum stage count path on arc-by-arc basis. A path-based cell stage count will be printed on all sequential and combinational arcs. The extraction strategy for computing stage weight will be as follows:

The stage weights in the extracted model are dumped as user-defined attributes. This will require defining three attributes at the arc level.

```
define ("aocv_weight","timing","float");
define ("clock_aocv_weight","timing","float");
The first attribute will be used for combinational and trigger arc and will be
dumped at arc level
as below.
timing () {
 timing_type : combinational;
 timing_sense : positive_unate;
 aocv_weight : 4;
 .
 .
}
```

The check arcs will be dumped with two separate stage weights for data and clock:

```
timing () {
 timing_type : setup_rising;
 aocv_weight : 4;
```

```
clock_aocv_weight : 3;
. .
}
```

### 9.1.14 AOCV Derating Mode

The AOCV derating mode can be specified as path-based, graph-based, or none using the following setting:

```
set timing extract model aocv mode {graph_based | path_based | none}
```

The global values are explained below:

- `graph_based`: Delays in ETM are derated using the graph-based stage counts.
- `path_based`: Delay in ETM are derated using total path stage count of worst path between those pin pairs.
- `none`: Models derated delays that contain the effect of user-derate but does not contain effect of AOCV derates. The default is set to `none`.

Before setting this global variable to `graph_based` or `path_based`, it is mandatory that the AOCV libraries are read in and AOCV analysis is enabled (using the `set analysis mode -aocv true` setting).

### Merging Model with stage\_weight Attribute

The merged timing model contains minimum stage\_weight defined between various input library files, hence the merged timing model is pessimistic.

### Points to be Considered for AOCV-ETM Flow

The following points need to be considered for a block-level AOCV run:

1. If analysis of a block is performed in the standalone mode, then you need to specify the stage weight seen at the port(s). The applied stage weight affects the internal weight count of the block for the IO paths. If these weights are not applied, the pessimistic analysis will be done for the interfacing paths of the block (when block is top). Hence the ETMs will also have pessimistic numbers for such paths.
2. The same ETM may not be used for multiple instantiation at the top level, due to the following reasons:
  - Different stage counts at the ports (as seen from the top) is possible.

- Different critical paths between the ports for different instances is possible.
- You can use ETM with most conservative stage count for all the instances, but this will lead to pessimistic analysis.
3. The ETMs are written taking AOCV derates into account. At the top level, you need to have two types of AOCV derate tables:
    - Cell level AOCV tables
    - Design level AOCV tables for net
  4. While loading ETM for model validation the AOCV derates are already modeled in the ETM. In this case, you should turn off AOCV analysis mode.

If design level cell AOCV derates are used at the top level, then the block delays (as seen from the top) will have the AOCV effect twice - once during model extraction at the block level and second at the top level from design level cell AOCV tables. In case design level AOCV tables are being passed with ETM at the top level, then you should provide a cell level table for ETM with derating as 1 for all the possible stage counts and/or distance.

### 9.1.15 PG Pin Modeling During Extraction

The power-ground rails of a block are modeled as PG pins of a macro in the corresponding ETM. Since the software infers the information of power-ground rails from CPF/UPF, these files should be read to model PG-pins in the ETM.

To write PG pin information in a timing model, you can use the `do_extract_model -pg` parameter. The low power attributes can be written at the library level, cell level, and pin level in the ETM.

### PG Modeling in ETM

Consider the following scenarios:

#### ■ **Modeling of voltage maps**

Voltage maps definitions are typically modeled inside the Liberty dotlib format at library scope level. It specifies the voltage value associated with the power rail name (voltage names). At cell level pg\_pin groups are associated with voltage values using the voltage names.

```
voltage_map (VDD, 0.8);
voltage_map (VSS, 0.0);
```

#### ■ **Modeling of power/ground pins**

At the cell level, the pg\_pin groups will be modeled in the ETM in the following way:

```
pg_pin (VDD1) {
 voltage_name : VDD ;
 pg_type : primary_power;
}
pg_pin (VSS1) {
 voltage_name : VSS ;
 pg_type : primary_ground ;
}
```

The supported pg\_type in model extractions: primary\_power, primary\_ground, internal\_power, internal\_ground. If a power rail is the output of a power switch cell, it will be marked as internal\_power/ground.

■ **Associating power and ground pins to signal pin**

This information will be written for all the logic input/output/in-out pins that are written in the ETM. For internal pins that are preserved inside the ETM dotlib (e.g., a pin on which the generated clock was asserted), no such information will be retained.

```
pin (A) {
 ...
 related_power_pin: VDD1;
 related_ground_pin: VSS1;
}
```

For ports, that are written as pins in ETM, the associated pin (driver or receiver of a net) will be checked and printed in the same way as related\_power\_pin and related\_ground\_pin information.

### **ETM Merging Requirements for Power/Ground-Aware ETMs**

- Voltage maps from two libraries will be merged successfully as long as the voltage rail names are distinct.
- The pg\_pins will be merged successfully as long as power rail names and the pg\_type match.
- The related\_power\_pin and related\_ground\_pin will be merged as long as they are available in all the libraries and refer to the same power rail name.

#### **9.1.16 Extracted Timing Models with Noise (SI) Effect**

To generate a timing model with SI effects contained within a block, you can use the following flow for a block under consideration:

- Load the database and the relevant information for performing STA/SI analysis.
- Perform SI analysis, or if you have an incremental SDF from the previous SI run just load the incremental SDF.

- Run model extraction. During this phase, the incremental delays will be added to the computed base delays for characterization of the dotlib table. When the final dotlib is written, it contains the effect of the SI analysis in terms of characterized delays.

The block ETM can then be instantiated in the top level flow and the required file can be loaded for performing top level analysis with ETM.

### 9.1.17 Extracted Timing Models with SOCV

To write ETMs with variation data, set the `timing_extract_model_write_lvf` global variable to `true`. This will enable separate mean-sigma modeling of all the delay, constraint, and transition values.

When set to `false`, the product of sigma multiplier and the sigma value of delay/constraint/transition will be added (or subtracted) to (or from) the corresponding mean value and modeled in the ETM.

### 9.1.18 Merging Timing Models

The software supports merging library models through the `merge_model_timing` command. You can generate ETM in various modes and then merge them in a single library (in which all the modes are defined inside a merged group) that can be used for timing optimization during design flow. These mode\_group/modes are defined in merged ETM as mode\_definition/mode\_value.

To merge two ETMs (corresponding to funct and scan views), use the following command:

```
merge_model_timing -library_file funct_etm.lib scan_etm.lib -modes funct scan -mode_group funct_scan -outfile merged_funct_scan.lib
```

These modes are defined in a merged ETM as:

```
mode_definition (funct_scan){
mode_value (funct){
}
mode_value (scan){
}
```

In a merged ETM, the arcs corresponding to different modes are defined as given below:

```
timing() {
mode(funct_scan," funct ");
min_delay_arc : "true" ;
related_pin :" SI_ClkIn " ;
}

timing() {
mode(funct_scan , " scan ");
```

```
related_pin :" EJ_TCK ";
```

To read arcs corresponding to funct (scan) mode, use set\_mode funct (scan). If the set\_mode command is not issued, the arcs corresponding to both the modes will be reported by the report\_timing command.

The single mode can be specified using the set\_mode command. An error is issued if more than one mode is passed as an argument to this command. The mode merging enables you to do the analysis in various modes on a single shell.

Some considerations while merging models are given below:

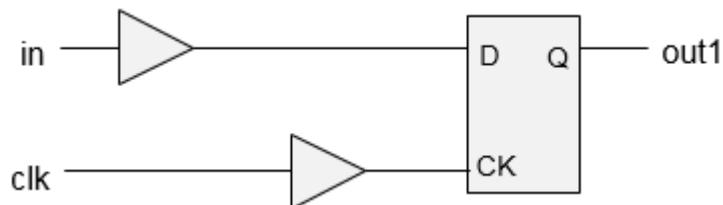
- All the input timing models should have the same cell name.
- If the timing arcs are not comparable i.e., the number of index values differ, then they are written as separate timing arcs with separate mode definitions.
- Boundary pins of single mode libraries should be equivalent in number and name while merging. An error message will be flagged if that is not the case.
- If you have generated clocks with the same name, as well as same definition in ETM libraries, then just this clock is preserved in the merged library, without issuing an error message.
- If you have generated a clock with the same name, but different divided\_by/multiply\_by/master\_pin/clock\_pin definition, then an error message is displayed stating that “genclk is defined with different specification in two modes, merging is not possible”.
- If there is a mismatch in timing arcs among the input libraries, then the merged library will contain the union of all the timing arcs with the respective mode. For example, if the first library has only setup arcs and the second library has only hold arcs then the merged library will have both setup and hold arcs with respective modes appended.
- If there are mismatches in internal pins among input libraries, that is, if a particular internal pin is missing in one or more libraries then that internal pin and its timing arcs will be part of a merged library with respective mode.
- For both maximum and minimum DRV, you can use the most conservative value. The minimum value from maximum DRVs will be used and the maximum value from minimum DRVs will be used in the merging.
- When same modes are specified in the modes list for two libraries, for example, merge model timing -library\_files "setup.lib hold.lib" - modes "M M" -mode\_group "MG", then the arcs from the two libraries will be assigned the same mode, that is, “M” while merging.

### 9.1.19 Limitations of ETM

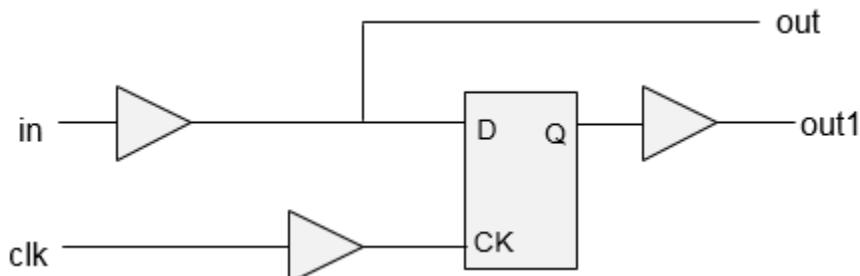
The limitations of ETM are as follows:

- In path-slew propagation mode, the slew/delay dependency on side inputs will be lost.
- Consider the following two cases:
  - clk-in check ac will depend on the slew at the D and CK pins of the register. Hence these check arcs are modeled as 2D-check arcs, as expected.
  - In the second case, the slew at D pin will depend upon the loading at the out port. Hence check arc value will essentially depend upon the slews at the D and CK pins and the out load as well. But currently model extraction does not support 3D modeling of check arcs.

**Figure 9-9 Check arc**



**Figure 9-10 Load dependent check arc**



The following timing report shows setup\_rising present in the ETM:

```

Path 1: VIOLATED Hold Check with Pin cppr_block/CLKIN
Endpoint: cppr_block/DATAIN (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of 'clk2'
Path Groups: {clk}
Other End Arrival Time 0.000
+ Hold 5.000
+ Phase Shift 0.000
= Required Time 5.000
Arrival Time 0.500
Slack Time -4.500

```

## Tempus User Guide

### Timing Model Generation

---

```

Clock Rise Edge 0.000
+ Input Delay 0.500
= Beginpoint Arrival Time 0.500
Timing Path:

Pin Arc Cell Edge Arrival
Time

DATAIN -> DATAIN ^ ^ 0.500
cppr_block/DATAIN cppr_block ^
 0.500

```

The following timing report shows no setup\_rising in the ETM:

```

Path 1: VIOLATED Hold Check with Pin cppr_block/CLKIN
Endpoint: cppr_block/DATAIN (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of 'clk2'
Path Groups: {clk}
Other End Arrival Time 0.000
+ Hold 5.000
+ Phase Shift 1.800
= Required Time 6.800
Arrival Time 0.500
Slack Time -6.300
Clock Rise Edge 0.000
+ Input Delay 0.500
= Beginpoint Arrival Time 0.500
Timing Path:

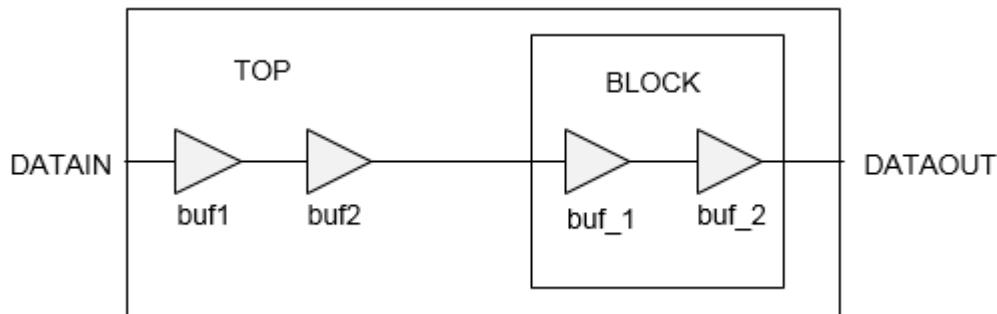
Pin Arc Cell Edge Arrival
Time

DATAIN -> DATAIN ^ ^ 0.500
cppr_block/DATAIN cppr_block ^
 0.500

```

- When ETM is read at the top, the information about the stage count inside the block is lost. Hence for paths coming in/out of the block, incorrect stage count is calculated while applying AOCV derate (on the top level instances which fall in this path).

**Figure 9-11 ETM at the top level**



## Tempus User Guide

### Timing Model Generation

---

When BLOCK netlist is read at the top level, the stage count for buf1 and buf2 is 4, but when an ETM of BLOCK is read, their stage count becomes 2.

The following example shows a timing report with BLOCK netlist:

```
Path 1: MET Late External Delay Assertion
Endpoint: DATAOUT2 (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of '@'
Path Groups: {clk}
Other End Arrival Time 0.000
- External Delay 0.010
+ Phase Shift 3.600
= Required Time 3.590
- Arrival Time 0.354
= Slack Time 3.236
Clock Rise Edge 0.000
+ Input Delay 0.000
= Beginpoint Arrival Time 0.000

Aocv Aocv Cell Pin
Stage Derate
Count

 DATAIN
4.000 1.169 BUF buf2/Y
4.000 1.169 BUF buf_2/Y
4.000 1.169 BUF BLOCK/buf7/Y
4.000 1.169 BUF BLOCK/buf8/Y
5.000 1.167 DATAOUT2
```

The following example shows a timing report with BLOCK ETM:

```
Path 1: MET Late External Delay Assertion
Endpoint: DATAOUT2 (^) checked with leading edge of 'clk'
Beginpoint: DATAIN (^) triggered by leading edge of '@'
Path Groups: {clk}
Other End Arrival Time 0.000
- External Delay 0.010
+ Phase Shift 3.600
= Required Time 3.590
- Arrival Time 0.325
= Slack Time 3.265
Clock Rise Edge 0.000
+ Input Delay 0.000
= Beginpoint Arrival Time 0.000

Aocv Aocv Cell Pin
Stage Derate
Count

 DATAIN
2.000 1.173 BUF buf2/Y
2.000 1.173 BUF buf_2/Y
3.000 1.171 cppr_block BLOCK/DATAOUT2
```

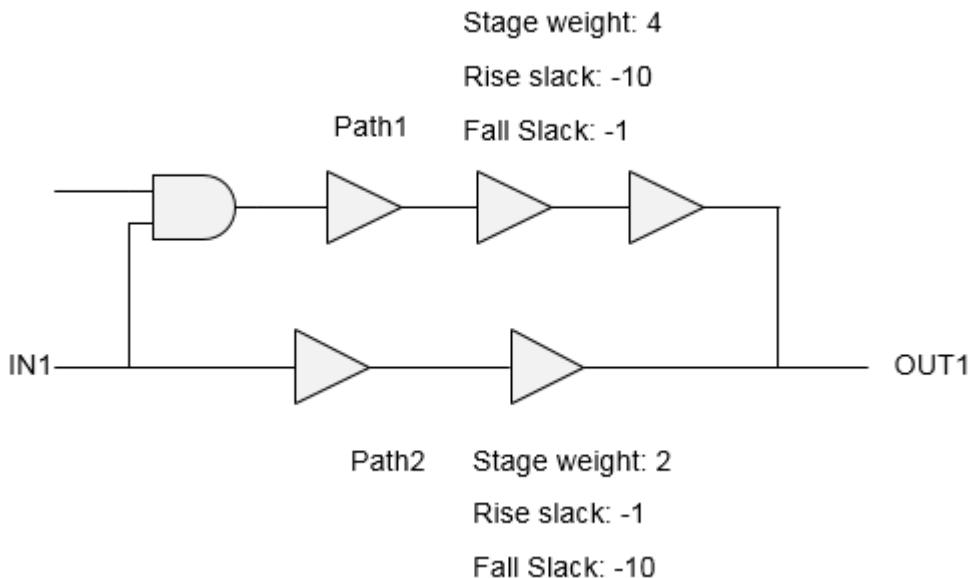
```
4.000 1.169 DATAOUT2
```

---

While modeling any arc, the rise and fall constraints are written separately. While modeling stage weights, the worst of rise/fall stage weight is written in ETM.

Consider the following example showing two paths with given stage weight and rise/fall slack values. The rise and fall arcs will be modeled corresponding to Path1 and Path2, respectively, between IN1/OUT1. The single value of stage weight (4 in this case) will be written in ETM. Ideally, the fall constraint should have the stage weight of 2 in this case, but due to the limitation, the stage weight becomes 4 adding to the pessimism in analysis.

**Figure 9-12 Modeling stage weights**



```
pin (DATAOUT) {
 direction : output ;
 timing() {
 timing_type : combinational ;
 timing_sense : positive_unate ;
 cell_rise (lut_timing_2){
 values(\n " -10, -20", \
 " -30, -40" \
);
 }
 cell_fall (lut_timing_2){
 values(\n " -10, -20", \
 " -30, -40" \
);
 }
 }
}
```

```
);
}
aocv_weight : 4.0000;
related_pin :" CLKIN ";
}
```

### 9.1.20 Validation of Generated ETM

The extracted models should be validated before stitching to the top level for their accuracy and coverage. This block-level validation can be achieved by comparing the interface timing of the extracted timing models against the original gate-level netlist, by using the write model timing and compare model timing commands.

A brief description of the ETM commands used in a validation flow is given below:

#### Commands Used in Validation Flow

##### ■ **do\_extract\_model**

The ETM is generated by using the do\_extract\_model command. The use model is given below:

```
do_extract_model top.lib -verilog_shell_file test.v -
verilog_shell_module test-assertions test.asrt
```

The following are the outputs:

- ❑ verilog\_shell\_file (test.v): Verilog having an ETM instantiated in it. In case the block constraints file has the set\_driving\_cell command, then the cell used there will be instantiated in test.v.
- ❑ verilog\_shell\_module(test): Module name of test.v.
- ❑ top.lib: The resulting ETM.
- ❑ test.asrt: The resulting exception file to be used in a validation flow.

##### ■ **write\_model\_timing**

The write model timing command writes a report on the interface timing of a specified netlist. The use model of the command is as follows:

```
write_model_timing -type slack ref.rpt
```

The report ref.rpt contains the following timing view properties to capture the timing characteristics of the design being extracted as a timing model:

- ❑ Slack or Arc Value: Reports the worst-case slack or arc value for each path from input port to clock, from clock to output port, and from input port to output port.

- ❑ Transition Time: Reports the actual transition time at each port for the four delay types: min\_fall, min\_rise, max\_fall, and max\_rise.
  - ❑ Capacitance: Reports the maximum total (lumped) capacitance at each port, and if available, the effective capacitance.
  - ❑ Design Rules: Reports all the design rules that apply to each port, including the maximum capacitance, minimum capacitance, maximum transition time, maximum fan-out (for input ports), and fan-out load (for output ports).
- **compare\_model\_timing**

The `compare_model_timing` command compares two reports generated by the `write_model_timing` command. If the timing parameter values in the two files are the same or within the specified tolerance, then the result is pass, or otherwise fail. The use model is as follows:

```
compare_model_timing -ref ref.rpt -compare comp.rpt -outFile final.rpt
percent_tolerance 3 -absolute_tolerance [expr 0.30/$timeUnit]
```

The `compare_model_timing` report shows the comparison results for individual paths, ports, and timing parameters. The resulting comparison report has the same sections as the interface timing report: slack, or arc value, transition time, capacitance, design rules. There are two tolerance settings as follows:

- ❑ Absolute tolerance: Indicates the absolute acceptable difference between compared parameters in library time units.
- ❑ Percentage tolerance: Shows the percentage of acceptable difference between compared parameters.

A path is flagged as a failure by the `compare_model_timing` command if it violates both the absolute and percentage tolerance values.

## Validation Flow- SMSC

- **Step1:** Generating timing model (ETM) (using the `do_extract_model` command) and the `write_model_timing` command output from the original session, as described below.

```
read_design
do_extract_model test.lib -verilog_shell_file test.v -
verilog_shell_module test -assertions test.asrt
write_model_timing -type slack ref.rpt
exit
```

- **Step2:** Loading the generated ETM, generating the `write_model_timing` report for the ETM session, and then comparing reports with the reference session using the `compare_model_timing` command as described below.

```
read_verilog test.v
read_lib test.lib
set_top_module test
read_sdc test.asrt
write_model_timing -type slack comp.rpt
compare_model_timing -ref ref.rpt -compare comp.rpt -outFile final.rpt
-percent_tolerance 3 -absolute_tolerance 0.003
```

**Note:** It is recommended to set

timing\_prefix module\_name with library genclk global variable to false, and timing library genclk use group name to true. For more information, refer to section on Generated Clocks.

At the end of a complete validation flow, the following files will be written:

- Files generated by do\_extract\_model (test.lib, test.v, test.asrt)
- Interface timing (write\_model\_timing) report from netlist session (ref.rpt)
- Interface timing (write\_model\_timing) report from ETM session (comp.rpt)
- Comparison (compare\_model\_timing) report of the above two files (final.rpt)

## Validation Flow - MMMC

- **Step1:** In MMMC configuration, only one view should be active before running the do\_extract\_model command, which is done by using the set\_analysis\_view - setup {\$viewName} -hold {\$viewName} command.

The model is extracted using the do\_extract\_model command with the following set of commands:

```
set viewList [all_analysis_view]
foreach viewName $viewList {
 set_analysis_view -setup {$viewName} -hold {$viewName}
 read_spef -rc_corner $rcCorner.spef
 file mkdir $viewName
 do extract model -view $viewName $viewName/test.lib -assertions \
 $viewName/test.asrt -verilog_shell_file $viewName/test.v \
 -verilog_shell module test
 write_model_timing -view $viewName -type slack -verbose $viewName/
 ref.rpt
}
exit
```

This will create a separate directory for each view in \$viewList, and files from each view will be written in the corresponding directory. The generated ETM and the write\_model\_timing report from the complete netlist will be saved in the corresponding (\$view) folder.

- **Step2:** For validation of ETM generated from multiple views, the following syntax can be used to read the ETMs from their corresponding directory.

```
foreach viewName $viewList {
 free_design
 read_verilog $viewName/test.v
 read_lib $viewName/test.lib
 set_top_module test
 read_sdc $viewName/test.asrt.$viewName
 write_model_timing -type slack -verbose $viewName/comp.rpt
 compare_model_timing -ref $viewName/ref.rpt -compare $viewName/comp.rpt
 -ignore_view -outFile $viewName/final.rpt -percent_tolerance 2 -
 absolute_tolerance 0.003
}
exit
```

It is recommended to set timing prefix module name with library genclk global variable to false, and timing library genclk use group name to true.

For more information, refer to [Generated Clocks](#).

At the end of a complete validation flow, the following will be written out in each view-specific directory for the corresponding view:

- ❑ Files generated by do extract model (test.lib, test.v, test.asrt.\$view)
- ❑ Interface timing (write model timing) report from netlist session (ref.rpt)
- ❑ Interface timing (write model timing) report from ETM session (comp.rpt)
- ❑ Comparison (compare model timing) report of above two files (final.rpt)

### 9.1.21 Auto-Validation of ETM

Tempus provides an automated validation flow for ETM, in which all the validation steps mentioned in the previous steps can be run automatically. This can be done by using the do extract model -validate parameter.

**Note:** When the -validate parameter is used, the existing validation directory will be removed and a new directory will be created. It is recommended to specify different directories for the output ETM file and auto-validation reports.

This flow will write out the required write model timing and compare model timing reports in \$val\_dir. At the end of the auto-validation, the shell with block netlist is retained. The comparison of both setup and hold checks is performed by the auto-validation.

The following summary is generated at the end of the auto-validation:

---

Total_fail   Slack_Fail   Cap_Fail   Trans_Fail   DRV_Fail   checkType
------------------------------------------------------------------------

5		2	1	0	2	setup
2		1	0	0	1	hold

### 9.1.22 ETM Extremity Validation

The ETM models the timing arcs in the design for a range of discrete input-slews/output-loads. The slew/load asserted using the SDC at the block-level during ETM characterization represents the anticipated context of the ETM instance at the top-level. However, the current validation of the ETM is done only for the input-slews/output-loads that have been asserted on the design during the ETM extraction. Therefore, the current validation of ETM is incomplete in the sense that the validation is not being done for a complete range of slew/load points for which ETM has been characterized.

Additionally, in GBA mode the problem of ETM validation is computationally more difficult. In GBA mode, the assertion of a slew on a particular input port may have an impact on the timing of a path starting from some other input port. Therefore, different combinations of input slews at the input ports will result in different timing behavior of the block in GBA mode. Since there can be exponentially large number of combination of slews at the input ports, validation of ETM in GBA mode is computationally more difficult.

The “Extremity-Validation” of ETM (also referred as EV\_ETM) validates ETM at the minimum/maximum value of the slew at the input ports and minimum/maximum value of the output load at the output ports, thereby validating the ETM to a greater extent of the possible scenarios at the top. The minimum/maximum slew at the input port is defined as the minimum/maximum slew for which that input port has been characterized in the ETM library. Similarly, the minimum/maximum load at the output port is defined as the minimum/maximum load for which that output port has been characterized in the ETM library.

The minimum/maximum value of the slew/load will yield four corners of the ETM context, as given below:

- Corner 1 (Fast): the minimum slew at the input ports and the minimum load at the output ports
- Corner 2: the minimum slew at the input ports and the maximum load at the output ports
- Corner 3: the maximum slew at the input ports and the minimum load at the output ports
- Corner 4 (Slow): the maximum slew at the input ports and the maximum load at the output ports

The validation flow in exhaustive mode is the same as that in non-exhaustive mode, that is,

- do\_extract\_model

- write model timing at the block level
- Read ETM at top level
- write\_model\_timing at the top level
- compare model timing

This flow is controlled by the

timing\_extract\_model\_exhaustive\_validation\_mode global variable.

The additional files generated in the EV-ETM (but not generated in normal validation) will be stored in the current working directory by default. However, the location to save these files can be controlled by the timing\_extract\_model\_exhaustive\_validation\_dir global variable. These files are written as hidden files.

Also note that the timing\_extract\_model\_exhaustive\_validation\_dir global variable should point to the same directory at the block and top level. When this global variable is not used at the block level, then the EV-ETM files (in step 1 and 2 above) will be dumped in the working directory at the block level. If the working directory at the top level is different than that at the block level, then at the top level

timing\_extract\_model\_exhaustive\_validation\_dir global variable should point to the block level working directory.

The compare\_model\_timing command will compare the write\_model\_timing reports written at the block level with the corresponding top-level report. The output of compare\_model\_timing command, corresponding to the four corners is written to a file named: <output\_file\_name>\_ev - where <output\_file\_name> is the name of the output-file specified using the compare\_model\_timing command. The results of four different corners are written in four different columns as shown below:

```
#Slack(SS)/Status Slack(SF)/Status Slack(FS)/Status Slack(FF)/Status Transition
Arc-Type From To

7.511[7.511]/PASS 7.511[7.511]/PASS 7.466[7.466]/PASS 7.466[7.466]/PASS rise/rise
setup CLK1 CLOCK1
9.666[9.665]/PASS 9.666[9.665]/PASS 9.336[9.321]/PASS 9.336[9.321]/PASS fall/rise
setup CLK3 CLOCK2
```

In this mode, there is no change in the use models of the do\_extract\_model, write\_model\_timing, and compare\_model\_timing commands. The behavioral difference (between default validation and EV\_ETM) is the creation of additional four files corresponding to the four corners.

If the slew propagation is set to worst, then the do\_extract\_model and write\_model\_timing commands will issue a warning.

Currently, the extremity validation feature is not supported with auto-validation. You will need to run the two-step validation flow in this mode.

### **9.1.23 Limitation/Implications of EV-ETM**

The EV-ETM flow has the following limitations/implications:

- The runtime of validation will appreciably increase.
- The EV-ETM will be supported only in path-based mode and not in the worst-case mode.
- The EV-ETM, though validates the ETM at the corners, cannot be considered as fully rigorous. For example, the proposed methodology does not validate the timing obtained by interpolation of the tables characterized in ETM.

### **9.1.24 Ability to Check Timing Models**

The `do_extract_model -check` parameter command checks the extracted timing model for any possible Liberty-related issue. When specified, the command displays detailed error and warning messages and their summaries.

## 9.2 Boundary Models

- 9.2.1 [Overview](#) on page 152
- 9.2.2 [Boundary Model Flow](#) on page 152
- 9.2.3 [Hierarchical Flow with Boundary Model](#) on page 153
- 9.2.4 [Clock Mapping in Top Level Run with Boundary Models](#) on page 157
- 9.2.5 [Recommendations for Generating Accurate Boundary Models](#) on page 158
- 9.2.6 [Glitch-Specific Boundary Models](#) on page 159
- 9.2.7 [Boundary Models in MMMC Mode](#) on page 163
- 9.2.8 [Save-Restore of Hierarchical Analysis](#) on page 163

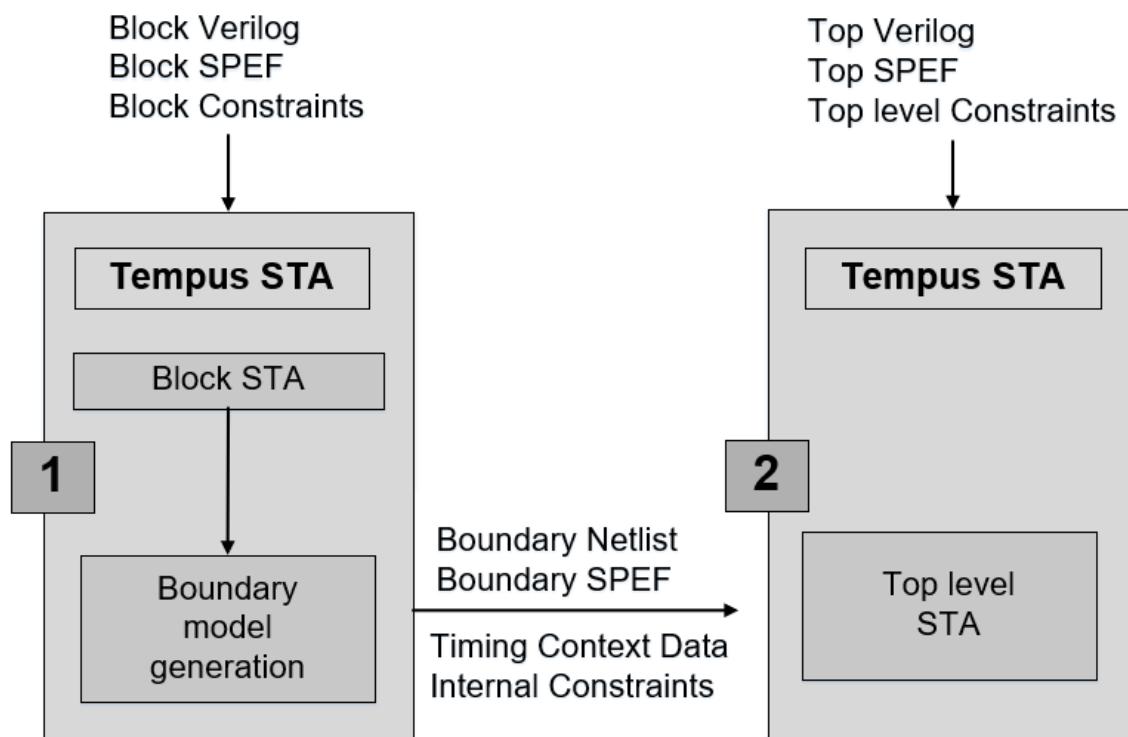
### 9.2.1 Overview

A boundary model for a module provides sufficient details of its interface paths in order to accurately analyze these paths for timing (including path-based analysis) when loading the model into timing analysis of a higher-level module. The model includes timing context information that improves the accuracy of the model, for example, the timing windows of attacker nets that are not on interface paths for SI analysis.

### 9.2.2 Boundary Model Flow

The following figure shows the flow for creating and using a boundary model.

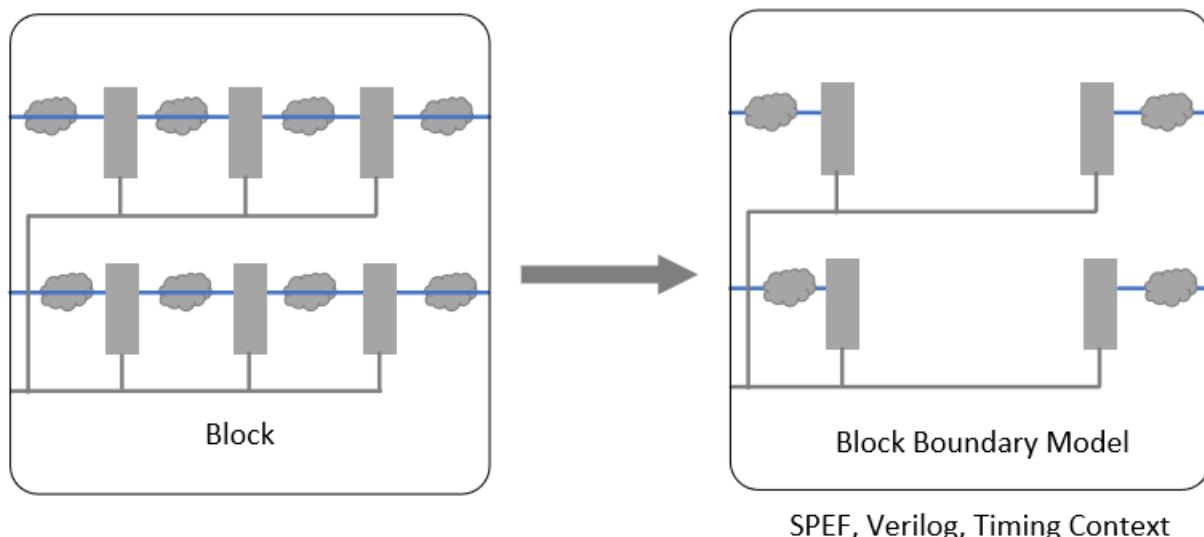
**Figure 9-13 Boundary Model Flow**



The first step is to perform a full timing analysis on the lower-level module. This analysis is done using block-level constraints for the module. After the analysis is complete, the boundary model can be generated.

The following figure shows a typical block and its boundary model.

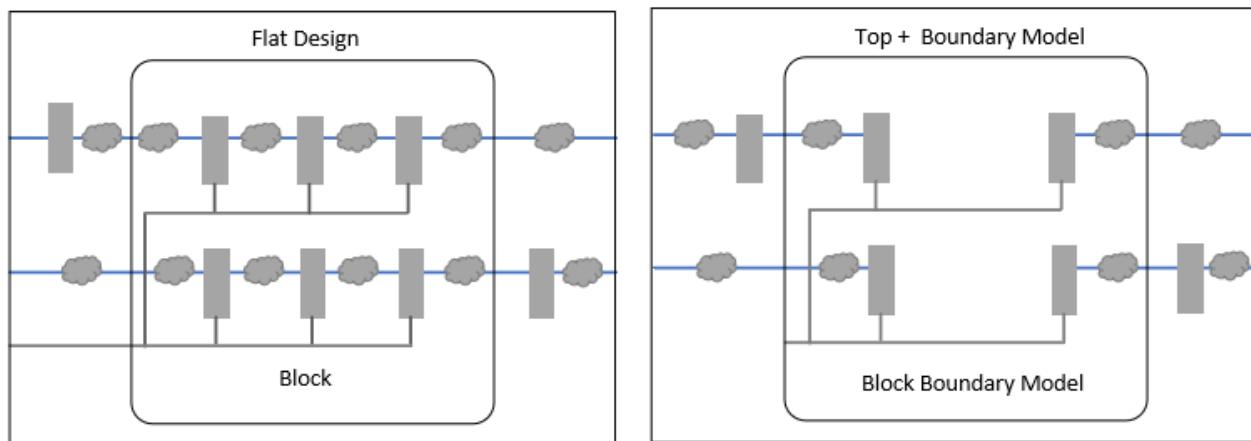
**Figure 9-14 Block with the generated boundary model**



This model can be subsequently loaded into a timing session for a higher-level module. The higher-level session is timed using flat constraints for those constraints that affect the interface paths of the instances of all the boundary model blocks.

The following figure shows the top level design with a block flat design vs block boundary model:

**Figure 9-15 Top level design with flat design vs block boundary model**



### 9.2.3 Hierarchical Flow with Boundary Model

The boundary model hierarchical flow is described below.

## Creating Boundary Models

For creating a boundary model of a block, you can use the `create_boundary_model` command.

To ensure that the correct context information is saved in the boundary model, you should make the following setting before running the `update_timing` command:

```
set timing_full_context_flow 1
```

The block run script can be modified as follows:

```
set timing_full_context_flow 1
read_verilog block.v
read_libs <>
set_top_module block
read_spef block.spef
read_sdc block.sdc
update_timing
create_boundary_model -dir bm
```

## Boundary Model Usage

In the top-level timing run, the boundary model can be loaded (instead of the full block Verilog and SPEF) by using the `read_boundary_model` command before the `set_top_module` command is run. The boundary models can be loaded for multiple blocks.

The following example shows a sample top run script for flat timing analysis:

```
read_verilog 'top.v block.v'
read_libs <>
set_top_module top
read_spef 'top.spef block.spef'
read_sdc top.sdc
update_timing
```

This script can be modified for top analysis with boundary models, as shown below:

```
read_verilog top.v
read_libs <>
read_netlist top.v -top top
read_boundary_model -dir bm
set_top_module top
read_spef top.spef
```

```
read_sdc top.sdc
update_timing
```

It may be noted that none of the higher-level SPEF files should contain parasitics for the boundary model block, because this will lead to extra parasitics being present on the primary I/O nets of the block instances. Therefore, for performing timing analysis with boundary models you must use the hierarchical SPEF.

If both the boundary model and Verilog of a block are read at the top level, then the model netlist will override the block netlist. This run will have a higher run time and memory usage as compared to the run with only block boundary models - with the same accuracy. Hence, it is recommended to remove the block Verilog from the top run script.

The non-default true setting of the `timing_full_context_flow` global variable in the top run will increase the run time/memory usage. Hence, this should be used in the top run, only if you need to create a scope from this run.

## Boundary Model Contents

A boundary model consists of the following:

- Netlist for a model in a binary form:

- Interface paths

The logic which constitutes the interface paths of a block are saved in the boundary model netlist. This includes the paths from the primary input ports to the sequential input pins, from the sequential output pins to the output ports, and from the input port(s) to the output ports. The interface paths are timed in the top level analysis.

- Extended fanin

The slew/arrival and the graph-based analysis (GBA) slack at an endpoint depends on its full fanin cone, therefore, the cone is also saved in the boundary model for an accurate top level timing analysis. The extended fanin logic is timed during the top level analysis.

