

Liberty™ NCX

User Guide

Version F-2011.06, June 2011

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2011 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

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

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

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

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

Trademarks (™)

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

Service Marks (SM)

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

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Contents

What's New in This Release	xii
About This Guide	xii
Customer Support	xiv
1. Running Liberty NCX	
Overview of Liberty NCX	1-2
Invoking Liberty NCX	1-3
Online Help	1-4
Characterization Settings.	1-5
Performing Concurrent Characterization	1-21
Using The Advanced Distributed Process Controller.	1-22
Distributed Model Extraction	1-22
Real-Time Control of Distributed Job Configuration	1-23
Liberty NCX Usage Flows.	1-23
2. Characterization Flows	
Characterization Input Data	2-2
Required Input Data.	2-2
Characterization Template Files	2-3
Previewing Generated Template Files	2-4
Characterization Output Data	2-5
Liberty NCX Operation and Log File.	2-7
Program Initialization Section	2-10

AutoFunction Section	2-11
Cell Sensitization Section	2-11
Arc Netlist Creation Section	2-12
Model Extraction Section	2-13
Program Summary Section	2-13
Partial Sensitization Flow	2-14
Automatic Function Recognition	2-16
Flip-Flop and Latch Cell Recognition	2-18
Scan Flip-Flop Cell Recognition	2-19
Two-Stage Synchronizer Cell Recognition	2-20
Integrated Clock-Gating Cell Recognition	2-20
Acquisition Types	2-21
CCS Timing	2-22
CCS Noise	2-22
CCS Power	2-23
Compact CCS Models	2-24
NLDM and NLPM Models	2-24
Variation-Aware Models	2-25
Incremental Characterization	2-28
Template Creation for Incremental Characterization	2-31
Library Creation	2-31
Incremental Characterization of More Cells	2-32
3. Template Files	
Template File Overview	3-2
Characterization Attributes	3-3
Sensitization	3-49
Sensitization Vectors	3-51
Specifying Sensitization for Timing Arcs	3-52
Specifying Sensitization for Non-Timing Arcs	3-56
Sensitization Vector Tables	3-57
Timing Vector Types	3-60
Power Vectors	3-60
Validity Checks	3-61

Function Statement Validity Check	3-61
Flip-Flop Group Validity Check	3-61
Latch Group Validity Check	3-62
Truth Table Validity Check	3-63
State Table Validity Check	3-63
Sensitization Truth Tables	3-64
Arc Generation Control	3-65
ncx_create_arcs Attribute	3-66
Partial Sensitization	3-69
Delay Arcs	3-70
Propagating Power Arcs	3-71
Constraint Arcs	3-72
Internal Leakage Power Arcs	3-73
Internal Power Arcs	3-73
Capacitance Extraction	3-73
Bidirectional Cell Arcs	3-74
Half-Unate Arcs	3-76
Example	3-76
Characterization Index Values	3-79
Percentage-of-Maximum Index Values	3-81
Number and Spacing of Index Values	3-81
Adaptive Selection of Index Values	3-82
Three-Dimensional Lookup Tables	3-85
Automatic Creation of Three-Dimensional Lookup Tables	3-86
Maximum Capacitance Acquisition	3-87
Input NLDM Capacitance Index Values	3-88
Variation-Aware Index Values	3-89
Conditional Characterization	3-90
Specifying Required Conditions	3-91
Specifying Conditional Attributes	3-93
Toggled and Non-Toggled Output Constraint Arcs	3-95
Conditional Attributes for Nonpropagating Arcs	3-95
Pin-Based Minimum Pulse Width	3-96
Timing Margins	3-96
Margin Types	3-97
Simple Margin Values	3-98
Margin Expressions	3-99

Combining Margin Types	3-105
Adding Margin Data To a Library As a Postprocess	3-106
Complex Cell Features	3-109
Interdependent Setup and Hold.....	3-109
User Customization	3-112
Minimum Setup Time for Pessimistic Delay.....	3-113
Differential Inputs and Outputs	3-114
Differential Pin Delay Measurement Modes	3-115
Delayed Differential Measurements	3-117
Skewed Differential Input Signals.....	3-118
Nonrail Voltage Swing	3-118
Constant Logic States on Inputs	3-119
Pin-Specific Timing Thresholds.....	3-120
Initialization Cycle	3-121
Synchronizer Circuit.....	3-121
Complex Sequential Cells	3-122
Publishing Complex Sequential Cell Descriptions	3-124
Simultaneous Switching.....	3-124
Custom Harness	3-125
External Termination Harness	3-126
Termination and Measurement Harness.....	3-127
Driver, Termination, and Measurement Harness	3-128
Bidirectional I/O Harness	3-129
Power and Ground Pin Connections.....	3-131
Examples.....	3-132
Low-Power Modeling.....	3-134
Multithreshold-CMOS Characterization.....	3-134
Transient Leakage Characterization	3-136
Transient Leakage Subtraction Implementation.....	3-137
HSPICE Measurement Statements	3-139
Nonpropagation Power Adjustment	3-140
Constraint Methodologies	3-141
Simple Violation Definitions	3-144
Advanced Violation Definitions	3-144
Configuring Delay Degradation Violations	3-147
Configuring Glitch Violations	3-148

Defining Node Sets.....	3-149
Accuracy Settings for Constraint Acquisition	3-150
User-Defined Attributes	3-151
4. I/O Cell Characterization	
I/O Cell Overview	4-2
Examples of Special Cases	4-3
Default arc: worst (PG1/PG2), user (PG3).....	4-3
Default arc: first (PG1), worst (PG2), user (PG3)	4-4
State arc: all (PG1), unpublished-worst (PG2/PG3)	4-4
State arc: user (PG1), unpublished-first (PG2), unpublished-user (PG3)	4-5
State arc: user (PG1), unpublished-worst (PG2), unpublished-user (PG3).....	4-5
Voltage Group Support	4-6
Voltage Groups Overview	4-6
Input Voltage Group Elements	4-7
Output Voltage Group Elements.....	4-8
Voltage Groups in the Library Group	4-9
Library and Cell Template.....	4-9
Netlist Changes	4-10
Delay Calculation with Voltage Groups	4-10
Voltage Group Limitations	4-11
I/O Cell .nodeset Feature	4-11
IBIS Model Generation	4-14
IBIS Model Generation Flow	4-16
IBIS Operating Condition Options	4-16
Time Step and Range	4-17
5. Library Conversion Support	
Model Adaptation System.....	5-2
Library Formatting Overview	5-2
CCS Model Compaction and Expansion	5-3
Variation-Aware Library Merging	5-3
CCS Noise Merging	5-5
CCS to ECSM Conversion.....	5-5
CCS Noise to ECSM Noise Conversion	5-7

CCS to NLDM Conversion	5-8
VA-CCS to S-ECSM Conversion	5-10
Generating Datasheets	5-11
Datasheet Generation Overview	5-12
Datasheets in HTML Format	5-15
Datasheets in Text Format	5-16
HTML and Text Datasheet Details	5-17
Logo (HTML)	5-17
Cell (HTML and Text)	5-17
Cell Description File (HTML only)	5-17
Function Description (HTML and Text)	5-18
Symbol (HTML Only)	5-23
Schematic (HTML Only)	5-23
Port Names (HTML and Text)	5-24
Cell Area (HTML and Text)	5-25
Pin Capacitance (HTML and Text)	5-25
Delay Data (HTML and Text)	5-27
Constraint Data (HTML and Text)	5-29
Power Data (HTML Only)	5-31
Generating Verilog Models	5-33
Overview	5-33
Generating Verilog Files	5-33
Creating UDPs	5-34
Creating Independent UDPs	5-34
Creating Custom UDPs	5-35
Verilog Model Details	5-42
Header Section	5-42
Function Description	5-44
Timing Checks	5-52
UDP Definition	5-67
Creating the Testbench	5-69
6. Transistor Mismatch Characterization	
Overview of Transistor Mismatch	6-2
Synthetic Variation Parameters	6-2
How Mismatch Characterization is Performed	6-5
Mismatch Characterization Options	6-6

Sigma and Mean Support for Mismatch Analysis	6-7
Defining the Model File to Control Mismatch Parameters	6-8
7. CCS Noise Characterization	
Overview of CCS Noise Characterization	7-2
Requirements for Noise Characterization	7-3
CCS Noise Characterization Setup	7-4
8. Troubleshooting	
Troubleshooting	8-2
Frequently Asked Questions	8-4
Appendix A. Generic Simulator Interface	
Overview	A-2
Simulator Interface Components	A-3
Invoking the Third-Party Simulator Interface	A-5
Appendix B. CCS Models	
Driver Models	B-2
Receiver Models	B-5
Index	

Preface

This preface includes the following sections:

- [What's New in This Release](#)
- [About This Guide](#)
- [Customer Support](#)

What's New in This Release

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

To see the *Liberty NCX Release Notes*,

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

<https://solvnet.synopsys.com/DownloadCenter>

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

2. Select Liberty NCX, and then select a release in the list that appears.
-

About This Guide

This user guide describes how to use Liberty NCX to perform library characterization from a set of cell functional descriptions, associated SPICE netlists, and process model files.

Audience

This manual is intended for engineers who create .lib libraries.

Related Publications

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

<https://solvnet.synopsys.com/DocsOnWeb>

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

- HSPICE
- Library Compiler

Conventions

The following conventions are used in Synopsys documentation.

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

Customer Support

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

Accessing SolvNet

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

To access SolvNet, go to the following address:

<https://solvnet.synopsys.com>

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

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

Contacting the Synopsys Technical Support Center

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

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

1

Running Liberty NCX

Liberty NCX is a software tool that generates a library in Liberty (.lib) format from a set of SPICE models, cell functional descriptions, and associated netlists. The library can then be used for timing, power, and noise analysis with various tools. In addition, Liberty NCX can convert existing libraries from one format to another.