- Internal attackers

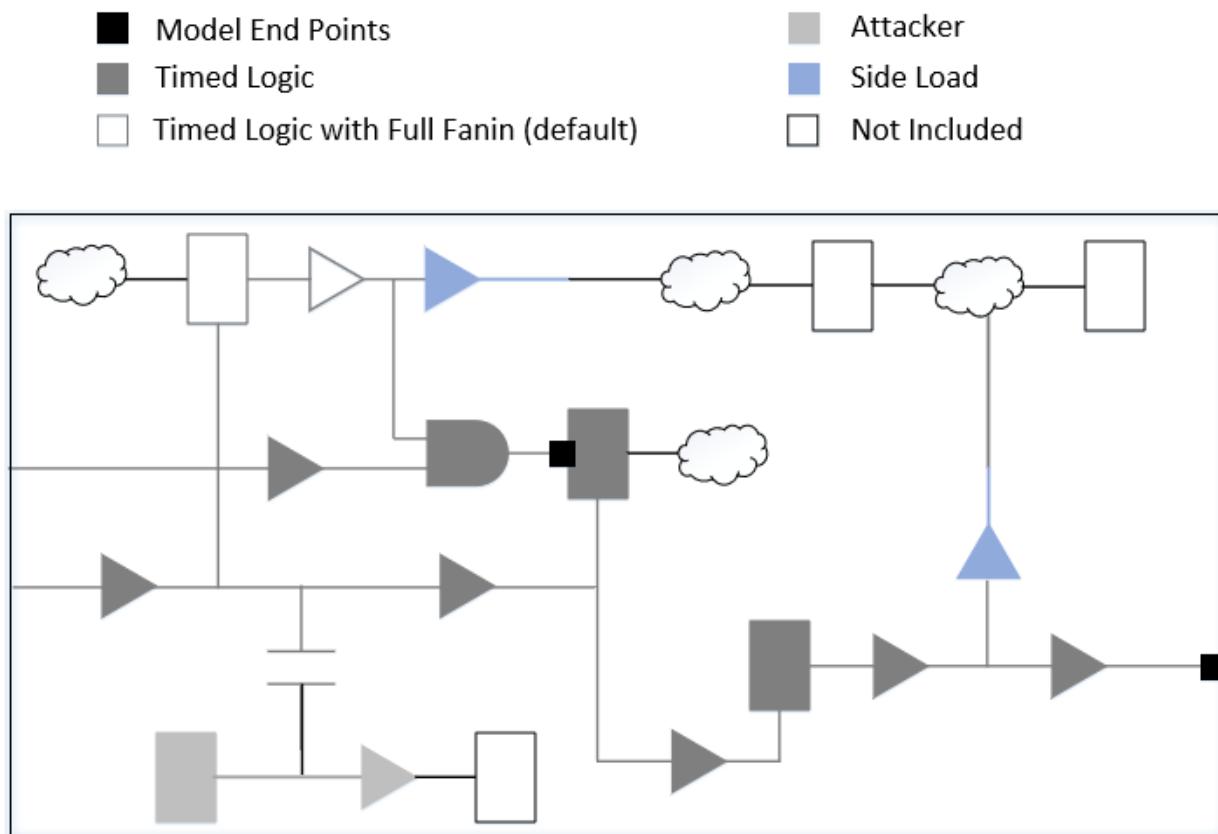
Internal path nets, that are SI attackers to the interface paths nets, along with their slew and timing windows (TW) are preserved in the boundary model. All the leaf driver and receiver instances of such nets are also included. These attacker nets are not timed in the top level analysis but their SI impact, as per the stored slew and timing windows adjusted for clock latencies, is taken into account for better accuracy.

- Side load receivers

The receiver instances on the interface path nets that are not part of the interface logic are included.

The following figure shows an example of a netlist saved for a boundary model of a simple block.

**Figure 9-16 Saved netlist for a boundary model**



The netlist contains the following data:

- Parasitics for the nets in the model in the SPEF format.
- The SDC constraints for elements on block internal paths. These constraints are not used by the top-level timing but are used if a block scope is generated for the boundary modeled block in the top-level session.
- Timing context data. Several types of timing data are saved to improve the accuracy of the model. Some of these are:
  - Timing windows and slew data for non-interface attacker nets.

- ❑ Clock definitions. The definitions of all the clocks used in the block-level timing is stored in the model, in order to automatically map block-level clocks to the top-level clocks. This mapping is required for timing window application.
- ❑ Constant values on pins of sequential instances that are not part of a boundary path. These constants can affect timing results of sequential instances and are therefore maintained in the boundary model. The extra constants on non-extended fanin pins are also maintained in the boundary model.

#### **9.2.4 Clock Mapping in Top Level Run with Boundary Models**

The context that gets saved in a boundary model is with respect to the clocks defined in the block run. The block level clocks, whose definition is also saved in the boundary model, are mapped to the top level clocks for the application of context data in the top run.

You can specify the mapping for some or all the clocks for each block instance by using a clock mapping file. You can specify a clock map file using the `read_boundary_model -clock_map_file` parameter.

The software attempts the auto clock mapping for clocks that do not have any user-specified mapping. The user-specified clock mapping settings have higher precedence over auto inferred clock mapping.

The auto clock mapping rules are given below:

- To map a clock ‘clkA’ defined at pin ‘pinA’ at a block:
  - ❑ The software will identify the source pin of clock clkA at block level, that is, pinA.
  - ❑ In the top level run with boundary model, the software will examine the clocks at pinA of the block, that is, at block/pinA.
    - Top level clock at block/pinA having waveform matching with clkA will be mapped to clkA.
    - If multiple clocks with matching waveforms are found, then the mapping of clkA is dropped due to ambiguity.
  - ❑ If no match is found, a check is made for the top-level clock with the same name (irrespective of clock source):
    - If a top level clock with name ‘clkA’ exists and has the same definition as block level ‘clkA’, then the mapping is successful.
    - If a top level clock with name ‘clkA’ exists and has a definition different from the block level ‘clkA’, then the mapping is ignored.

- Mapping virtual clocks:
  - Block virtual clocks are mapped only to the top virtual clocks.
  - Clocks with the same name/waveform will be mapped.
- Mapping generated clocks:
  - Block generated clocks will be mapped only with the top generated clocks, and cannot be mapped with top master clock.
  - Block master clock can be mapped with the top generated clocks.

The details of all the mapped and unmapped clocks are written in files named - blkname\_clock\_map.txt and blkName\_unmapped\_clocks.txt respectively, where blkName is the name of the block module. These files are written for every boundary modeled block. If a block is multiply instantiated at the top level, then the software writes mapped and unmapped clocks corresponding to each instance of a block.

If a block-level clock is not mapped to the top-level clock, then the timing windows (saved in the boundary model context), which reference the unmapped clock, are treated as infinite windows.

In addition to finding the corresponding clock, the timing window application applies a latency adjustment for a better correlation between the saved timing windows and the timing windows present in a full flat run. This latency adjustment is to account for the difference in clock arrival time between the block-level run and the top-level run. This is based on the difference in the arrival time between the matching clock phase on the block instance pin and the arrival time seen in the block-level run. This latency adjustment is made for each boundary model instance and clock, when there is a matching phase found. No latency adjustment is done where mapping is done by the clock name.

### 9.2.5 Recommendations for Generating Accurate Boundary Models

The timing correlation between a fully flat higher level run and a higher level run using one or more boundary models is expected to be close. However, the following factors can negatively affect the correlation:

- The timer settings between the lower-level timing run that generate the model and the higher-level run which loads the mode should be consistent.
- The SDC constraints applied in the lower-level timing run and the constraints that are applied to instances of the model in the higher level timing run should be consistent.
- Multiple SI passes: The timing windows in the boundary model context data are the timing windows used in the final SI delay calculation pass in the block level run. These

timing windows will be used for every SI delay calculation pass in the top-level run. If there are multiple SI passes, the timing window convergence can differ between the full flat top-level run and the top-level run with the model. Hence, it is recommended to use two SI iterations in both the block and top runs for good correlation.

- Clock Mapping: A successful mapping of all the block clocks is necessary for performing timing correlation between the top flat vs top boundary model runs. If there are unmapped clocks, you can use the `set_si_mode -use_infinite_TW true` command setting for the block as well as top runs for correlation purpose. If clock mapping is not successful and this setting is not used, then the top run with the boundary model will be pessimistic compared to the top flat run.
- Power Supply Voltage: If the power supply voltage setting is different in the block and the top CPF, then the cell delay and slew will change in two runs. This difference affects the slew and timing window calculation of the non-interface attacker nets. Due of this timing window difference, the top scope STA run can either get pessimistic or optimistic as compared to a full flat run.

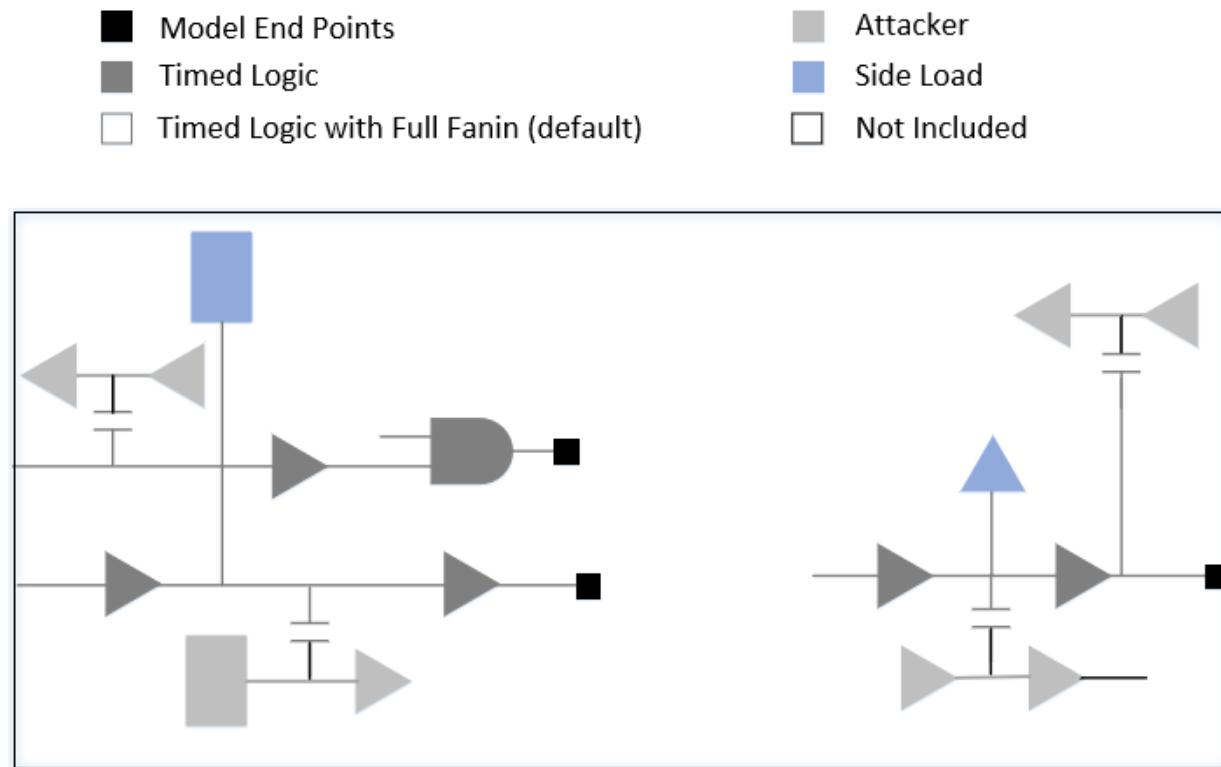
### 9.2.6 Glitch-Specific Boundary Models

To improve efficiency of glitch-only analysis, a glitch-specific model can be created using the `create_glitch_boundary_model` command.

For glitch-only analysis, the software will keep N-levels (default is 2 levels) of logic (controlled by the `create_glitch_boundary_model -max_stages` command parameter) from the boundary along with the RCs connected to any of those nets. If a multi-stage cell is encountered, then the noise will not propagate through the logic and multi-stage cells will be considered as path end or start points.

The following figure shows an example of a netlist saved for a boundary model of a simple block for a maximum of 2 levels of glitch stages.

**Figure 9-17 Saved netlist for a boundary model with 2 levels of glitch stages**



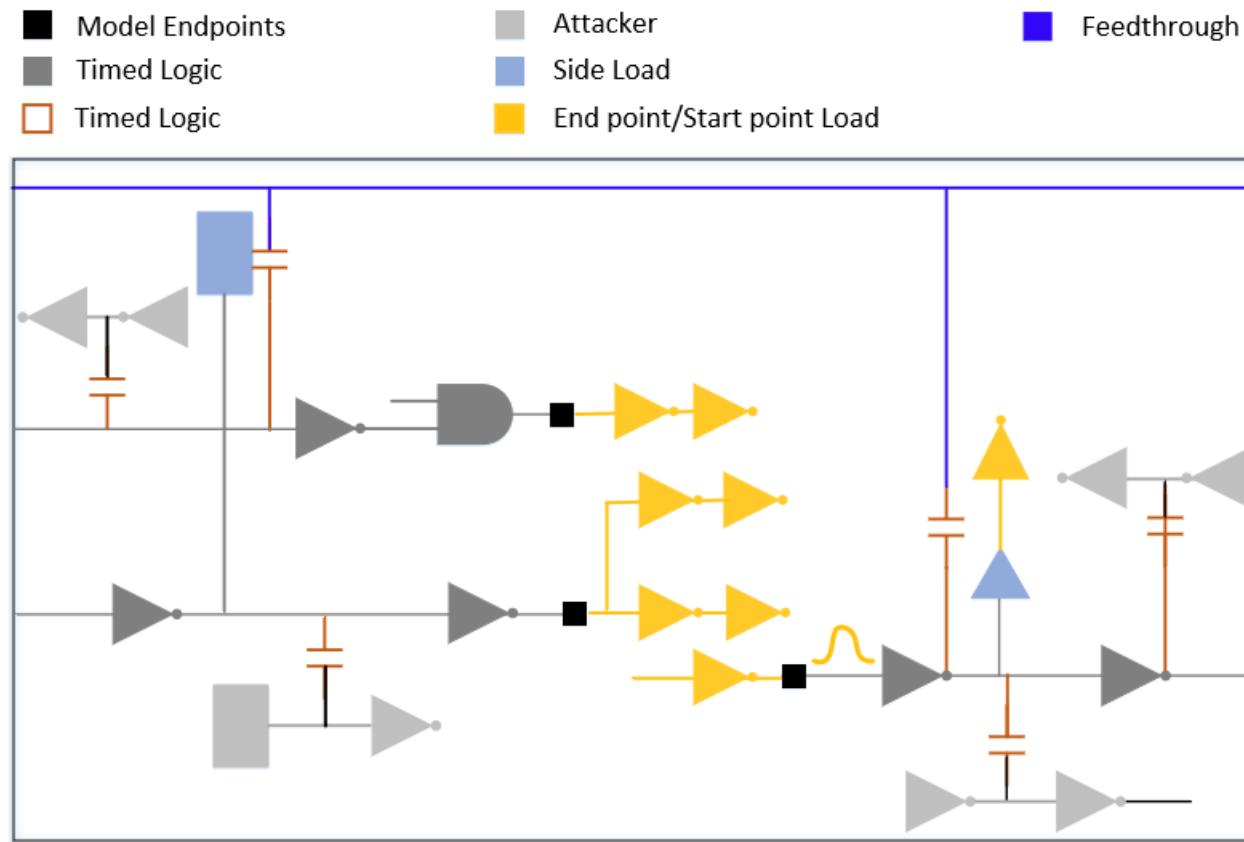
If a cell is encountered without any noise model, then it is treated as a single-stage cell for the purpose of logic level tracing. The next stage after the end point is also kept as load so that the slew/waveform can be put on the driver of the net connected to the start point, but is not analyzed when the block is used. The driver stage of the start point is kept as a driver and noise is injected if the cell is single-stage or a buffer. This logic is described below:

- The internal path nets that are SI attackers to the N-level path nets. All the leaf drivers and receiver instances of such nets are also included.
- The receiver instances on the N-level path nets that are not part of the interface logic are included. These side load receivers can occur in the clock network and on the paths from the register to the output port paths.
- The end points are saved to ensure proper loading, but they are not analyzed when the boundary model is used.
- The attacker driver and receiver are saved, but not analyzed when the boundary model is used.
- The analyzed noise on the drivers to the start points is saved and injected when the driver is a single-stage cell or buffer.

- The feedthrough is saved as part of the boundary model.

The following figure illustrates the logic described above:

**Figure 9-18 Glitch-specific boundary model with feedthrough**



The glitch correlation between a fully flat higher level run and a higher level run using one or more boundary models is expected to be close. However, the following are some of the factors that can negatively affect the correlation:

- Boundary slews at the block and the top level will not match perfectly, but the expectation from the boundary constraints is that they will be worst case. The differences in slews can cause differences in glitch results.
- User errors in the form of inconsistent SI settings between the block level and the top level.

### Glitch Boundary Model Usage

The following command is used to generate a glitch boundary model:

■ create glitch boundary model

In addition, the following global variable must be set prior to glitch/delay calculation so that the attacker timing windows are stored as part of the boundary model:

```
set timing_full_context_flow 1
```

The default value of this variable is 0.

### **Glitch Boundary Model Contents**

The contents of a glitch boundary model are as follows:

- <block>.db: Contains the Verilog files that are equivalent to the block:
  - model\_<#>.v.gz
- bound\_libs.txt:
  - File containing the timing libraries used in the block.
- context: Context of the boundary model:
  - ct.gz
  - es.gz
  - et.gz
  - glitch.tcl.gz
  - timestamp
  - tw.gz
- libs
  - File containing all the timing libraries loaded from the Tcl to create the model.
- Partition: Contains the SPEF file and partition information:
  - SPEF file
  - endpoints.gz
  - endpoints2.gz
  - stoppoints2\_wf.gz
  - Ignoredpins.gs

### **9.2.7 Boundary Models in MMMC Mode**

The boundary model flow is MMMC-aware. The requirement is that the block run used to generate the boundary model should be CMMMC with the same (or larger) set of views as in the top CMMMC run.

The boundary model generated from the MMMC environment saves the SPEF data for all the active RC corners, the constraints for all active modes, and the context for all the active views.

### **9.2.8 Save-Restore of Hierarchical Analysis**

A hierarchical analysis run with a boundary model can be saved and restored like a non-hierarchical run. In addition, boundary models can be generated from the restored sessions without performing a timing update.

## 9.3 SmartScope Hierarchical Analysis

- 9.2.1 [Overview](#) on page 165
- 9.2.2 [Why Use SmartScope Analysis](#) on page 166
- 9.2.3 [Choosing Blocks to Use for SmartScope Analysis](#) on page 166
- 9.2.4 [SmartScope Analysis Flow](#) on page 167
- 9.2.5 [Running SmartScope Hierarchical Analysis](#) on page 169
- 9.2.6 [Scope Generation from Distributed STA \(DSTA\)](#) on page 171

### 9.3.1 Overview

The SmartScope hierarchical analysis feature allows you to re-time a modified block with its context without having to re-time the full design. The feature also allows timing at the top-level without having to time the full internals of the blocks. There are two aspects of SmartScope analysis, which can be used either separately or together:

- Boundary models can be generated for the blocks and used at the top level. This provides a timing analysis of the complete top level paths with a reduced runtime.
- Scope models can be generated from a top level run for the instance(s) of a block and the surrounding context. The scope models can be generated either from a full flat run, or a top level run with boundary models.

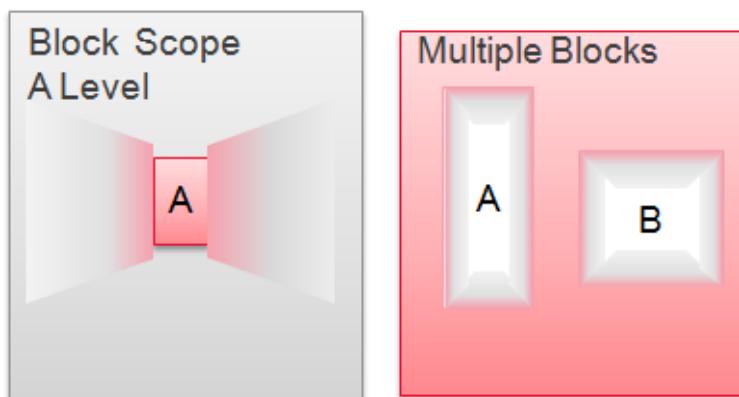
Using both aspects together allows for a complete hierarchical analysis without having to make a full flat timing run.

The scopes mentioned above are referred to as block scope, because they contain the block instance(s) along with the surrounding fanin and fanout logic.

A scope of the top-level logic can also be generated, which is referred to as a top scope. A top scope is the dual of a block scope, such that it contains all of the top-level logic, and the interface paths of the hierarchical child instances. The top scope is analogous to a top level run with boundary models.

The following figure illustrates block scope and top scope.

**Figure 9-19 Top scope and multiple block scope logic**



The SmartScope hierarchical analysis provides the following features:

- Scopes can be generated for blocks from either STA or DSTA run.

- Scopes can be generated from an STA run without a full flat timing analysis.
- For multi-instantiated blocks, the scope analysis can be limited to a single copy of the block internals from one of the instances. For the other instances, only the boundary paths will be analyzed.
- A scope for the top-level logic can be generated from either full STA or DSTA run.

### **9.3.2 Why Use SmartScope Analysis**

The SmartScope Analysis feature enables you to take a full chip and reduce the amount of timing analysis required. This restricts the scope of the timing analysis down to few modules that are relevant, thus reducing the run time and providing a better constraint accuracy than a block-level script.

Additionally, SmartScope Analysis reduces CPU requirements, and increases the quality of constraints. The budgetary constraints cannot fully capture the top level environment.

The top-level constraints can be different than budgetary constraints for the following reasons:

- Top level design may tie in different clocks than expected.
- Full chip introduces new pin drives and loads.
- Full chip introduces new module input and output expectations.
- Full chip introduces new clock latencies and CRPR.
- Full chip introduces new constants or constraints many be added.

### **9.3.3 Choosing Blocks to Use for SmartScope Analysis**

SmartScope allows selection of modules to analyze or to exclude from analysis. If a large part of a full design is analyzed, then the run time will be as high as a full flat run. Selecting the right amount of the design for analysis is an important choice, as described in the following section.

#### **Good Choices for Block Selection**

- High level modules that have a SPEF:  
These blocks match the layout hierarchy. It is likely that they are owned by a single designer and any failing timing paths will be addressed by changing this level of hierarchy. If the module represents 10-30 percent of the full flat design, it is a great choice.

- Modules that have a large number of timing issues:  
These blocks will need to be changed and will be candidates to be retimed using scope.  
As the block is changes the block scope can be used to analyze the effects.
- Modules that have no timing violations:  
These are good blocks to exclude from the top-level analysis. There is no advantage in running timing analysis on the internals of a block that is not changing.

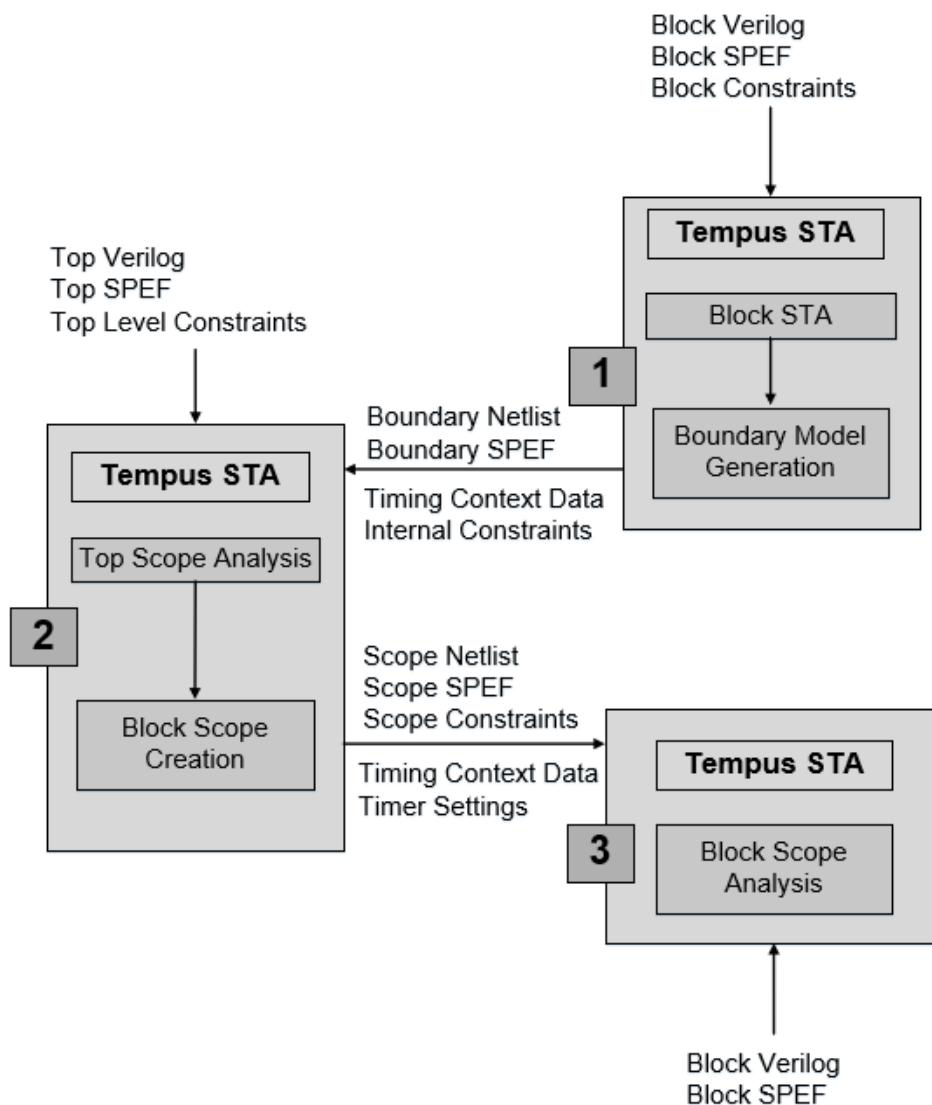
### **9.3.4 SmartScope Analysis Flow**

The full SmartScope hierarchical analysis flow is as follows:

- For each block of interest, a block-only timing run with STA is made which generates a boundary model for the block. The boundary mode consists of a netlist, SPEF, timing context data, clock information, and internal constraints.
- A timing run is made from the top level which reads in boundary models for the blocks rather than the full netlist.
- A block scope is generated in the top level run for each of the blocks of interest.

The flow is shown in the diagram below:

**Figure 9-20 SmartScope Hierarchical Block Scope Flow**

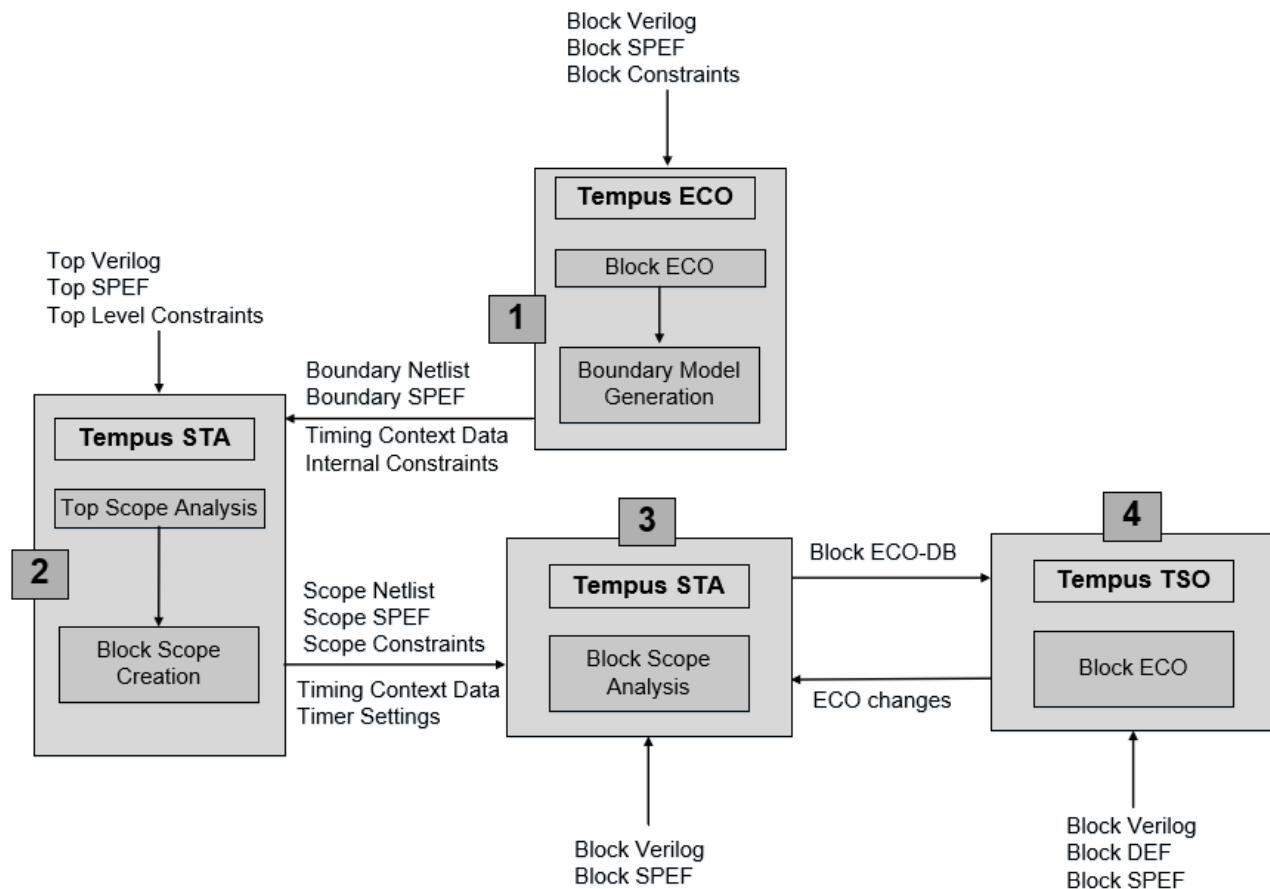


SmartScope analysis is designed to be used with Tempus ECO. For ECO purposes, the flow can be extended as follows:

- From the top-level run with boundary models, generate an ECO-DB for the top-level logic.
- Run Tempus ECO with the top-level netlist and the generated ECO-DB.
- Run timing analysis using the scope models for each block, and then generate an ECO-DB for each block.
- Run Tempus ECO with each block-level netlist and the corresponding ECO-DB.

The following diagram illustrates the block scope flow for Tempus ECO:

**Figure 9-21 SmartScope Hierarchical Block Scope ECO Flow**



For timing of the top-level logic with the context of the block interfaces, an alternative to using boundary models is to generate a scope of the top-level logic from a full flat run. This type of scope is referred to as a top scope rather than a block scope.

An ECO-DB for the top-level logic can be generated from a timing run of the top scope. This has the disadvantage of requiring a flat run, but has the advantage of not requiring good block-level constraints for a timing run of the block.

### 9.3.5 Running SmartScope Hierarchical Analysis

This section provides the specifics of how to run full SmartScope analysis from Tempus STA sessions.

To generate the boundary model for a block, make a block-level timing run using the following commands:

- `set timing_full_context_flow 1`

The `timing_full_context_flow` variable is used to capture the estimated slews and timing windows during `update_timing` as part of the timing context in the boundary model.

- `create_boundary_model -dir path [-overwrite]`

The `create_boundary_model` command generates a boundary model from the full design.

**Note:** The `timing_full_context_flow` variable needs to be set ahead of any timing update, and the `create_boundary_model` command needs to be run after a full timing update. The command will implicitly run `update_timing`, if necessary.

For the top level timing run, the following commands should be used.

- `read_boundary_model -dir path [-clock_map_file file]`

The `read_boundary_model` command is used for each cell boundary model generated by the block-level runs. Multiple commands can be specified if there are multiple models. The commands must be issued prior to the `set_top_module` command.

- `set timing_full_context_flow 1`

If scopes are to be generated from the run, then set the `timing_full_context_flow` to 1. This setting must be done ahead of any timing update.

- `create_scope_model {-cell cell | -instance inst_list | -cell_notier cell | -instance_notier inst_list} [-exclude_cell cell_list] [-exclude_instance inst_list] -path dir [-rep_instance inst] [-overwrite]`

The `create_scope_model` command generates a top or block scope and should be specified after the `update_timing` command. The command will implicitly run the `update_timing` command, if necessary. Multiple commands can be issued to generate different scopes.

For multi-instantiated cells, the `-cell` parameter will limit the corresponding internal endpoints included in the scope to a single instance. The other instances will only include boundary paths, either input-to-register or register-to-output. You can specify which instance is to be fully included with the `-rep_instance` parameter; otherwise an instance will be determined by the software.

For the scope timing run, the following commands should be used.

- read\_scope\_verilog file\_list

The `read_scope_verilog` command is optional, but should be issued in cases where there is a modified version of the block Verilog which is to be run with the scope. The module definitions in the supplied file(s) will overwrite the modules saved in the scope. Multiple commands can be used, and the commands must be issued prior to the `read_scope` command.

- read\_scope\_spef [-scaleRC] [{-rc\_corner name} | {-early\_rc\_corner name} {-late\_rc\_corner name}]

The `read_scope_spef` command is optional, but should be issued in cases where there is a modified version of the block SPEF, which is to be run with the scope. The supplied file(s) will replace the corresponding files specified in any subsequent `read_spef` commands. Multiple commands can be used, and the commands must be issued prior to the `read_scope` command.

- read\_scope -dir path

The `read_scope` command reads a scope. The scope includes the netlist, SPEF, constraints, settings, modes, view definition, and so on. When the scope is loaded, the STA session is ready for timing updates and reporting.

### 9.3.6 Scope Generation from Distributed STA (DSTA)

The block and top scope can be generated from a DSTA session.

Given below are some of the differences between scope generation in STA and DSTA modes:

- The scope generation command is `distribute_create_scope_model` (in DSTA) instead of `create_scope_model`.
- For multi-instantiated blocks, the scope will contain the internal register-to-register logic for analysis for all the instances instead of a single representative instance.

Note that the `timing_full_context_flow` global variable should be set for DSTA scope generation in the same way as for STA scope generation.

The DSTA-generated scopes are loaded in a STA session in exactly the same way as a STA-generated scope.

## 9.4 Prototype Timing Modeling

- 9.3.1 [Overview](#) on page 173
- 9.3.2 [Inputs and Outputs](#) on page 173
- 9.3.3 [Using the PTM Flow to Generate Dotlib Model](#) on page 174
- 9.3.4 [Handling of Bus Bits in PTM](#) on page 178

### 9.4.1 Overview

The prototype timing model (PTM) feature enables you to define and generate a prototype timing model database, which contains preliminary information about the block, such as ports, delay arcs, timing check or constraints. This information is then used to generate a library (.lib), Verilog (.v), and What-If (.cmd) command files.

PTM models are useful for performing feasibility studies on a logical block. In addition, PTM can be used to generate preliminary timing information, which becomes the validation check point for early development, and evolves with design.

With PTM, you can easily create a block without providing many details, which would then provide preliminary timing information for it. This block can be replaced by a detailed timing model later in the design cycle to get more accurate timing information.

This chapter describes how to define and generate prototype timing models (PTM). It explains the inputs and outputs for PTM flow and the procedure to use PTM models.

### 9.4.2 Inputs and Outputs

- Inputs
- Outputs

#### Inputs

- Standard cell libraries or macros.
- A set of PTM commands to define the PTM model.

#### Outputs

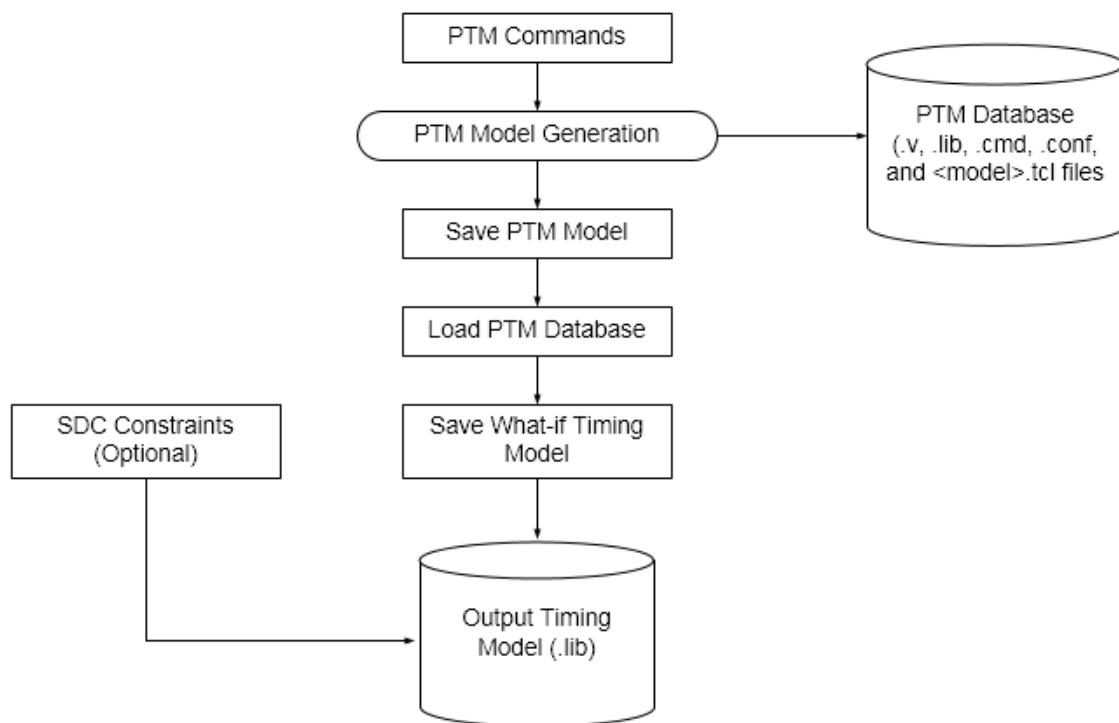
A PTM database that contains the following information:

- Verilog (.v) model: Contains the port definition with the instantiated PTM cell model.
- Library (.lib) model: Contains pin or bus definitions (direction and capacitance).
- What-If Command file (.cmd): Contains the mapped What-If commands.
- Configuration file (.conf): Contains information to load the generated library and Verilog models with the read\_design command.
- Top-level Tcl script: Contains the commands to load the configuration file, source the What-If constraints, and write the timing model and assertions.

### 9.4.3 Using the PTM Flow to Generate Dotlib Model

The following diagram shows how the PTM commands are used to generate the PTM model database. This PTM model database is later used to save a What-If timing model, which can be used to generate a dotlib model that contains detailed timing information. You can optionally load the SDC constraints while generating the output dotlib model.

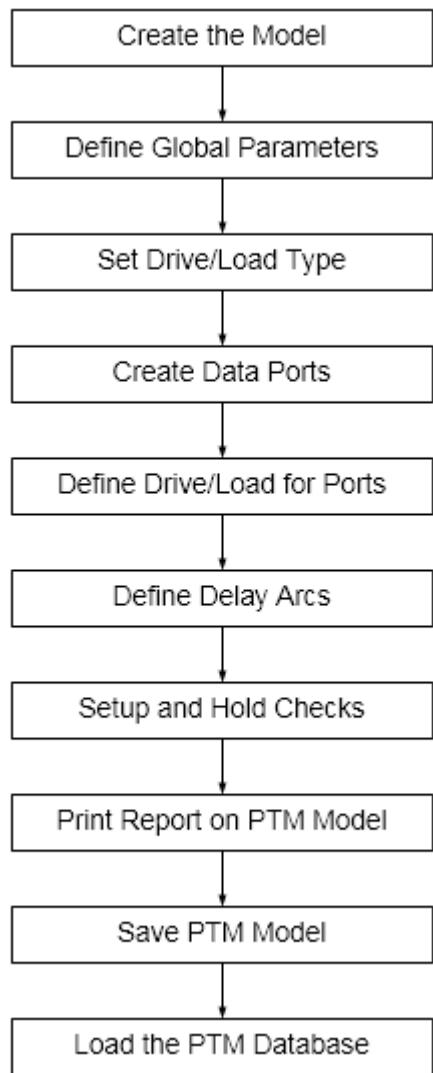
**Figure 9-22 Flow Diagram**



### Command Flow

The following figure shows the PTM command flow for generating a dotlib model:

**Figure 9-23 Sample PTM Model Flow**



### Generating a PTM Model

The following procedure shows how to define a PTM model and use it to generate a dotlib timing model, which can later be instantiated in a top-level design:

1. To create the PTM model, use the [create\\_ptm\\_model](#) command:

```
create_ptm_model dma
```

**Note:** All other PTM commands will work only after this command is executed.

2. To define global values for setup check arcs, hold check arcs, or sequential delay arcs across the PTM model, use the [set\\_ptm\\_global\\_parameter](#) command:

```
set_ptm_global_parameter -param setup -value 1.5
set_ptm_global_parameter -param hold -value 1.5
set_ptm_global_parameter -param clk_to_output -value 1.5
```

**Note:** The global value specified with this command is added to the arc specific delay value. As a result, the final arc's delay is calculated as a sum of the global parameter value and the arc delay specified using the [create\\_ptm\\_constraint\\_arc](#) or the [create\\_ptm\\_delay\\_arc](#) command.

3. To create drive types for specifying the drive strength equivalent to existing library cells, use the [create\\_ptm\\_drive\\_type](#) command:

```
create_ptm_drive_type -lib_cell INVX1 stdDrive
```

4. To create a load type by assigning it a unique name, use the [create\\_ptm\\_load\\_type](#) command:

```
create_ptm_load_type -lib_cell INVX4 stdLoad
```

Typically, the load type represents a capacitance value from a library input or bidirectional pin. Use multiple [create\\_ptm\\_load\\_type](#) commands to create multiple load types for a PTM model.

5. To create PTM data ports for the current model, use the [create\\_ptm\\_port](#) command:

```
create_ptm_port -type input data_in
create_ptm_port -type output data_out
create_ptm_port -type input data_test_1
create_ptm_port -type inout data_test_2
create_ptm_port -type input bcstop
create_ptm_port -type input bdis1
```

6. To create a PTM clock port for the current model, use the [create\\_ptm\\_port](#) command:

```
create_ptm_port -type clock clock_in
```

7. To set the drive for the output and bidirectional ports, use the [set\\_ptm\\_port\\_drive](#) command:

```
set_ptm_port_drive -type stdDrive data_out
```

The drive type specifies the type of library cell within the model as driver on the interface nets. You cannot specify two different drivers for two timing arcs ending at the same output port.

8. To define a load for the PTM model ports by specifying a capacitance value, use the [set\\_ptm\\_port\\_load](#) command:

```
set_ptm_port_load -value 4.0 bdis1
```

9. To set the delay for combinational or sequential paths from input ports to the output ports, use the [create\\_ptm\\_delay\\_arc](#) command:

```
create_ptm_delay_arc -from clock_in -to data_out -edge rise -value 0.05
create_ptm_delay_arc -from data_test_1 -to data_test_2 -value 1.0
create_ptm_delay_arc -from data_test_2 -to data_out -value 1.5
```

**Note:** If you do not specify the drive type on the output port, a constant timing table is created, which signifies that the delay is constant for this arc. This means that the delay does not vary with respect to input slew and output capacitance.

10. To define the timing check at data input port with a reference clock port, use the [create\\_ptm\\_constraint\\_arc](#) command:

```
create_ptm_constraint_arc -setup -value 0.3 -from clock_in -to
data_in -edge rise
create_ptm_constraint_arc -setup -value 0.4 -from clock_in -to
data_in -edge fall
create_ptm_constraint_arc -hold -value 0.1 -from clock_in -to data_in -
edge rise
create_ptm_constraint_arc -hold -value 0.2 -from clock_in -to data_in -
edge fall
```

11. To generate a report on ports, arcs, and global delay parameter details for the current PTM model, use the [report\\_ptm\\_model](#) command:

```
report_ptm_model
```

12. To save the PTM model in dotlib library file format, use the [save\\_ptm\\_model](#) command:

```
save_ptm_model
```

13. To use the PTM database for generating a dotlib timing model, source the dma.tcl file:

```
source dma.tcl
```

14. To instantiate the model block in a top-level design, use the [read\\_lib](#) command:

```
read_lib dma.lib
```

This serves the purpose of validating the top-level timing information related to the instantiated block.

## PTM Example

The following example provides a list of commands that can be used to generate a PTM model illustrated in the figure (*Sample PTM Block*) below:

```
Start of model dma
create_ptm_model dma

Define drive and load type
create_ptm_drive_type -lib_cell INVX1 stdDrive
create_ptm_load_type -lib_cell inv_hivt_4 stdLoad

Create data ports
create_ptm_port -type input data_in
create_ptm_port -type output data_out
```

```

create_ptm_port -type input data_test_1
create_ptm_port -type inout data_test_2

Create clock ports
create_ptm_port -type clock clock_in

Define drive for ports
set_ptm_port_drive -type stdDrive data_out

Define load for ports
set_ptm_port_load -type stdload data_test_2

Define delay arcs
Data delays
create_ptm_delay_arc -from data_test_1 -to data_test_2 -value 1.0
create_ptm_delay_arc -from data_test_2 -to data_out -value 0.5

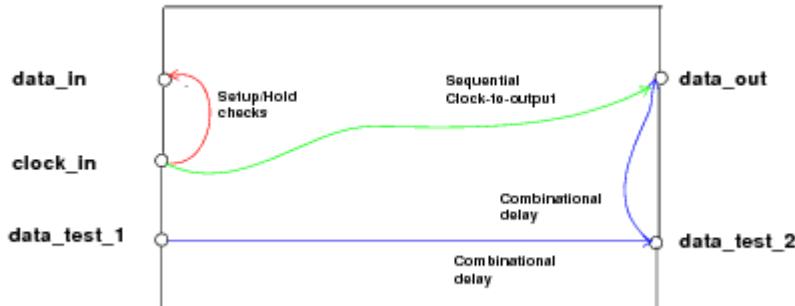
Sequential clock to output delay
create_ptm_delay_arc -from clock_in -to data_out -edge rise -value 0.05

Setup and hold checks
create_ptm_constraint_arc -setup -value 0.3 -from clock_in -to data_in -edge rise
create_ptm_constraint_arc -setup -value 0.4 -from clock_in -to data_in -edge fall

create_ptm_constraint_arc -hold -value 0.1 -from clock_in -to data_in -edge rise
create_ptm_constraint_arc -hold -value 0.2 -from clock_in -to data_in -edge fall

```

**Figure 9-24 Sample PTM Block**



#### 9.4.4 Handling of Bus Bits in PTM

The following examples provide a list of commands that can be used to handle bus bits in PTM:

## Defining Bus Bits

To create a bus port along with bus members definition, use the `create_ptm_port` command. An integer specified with a bus name designates the number of bus bits, as shown below:

```
create_ptm_port -type input test_input[31:0]
```

## Creating Arcs from/to Multiple Bits of a Bus

### ***Delay ARC***

#### **Example 1:**

The example shows how to create delay arc on pin, `test_clk`, with respect to each bit of `test_input` bus:

```
create_ptm_delay_arc -from test_input[31:0] -to test_clk -value 1.0
```

#### **Example 2:**

The example shows how to create delay arc on pin, `test_inout[7:0]`, with respect to each bus bit, `test_input[31:0]`:

```
create_ptm_delay_arc -from test_input[31:0] -to test_inout[7:0] -value 5.0
```

This command creates 32 arcs on all 8 bus pins, `test_inout [7:0]`.

#### **Example 3:**

The example shows how to create delay arc on pin, `test_inout[7:0]`, with respect to selected bus bit:

```
create_ptm_delay_arc -from test_input[20:0] -to test_inout[7:0] -value 5.0
```

This command creates 21 arcs on all 8 bus pins, `test_inout [7:0]`.

### ***Constraint ARC***

#### **Example 1:**

The example shows how to create constraint arc on bus bits, `test_input[31:0]`, with respect to clock pin, `test_clk1`:

```
create_ptm_constraint_arc -setup -value 0.3 -from test_clk1 -to test_input[31:0] -edge rise
```

#### **Example 2:**

The example shows how to create constraint arcs with different values for selected bits of bus, `test_input[31:0]`, with respect to clock pin, `test_clk1`:

```
create_ptm_constraint_arc -setup -value 0.3 -from test_clk1 -to
test_input[3:0] -edge rise
create_ptm_constraint_arc -setup -value 0.8 -from test_clk1 -to
test_input[7:4] -edge rise
```

## Define DRV on Bus Bits

***Set load on bus bits:***

### Example 1:

The example shows how to create a load type by assigning it a unique name using the `create_ptm_load_type` command:

```
create_ptm_load_type -lib_cell ABC -input_pin A load1
```

The example shows how to set the load type, `load1` on all the bus bits using the `set_ptm_port_load` command:

```
set_ptm_port_load -type load1 test_input[31:0]
```

### Example 2:

The example shows how to set the different load type on specified bus bits using the `set_ptm_port_load` command:

```
create_ptm_load_type -lib_cell ABC -input_pin A load1
set_ptm_port_load -type load1 test_input[21:0]
create_ptm_load_type -lib_cell XYZ -input_pin A load2
set_ptm_port_load -type load2 test_input[31:22]
```

This will set `load1` on 0-21 bits and `load2` on 22-31 bits of the `test_input` bus.

### Example 3:

The example shows how to define a load for ports by specifying a capacitance value using the `set_ptm_port_load` command:

```
set_ptm_port_load -value 4.0 test_input[31:0]
```

***To set DRV on bus bits:***

### Example 1:

The example shows how to specify a value for the `max_transition`, `max_capacitance`, and `max_fanout` design constraints:

```
set_ptm_design_rule -max_transition 1.1 -port test_input[31:0]
set_ptm_design_rule -max_capacitance 1.2 -port test inout [7:0]
set_ptm_design_rule -max_fanout 4 -port test inout[7:0]
```

**Example 2:**

The example shows how to specify a value for the `max_transition`, `max_capacitance`, and `max_fanout` using the `set_ptm_design_rule` command on specified bit:

```
set_ptm_design_rule -max_transition 1.5 -port test_input[21:0]
set_ptm_design_rule -max_fanout 5 -port test inout[4:0]
set_ptm_design_rule -max_capacitance 1.8 -port test inout[4:0]
```

## 9.5 Interface Logic Models

- 9.4.1 [Overview](#) on page 183
- 9.4.2 [Creating ILMs](#) on page 183
- 9.4.3 [Specifying ILM Directories at the Top Level](#) on page 186
- 9.4.4 [ILMs Supported in MMMC Analysis](#) on page 187
- 9.4.5 [ILM Validation Flow](#) on page 188
- 9.4.6 [Use of SDF, SDC, and XTWF in ILM Flow](#) on page 190
- 9.4.7 [Creating XILM](#) on page 193
- 9.4.8 [Using ILM/XILM for Hierarchical Analysis](#) on page 194

### 9.5.1 Overview

Models are compact and accurate representations of timing characteristics of a block.

Interface Logic Model (ILM) is a structural representation of a block, specifically a subset of the block's structure including instances along the I/O timing paths, clock-tree instances, and instances or net coupling affecting the signal integrity (SI) on I/O timing paths. It is a compact and accurate representation of timing characteristics of a block. Instead of using a blackbox at the top level, you create an ILM at the block level and use it as you would use a blackbox.

The ILM is not only a netlist; it comprises at least three files: verilog netlist, SPEF, and SDC constraints. The SI ILM logic model, or XILM additionally contains capacitances and other logic needed to model SI effects. An XILM would contain netlist, SPEF, and SDCs, and also a timing window file (.xtwf).

The advantages of using ILMs are as follows:

- More accurate analysis than a black box flow
  - More SI aware than combined `.lib` or `.cdb` approach
  - Can model clock generator inside block
  - More accurate timing and SI reduces the number of design iterations to close timing and SI
- No need to characterize blocks
  - Works on a actual design data
- Can be used in the initial prototyping stage for very big designs when loading full design data is not feasible.
  - Allows you to modify only top-level data
  - Fully preserves implemented partitions
- Uses the original constraint file for top-level analysis
  - No abstraction for timing exceptions

### 9.5.2 Creating ILMs

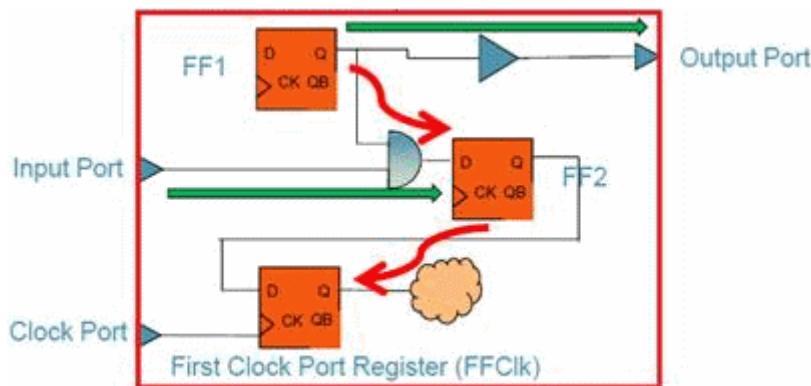
In the hierarchical design flow, you load a detailed block-level implementation of a block and its parasitic data, then specify the `write_ilm` command to create an ILM for the block. This command creates the specified directory containing ILM files.

**Note:** An ILM created for an incomplete block is not as accurate as an ILM created for a complete block. Always use ILMs for complete blocks to complete the top-level design.

The software generates ILM data for timing and signal integrity (SI):

- ILM data for timing  
The model contains the netlist of the circuitry leading from the I/O ports to interface sequential instances (that is, registers or latches), and from interface sequential instances to I/O ports. The clock tree leading to the interface registers is preserved.
- Internal register-to-register paths, if internal logic is not part of the interface path.  
However, in special cases some internal paths will be kept, as shown in the following example:

**Figure 9-25 Internal Paths**



- Internal paths  
Internal paths controlled by different clock, or clocks connected to the ILM module through different ports. If the logic between the I/O ports is pure combinational, it is preserved in an ILM.
- ILM data for SI  
The model includes all of the above, plus attacker drivers or nets which affect I/O paths. It also includes the timing window files in the ILM model directory.

### Example ILM Creation

The following command generates the ILM ignoring certain ports and including certain objects in the `BlockB` directory. It also writes out the ILM SDF file under the `BlockB` directory.

```
write_ilm -writeSDF -ignore_ports $ignorePorts -include_object $includeObjects -
dir BlockB
```

### **Sample Summary Report**

The following is a sample summary report generated after running the `write_ilm` command:

```

 createInterfaceLogic Summary

Model Reduced Instances Reduced Registers

ilm_data
+
setup 4/16 (25%) 0/3 (0%)
hold 4/16 (25%) 0/3 (0%)
setupHold 4/16 (25%) 0/3 (0%)

si_data
+
setup 3/16 (18%) 0/3 (0%)
hold 3/16 (18%) 0/3 (0%)
setupHold 3/16 (18%) 0/3 (0%)

```

In this report, the reduction ratio in the `ilm_data` model is 25 percent which means that 4 out the total 16 instances for this block have been eliminated.

**Note:** You can run the `set_ilm_mode -keepHighFanoutPort false` command for improving the reduction ratio.

### **Preserving Selected Instances in ILMs**

The `write_ilm` command can preserve specified instances and/or nets. The `-include_object` parameter of the `write_ilm` command accepts a collection of instances and/or nets. All the specified instances and nets are preserved in the generated ILM.

### **Creating ILMs for Shared Modules**

You can use the same sub-block module in different ILM blocks, enabling reuse of versatile modules. The `write_ilm` command considers constant propagate, so that only the enabled parts of a module are considered when creating ILMs for the reused modules. Because the Tempus database cannot handle the same module name in different circuits, the software automatically modifies the module names with the following rule:

```
topModuleName+timestamp+$+moduleName
```

As an example, one ILM block (ModuleA) uses an ALU module (ALU) as an unsigned ALU, and a second block (ModuleB) uses the ALU as a signed ALU. You can change the input signal to use the ALU differently, setting one ALU as sign enabled and the other to off. When you run the `write_ilm` command, the software considers only the enabled parts of the ALU when creating ILMs for ModuleA and ModuleB. The software also ensures that the name of the ALU module in ModuleA and the name of the ALU module in ModuleB are different.

### **Creating ILMs Without Using Tempus Database**

If you do not have Tempus database for an implemented block but have a Verilog netlist, constraints, and SPEF and TWF for that block, then use the `import_ilm_data` command to store data in the ILM/XILM format.

The following example shows the `import_ilm_data` command in case of a MMMC design:

```
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -verilog myfile.v
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-spef max.spef.gz -rcCorner rcMax
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-spef typ.spef.gz -rcCorner rcTyp
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-spef min.spef.gz -rcCorner rcMin
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMaxMax.sdc -viewName funct-devSlow-rcMax
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMaxTyp.sdc -viewName funct-devSlow-rcTyp
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc tstMaxMax.sdc -viewName test-devSlow-rcMax
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc funMinMin.sdc -viewName funct-devFast-rcMin
import_ilm_data -si -dir block_A.ILM -cell block_A -mmmc -incr \
-sdc tstMinMin.sdc -viewName test-devFast-rcMin
```

The following example shows the `import_ilm_data` command in case of a non-MMMC design:

```
import_ilm_data -dir test -verilog \ ILM/ilmsetup/ilms_data/\
BlockA_0/BlockA_0_postRoute.v.gz -cell BlockA_0 -spef \ ILM/ilmsetup/ilms_data/\
BlockA_0\ BlockA_0_postRoute.spef.gz -\ sdc\ILM/ilmsetup/ilms_data/BlockA_0\
BlockA_0_postRoute.core.sdc.gz -si \
-overwrite -xtwf ILM/ilmsetup/si_ilms_data/BlockA_0/BlockA_0_SI.xtwf.gz
```

### **9.5.3 Specifying ILM Directories at the Top Level**

You can use the `specify_ilm` command to use the ILM data for a block at the top partition level rather than using the default dotlib model. You can run `specify_ilm` multiple times in the same session. Each time you run this command, the software overwrites the previous

setting for a block. If master/clones exist in a design, the cell name will have the name of the master partition.

You can use the `unspecify_ilm` command to revert to using the dotlib model for a block.

**Note:** You must use the `specify_ilm` (or `unspecify_ilm`) command before specifying `read_ilm`. This is because the ILM settings cannot be changed after using `read_ilm`.

### Example Top-Level Signoff Flow with ILMs

1. Before starting the Tempus tool, prepare the top-level Verilog file, if needed.

You need the following in the top-level directory:

- ❑ A Verilog netlist that includes dummy modules for the blocks (ILM or Liberty) in the design
- ❑ A view definition file if you have a MMMC design. Create or load a view definition file that contains the following:

```
set_analysis_view -setup {model_slowCorner} -hold {model_f astCorner}
```

2. Start a Tempus session from the top-level module directory within the directory where the partitions are saved.

3. Load the config file, including the top-level netlist, libraries, top-level constraint file, and ILM directory name.

```
specify_ilm -cell block_A -dir ../block_A/block_A.ILM
specify_ilm -cell block_B -dir ../block_B/block_B.ILM
read_design filename
```

4. Run `read_spef` to load the top-level Standard Parasitic Extraction File (SPEF).

5. Run the `read_ilm` command.

6. Run the `report_timing` command to perform timing analysis.

#### 9.5.4 ILMs Supported in MMMC Analysis

Cadence strongly recommends that you use ILMs in the MMMC mode. If you have a non-MMMC design, create and load a view definition file that contains the following:

```
set_analysis_view -setup {model_slowCorner} -hold {model_fastCorner}
```

The MMMC analysis for designs including ILMs is identical to MMMC analysis for black box designs except for the following considerations:

1. Views, modes, and corners at the top and partition levels must have same names.

2. When you use `create constraint mode` or `update constraint mode` to specify constraints for MMMC, you must specify the ILM constraints using the `-ilm_sdc_files` parameter (that is, timing in the presence of ILMs get constraints from the `-ilm_sdc_files` parameter, not the `-sdc_files` parameter). The SDC files specified with the `-ilm_sdc_files` parameter should be the constraint file of the full-chip flat netlist, where it allows referencing nets or pins internal to the ILM model.

**Note:** In the current ILM flow, the SDC constraints (originally specified against the complete flat netlist for the design) that reference parts of the design that were pruned cause warnings and errors during constraint loading. In this release, you can set the temporary `timing suppress ilm constraint mismatches` global variable to `true` to suppress all error and warning messages related to the unfound objects. Note that this command might also suppress error messages that might be of use (that is, where the top-level pins or nets or instances cannot be found).

### 9.5.5 ILM Validation Flow

The ILM validation flow can be used to validate a model at the block level. For example, an ILM netlist and the .sdc file can be read in a separate Tempus session and timing analysis can be run on all paths. Then, the results can be compared against timing for the same path during full-block implementation.

Use the following steps to run the automated ILM validation flow:

1. Load a block level design.
2. Load the SPEF file.
3. Run the following command:

```
write_ilm -dir ilm_dir -validation_dir validation_dir
```

Two directories are created - one for saving the ILM data and the second for saving the validation information. The validation directory contains following files:

**Table 9-1 List of Files in the Validation Directory**

File	Description
init.tcl	Used to initialize the ILM data.
nets.tcl	Used to get the list of net names in which only the real ILM nets are included and other nets like attacker nets are excluded.
full.tcl	Used to run timing analysis on the original netlist using the <code>write model timing</code> command.

## Tempus User Guide

### Timing Model Generation

---

xilm.tcl	Used to run timing analysis on the generated ILM data using the <code>write_model_timing</code> command.
rpt.tcl	Compares the output of original netlist with the output of ILM data using the <code>compare_model_timing</code> command.
validation.src	Used to run the ILM validation flow by sourcing the Tcl files automatically.

4. Switch to the validation directory.

5. Source the validation.src file.

After running the ILM validation flow, the validation.sum file is generated. This file provides the summary of the validation checks.

### Sample ILM Validation Summary Report

```
#-----#
Tolerance FAIL/PASS Ave
(abs,per) AbsDiff Max
(abs,per) AbsDiff
#-----#
ILM SETUP Slack (0.050 , 2%) 0/6 0.000 0.000 #
 PinCap (0.050 , 2%) 0/5 0.000 0.00 #
 Tran (0.050 , 2%) 0/5 0.000 0.000 #
 DRV (, 2%) 0/5 -- NA #
ILM HOLD Slack (0.050 , 2%) 0/0 NA NA #
 PinCap (0.050 , 2%) 0/5 0.000 0.000 #
 Tran (0.050 , 2%) 0/5 0.000 0.000 #
 DRV (, 2%) 0/5 -- NA #
XILM SETUP Slack (0.100 , 3%) 0/6 0.000 0.000 #
 PinCap (0.100 , 3%) 0/5 0.000 0.000 #
 Tran (0.100 , 3%) 0/5 0.000 0.000 #
 DRV (, 3%) 0/5 -- NA #
XILM HOLD Slack (0.100 , 3%) 0/0 NA NA #
 PinCap (0.100 , 3%) 0/5 0.000 0.000 #
 Tran (0.100 , 3%) 0/5 0.000 0.000 #
 DRV (, 3%) 0/5 -- NA #
#-----#
ILM/XILM Validation PASS
```

#### Note:

- When `write_ilm` is specified, all the views are generated for multi-corner, multi-mode (MMMC) analysis.
- The automated ILM validation performs both ILM/XILM validation depending upon the data available.

## Interactive Use of ILMs

The Global Timing Debugger (GTD) also requires the design to be in a flattened state. The GTD displays rows with instances or nets as grayed out, which are internal to ILM.

### 9.5.6 Use of SDF, SDC, and XTWF in ILM Flow

#### ■ High-Level view of the ILM-based flow

The ILM flow falls into two essential flows that are placed in a bottom-up/top-down fashion. The two flows are done separately, and in separate sessions.

- ❑ ILM generation flow, usually done bottom-up.
- ❑ ILM analysis flow, usually done top-down.  
For example, a hierarchical design made up of BlockA and BlockB sub-blocks.
- ❑ In the ILM generation flow, each sub-block has to be implemented as an ILM (`write_ilm`) after timing and SI analysis sign-off.
- ❑ In the ILM analysis flow, the whole design is stitched together by reading the ILM models for BlockA and BlockB, top level netlist, top level SPEF, top level TWF and chip level SDC constraints to perform top level timing and SI analysis. This usually takes place in a separate session.

#### Use of XTWF Generated in ILM

The `write_ilm` command creates a directory structure containing the ILM data for timing and SI analysis as described below:

The information is saved in the following directories:

#### ■ For Timing:

```
$dir/ilm_data
*.v
*.sdc
*.spef
```

#### ■ For SI:

```
$dir/si_ilm_data
*.v
*.sdc
*.spef
*.xtwf
```

The XTWF is a timing file that contains timing window (the earliest and the latest possible arrival times that a signal may arrive on a net or a pin) for attacker nets. Tempus does use the

XTWF for SI analysis in the ILM flow. The TWF file is usually used in Tempus if you are performing timing analysis in third-party tool and you want to bring the design into Tempus for SI analysis.

The ILM model is self-contained with all the needed information for both Timing and SI analysis which is suitable for the ILM bottom-up generation/top down analysis flow. See example below:

***Example 1: Block based ILM generation for a sub-module called blockA***

```
read_lib "$libDir/worst.lib"
read_lib -cdb "$libDir/worst.cdb"
read_verilog "$designDir/blockA.v"
set_top_module blockA
read_sdc "$designDir/blockA.sdc"
set_dc_sources -global 1.65
read_spef "$designDir/blockA.spef"
set_delay_cal_mode -siAware true
report_timing
write_sdf blockA.sdf
write_ilm -dir blockA
```

In a new session (when the hierarchical design is all stitched together), you read the ILMs of the sub-blocks to perform top-level STA and SI analysis sign-off.

***Example 2: Top-Level ILM based flow of design made up of two sub-module blockA and blockB***

**For SI Analysis:**

```
specify_ilm -cell blockA -dir dir1
specify_ilm -cell blockB -dir dir2
read_lib, read_lib -cdb ...
read_verilog top.v
set_top_module TOP
read_spef top.spef
read_sdc top.sdc
read_ilm
set_delay_cal_mode -siAware true
report_timing
write_sdf
```

### For Timing Analysis:

```
specify_ilm -cell blockA -dir dir1
specify_ilm -cell blockB -dir dir2
read_lib
read_verilog top.v
set_top_module TOP
read_spef top.spefread_sdc top.sdc
read_ilm
report_timing
...
```

### ILM Models and SDC-Generated Constraints

The SDC files generated within the ILM directories do not reflect a complete normal set of constraints. You will notice that only set\_annotated\_transition and set\_case\_analysis constraints are written in the SDC.

The SDC constraints found in the ILM directories are primarily used to save the input slew information for loop back path between interface logic and non-interface logic. This feature provides higher accuracy for both Timing/SI analysis using the ILM/XILM flow in Innovus/Tempus. Here is a scenario describing how the slews of reduced nets are stored for better accuracy of timing and SI:

An ILM instance inst1 has following two input pins:

- **Input pin1:** connects with ILM path, its input slew will be calculated and propagated from an ILM path.
- **Input pin2:** connects with non-ILM path which is not kept in ILM netlist. This often happens in a loop-back path. The path is reduced in which as a result its pin becomes dangling but its input slew information is needed to calculate the delay of the inst1. In this case, Tempus uses the slew kept in ILM/XILM .sdc file instead of the default input slew which improves the ILM analysis accuracy.

To perform timing analysis using ILM data files, you will need to use the chip level SDC constraints and the generated SDC constraints for the ILM block. The idea here is that these constraints are giving high accuracy while not colliding with top-level SDC constraints during ILM top down analysis flow.

### Example:

Generating ILM for BlockA:

```
read_lib tech.lib
```

```
read_verilog blockA.v
set_top_module blockA
read_sdc blockA.sdc
read_spef blockA.spef
write_ilm -dir blockA
report_timing > sta1.rpt
```

Now to use the ILM data for timing analysis and suppose the file names in the ILM directory are: ilm.v, ilm.spef, ilm.sdc, ilm.xtwf.

Simply read the original SDC and the SDC generated for the ILM.

**Example:**

```
read_lib tech.lib
read_verilog ilm.v
set_top_module blockA
read_sdc ilm.sdc
read_sdc blockA.sdc *
read_spef ilm.spef
report_timing > sta2.rpt
```

The timing report of the ILM interface paths should be consistent with the timing report of original full netlist (sta1.rpt should match sta2.rpt). This is usually done for validation purposes and not a real usage of the ILM flow.

If an ILM is read back to represent a sub-module in a hierarchical design, then to perform timing analysis, you will need to load the top level original constraints only. The timing report of all the top nets and interface paths should be consistent with original full flat netlist. This is the real usage of ILM as it reflects accurate timing and SI information derived from top level constraints.

### 9.5.7 Creating XILM

In Tempus, ILMs and XILMs are created using the `write_ilm` command.

To create an ILM or XILM for a block in Tempus, you must be at that block's hierarchy level; you cannot write a block ILM from the top-chip level of hierarchy. To provide the data needed for the ILM, you need the block's netlist, timing libraries, SPEF annotation, and SDC constraints.

In the following example, an XILM is created for a sub-block, named blockA:

```
read_lib "$libDir/worst.lib"
read_lib -cdb "$libDir/worst.cdb"
```

```
read_verilog "$designDir/blockA.v"
set_top_module blockA
read_sdc "$designDir/blockA.sdc"
set_dc_sources -global 1.65
read_spef "$designDir/blockA.spef"
write_ilm -dir blockA
set_delay_cal_mode -siAware true
report_timing
write_sdf blockA.sdf
```

When ILM is created using the `write_ilm` command, a directory named `blockA` will be created with the `MMMC` sub-directory. Within this directory you will see another level of sub-directories, named `ilm_data` and `si_ilm_data`, depending upon the `-modelType` parameter settings.

### 9.5.8 Using ILM/XILM for Hierarchical Analysis

The Tempus command for loading an ILM or XILM is `read_ilm`. To use one or more ILMs, use the `read_ilm` command just once. If there are multiple ILM blocks, specify them in a list, as shown in the example below. The `-cell` and `-dir` parameters accept Tcl lists, but note that the order of the cells and directories must match, for example:

```
specify_ilm -cell cell1 -dir dir1
specify_ilm -cell cell2 -dir dir2
read_lib
read_verilog top.v
set_top_module TOP -ignore_undefined_cell
read_spef top.spef
read_ilm
report_timing
```

You must run `read_ilm` after `read_spef`, and before `report_timing`. If you get an error such as:

```
**ERROR: (xxxILM-334) : Specified ILM blockA has non-empty module definition; no ILM created for it.
```

You will need to remove gates, assign statements, and so on from the `blockA` module in the top verilog netlist, because an ILM cannot overwrite the existing logic. The module, if it exists in the verilog netlist, must be empty.

```
create_constraint_mode -name mymode -sdc_files {file1_top_level.sdc
file2_top_level.sdc ...} \
-ilm_sdc_files {file1_chip_level.sdc file2_chip_level.sdc ...}
```

If ILM constraint files (at the chip level SDC) are not specified with the `create_constraint_mode` command (in the `viewdefinition.tcl` while reading the design),

## **Tempus User Guide**

### Timing Model Generation

---

then these can be read using the update constraint mode command (after the specify ilm command is issued), as shown in the following example:

```
update_constraint_mode -name mymode \
-ilm_sdc_files {file1_chip_level.sdc file2_chip_level.sdc ...}
```

---

## **Tempus Power Integrity Analysis**

---

- 10.1 [Tempus Power Integrity Analysis Overview](#) on page 197
- 10.2 [Licensing Requirements](#) on page 200
- 10.3 [Tempus Power Integrity Flow](#) on page 201
- 10.4 [Performing Tempus PI Analysis](#) on page 202
- 10.5 [Tempus PI Result Analysis and Reporting](#) on page 208
- 10.6 [User-Defined Critical Paths Support](#) on page 211
- 10.7 [Enabling Proximity Aggressors in Tempus PI](#) on page 212
- 10.8 [Sequential Activity and Simulation Period](#) on page 215
- 10.9 [Tempus PI Related STA Commands](#) on page 216

## 10.1 Tempus Power Integrity Analysis Overview

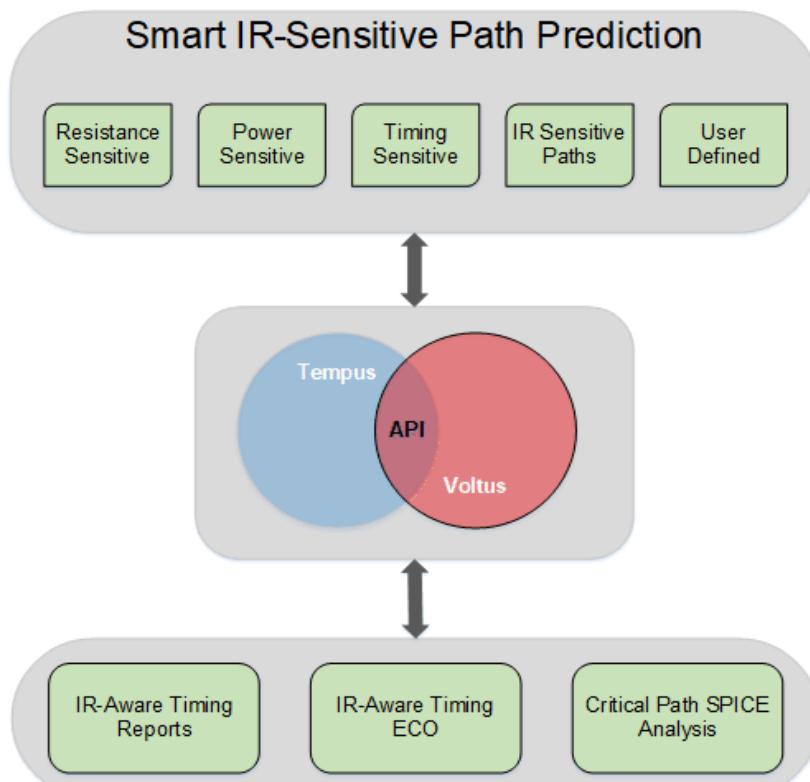
The Tempus Power Integrity Analysis (Tempus PI) solution provides a capability to run IR analysis that is timing aware, and timing analysis that is IR-aware. Tempus PI is a seamless integration of the Tempus static timing analysis (STA) and Voltus power and IR drop technologies. Both IR analysis and STA analysis can be performed on a single platform, so no external file or database transfer is required.

Traditionally, the IR-aware STA analysis mechanisms involve running either vector-based or vectorless dynamic simulations to compute the IR drop for each instance. As both these approaches are not truly timing-aware, it is very difficult to identify all potential IR-induced STA violations using these approaches.

The primary objective of the Tempus PI flow is to identify timing-critical and IR-sensitive timing paths, to accurately predict IR drop on these paths with directed vectorless analysis, and to enable automated fixing of these critical paths using Tempus ECO. The IR-sensitive timing paths are paths that may have sufficient positive slack but degrade significantly with IR drop. This timing/sensitivity-driven IR analysis is multi-cycle (transient IR simulation), functional, and vectorless. It also involves propagating the states/events of the scheduled critical paths through the fanout combinational logic using the Voltus state propagation engine.

The IR drop and timing events are natively aligned within the tool because Tempus STA analysis and Voltus IR analysis share all information, such as IR drop, timing windows, and switching within the timing windows. The following diagram illustrates the Tempus-Voltus integration in the Tempus PI flow:

**Figure 10-1 Tempus PI Flow**



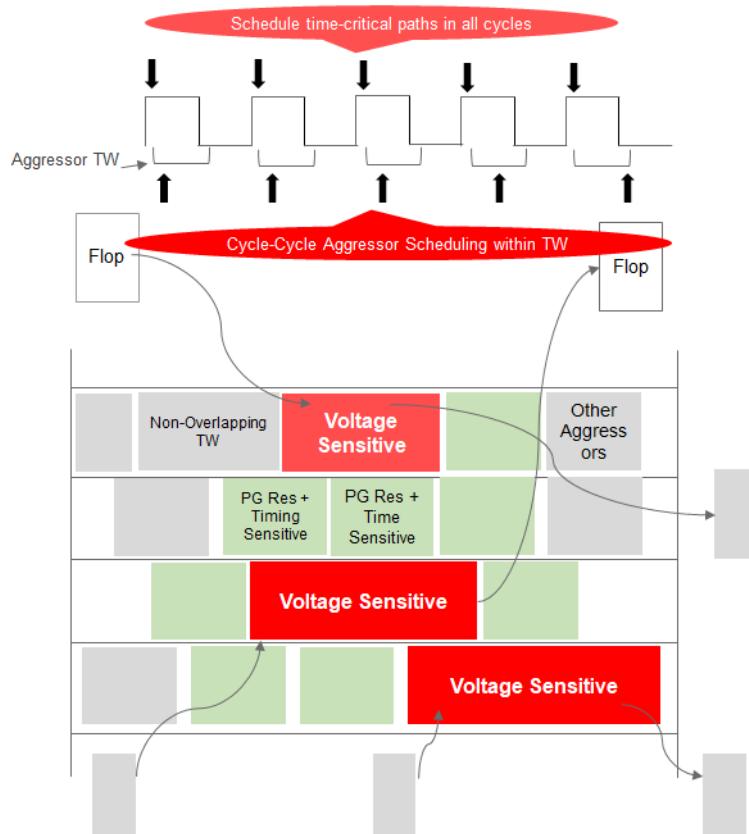
The following are the key features of the Tempus PI flow:

- Identify IR-sensitive paths - the critical paths are timed using the traditional VDD margins on cell delays using max IR drop. The IR-sensitive paths are not critical paths but can be critical in the IR conditions. These IR-sensitive paths are identified among many paths, such as the resistance sensitive paths, power sensitive paths, timing sensitive paths, IR-sensitive paths for neighboring cells, and user-defined paths. The IR-sensitive critical paths are scheduled to switch in all cycles of the transient IR simulation.
- Create functional path switching scenarios - it creates realistic worst-case switching scenarios for the path and for the environment around it. In this flow, the software is aware of the delays along the path, and also the delays of paths in the neighborhood. It creates switching scenarios that are timing slack and timing path aware.
- Identify proximity aggressors - it finds proximity aggressors around voltage and time-critical instances based on power-grid resistance and timing sensitivity. These proximity aggressors are modeled to switch at different times within the timing windows from cycle to cycle. The other aggressors are modeled using the vectorless algorithm to meet activity target and package effects.

**Tempus User Guide**  
Tempus Power Integrity Analysis

---

**Figure 10-2 Tempus PI Flow**



- Calculate impact of delay/timing on voltage and time critical paths using cycle-cycle voltage statistic.

Following is a comparative analysis of a slack report (histogram of timing and number of paths/endpoints for the given design) from the traditional timing signoff flow and a slack report from the Tempus PI flow:

Traditional Timing Signoff (With IR Derate)		Tempus Power Integrity	
Slack Range	Paths	Slack Range	Paths
-50ps : -40ps	0	-50ps : -40ps	0
-40ps : -30ps	0	-40ps : -30ps	1
-30ps : -20ps	0	-30ps : -20ps	1
-20ps : -10ps	0	-20ps : -10ps	1

**Tempus User Guide**  
Tempus Power Integrity Analysis

---

-10ps : 0ps	0	-10ps : 0ps	68
0ps : 10ps	0	0ps : 10ps	231
10ps : 20ps	6	10ps : 20ps	961
20ps : 30ps	99	20ps : 30ps	824
30ps : 40ps	446	30ps : 40ps	3
40ps : 50ps	1516	40ps : 50ps	2
50ps : 60ps	25	50ps : 60ps	0
60ps+	0	60ps+	0
Total Paths Identified	2092	Total Paths Identified	2092

Here, you can observe the following:

- There are a total of 2092 paths.
- In the case of the traditional timing signoff flow, there are no negative slacks, and the slack bin starts at the +50ps to +60ps range. The worst path is in the +10 to +20ps slack range.
- In the case of the Tempus PI true signoff flow, 71 paths are failing timing with negative slack values. The worst path is in the -30ps to -40ps range.
- Using the Tempus PI generated vectors with a more aggressive path analysis in the neighborhood, slack has moved to a negative range. This indicates a major shift in the overall delay.

These 71 failing paths that can affect performance on silicon can be fixed using the Tempus PI ECO feature.