This chapter contains the following sections:

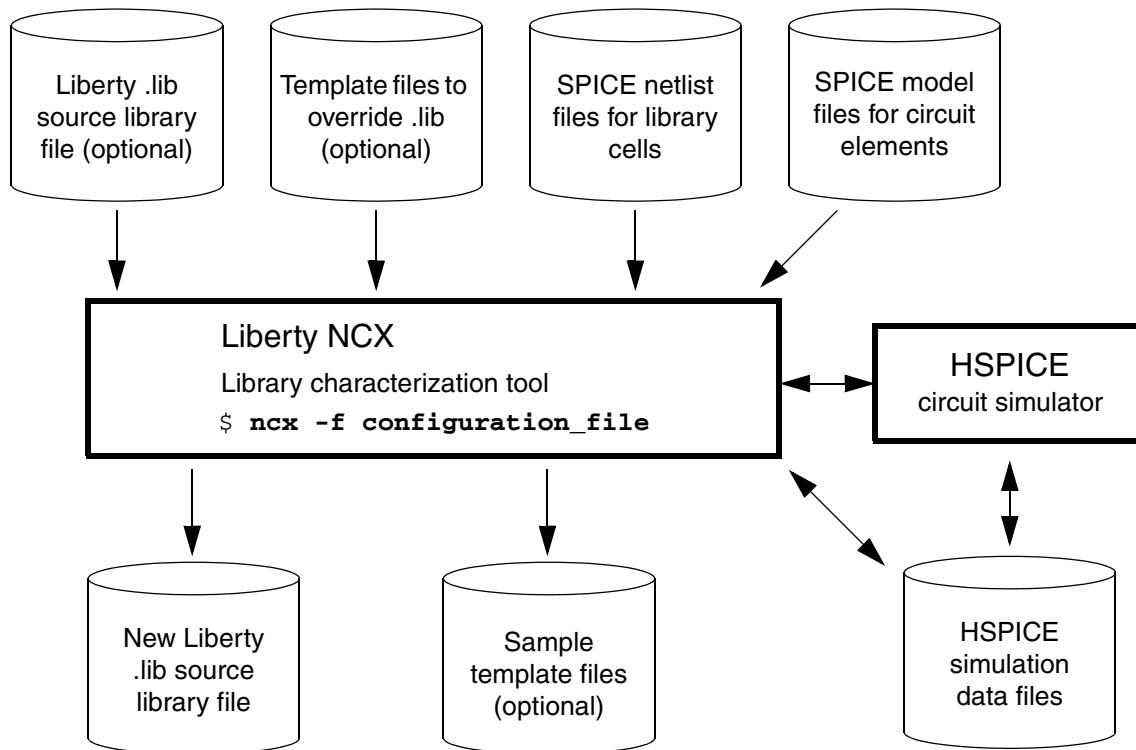
- [Overview of Liberty NCX](#)
- [Invoking Liberty NCX](#)
- [Liberty NCX Usage Flows](#)

Overview of Liberty NCX

Liberty NCX is a software tool that generates a library in Liberty (.lib) format from a set of SPICE models, cell functional descriptions, and associated netlists. The generated library can then be used for timing, power, and noise analysis with compatible tools such as Library Compiler, IC Compiler, Design Compiler, and PrimeTime. The tool can generate CCS, NLDM, and NLPM models for the library.

Figure 1-1 shows the data files used for characterization. You start with either an existing seed library or a set of template files containing high-level descriptions of each cell, a set of SPICE netlist files for the cells to be characterized, and a set of SPICE model files for the circuit elements used in the cell netlists.

Figure 1-1 Liberty NCX Characterization Data



You can also use Liberty NCX to perform various types of formatting operations on existing libraries, such as merging data from different libraries or converting library models from one form to another. The set of formatting capabilities is called the Model Adaptation System.

You invoke Liberty NCX by entering the `ncx` command at the operating system prompt. Liberty NCX reads cell descriptions from the seed library or cell template files, reads SPICE netlists and models, sets up SPICE simulations of the cells, runs the simulations using

HSPICE or a compatible simulator, gathers the simulation results, generates the library cell characterization data, and writes out a .lib library file containing the newly characterized cells.

Liberty NCX gets the cell functionality information from the descriptions contained in the existing .lib seed library or user-supplied template files. You can add more cells or override the parameters for existing cells by providing a library template file for global parameters or individual cell template files for the cells and cell parameters being characterized.

Invoking Liberty NCX

Before you can use Liberty NCX, you must install the software and obtain a license to use the product. You must also have HSPICE or a compatible circuit simulator.

You first create a configuration file, also known as the command file, that sets a number of configuration variables for the characterization run. Then you invoke Liberty NCX, which reads and applies the configuration variable settings from the specified file. For example, you might create a configuration file called “myconfig” containing the following configuration variable definitions:

```
set model_file model/model.best
set netlist_dir netlist
set simulator_exec /remote/ckt/hspice/bin/hspice
set input_library a_12cells.lib
set output_library aout.lib
set work_dir tmp
set farm_type LSF
set ccs_timing true
set nldm true
set templates true
```

Then, to invoke Liberty NCX with the option settings contained in the configuration file, use the following command:

```
% ncx -f myconfig
```

For a characterization task, the configuration file must specify the SPICE model file name, the SPICE netlist directory, and the SPICE simulator executable. The “mycmd” example also specifies the input library name, the output library name, the working directory name, and the distributed processing farm type. It also specifies that both CCS and NLDM timing models are to be generated in the library and that template files are to be generated for future specification of characterization parameters.

For a library formatting task using the Model Adaptation System, the ncx configuration file must specify the input library name, the output library name, and parameters for the formatting or conversion operation.

Online Help

To get a listing of the possible configuration variable settings, use the help option:

```
% ncx -help command
...
Files/folders:
  set input_library : (string) input seed library file name
  set output_library : (string) output library file name
  set output_cell_library
    : (string) name template for output cell libraries
  set log_file      : (string) output log file name [ncx.log]
  set work_dir      : (string) directory to contain NCX output files ...
...
...
```

You can view man pages for many Liberty NCX attributes and attribute settings. At the operating system prompt, enter man followed by the attribute name or setting. For example,

```
% man templates
...
NAME
  templates

TYPE
  boolean

DEFAULT
  false

DESCRIPTION
  Causes generation of sample templates for the library and cells. You can modify the generated template files to specify the characterization parameters for future runs of Liberty NCX. The files are written to the directory defined by output_template_dir(2).

  Specified in the configuration file.

EXAMPLES
  set templates true

SEE ALSO
  input_template_dir(2)
  output_template_dir(2)
  timing_arcs_to_template(2)
  sensitization_to_template(2)
  library_template_file(2)
  template_suffix(2)
...
...
```

For a long man page, press the space bar to view successive screenfuls of text. To close the man page, type the “q” key (for “quit”).

To see the man pages, the `MANPATH` environment variable must be properly set. In a C shell, add the following line to the `.cshrc` file:

```
setenv MANPATH $SYNOPSYS_NCX_ROOT/ncx/man
```

In a Bourne, Korn, or Bash shell, add the following line to the `.profile`, `.kshrc`, or `.bashrc` file:

```
MANPATH=$SYNOPSYS_NCX_ROOT/ncx/man  
export MANPATH
```

Characterization Settings

The following tables briefly describe the characterization settings:

[Table 1-1 File and Directory Settings](#)

[Table 1-2 Characterization Flow Settings](#)

[Table 1-3 Simulation Settings](#)

[Table 1-4 Compute Farm Settings](#)

[Table 1-5 IBIS Model Generation Settings](#)

[Table 1-6 Timing, Power, and Noise Model Acquisition Settings](#)

[Table 1-7 Template Usage Settings](#)

[Table 1-8 Output Format Control Settings](#)

[Table 1-9 Program Control Settings](#)

[Table 1-10 Library Postprocessing Settings](#)

The following tables briefly describe the Model Adaptation System settings:

[Table 1-11 Library Format Translation Settings](#)

[Table 1-12 Compact/Expand CCS Settings](#)

[Table 1-13 Merge to One Variation-Aware Library Settings](#)

[Table 1-14 Merge to One CCS Noise Library Options](#)

[Table 1-15 Format CCS to ECSV Options](#)

[Table 1-16 Format Variation-Aware CCS to S-ECSV Options](#)

[Table 1-17 Format CCS to NLDM Library Options](#)

Table 1-1 File and Directory Settings

Configuration setting	Description	Default
input_library	The name of the existing Liberty (.lib) source file. Liberty NCX uses this library to gather cell function descriptions. If no input library is specified, Liberty NCX generates an entirely new library using the library and cell templates.	
log_file	The name of the file in which Liberty NCX stores the session log.	ncx.log
output_cell_library	If specified, generates an individual output Liberty (.lib) file for each cell characterized by Liberty NCX in the current run. The extension is kept, but the base name of the specified library name is discarded and replaced with each cell name. This option can be used together with the output_library setting.	
output_library	The name of the new Liberty (.lib) library file created by Liberty NCX.	
work_dir	The name of the working directory used by Liberty NCX to write all working files.	work

Table 1-2 Characterization Flow Settings

Configuration setting	Description	Default
auto_function	Invokes AutoFunction automatic cell function recognition. For details, see “ Automatic Function Recognition ” on page 2-16.	true

Table 1-2 Characterization Flow Settings (Continued)

Configuration setting	Description	Default
compact	If CCS models are being generated, causes generation of driver models in compact form. Compact models contain the same current-source information as standard models, but with the information encoded as a set of current-versus-voltage waveform parameters to reduce the size of the characterized library. When the <code>compact</code> configuration variable is set to <code>true</code> (the default), Liberty NCX performs compaction for CCS power and CCS timing models.	<code>true</code>
ibis	Runs the Input/Output Buffer Information Specification (IBIS) model generation flow in Liberty NCX. <code>set ibis {true false}</code> For more information, see Chapter 4, “I/O Cell Characterization.”	<code>false</code>
noise	Causes acquisition of CCS noise models.	<code>false</code>
power	Causes acquisition of NLPM and CCS power models.	<code>false</code>
preanalysis_opt	Allows you to save the pre-analysis optimization information for performing characterization on other corners. If the template files do not include optimization information and this variable is set to <code>auto</code> , the default setting, Liberty NCX generates and saves new optimization information in the template files. If the optimization information already exists, Liberty NCX does not rerun the optimization; it reuses the information in the template files. If you want to rerun optimizations whether or not optimization information is included in the template files, set <code>preanalysis_opt</code> to <code>force</code> . If you want Liberty NCX to ignore the existing optimization information during characterization, set <code>preanalysis_opt</code> to <code>false</code> .	<code>auto</code>
prechar	Allows you to review and modify the templates before performing characterization. When you set <code>prechar</code> to <code>true</code> , Liberty NCX writes the information from your seed library to the template files and stops after the optimization is complete.	<code>false</code>
soi	Performs silicon-on-insulator (SOI) characterization.	<code>false</code>
soi_mode	Post-processes SOI library to generate min/max library.	<code>false</code>

Table 1-2 Characterization Flow Settings (Continued)

Configuration setting	Description	Default
templates	Causes generation of sample templates for the library and cells. You can modify the generated template files to specify the characterization parameters for future runs of Liberty NCX. The files are written to the directory defined by the <code>output_template_dir</code> variable.	false
timing	Causes acquisition of NLDM and CCS timing models.	false

Table 1-3 Simulation Settings

Configuration setting	Description	Default
enable_generic_simulator_interface	Enables the use of a generic (third-party) translator script for the simulator interface.	false
hspice_server	Liberty NCX supports the HSPICE client/server mode if you have HSPICE version A-2008.03-SP1 or later. Using the client/server mode improves Liberty NCX performance by minimizing the overhead associated with license checkout and model processing in HSPICE.	false
leakage_model_file	Allows you to define a different process model just for leakage power acquisition. The value should be the full path to the desired process file as shown: <code>set leakage_model_file leakage_model_path</code>	
model_file	Specifies the name of the SPICE model file containing the process and device models for the circuit elements used in the cell netlists.	
netlist_dir	Specifies the directory containing the SPICE netlists for the cells being characterized. There must be a separate SPICE netlist file for each cell being characterized.	
netlist_file	Specifies the name of a file containing the SPICE netlists for the cells being characterized. If you specify a relative path, the path is relative to the directory specified by <code>netlist_dir</code> . Liberty NCX splits all of the subcircuit descriptions in the file into separate netlists, each named <code>cellname.netlist_suffix</code> , in a subdirectory called “netlists” in the directory specified by <code>work_dir</code> . The subcircuit descriptions created in this manner cannot be used hierarchically within other subcircuits.	