## 10.2 Licensing Requirements

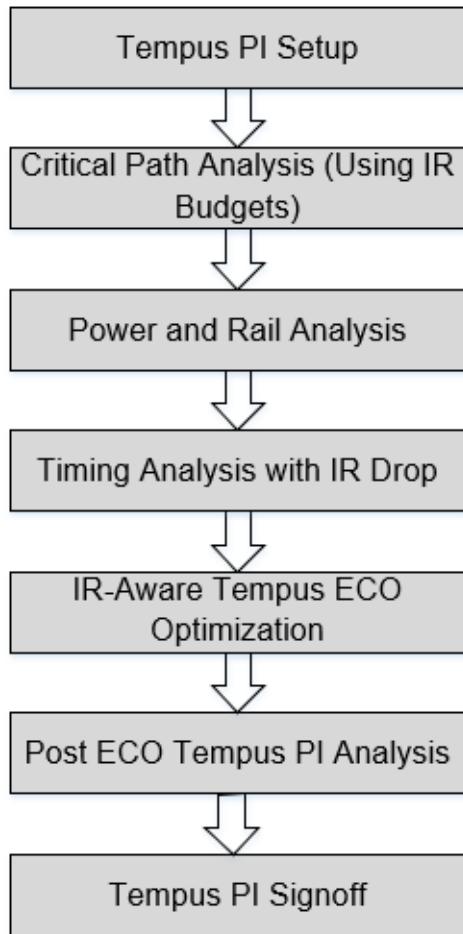
Tempus PI can be invoked from either a Tempus or a Voltus session. This feature requires 3 licenses: Tempus XL (TPS200), Voltus XL (VTS200), and the Tempus Power Integrity Option (TPS600). The following table describes the licensing use model:

Software Used	Base License	Command Used	Additional Licenses

Tempus	Tempus XL	set_power_analysis_mode -enable_tempus_pi true	Tempus PI + Voltus XL
Voltus	Voltus XL	set_power_analysis_mode -enable_tempus_pi true	Tempus PI + Tempus XL

## 10.3 Tempus Power Integrity Flow

The following diagram illustrates the Tempus PI flow:



To perform Tempus PI analysis, perform the following steps:

1. Tempus PI Setup: Load the input design database. This includes LEF, timing libraries, netlist, SDC, DEF, and SPEF.

Timing libraries are needed for multiple voltages below the nominal operating voltage, preferably in the step size of 10% to 15% of the lowest voltage. This is required for accurate STA analysis with voltage interpolation because timing analysis will be done at the software computed post-IR effective instance voltages, instead of the nominal voltage. Recommendation is to have at least three different voltage libraries to enable accurate non-linear delay interpolation.

2. Critical Path Analysis: Identify the timing-critical and IR-sensitive paths using the user-specified IR budgets. The identified critical paths are scheduled in dynamic vectorless simulation.

The user-specified IR budgets are the estimated worst IR drops expected in the design.

3. Power and Rail Analysis: Perform directed vectorless power analysis by scheduling the critical paths identified in step 2. After power analysis is done, run rail analysis to generate the Effective Instance Voltage (EIV) database.
4. Timing Analysis with IR Drop: Specify the EIV database generated from rail analysis as input to timing analysis, and then rerun STA analysis. The IR-induced timing violations are reported post this step.
5. IR-Aware Tempus-ECO Optimization: Perform ECO fixing on timing violations reported after IR-aware STA analysis.
6. Post-ECO Tempus PI Analysis: Run the Tempus PI analysis again to re-check the results after Tempus ECO.

This flow generates an EIV database with better timing and potentially better IR drop.

- 1.

## 10.4 Performing Tempus PI Analysis

This section describes the various inputs and outputs of the Tempus PI analysis flow.

### ***Required Input Files***

- Design data (LEF, timing libraries, netlist, SDC, DEF, SPEF)
- Technology PGV

### ***Procedure***

To perform Tempus PI analysis, complete the following steps:

## Step 1: Tempus STA Analysis Using IR Derates

1. Load the design data.

### Tcl command:

```
read_lib -lef $lefs
read_view_definition ../design/viewDefinition.tcl
read_verilog ../design/postRouteOpt.enc.dat/super_filter.v.gz
set_top_module super_filter
read_def ../design/super_filter.def.gz
```

### Tcl command - Snippet of viewDefinition.tcl:

```
create_library_set -name LS_wc_1p00 -timing
./standard_cells_1p00_125C.lib

create_library_set -name LS_wc_0p90 -timing
./standard_cells_0p90_125C.lib ;# This library set will be used to
improve interpolation later in the flow

create_library_set -name LS_wc_0p85 -timing
./standard_cells_0p85_125C.lib

create_rc_corner -name RC_wc_125 -T 125

create_delay_corner -name DC_wc_1p00 -rc_corner RC_wc_125 -
library_set LS_wc_1p00
create_delay_corner -name DC_wc_0p85 -rc_corner RC_wc_125 -
library_set LS_wc_0p85

create_constraint_mode -name CM_wc -sdc_files ./design.sdc

create_analysis_view -name AV_wc_1p00 -constraint_mode CM_wc -
delay_corner DC_wc_1p00 ;# This is the non-derated/non-margined view
at 1.0v for IR drop analysis
create_analysis_view -name AV_wc_0p85 -constraint_mode CM_wc -
delay_corner DC_wc_0p85 ;# This is the traditional STA view at 0.85V
with voltage margined/derated for IR drop analysis
```

2. Specify the SPEF file for signal net parasitic information.

### Tcl command:

```
read_spef -rc_corner RC_wc_125
../design/postRouteOpt_RC_wc_125.spef.gz
```

3. Specify the global IR budgets for critical path identification.

**Tcl command:**

```
update_delay_corner -name DC_wc_1p00
-early_estimated_worst_irDrop_factor 0.15
-late_estimated_worst_irDrop_factor 0.15
```

4. Run STA analysis using the voltage derates applied in the previous step.

**Tcl command:**

```
update_timing -full
```

## Step 2: Power and IR Drop Analysis (EIV DB Generation)

1. Specify the directory where the power results will be stored.

**Tcl command:**

```
set_power_output_dir dynVecLessPowerResults
```

2. Define the power analysis mode. The Tempus PI analysis flow is applicable only to the dynamic vectorless power calculation method.

**Tcl command:**

```
set_power_analysis_mode \
 -method dynamic_vectorless \
 -disable_static true \
 -enable_state_propagation true \

 -power_grid_library { \
 ../../data/pgv_dir/tech_pgv/techonly.cl \
 ../../data/pgv_dir/stdcell_pgv/stdcells.cl \
 ../../data/pgv_dir/macro_pgv/macros_pll.cl \
 } \
 -enable_tempus_pi true \

```

3. Define switching activity of the design. This is needed for dynamic vectorless analysis.

**Tcl command:**

```
set_default_switching_activity -input_activity 0 \
 -clock_gates_output 2.0 -seq_activity 0.2
```

The default switching activity for sequential cells (-seq\_activity) in percentage (ratio 0.0 to 1.0) specifies the number of fully expanded data paths that will be scheduled to switch in the Tempus PI flow in each cycle.

4. Define the power simulation period and resolution.

**Tcl command:**

```
set_dynamic_power_simulation -resolution 50ps -period 320ns
```

The dynamic power simulation period defines the number of power analysis cycles to be run.

**Note:** The user-defined sequential activity per cycle

(`set_default_switching_activity -seq_activity`) and the total power analysis simulation period (`set_dynamic_power_simulation -period`) control the number of critical path end-points that will be covered per simulation cycle and the total simulation time. Allowing a larger power simulation period will allow Tempus PI to cover all the end-points.

For more information on setting the appropriate values for sequential activity and simulation period, you can refer to section “[Sequential Activity and Simulation Period](#) on page 215”.

5. Run power analysis.

**Tcl command:**

```
report_power
```

6. Define the rail analysis mode.

**Tcl command:**

```
set_rail_analysis_mode \
 -method dynamic \
 -accuracy hd \
 -analysis_view AV_wc_1p00 \
 -power_grid_library { \
 ../../data/pgv_dir/tech_pgv/techonly.cl \
 ../../data/pgv_dir/stdcell_pgv/stdcells.cl \
 ../../data/pgv_dir/macro_pgv/macros_pll.cl \
 } \
 -enable_rlp_analysis true \
 -verbosity true \
 -temperature 125 \
 -eiv_eval_window switching \
 -eiv_method {bestavg worstavg}
```

7. Optional. If you do not load the CPF file, use the `set_pg_nets` command to define the power nets in the design, and assign attributes to these nets. Then, use `set_rail_analysis_domain` to define the power domains.

**Tcl command:**

```
set_pg_nets -net VDD_AO -voltage 1 -threshold 0.95 -force
set_pg_nets -net VDD_column -voltage 1 -threshold 0.95 -force
set_pg_nets -net VDD_ring -voltage 1 -threshold 0.95 -force
set_pg_nets -net VDD_external -voltage 1 -threshold 0.95 -force
set_pg_nets -net VSS -voltage 0.0 -threshold 0.05 -force
set_rail_analysis_domain -name ALL -pwrnets { VDD_AO VDD_external } \
 -gndnets VSS
```

8. Define the location of the voltage sources for each power/ground net.

**Tcl command:**

```
set_power_pads -net VDD_AO -format xy -file
 ../../design7/super_filter_VDD_AO.pp
set_power_pads -net VDD_external -format xy -file
 ../../design7/super_filter_VDD_external.pp
set_power_pads -net VSS -format xy -file
 ../../design7/super_filter_VSS.pp
```

- 9.** Define the rail simulation resolution, and specify the location of the binary current files generated through dynamic power analysis (step 5).

**Tcl command:**

```
set_dynamic_rail_simulation -resolution 50ps
set_power_data -scale 1 -format current [glob
dynVecLessPowerResults/*.ptiavg]
```

- 10.** Run rail analysis.

**Tcl command:**

```
analyze_rail -output ./dynVecLessRailResults -type domain ALL
```

The Voltus console will display rail analysis statistics, as shown in the following snippet:

```
Rail Analysis completed successfully.
Finished Rail Analysis at 19:32:45 02/10/2019 (cpu=0:01:12,
real=0:02:33, peak mem=1697.00MB)
Current Voltus IC Power Integrity Solution resource usage: (total
cpu=0:09:48, real=1:06:51, mem=1697.00MB)

***Info: Found 70 instances with IR drop violations in design based on
eiv_method: Worst and eiv_eval_window: elapse.
```

### **Step 3: Tempus IR-Aware STA Using EIV Database (**

- 1.** Calculate IR-aware slack by specifying the IR analysis output as input to the Tempus run:

**Tcl command:**

```
set_delay_cal_mode -early_irdrop_data_type best_average \
 -late_irdrop_data_type worst_average
update_delay_corner -name DC_wc_1p00 \
 -irdrop_data ./dyn_rail_phaseIII/ALL_25C_dynamic_1
update_timing -full
report_analysis_summary > pi_summary.gba.rpt
report_timing -max_paths 200000 -retiming path_slew_propagation -
retiming_mode exhaustive > pi_timing.pba.rpt
```

**Note:** The following parameteris specific to the Tempus PI analysis flow:

- ❑ `-irdrop_data directory_name` - specifies the IR drop files or directory of files to apply to both the early and late delay calculation for the specified delay corner object.

#### **Step 4: Tempus ECO Fixing**

Fix the IR drop paths, identified by Tempus PI as timing critical paths, using Tempus ECO.

##### **Tcl command:**

```
set_eco_opt_mode -retime path_slew_propagation
set_eco_opt_mode -pba_effort high
set_eco_opt_mode -allow_multiple_incremental true
set_eco_opt_mode -verbose true
set_eco_opt_mode -eco_file_prefix tempus_pi
checkPlace
deleteFiller
eco_opt_design -setup
```

## **10.5 Tempus PI Result Analysis and Reporting**

You can analyze the timing reports generated from the traditional timing signoff and the Tempus PI flows to determine the instances in the high IR drop region. When you compare the timing reports in the following snapshots, you will observe that the timing path that had positive slack in the traditional analysis flow has negative slack in the Tempus PI analysis flow:

# Tempus User Guide

## Tempus Power Integrity Analysis

---

### ***Worst-case Path Analysis (Traditional)***

```
#####
Generated by: Cadence Voltus IC Power Integrity Solution 19.11-e091_1
OS: Linux x86_64 (Host ID sjfdsl26)
Generated on: Wed Jun 26 02:43:49 2019
Design: super_filter
Command: report_timing -retime path_slew_propagation -path_group pll_clk > timing.rpt
#####
Path 1: MET Setup Check with Pin AO1/SUM_reg_3_15/CK
Endpoint: AO1/SUM_reg_3_15/D (^) checked with leading edge of 'pll_clk'
Beginpoint: bypass (^) triggered by leading edge of 'io_clk'
Path Groups: (pll_clk)
Retime Analysis (Data Path-Slew)
Other End Arrival Time 109.300 (-3.00S) | 109.300 | 0.000
+ Other End Clock Edge 0.000
- Setup 2.300 (+3.00S) | 2.300 | 0.000
+ Phase Shift 380.000
+ CPFR Adjustment 2.600 (+3.00S) | 2.600 | 0.000
- Uncertainty 20.000
= Required Time 469.600 (-3.00S) | 469.600 | 0.000
- Arrival Time 387.200 (+3.00S) | 387.200 | 0.000
= Slack Time 82.400 (-3.00S) | 82.400 | 0.000
= Slack Time(original) 77.100 (-3.00S) | 77.100 | 0.000
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 111.200 (+3.00S) | 111.200 | 0.000
= Beginpoint Arrival Time 111.200 (+3.00S) | 111.200 | 0.000

Cell Arc Edge Fanout Load Retime Retime Retime Arrival Voltage Pin
Slew Incr Delay Time
Delay

- CP ^ ^ 6 8.409 23.000 - - 111.200 0.898 pinA/CP
FE_OFCl_bypass CP ^ -> Q ^ ^ - 4.224 13.400 0.000 32.100 143.300 0.898 pinA/Q
ring/gll31 - ^ 4 4.224 13.600 0.000 1.200 144.500 0.898 pinB/I
ring/gll31 I ^ -> Z ^ ^ - 23.220 66.800 0.000 31.500 176.000 0.898 pinB/Z
column/gll31 - ^ 12 23.220 72.700 9.000 25.700 201.700 0.898 pinC/I
column/gll31 I ^ -> ZN v v - 9.353 43.700 0.000 30.700 232.400 0.898 pinC/ZN
ISO_RULE_ISO_HIER_INST_4/gl - v -> ZN v 11 9.353 46.100 1.500 8.700 241.100 0.898 pinD/B1
ISO RULE ISO HIER INST 4/gl B1 v -> ZN ^ ^ - 6.728 63.900 0.000 44.800 285.900 0.898 pinD/ZN
AO1/add_94_37_I1/g70 - ^ 1 6.728 64.100 8.100 10.400 296.300 0.898 pinE/I
AO1/add_94_37_I1/g70 I ^ -> Z ^ ^ - 29.605 84.600 0.000 42.000 338.300 0.898 pinE/Z
AO1/add_94_37_I1/g702 - ^ 1 29.605 98.400 20.000 48.900 387.200 0.898 pinF/D4 ->

```

# Tempus User Guide

## Tempus Power Integrity Analysis

---

### **Worst-case path analysis (Tempus PI)**

```
#####
Generated by: Cadence Voltus IC Power Integrity Solution 19.11-e091_1
OS: Linux x86_64(Host ID sjfdsl26)
Generated on: Wed Jun 26 02:43:49 2019
Design: super_filter
Command: report_timing -retime path_slew_propagation -path_group pll_clk > timing.rpt
#####
Path 1: MET Setup Check with Pin AO1/SUM_reg_3_15/CK
Endpoint: AO1/SUM_reg_3_15/D (^) checked with leading edge of 'pll_clk'
Beginpoint: bypass (^) triggered by leading edge of 'io_clk'
Path Groups: (pll_clk)
Retime Analysis (Data Path-Slew)
Other End Arrival Time 107.700 (-3.00S) | 107.700 | 0.000
+ Other End Clock Edge 0.000
- Setup 11.300 (+3.00S) | 11.300 | 0.000
+ Phase Shift 380.000
+ CFP Adjustment 3.300 (+3.00S) | 3.300 | 0.000
- Uncertainty 20.000
= Required Time 459.700 (-3.00S) | 459.700 | 0.000
- Arrival Time 485.100 (+3.00S) | 485.100 | 0.000
= Slack Time -25.400 (-3.00S) | -25.400 | 0.000
= Slack Time(original) -46.750 (-3.00S) | -46.750 | 0.000
Clock Rise Edge 0.000
+ Clock Network Latency (Prop) 112.100 (+3.00S) | 112.100 | 0.000
= Beginpoint Arrival Time 112.100 (+3.00S) | 112.100 | 0.000

Cell Arc Edge Fanout Load Retime Retime Retime Arrival Voltage Pin
Slew Incr Delay Delay Time

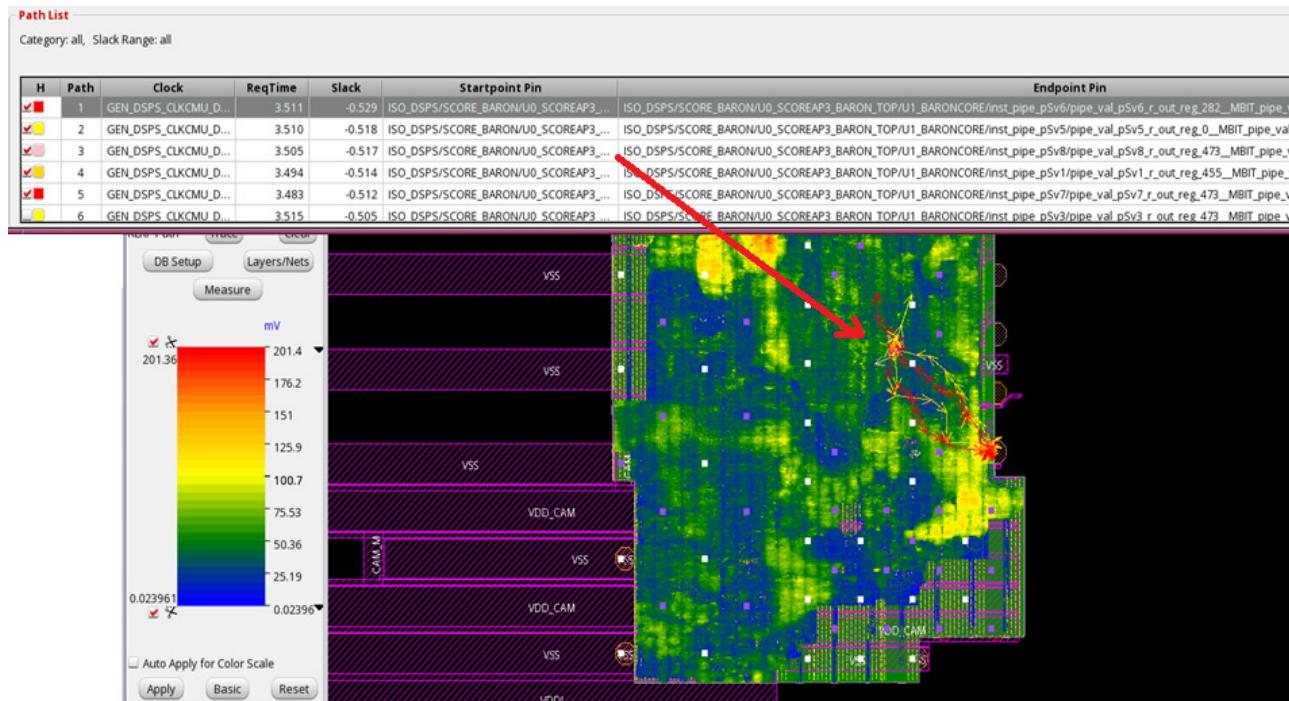
- CP ^ ^ 6 8.409 24.300 - - 112.100 0.904 pinA/CP
FE_OFC1_bypass CP ^ -> Q ^ ^ - 4.224 13.400 0.000 31.700 143.800 0.904 pinA/Q
ring/g1l31 - ^ 4 4.224 13.000 0.100 1.100 144.900 0.880 pinB/I
ring/g1l31 I ^ -> Z ^ ^ - 23.220 67.100 0.000 32.100 177.000 0.880 pinB/Z
column/g1l31 - ^ 12 23.220 58.800 8.400 21.000 198.000 0.773 pinC/I
column/g1l31 I ^ -> ZN v v - 9.353 46.400 0.000 42.900 240.900 0.773 pinC/ZN
ISO_RULE_ISO_HIER_INST_4/gl - v 11 9.353 51.800 1.700 7.200 248.100 0.831 pinD/B1
ISO_RULE_ISO_HIER_INST_4/gl Bl v -> ZN ^ - 6.728 69.100 0.000 49.500 297.600 0.831 pinD/ZN
AO1/add_94_37_I1/g70 - ^ 1 6.728 52.400 8.500 3.700 301.300 0.664 pinE/I
AO1/add_94_37_I1/g70 I ^ -> Z ^ ^ - 29.605 119.600 0.000 81.000 382.300 0.664 pinE/Z
AO1/add_94_37_I1/g702 - ^ 1 29.605 334.500 42.200 102.800 485.100 0.696 pinF/D4 ->

```

## Tempus User Guide

### Tempus Power Integrity Analysis

The following snapshot shows the top five register2register critical paths (*Timing SI -> Debug Timing* menu option) and the corresponding IR drop heat map (*Power Rail -> Power Rail Plots* menu option):



## 10.6 User-Defined Critical Paths Support

The default behavior of Tempus PI is to identify and schedule IR-sensitive and timing-critical paths for dynamic IR drop analysis. This enables designers to identify potential IR-induced violations.

Through the User-Defined Critical Paths (UDCP) support, the tool provides a mechanism for specifying a collection of timing paths that are to be prioritized for scheduling. This feature is useful when designers have prior information on certain timing paths that may switch together, and can potentially cause IR-induced STA violations.

After the user-specified paths are scheduled, the Tempus PI algorithm for identifying and scheduling IR-sensitive paths remains the same for the remaining simulation time.

### Tcl command:

```
set_power_analysis_mode -user_defined_critical_paths <timing path collection>
```

This command must be specified before the `report power` command.

The UDCP collection can be built by using the `-collection` option of the `report_timing` command. Refer to the Tempus Text Command Reference document for more details on this option.

## 10.7 Enabling Proximity Aggressors in Tempus PI

Proximity aggressors refer to those instances which, when switching, can induce additional IR drop on the critical victim instances lying in the vicinity. Such aggressor instances are resistively linked with the victim instances via PG grid.

The Tempus PI flow identifies and schedules the timing-critical and voltage-sensitive paths, along with switching the fanout cone of the critical victim instances using the state propagation algorithm. Though some aggressors for these critical instances are implicitly modeled due to the transient behavior of toggling fanout cones, the aggressor identification and scheduling for each victim instance is not explicitly done, which could lead to optimism in the analysis.

Tempus PI provides an additional capability for proximity aggressor modeling, wherein:

- An aggressor database is generated containing top N resistively-linked aggressors to each of the instances in the design, N being user controlled.
- The generated aggressor database is consumed by the `report_power` command to schedule the aggressors for all critical instances.
- Timing Window (TW) overlap is checked while scheduling these aggressors, and aggressors that are not overlapping with victim's TW are dropped.

### Tcl commands:

```
Specify the number. of aggressors for each victim
redirect -quiet {puts "setvar proximity_level 10"} > rail.inc
set_advanced_rail_options -voltus_rail_include_file_begin
./rail.inc

Generate proximity aggressor database
analyze_resistance -domain ALL -output_dir agg_db -method rlrp

#Link the generated aggressor DB
set_power_analysis mode -specify_aggressor_db_path
agg_db/ALL_25C_reff_1
```

## Sample Script

```
Load the design
read_lib -lef $lefs
read_physical -lefs $lefs
read_view_definition read_mmmc ../design/viewDefinition.tcl
read_verilog ../design/postRouteOpt.enc.dat/super_filter.v.gz
set_top_module super_filter
read_netlist ../design/postRouteOpt.enc.dat/super_filter.v.gz -top super_filter
init_design
read_def ../design/super_filter.def.gz
read_spef -rc_corner RC_wc_125 ../design/postRouteOpt_RC_wc_125.spef.gz

#Apply estimated IR drop budgets
update_delay_corner -name DC_wc_1p00
-early_estimated_worst_irDrop_factor 0.15
-late_estimated_worst_irDrop_factor 0.15

#Configure Tempus PI power analysis
set_power_output_dir dynVecLessPowerResults
set_power_analysis_mode \
 -method dynamic_vectorless \
 -disable_static true \
 -enable_state_propagation true \
 -power_grid_library { \
 ../data/pgv_dir/tech_pgv/techonly.cl \
 ../data/pgv_dir/stdcell_pgv/stdcells.cl \
 ../data/pgv_dir/macro_pgv/macros_pll.cl \
 } \
 -enable_tempus_pi true
set_default_switching_activity -input_activity 0 \
 -period 4.0 -clock_gates_output_ratio 0.5 -sequential_activity 0.2
set_dynamic_power_simulation -resolution 10ps -period 20ns

#Configure rail analysis
set_rail_analysis_mode \
 -method dynamic \
 -accuracy hd \
 -analysis_view AV_wc_1p00 \
 -power_grid_libraries { \
 ../data/pgv_dir/tech_pgv/techonly.cl \

```

## Tempus User Guide

### Tempus Power Integrity Analysis

---

```
..../data/pgv_dir/stdcell_pgv/stdcells.cl \
..../data/pgv_dir/macro_pgv/macros_pll.cl \
} \
-enable_rlrp_analysis true \
-verbosity true \
-temperature 125 \
-eiv_eval_window switching \
-eiv_method {bestavg worstavg}
set_pg_nets -net VDD_AO -voltage 1 -threshold 0.95 -force
set_pg_nets -net VDD_column -voltage 1 -threshold 0.95 -force
set_pg_nets -net VDD_ring -voltage 1 -threshold 0.95 -force
set_pg_nets -net VDD_external -voltage 1 -threshold 0.95 -force
set_pg_nets -net VSS -voltage 0.0 -threshold 0.05 -force
set_rail_analysis_domain -domain_name ALL -pwrnetspower_nets { VDD_AO VDD_external
} -gndnetsground_nets VSS
set_power_pads -net VDD_AO -format xy -file/design/super_filter_VDD_AO.pp
set_power_pads -net VDD_external -format xy -file
..../design/super_filter_VDD_external.pp
set_power_pads -net VSS -format xy -file/design/super_filter_VSS.pp
set_dynamic_rail_simulation -resolution 10ps

Specify the number of aggressors for each victim
redirect -quiet {puts "setvar proximity_level 10"} > rail.inc
set_advanced_rail_options -voltus_rail_include_file_begin ./rail.inc

#Generate proximity aggressor DB
analyze_resistance -domain ALL -output_dir agg_db -method rlrp

#Run power analysis
set_power_analysis_mode -specify_aggressor_db_path agg_db/ALL_25C_reff_1
report_power

#Run rail analysis
set_power_data -scale 1 -format current [glob dynVecLessPowerResults/*.ptiavg]
analyze_rail -output_dir ./dynVecLessRailResults -type domain ALL

#Run STA using EIV DB
set_delay_cal_mode -early_irdrop_data_type best_average \
 -late_irdrop_data_type worst_average
update_delay_corner -name DC_wc_1p00 \
 -irdrop_data ./dyn_rail_phaseIII/ALL_25C_dynamic_1
```

```
update_delay_corner -name DC_wc_1p00 \
 -timing_condition {TC1 TC2} \
 -irdrop_data ./dyn_rail_phaseIII/ALL_25C_dynamic_1
update_timing -full
report_analysis_summary > pi_summary.gba.rpt
report_timing -max_paths 200000 -retime path_slew_propagation -retime_mode
exhaustive -max_slack 0 > pi_timing.pba.rpt
```

## 10.8 Sequential Activity and Simulation Period

### Determining Sequential Activity

Sequential activity is one of the important input parameters, which govern the overall toggling in the design, during directed vectorless simulations. If you are a user of the traditional vectorless dynamic analysis, you would already know what sequential activity to provide, as the same setting serves as input to both the traditional and Tempus PI vectorless runs.

However, if you are primarily a user of the vector-based dynamic analysis, then an existing capability from Voltus can be used to report the average sequential activity based on the input vector. This reported average sequential activity can be used (along with some margining) as input sequential activity in the Tempus PI setup.

#### Tcl command:

```
set_power_include_file ./pm.inc
#Contents of "pm.inc":
ChipPwr rActivityCalculationCycleCountFix 1
ChipPwr rReportInstanceInGateVcdFlow 1
```

The above command is to be added before running `report_power` in the vector-based analysis.

The average sequential activity will be reported in the output file "<output power directory>/voltus\_power.stateprop.stats".

### Sample Output

---

```
* Voltus Power Analysis - Power Calculator - Version v21.10-d109_1 64-bit
(06/08/2020 10:34:09)
* Copyright 2007, Cadence Design Systems, Inc.
*
* Date & Time: 2020-Jun-17 17:02:36 (2020-Jun-18 00:02:36 GMT)
```

```
*
*-----
*
* Total number of data nets : 3921235
* Constant data nets skipped : 0
* NoTiming data nets skipped : 0
* Average data activity : 0.291627
* Number of iterations : 0
* Sequential activity : 0.05
*
...
...
```

### Determining Simulation period

Simulation period determines the simulation period for Tempus PI dynamic analysis. Larger the simulation period, more will be the number of paths covered in the analysis.

The following formula can be used for guidance to set the simulation period:

$$\text{sim\_cycles} = (2/\text{seq\_act}) * N$$

Where,

- seq\_act: input sequential activity (for eg: 0.05, 0.1 etc.)
- N: number of nworst paths to be covered per endpoint

$$\text{sim\_period} = \text{sim\_cycles} * \langle \text{time period of dominant clock} \rangle$$

Recommendation is to have  $N \geq 1$ , that is, at least one path per endpoint should be scheduled in the analysis.

## 10.9 Tempus PI Related STA Commands

The following STA commands allow you to customize the Tempus PI flow:

- set\_delay\_cal\_mode
- create\_delay\_corner/update\_delay\_corner
- set\_irdrop\_mode

These are described below.

**set\_delay\_cal\_mode**

:

Option Name	Description
-irdrop_data_type {best_average worst best average worst_average}	Allows you to choose from the different EIV computation mechanisms of analyze_rail for STA analysis.  This type applies to both early and late analysis.  <i>Default:</i> worst  See set_rail_analysis_mode -eiv_method for more details.
-early_irdrop_data_type {best_average worst best average worst_average}	Allows you to choose from the different EIV computation mechanisms of analyze_rail for early STA analysis.  This option takes precedence over -irdrop_data_type.  <i>Default:</i> worst  See set_rail_analysis_mode -eiv_method for more details.
-late_irdrop_data_type {best_average worst best average worst_average}	Allows you to choose from the different EIV computation mechanisms of analyze_rail for late STA analysis.  This option takes precedence over -irdrop_data_type.  <i>Default:</i> worst  See set_rail_analysis_mode -eiv_method for more details.
-irDrop_window_based {none late early both}	Specifies the analysis type for which the window-based EIVs are to be used for STA analysis.  For the remaining analysis types, the elapse EIV will be used.  Refer to the -eiv_eval_window option of set_rail_analysis_mode for more details on the window and elapse EIV.

**Tempus User Guide**  
Tempus Power Integrity Analysis

---

Option Name	Description
<code>-irDrop_default_scaler &lt;value&gt;</code>	<p>Specifies the default IR drop to be assumed for instances that are not scheduled in IR analysis for both the early and late STA analysis.</p>
	<p>Value to be specified as ratio of VDD. Allowed values are between 0 and 1.</p>
	<p><i>Default:</i> 0</p>
<code>-early_irDrop_default_scaler &lt;value&gt;</code>	<p>Specifies the default IR drop to be assumed for instances that are not scheduled in IR analysis for the early STA analysis.</p>
	<p>Value to be specified as ratio of VDD. Allowed values are between 0 and 1.</p>
	<p>This option takes precedence over – <code>irDrop_default_scaler</code>.</p>
	<p><i>Default:</i> 0</p>
<code>-late_irDrop_default_scaler &lt;value&gt;</code>	<p>Specifies the default IR drop to be assumed for instances that are not scheduled in IR analysis for the late STA analysis.</p>
	<p>Value to be specified as ratio of VDD. Allowed values are between 0 and 1.</p>
	<p>This option takes precedence over – <code>irDrop_default_scaler</code>.</p>
	<p><i>Default:</i> 0</p>
<code>-irDrop_value_check_threshold &lt;value&gt;</code>	<p>Specifies a maximum value of IR drop, above which warning message IMPDC-3280 will be displayed. Value is specified as ratio of VDD. Allowed values are between 0 and 1.</p>
	<p><i>Default:</i> 0.25</p>
	<p><b>Note:</b> This option only controls the warning message, and does not limit the IR drop used in STA analysis.</p>
<code>-irDrop_value_guard_threshold &lt;value&gt;</code>	<p>Limits the EVs of all instances to the specified value for STA analysis. The value is specified as ratio of VDD. Allowed values are between 0 and 1.</p>
	<p><i>Default:</i> 1</p>

**Tempus User Guide**  
Tempus Power Integrity Analysis

---

Option Name	Description
<code>-use_diff_volt_in_eiv_timing_analysis &lt;bool&gt;</code>	<p>When set to <code>true</code>, it enables a differential mode that is needed when the IR analysis supply voltage is different from the STA nominal supply voltage.</p> <p>In this mode, the tool uses the IR drop values instead of EIVs from the EIV database, and then applies these IR drop values in STA analysis nominal supply voltage to compute EIVs.</p> <p><i>Default:</i> false where EIV values are directly used as effective voltages for delay calculation.</p>

### **create\_delay\_corner/update\_delay\_corner**

---

Option Name	Description
<code>-irdrop_data &lt;files or directories&gt;</code>	Specifies the IR drop files or directories to be used in both the early and late delay calculation for the specified delay corner object.
<code>-early_irdrop_data &lt;files or directories&gt;</code>	Specifies the IR drop files or directories to be used for the early corner object.  This option takes precedence over <code>-irdrop_data</code> .
<code>-late_irdrop_data &lt;files or directories&gt;</code>	Specifies the IR drop files or directories to be used for the late corner object.  This option takes precedence over <code>-irdrop_data</code> .
<code>-estimated_worst_irDrop_factor &lt;value&gt;</code>	Specifies the estimated IR drop to be used for all instances in both the early and late STA analysis. Value specified is in ratio of VDD.  This value is honored only when the EIV database has not been specified using any of the <code>*irdrop_data</code> options.  <i>Default:</i> 0

**Tempus User Guide**  
Tempus Power Integrity Analysis

---

<b>Option Name</b>	<b>Description</b>
<code>-early_estimated_worst_irDrop_factor &lt;value&gt;</code>	<p>Specifies the estimated IR drop to be used for all instances in the early STA analysis. Value specified is in ratio of VDD.</p> <p>This value is honored only when the EIV database has not been specified using any of the <code>*irdrop_data</code> options.</p> <p>This option takes precedence over the <code>-estimated_worst_irDrop_factor</code> option.</p> <p><i>Default:</i> 0</p>
<code>-late_estimated_worst_irDrop_factor &lt;value&gt;</code>	<p>Specifies the estimated IR drop to be used for all instances in the late STA analysis. Value specified is in ratio of VDD.</p> <p>This value is honored only when the EIV database has not been specified using any of the <code>*irdrop_data</code> options.</p> <p>This option takes precedence over the <code>-estimated_worst_irDrop_factor</code> option.</p> <p><i>Default:</i> 0</p>

### **set\_irdrop\_mode**

---

<b>Option Name</b>	<b>Description</b>
<code>-corner &lt;delay corner name&gt;</code>	Specifies the delay corner to which the command needs to be applied.
<code>-clock</code>	Applies the specified factors and thresholds to the clock network only.  Default: Applies to both the clock and data networks
<code>-data</code>	Applies the specified factors and thresholds to the data network only.  Default: Applies to both the clock and data networks

-	Specifies a factor by which both the early and late IR drop values are scaled. A different scaling factor can be specified for each PG rail.
-	Specifies a factor by which the early IR drop values are scaled. A different scaling factor can be specified for each PG rail.
-	This option takes precedence over - <code>irdrop_guard_band_scale_factor</code> .
-	Specifies a factor by which the late IR drop values are scaled. A different scaling factor can be specified for each PG rail.
-	This option takes precedence over - <code>irdrop_guard_band_scale_factor</code> .
<code>-irdrop_min_threshold {&lt;rail name&gt;@&lt;absolute_min_irdrop_threshold&gt;}</code>	Specifies the minimum limit for the IR drop used in the design across all instances for both the early and late analysis.  All instances having IR drop lower than the specified value will be assumed to have the user-specified threshold as the IR drop.
<code>-early_irdrop_min_threshold {&lt;rail name&gt;@&lt;absolute_min_irdrop_threshold&gt;}</code>	Specifies the minimum limit for the IR drop used in the design across all instances for the early STA analysis.  All instances having IR drop lower than the specified value will be assumed to have the user-specified threshold as the IR drop.
<code>-late_irdrop_min_threshold {&lt;rail name&gt;@&lt;absolute_min_irdrop_threshold&gt;}</code>	This option takes precedence over - <code>irdrop_min_threshold</code> .  Specifies the minimum limit for the IR drop used in the design across all instances for the late STA analysis.  All instances having IR drop lower than the specified value will be assumed to have the user-specified threshold as the IR drop.
	This option takes precedence over - <code>irdrop_min_threshold</code> .

**Tempus User Guide**  
Tempus Power Integrity Analysis

---

## **Low Power Flows**

---

- 11.1 [Low Power Overview](#) on page 224
- 11.2 [Low Power Flow](#) on page 224
  - [Low Power Flow for CPF](#) on page 224
  - [Low Power Flow for PGV Flow](#) on page 227
- 11.3 [PGV Flows](#) on page 229
  - [PGV SMSC Flow](#) on page 229
  - [PGV MMMC Flow](#) on page 232
- 11.4 [Importing the Design in Low Power Flow](#) on page 235
- 11.5 [Non-MMMC Flow Settings in Low Power Flow](#) on page 235

## 11.1 Low Power Overview

Along with capturing the design and technology-related power constraints, the Common Power Format (CPF) defines timing analysis views based on a combination of different operating corners, power modes, and timing library sets. CPF does not contain RC corner information. To update the RC corners for each timing view, you must use the Tempus command to define the RC corners.

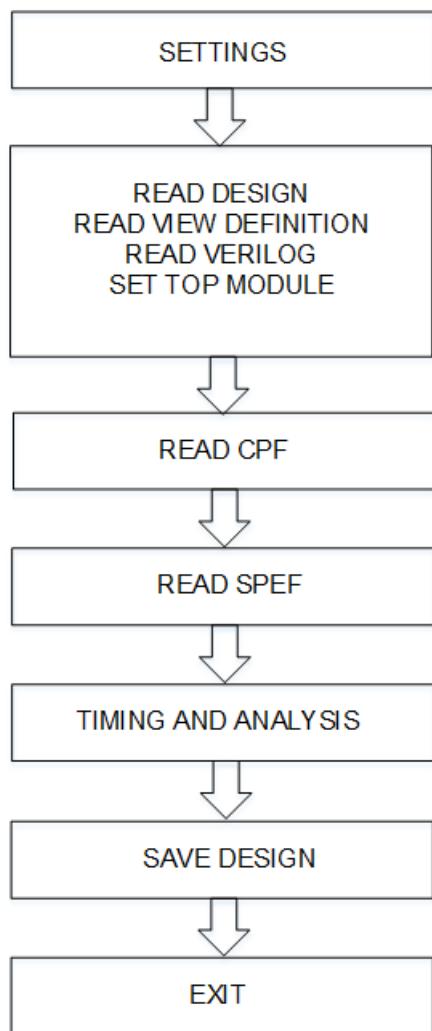
## 11.2 Low Power Flow

The Low Power flow is described below for CPF and PG Verilog (PGV):

### Low Power Flow for CPF

The below flow shows the steps for setting-up Tempus using the CPF file:

**Figure 11-1 Low Power Flow for CPF**



### ***Sample CPF File Script***

You can use the following settings for Tempus using the CPF file:

1. Specify the number of CPUs to enable multi-threading:

```
set_multi_cpu_usage -localCpu n
```

2. Load the design data:

```
read_view_definition <abc. tel>
```

3. Load the libraries:

```
create_library_set
```

4. Create RC and Delay calculation corners:

```
create_rc_corner
create_delay_corner
```

5. Group SDC constraints files

```
create_constraint_mode
```

6. Create an analysis view object that associates a delay calculation corner with a constraint mode.

```
create_analysis_view
```

7. Set the analysis views used for setup and hold analysis:

```
set_analysis_view
```

8. Schedule reading of structural gate-level verilog netlist:

```
read_verilog <abc.v>
```

9. Set the top module name, and checks for consistency between Verilog netlist and timing libraries:

```
set_top_module <abc>
```

10. Specify the CPF file, which contains the power domain definitions:

```
read_power_domain -cpf <xyz.cpf>
```

11. Load the cell parasitics:

```
read_spef -rc_corner
```

12. Run a full timing update for the current design:

```
update_timing -full
```

13. Generate a report of all defined views in the design:

```
report_analysis_views
```

14. Specify the instance(s) for which the report is to be printed.

```
report_instance_library <instance>
```

15. Report the voltage of the cell or net timing arc.

```
report_delay_calculation -voltage
```

16. Generate the timing report:

```
report_timing
```

17. Save the design:

```
save_design -overwrite
```

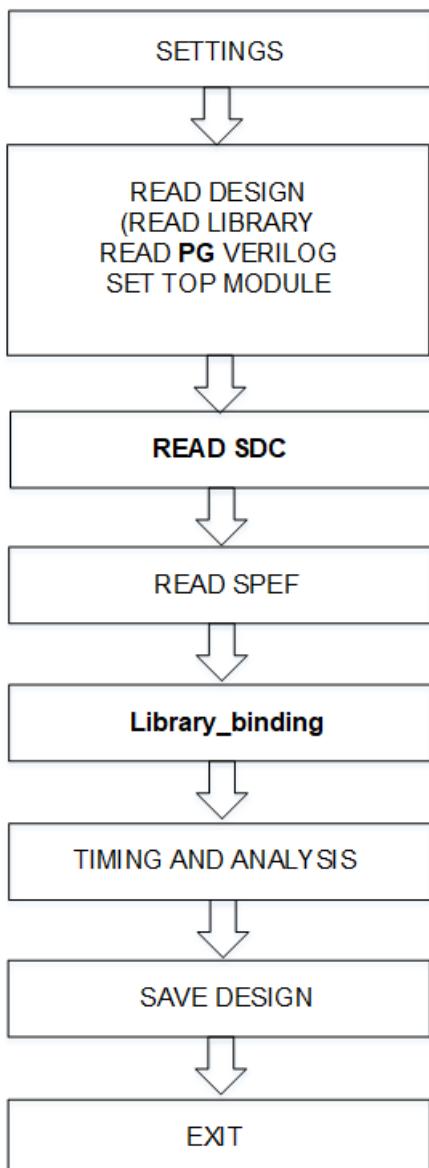
18. Exit the design:

exit

## Low Power Flow for PGV Flow

The following figure shows the steps for setting-up Tempus using the PGV flow:

**Figure 11-2 Low Power Flow for PGV**



### **Sample PGV Flow Script**

You can use the following settings for Tempus using the PGV flow:

1. Specify the number of CPUs to enable multi-threading:

```
set_multi_cpu_usage -localCpu n
```

2. Read library file.

```
read lib < >
```

3. Schedule reading of structural gate-level verilog netlist:

```
read_verilog <PG verilog.v>
```

4. Set the top module name, and check for consistency between the Verilog netlist and timing libraries:

```
set_top_module <abc>
```

5. Load the timing constraints file:

```
read_sdc < >
```

6. Load the cell parasitics:

```
read_spf -rc_corner
```

7. Initiate power intent inference and library binding:

```
set_lib_binding_by_voltage -power_net_voltages
```

8. Run a full timing update for the current design:

```
update_timing -full
```

9. Generate a report of all defined views in the design.

```
report_analysis_views
```

10. Report library related information for an instance:

```
report_instance_library <instance>
```

11. Report the voltage of the cell or net timing arc

```
report_delay_calculation -voltage
```

12. Generate the timing report:

```
report_timing
```

13. Overwrite the previously saved session with a new session:

```
save_design -overwrite
```

14. Exit the design:

```
exit
```

## 11.3 PGV Flows

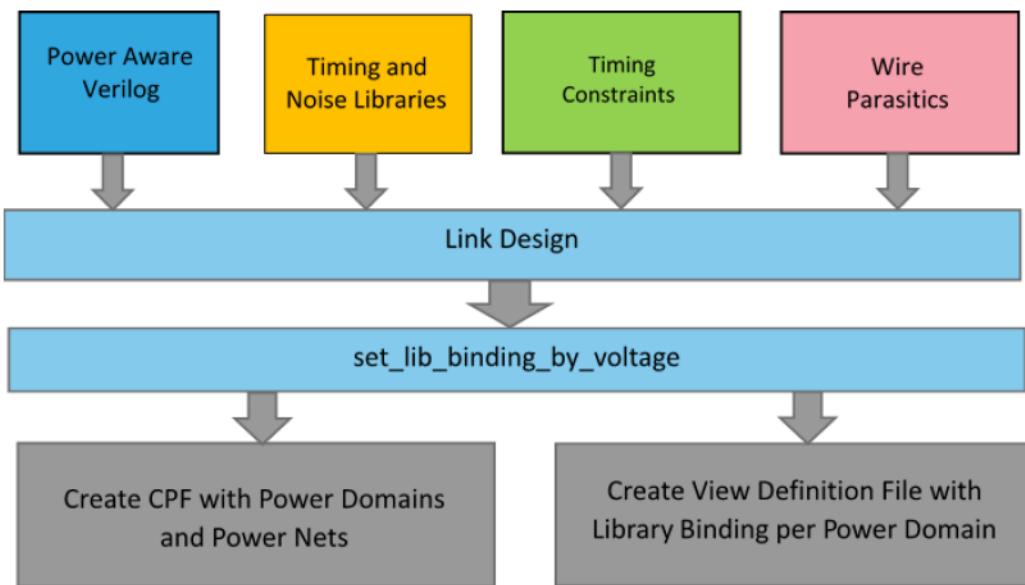
The PGV flows in Tempus can be categorized as below, based on the various timing analysis practices:

- PGV SMSC Flow (without MMMC)
- PGV MMMC Flow

### PGV SMSC Flow

In the PGV SMSC flow, timing analysis can be performed in a separate session for each view (SMSC flow). The following figure represents the PGV SMSC flow:

**Figure 11-3 PGV SMSC Flow**



In the PGV SMSC flow, Tempus requires a PGV netlist as input along with other essential information required for static timing analysis. Run the `set_lib_binding_by_voltage` command after running the `set_top_module` command, which initiates auto inferencing of power intent and view definition. The `set_lib_binding_by_voltage` command provides voltages for each power net in the design.

In PGV, the instantiation of each library cell includes power and ground pins and their connectivity, as shown in the following example:

```
module block1 (clk, in1, in2, out1, rail1, rail2, gnd);
input clk, in1, in2;
```

## Tempus User Guide

### Low Power Flows

---

```
output out1;
input rail1,rail2, gnd;
.BUFX1 i0 (.A(in1), .Y(n0), .VDD12(rail1), .VSS(gnd));
.NOR_HVT i1 (.N_CORE_O(n1), .CNTL_CORE_I(n0), .PORTS_ON_33_I(in2), .VDD12(rail1),
.VDD$33(rail2), .VSS(gnd));
.BUFX1 i2 (.A(n1), .Y(n3), .VDD12(rail1), .VSS(gnd));
.DFFHQX1 i3 (.CK(clk), .D(n3), .Q(out1), .VDD12(rail1), .VSS(gnd));
endmodule
```

### ***SMSC PGV Flow Script***

You can assert voltages for power rails using the `set_lib_binding_by_voltage` command, which initiates power intent inference and library binding. The following example presents the usage of the `set_lib_binding_by_voltage` command:

```
set_lib_binding_by_voltage
-power_net_voltages {<power_net_name> @<Voltage> <power_net_name>@<Voltage>}
[-process <num>]
[-temperature <num>]
```

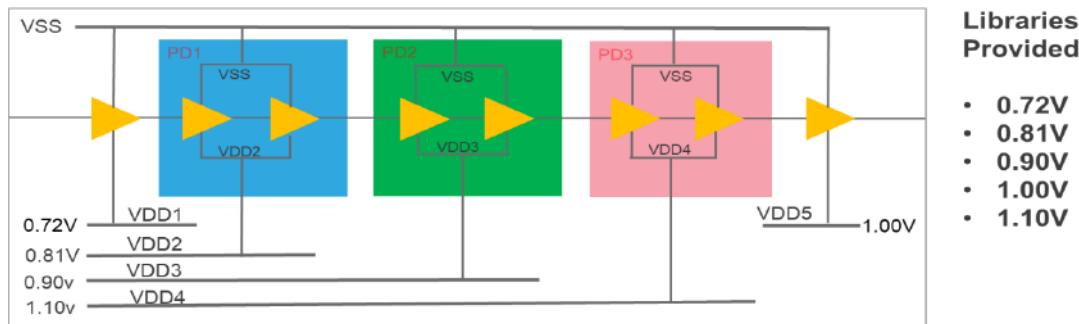
You can specify the process and temperature of the corner using the `-process` and `-temperature` parameters of the `set_lib_binding_by_voltage` command along with the power rail voltages. You can provide all the libraries required for all the power domains upfront using the `read_lib` command. During the library binding, Tempus creates library sets for each power domain-based matching PVT conditions.

Following is the PGV SMSC flow script:

```
read_lib {list of Liberty files}
read_verilog <power_aware_verilog_file>
set_top_module -ignore_undefined_cell <top_name>
read_sdc <sdc_file>
read_spf <spf_file>
set_lib_binding_by_voltage -power_net_voltages {net1@v1 net2@v2 .. } -process 1 -
temperature 125
report_timing ...
```

### ***PGV SMSC Power Intent Inference***

Following is an example of the MSV design:

**Figure 11-4 PGV SMSC Power Intent Interface**

In this example, the PGV flow creates an internal CPF with five power domains, one each for every power net, as shown below:

```

set_cpf_version 1.0e
set_hierarchy_separator "/"
set_design top
create_power_nets -nets VDD1
create_power_nets -nets VDD2
create_power_nets -nets VDD3
create_power_nets -nets VDD4
create_power_nets -nets VDD5
create_ground_nets -nets gnd
create_power_domain -name pd_VDD1 -default
update_power_domain -name pd_VDD1 -primary_power_net VDD1 -primary_ground_net gnd
create_power_domain -name pd_VDD2 -instances {PD1}
update_power_domain -name pd_VDD2 -primary_power_net VDD2 -primary_ground_net gnd
create_power_domain -name pd_VDD3 -instances {PD2}
update_power_domain -name pd_VDD3 -primary_power_net VDD3 -primary_ground_net gnd
create_power_domain -name pd_VDD4 -instances {PD3}
update_power_domain -name pd_VDD4 -primary_power_net VDD4 -primary_ground_net gnd
create_power_domain -name pd_VDD5
update_power_domain -name pd_VDD5 -primary_power_net VDD5 -primary_ground_net gnd
create_global_connection -net VDD1 -domain pd_VDD1 -pins VDD
create_global_connection -net gnd -domain pd_VDD1 -pins VSS
... Similarly for VDD2 thru VDD5
end_design

```

The `set_lib_binding_by_voltage` command also creates the view definition file for single corner analysis and assigns the operating voltage for each power domain accordingly. It also creates library sets for each power domain based on the assigned PVT and associates the corresponding library set for the power domains.

#### PG Net Voltages for Early and Late Analysis in PGV SMSC Flow

You can specify various early and late voltages for power rails using the `-early_power_net_voltages` and `-late_power_net_voltages` parameters of the `set_lib_binding_by_voltage` command. These parameters do not impact the inference of power domains.

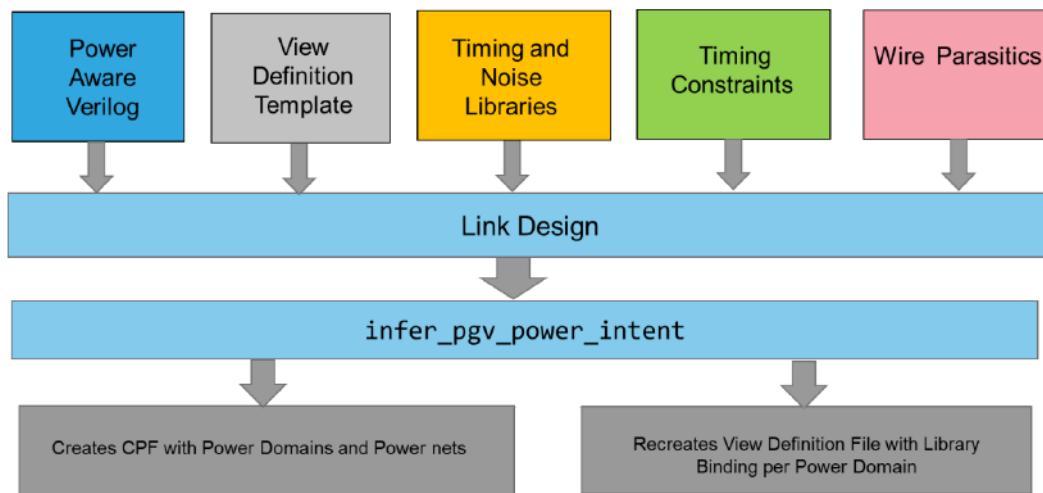
However, library binding will look for a library for both late and early voltages of each instance. As a result, the view definition file written out internally will consist of both early and late library sets and late and early `op_conds`.

Late analysis will use late operating condition and late library set. Similarly, early analysis is done based on early operating condition and early library set.

#### PGV MMMC Flow

In the PGV MMMC flow, timing analysis can be performed concurrently on multiple views. The following figure represents the PGV MMMC flow:

**Figure 11-5 PGV MMMC Flow**



In addition to a PGV netlist, the PGV MMMC flow requires a view definition template file. In this template file, you can define all the required analysis views, including the corresponding RC corner, constraint mode, and delay corner.

However, you do not need to define a delay corner definition to the power domain level. That means, you only need to specify the design-level operating conditions. You can also create a

single library set, which includes all the libraries to cover all the power domain voltages in a given timing analysis corner.

Power domains and power nets are independent of analysis corners or, rather, same for all corners. Only the voltages assigned to power nets are different for different analysis corners. You can define specific voltages for specific power nets in the view definition template file. For this requirement, use the `create_delay_corner -pg_net_voltages` parameter, as shown below:

```
create_delay_corner -pg_net_voltages \
{<power_net_name> @<Voltage> <power_net_name>@<Voltage> ... }
```

This parameter serves the same purpose as `set_lib_binding_by_voltage -power_net_voltages`. After the design is loaded with the template view definition file, you can use the `infer_pgv_power_intent` command to initiate power intent inferencing.

This command creates an individual power domain for each power net, similar to the PGV flow without MMMC (SMSC flow). This command also updates the view definition file to update `delay_corners` with the power domain information.

This command creates library sets for each power domain based on PVT. Power domain PVT is inferred based on the voltage from the associated power net of the domain, and the process and temperature of the design-level PVT for the corner.

### ***View Definition Template***

The view definition template is presented below:

```
create_library_set -name libset_all_dc1\
-timing [list stdLib_volt1.lib stdLib_volt2.lib stdLib_volt3.lib\
IO_volt1.lib IO_volt2.lib IO_volt3.lib]
create_rc_corner -name default_rc_corner
create_delay_corner -name dc1 -library_set [list libset_all_dc1]\ -opcond slow -\
opcond_library stdLib_volt2.lib \
-pg_net_voltages {VDD1@0.535 VDD2@0.7 VDD3@1.05} -rc_corner default_rc_corner
create_library_set -name libset_all_dc2\
-timing [list stdLib_volt4.lib stdLib_volt5.lib stdLib_volt6.lib\
IO_volt4.lib IO_volt5.lib IO_volt6.lib]
create_rc_corner -name default_rc_corner
create_delay_corner -name dc2 -library_set [list libset_all_dc2] -opcond best -\
opcond_library stdLib_volt4.lib \
-pg_net_voltages {VDD1@0.435 VDD2@0.65 VDD3@0.9} -rc_corner default_rc_corner
create_constraint_mode -name mode_func -sdc_files [list func.sdc]
create_analysis_view -name view1_func constraint_mode mode_func -delay_corner dc1
create_analysis_view -name view2_func -constraint_mode mode_func -delay_corner dc2
```

```
set_analysis_view -setup view1_func -hold view1_func
```

**Note:**

- The library set for each delay corner includes libraries for multiple voltages, which are assigned to different power domains during power intent inferencing and recreating the view definition file.
- The `opcond` specified for each delay corner provides the process and temperature values for the corner.
- The voltage for each power net is specified with the `create_delay_corner -pg_net_voltages` parameter.

***MMMC PGV Flow Script***

Following is the PGV MMMC flow script:

```
read_view_definition view_definition_template.tcl
read_verilog <power_aware_verilog_file>
set_top_module -ignore_undefined_cell <top_name>
infer_pgv_power_intent
set_analysis_view -setup [...] -hold [...]
report_timing ...
```

Run the `set_analysis_view` command after running the `infer_pgv_power_intent` command to activate the MMMC views for PGV analysis. Tempus updates the MMMC view definition with power domains for each corner and corresponding library sets after the `set_analysis_view` command is run.

The view definition template file preferably has only one active view. It is recommended not to specify all active views in this file.

***PGV MMMC Power Intent Inference***

Power domain inferencing in the PGV MMMC flow is the same as in the PGV SMSC flow. The view definition template file creation is also very similar, except that the PGV MMMC flow has multiple corners/views defined.

***PG Net Voltages for Early and Late Analysis in PGV MMMC Flow***

Similar to the PGV SMSC flow, you can specify various early and late voltages for power rails in the PGV MMMC flow. You can specify the `-early_pg_net_voltages` and `-`

late\_pg\_net\_voltages parameters of the `create_delay_corner` command in the MMMC view definition template to assign various early/late voltages for power rails in a given corner.

During library binding, libraries for both late and early voltages of each instance will be bound to each cell as late/early libraries. As a result, the view definition template file written out internally will consist of both the early and late library sets and late and early `op_conds` for each corner.

Late analysis will use late operating condition and late library set. Similarly, early analysis is done based on early operating condition and early library set.

## 11.4 Importing the Design in Low Power Flow

Use the following Tempus commands to import the timing libraries, cdb library (for signal integrity analysis), and netlist in low power flow:

```
read_view_definition -cpf cpffile
read_verilog netlist
set_top_module top_module
init_design
```

## 11.5 Non-MMMC Flow Settings in Low Power Flow

Use the below Non-MMMC flow settings in low power flow:

- Load the constraint and parasitic data files:

```
read_sdc constraint_files
read_spf parasitic_data_files
```

- Set the operating conditions for each power domain:

```
set_op_cond condition_name1 -library library_name1 -powerdomain domain_name1
set_op_cond condition_name2 -library library_name2 -powerdomain domain_name2
```

- Load both the Min and Max timing libraries:

```
set_timing_library -max Max -min Min
set_op_cond -max condition_name1 -maxlibrary maxLibrary_name1 -min
condition_name2 -minlibrary minLibrary_name1 -powerdomain domain_name1
```

After completing the above steps, you can run other Tempus commands to continue with the timing flow.

## **Appendix**

---

This chapter covers the following topics:

- [Appendix - Timing Debug GUI](#) on page 237
- [Appendix - Multi-CPU Usage](#) on page 263
- [Appendix - Base Delay, SI Delay, and SI Glitch Correlation with SPICE](#) on page 269
- [Appendix - Supported CPF Commands](#) on page 279
- [CPF 1.0e Script Example](#) on page 280
- [CPF 1.0 Script Example](#) on page 287
- [CPF 1.1 Script Example](#) on page 297
- [Supported CPF 1.0 Commands](#) on page 305
- [Supported CPF 1.0e Commands](#) on page 314
- [Supported CPF 1.1 Commands](#) on page 325
- [Path Exception Priorities](#) on page 337

## Appendix - Timing Debug GUI

- [Overview](#) on page 237
- [Timing Debug Flow](#) on page 238
- [Generating Timing Debug Report](#) on page 239
- [Displaying Violation Report](#) on page 239
- [Analyzing Timing Results](#) on page 239
  - [Viewing Power Domain Information](#) on page 245
- [Creating Path Categories](#) on page 246
  - [Creating Predefined Categories](#) on page 246
  - [Creating New Categories](#) on page 248
  - [Creating Sub-Categories](#) on page 249
  - [Hiding path categories](#) on page 252
  - [Reporting Path Categories](#) on page 252
  - [Using Categories to Analyze Timing Results](#) on page 253
  - [Analyzing MMMC Categories](#) on page 254
  - [Manual Slack Correction of Categories](#) on page 256
- [Editing Table Columns](#) on page 257
  - [Cell Coloring](#) on page 259
- [Viewing Schematics](#) on page 260
- [Running Timing Debug with Interface Logic Models](#) on page 261

### Overview

Tempus provides the Global Timing Debug feature for debugging the timing results. The various Timing Debug forms provide easy visual access to the timing reports and debugging tools.

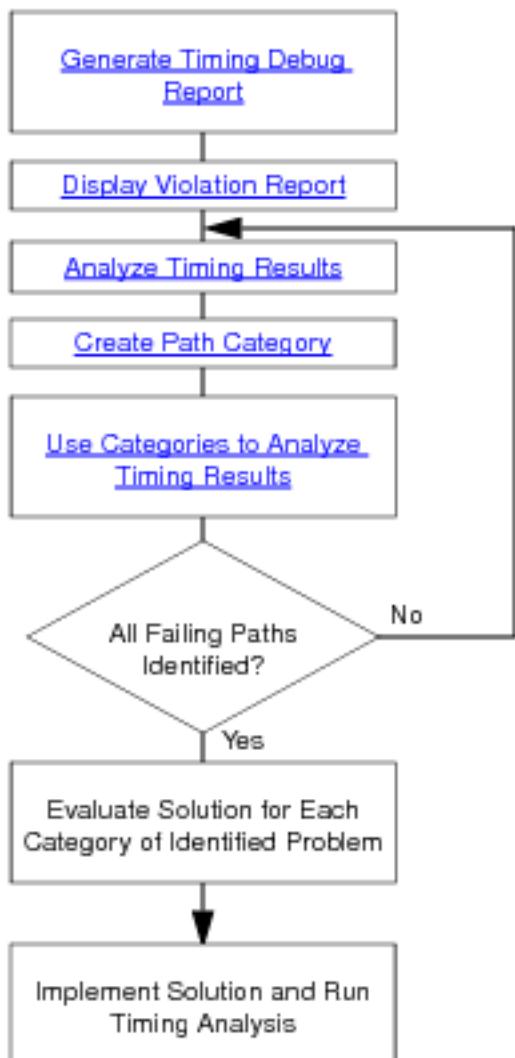
You can group all paths that are failing for the same reason and apply solutions for faster timing closure. You can cross-probe between the timing paths in the timing report and display area under Layout Tab in Tempus.

**Note:** If you have a previously saved timing debug report, you can use the timing debug feature even when the design is not loaded in the Tempus session.

## Timing Debug Flow

You can generate a detailed violation report to list the details of all violating paths. You can then use the timing debug capability to visually identify problems with critical paths in this report. After identifying the problems, you group all paths with the same problem under a single category. You can define several categories to capture all problems related to the violating paths before fixing the problems and running timing analysis again.

Following is the flow for debugging timing results.



**Note:** The file `gtd.pref.tcl` is loaded automatically at launch.

## Generating Timing Debug Report

Tempus uses a machine readable timing report to display timing debug information. The report is generated in the ASCII format and contains details of all violating paths. By default, the report has `.mtarpt` extension.

To generate a violation report, use one of the following options:

- Use `report_timing -machine_readable` command

You can also generate text-format report from a machine readable report.

To generate the text report, use the following:

- Use the `write_text_timing_report` command.
- Use the *Write Textual Timing Report* form.

## Displaying Violation Report

To analyze the timing results, you need to load the machine readable timing report in Tempus.

To display the violation report, use one of the following options:

- Specify the file name in the *Display/Generate Timing Report* form.

**Note:** To select an existing file, deselect the *Generate* option before clicking on the directory icon to the right of the *Timing Report File* field. By default, the global timing debug engine uses the following command to generate a machine-readable timing analysis report for the GUI display:

```
report_timing -machine_readable -max_points 10000 -max_slack 0.75
```

- Use the `load_timing_debug_report` command.

**Note:** Use the Append to Current Report option in the *Display/Generate Timing Report* form to load multiple reports in a single session.

## Analyzing Timing Results

Tempus provides Timing Debug feature to visually analyze timing problems.

You analyze the following data in the *Analyze* tab in the main Tempus window:

## Tempus User Guide

### Appendix

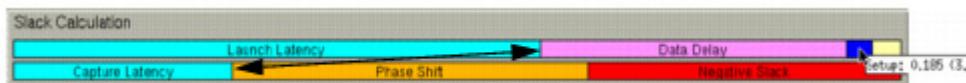
- Visual display of passing and failing paths as a histogram. Failing paths are represented in red and passing paths are represented in green color. The goal of timing debug process is to identify paths that fall in red category.
- Details of the critical paths in the Path list. You identify a critical path in this list for further analyses using the Timing Path Analyzer form.
- Visual display of paths reported in different timing reports. When you load multiple debug reports in a single timing debug session, the paths are displayed in different colors corresponding to the report file they are coming from. You can move the cursor over a path to display the name of the report file.

You analyze the following data in the *Timing Path Analyzer* form:

- Slack calculation bars for arrival and required times. You can identify clock skew issues, latency balancing or large clock uncertainty issues using these bars.

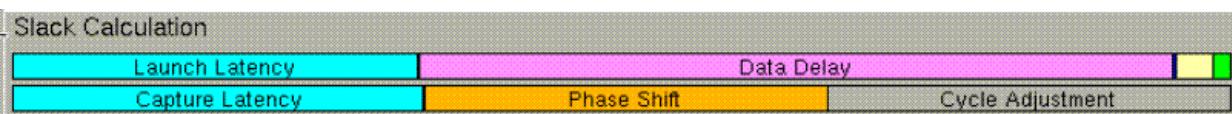
The following examples illustrate the problems that you can identify using the slack calculation bars in the *Timing Path Analyzer* form.

#### Example 9-1



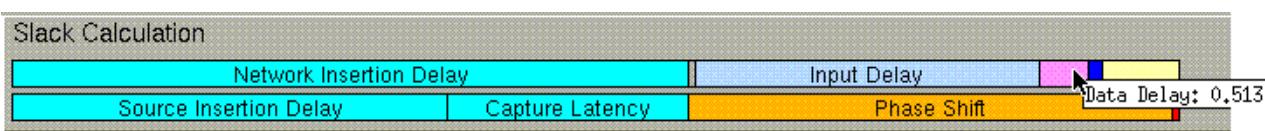
Launch and capture latency components are not aligned. Therefore there can be large clock-latency mismatch in this path.

#### Example 9-2



The cycle adjustment bar in the required time indicates presence of multicycle path.

#### Example 9-3

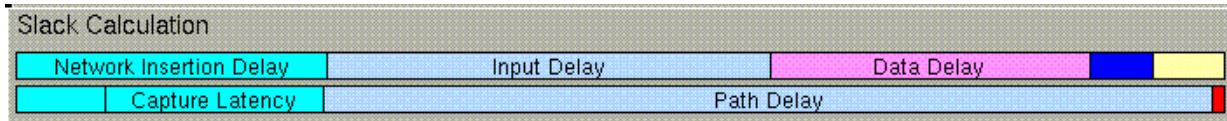


Large input delay in an I/O path is represented by the blue bar in the arrival time.

## Tempus User Guide

### Appendix

#### Example 9-4



Path Delay bar in the required time indicates a `set_max_delay` constraint.

- Path details including launch and capture. This information is provided as tabs in the Timing Path Analyzer form. You can click on a single path to display it in the design display area. You can select each element consecutively to trace the entire path in the design display area. This form has a Status column that indicates the status of the path as follows:

Flag	Description
a	Assign net
b	Blackbox instance
c	Clock net
cr	Cover cell
f	Preplaced instance
i	Ignore net
s	Skip route net
t	"don't touch" net
t	instance marked as "don't touch"
T	instance not marked as "don't touch" when the module it belongs to is marked as "don't touch"
u	Unplaced cell
x	External net

- SDC related to the path. The Path SDC tab displays the SDC constraints related to the selected path. The list contains the name of the SDC file, the line number that indicates the position of the constraint in the SDC file, and the constraint definition.

The commands that can be displayed in the Path SDC tab are:

- `create_clock`

## Tempus User Guide

### Appendix

---

- create\_generated\_clock
- group\_path
- set\_multicycle\_path
- set\_false\_path
- set\_clock\_transition
- set\_max\_delay
- set\_min\_delay
- set\_max\_fanout
- set\_fanout\_load
- set\_min\_capacitance
- set\_max\_capacitance
- set\_min\_transition
- set\_max\_transition
- set\_input\_transition
- set\_capacitance
- set\_drive
- set\_driving\_cell
- set\_logic\_one
- set\_logic\_zero
- set\_dont\_use
- set\_dont\_touch
- set\_case\_analysis
- set\_input\_delay
- set\_output\_delay
- set\_annotated\_check
- set\_clock\_uncertainty
- set\_clock\_latency
- set\_propagated\_clock
- set\_load

- set\_disable\_clock\_gating\_check
- set\_clock\_gating\_check
- set\_max\_time\_borrow
- set\_clock\_groups

When you create a constraint on the command line, the Path SDC tab interactively displays the result of the additional constraint.

**Note:** MMMC views are not displayed interactively.

**Note:** You can create a path category directly from SDC constraints in the Path SDC form. When you right-click a constraint and view the *Create Path Category* form, to see the line number (from the SDC file) and the name of the constraint.

In Tempus, you can also create a path category based on SDC constraint using the `-sdc` parameter of the `create_path_category` command:

```
create_path_category -name category_name -sdc {file_name line_number}
```

where *file\_name* is the name of the constraint file and *line\_number* is the line number of the SDC constraint.

- Schematic display of the path. The Schematics tab displays the gate-level schematic view of the critical path. For more information, see *Viewing Schematics*.
- Timing interpretation for the path. This feature provides rule-based path analysis to help you discover sources of potential timing problems in a path. By default, the software performs the following checks on the following rules:

#### ***Path structure***

- Transparent Latch in Path
- Clock Gating
- Hard Macros
- HVT Cells
- Buffering List
- Net Fanout
- Level Shifters
- Isolation Cells

- Net too long
- Couple capacitance ratio

#### ***Timing and constraints***

- Large Skew
- Divider in Clock Path
- Total SI Delay
- SI Delay
- External Delay

#### ***Floorplan***

- Fixed Cells
- Distance from start to end
- Distance of repeater chain
- Detour
- Multiple power domains

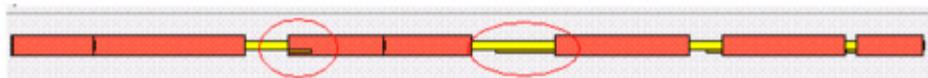
#### ***DRVs***

- Max transitions
- Max capacitance
- Max fanout

You can customize the type of timing information reported. The *Edit Timing Interpretation* GUI Tempus lets you add, modify, or delete rules you want the tool to check and report.

- Timing bar to analyze delays associated with instances and nets in a path. Use this information to identify issues related to large instance or net delays, repeater chains, paths with large number of buffers, and large macro delays. The small bars

superimposed on net delays or within element delays show incremental (longer or shorter) delays due to noise effects:



- Hierarchical representation of the path in the Hierarchy View field. This representation of the path-delay shows the traversal of a path through the design hierarchy drawn on the time axis. A longer arrow means that there are more instances on its path. Use this information to see the module where the path is spending more time or to identify inter-partition timing problems.

### Viewing Power Domain Information

While debugging critical paths on MSV designs, it is useful to be able to identify power domains and low power cells. The Timing Debug feature displays this information in the following ways:

- The level shifters and isolation cells are listed in the Timing Interpretation tab of the Timing Path Analyzer.

Path Structure

- 1.1 Transparent Latch in Path
  - mult\_5\prod\_reg[6]
  - mult517out\_reg[6]
- 1.2 Clock Gating
- 1.3 Hard Macro
- 1.4 HVT Cells (contain HVT)
- 1.5 Buffering Chain (> 5)
- 1.6 Net Fanout (> 5)
- 1.7 Level Shifters
  - CPF\_ISO\_HIER\_INST\_9/g30
  - CPF\_ISO\_HIER\_INST\_9/g30

Timing and Constraints

Floorplan

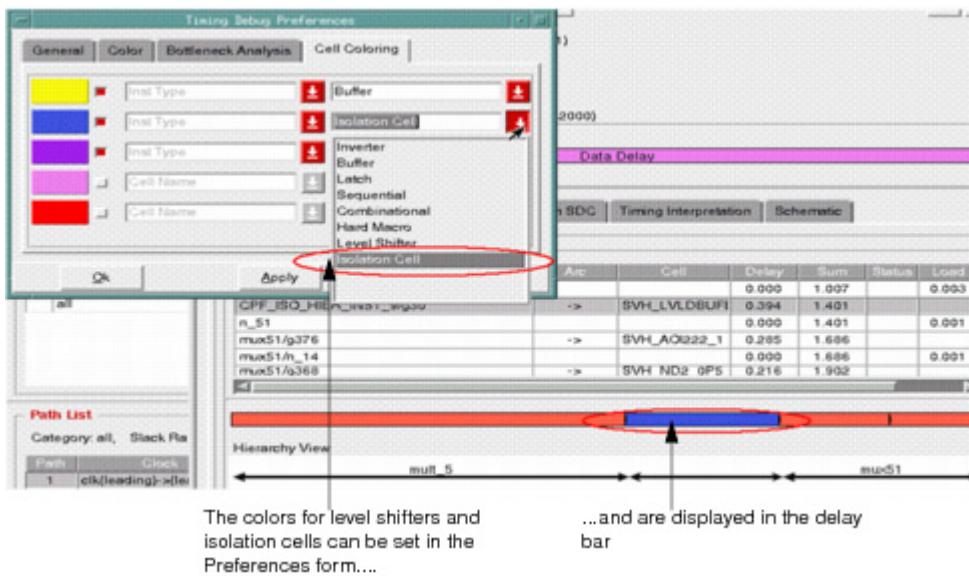
- 3.1 Fixed Cells
- 3.2 Distance from Start to End (> 5000)

Level shifters and Isolation Cells are displayed in the Timing Implementation tab

## Tempus User Guide

### Appendix

- The delay bar of the Timing Path Analyzer can display the level shifters and isolation cells. You can also use the Preferences form to specify the colors in which the level shifters and isolation cells are displayed.



## Creating Path Categories

After analyzing the paths in the timing report, you identify problems in various paths. Then you create a group of paths such that all paths in that group have the same timing problem and can be fixed at the same time. In timing debug such a group of paths is called a category. In Tempus, you can either define your own category or use predefined categories to group your paths. The categories that you define are then displayed in the *Path Category* field in the Analysis tab. The form also displays the paths associated with each category.

### Creating Predefined Categories

There are following predefined categories:

- Basic Path Group  
Creates standard path categories according to basic path groups.

The basic path groups are:

- Register to register
- Input to register
- Register to output

- Input to Output

Registers can be macros, latches, or sequential-cell types. To create categories by basic path groups, choose the *Analysis* tab - *Analysis* drop-down menu - *Path Group Analysis* option.

- Clock Paths

Creates categories according to launch clock - capture clock combinations.

Following categories are created:

- Paths with clock fully contained in a single domain, clk1.
- Paths with clocks starting one clock domain and ending at another, clk1->clk2.

To create categories for clock paths, choose the *Analysis* tab - *Analysis* drop-down menu - *Clock Analysis* option.

- Hierarchical Floorplan

Creates categories according to the hierarchical characteristics of a path.

- View

Creates categories according to the view for which the path was generated.

To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *View Analysis* option.

- False Paths

Creates a category with paths defined as False paths.

To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *Critical False Path* option.

- Bottleneck

Creates categories based on instances that occur often in critical paths.

To create view path categories, choose the *Analysis* tab - *Analysis* drop-down menu - *Bottleneck Analysis* option.

- DRV Analysis

Generates or loads a DRV report containing capacitance, transition, or fanout violations. Paths that are affected by the selected DRV types are grouped in a category.

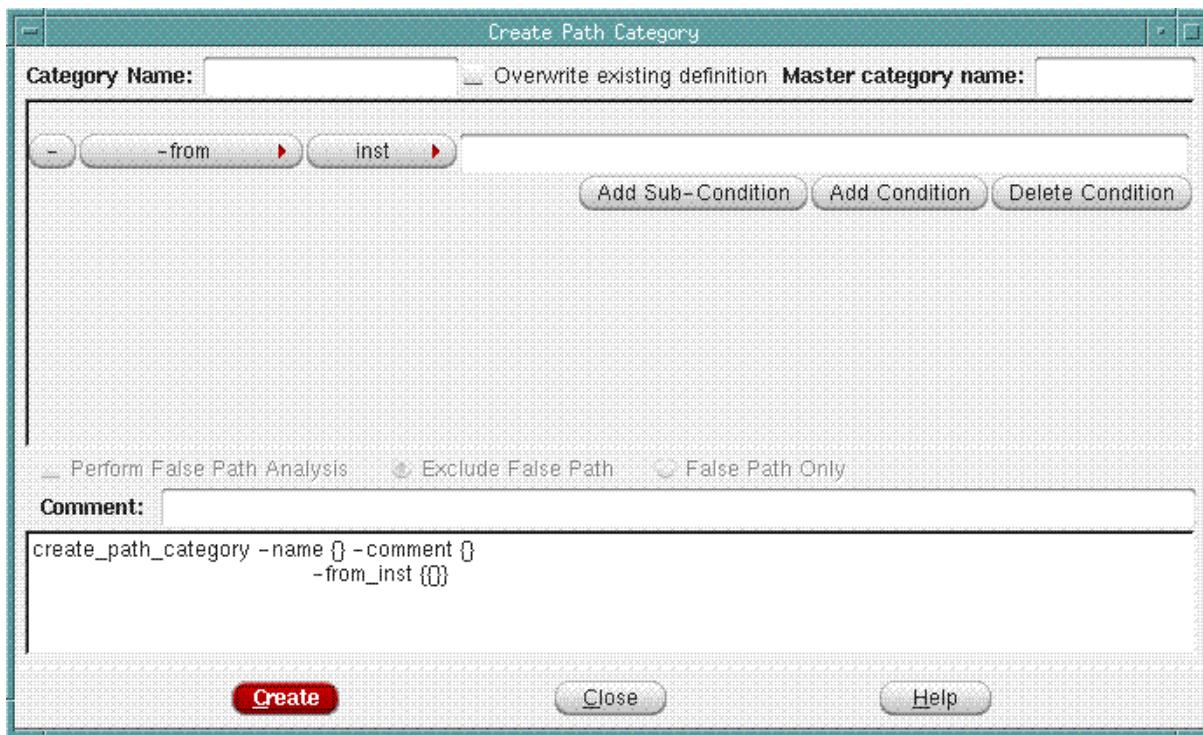
## Tempus User Guide

### Appendix

To create or load this report, choose the *Analysis* tab - *Analysis* drop-down menu - *DRV Analysis* option.

#### Creating New Categories

To define a new category, use the *Create Path Category* form. The Create Path Category form contains drop-down menus with conditions that you use to define a path category. The conditions are characteristics that a path must have to be added to the named category. You can define multiple conditions that a path must meet to be added to the category.



The category that you create is added in the *Path Category* field in the *Analysis* tab. All paths that meet the conditions set for this category are grouped under the category name. Paths are separated automatically according to MMMC views into different categories, for example:

```
CLOCK1<View_test_mode>
CLOCK2<View_mission_mode>
```

- Double-click on the category name in the *Path Category* field in the *Analysis* tab to display the list of paths in the *Path List* field.

**Note:** You can add a comment in the *Comment* field to record any notes that you would like to include with the category. The comment appears in the category report file.