Table 1-3 Simulation Settings (Continued)

Configuration setting	Description	Default
netlist_suffix	Specifies the file name extension used by Liberty NCX to recognize the SPICE netlist files in the cell netlist directory.	.spc
simulation_dir	Specifies the top-level directory used for storing the simulation data files. Within this directory, Liberty NCX stores data in cell directories. Within each cell directory, Liberty NCX stores data in the arc directories.	.
simulator_exec	Specifies the absolute path to the SPICE circuit simulator executable. Supported simulators are <code>hspice</code> , <code>eldo</code> , and <code>spectre</code> .	
simulator_output_extractor	Specifies the SPICE extractor script file for the generic (third-party) simulator interface.	
simulator_setup_file	Specifies the SPICE simulator setup file name.	
simulator_type	Specifies the type of circuit simulator: <code>hspice</code> , <code>eldo</code> , or <code>spectre</code> . The default simulator is <code>hspice</code> . You can also invoke the Eldo simulator by setting the <code>simulator_type</code> to <code>eldo</code> or use the Spectre simulator by setting <code>simulator_type</code> to <code>spectre</code> . For example, set <code>simulator_type eldo</code> .	<code>hspice</code>
spice_netlist_translator	Specifies the SPICE translator script file for the generic (third-party) simulator interface.	

Table 1-4 Compute Farm Settings

Configuration setting	Description	Default
advanced_dp	Enables an advanced distributed process controller that provides better performance. For details, see “ Using The Advanced Distributed Process Controller ” on page 1-22.”	false
backup_failed_sims	Keep a backup of failed simulation directories.	false
bundle_size	Specifies the number of simulations combined into a single compute farm job. Setting a larger value reduces farm management overhead but increases the runtime of each compute farm job.	50

Table 1-4 Compute Farm Settings (Continued)

Configuration setting	Description	Default
concurrent_ncx	When set to a value of 2 or greater, enables concurrent characterization of the library using the specified number of concurrent processes. For more information, see “Performing Concurrent Characterization” on page 1-21 .	
concurrent_ncx_farm_type	Specifies the type of compute farm to be used for the concurrent characterization processes. Valid values are the same as for the <code>farm_type</code> configuration variable. A value of <code>no_farm</code> launches the concurrent NCX processes on the current host machine.	no_farm
concurrent_ncx_queue_name	Specifies the compute farm queue name to be used for the concurrent characterization processes. Usage is the same as for the <code>queue_name</code> configuration variable.	
concurrent_ncx_resource	Specifies the compute farm resource requirements for the concurrent characterization processes, such as memory, swap space, CPU load, or host type. Usage is the same as for the <code>resource</code> configuration variable.	
constraint_bundle_size	Specifies the number of constraint simulations per farm job.	1
farm_mgr_exec	Specifies the location of the user script that manages the custom farm. Specifying this variable is mandatory when the <code>farm_type</code> variable is set to <code>user</code> .	
farm_retry_interval	Specifies the number of seconds the tool should wait before resubmitting a failed job to the compute farm again.	3
farm_retry_limit	Specifies the maximum number of farm job resubmission attempts for the same job. See also the <code>update_interval</code> variable, which specifies the time interval between the compute farm status checks.	3
farm_type	Specifies the multiprocessor farm system. Can be set to <code>LSF</code> (Platform Computing LSF), <code>SGE</code> (Sun Grid Engine), <code>user</code> , or <code>nofarm</code> (local processor usage only). For LSF or SGE, you must run Liberty NCX on a submit machine that can accept <code>bsub</code> (LSF) or <code>qsub</code> (SGE) job submission variables. Setting <code>user</code> allows you to specify your own script for managing the farm.	LSF

Table 1-4 Compute Farm Settings (Continued)

Configuration setting	Description	Default
<code>farm_update_file</code>	Set <code>farm_update_file</code> to point to a specified file, and you can update the file at any point during the NCX run. For more information, see “ Real-Time Control of Distributed Job Configuration ” on page 1-23.	
<code>job_done_identifier</code>	Identifiers in <code>ncxjob.*</code> files that indicate job termination.	
<code>max_jobs</code>	Specifies the maximum number of jobs that can be active on the compute farm at the same time. Liberty NCX postpones submitting new jobs when the current number of unfinished (pending and running) jobs reaches the specified number. If the number of jobs submitted for execution exceeds the current number of available licenses, the extra simulation jobs will fail.	100
<code>max_job_time</code>	Specifies the maximum allowable execution time for any one submitted job in the compute farm, in minutes of CPU time. A job fails if its execution time exceeds this value.	600
<code>model_extraction_to_farm</code>	Specifies that Liberty NCX perform distributed model extraction for simulations that are run on the farm. This reduces runtime and optimizes disk usage by creating compact, intermediate model files that eliminate the need to store simulator output files. To use distributed model extraction: <code>set model_extraction_to_farm to true.</code> Note: Distributed model extraction is currently supported only for the HSPICE simulator. For more information about distributed model extraction, see “ Distributed Model Extraction ” on page 1-22.	false
<code>ncx_exec</code>	Platform-independent Liberty NCX executable for distributed model extraction.	
<code>project_name</code>	Specifies the compute farm project name. <code>project_name</code> is often used on SGE to address <code>qsub -P project_name</code> .	ncxtest
<code>queue_name</code>	Specifies the compute farm queue name.	normal

Table 1-4 Compute Farm Settings (Continued)

Configuration setting	Description	Default
resource	Specifies the compute farm resource requirements such as memory, swap space, CPU load, or host type. LSF resources can be displayed with the <code>lsinfo</code> command. For example, <code>set farm_type LSF</code> and <code>set resource opteron</code> will cause Liberty NCX to run AMD Opteron Linux systems in the LSF farm by submitting jobs using <code>bsub -R opteron script_file.bat</code> . Possible SGE resource names include arch (solaris64, glinux), cputype (sparc, ia32, amd64), os_bit (64, 32), and os_distribution (SunOS, SuSE, redhat). For example, <code>set farm_type SGE</code> and <code>set resource arch=solaris64</code> sets the architecture to Solaris. <code>resource</code> is often used on LSF to address <code>bsub -q queue_name</code> .	
update_interval_time	Specifies the update interval time, in seconds. This is the time interval between status checks of the compute farm. If set to too small a value, the status check requests may overwhelm the compute farm manager and degrade performance.	60

Table 1-5 IBIS Model Generation Settings

Configuration setting	Description	Default
ibis_dir	Specifies the intended location of IBIS model files generated by Liberty NCX. One IBIS model is placed in this directory for each cell that is to have an IBIS model generated. The valid values are of <code>string</code> type. For more information, see Chapter 4, “I/O Cell Characterization.”	
ibis_version	Specifies the version of the IBIS model generated by Liberty NCX. Valid values are: 3.2, 4.0 or greater. For more information, see Chapter 4, “I/O Cell Characterization.”	4.0
sim_debug_mode	Enable netlist debug mode.	true

Table 1-6 Timing, Power, and Noise Model Acquisition Settings

Configuration setting	Description	Default
auto_function	Automatically specifies functional information, such as flip-flop groups, latch groups, state tables, and functions in either the seed library or the template files for you. Autofunction recognition generates the functional information by performing a topological analysis of the SPICE netlist.	false
capacitance	Causes acquisition of capacitance and CCS receiver models.	true
noise	Causes acquisition of CCS noise models.	false
ccs_power	Causes acquisition of CCS power models.	true
ccs_timing	Causes acquisition of CCS timing models.	true
compact_power	Sets the <code>compact_power</code> variable to <code>true</code> to enable compact CCS power. If you set <code>compact_power</code> to <code>false</code> , Liberty NCX generates expanded CCS power models.	true
compact_timing	Sets the <code>compact_timing</code> variable to <code>true</code> to enable compact CCS timing. If you set <code>compact_timing</code> to <code>false</code> , Liberty NCX generates expanded CCS timing model.	true
constraint	Causes the acquisition of timing constraint or violation arcs: setup, hold, recovery, removal, and minimum pulse width. If you are only interested in delay and slew, set this option to <code>false</code> to reduce the runtime.	true
delay	Causes acquisition of delay arcs and CCS driver models.	true
design_rules	Causes extraction of maximum capacitance and maximum transition rules.	false
max_capacitance	Extracts maximum capacitance if <code>design_rules</code> is true.	true
max_transition	Extracts maximum transition time if <code>design_rules</code> is true.	true
mismatch	Generates transistor mismatch models.	false
nldm	Causes acquisition of NLDM timing models.	true
nlpm	Causes acquisition of NLPM power models.	false

Table 1-6 Timing, Power, and Noise Model Acquisition Settings (Continued)

Configuration setting	Description	Default
shpr_constraint	Causes characterization of interdependent setup and hold models in addition to conventional setup and hold models.	false
variation	Causes generation of a single, merged variation-aware library containing cells characterized at given sets of variation parameter values specified in the library template.	false
variation_leakage	Causes generation of variation-aware leakage models.	false

Table 1-7 Template Usage Settings

Configuration setting	Description	Default
input_template_dir	Specifies the name of the directory containing the input template files. These files specify the cells that need to be characterized and the library and cell characterization parameters.	.
output_template_dir	Specifies the name of the directory where the output template files are to be written. These files can be used as input template files in future runs of Liberty NCX.	
library_template_file	Specifies the name of the input/output library template file. If you do not set this parameter, Liberty NCX uses the input library name and appends ".opt".	
library_name	Specifies the name of the library.	
template_suffix	Specifies the template file name extension used by Liberty NCX to identify template files.	.opt
timing_arcs_to_template	Specifies whether to explicitly write timing arcs to the output template, if generated by Liberty NCX.	false
sensitization_to_template	Specifies whether to write sensitization information to the output template.	true
arc_specs_to_template	Writes arc specifications into templates.	true
indexes_to_template	Writes index variables into templates with values set to zero.	true

Table 1-8 Output Format Control Settings

Configuration setting	Description	Default
driver_waveform_to_library	Saves the predriver information into the output library. The saved description provides enough information to reproduce the exact predriver waveform used for characterization.	true
failed_cells_to_library	Writes failed cells to output library.	false
only_active_cells_to_library	Specifies whether to write only the cells that have been characterized by Liberty NCX to the output library.	false
output_library_compression	Specifies the compression that should be applied to the Liberty (.lib) library file created by Liberty NCX. Allowed values are none and gzip.	none
precision	Specifies the number of digits of precision used to represent floating-point values in the generated library.	7
sensitization_to_library	Writes cell sensitization information to the output library.	false
sort_output_library	Sorts cells in output library in case-independent alphanumeric order.	false
use_driver_waveform_from_library	Specifies that the predriver information in the seed library be used for characterization.	false