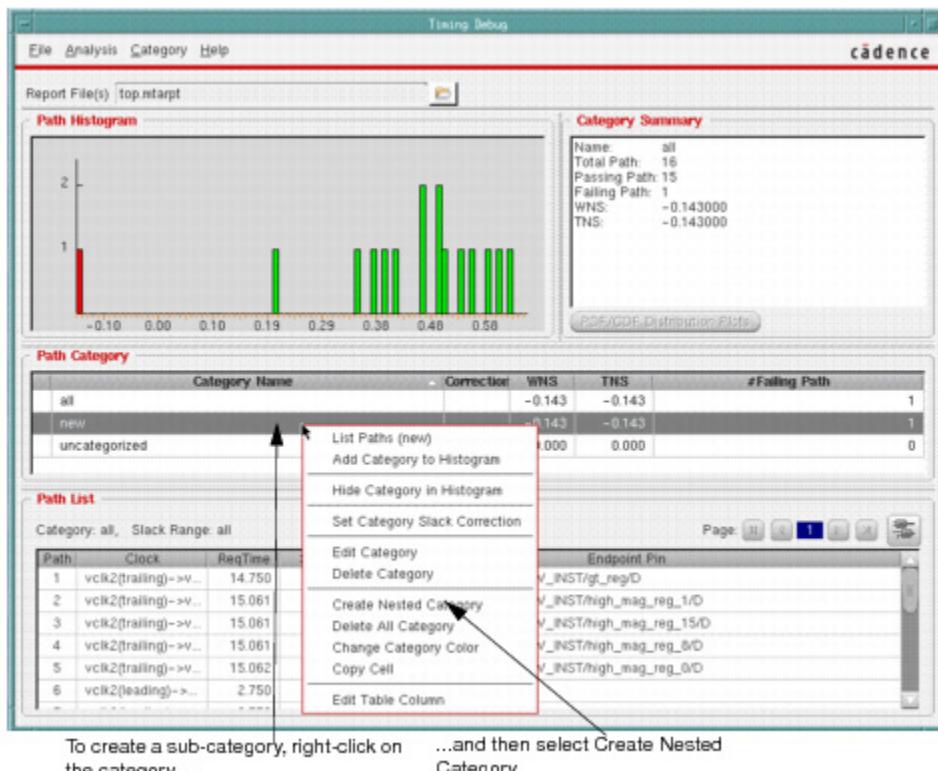
## Creating Sub-Categories

You can create sub-categories based on existing categories. While analyzing a sub-category, global timing debug will traverse the paths in the master category instead of all the paths in the current report.

- [Creating Sub-Categories through the GUI](#) on page 249
- [Creating Sub-Categories through Command Line](#) on page 250

### *Creating Sub-Categories through the GUI*

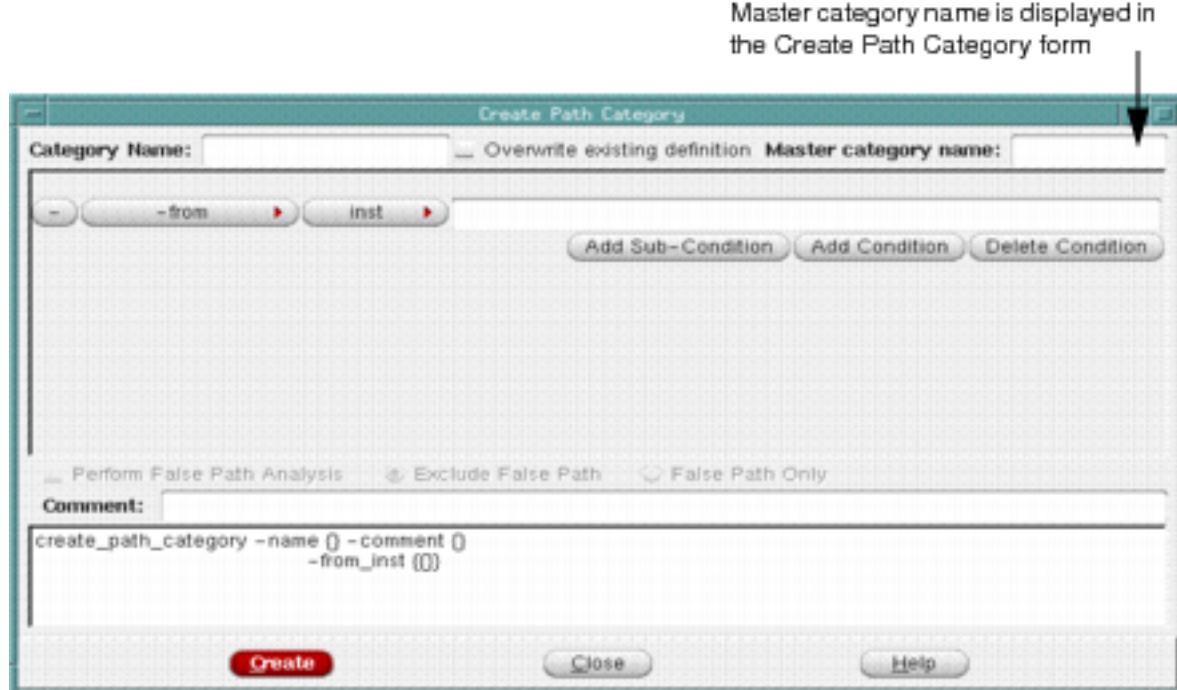
- In the GUI, you can create a sub category as follows:
- Right-click on a path category, and select *Nested Category*.



## Tempus User Guide

### Appendix

The *Create Path Category* form is displayed.



In the Master category name field, the name of the category you selected previously is displayed. Create one or more subcategories as explained in *Creating Path Categories*.

**Note:** You can create nested sub-categories, that is, you can further create sub-categories for a sub-category.

You can also use the Category - Create menu command to bring up the Create Path Category form. In the *Master category name* field, type the name of the category for which you want to create the sub-category, and then create one or more subcategories as explained in *Creating Path Categories*.

### ***Creating Sub-Categories through Command Line***

Use the `-master` parameter of the `create_path_category` command to create sub-categories. The category created will be a sub-category of the category name specified with the `-master` parameter.

The following other commands also support sub-categories; to run these commands only on the sub-categories of a particular master category, specify the master category name with the `-master` parameter.

- `analyze_paths_by_basic_path_group`

## Tempus User Guide

### Appendix

---

- analyze\_paths\_by\_bottleneck
- analyze\_paths\_by\_clock\_domain
- analyze\_paths\_by\_critical\_false\_path
- analyze\_paths\_by\_drv
- analyze\_paths\_by\_hierarchy
- analyze\_paths\_by\_view

**Note:** If the parent category of a sub-category is deleted, the sub-category cannot be edited or changed anymore. However, the sub-category is still displayed in case you want to refer to it.

#### ***Viewing Sub-Categories***

The subcategories for a master category are displayed in a hierarchically numbered list below the master category. As an illustration, consider the example shown here:

Path Category	
	Category Name
	all
	master_category1
(1)	nested_category_1a
(2)	nested_category_2
(1)	nested_category_1b
	uncategorized

In this example:

- master\_category1 is the master category
- nested\_category\_1a and nested\_category\_1b are the sub-categories of master\_category1.  
The prefix (1) is displayed with nested\_category\_1a and nested\_category\_1b.
- nested\_category\_2 is the sub-category of nested\_category\_1a.  
The prefix (2) is shown with nested\_category\_2.

#### Hiding path categories

To remove a path category from the histogram display, right-click on a path and select *Hide Category*. The category name in the category list is not hidden, but is marked with an "H" as hidden.

#### Reporting Path Categories

To generate a report containing information about path categories, use the following options:

- Use the `write_category_summary` command
- Use the *Write Category Report File GUI*

The text file contains the following information:

- Category name
- Total number of paths
- Number of passing paths
- Number of failing paths
- Worst negative slack
- Total negative slack
- TNS

## Tempus User Guide

### Appendix

Sample report:

Category name	Total path	Passing Path	Failing path	WNS	TNS
test_clock<view_test_100MHz_1.00V>	3869	768	3101	-5.699	-4802.857
Clock Domain Analysis					
@->test_clock<view_test_100MHz_1.00V>	484	55	429	-2.292	-378.432
Clock Domain Analysis					
my_clk-><view_mission_166MHz_1.08V>	1	2	11	1.931	-1.931
Clock Domain Analysis					
my_clk_2x<view_mission_166MHz_1.08V>	156	154	2	-0.013	-0.21
Clock Domain Analysis					
cat1875	77	13	64	-3.524	-100.893
Category of paths that cross i_1875 and start with test_clock - need to change the uncertainty value -advised --by Don					

### Using Categories to Analyze Timing Results

You can use the categories that you create to group the timing paths, and display them as histogram in the *Analysis* tab. The *Analysis* tab displays the category details in the *Path Category* field. You can perform the following tasks in the *Analysis* tab to analyze the timing results:

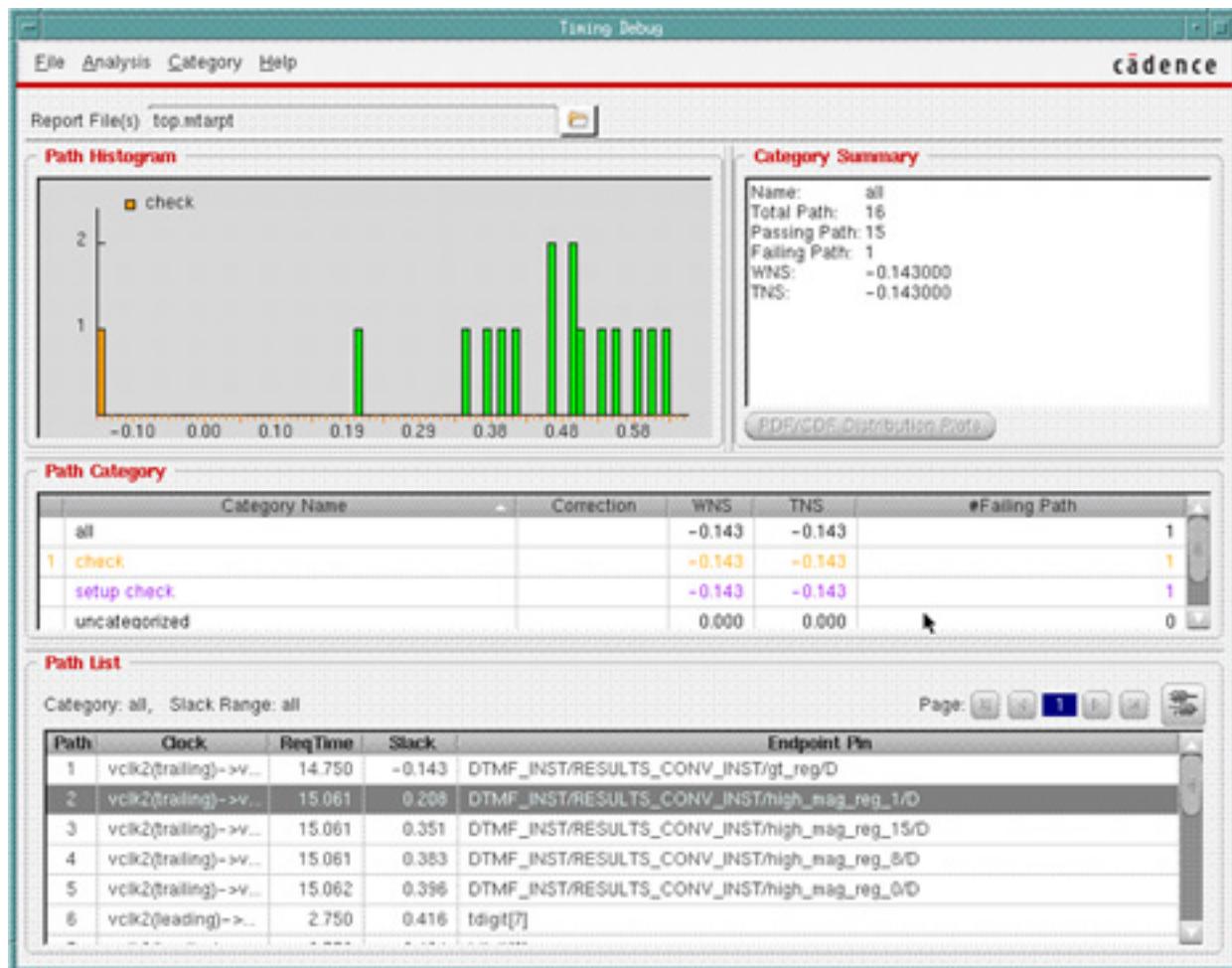
Double-click on any category to display the details of the paths grouped in that category in the *Path List* field.

Right-click on the category name and select *Add to Histogram* option. The paths related to the selected category are highlighted in a different color in the histogram. This gives you a visual representation of the number of paths that meet the conditions in that category and can possibly have the same timing problem.

## Tempus User Guide

### Appendix

For example, in the following figure the *SetupCheck* category was added to the histogram.



Analyzing the resulting the *Analysis* tab gives you information for fixing problems related to larger sets of timing paths. After identifying the problems, you can make the required changes such as modify floorplan, script or SDC files and run timing analysis again for further analysis.

### Analyzing MMMC Categories

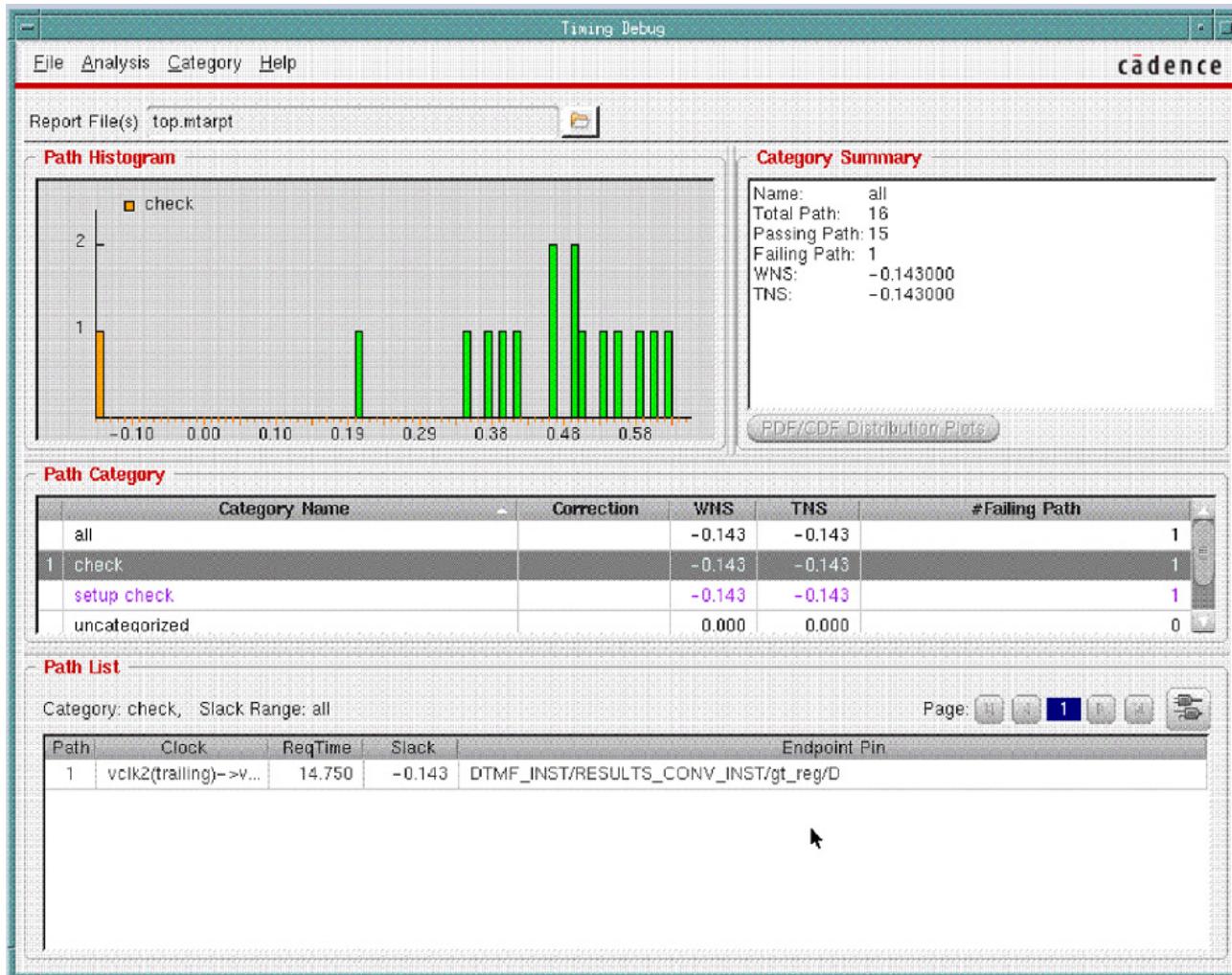
Paths are separated automatically according to MMMC views into different categories, for example, the following figure shows two categories based on MMMC views:

1. view\_mission\_140MHz\_1.0V

## Tempus User Guide

### Appendix

2. view\_mission\_166MHz\_1.8V



Do the following:

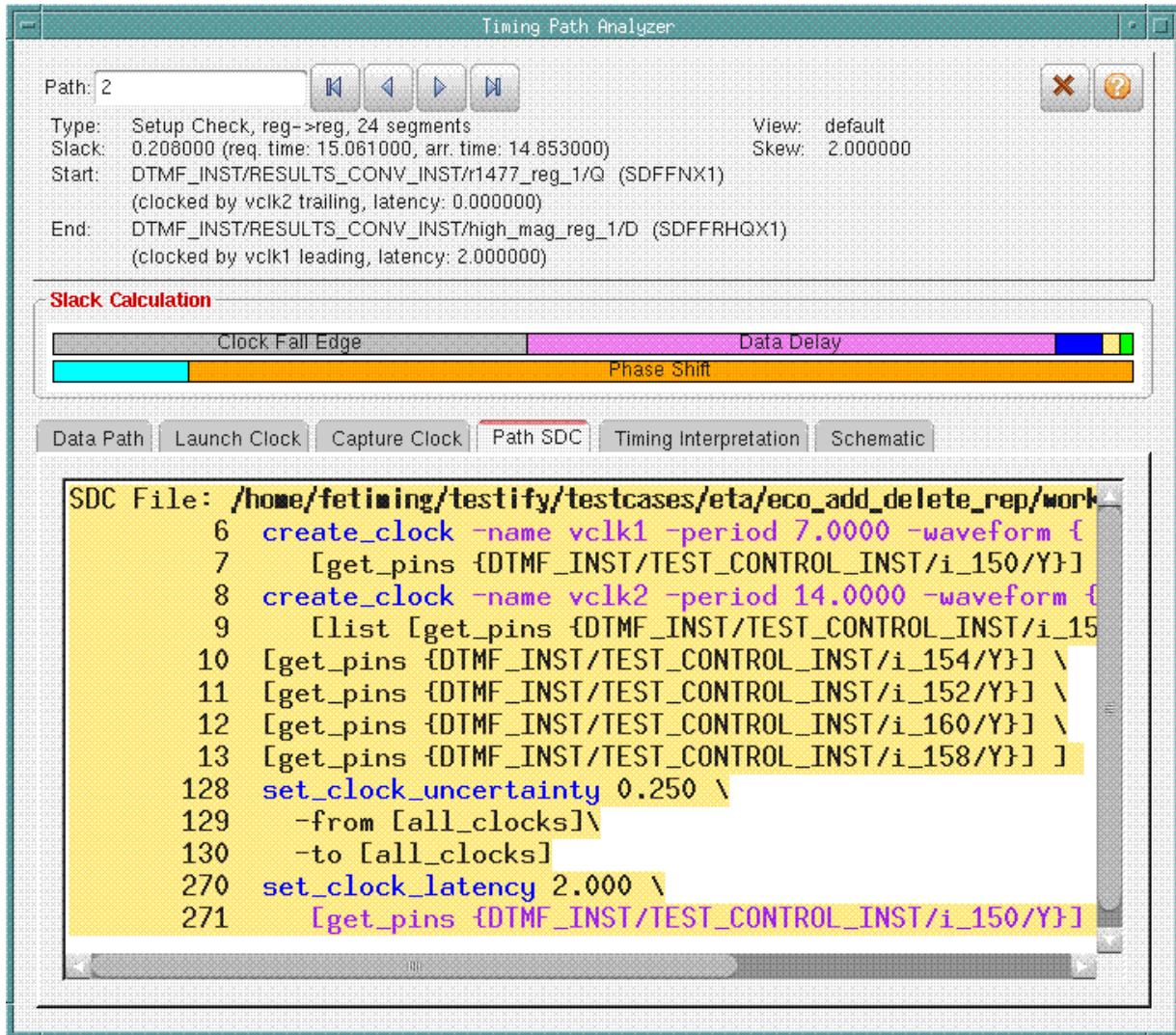
- Right-click on one of `view_mission_140MHz_1.0V` and choose *List Paths*.
- Right-click on one the paths and choose *Show Timing Path Analyzer*.

The Timing Path Analyzer is displayed.

## Tempus User Guide

### Appendix

- Click on the *Path SDC* tab to display the SDCs:



Note that the SDCs relative to mode mission\_140MHz that produced the path are highlighted.

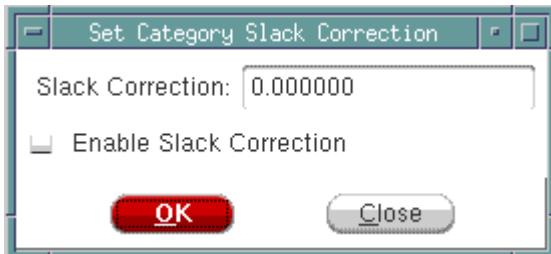
### Manual Slack Correction of Categories

Use the Set Category Slack Correction form to specify the estimated slack correction for the selected category of paths. A slack correction that you apply to a category modifies all the paths in that category. If a path belongs to several categories, all the correction from the categories are added. The worst negative slack and total negative slack values of a category can be affected by the correction applied to another category.

Once you enable the slack correction, the histogram is updated to reflect the slack correction. An asterisk (\*) is added next to the slack value of paths that belong to this category in the *Path List* field in the *Analysis* tab. Paths are reordered based on new specified slack. This allows you to filter out the paths that can be fixed and work on the remaining paths.

To access the Set Category Slack Correction form complete the following steps:

- Click on the *Analysis* Tab in the main Tempus window.
- Right click on the category name in the *Path Category* field.
- Choose the *Set Category Slack Correction* option.



To disable the set slack correction value:

- Right click on the category name in the *Path Category* field.
- Choose the *Deactive Category Slack correction* option.

## Editing Table Columns

You can customize the dimensions and contents of table columns to suit your needs.

- To begin customizing a table, click on the *Analysis* tab, then right-click on a path.

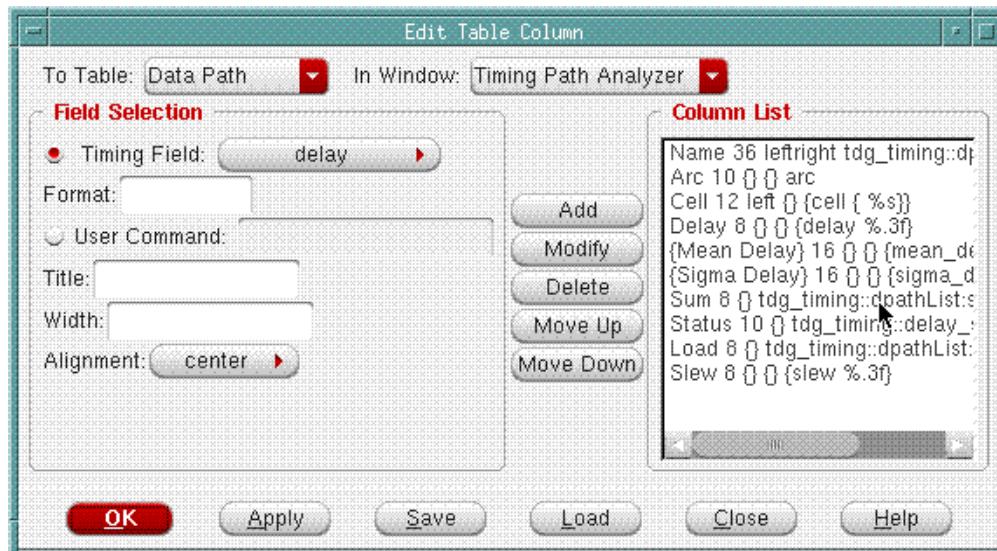
A drop-down menu is displayed.

- Select *Edit Table Column*.

## Tempus User Guide

### Appendix

The *Edit Table Column* form is displayed.



- Choose the timing window that contains the table to want to customize.
- Choose the table whose columns you want to customize. The selections change according to the timing window you choose.
- Choose a column item or specify a command.

For commands, specify the procedure you want to use to determine the information you want to include in the column. Source the file containing the procedure before you specify the procedure here.

For example:

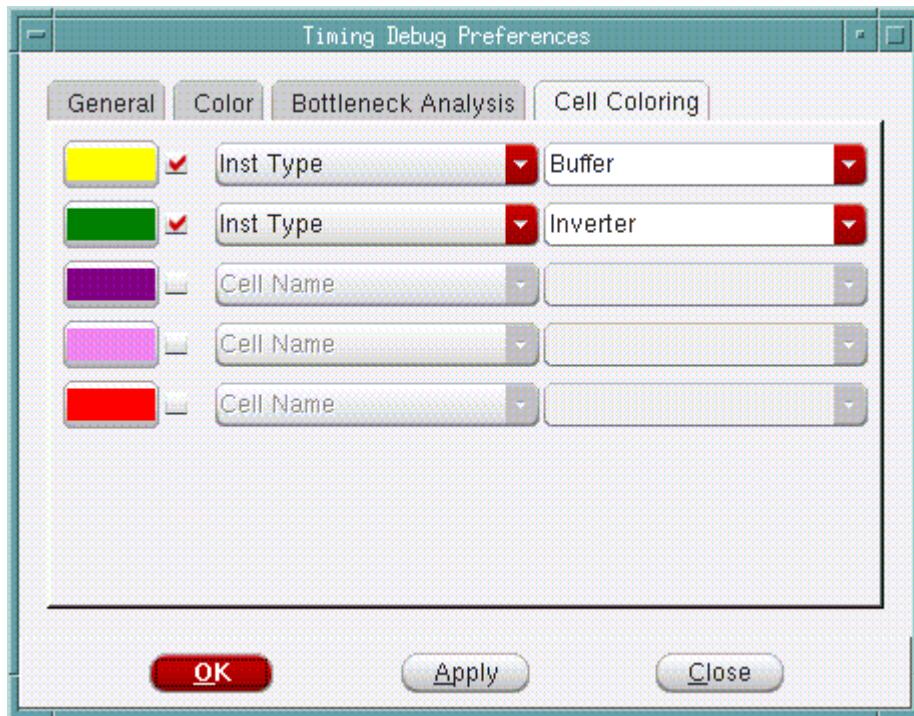
```
Combine fedge (from edge) and tedge (to edge) #information into a single field
proc my_get_edge {id var} {
 upvar #0 $var p
 ($ added before p)
 if {$p(type) == "inst"} {
 return "$p(fedge) -> $p(tedge)"
 } elseif {$p(type) == "port"} {
 return $p(fedge)
 } else {
 return ""
 }
}
```

- Build the column list.
  - *Add* adds a column to the column list.

- ❑ *Modify* let you modify characteristics. Click on a column in the column list. Edit the information, then click *Modify*.
- ❑ *Delete* removes a column from the column list.
- ❑ *Move Up* moves a column up in the column list. This effectively moves a column to the left in the table.
- ❑ *Move Down* moves a column down in the column list. This effectively moves a column to the right table.
- (Optional) Click *Load*. The opens the GTD (Global Timing Debug) Preferences form. Specify a file name.

### Cell Coloring

Use the *Cell Coloring* page of the Timing Debug Preferences form to choose colors for specific cells in the delay bar.



When you assign colors, this same colors will be restored when you start a new session.

In the *Cell Name Selection Elements* field for each color, you can choose whether you are providing one of the following:

## Tempus User Guide

### Appendix

- Cell name
- Instance
- Procedure that you have defined

The procedure is invoked with the full instance name as the argument. You must source the file containing the procedure before you use this feature.

For example:

```
Colors when the instance name contains "core/block1"
proc belongs_to_block1 {inst_name} {
 if [regexp {core/block1} $inst_name] {
 return 1
 } else {
 return 0
 }
}
```

## Viewing Schematics

The *Critical Path Schematic Viewer* displays the gate-level schematic view of the critical path. To display the Schematic Viewer, click on the schematics icon in the *Path List* field of the *Analysis* tab. You can display additional paths in the Schematic Viewer by using the middle mouse button to drag the path from path list to Schematic Viewer.

You can also display the Schematics by selecting the Schematics tab in the *Timing Path Analyzer* form. The form is displayed when you double-click on a critical path in the Path List in the *Analysis* tab.

On displaying the Schematic Viewer, you can see the power instance colored and the power domain information displayed in a popup message box as well as in terminal.



You can perform the following tasks in the Module Schematic Viewer:

- View the Gate-level design elements.
- Select an element in the schematic.
  - Click on an object in the schematic to select and highlight it. When you move the cursor to an object, the object type and name of the object appear in the information box.
- Scroll over an object to display the object type and name of the object in the *Object* field.
- Cross-probe between the Schematics window and *Path List* field.
  - Select a path and left-click on the Schematics button above the Path List Table. (This is the button at the far-right side, just above the table).
  - To show multiple paths, select another path, and drag and drop it to the Schematics window.
- Use the menu options provided in the Schematic Viewer. To access the menu options, you can either click on the menu bar or right-click on an object in the schematic. You can use the menu options to perform the following tasks:
  - Manipulate schematic views of fan-in and fan-out cones.
  - Trace connectivity between drivers, objects, and loads.
  - Move between different levels of instance views.

## Running Timing Debug with Interface Logic Models

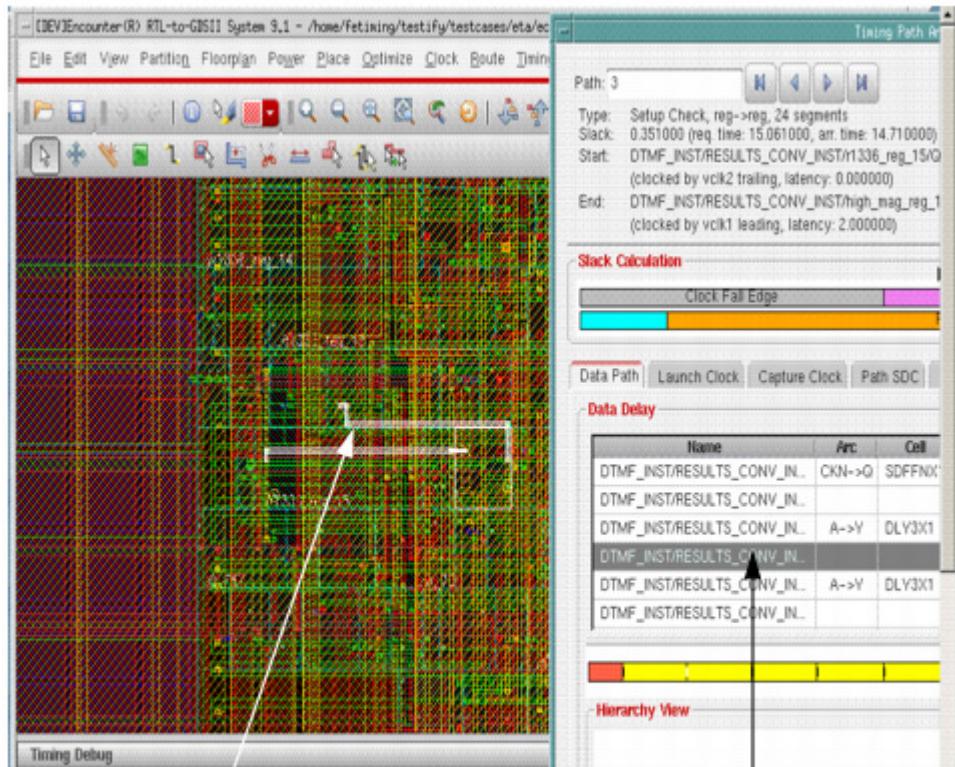
You can use the timing debug feature with designs containing Interface Logic Models (ILMs).

- The Timing Path Analyzer - Path SDC form displays ILM SDCs rather than the original SDCs.

## Tempus User Guide

### Appendix

- The software highlights the entire ILM module instead of the instances and nets inside the ILM. The instances and the nets inside the ILMs are greyed out in the Timing Path Analyzer - Path SDC form.



## Appendix - Multi-CPU Usage

- [Overview](#) on page 263
- [Running Distributed Processing](#) on page 264
- [Running Multi-Threading](#) on page 264
- [Memory and Run Time Control](#) on page 265
- [Checking the Distributed Computing Environment](#) on page 267
- [Setting and Changing the License Check-Out Order](#) on page 267
- [Limiting the Multi-CPU License Search to Specific Products](#) on page 267
- [Releasing Licenses Before the Session Ends](#) on page 267
- [Controlling the Level of Usage Information in the Log File](#) on page 268
- [Where to Find More Information on Multi-CPU Licensing](#) on page 268

### Overview

You can accelerate portions of the design flow by using multiple-CPU processing. The Tempus Timing Signoff Solution software has the following multiple-CPU modes:

- Multi-threading
  - In this mode, a job is divided into several threads, and multiple processors in a single machine process them concurrently.
- Distributed processing
  - In this mode, a job is processed by two or more networked computers running concurrently.

You can configure multiple-CPU processing by using the commands described in the *Multiple-CPU Processing Commands* chapter of the *Tempus Text Command Reference*.

The following Tempus features support multiple-CPU processing:

- Multi-mode multi-corner timing analysis
  - For information, see [Distributed Multi-Mode Multi-Corner Analysis](#)
- Timing analysis reporting commands specified by `run_threaded_commands`

For information, see `run_threaded_commands` in the *Tempus Text Command Reference*.

- Signal integrity analysis

For information, see `set_delay_cal_mode` in the *Signal Integrity Analysis Commands* chapter of the *Tempus Text Command Reference*.

## Running Distributed Processing

To run the software in distributed processing mode, the following two commands are required:

- `set_distribute_host`

Use this command to specify a configuration file for distributed processing or create the configuration for the remote shell, secure shell, or load-sharing facility queue to use for distributed processing. If you request more machines than are available, most applications wait until all requested machines are available.

To display the current setting for `set_distribute_host`, use the `get_distribute_host` command.

- `set_multi_cpu_usage`:

Use this command to specify the maximum number of computers to use for processing.

To display the current setting for `set_multi_cpu_usage`, use the `get_multi_cpu_usage` command.

## Running Multi-Threading

To run the software in multi-threading mode, the following command is required:

- `set_multi_cpu_usage`

Use this command to specify the number of threads to use. Upon completion, the log file generated by each thread is appended to the main log file.

**Note:** The `-localCpu` parameter limits the number of threads running concurrently. Although the software can create additional threaded jobs during run time, depending on the application in use, only the number of threads specified with this parameter are run at a given time.

If you ask for more threads than are available, the software issues a warning and runs with the maximum number of available threads.

For example, to run signal integrity analysis with four threads, specify the following commands:

```
set_multi_cpu_usage -localCpu 4
set_delay_cal_mode -siAware true
report_timing
```

## Memory and Run Time Control

Use the `report_resource` command to report memory/run time in multiple-CPU processing. This command allows you to determine how much memory is being used at any time and of what form (physical vs. virtual), and to determine real time and CPU time. You can use the `-verbose` parameter of the `report_resource` command to get detailed memory usage information.

When you run `report_resource -verbose`, the following detailed memory information is displayed:

Current (total cpu=0:00:12.9, real=0:05:48, peak res=275.8M, current mem=383.9M)				
Cpu(s) 2, load average: 4.63				
Mem: 16443800k total, 16378412k used, 65388k free, 105704k buffers				
Swap: 16777208k total, 17460k used, 16759748k free, 12528212k cached				
Memory Detailed Usage:				
	Data Resident Set(DRS)	Private Dirty(DRT)	Virtual Size(VIRT)	Resident Size(RES)
Total current:	383.9M	275.8M	854.1M	358.9M
peak:	383.9M	275.8M	854.1M	358.9M

- Cpu (s) is the number of available processors in the machine.
- Load average is the system load averages for the past 1 minute.
- Mem and Swap are the current memory information of the machine.  
The value of `MEM` in the LSF report corresponds to the value of `RES` in the `report_resource` report, and the value of `SWAP` in the LSF report corresponds to the value of `VIRT` in the `report_resource` report.
- Data Resident Set (DRS) is the amount of physical memory devoted to other than executable code. "current mem" shows this value (Total current DRS) .

## Tempus User Guide

### Appendix

---

- Private Dirty (DRT) is the memory which must be written to disk before the corresponding physical memory location can be used for some other virtual page. "peak res" shows this value (Total peak DRT). This is the minimum number that you must reserve to run the program.
- Virtual Size (VIRT) is the total amount of virtual memory used by the task. It includes the swapped and non-swapped memory.
- Resident Size (RES) is the non-swapped physical memory a task has used. The number of "Total Peak RES" is the recommended physical memory to reserve.

The `-verbose` parameter also works in conjunction with the `-peak` and `-start/-end` parameters of the `report_resource` command. When you run the local distributed host (`set_distribute_host -local`) command, the memory information will include the memory consumed by master and clients. Otherwise, the master and client details are not displayed.

The following command script specifies to display detailed memory information during the `report_timing` command:

```
report_resource -start report_timing
set_distribute_host -local
set_multi_cpu_usage -localCpu 8
report_timing -machine_readable -max_paths 100
report_resource -end report_timing -verbose
```

**Note:** For `-start/-end` parameters, use `-verbose` with the `-end` parameter only.

The following message is displayed:

Ending "report_timing" (total cpu=0:57:18, real=0:33:24, peak res=6493.1M, current mem=5305.0M)				
Memory Detailed Usage:				
	Data Resident Set(DRS)	Private Dirty(DRT)	Virtual Size(VIRT)	Resident Size(RES)
Total current:	5305.0M	4012.1M	5919.2M	4255.4M
peak:	10712.8M	6493.1M	15090.3M	7312.5M
Master current:	5305.0M	4012.1M	5919.2M	4255.4M
peak:	5565.5M	4055.1M	6064.5M	4298.4M
Task peak:	748.8M	368.7M	1219.4M	456.4M

Task `peak` reports peak value of each item from all clients, therefore, it is possible that eight values come from eight different clients.

## Checking the Distributed Computing Environment

To check if distributed processing can work in the software environment, use the `check_multi_cpu_usage` command. This command checks if the specified CPUs can be accessed.

## Setting and Changing the License Check-Out Order

To change the license check-out order, use the following command:

```
set_multi_cpu_usage -licenseList {tempusl cndc tempusxl eds1 edsxl}
```

For information on the default check-out order, see [Encounter Digital Implementation System Packaging and Licensing](#).

## Limiting the Multi-CPU License Search to Specific Products

Each base license allows a set of specific licenses to be used for Multi-CPU processing. This list can be obtained from the `get_multi_cpu_usage` command after invoking the software.

```
[DEV]tempus 24> get_multi_cpu_usage
Total CPU(s) Enabled: 8
Current License(s): 1 Tempus_Timing_Signoff_XL
keepLicense: true
licenseList: tpsmp
true
```

This license list can be customized from among the available choices by using the `set_multi_cpu_usage -licenseList` command.

## Releasing Licenses Before the Session Ends

By default, the software holds multi-CPU licenses for the duration of the current session. To release the multi-CPU licenses before the Tempus session ends, complete one of the following steps:

- Before running any multi-CPU applications, specify the following command to keep the acquired multiple CPU-licenses until the current session ends:

```
set_multi_cpu_usage -keepLicense true
```

To display the current setting for `set_multi_cpu_usage -keepLicense`, use the `get_multi_cpu_usage -keepLicense` command.

- At the point when you want to release the multi-CPU licenses (for example, when global placement finishes), specify the following command:

```
set_multi_cpu_usage -releaseLicense
```

## Controlling the Level of Usage Information in the Log File

Use the following command to set the level of usage information in the log file:

```
set_multi_cpu_usage -threadInfo {0 | 1 | 2}
```

By default, the software does not write starting and ending information for threads or timing details to the log file, but you can change this behavior by specifying 1 or 2 for the `-threadInfo` parameter.