Table 1-9 Program Control Settings

Configuration setting	Description	Default
autofix	Attempts the automatic fixing of errors. With automatic fixing, if the clock period is too short to allow a cell to complete a full transition, Liberty NCX increases the clock width and resimulates the failing arcs. If Liberty NCX cannot extract an accurate model, it resimulates the failing arc using the HSPICE runlv parameter set to 6, for higher accuracy at the cost of additional runtime.	true

Table 1-9 Program Control Settings (Continued)

Configuration setting	Description	Default
fix_nldm_timing	<p>Ensures monotonically increasing cell delays with increasing output load capacitance in the generated NLDM timing models.</p> <p>This variable accepts the following values: <code>delay</code>, <code>slew</code>, <code>both</code>, and <code>none</code>. When you set <code>fix_nldm_timing</code> to <code>delay</code>, the tool ensures monotonically increasing cell delays with increasing output load capacitance in the generated NLDM timing models.</p> <p>When you set the variable to <code>slew</code>, the tool ensures monotonically increasing cell transitions with increasing output load capacitance in the generated NLDM timing models.</p> <p>When you set the variable to <code>both</code>, the tool ensures monotonicity for cell delays and cell transition tables in the generated NLDM models.</p> <p>When you set the variable to <code>none</code>, the tool does not ensure monotonicity for cell delay and transition tables.</p> <p>For backward compatibility, the tool internally translates <code>true</code> to <code>delay</code> and <code>false</code> to <code>none</code>.</p>	delay
cleanup	<p>Specifies the level of cleanup done by the deletion of simulation result files after characterization. There are five possible settings: 0, 1, 2, 3, and 4. A setting of 0 results in no cleaning; all simulation results are preserved. A setting of 1 causes simulation results to be preserved in gzip-compressed format. A setting of 2 causes all simulation result files to be deleted. A setting of 3 causes the entire simulation directory, including netlists, to be deleted. A setting of 4 causes the deletion of simulation directories for all successful cells. For any of these settings, the cell simulation directories are preserved if there is a characterization failure of any kind.</p>	1
reuse	<p>Causes Liberty NCX to use existing simulation results from previous runs, if available. Reusing existing simulation data saves runtime, but the data must be valid for the current run in order to get accurate results.</p>	false
test_simulator	<p>Generates and simulates a small test circuit prior to the start of characterization, in order to ensure valid simulator and model settings.</p>	true

Table 1-10 Library Postprocessing Settings

Configuration setting	Description	Default
margin_data_to_library	Controls the addition of the margin data as user-defined attributes to the library. Margin data is an expression used for margin calculations, calculated margin values, and so forth. Using these attributes, the margins can be recalculated, added, or removed in the postprocess step. By default, this attribute is set to <code>false</code> , so no margin data is written to the library.	<code>false</code>
postprocess	Enables the library adjustment flow. During this process, Liberty NCX reads an input library and performs postprocessing without characterization (such as addition or removal of margins, change in the delay threshold, and so forth). In the postprocessing step, no simulations are done.	<code>false</code>

Table 1-11 Library Format Translation Settings

Configuration setting	Description	Default
expand_ccs	Converts a compact CCS library into an expanded CCS library.	<code>false</code>
compact_ccs	Converts an expanded CCS library into a compact CCS library.	<code>false</code>
va_merge	Merges a set of variation-aware libraries into a single library. Both compact and expanded CCS libraries are accepted. The output library uses compact CCS.	<code>false</code>
ccsn_merge	Merges a library with CCS timing models and another library with CCS noise models into a single library that has both types of models.	<code>false</code>
ccs2ecsm	Converts a CCS library into a Cadence ECSM library.	<code>false</code>
vaccs2seccsm	Converts a variation-aware CCS library to a statistical S-ECSM library.	<code>false</code>
ccs2nldm	Converts a CCS library into an NLDM library.	<code>false</code>

Table 1-12 Compact/Expand CCS Settings

Configuration setting	Description	Default
input_library	Specifies the name of the existing library file used as input to the formatting operation.	
output_library	Specifies the name of the output library file produced by the formatting operation.	

Table 1-13 Merge to One Variation-Aware Library Settings

Configuration setting	Description	Default
nominal_library	Specifies the name of the nominal variation-aware library, followed by the parameter names and nominal values: <i>lib_name par1 val1 par2 val2 ...</i>	
va_library_list	Specifies the names of the 2N variation-aware libraries to be merged into a single library, each with its parameter values in the same order as specified for the nominal library: <i>{lib1_name val1 val2 ... lib2_name val1 val2 ... lib2N val1 val2 ... }</i>	
output_library	Specifies the name of the unified variation-aware library created by merging the input libraries.	
slew_indexes	Specifies a list of the slew index values that are to be retained in the output library (for example, {1 2 3 5} to retain the first, second, third, and fifth index values), or the string <code>all</code> to retain all the slew index values. This setting affects only the libraries in the <code>library_list</code> . All index values in the nominal library are always retained.	all
load_indexes	Specifies a list of the load index values that are to be retained in the output library (for example, {1 2 3 5} to retain the first, second, third, and fifth index values), or the string <code>all</code> to retain all the load index values. This setting affects only the libraries in the <code>library_list</code> . All index values in the nominal library are always retained.	all

Table 1-14 Merge to One CCS Noise Library Options

Configuration setting	Description	Default
timing_library	Specifies the existing library containing CCS timing models.	
noise_library	Specifies the existing library containing CCS noise models.	
output_library	Specifies the newly generated output library file.	

Table 1-15 Format CCS to ECSM Options

Configuration setting	Description	Default
library_list	Specifies a single library to be converted from CCS format to Cadence ECSM format, or a list of CCS libraries to be scaled to produce an ECSM library. The slew and load index values of the first library are used in the output library. Both compact and expanded CCS libraries are accepted.	
output_library	Specifies the new ECSM library file name.	
process	Specifies the process parameter value of the generated library. Liberty NCX scales the process data between the input libraries to produce the output library.	process value in first library
voltage	Specifies the voltage parameter value of the generated library. Liberty NCX scales the voltage data between the input libraries to produce the output library.	voltage value in first library
temperature	Specifies the temperature parameter of the generated library. Liberty NCX scales the temperature data between the input libraries to produce the output library.	temperature value in first library
sample	Specifies the sequence of voltage sample points in the ECSM library.	{0.05 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 0.95}
capacitance_mode	Specifies which of the two CCS receiver capacitance values to use as the ECSM receiver capacitance: first, second, ave, min, or max (first value, second value, average of two values, smaller value, or larger value).	first
mode	Specifies which parts of the CCS models are converted: timing, noise, or both.	timing

Table 1-15 Format CCS to ECSM Options (Continued)

Configuration setting	Description	Default
ecsm_vhtolerance	Specifies the variation in voltage that can be considered negligible when the signals are high.	0.0
ecsm_vltolerance	Specifies the variation in voltage that can be considered negligible when the signals are low.	0.0

Table 1-16 Format Variation-Aware CCS to S-ECSM Options

Configuration setting	Description	Default
input_library	Specifies input seed library file name.	
output_library	Specifies output library file name.	
capacitance_mode	Specifies which of the two CCS receiver capacitance values to use as the ECSM receiver capacitance: first, second, ave, min, or max (first value, second value, average of two values, smaller value, or larger value).	first
sample	Specifies the sequence of voltage sample points.	{0.05 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 0.95}
s_ecsm_param_class_unit_list	Provides list of parameters, classes, and units: <i>param1 class1 unit1 ... paramn classn unitn</i>	

Table 1-17 Format CCS to NLDM Library Options

Configuration setting	Description	Default
library_list	Specifies a single library to be converted from CCS format to Cadence ECSM format, or a list of CCS libraries to be scaled to produce the ECSM library. The slew and load index values of the first library are used in the output library. Both compact and expanded CCS libraries are accepted.	
output_library	Specifies the new NLDM library file name.	

Table 1-17 Format CCS to NLDM Library Options (Continued)

Configuration setting	Description	Default
process	Specifies the process parameter value of the generated library. Liberty NCX scales the process data between the input libraries to produce the output library.	process value in first library
voltage	Specifies the voltage parameter value of the generated library. Liberty NCX scales the voltage data between the input libraries to produce the output library.	process value in first library
temperature	Specifies the temperature parameter of the generated library. Liberty NCX scales the temperature data between the input libraries to produce the output library.	temperature value in first library
nldm_cap	Controls NLDM capacitance conversion method.	c1

Performing Concurrent Characterization

You can control the number of concurrent Liberty NCX subprocesses used for library characterization by setting the `concurrent_ncx` variable in the configuration file.

The `concurrent_ncx` variable is undefined by default. This results in a single characterization process on the host machine, and no concurrent characterization is performed.

When it is set to an integer value of 2 or greater, concurrent characterization is enabled, and the integer value specifies the number of subprocesses to launch. This number should not exceed the available processor and license resources. The total number of required Liberty NCX licenses is equal to the value of the `concurrent_ncx` variable.

When concurrent characterization is enabled, the parent Liberty NCX process invokes the specified number of additional Liberty NCX subprocesses. Characterization of the library cells is automatically distributed across the subprocesses. Once all subprocesses complete characterization of their cell sets, the subprocesses terminate, and the parent process continues with the merged set of data.

You can specify the location of the concurrent characterization processes with the `concurrent_ncx_farm_type` configuration variable. A value of `no_farm` launches the processes on the current host machine. A value of `LSF`, `SGE`, or `user` launches the processes on the specified compute farm. When a compute farm is used, you can use the `concurrent_ncx_resource` and `concurrent_ncx_queue_name` configuration variables to

specify additional information for the farm job submission commands. You should ensure that these distributed characterization processes have job submission privileges to launch their own simulations using the `farm_type` method.

Note:

When concurrent characterization is enabled, the `max_jobs` configuration variable specifies the maximum number of farm jobs across all subprocesses. Liberty NCX manages the use of the farm jobs across the characterization processes to stay within the specified `max_jobs` limit value.

Using The Advanced Distributed Process Controller

You can use the advanced distributed process controller by setting the `advanced_dp` configuration variable to `true`. When enabled, distributed jobs are launched once, and kept active and reused for multiple simulation bundles. When a compute farm is used, this minimizes the amount of time spent waiting for pending compute farm jobs to become active.

The advanced distributed process controller requires that the distributed jobs be run on machines whose platform and operating system match the parent process. When a compute farm is specified with the `farm_type` configuration variable, you can use the `resource` configuration variable to specify job resource requirements if your compute farm contains multiple compute platforms and operating systems.

The advanced distributed process controller is disabled by default to allow for a mix of distributed job host machine characteristics.

Note:

The advanced distributed process controller does not support the `user` farm type.

Distributed Model Extraction

Liberty NCX can perform distributed model extraction for simulations that are run on the farm. This reduces runtime and optimizes disk usage by creating compact, intermediate model files that eliminate the need to store simulator output files. Liberty NCX uses a `/tmp` temporary directory on the farm machines for storing SPICE simulation results, which greatly reduces I/O operations on the network disk when a large number of simulations are being run in parallel.

Liberty NCX also transfers the log and model files only to the network disk after simulation and extraction is complete when you set the `cleanup` configuration variable to `2`. If you also want to store simulation results on the network disk, you can set `cleanup` to `1` in the input configuration file.

To use distributed model extraction, set `model_extraction_to_farm` to `true` in the configuration file, as shown:

```
set model_extraction_to_farm true
```

The `model_extraction_to_farm` variable is set to `false` by default.

Note:

If a simulator other than HSPICE is used, the generic simulator interface must be used. For more information, see [Appendix A, “Generic Simulator Interface.”](#)

Real-Time Control of Distributed Job Configuration

You can use the `farm_update_file` configuration variable to enable real-time control of distributed job configuration settings while characterization is running. This can be used to alter the job resource requirements based on the time of the day, the load on the compute farm, and other dynamic factors. Set the `farm_update_file` variable in the configuration file to point to a specified file that defines the distributed job configuration settings. The contents of the file can contain one or more of the following configuration variables:

```
set max_jobs value  
set update_interval value  
set bundle_size value  
set constraint_bundle_size value
```

This file can be updated at any point during the Liberty NCX run. The new settings are applied to subsequent distributed jobs.

You can also use the `farm_update_file` capability together with distributed model extraction. The settings are enabled the next time Liberty NCX dispatches jobs to the queue.

Liberty NCX Usage Flows

Liberty NCX can be used to perform two kinds of tasks: characterization and library formatting. Characterization means creating a new library or adding new cell models to an existing library. Library formatting means converting the data in an existing library to a different format. The set of library formatting capabilities is called the Model Adaptation System.

When Liberty NCX performs characterization, it creates a new cell library or adds new models to an existing library in Liberty (.lib) format. To characterize a cell, it runs SPICE simulations of the cell under various conditions and records the cell behavior into its database. Then it writes out the cell model in Liberty format. You can use Library Compiler

or a similar tool to compile the .lib description into a form that can be used by timing, power, and noise analysis tools. The characterization process is described in [Chapter 2, “Characterization Flows.”](#)

If you specify the name of an existing library for a characterization task, Liberty NCX performs recharacterization of that library. Otherwise, it performs characterization of an entirely new library. A set of template files is usually required to specify the parameters such as the list of cells to be characterized and the sensitization of the cells. The template file format is described in [Chapter 3, “Template Files.”](#)

For information about characterization I/O cells, including the generation of data in IBIS format, see [Chapter 4, “I/O Cell Characterization.”](#)

Liberty NCX can perform the following types of library formatting:

- Compacting/expanding CCS models
- Variation-aware library merging
- CCS Noise model merging
- CCS-to-ECSM conversion
- VA-CCS-to-S-ECSM conversion
- CCS-to-NLDM conversion

These operations are described in [Chapter 5, “Library Conversion Support.”](#)

2

Characterization Flows

To perform characterization, Liberty NCX runs circuit simulations of the library cells to determine the cell behavior. It writes out a description of the cell characteristics in Liberty (.lib) format. You can then use Library Compiler to generate library data that can be used for timing, power, and noise analysis.

The characterization process is described in the following sections:

- [Characterization Input Data](#)
- [Characterization Output Data](#)
- [Liberty NCX Operation and Log File](#)
- [Partial Sensitization Flow](#)
- [Automatic Function Recognition](#)
- [Acquisition Types](#)
- [Incremental Characterization](#)

Characterization Input Data

To perform characterization, you must provide the SPICE netlist files for the library cells and the SPICE model file for the circuit elements used in the cell netlists. You can optionally provide an existing .lib library file and one or more template files to specify the library or cell characterization parameters. For example, you can add new cells to an existing library or recharacterize existing cells using new parameter settings. Other cells in the existing library are left unchanged. The template files specify the cells to be characterized and the parameters for characterization.

If you are creating a new library, then it is not necessary to provide an existing library file. In that case, you must provide a library template file and one or more cell template files to specify the parameters for the new library and its cells.

Required Input Data

There must be a SPICE netlist for each library cell being characterized. The netlist files must reside in the directory specified by the `netlist_dir` setting in Liberty NCX.

You can also read in a single file containing all the netlists. Use `netlist_file` in the Liberty NCX configuration file to specify the name of a file containing the SPICE netlists. If you specify a relative path, the path is relative to the directory specified by `netlist_dir`. Liberty NCX splits all of the subcircuit descriptions in the file into separate netlists, each named `cellname.netlist_suffix`, in a subdirectory called “netlists” in the directory specified by `work_dir`. The subcircuit descriptions created in this manner cannot be used hierarchically within other subcircuits.

The SPICE model file must contain all of the models for the circuit elements used in the cell netlists. The file name must be specified by the `model_file` setting of Liberty NCX.

If you provide a .lib source file in Liberty format as input to Liberty NCX, the template files specify the cells and timing arcs that are to be characterized. After performing characterization, Liberty NCX generates a separate .lib source file with the new characterization data. It does not overwrite or modify the original .lib source file unless you specify the same name for the output library. You specify the name of the input and output library files with the `input_library` and `output_library` settings in Liberty NCX.

Characterization Template Files

You can use template files to specify library or cell characterization parameters. The parameters in the template files override those in any input library.

A library-level template file specifies library characterization parameters such as units, delay thresholds, slew thresholds, and derating factors.

Here is an example of a library template file, “synop_lib.opt”:

```
delay_model : table_lookup ;
date : 6-DEC-2006 ;
revision : 0.000000 ;
time_unit : 1ns ;
voltage_unit : 1V ;
pulling_resistance_unit : 1kohm ;
current_unit : 1uA ;
nom_voltage : 1.260000 ;
nom_temperature : -40.000000 ;
nom_process : 1.000000 ;
input_threshold_pct_rise : 50.000000 ;
output_threshold_pct_rise : 50.000000 ;
input_threshold_pct_fall : 50.000000 ;
output_threshold_pct_fall : 50.000000 ;
slew_lower_threshold_pct_rise : 20.000000 ;
slew_upper_threshold_pct_rise : 80.000000 ;
slew_lower_threshold_pct_fall : 20.000000 ;
slew_upper_threshold_pct_fall : 80.000000 ;
slew_derate_from_library : 1.000000 ;
global_vdd : VDD ;
global_vss : VSS ;
driver_model : snps_pedriver ;
active_drv_netlist : /remote/dept5116d/nanda/P4/
test/ncx/db/test1/netlist/invx4.spc ;
active_drv_input : A ;
active_drv_output : Y ;
active_drv_supply_node : VDDC ;
active_drv_supply_voltage : 1.2 ;
active_drv_unate : negative ;
active_drv_slew : 2e-9 ;
do {
    INvx4
}
```

A cell-level template file specifies cell characterization parameters such as scaling factors, area, and sensitization.

Here is an example of a cell template file, “nand2x2.opt”:

```
scaling_factors : NAND2X2_factors ;
area : 2.822000 ;
cell_footprint : nand2 ;
sensitization {
    A, B : Y ;
    01, 0 : 1 ;
    0, 01 : 1 ;
    10, 0 : 1 ;
    1, 01 : f ;
    0, 10 : 1 ;
    01, 1 : f ;
    1, 10 : r ;
    10, 1 : r ;
}
```

To create a set of sample template files that you can edit, set the `templates` option to `true` when you run Liberty NCX. This generates template files for the existing source library and all cells in that library, with the characterization parameters set to the same values found in the source library. The template files are written to the working directory specified by the `work_dir` setting. The default extension for template file names is `.opt`. To generate files with a different extension, use the `template_suffix` setting of Liberty NCX.

You can then copy and edit the template files to override the parameters for the new library created in the next Liberty NCX run. To generate a new cell, copy and edit one of the existing cell templates. Place the edited template files in a directory and specify that directory with the `input_template_dir` setting of Liberty NCX.

For more information, see [Chapter 3, “Template Files.”](#)

Previewing Generated Template Files

You can review and modify the templates before performing characterization by setting the `prechar` configuration variable to `true` in the configuration file. When you do this, Liberty NCX writes the information from your seed library to the template files and stops after the optimization step is complete. The `prechar` variable is set to `false` by default.

You can save the preanalysis optimization information and use it for performing characterization on other corners. If the template files do not include optimization information and the `preanalysis_opt` configuration setting is `auto` (the default setting), Liberty NCX generates and saves new optimization information in the template files. If the optimization information already exists, Liberty NCX does not rerun the optimization; it reuses the information in the template files.

If you want Liberty NCX to rerun optimizations whether or not optimization information is included in the template files, set the `preanalysis_opt` option to `force`. If you want Liberty NCX to ignore the existing optimization information during characterization, set the `preanalysis_opt` variable to `false`.

To save pre-analysis optimization information, first create optimizations for a corner. Then enter the following in the configuration file:

```
set prechar true  
set preanalysis_opt auto /* the default setting */  
set templates true  
set timing|power true  
set output_template_dir ./output_template
```

Liberty NCX generates the templates and then saves the optimization information in the templates inside the `ncx_optimization` group. To use the optimization information for performing characterization on other corners, set the following variable in the configuration file:

```
set input_template_dir ./output_template
```

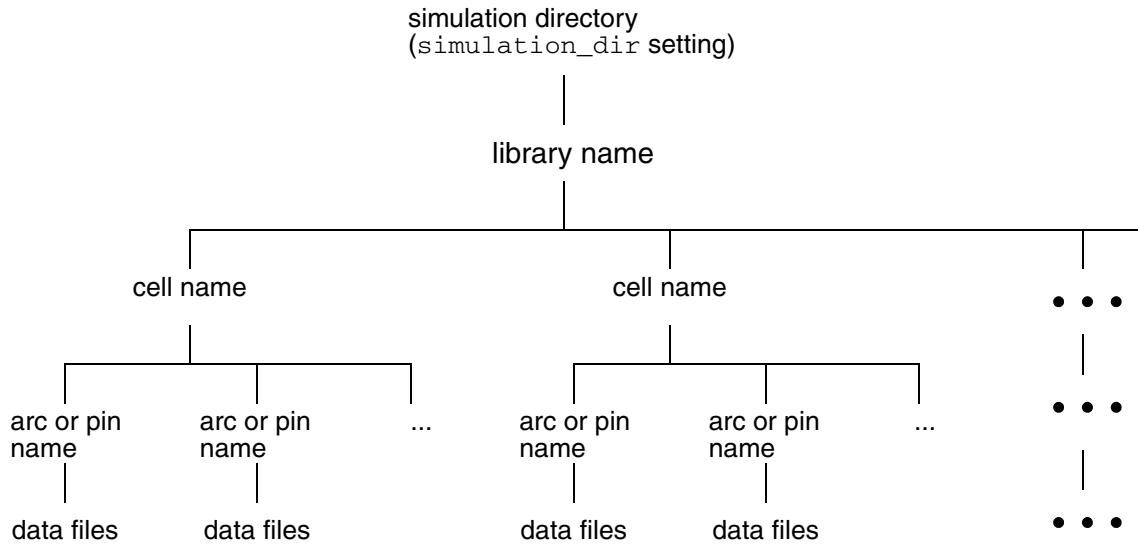
Characterization Output Data

The result of characterization is a new `.lib` library file in Liberty format. You can specify the name of the generated file with the `output_library` configuration variable.