- Specify 1 to write the final message to the log file.
- Specify 2 to write additional starting/ending information for each thread.

## Where to Find More Information on Multi-CPU Licensing

See *Encounter Digital Implementation System Packaging and Licensing* for more information on multi-CPU licenses.

## **Appendix - Base Delay, SI Delay, and SI Glitch Correlation with SPICE**

- [Overview](#) on page 270
- [Spice Deck Generation](#) on page 270
- [Running Path Simulation and Results](#) on page 275
- [Results of the Simulation](#) on page 275
- [Debugging Correlation Issues](#) on page 277

## Overview

To maintain signoff accuracy of the design, designers often use SPICE to correlate critical paths in timing analysis with path simulation. Tempus offers a built-in critical path simulation for base delay correlation with SPICE.

This chapter describes how to perform SPICE correlation with base delay timing in Tempus. The methodology is explained by providing an overview of the [create\\_spice\\_deck](#) command used for SPICE deck generation and then, by explaining the process to run path simulation.

## Spice Deck Generation

The [create\\_spice\\_deck](#) command is used to generate the SPICE deck, which means that the SPICE trace for a path is generated.

The output SPICE deck includes the following details:

- All nets in the path and their instance connections
- Standard cell gate information for the instances and their port connections
- Initial conditions and voltage sources
- Measure statements for slew and delay measurements
- RC parasitic network information

## Using [create\\_spice\\_deck](#) Command Parameters for SPICE Deck Generation

The following parameters of the [create\\_spice\\_deck](#) command are frequently used for SPICE deck generation in base timing analysis correlation.

### **-report\_timing {report\_timing\_options}**

#### **Description:**

Specifies the path(s) for which SPICE deck needs to be generated. SPICE deck will contain the SPICE trace of the path exactly as reported by the [report\\_timing](#) options.

#### **Options Used:**

## Tempus User Guide

### Appendix

---

- `-path_type full_clock`: Is used when launch or capture clock path is also needed in the SPICE deck.
- `-late/-early`: Is used to generate a SPICE deck for the setup or hold mode.

**Note:** Without using the `-report_timing` options, SPICE deck will be generated for worst path identified by the software.

#### **-run\_path\_simulation**

##### **Description:**

Enables the tool to perform path simulation using the Spectre™ simulator in Tempus.

**Note:** If this option is used, it is recommended to use the `report_timing -retime path_slew_propagation` option of `create_spice_deck` for more accurate correlation with SPICE because SPICE deck is path specific.

#### **-subckt\_file filename**

##### **Description:**

Specifies the name of the SPICE sub-circuit file for the library cells. This option is used to get the port directions.

**Note:** When `create_spice_deck` is specified with `-run_path_simulation` and this option is not specified, Tempus uses the SPICE sub-circuits from the cdB files loaded in the design.

Recommended option.

#### **-model\_file filename**

##### **Description:**

Specifies the name of the SPICE models file. This option must be used when you specify a SPICE sub-circuit file for simulation.

**Note:** When `create_spice_deck` is specified with `-run_path_simulation` and this option is not specified, Tempus uses the SPICE models from the cdB files loaded in the design.

Recommended option.

#### **-Spectre path\_to\_Spectre**

##### **Description:**

Specifies the location of the Spectre™ version, which is to be used for path simulation.

**-spice\_include "spice\_options"****Description:**

Specifies the user-defined SPICE statements that are written to the SPICE deck. For example, .measure statement or any SPICE option represents a user-defined SPICE statement.

**-outdir spice\_dir****Description:**

Specifies the name of the output directory that contains the generated SPICE decks. The default directory name is `tempus_pathsim`.

**Note:**

- If the `-run_path_simulation` option is not specified, SPICE deck is saved in the specified directory as `path_<index_of_path>_setup.sp`.
- If a single path is specified, the SPICE deck filename is saved as `path_1_setup.sp`.
- If more than one path is specified (using the `max_path/max_point/nworst` options of `report_timing`), SPICE decks are saved as `path_1_setup.sp`, `path_2_setup.sp`, and so on.
- If the `-run_path_simulation` option is specified in addition to the `path_1_setup.sp` file, some result files are also saved as mentioned in the *Running Path Simulation and Results* section.

**-side\_path\_level number****Description:**

Specifies the number of levels for the side path stages to be included in the SPICE deck. By default, the side path stages are not included in the SPICE deck. You can assign the maximum value of 4 for a side path stage. As you increase `-side_path_level`, the size of the SPICE deck and simulation run time will increase.

The default value of `-side_path_level` is 0, which means that only lumped cap model of side loads are provided.

**Note:** It is recommended to use value of 1 to include real gate load. It provides good accuracy with reasonable SPICE deck size and simulation run time.

**-power {list\_of\_power\_nets} | -ground {list\_of\_ground\_nets}**

Specifies the list of global names for power/ground nets.

If the `-power` and `-ground` options are not specified, Tempus retrieves this information from the power/ground supplies set via other ways in the design, such as the `set_dc_sources` command used from libraries/cdb and so on. If the information is not available, Tempus uses the default list of power node names (VDD and VCC) and ground node names (GND and VSS).

For more details on the parameters and options, see `create_spice_deck` in the *Tempus Text Command Reference Guide*.

#### **-xtalk {delay | vl\_glitch | vh\_glitch}**

The `-xtalk` parameter generates Spice deck for a single net (not a path).

For example, the following commands create a VL and VH Spice deck for the receiver pin `x1/A` and will place them in the `spice_rip/outlier` directory:

```
create_spice_deck -xtalk vl_glitch -receiver_pin x1/A -outdir spice_rip/outlier -
model_file models.sp -subckt_file base.sp
create_spice_deck -xtalk vh_glitch -receiver_pin x1/A -outdir spice_rip/outlier -
model_file models.sp -subckt_file base.sp
```

If you do not specify the `report_timing -point_to_point` parameter, the Spice deck can be generated (with victim sweep). However, path level Spice deck generation is not supported.

## Examples for SPICE Deck Generation Command

The following section shows examples of a generated SPICE deck.

**Example 1:** The following section in SPICE deck specifies global supplies and transient analysis window time.

```
*-----
*Command and Options
*-----
.GLOBAL vdd vbbnw
V0Xvdd vdd 0 1.0
V0Xvbbnw vbbnw 0 1.0
.GLOBAL vss vbbpw
V0Xvss vss 0 0
V0Xvbbpw vbbpw 0 0
.TEMP -40.0000
.TRAN 1p 10256p
*-----
```

**Example 2:** The following section shows an example of a net and its instance connections.

## Tempus User Guide

### Appendix

```
*-----
*Net Instances
*-----
*Net module1/inst_1/net
*-----
Xmodule1/inst_1/net module1/inst_1/q
+ module1/inst_2/a wc:module1/inst_1/q
*-----
```

**Example 3:** The following section shows an example of a gate instance and its port connections.

```
*-----
*Gate Instances
*-----
*Gate module1/inst_2
*-----
*Cell ports in order : a y vss vdd vbbpw vbbnw
*-----
Xmodule1/inst_2/a module1/inst_2/y
+ 0 module1/inst_2/vdd 0 module1/inst_2/vbbnw
+ inv1
*-----
```

**Example 4:** The following section shows the measurement statements for delay and slew.

```
*-----
*Measurement statements
*-----
.MEASURE TRAN Stage_0001 module1/inst_1/clk_DELAY_
module1/inst_1/q ←
+ TRIG V(module1/inst_1/clk) VAL=0.5 TD=167.9p RISE=1
+ TARG V(module1/inst_1/q) VAL=0.5 TD=167.9p RISE=1
.MEASURE TRAN Stage_0001 module1/inst_1/q_SLEW
+ TRIG V(module1/inst_1/q) VAL=0.30 TD=167.9p RISE=1 ←
+ TARG V(module1/inst_1/q) VAL=0.70 TD=167.9p RISE=1
.MEASURE TRAN Stage_0002 module1/inst_1/q_DELAY_
module1/inst_2/a ←
+ TRIG V(module1/inst_1/q) VAL=0.5 TD=167.9p RISE=1
+ TARG V(module1/inst_2/a) VAL=0.5 TD=167.9p RISE=1
.MEASURE TRAN Stage_0002 module1/inst_2/a_SLEW
+ TRIG V(module1/inst_2/a) VAL=0.30 TD=167.9p RISE=1 ←
+ TARG V(module1/inst_2/a) VAL=0.70 TD=167.9p RISE=1

```

clk to q delay of module1/inst\_1

Slew at module1/inst\_1/q pin

Delay and Input slews

**Note:** The **DELAY** and **SLEW** keywords in the **MEASURE** statements indicate that these are the measurement statements for delay and slew, respectively.

Also, stage numbers (Stage\_0001 and Stage\_0002) in the MEASURE statement indicate the respective stages; first and second stage, in the timing path, which makes it easier for correlation with the `report_timing` report.

## Running Path Simulation and Results

You can perform path simulation in the following two ways:

- Using the Spectre™ path simulator available in the Tempus installation, or `create_spice_deck -run_path_simulation`.
- Using the standalone (outside the Tempus environment) path simulation that can be run using Spectre™ or any simulator that understands the SPICE syntax.

### Using SPECTRE Path Simulator

You can use the `create_spice_deck -run_path_simulation` parameter to run the path simulation. In addition, you can specify the Spectre™ version using the `-spectre` parameter of the `create_spice_deck` command. If the path is not specified with the `-spectre` option, the software will check if you have the Spectre™ location defined in your path. If it is not defined in your path, it will use the Spectre™ version that is available under the Tempus installation. The generated log file will have the Spectre™ version used for path simulation.



### Results of the Simulation

A table of timing (see figure below) with the slew, delay, and arrival columns from `report_timing` and path simulation, which are used for correlation comparison, is displayed.

## Tempus User Guide

### Appendix

If `report_timing -path_type full_clock` is used, two separate tables for launch and capture paths are also shown.

Path 1 : MET Late External Delay Assertion												
Endpoint : out (v) checked with leading edge of 'Phi'												
Beginpoint : in (v) triggered by leading edge of 'Phi'												
+--												
Other End Arrival Time 0.000												
-	External Delay	0.345										
+	Phase Shift	18.000										
+	CPPR Adjustment	0.000										
=	Required Time	9.655										
+-----												
-	Arrival Time	0.502										
=	Slack Time	9.153										
+-----												
	Clock Rise Edge	0.000										
+	Input Delay	0.123										
=	Beginpoint Arrival Time	0.123										
+-----												
Table : timing on the data path												
net	instance	cell	pin	edge	arrival	delay	slew	load	sArrival	sDelay	sSlew	
in			in	v	0.123		0.050	0.009	0.123		0.050	
net_1	U1	BUFX4	U1/Y	v	0.241	0.118	0.055	0.009	0.251	0.128	0.031	
net_2	U2	BUFX4	U2/Y	v		0.362	0.121	0.055	0.009	0.367	0.116	0.031
out	U3	BUFX4	U3/Y	v	0.582	0.140	0.079	0.024	0.473	0.186	0.022	
			out	v	0.502	0.000	0.079	0.024	0.473	0.000	0.022	

## Using Standalone Path Simulation

If the `create_spice_deck` command is run without the `-run_path_simulation` option, it will save the SPICE deck of the path (`path_1_setup.sp`) in the specified directory (which is specified using the `-outdir` option). By default, it saves the SPICE deck in `tempus_pathsim` directory.

Using the standalone simulation generates the SPICE deck in the user-specified directory. The SPICE deck of the path (`path_1_setup.sp`) is saved at the specified path within the specified directory. Specify the directory through the `-outdir` parameter of the `create_spice_deck` command. By default, the generated SPICE deck is saved in the `tempus_pathsim` directory.

You can also run standalone path simulation using Spectre™ or any other simulator, which understands the SPICE syntax on the SPICE deck (`path_1_setup.sp`) saved by Tempus.

## Results of the Simulation

The `path_1_setup.measure` file is generated, which can be used to extract path simulation results.

The figure below is an example of the `path_1_setup.measure` file, which shows the slew and delay measurements of the two stages. This file can be easily post-processed to extract simulation results. For example, sum of all “delay” stages will give the path delay of the total path.

#### Sample `path_1_setup.measure` File

```
Exported variables from results directory: ./new.raw

date : 3:53:38 AM, Wed Aug 8, 2012
design : *
simulator : spectre

Measurement Name : transient1
Analysis Type : tran
stage_0001_u_cpu/ff1/clk_delay_u_cpu/ff1/q = 1.08834e-10
stage_0001_u_cpu/ff1/q_slew = 4.41217e-11
stage_0002_u_cpu/inst1/b_slew = 4.43192e-11
stage_0002_u_cpu/ff1/q_delay_u_cpu/inst1/b = 4.01084e-13
~
```

#### Debugging Correlation Issues

If the results between the SPICE path simulation and STA results do not correlate, you need to check the following:

- Ensure that the `.lib/.cdb` files are consistent with the SPICE models/subckt files being used for path simulation. Characterize libraries with the same SPICE models/subckt files which are used for running path simulation.
- Designers compare simulation results with the `report_timing` output. In such cases, since SPICE deck is always path-based, the comparison must always be done with the path-based analysis (PBA) report in Tempus. The PBA report can be generated using the `report_timing -retime path_slew_propagation` command. For slew/delay comparison with SPICE, make sure to use the `retime_slew` and `retime_delay` columns of the `report_timing` command instead of `slew` and `delay`, respectively.
- If there is any miscorrelation in any particular stage in delay or slew, then ensure that the slew/delay reported in Tempus analysis is not using extrapolation of library data. You can search for the IMPTS-403 message (Delay calculation was forced to extrapolate...) in `update_timing` as well as for the report of extrapolated arcs in a file with the nomenclature below written out by Tempus:

## Tempus User Guide

### Appendix

---

```
<top_cell>_dc_outbound_<view>_* .rpt
```

- SPICE deck generation and built-in path simulation do not account for any timing derates that may have been specified. Derate values can come from various sources, such as user-defined derates, AOCV derates, and so on.

For correlation comparison, ensure that you compare un-derated STA numbers from Tempus. To do this, you need to load the design in Tempus without any derates.

- Another source of miscorrelation could be the waveform used for simulation. Check that the library file has normalized driver waveforms (NDW) specified.
- Another potential source of differences between SPICE path simulation and STA results is the difference in analysis mode in both runs. This can happen when STA is done in the On-Chip Variation (OCV) mode (which is the default in Tempus) because the SPICE simulation is done only in one corner, either MAX or MIN. This means in SPICE both the launch and capture paths are using the same corner SPICE data for path simulation results.
- If path being correlated has latches, Tempus timing will have time borrowing during timing analysis while SPICE run will not have any time borrowing. In that case, turn off time borrowing in Tempus using `set timing_use_latch_time_borrow false` setting for correlation with SPICE.

## **Appendix - Supported CPF Commands**

- [CPF 1.0e Script Example](#) on page 280
- [CPF 1.0 Script Example](#) on page 287
- [CPF 1.1 Script Example](#) on page 297
- [Supported CPF 1.0 Commands](#) on page 305
- [Supported CPF 1.0e Commands](#) on page 314
- [Supported CPF 1.1 Commands](#) on page 325

## CPF 1.0e Script Example

The following section contains an example of the CPF 1.0e file using a sample design and library.

For list of supported CPF commands and options within Encounter product family, see "[Supported CPF 1.0e Commands](#)".

```
#-----
setting
#-----

set_cpf_version 1.0e
set_hierarchy_separator /

#-----
define library_set/cells
#-----

define_library_set -name wc_0v81 -libraries { \
 /LIBS/timing/library_wc_0v81.lib }
define_library_set -name bc_0v81 -libraries { \
 /LIBS/timing/library_bc_0v81.lib }
define_library_set -name wc_0v72 -libraries { \
 /LIBS/timing/library_wc_0v72.lib }
define_library_set -name bc_0v72 -libraries { \
 /LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
 VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
 NSLEEP -valid_location to
define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
 -enable ISO -valid_location to

define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
 0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
 -output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
 0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
 -output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
```

## Tempus User Guide

### Appendix

---

```
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to

define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD

define_state_retention_cell -cells { MSSRPG* } -cell_type \
master_slave -clock_pin CP -restore_check !CP -save_function !CP \
-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \
-ground VSS
define_state_retention_cell -cells { BLSRPG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS

#-----
macro models
#-----

#-----
top design
#-----
set_design top

create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
```

## Tempus User Guide

### Appendix

---

```
create_power_nets -nets VDD_sw -voltage { 0.72:0.81:0.09 } -internal \
 -peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
 -average_ir_drop_limit 0
create_power_nets -nets VDDL_sw -voltage 0.72 -internal -peak_ir_drop_limit 0 \
 -average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
 -average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
 -external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
 -average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
 -average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
 -average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
create power domains
#-----
create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifisp \
 IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
 -external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
 IOPADS_INST/Pibiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \
 IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } -boundary_ports { \
 ibias reset \
 refclk vcom vcop pllrst }
create_power_domain -name PDram_virtual
create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
 -shutoff_condition !INST/PM_INST/power_switch_enable \
 -secondary_domains { PDram_virtual }
create_power_domain -name PDTdsp -instances { INST/RAM_128x16_TEST_INST1 \
 INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
 !INST/PM_INST/power_switch_enable -secondary_domains { PDdefault }

#-----
```

## Tempus User Guide

### Appendix

---

```
set instances
#-----
set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
{ {RAM_DEFAULT PDtdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
-default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
-primary_ground_net VSS

end_macro_model

#-----
create power modes
#-----
create_power_mode -name PMdvfs1 -default -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDtdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDshutoff_io@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2 -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDtdsp@nom_0v72 PDram@nom_0v72 PDshutoff_io@nom_0v72 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDshutoff_io@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMscan -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDtdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1 -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDtdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }
```

## Tempus User Guide

### Appendix

---

```
create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 -domain_corners { \
 PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDtdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
 PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off -domain_corners { \
 PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off -domain_corners { \
 PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode \
 PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
 PDdefault@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode \
 PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
 PDdefault@PMdvfs1_wc }

create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 -domain_corners { \
 PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDtdsp@PMdvfs2_bc PDram@PMdvfs2_bc \
 PDshutoff_io@PMdvfs2_bc }

create_analysis_view -name AV_dvfs2_WC -mode PMdvfs2 -domain_corners { \
 PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDtdsp@PMdvfs2_wc PDram@PMdvfs2_wc \
 PDshutoff_io@PMdvfs2_wc }

create_analysis_view -name AV_PMdvfs2_off_BC -mode PMdvfs2_off -domain_corners \
 { PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDshutoff_io@PMdvfs2_bc }

create_analysis_view -name AV_PMdvfs2_off_WC -mode PMdvfs2_off -domain_corners \
 { PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDshutoff_io@PMdvfs2_wc }

create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode \
 PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
 PDdefault@PMdvfs2_bc }

create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode \
 PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
 PDdefault@PMdvfs2_wc }

create_analysis_view -name AV_scan_BC -mode PMscan -domain_corners { \
 PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDtdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
 PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_scan_WC -mode PMscan -domain_corners { \
 PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDtdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
 PDshutoff_io@PMdvfs1_wc }

#-----
create rules
#-----
```

```
create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
create_power_switch_rule -name PDtdsp_SW -domain PDtdsp -external_power_net \
```

## Tempus User Guide

### Appendix

---

VDD

```
create_isolation_rule -name ISORULE1 -isolation_condition { \
 !INST/PM_INST/isolation_enable } -from { PDtdsp } -to { PDdefault } \
 -isolation_target from -isolation_output high
create_isolation_rule -name ISORULE3 -isolation_condition { \
 !INST/PM_INST/isolation_enable } -from { PDram } -to { PDdefault } \
 -isolation_target from -isolation_output high
create_isolation_rule -name ISORULE4 -isolation_condition { \
 !INST/PM_INST/spi_ip_isolate } -from { PDshutoff_io } \
 -isolation_target from -isolation_output low

create_level_shifter_rule -name LSRULE_H2L3 -from { PDdefault } -to { PDram } \
 -exclude { INST/PM_INST/power_switch_enable }
create_level_shifter_rule -name LSRULE_H2L_PLL -from { PDpll }
create_level_shifter_rule -name LSRULE_L2H2 -from { PDram } -to { PDdefault }

create_state_retention_rule -name \
 INST/DSP_CORE_INST0/PDtdsp_retention_rule -instances { \
 INST/DSP_CORE_INST0 } -save_edge !INST/DSP_CORE_INST0/clk
create_state_retention_rule -name \
 INST/DSP_CORE_INST1/PDtdsp_retention_rule -instances { \
 INST/DSP_CORE_INST1 } -restore_edge \
 !INST/PM_INST/state_retention_restore -save_edge \
 INST/PM_INST/state_retention_save

#-----
update domains/modes
#-----
update_nominal_condition -name nom_0v81 -library_set wc_0v81
update_nominal_condition -name nom_0v72 -library_set wc_0v72

update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net \
 VSS
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
 -primary_ground_net VSS
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net \
 Avss
update_power_domain -name PDram_virtual -primary_power_net VDDL \
 -primary_ground_net VSS
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net \
```

## Tempus User Guide

### Appendix

---

```
VSS
update_power_domain -name PDtdsp -primary_power_net VDD_SW -primary_ground_net \
VSS

update_power_mode -name PMdvfs1 -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_off -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs2 -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_off -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMscan -sdc_files ../RELEASE/mmmc/scan.sdc

#-----
update rules
#-----
update_power_switch_rule -name PDram_SW -cells { HEADERHVT1 } -prefix \
CDN_SW_RAM -peak_ir_drop_limit 0 -average_ir_drop_limit 0
update_power_switch_rule -name PDtdsp_SW -cells { HEADERHVT2 } -prefix \
CDN_SW_TDSP -peak_ir_drop_limit 0 -average_ir_drop_limit 0

update_isolation_rules -names ISORULE1 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE3 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE4 -location to -prefix CPF_ISO_

update_level_shifter_rules -names LSRULE_H2L3 -location to -cells { LVLHLD* \
} -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_H2L_PLL -location to -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_L2H2 -location to -prefix CPF_LS_

update_state_retention_rules -names \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -cell_type master_slave
update_state_retention_rules -names \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -cell_type ballon_latch

#-----
end
#-----
end_design
```

## CPF 1.0 Script Example

The following section contains an example of the CPF 1.0 file using a sample design and library.

For list of supported CPF commands and options within, see "[Supported CPF 1.0 Commands](#)".

```
set_cpf_version 1.0

#####
#
Technology portion of the CPF:
Defining the special cells for low-power designs
#
#####

#####
High-to-Low level shifters
#####

define_level_shifter_cell -cells LVLH2L* \
 -input_voltage_range 0.8:1.0:0.1 \
 -output_voltage_range 0.8:1.0:0.1 \
 -direction down \
 -output_power_pin VDD \
 -ground VSS \
 -valid_location to

#####
Always-on High-to-low level shifters
#####

define_level_shifter_cell -cells AOLVLH2L* \
 -input_voltage_range 0.8:1.0:0.1 \
 -output_voltage_range 0.8:1.0:0.1 \
 -direction down \
 -output_power_pin TVDD \
 -ground VSS \
 -valid_location to
```

## Tempus User Guide

### Appendix

---

```
#####
Low-to-High Level Shifters
#####
define_level_shifter_cell -cells LVLL2H* \
 -input_voltage_range 0.8:1.0:0.1 \
 -output_voltage_range 0.8:1.0:0.1 \
 -input_power_pin VDDI \
 -output_power_pin VDD \
 -direction up \
 -ground VSS \
 -valid_location to

#####
Low-to-High level shifting plus isolation combo cells
#####
define_level_shifter_cell -cells LVLCIL2H* \
 -input_voltage_range 0.8:1.0:0.1 \
 -output_voltage_range 0.8:1.0:0.1 \
 -output_voltage_input_pin ISO \
 -input_power_pin VDDI \
 -output_power_pin VDD \
 -direction up \
 -ground VSS \
 -valid_location to

#####
Isolation cells
#####

define_isolation_cell -cells LVLCIL2H* \
 -power VDD \
 -ground VSS \
 -enable ISO \
 -valid_location to

#####
Power switch cells: headers
#####

define_power_switch_cell -cells {HEADERHVT HEADERAOPHVT} \
```

## Tempus User Guide

### Appendix

---

```
-power_switchable VDD -power TVDD \
-stage_1_enable !ISOIN1 \
-stage_1_output ISOOUT1 \
-stage_2_enable !ISOIN2 \
-stage_2_output ISOOUT2 \
-type header

#####
SRPG cells
#####

define_state_retention_cell -cells { SRPG2Y } \
-clock_pin CLK \
-power TVDD \
-power_switchable VDD \
-ground VSS \
-save_function "SAVE" \
-restore_function "!NRESTORE"

#####
Always-on cells: buffers and level shifters
#####

define_always_on_cell -cells {AOBUFF2Y AOLVLH2L*} \
-power_switchable VDD -power TVDD -ground VSS
#####
#
Design part of the CPF
#
#####

set_design top

set_hierarchy_separator "/"

set constraintDir ../CONSTRAINTS

set libdir ../LIBS

#####
create the power and ground nets in this design
#####
```

## Tempus User Guide

### Appendix

---

```
#####
VDD will connect the power follow-pin of the instances in the always-on
#power domain
VDD_core_SW will connect the power follow-pin of the instances in the
#switchable power domain and is the power net that can be shut-off
VDD_core_AO is the always-on power net for the switchable power domain

create_power_nets -nets VDD -voltage {0.8:1.0:0.1}
create_power_nets -nets VDD_core_AO -voltage 0.8
create_power_nets -nets VDD_core_SW -internal -voltage 0.8
create_power_nets -nets AVDD -voltage 1.0

create_ground_nets -nets VSS
create_ground_nets -nets AVSS

#####
Creating three power domains:
AO is the default always-on power domain
CORE is the switchable power domain
PLL is another always-on power domain
Also specifying the power net-pin connection in each power domain
#####

#####
For power domain "AO"
#####

create_power_domain -name AO -default
update_power_domain -name AO -internal_power_net VDD

create_global_connection -domain AO -net VDD -pins VDD
create_global_connection -domain AO -net VSS -pins VSS
create_global_connection -domain AO -net VDD_core_SW -pins VDDI

#####
For power domain "CORE"
#####

create_power_domain -name core -instances CORE_INST \
-shutoff_condition {PWR_CONTROL/power_switch_enable}
```

## Tempus User Guide

### Appendix

---

```
update_power_domain -name core -internal_power_net VDD_core_SW

create_global_connection -domain CORE -net VSS -pins VSS
create_global_connection -domain CORE -net VDD_core_AO -pins TVDD
create_global_connection -domain CORE -net VDD_core_SW -pins VDD

#####
For power domain "PLL"
#####

PLL contains a single PLL macro and five top-level boundary ports which
#connect to the PLL macro directly

create_power_domain -name PLL -instances PLLCLK_INST -boundary_ports \
{refclk vcom vcop ibias pllrst}
update_power_domain -name PLL -internal_power_net AVDD

create_global_connection -domain PLL -net AVDD -pins avdd!
create_global_connection -domain PLL -net AVSS -pins agnd!
create_global_connection -domain PLL -net VDD -pins VDDI
create_global_connection -domain PLL -net AVDD -pins VDD
create_global_connection -domain PLL -net AVSS -pins VSS

#####
#####

set lib_1p1_wc "$libdir/technology45_std_1p1.lib"
set lib_1p3_bc "$libdir/technology45_std_1p3.lib"
set lib_1p0_bc "$libdir/technology45_std_1p0.lib"
set lib_0p8_wc "$libdir/technology45_std_0p8.lib"

set lib_ao_wc_extra "\$libdir/technology45_lvll2h_1p1.lib \
"
set lib_ao_bc_extra "\$libdir/technology45_lvll2h_1p3.lib \
"
set lib_core_wc_extra "\$libdir/technology45_lvh21_0p8.lib \
\$libdir/technology45_headers_0p8.lib \
```

## Tempus User Guide

### Appendix

---

```
$libdir/technology45_sprg_ao_0p8.lib \
"
set lib_core_bc_extra "\$libdir/technology45_lvh2l_1p0.lib \
\$libdir/technology45_headers_1p0.lib \
\$libdir/technology45_sprg_ao_1p0.lib \
"
"
set lib_pll_wc "\$libdir/pll_slow.lib \
\$libdir/ram_256x16_slow.lib \
\$libdir/rom_512x16_slow.lib \
"
"
set lib_pll_bc "\$libdir/pll_fast.lib \
\$libdir/ram_256x16_fast.lib \
\$libdir/rom_512x16_fast.lib \
"
#####
Define library sets
#####
define_library_set -name ao_wc_0p8 -libraries "\$lib_0p8_wc \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p0 -libraries "\$lib_1p0_bc \$lib_ao_bc_extra"

define_library_set -name ao_wc_1p1 -libraries "\$lib_1p1_wc_base \$lib_ao_wc_extra"
define_library_set -name ao_bc_1p3 -libraries "\$lib_1p3_bc_base \$lib_ao_bc_extra"

define_library_set -name core_wc_0p8 -libraries "\$lib_0p8_wc \$lib_core_wc_extra"
define_library_set -name core_bc_1p0 -libraries "\$lib_1p0_bc \$lib_core_bc_extra"

define_library_set -name pll_wc_1p1 -libraries "\$lib_pll_wc"
define_library_set -name pll_bc_1p3 -libraries "\$lib_pll_bc"

#####
Create operating corners
#####

create_operating_corner -name BC_PVT_AO_L \
-process 1 -temperature 0 -voltage 1.0 \
-library_set ao_bc_1p0
```

## Tempus User Guide

### Appendix

---

```
create_operating_corner -name WC_PVT_AO_L \
 -process 1 -temperature 125 -voltage 0.8 \
 -library_set ao_wc_0p8

create_operating_corner -name BC_PVT_AO_H \
 -process 1 -temperature 0 -voltage 1.3 \
 -library_set ao_bc_1p3

create_operating_corner -name WC_PVT_AO_H \
 -process 1 -temperature 125 -voltage 1.1 \
 -library_set ao_wc_1p1

create_operating_corner -name BC_PVT_CORE \
 -process 1 -temperature 0 -voltage 1.0 \
 -library_set core_bc_1p0

create_operating_corner -name WC_PVT_CORE \
 -process 1 -temperature 125 -voltage 0.8 \
 -library_set tdsp_wc_0p8

create_operating_corner -name BC_PVT_PLL \
 -process 1 -temperature 0 -voltage 1.3 \
 -library_set core_bc_1p3

create_operating_corner -name WC_PVT_PLL \
 -process 1 -temperature 125 -voltage 1.1 \
 -library_set tdsp_wc_1p1

#####
Create and update nominal conditions
#####

create_nominal_condition -name high_ao -voltage 1.1
update_nominal_condition -name high_ao -library_set ao_wc_1p1

create_nominal_condition -name low_ao -voltage 0.8
update_nominal_condition -name low_ao -library_set ao_wc_0p8

create_nominal_condition -name low_core -voltage 0.8
```

## Tempus User Guide

### Appendix

---

```
update_nominal_condition -name low_core -library_set core_wc_0p8

create_nominal_condition -name high_pll -voltage 1.1
update_nominal_condition -name high_pll -library_set pll_wc_1p1

create_nominal_condition -name off -voltage 0

#####
Create and upDate four power modes
#####

create_power_mode -name PM_HL_FUNC \
 -domain_conditions {AO@high_ao CORE@low_core PLL@high_pll} \
 -default
update_power_mode -name PM_HL_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_HL_TEST \
 -domain_conditions {AO@high_ao CORE@low_core PLL@high_pll}
update_power_mode -name PM_HL_TEST -sdc_files ${constraintDir}/top_test.sdc

create_power_mode -name PM_HO_FUNC \
 -domain_conditions {AO@high_ao CORE@off PLL@high_pll}
update_power_mode -name PM_HO_FUNC -sdc_files ${constraintDir}/top_func.sdc

create_power_mode -name PM_LO_FUNC \
 -domain_conditions {AO@low_ao CORE@off PLL@high_pll}
update_power_mode -name PM_LO_FUNC -sdc_files ${constraintDir}/top_slow.sdc

#####
Creating ten analysis views
#####

create_analysis_view -name AV_HL_FUNC_MIN_RC1 -mode PM_HL_FUNC \
 -domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MIN_RC2 -mode PM_HL_FUNC \
 -domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}

create_analysis_view -name AV_HL_FUNC_MAX_RC1 -mode PM_HL_FUNC \
 -domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}
create_analysis_view -name AV_HL_FUNC_MAX_RC2 -mode PM_HL_FUNC \
 -domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}
```

## Tempus User Guide

### Appendix

---

```
create_analysis_view -name AV_HL_SCAN_MIN_RC1 -mode PM_HL_TEST \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HL_SCAN_MAX_RC1 -mode PM_HL_TEST \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_HO_FUNC_MIN_RC1 -mode PM_HO_FUNC \
-domain_corners {AO@BC_PVT_AO_H CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_HO_FUNC_MAX_RC1 -mode PM_HO_FUNC \
-domain_corners {AO@WC_PVT_AO_H CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

create_analysis_view -name AV_LO_FUNC_MIN_RC1 -mode PM_LO_FUNC \
-domain_corners {AO@BC_PVT_AO_L CORE@BC_PVT_CORE PLL@BC_PVT_PLL}
create_analysis_view -name AV_LO_FUNC_MAX_RC1 -mode PM_LO_FUNC \
-domain_corners {AO@WC_PVT_AO_L CORE@WC_PVT_CORE PLL@WC_PVT_PLL}

#####
Creating and updating the rules for the insertion
of power switch, level shifter, isolation cell
#####

#####
One power switch rule
#####

create_power_switch_rule -name PWRSW_CORE -domain CORE \
-external_power_net VDD_core_AO

update_power_switch_rule -name PWRSW_CORE \
-cells HEADERHVT \
-prefix CDN_SW_ \
-acknowledge_receiver SIWTCH_ENOUT
#####
One isolation rule using level-shifting and isolation combo cells
#####

create_isolation_rule -name ISORULE -from CORE \
-isolation_condition "!PWR_CONTROL/isolation_enable" \
-isolation_output high

update_isolation_rules -names ISORULE -location to -cells LVLCIL2H2Y
```

## Tempus User Guide

### Appendix

---

```
#####
Three level shifting rules
#####

For signals from AO to CORE

create_level_shifter_rule -name LSRULE_H2L -from AO -to CORE \
-exclude {PWR_CONTROL/power_switch_enable PWR_CONTROL \
/state_retention_enable PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L -cells LVLH2L2Y -location to

Only for the control signals from AO to CORE

create_level_shifter_rule -name LSRULE_H2L_AO -from AO -to CORE \
-pins {PWR_CONTROL/power_switch_enable PWR_CONTROL/state_retention_enable\
PWR_CONTROL/state_retention_restore}
update_level_shifter_rules -names LSRULE_H2L_AO -cells AOLVLH2L2Y -location to

For signals from PLL to AO

create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to AO
update_level_shifter_rules -names LSRULE_H2L_PLL -cells LVLH2L2Y -location to

#####

One SRPG rule
#####

create_state_retention_rule -name SRPG_CORE \
-domain CORE \
-restore_edge {!PWR_CONTROL/state_retention_restore} \
-save_edge {PWR_CONTROL/state_retention_enable}

update_state_retention_rules -names SRPG_CORE \
-cell SRPG2Y \
-library_set tdsp_wc_0v792

end_design
```

## CPF 1.1 Script Example

The following section contains an example of the CPF 1.1 file using a sample design and library.

For list of supported CPF commands and options within Encounter product family, see [Supported CPF 1.1 Commands](#).

```
#-----
setting
#-----

set_cpf_version 1.1
set_hierarchy_separator /

#-----
define library_set/cells
#-----

define_library_set -name wc_0v81 -libraries { \
 /LIBS/timing/library_wc_0v81.lib }
define_library_set -name bc_0v81 -libraries { \
 /LIBS/timing/library_bc_0v81.lib }
define_library_set -name wc_0v72 -libraries { \
 /LIBS/timing/library_wc_0v72.lib }
define_library_set -name bc_0v72 -libraries { \
 /LIBS/timing/library_bc_0v72.lib }

define_always_on_cell -cells {PTLVLHLD* AOBUFF*} -power_switchable \
VDD -power TVDD -ground VSS

define_isolation_cell -cells { LVLLH* } -power VDD -ground VSS -enable \
NSLEEP -valid_location to
define_isolation_cell -cells { ISOHID* ISOLOD* } -power VDD -ground VSS \
-enable ISO -valid_location to

define_level_shifter_cell -cells { LVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { PTLVLHLD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction down \
-output_power_pin TVDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHCD* } -input_voltage_range \
```

## Tempus User Guide

### Appendix

---

```
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 \
-output_voltage_input_pin NSLEEP -direction up -input_power_pin VDDL \
-output_power_pin VDD -ground VSS -valid_location to
define_level_shifter_cell -cells { LVLLHD* } -input_voltage_range \
0.72:0.81:0.09 -output_voltage_range 0.72:0.81:0.09 -direction up \
-input_power_pin VDDL -output_power_pin VDD -ground VSS -valid_location to

define_power_switch_cell -cells { HEADERHVT1 HEADERHVT2 } \
-stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 -stage_2_enable \
NSLEEPIN2 -stage_2_output NSLEEPOUT2 -type header -power_switchable VDD \
-power TVDD -stage_1_on_resistance 10 - stage_2_on_resistance 10

define_state_retention_cell -cells { MSSRPG* } -cell_type \
master_slave -clock_pin CP -restore_check !CP -save_function !CP \
-always_on_components { DFF_inst } -power_switchable VDD -power TVDD \
-ground VSS
define_state_retention_cell -cells { BLSRPG* } -cell_type ballon_latch \
-clock_pin CP -restore_function !NRESTORE -save_function SAVE \
-always_on_components { save_data } -power_switchable VDD -power TVDD \
-ground VSS

#-----
macro models
#-----

#-----
top design
#-----
set_design top

create_operating_corner -name PMdvfs2_bc -voltage 0.88 -process 1 -temperature \
0 -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc -voltage 0.99 -process 1 -temperature \
0 -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc -voltage 0.81 -process 1 -temperature \
125 -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc -voltage 0.72 -process 1 -temperature \
125 -library_set wc_0v72

create_power_nets -nets VDD -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
```

## Tempus User Guide

### Appendix

---

```
create_power_nets -nets VDD_EQ -voltage { 0.72:0.81:0.09 } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_SW -voltage { 0.72:0.81:0.09 } -internal \
-peak_ir_drop_limit 0 -average_ir_drop_limit 0
create_power_nets -nets VDDL -voltage 0.72 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDDL_SW -voltage 0.72 -internal -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets Avdd -voltage 0.81 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_power_nets -nets VDD_IO -voltage { 0.72:0.81:0.09 } \
-external_shutoff_condition { io_shutoff_ack } -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_ground_nets -nets Avss -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0
create_ground_nets -nets VSS -voltage 0.00 -peak_ir_drop_limit 0 \
-average_ir_drop_limit 0

create_nominal_condition -name nom_0v81 -voltage 0.81
create_nominal_condition -name nom_0v72 -voltage 0.72

#-----
create power domains
#-----
create_power_domain -name PDdefault -default
create_power_domain -name PDshutoff_io -instances { IOPADS_INST/Pspifsip \
IOPADS_INST/Pspidip } -boundary_ports { spi_fs spi_data } \
-external_controlled_shutoff -shutoff_condition io_shutoff_ack
create_power_domain -name PDpll -instances { INST/PLLCLK_INST \
IOPADS_INST/Pbiasip IOPADS_INST/Ppllrstip IOPADS_INST/Prefclkip \
IOPADS_INST/Presetip IOPADS_INST/Pvcomop IOPADS_INST/Pvcopop } -boundary_ports { \
ibias reset \
refclk vcom vcop pllrst }
create_power_domain -name PDram_virtual
create_power_domain -name PDram -instances { INST/RAM_128x16_TEST_INST } \
-shutoff_condition !INST/PM_INST/power_switch_enable \
-base_domains { PDram_virtual }
create_power_domain -name PDTdsp -instances { INST/RAM_128x16_TEST_INST1 \
INST/DSP_CORE_INST0 INST/DSP_CORE_INST1 } -shutoff_condition \
!INST/PM_INST/power_switch_enable -base_domains { PDdefault }
```

## Tempus User Guide

### Appendix

---

```
#-----
set instances
#-----
set_instance INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -domain_mapping \
{ {RAM_DEFAULT PDtdsp} }

set_macro_model ram_256x16A

create_power_domain -name RAM_DEFAULT -boundary_ports { A* D* CLK CEN WEN Q* } \
-default -external_controlled_shutoff

create_state_retention_rule -name RAM_ret -instances { mem* } -save_edge !CLK

update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
-primary_ground_net VSS

end_macro_model ram 256x16A

#-----
create power modes
#-----
create_power_mode -name PMdvfs1 -default -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDtdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDshutoff_io@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs1_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2 -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDtdsp@nom_0v72 PDram@nom_0v72 PDshutoff_io@nom_0v72 \
PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_off -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v72 PDshutoff_io@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMdvfs2_shutoffio_off -domain_conditions { \
PDpll@nom_0v81 PDdefault@nom_0v72 PDram_virtual@nom_0v72 }

create_power_mode -name PMscan -domain_conditions { PDpll@nom_0v81 \
PDdefault@nom_0v81 PDtdsp@nom_0v81 PDram@nom_0v72 PDshutoff_io@nom_0v81 \
PDram_virtual@nom_0v72 }

create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1 -domain_corners { \
```

## Tempus User Guide

### Appendix

---

```
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDshutoff_io@PMdvfs1_wc }

create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode \
PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
PDdefault@PMdvfs1_bc }

create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode \
PMdvfs1_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
PDdefault@PMdvfs1_wc }

create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDTdsp@PMdvfs2_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs2_bc }

create_analysis_view -name AV_dvfs2_WC -mode PMdvfs2 -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDTdsp@PMdvfs2_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs2_wc }

create_analysis_view -name AV_PMdvfs2_off_BC -mode PMdvfs2_off -domain_corners \
{ PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDshutoff_io@PMdvfs2_bc }

create_analysis_view -name AV_PMdvfs2_off_WC -mode PMdvfs2_off -domain_corners \
{ PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDshutoff_io@PMdvfs2_wc }

create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode \
PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_bc \
PDdefault@PMdvfs2_bc }

create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode \
PMdvfs2_shutoffio_off -domain_corners { PDpll@PMdvfs1_wc \
PDdefault@PMdvfs2_wc }

create_analysis_view -name AV_scan_BC -mode PMscan -domain_corners { \
PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDTdsp@PMdvfs1_bc PDram@PMdvfs2_bc \
PDshutoff_io@PMdvfs1_bc }

create_analysis_view -name AV_scan_WC -mode PMscan -domain_corners { \
PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDTdsp@PMdvfs1_wc PDram@PMdvfs2_wc \
PDshutoff_io@PMdvfs1_wc }

#-----
create rules
#-----
```

## Tempus User Guide

### Appendix

---

```
create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
create_power_switch_rule -name PDtdsp_SW -domain PDtdsp -external_power_net \
VDD

create_isolation_rule -name ISORULE1 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDtdsp } -to { PDdefault } \
-isolation_target from -isolation_output high
create_isolation_rule -name ISORULE3 -isolation_condition { \
!INST/PM_INST/isolation_enable } -from { PDram } -to { PDdefault } \
-isolation_target from -isolation_output high
create_isolation_rule -name ISORULE4 -isolation_condition { \
!INST/PM_INST/spi_ip_isolate } -from { PDshutoff_io } \
-isolation_target from -isolation_output low

create_level_shifter_rule -name LSRULE_H2L3 -from { PDdefault } -to { PDram } \
-exclude { INST/PM_INST/power_switch_enable }
create_level_shifter_rule -name LSRULE_H2L_PLL -from { PDpll }
create_level_shifter_rule -name LSRULE_L2H2 -from { PDram } -to { PDdefault }

create_state_retention_rule -name \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -instances { \
INST/DSP_CORE_INST0 } -save_edge !INST/DSP_CORE_INST0/clk
create_state_retention_rule -name \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -instances { \
INST/DSP_CORE_INST1 } -restore_edge \
!INST/PM_INST/state_retention_restore -save_edge \
INST/PM_INST/state_retention_save

#-----
update domains/modes
#-----

update_nominal_condition -name nom_0v81 -library_set wc_0v81
update_nominal_condition -name nom_0v72 -library_set wc_0v72

update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net \
VSS -equivalent_power_nets VDD_EQ
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
-primary_ground_net VSS
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net \
Avss
update_power_domain -name PDram_virtual -primary_power_net VDDL \
```

## Tempus User Guide

### Appendix

---

```
-primary_ground_net VSS
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net \
VSS
update_power_domain -name PDtdsp -primary_power_net VDD_sw -primary_ground_net \
VSS

update_power_mode -name PMdvfs1 -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_off -sdc_files ../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs1_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs1.sdc
update_power_mode -name PMdvfs2 -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_off -sdc_files ../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMdvfs2_shutoffio_off -sdc_files \
../RELEASE/mmmc/dvfs2.sdc
update_power_mode -name PMscan -sdc_files ../RELEASE/mmmc/scan.sdc

#-----
update rules
#-----
update_power_switch_rule -name PDram_SW -cells { HEADERHVT1 } -prefix \
CDN_SW_RAM -peak_ir_drop_limit 0 -average_ir_drop_limit 0
update_power_switch_rule -name PDtdsp_SW -cells { HEADERHVT2 } -prefix \
CDN_SW_TDSP -peak_ir_drop_limit 0 -average_ir_drop_limit 0

update_isolation_rules -names ISORULE1 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE3 -location to -prefix CPF_ISO_
update_isolation_rules -names ISORULE4 -location to -prefix CPF_ISO_

update_level_shifter_rules -names LSRULE_H2L3 -location to -cells { LVLHLD* \
} -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_H2L_PLL -location to -prefix CPF_LS_
update_level_shifter_rules -names LSRULE_L2H2 -location to -prefix CPF_LS_

update_state_retention_rules -names \
INST/DSP_CORE_INST0/PDtdsp_retention_rule -cell_type master_slave
update_state_retention_rules -names \
INST/DSP_CORE_INST1/PDtdsp_retention_rule -cell_type balloon_latch

#-----
end
```

## **Tempus User Guide**

### Appendix

---

```
#-----
end_design
```

## Tempus User Guide

### Appendix

---

## Supported CPF 1.0 Commands

**Note:** The following commands are supported unless otherwise noted.

Command Name	Option	Notes
		N/A = not available in this release
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
create_bias_net		
	-net	
	-driver	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_ground_nets		
	-nets	
	-voltage	N/A
	-internal	N/A
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A

## Tempus User Guide

### Appendix

---

	-average_ir_drop_limit	N/A
create_isolation_rule		
	-name	
	-isolation_condition	
	-pins	
	-from	
	-to	
	-isolation_target	N/A
	-isolation_output	
	-exclude	
create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		N/A
	-start_condition	
create_nominal_condition		
	-name	
	-voltage	
	-pmos_bias_voltage	N/A
	-nmos_bias_voltage	N/A
create_operating_corner		
	-name	
	-voltage	
	-process	
	-temperature	
	-library_set	

## Tempus User Guide

### Appendix

---

create_power_domain		
	-name	
	-default	
	-instances	
	-boundary_ports	
	-shutoff_condition	
	-default_restore_edge	
	-default_save_edge	
	-power_up_states	N/A
create_power_mode		
	-name	
	-domain_conditions	
	-default	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-internal	
	-user_attributes	Accessible by getCPFUserAttr
	-peak_ir_drop_limit	N/A
	-average_ir_drop_limit	N/A
create_power_switch_rule		
	-name	
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	

## Tempus User Guide

### Appendix

---

	-domain	
	-instances	
	-restore_edge	
	-save_edge	
define_always_on_cell	-cells -power_switchable -power -ground_switchable -ground	
define_isolation_cell	-cells -library_set -always_on_pin -power_switchable -ground_switchable -power -ground -valid_location	
	-non_dedicated	N/A
	-enable	
define_level_shifter_cell	-cells -library_set -always_on_pin -input_voltage_range -output_voltage_range -direction	N/A
	-output_voltage_input_pin	N/A

## Tempus User Guide

### Appendix

---

	-input_power_pin	
	-output_power_pin	
	-ground	
	-valid_location	
define_open_source_input_pin		
	-cells	
	-pin	
	-library_set	
define_power_clamp_cell		N/A
	-cells	
	-data	
	-ground	
	library_set	
	-power	
define_power_switch_cell		
	-cells	
	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-power_switchable	
	-power	
	-ground	
	-ground_switchable	
	-on_resistance	Accessible by ::CPF::getCpfPsoCell

## Tempus User Guide

### Appendix

---

	-stage_1_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-stage_2_saturation_current	Accessible by ::CPF::getCpfPsoCell
	-leakage_current	Accessible by ::CPF::getCpfPsoCell
define_state_retention_cell		
	-cells	
	-library_set	
	-always_on_pin	N/A
	-clock_pin	N/A
	-restore_function	
	-restore_check	N/A
	-save_function	
	-save_check	N/A
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_library_set		
	-name	
	-libraries	
end_design		
identify_always_on_driver		N/A
identify_power_logic		
	-type	
	-instances	
set_array_naming_style		
set_cpf_version		
set_design		

## Tempus User Guide

### Appendix

---

	-ports	
	module	
set_hierarchy_separator		
set_instance		
	-port_mapping	
	-merge_default_domains	
	hier_instance	
set_power_target		N/A
set_power_unit		N/A
set_register_naming_style		
set_switching_activity		
	-all	
	-pins	
	-instances	
	-hierarchical	
	-probability	
	-toggle_rate	
	-clock_pins	N/A
	-toggle_percentage	N/A
	-mode	N/A
set_time_unit		
update_isolation_rules		
	-names	'
	-location	
	-cells	
	-library_set	
	-prefix	
	-combine_level_shifting	N/A
	-open_source_pins_only	

## Tempus User Guide

### Appendix

---

-update_level_shifter_rules		
	-names	
	-location	
	-cells	
	-library_set	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-name	
	-internal_power_net	
	-internal_ground_net	
	-min_power_up_time	N/A
	-max_power_up_time	N/A
	-pmos_bias_net	N/A
	-nmos_bias_net	N/A
	-user_attributes	Accessible by ::CPF::getCpfUserAttr
	-rail_mapping	N/A
	-library_set	
update_power_mode		
	-name	
	-activity_file	N/A
	-activity_file_weight	N/A
	-sdc_files	
	-peak_ir_drop_limit	N/A
	-average_ir_dropt_limit	N/A
	-leakage_power_limit	N/A

## Tempus User Guide

### Appendix

---

	-dynamic_power_limit	N/A
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-library_set	
	-prefix	
	-peak_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
	-average_ir_drop_limit	Accessible by ::CPF::getCpfUserAttr
update_state_retention_rules		
	-name	
	-cell_type	
	-cell	
	-library_set	

## Tempus User Guide

### Appendix

---

## Supported CPF 1.0e Commands

**Note:** The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
create_analysis_view		
	-name	
	-mode	
	-domain_corners	
	-group_views	
	-user_attributes	
create_assertion_control		
	-name	Unsupported
	-assertions	Unsupported
	-domains	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-net	
	-driver	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_global_connection		
	-net	
	-pins	
	-domain	
	-instances	
create_power_domain		

## Tempus User Guide

### Appendix

---

	-name	
	-instances	
	-boundary_ports	
	-default	
	-shutoff_condition	
	-external_controlled_shutoff	
	-default_isolation_condition	
	-default_restore_edge	
	-default_save_edge	
	-default_restore_level	Supported
	-default_save_level	Supported
	-power_up_states	Unsupported
	-active_state_condition	Unsupported
	-secondary_domains	
create_ground_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_isolation_rule		
	-name	
	-isolation_condition	
	-no_condition	Unsupported
	-pins	
	-from	
	-to	

## Tempus User Guide

### Appendix

---

	-exclude	
	-isolation_target	
	-isolation_output	
	-secondary_domain	
create_level_shifter_rule		
	-name	
	-pins	
	-from	
	-to	
	-exclude	
create_mode_transition		
	-name	
	-from	
	-to	
	-start_condition	
	-end_condition	
	-cycles	
	-clock_pin	
	-latency	
create_nominal_condition		
	-name	
	-voltage	
	-ground_voltage	
	-state	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
create_operating_corner		
	-name	
	-voltage	

## Tempus User Guide

### Appendix

---

	-ground_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-nmos_bias_voltage	Unsupported
	-process	
	-temperature	
	-library_set	
create_power_mode		
	-name	
	-default	
	-group_modes	
	-domain_conditions	
create_power_nets		
	-nets	
	-voltage	
	-external_shutoff_condition	
	-user_attributes	Supported: query getCPFUserAttributes
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
create_power_switch_rule		
	-name	
	-domain	
	-external_power_net	
	-external_ground_net	
create_state_retention_rule		
	-name	
	-domain	
	-instances	
	-exclude	

## Tempus User Guide

### Appendix

---

	-restore_edge	
	-save_edge	
	-restore_precondition	
	-save_precondition	
	-target_type	
	-secondary_domain	
define_always_on_cell		
	-cells	
	-library_set	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
define_isolation_cell		
	-cells	
	-library_set	
	-always_on_pins	
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
	-valid_location	
	-enable	
	-no_enable	Unsupported
	-non_dedicated	
define_level_shifter_cell		
	-cells	
	-library_set	
	-always_on_pins	

## Tempus User Guide

### Appendix

---

	-input_voltage_range	
	-output_voltage_range	
	-ground_output_voltage_range	Unsupported
	-ground_output_voltage_range	
	-direction	
	-input_power_pin	
	-output_power_pin	
	-input_ground_pin	Unsupported
	-output_ground_pin	Unsupported
	-ground	
	-power	Unsupported
	-enable	
	-valid_location	
define_library_set		
	-name	
	-libraries	
	-user_attributes	cdb: specify cdb libraries for the library set aocv: specify aocv libraries for the library set
define_power_clamp_cell		
	-location	Unsupported
	-within_hierarchy	Unsupported
	-cells	Unsupported
	-prefix	Unsupported
define_power_switch_cell		
	-cells	

## Tempus User Guide

### Appendix

---

	-library_set	
	-stage_1_enable	
	-stage_1_output	
	-stage_2_enable	
	-stage_2_output	
	-type	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-power_switchable	
	-power	
	-ground_switchable	
	-ground	
	-on_resistance	Supported (for use with addPowerSwitch)
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-leakage_current	Supported (for use with addPowerSwitch)
define_state_retention_cell		
	-cells	
	-library_set	
	-cell_type	
	-always_on_pins	
	-clock_pin	
	-restore_function	
	-save_function	
	-restore_check	
	-save_check	

## Tempus User Guide

### Appendix

---

	-always_on_components	Unsupported
	-power_switchable	
	-ground_switchable	
	-power	
	-ground	
end_macro_model		
end_power_mode_control_group		
get_parameter		
include		
identify_power_logic		
	-type	Only "isolation" is supported for the -type
	-instances	Supported
	-module	Supported
set_array_naming_style		
set_cpf_version		
set_hierarchy_separator		
set_design		
	-ports	
	-honor_boundary_port_domain	
	-parameters	
set_equivalent_control_pins		
	-master	Unsupported
	-pins	Unsupported
	-domain	Unsupported
	-rules	Unsupported
set_floating_ports		Unsupported

## Tempus User Guide

### Appendix

---

set_input_voltage_tolerance		
	-ports	Unsupported
	-bias	Unsupported
set_instance		
	-design	
	-model	
	-port_mapping	
	-domain_mapping	
	-parameter_mapping	
set_macro_model		
set_power_mode_control_group		
	-name	
	-domains	
	-groups	Unsupported
set_power_target		
	-leakage	Unsupported
	-dynamic	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-pins	Supported
	-instances	Supported
	-hierarchical	Supported
	-probability	Supported
	-toggle_rate	Supported
	-clock_pins	Unsupported
	-toggle_percentage	Unsupported

## Tempus User Guide

### Appendix

---

	-mode	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
	-open_source_pins_only	Supported
update_level_shifter_rules		
	-names	
	-location	
	-within_hierarchy	
	-cells	
	-prefix	
update_nominal_condition		
	-name	
	-library_set	
update_power_domain		
	-name	
	-primary_power_net	
	-primary_ground_net	
	-pmos_bias_net	Unsupported
	-nmos_bias_net	Unsupported
	-user_attributes	Supported: query getCPFUserAttributes
	-transition_slope	Unsupported
	-transition_latency	Unsupported

## Tempus User Guide

### Appendix

---

	-transition_cycles	Unsupported
update_power_mode		
	-name	
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-sdc_files	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
	-leakage_power_limit	Unsupported
	-dynamic_power_limit	Unsupported
update_power_switch_rule		
	-name	
	-enable_condition_1	
	-enable_condition_2	
	-acknowledge_receiver	
	-cells	
	-gate_bias_net	Unsupported
	-prefix	
	-peak_ir_drop_limit	
	-average_ir_drop_limit	
update_state_retention_rules		
	-names	
	-cell_type	
	-cells	
	-set_rest_control	Unsupported

## Tempus User Guide

### Appendix

---

## Supported CPF 1.1 Commands

**Note:** The following commands and options are supported unless otherwise noted.

Command Name	Option	Notes
asset_illegal_domain_configurations		
	-domain_conditions	
	-group_modes	
	-name	
create_analysis_view		
	-domain_corners	
	-group_views	
	-mode	
	-name	
	-user_attributes	
create_assertion_control		
	-assertions	Unsupported
	-domains	Unsupported
	-exclude	
	-name	Unsupported
	-shutoff_condition	Unsupported
	-type	Unsupported
create_bias_net		
	-average_ir_drop_limit	
	-driver	
	-net	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes

## Tempus User Guide

### Appendix

---

create_global_connection		
	-domain	
	-instances	
	-net	
	-pins	
create_ground_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_isolation_rule		
	-exclude	
	-from	
	-isolation_condition	
	-isolation_output	
	-isolation_target	
	-name	
	-no_condition	
	-pins	
	-secondary_domain	
	-to	
	-force	
create_level_shifter_rule		
	-exclude	
	-from	
	-name	

## Tempus User Guide

### Appendix

---

	-pins	
	-to	
	-force	
create_mode_transition		
	-clock_pin	
	-cycles	
	-end_condition	
	-from	
	-latency	
	-name	
	-to	
	-start_condition	
create_nominal_condition		
	-ground_voltage	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-state	Unsupported
	-voltage	
create_operating_corner		
	-ground_voltage	Unsupported
	-library_set	
	-name	
	-nmos_bias_voltage	Unsupported
	-pmos_bias_voltage	Unsupported
	-process	
	-temperature	
	-voltage	
create_power_domain		

## Tempus User Guide

### Appendix

---

	-active_state_condition	Unsupported
	-base_domains	
	-boundary_ports	
	-default	
	-default_isolation_condition	
	-default_restore_edge	
	-default_restore_level	Supported
	-default_save_edge	
	-default_save_level	Supported
	-external_controlled_shutoff	
	-instances	
	-name	
	-power_up_states	Unsupported
	-shutoff_condition	
create_power_mode		
	-default	
	-domain_conditions	
	-group_modes	
	-name	
create_power_nets		
	-average_ir_drop_limit	
	-external_shutoff_condition	
	-nets	
	-peak_ir_drop_limit	
	-user_attributes	Supported: query getCPFUserAttributes
	-voltage	
create_power_switch_rule		
	-domain	

## Tempus User Guide

### Appendix

---

	-external_ground_net	
	-external_power_net	
	-name	
create_state_retention_rule		
	-domain	
	-exclude	
	-instances	
	-name	
	-restore_edge	
	-restore_precondition	
	-save_edge	
	-save_precondition	
	-secondary_domain	
	-target_type	
define_always_on_cell		
	-cells	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
define_isolation_cell		
	-always_on_pins	
	-cells	
	-enable	
	-ground	
	-ground_switchable	
	-library_set	
	-no_enable	

## Tempus User Guide

### Appendix

---

	-non_dedicated	
	-power	
	-power_switchable	
	-valid_location	
define_level_shifter_cell		
	-always_on_pins	
	-cells	
	-direction	
	-enable	
	-ground	
	- ground_input_voltage_range	
	- ground_output_voltage_range	
	-input_ground_pin	
	-input_power_pin	
	-input_voltage_range	
	-library_set	
	-output_ground_pin	
	-output_power_pin	
	-output_voltage_range	
	-power	
	-valid_location	
define_library_set		
	-libraries	
	-name	

## Tempus User Guide

### Appendix

	-user_attributes	cdb: specify cdb libraries for the library set aocv: specify aocv libraries for the library set
define_power_clamp_cell		
	-cells	Unsupported
	-data	Unsupported
	-power pin	Unsupported
	-ground	Unsupported
	-library_set	
define_power_switch_cell		
	-cells	
	-enable_pin_bias	Unsupported
	-gate_bias_pin	Unsupported
	-ground	
	-ground_switchable	
	-leakage_current	Supported (for use with addPowerSwitch)
	-library_set	
	-power	
	-power_switchable	
	-stage_1_on_resistance	
	-stage_2_on_resistance	
	-stage_1_enable	
	-stage_1_output	
	-stage_1_saturation_current	Supported (for use with addPowerSwitch)
	-stage_2_enable	
	-stage_2_output	

## Tempus User Guide

### Appendix

---

	-stage_2_saturation_current	Supported (for use with addPowerSwitch)
	-type	
define_state_retention_cell		
	-always_on_components	Unsupported
	-always_on_pins	
	-cell_type	
	-cells	
	-clock_pin	
	-ground	
	-ground_switchable	
	-library_set	
	-power	
	-power_switchable	
	-restore_check	
	-restore_function	
	-save_check	
	-save_function	
end_design		
end_macro_model		
end_power_mode_control_group		
get_parameter		
include		
identify_power_logic		
	-instances	Supported
	-module	Supported
	-type	Only "isolation" is supported for the -type
set_array_naming_style		

## Tempus User Guide

### Appendix

---

set_cpf_version		
set_hierarchy_separator		
set_design		
	module	
	-ports	
	- honor_boundary_port_domain	
	-parameters	
set_equivalent_control_pins		
	-domain	Unsupported
	-master	Unsupported
	-pins	Unsupported
	-rules	
set_floating_ports		Unsupported
set_input_voltage_tolerance		
	-bias	Unsupported
	-ports	Unsupported
set_instance		
	-design	
	-domain_mapping	
	-model	
	-parameter_mapping	
	-port_mapping	
set_macro_model		
set_power_mode_control_group		
	-domains	
	-groups	Unsupported
	-name	

## Tempus User Guide

### Appendix

---

set_power_target		
	-dynamic	Unsupported
	-leakage	Unsupported
set_power_unit		
set_register_naming_style		
set_switching_activity		
	-all	Supported
	-clock_pins	Unsupported
	-hierarchical	Supported
	-instances	Supported
	-mode	Supported
	-pins	Supported
	-probability	Supported
	-toggle_percentage	Unsupported
	-toggle_rate	Supported
set_time_unit		
set_wire_feedthrough_ports		
update_isolation_rules		
	-cells	
	-location	
	-names	
	-open_source_pins_only	Supported
	-prefix	
	-within_hierarchy	
update_level_shifter_rules		
	-cells	
	-location	
	-names	
	-prefix	

## Tempus User Guide

### Appendix

---

	-within_hierarchy	
update_power_domain		
	-equivalent_ground_nets	
	-equivalent_power_nets	
	-name	
	-nmos_bias_net	Unsupported
	-pmos_bias_net	Unsupported
	-primary_ground_net	
	-primary_power_net	
	-transition_cycles	Unsupported
	-transition_latency	Unsupported
	-transition_slope	Unsupported
	-user_attributes {{disable_secondary_domain s {list of domains}}}	Supported: query getCPFUserAttributes
update_power_mode		
	-activity_file	Unsupported
	-activity_file_weight	Unsupported
	-average_ir_drop_limit	
	-dynamic_power_limit	Unsupported
	-leakage_power_limit	Unsupported
	-name	
	-peak_ir_drop_limit	
	-sdc_files	
update_power_switch_rule		
	-acknowledge_reciever_1	
	-acknowledge_reciever_2	
	-average_ir_drop_limit	
	-cells	

## Tempus User Guide

### Appendix

---

	-enable_condition_1	
	-enable_condition_2	
	-gate_bias_net	Unsupported
	-name	
	-peak_ir_drop_limit	
	-prefix	
update_state_retention_rules		
	-cell_type	
	-cells	
	-names	
	-set_rest_control	Unsupported

## Path Exception Priorities

The following are the path exception priorities if a path in the design matches more than one path exception:

- set\_false\_path
- set\_min\_delay, set\_max\_delay
- set\_multicycle\_path

If there is more than one exception of a given type, for example, the set\_multicycle\_path command, the path exception that is more specific has higher priority. A path exception is more specific if it specifies a longer path than the other. For example, the -from, -to options will have priority over the -from option.

If the path has the same number of reference points, then:

- -from option has priority over the -to option
- -to option has priority over the -through option
- -clock\_from option has priority over the -clock\_to option

**Note:** To check for ignored path exceptions, use the `report_path_exceptions -ignored` command.

The following list shows the priorities (highest to lowest) for path exceptions applied to the same path.

### Path Exception Priorities

1. set\_false\_path (Highest)
2. set\_max\_delay -from pin\_list
3. set\_max\_delay -to pin\_list
4. set\_max\_delay -through pin\_list
5. set\_max\_delay -clock\_from clkwave\_name <<<<<
6. set\_max\_delay -clock\_to clkwave\_name <<<<<
7. set\_max\_delay (The most constraining adjustment has the higher priority over less constraining adjustments.)
8. set\_multicycle\_path -from pin\_list
9. set\_multicycle\_path -to pin\_list

---

# Glossary

---

The below glossary defines the terms and concepts that you should understand to use Tempus effectively.

**A****arc**

Is the timing relation between two points. This could either be a cell arc or net arc.

**arrival time**

Is the time elapsed from the source of launch clock till the arrival of the data at the end point.

**asynchronous clocks**

Two clocks that do not have a fixed phase relationship between their time period.

**attribute**

A value related to an object.

**attacker**

A net that causes undesirable cross-coupling effects on a victim net.

**B****bindkey**

A keyboard key or mouse button linked (bound) to a Cadence command or function name.

**bounding box**

## Tempus User Guide

### Glossary

---

A rectangular area, identified by a lower-left point and an upper-right point. Typically used to identify the area of an object (a path or an instance) or a collection of objects (the selected set).

#### **branch**

A path between two nodes. Each branch has two associated quantities, a potential and a flow, with a reference direction for each.

## C

#### **capacitance**

The function of a design to store energy in the form of an electrostatic field.

#### **cell**

A component of a design; a collection of different aspects (representations) of component implementations, such as its schematic, layout, or symbol representations. A design object consisting of a set of views that can be stored and referenced independently. A cell can include other cells, forming a hierarchical design. A cell is an individual building block of a chip or system. In the database, a cell contains all the cellviews of that cell.

An inverter and a buffer are examples of a small cell. A decoder register, arithmetic logic unit (ALU), memories, complete chips, and printed circuit boards are examples of large cells.

#### **cell delay**

Represents the time difference of a signal when transmitted from an input to an output.

#### **cellview**

A specific representation (view) of a cell. A particular representation of a particular component, such as the physical layout of a flip-flop or the schematic symbol of a NAND gate. A database object containing all the information unique to a particular representation of a particular component. Cellviews are classified by their view type. Each cellview has a view name and can have one or more versions.

#### **clock**

Timing signals generated periodically by a device in the digital domain.

#### **clock gating**

One of the basic power-saving techniques where the clock is gated in a logic block only when a next state logic is required to be captured.

**clock latency**

Is the amount of time taken from the clock origin point to the clock pin of the flop (sink pin).

**clock path**

Is the timing path from the clock source to the sink(clock) pin of the flop.

**clock skew**

Is the difference between the clock arrival time at two different nodes.

**clone**

A layout structure that has been replicated from a specified source structure already implemented in the layout view. Each clone inherits its physical characteristics from the layout structure on which it is based, and the connectivity information from the associated schematic structure.

**constraint**

The restrictions set on the objects in a layout or schematic to meet the routing or placement requirements in a design.

**corner**

A corner is defined as a set of libraries (characterized for process, voltage, and temperature variations) and net delay, which together account for delay variations. Corners are not dependent on functional settings; they are meant to capture variations in the manufacturing process, along with expected variations in the voltage and temperature of the environment in which the chip will operate.

**coupled capacitance**

Is the parasitic capacitance between two wires.

**coupling**

Is the transfer of electrical energy from one circuit segment to another. In the context of Signal integrity, coupling refers to the capacitive coupling that is often unintended because of the capacitance between two wires that are adjacent to each other. Usually, one signal capacitively couple with another signal and causes noise.

**critical path**

Is defined as the path that has the worst amount of slack relative to the desired arrival time.

**crosstalk**

Unwanted interaction between signals on different electrical nets in proximity because of coupling capacitance between them.

**cross coupling**

Is the effect introduced on neighboring nets because of the coupling capacitance between them.

**current design**

Is the top-level design in a design hierarchy.

**D****data path**

Is the timing path between output of the launch flop to the input of capture flop.

**default value**

The value used by the software unless you specify otherwise. The default is frequently the initial state.

**delay**

Is the time that it takes a signal to propagate from a given point to another given point.

**delay calculation**

Is the term used in an integrated circuit design for the sum of the delay of logic gates and delay of wires attached to it; basically the delay of any path from a start point to end point. This also refers to the process by which cell delay or net delay is calculated.

**design**

The collection of all data in a cellview, including all nested data, starting at the top and expanding the hierarchy.

**design library**

A library that contains data for the current design. It is usually in the designer's own directory or in the design group's directory.

**design verification**

Process of verifying the functional and performance requirements of a design.

**device**

A design element that has both a symbol view and a layout view, with corresponding pins.

**distributed processing**

Runs a program on a distributed computing system of multiple independent machines that communicate through a network.

**distortion**

Is the changing of any of the attributes of a signal, such as amplitude, pulse width, rise time, and so on because of the medium, such as a connector or a cable through which the signal flows.

**DMMC**

Abbreviation for Distributed Multi-mode multi-corner timing analysis. It is a method for processing timing analysis views where the software automatically distributes the runs to multiple machines for analysis.

**driver**

A primitive device or behavioral construct that affects the digital value of a signal.

**DSTA**

Abbreviation for Distributed Static Timing Analysis. It is the Master/Client architecture that fully leverages multiple threads on both the Master/Client processes allowing maximum scalability, even with the growing size and complexity of the designs.

**dynamic analysis**

Dynamic analysis is a method of analyzing a circuit to obtain operating currents and voltages. It is a time-based analysis, where the circuit netlist is analyzed over a specified period, such as a clock cycle.

**E****endpoint**

Point in a design where the timing path ends. This is either a primary output port or the launching pin of a sequential cell (CK pin of a flop or EN pin of a latch).

**environment**

The hardware and software setup and conditions within which the system operates.

**extrapolation**

Using two or more points on a curve to determine an intermediate curve point that provides a more accurate value than the provided available points. For example, calculating delay for cases where the inputs are lying outside of look up table.

**F****false path**

Is a timing path not required to meet its timing constraints for the design to function properly.

**fanin**

Is the number of pins or ports in the cone of logic connected to the input of a logic gate.

**fanout**

Is the number of gate inputs that can connect to the gate output. For example, with a fanout of 4, one TTL gate can drive 4 others.

**floorplanning**

Placing ports, cells, memories, macros, and so on to either create a rough plan to estimate whether a design meets the timing and routability criteria, or create a starting point for design implementation.

**G**

**G**

**GBA**  
Abbreviation for Graph Based Analysis. It calculates the worst-case delays of all cells considering the worst-case slew for all the inputs of a gate when the worst slew propagation is ON.

**GUI**

Abbreviation for Graphical User Interface.

**glitch**

Unintended noise pulse induced on the neighboring nets due to capacitive (or inductive) coupling.

**H****hierarchical design**

A hierarchical design is a way of structuring the logic at more than one level of abstraction; that is, a symbol at one level of abstraction represents the internal contents of a cell at a higher level of hierarchy.

**hierarchy**

Nested design levels, such as instances within a cell. By default, you open the top level in the hierarchy when you open a cellview.

**hierarchy representation**

Specification of how the physical hierarchy is generated from your logical design, including which logical components are to be generated or ignored in the physical implementation and which physical views are used to implement logical components.

**I****IR drop**

IR drop is a reduction in the voltage that occurs on the VDD network of an integrated circuit because of the current flowing through the resistance of the VDD network itself ( $V = I \times R$ ).

**impedance**

The opposition of an electronic component to the flow of current.

**interconnect**

Refers to the physical wires and vias between the gates and cells in an integrated circuit.

**instance**

A database object that creates a level of hierarchy. An instance creates an occurrence of the referenced cellview.

**L****latch**

Latch is a level-sensitive register with three ports - input, output, and enable. When enable is active, input drives output, which means that the latch is open or transparent. When enable is inactive, the output is kept at the existing value.

**latency**

Is the time taken by a clock signal to be transmitted from the input source to output.

**layer**

A layer refers to a mask material. Various database shape objects are created referencing a layer and a purpose.

**library**

A library is a logical collection of design data implemented as a physical collection of the directories and files that can reside anywhere in the file system. A library can be shared by all users or controlled by a single person.

**M****man page**

## **Tempus User Guide**

### Glossary

---

A description of a command, variable, or message code displayed using the man command in Tempus. For example, entering man report\_timing at the Tempus prompt displays the man page showing the description of the command and its related parameters.

#### **miller capacitance**

Is the coupling capacitance between nets associated with the coupling between the gate and the source/drain of a transistor.

#### **mode**

A mode is defined by a set of clocks, supply voltages, timing constraints, and libraries. It can also have annotation data, such as SDF or parasitics files. Many chips have multiple modes such as functional modes, test mode, sleep mode, and so on.

#### **modeling**

Refers to the creation and simulation of electronic circuits on a machine.

#### **multithreading**

Ability of a processing unit to run multiple processes or threads concurrently.

#### **MMMC**

Abbreviation for Multi Mode Multi Corner. MMMC is the ability to configure the software to support multiple combinations of modes and corners, and to evaluate them concurrently.

#### **MSV Designs**

Abbreviation for Multi Supply Voltage Design where the design is using more than one voltage sources.

## **N**

#### **net**

A logical signal connection between a set of pins on different instances. After routing, a net consists of routed wires on the routing layers.

#### **netlist**

Is a collection of related lists of the terminals (pins) in a circuit along with their interconnections.

**O****optimization**

Ability to achieve the most efficient design of a product. There are various types of optimizations, which are performed by different tools in the design flows for chips, boards, and so on.

**P****path**

A path object is a shape-based object represented by a point array with a specified width, style, and layer-purpose pair.

**path search**

The list of directories the software searches for files, libraries, and commands.

**parasitic capacitance**

Parasitic capacitance is any capacitance causing wanted or unwanted effects between proximity conductors or in devices.

**PBA**

Abbreviation for Path Based Analysis. Here, a timing path that is found by graph-based analysis (GBA), is re-timed for increased accuracy and for removing pessimism due to slew merging effects, AOCV stage counts, and so on.

**peak current**

Peak current is the maximum value of a current waveform.

**phase timing**

Is the timing relationship between the edges of a clock and other clock signals or data signals.

**placement and routing**

Process of placing and routing the connections of circuit.

**pin**

A physical implementation of a terminal. You can place pins on any layer. Pins have a figure defining the physical implementation and a terminal defining the type, such as the input, output, and so on. Pins also have names, access directions and placement status.

**ports**

Endpoints that let the model and the application to interact during simulation.

**propagation delay**

Is the speed with which a signal travels on a conductor path.

**properties**

Properties are a name-value pair defining an attribute or characteristic. Properties can be created on any database object. A property can be edited and deleted. Certain properties are mandatory for certain applications. Properties are defined and managed by the application.

**pulse**

Pulse is a standard representation of a waveform that is typically used as a voltage source in circuit simulation. The waveform displays a recurring pattern.

**R****receiver**

A primitive device or behavioral construct that samples the digital value of a signal.

**required time**

Is the latest time at which a signal can arrive without making the clock cycle longer than desired.

**routing**

Physically connecting objects in a design according to the design rules set in the reference library or technology file.

**RCDB**

## **Tempus User Guide**

### Glossary

---

Is Binary RC Database. It is a random-access format that provides parasitics information for timing analysis and reduces memory footprint. The RC database has better runtime and memory performance.

#### **RSH**

Abbreviation for Remote Shell. It is the method of launching jobs on remote machines.

#### **S**

##### **scaling**

Scaling is the process of globally reducing or enlarging a design. Scaling is typically used to match a design to a specific process.

##### **session window**

The current working window. A session window can contain multiple tabs, each potentially running a different application. You can have more than one session window open at a time, each with a workspace of its own.

##### **skew**

Is the difference in the timing that arises when a clock transition occurs.

##### **slack**

Is associated with each connection and is the difference between the required time and the arrival time. A positive slack  $s$  at a node implies that the arrival time at that node may be increased by  $s$  without affecting the overall delay of the circuit. Whereas, a negative slack implies that a path is too slow, and the path must be sped up (or the reference signal delayed) if the whole circuit is to work at the desired speed.

##### **slew**

Is the time taken by a signal to change from rise to fall or from fall to rise . This is also known as the transition time.

##### **signal integrity**

Refers to a signal's freedom from noise and the unpredictable delay caused by increasing the coupling capacitance between neighboring lines.

##### **standalone**

## **Tempus User Guide**

### Glossary

---

A mode in which a machine can run only locally installed and licensed applications.

#### **startpoint**

Is a point in a design where the timing path begins. This is either a primary input port or the capturing pin of a sequential cell (D pin of a flop, a latch or synchronous reset/set pins of a reset/set flop).

#### **static analysis**

Method of analyzing a circuit without using time-based simulation.

#### **superthreading**

Is a type of multithreading that enables different threads to be executed by a single processor without truly executing them at the same time.

#### **SDF file**

Abbreviation for Standard Database Format files. These are used to store values and the data of fixed lengths in a defined way and can easily be transferred across databases and programs.

#### **SMSC**

Abbreviation for Single Mode Single Corner. This is the ability to configure the software to support single mode and single corner at one time.

#### **STA**

Abbreviation for Static Timing Analysis. It is a simulation method of computing the expected timing of a digital circuit without requiring a simulation of the full circuit.

#### **SSH**

Abbreviation for Secure Shell. It is the method of securely launching jobs on remote machines.

#### **SPEF**

SPEF is an acronym for Standard Parasitic Exchange Format, which is a form of SPF that allows true coupling capacitance to be carried between nodes and is passed to downstream analysis tools.

#### **synchronous**

## Tempus User Guide

### Glossary

---

When specific transitions occur at the same time between two clocks or signals, thereby maintaining a timing relationship with each other.

## T

### **task**

Work objective. For example, a hierarchical task comprising a complex objective, such as optimizing device sizing, or a simple task such as adding a new pin to a schematic.

### **time borrowing**

Is a technique that allows logic to automatically borrow time from the next cycle, thereby reducing the time available for the data to arrive for the following cycle or permitting the logic to use the slack from the previous cycle in the current cycle.

### **timing arc**

Is the path set when a signal travels from one input to another output.

### **tool**

Is a shorthand term for an EDA product in the electronic design automation industry.

### **top-down design**

An approach to hierarchical design that uses estimates and floorplanning to start at the top level of a design.

### **transistor**

Is a semiconductor device, which is used to amplify a signal, or to open and close a circuit.

### **transient analysis**

Transient analysis is a time-based analysis of a system.

### **tapeout**

Process of transferring the final chip layout design files to a mask-making vendor.

## V

## **Tempus User Guide**

### Glossary

---

#### **via**

A via is a connection point between two adjacent metal routing layers defined by a pad in each layer.

#### **victim**

A victim net receives undesirable cross-coupling effects from any neighboring attacker net.

#### **VDD**

VDD is the common name of the power node, which is the voltage supply. It is the same as VCC.

#### **VSS**

VSS is the common name of the ground node, which is a conducting body used as the electrical current return. It is the same as GND.

#### **W**

#### **wire**

Wire refers to the routed physical metal conducting an electrical signal in contrast to a net, which refers to the logical electrical signal that is being transferred.