If you set the `output_library_compression` configuration variable to `gzip`, the output library is compressed with gzip to reduce disk space usage. Such compressed libraries can be directly read in and compiled with Library Compiler.

Liberty NCX uses the HSPICE simulator to simulate the timing arcs specified in the source `.lib` file and template files. The SPICE circuit files, stimulus files, and results files are stored in a directory structure like the one shown in [Figure 2-1 on page 2-6](#).

Figure 2-1 Directory Structure for Simulation Data Files



If you set the `cleanup` option to 0, Liberty NCX retains the simulation files so that you can examine them to find out how characterization was performed or to debug any problems found with the generated library. A higher cleanup setting (1, 2, or 3) removes more of the simulation data files.

The simulation database directories contain the simulation data. This information is used to extract the cell and arc model characteristics. The database is created in the current working directory by default, or in the directory specified by the `simulation_dir` setting in Liberty NCX.

The database root name matches that of the library being created. The library directory contains the cell directories, which are named after the cells in the library. Each cell directory contains arc and pin directories, which contain the simulation data for individual timing arcs and other characterization data.

The arc and pin directories are named according to the type of data being characterized. For example, a cell directory `ACHCINX2` might contain a directory named `tCO_A_002f`. This directory name represents a timing arc from pin `A` to pin `CO` of cell `ACHINX2`, which is set up to simulate a falling transition on the primary pin `CO`. In directory `tCO_A_002f`, you can find the simulation input and simulation output data files for the timing arc or other characterized circuit.

In general, the arc and pin subdirectory names are constructed according to the following conventions:

- The first character signifies the type of acquisition: `t` for timing (which includes capacitance and active power), `p` for unpropagated arc power, `c` for capacitance, `v` for violation, or `d` for DC leakage power.
- The characters that follow specify the name of the associated primary pin, which is either the output pin of a delay arc or an input pin whose capacitance is being acquired.
- After that, the characters that follow, if present, specify the name of the related pin of a timing arc or constraint arc.
- After that are some number characters corresponding to the order of appearance in the library database.
- The last character specifies the type of transition associated with the primary pin, either `f` for a falling transition or `r` for a rising transition.

Liberty NCX Operation and Log File

[Figure 2-2 on page 2-8](#) and [Figure 2-3 on page 2-9](#) show the steps performed by Liberty NCX during characterization.

Liberty NCX first reads in the existing input library, if provided, and the template files, and puts the library information into its internal database. Then it sensitizes the cell arcs (determines the conditions and input transitions) for characterization and builds the netlists for simulation. It runs each simulation on the host machine or submits each simulation for execution on a compute farm.

From the results of each simulation run, Liberty NCX extracts the library models for each cell and each arc simulated. After checking the model data, it writes the output library in .lib format. It optionally writes out new template files that can be modified and used in the next characterization run.

When Liberty NCX performs these operations, it writes out a log file showing the configuration options, input files used, output files produced, and the status of each operation, including any error and warning messages. The log file can be helpful in debugging any problems reported during the characterization process.

Figure 2-2 Liberty NCX Operation

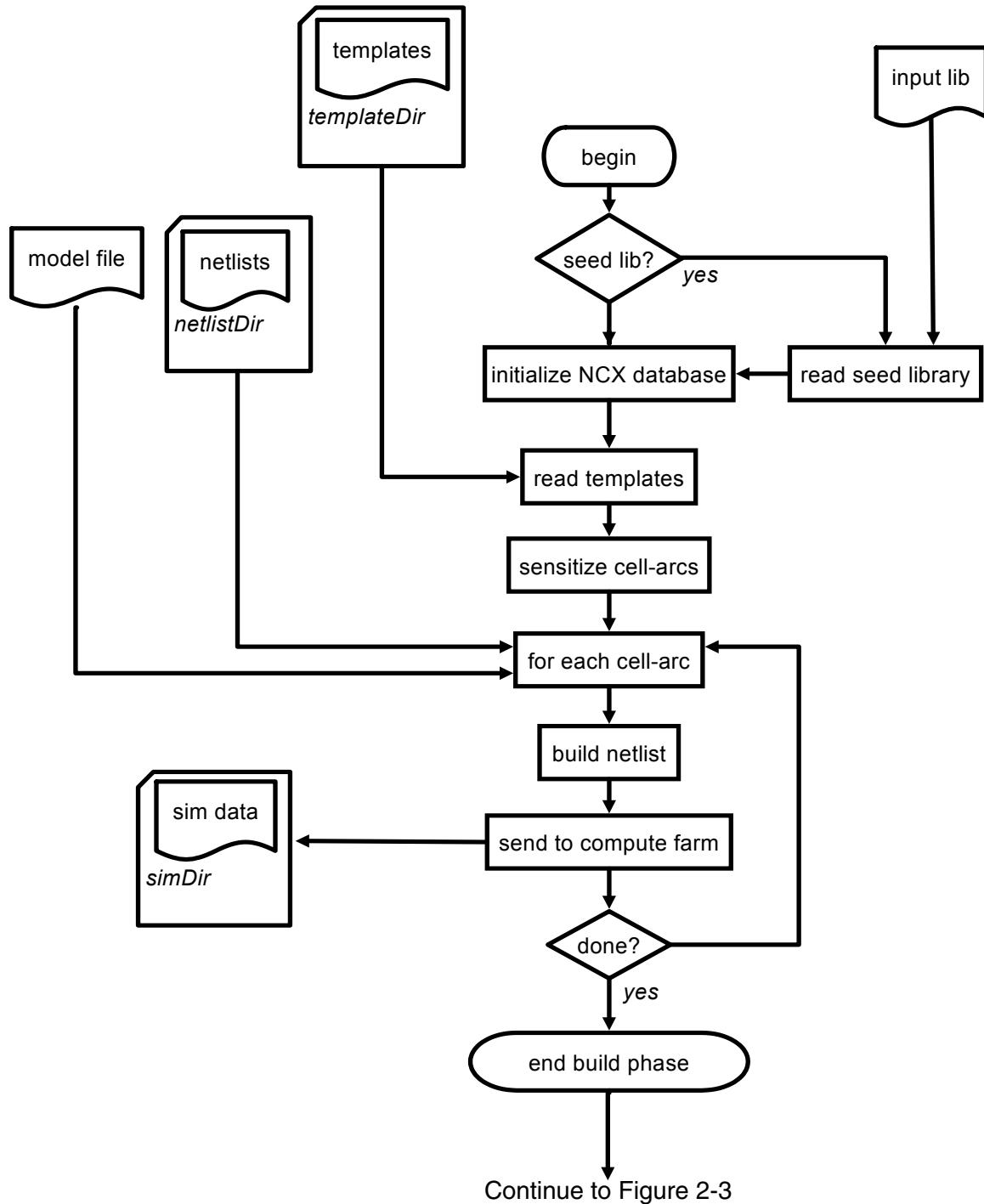
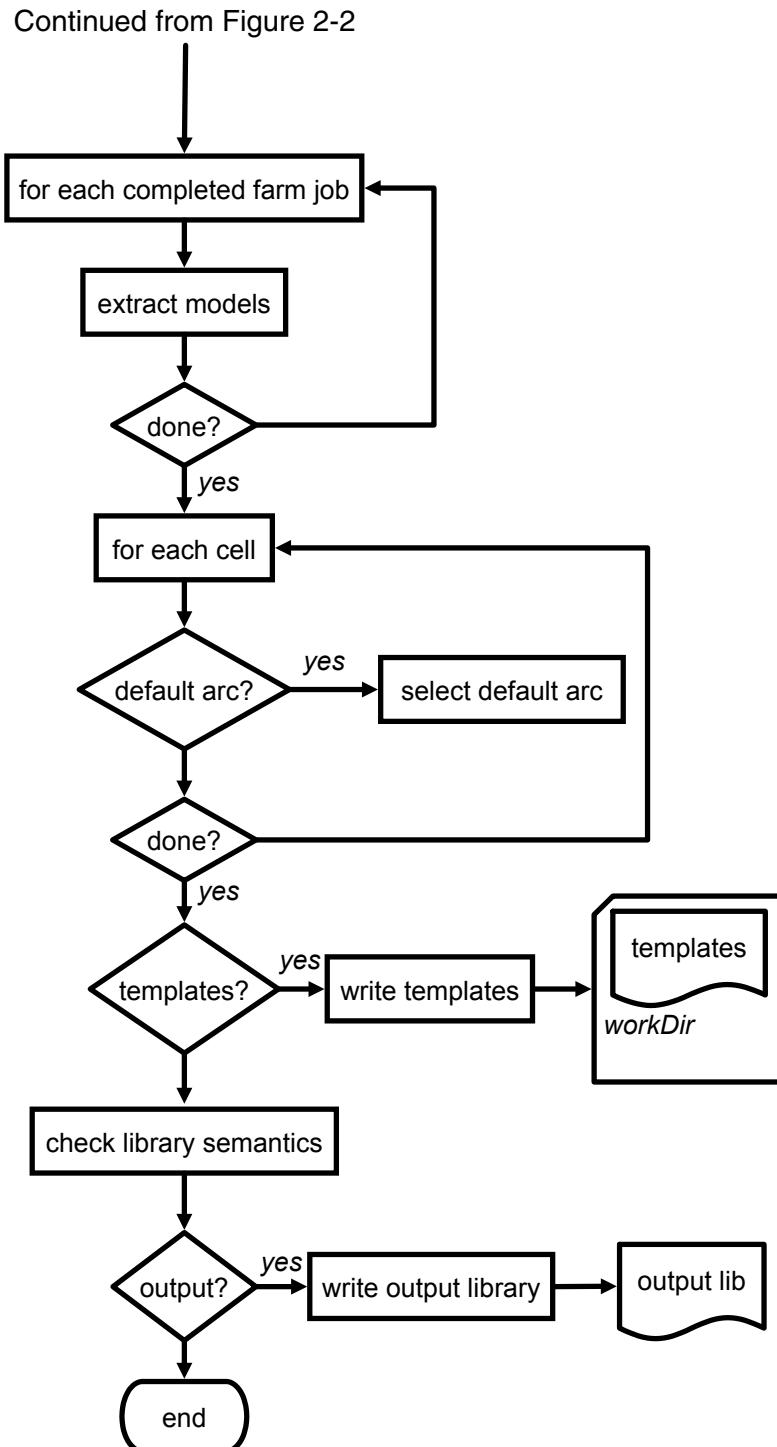


Figure 2-3 Liberty NCX Operation (Continued)

By default, the log file is written to the current working directory and is named ncx.log. You can specify a different file name with the `log_file` setting in Liberty NCX.

The log file contains the following sections:

- Program initialization
- AutoFunction recognition
- Cell sensitization
- Characterization netlist creation
- Model extraction
- Program summary

Program Initialization Section

The program initialization section is the first section of the log file. It shows the Liberty NCX program release number and build date, the program invocation command (including all configuration file settings and command-line arguments), and the settings of all possible options, including default settings not specified by the user. This section reports the program configuration and startup settings.

The following is an example of a program initialization section:

```
Liberty NCX (TM) F-2011.06
... (copyright notice) ...

...
Program configuration:
Files/folders:
    output_library      : ncx.lib
    log_file            : /remote/techp5/gra/p1/ncx.log
    work_dir            : /remote/techp5/gra/p1/work
Flows:
    preanalysis_opt     : auto
    prechar             : false (user)
    timing              : true (user)
    power               : false (user)
    noise               : false
    compact              : false (user)
    templates            : true (user)
    ibis                : false
    auto_function        : true (user)
    soi                 : false
    soi_mode             : false
```

```
Simulation settings:  
  simulator_exec      : /global/apps/hspice/bin/hspice (user)  
  simulator_type       : hspice  
  ...  
Compute farm:  
  farm_type           : LSF  
  queue_name          : normal  
  ...  
Output format control:  
  ...  
Program control:  
  ...  
Timing model acquisition options:  
  ...  
Template usage options:  
  ...  
...  
--- begin flow...  
...
```

AutoFunction Section

The AutoFunction section reports the names of the cells whose functions are being automatically derived from the SPICE circuit topology. For example,

```
--- Running AutoFunction ...  
Running AutoFunction on cesom8x [1/6]  
  
Running AutoFunction on cepom8x [2/6]  
...  
done
```

For information about the AutoFunction capability, see “[Automatic Function Recognition](#)” on [page 2-16](#).

Cell Sensitization Section

After the input library (if specified) and template files (if present) have been read into the program database, Liberty NCX sensitizes each cell. In other words, it generates the stimulus at the cell input pins necessary to produce a simulation measurement of the desired characteristic, such as delay or slew.

No simulation is performed at this stage. Liberty NCX analytically derives the functionality of the cell from Boolean expressions, truth tables, state tables, flip-flop groups, and latch groups defined in the input library or template files. You can also specify the cell sensitization explicitly in the template files, as explained in the section called “[Sensitization](#)” on [page 3-49](#).

The following is an example of a cell sensitization section in a log file:

```
--- generating sensitizations...

cell ncx::lagcesom8x (1 of 3) (3in/2out/1int pins)
    user-specified clock pin CK
processing state N_9_M30_S...
    detected active L clock CK
    detected low-level-sensitive clock-gating stage
creating delay arcs...
    combinational (positive_unate) CK -> GCK
    combinational (positive_unate) E -> OBS
2 delay arcs created

creating constraint arcs...
    min_pulse_width CK -> CK
    setup_rising CK -> E
    hold_rising CK -> E
    setup_rising CK -> SE
    hold_rising CK -> SE
5 constraint arcs created
...
--- checking indices and signal levels...

--- pre-analyzing cells...

--- checking pre-analysis simulations...

--- processing pre-analysis results...
...
```

Any cell arcs that cannot be sensitized are marked with warning lines.

Arc Netlist Creation Section

After sensitization is complete, Liberty NCX proceeds to build the simulation database by performing the following steps:

- It creates a simulation subdirectory for each active cell arc. The directory has a simulation netlist and a batch executable file for the compute farm.
- It acquires any prerequisite characteristics, for example, three-state cell output pin capacitance. It gets this information by running a quick simulation on the local machine.
- It submits the simulation job to the compute farm, if used. If the simulations are to be run on the local machine, they are run in the next stage, the model extraction stage.

The following is an example of an arc netlist creation section in a log file:

```
--- building netlists for nominal analysis ...
cell ncx::lagcesom8x (1 of 3) ()
    applying optimizations...done
...
```

No netlists are generated if CCS and NLDM model generation are both disabled (in other words, the `ccs_timing` and `nldm` options are both set to false). You might want to do this to generate Liberty NCX templates from an existing library to use in a future session.

Model Extraction Section

The model extraction phase is typically the most time-consuming stage of the characterization process because it depends on the completion of simulation jobs. As Liberty NCX completes simulations, whether on the compute farm or the local machine, it extracts the specified models and stores them in the database.

Along the way, Liberty NCX ensures that simulation netlists are continuously fed to the compute farm or local machine as resources become available.

The final stage of model extraction is the extraction of default arcs for any cells that contain them. This requires the selection of one of a set of conditional arcs as the default arc.

The following is an example of an model extraction section in a log file:

```
--- checking nominal simulation status...
Flushing all jobs...
    [35 jobs submitted]
Fri Oct 29 08:52:09 2010
checking job status... 0 nets queued, 30 jobs running, 5 pending, 0 done.
checking job status in 30 seconds at 8:52:39

Fri Oct 29 08:52:40 2010
checking job status... 0 nets queued, 30 jobs running, 5 pending, 0 done.
checking job status in 30 seconds at 8:53:10
...
```

Program Summary Section

The program summary is the last section of the log file. This section should be checked at the conclusion of every Liberty NCX run. It lists any failures encountered during the process and shows relevant statistics such as the total runtime.

The following is an example of a program summary section in a log file:

```
writing
/remote/techp5/gra/p1/work/SLOW_125_0P9_NCX.opt...done
total template write time: 0.004 seconds (elapsed)
...
deleting sensitization templates...done
...
Checking semantics...1...2...done
...
writing SLOW_125_0P9_NCX to ncx.lib...done
...
done

total program time      : 309.960 seconds (5.17 min.)
```

Partial Sensitization Flow

Liberty NCX supports a partial sensitization flow, which allows you to augment the tool's automatic sensitization ability and enables full cell characterization. The partial sensitization flow is useful in supporting complex cells where automatic sensitization fails to either create or sensitize arcs. Errors encountered during the automatic sensitization process are handled by Liberty NCX to distinguish between a fatal error and an error that allows creation of templates with partial sensitization information, thereby enabling a partial sensitization flow. Messages are generated to provide information where manual intervention is necessary.

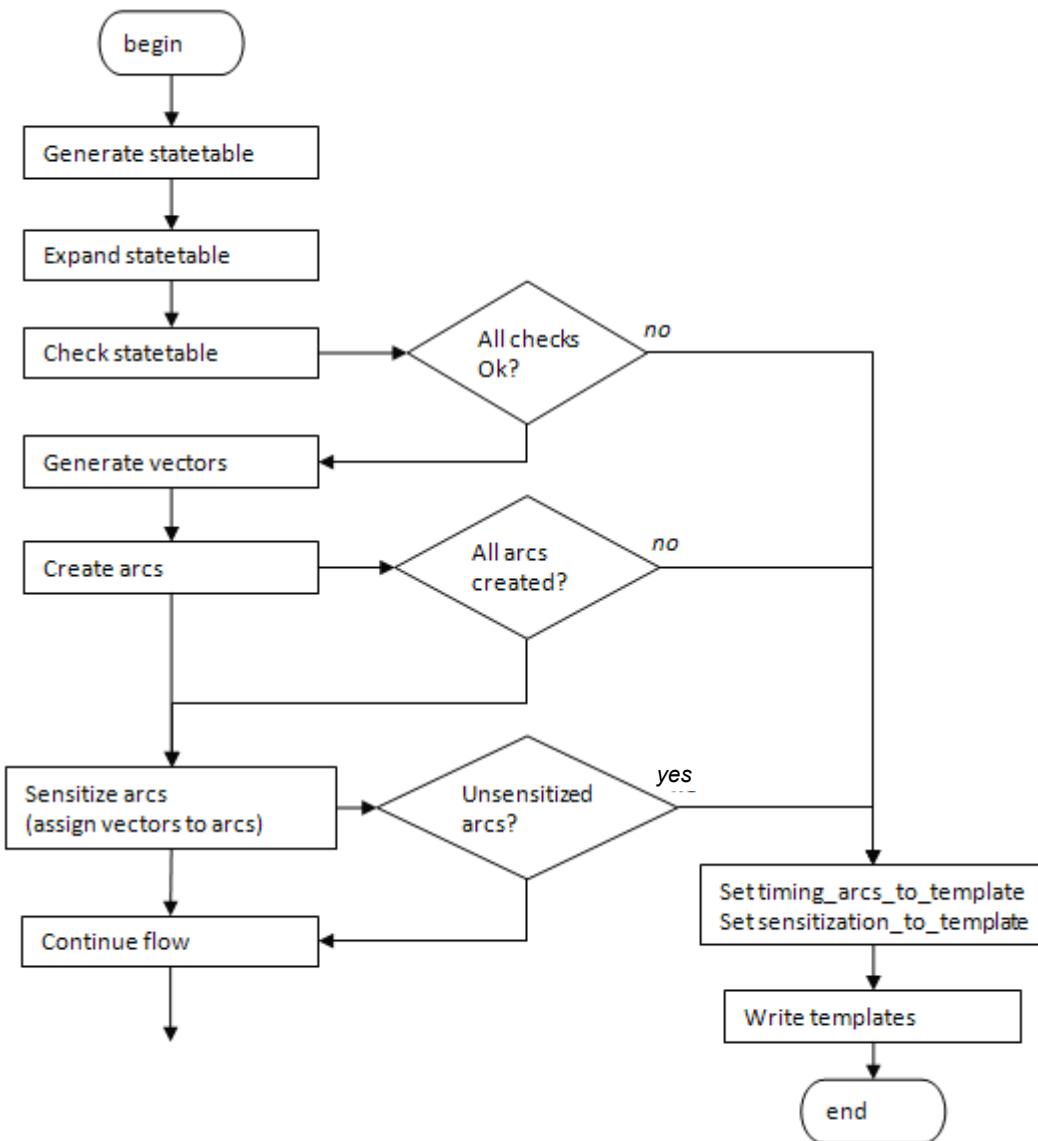
Partial sensitization flow has the following features:

- Partial arc sets are created in the autosensitization flow with explicit identification of any errors encountered.
- A Liberty NCX session may be terminated in the presence of nonsensitized arcs with explicit identification of those arcs in the log file, and templates are generated with all arcs and possible sensitizations.
- Precise and early identification of errors encountered in the sensitization phase are provided to enable better debugging or repair.

Liberty NCX terminates after the sensitization phase if it encounters an error. It then automatically triggers the generation of templates with all arc and sensitization information. This information explicitly describes the manual intervention that is required for each particular arc. For example, it describes the required intervention for an arc that cannot be sensitized or an arc that has an incorrect cell descriptor specified.

The log file lists all unsensitized arcs, and the cell template files contain arcs commented with comments noting that sensitization does not exist. The flow diagram in [Figure 2-4 on page 2-15](#) is a high-level illustration of the sensitization flow, describing where breakpoints are appropriate to enable manual intervention and debugging.

Figure 2-4 Sensitization Flow



The following example shows log messages and associated template content that might be generated.

The first excerpt is from the end of the log file.

```
Failed cells [1/5] (4 arcs):
AN02D1
    positive_unate arc 0: A1 -> Z () (model extraction) tZ_A1_0000r
    positive_unate arc 0: A1 -> Z () (model extraction) tZ_A1_0000f
    positive_unate arc 1: A2 -> Z () (model extraction) tZ_A2_0001r
    positive_unate arc 1: A2 -> Z () (model extraction) tZ_A2_0001f
```

The second excerpt is the corresponding template file and would contain the following comments:

```
pin Z {
    direction : output ;
    max_transition : 6.0000000 ;
    function : A2+A1 ;
    ncx_internal_power_rise_input_transition_time_index : 0 ;
    ncx_internal_power_fall_input_transition_time_index : 0 ;
    ncx_internal_power_rise_total_output_net_capacitance_index : 0 ;
    ncx_internal_power_fall_total_output_net_capacitance_index : 0 ;
    timing {
        *** arc id: Z_A1_0000
        ****
        *** failed model extraction
        ****
        related_pin : A1 ;
        timing_sense : positive_unate ;
        ncx_rise_input_net_transition_index : 0 ;
        ncx_fall_input_net_transition_index : 0 ;
        ncx_rise_total_output_net_capacitance_index : 0 ;
        ncx_fall_total_output_net_capacitance_index : 0 ;
        ncx_wave_rise : A1 01 \
            A2 0 \
            Z 01 ;
        ncx_wave_fall : A1 10 \
            A2 0 \
            Z 10 ;
    }
}
```

Automatic Function Recognition

Liberty NCX supports automatic recognition of cell and pin functions by topological analysis of the SPICE netlist. This feature is called AutoFunction. When the AutoFunction feature is enabled, you are not required to specify functional information for certain types of cells such as combinational logic cells, 1-bit flip-flops and latches, and 1-bit sequential integrated clock-gating cells. The tool automatically determines the cell and pin functions, sensitizes the cell inputs for characterization, and writes out a complete model in Liberty syntax.

When you use the AutoFunction feature, you can proceed directly with full timing, noise and power characterization using the automatically generated functional information.

Alternatively, you can choose to write out the generated functional information in a set of template files by setting the `templates` and `prechar` attributes to `true` in the configuration file. In that case, you can examine, and optionally modify, the templates before they are used in a subsequent characterization run.

To enable automatic function recognition, set the `auto_function` attribute to either `auto` or `true` in the configuration file. The attribute settings are defined as follows:

- `false` – Disables the AutoFunction recognition capability. This is the default behavior.
- `auto` – Enables the AutoFunction recognition capability only for cells that are missing functional definitions, such as flip-flop groups, latch groups, state table statements, and Boolean expressions in the seed library or template file. Unsupported cells are reported as “Failed” cells only if a functional definition has not been provided in the seed library or cell template.
- `true` – Enables the AutoFunction recognition capability for all supported cell types. The AutoFunction representation takes precedence over any cell function provided in the seed library or template file. For combinational cells, if the seed library or template file function differs from the AutoFunction representation, the difference is reported as a warning. Unsupported cell types are always reported as “Failed” cells.

The `ncx_skip_auto_function` cell template attribute, when set to `true`, causes AutoFunction analysis to be skipped for that cell, irrespective of the cell template content and the `auto_function` attribute setting.

The AutoFunction capability supports the following types of cells:

- Any combinational cell.
- Any simple 1-bit sequential cell, such as a flip-flop or latch, when the output pin functions are defined in the template file with the `ncx_auto_function_pin_pattern` attribute.
- Any two-stage sequential synchronizer cell, where both sequential elements are clocked by the same clock.
- Any clock-gating cell with a 1-bit sequential state when the cell function is identified with the `clock_gating_integrated_cell` attribute and the clock input, clock enable, and clock output pins are identified with the `ncx_auto_function_pin_pattern` attribute.

The AutoFunction feature determines the function of a combinational cell by applying all combinations of input values and looking at the output values. However, for a sequential cell, you need to provide additional information about the pin functions.

The following cell template attribute allows you to specify the name patterns for pins and thereby identify the pin functions.

```
ncx_auto_function_pin_pattern : pin_type1 pin pin pin_type2 pin;
```

You can specify multiple pin names for each pin type pattern. Liberty NCX recognizes each subsequent pin type keyword as the start of a new pin pattern. Line continuation can be used for long pin pattern specifications.

It is recommended that you use `ncx_condition` conditional characterization groups in the library template file to supply pin pattern information for cell families. You can do this by using wildcards with the `ncx_condition_cell_name` attribute to match all cells in a cell family. The `ncx_condition` precedence rules can be used to first define information for an entire cell family, then augment it with additional or different information for cell subfamilies. For more information on conditional groups, see “[Conditional Characterization](#)” on [page 3-90](#).

Flip-Flop and Latch Cell Recognition

For AutoFunction to recognize a flip-flop or latch, the cell template file must identify the following pin types, if they exist in the cell:

- `q_pin`
- `qn_pin`

If not specified, the `q_pin` and `qn_pin` pin patterns default to the values `Q` and `QN`, respectively.

Sequential cells with preset and clear pins are supported. AutoFunction explores the cell functionality to determine what preset and clear pins are present, if any.

The following example identifies any pin named `Q` or `Q1` as the D flip-flop output pin (pin type `q_pin`) and any pin named `QN1` or `QB` as the inverted data output pin (pin type `qn_pin`):

```
ncx_auto_function_pin_pattern : q_pin Q Q1 qn_pin QN1 QB;
```

The pin pattern statement can be embedded in a condition group to limit the scope, as in the following example:

```
ncx_condition {
    ncx_condition_cell_name : DFFR*;
    ncx_condition_model : cell ;
    ncx_auto_function_pin_pattern : q_pin Q Q1 qn_pin QN1 QB;
}
```

By default, AutoFunction models a sequential cell using the Liberty `ff` group or `latch` group statements. Liberty `ff` and `latch` groups provide clearer cell function definitions, and are used by default. To have AutoFunction create a `statetable` group instead, set the `ncx_auto_function_use_seq_group_attr` attribute to `false` in the cell template. The `statetable` form of syntax is more flexible than the `ff` or `latch` groups, but may not be supported by all downstream tools.

Scan Flip-Flop Cell Recognition

AutoFunction supports the recognition of 1-bit multiplexed scan D-type flip-flops used in design-for-test (DFT) applications. A scan cell can function as either a flip-flop or a scan register element.

For AutoFunction to recognize a scan flip-flop, the basic sequential cell information should be provided as described in the “[Flip-Flop and Latch Cell Recognition](#)” section. In addition, the template file must additionally identify the following pin types, if they exist in the cell:

- `test_in_pin`
- `test_en_pin`
- `test_out_pin`

The `test_in_pin` pin type identifies the pin used as the scan-in data pin and the `test_en_pin` pin type identifies the scan enable pin that enables the capture of scan data for the flip-flop. If the scan cell has a dedicated scan output pin, identify it by specifying the `test_out_pin` pin type.

The cell template should also set the attribute `ncx_auto_function_add_test_cell` to `true`. This adds the test cell functionality to the scan cell, resulting in generation of a Liberty `test_cell` group.

Scan cells with preset and clear pins are supported. AutoFunction explores the cell functionality to determine what preset and clear pins are present, if any.

The following example demonstrates the template file description for a family of simple scan flip-flops:

```
ncx_condition {
    ncx_condition_cell_name : SDFF*;
    ncx_condition_model : cell ;
    ncx_auto_function_pin_pattern : q_pin Q qn_pin QB \
                                    test_in_pin SD test_en_pin SE ;
    ncx_auto_function_add_test_cell : true ;
}
```

Two-Stage Synchronizer Cell Recognition

AutoFunction supports the recognition of two-stage flip-flop synchronizer cells. Each flip-flop stage can be clocked by a different clock edge, but both stages must be clocked by the same clock signal.

For AutoFunction to recognize a two-stage synchronizer cell, the basic sequential cell information should be provided as described in the “[Flip-Flop and Latch Cell Recognition](#)” section. In addition, the template file must additionally identify the following pin type:

- `internal_pin`

The `internal_pin` pin type specifies the intermediate internal node in the cell subcircuit netlist that represents the output of the first flip-flop. It does not represent a Liberty pin name, as pin list definitions normally do.

Synchronizer cells with preset and clear pins are supported. AutoFunction will explore the cell functionality to determine what preset and clear pins are present, if any. Scan synchronizer flip-flop cells are not supported.

Integrated Clock-Gating Cell Recognition

For AutoFunction to recognize a sequential integrated clock-gating cell, the cell template file must identify the following pin types, if they exist in the cell:

- `clk_pin`
- `en_pin`
- `q_pin`
- `test_en`

The following example demonstrates the template file description for a sequential integrated clock-gating cell:

```
ncx_condition_ckgo {
    ncx_condition_cell_name : CKGO* ;
    ncx_condition_model : cell ;
    clock_gating_integrated_cell : latch_negedge_precontrol ;
    ncx_auto_function_pin_pattern: clk_pin CK en_pin E q_pin GCK ;
}
```

The `q_pin`, `clk_pin`, and `en_pin` pin types are required for integrated clock-gating cell recognition. They identify the sequential output pin, clock input pin, and clock enable pin, respectively. If the cell has a test enable pin, identify it by specifying the `test_en` pin type.

If the integrated clock-gating cell has a combinational output in addition to the gated clock output, it is marked with the `clock_gate_obs_pin` Liberty attribute.

The `clock_gating_integrated_cell` attribute must be set to the same string used in Liberty format to describe the functionality of the integrated clock-gating cell:

```
generic
latch_posedge
latch_posedge_precontrol
latch_posedge_postcontrol
latch_posedge_precontrol_obs
latch_posedge_postcontrol_obs
latch_negedge
latch_negedge_precontrol
latch_negedge_postcontrol
latch_negedge_precontrol_obs
latch_negedge_postcontrol_obs
none_posedge
none_posedge_control
none_posedge_control_obs
none_negedge
none_negedge_control
none_negedge_control_obs
```

For the `generic` clock-gating cell type, AutoFunction models the sequential integrated clock-gating cell with a `latch` group. For all other types, AutoFunction models the integrated clock-gating cell using a `Liberty statetable` group statement for the sequential memory node of the cell and a `Liberty state_function` attribute for the output pin.

For complete descriptions of these clock-gating cell types, see the *Library Compiler Methodology and Modeling Functionality in Technology Libraries User Guide*, available on SolvNet.

Acquisition Types

Liberty NCX can acquire a variety of behavioral models. For example, you can generate models using the Composite Current Source (CCS) format, a newer and more advanced format for accurate analysis of technologies at 90 nm and below. You can selectively choose to generate any combination of CCS timing, CCS noise, and CCS power models for timing analysis in PrimeTime, noise analysis in PrimeTime SI, or power analysis in PrimeTime PX.

You can also acquire models in the Nonlinear Delay Model (NLDM) format for timing and noise analysis and Nonlinear Power Model (NLPM) for power analysis. These are older library models based on voltage-source rather than current-source circuit elements.

If you want both CCS and NLDM models, you can optionally generate CCS models first, and then convert the library data from CCS to NLDM format using the `ccs2nldm` formatting option of the Model Adaptation System. For information on the conversion process, see [“CCS to NLDM Conversion” on page 5-8](#).

You can also convert a CCS library data to Effective Current Source Model (ECSM) format by using the `ccs2ecsm` formatting option of the Model Adaptation System. For details, see [“CCS to ECSM Conversion” on page 5-5](#).

CCS Timing

During characterization of each timing arc for CCS timing, Liberty NCX measures the input voltage, output current, and output voltage as a function of time for varying input transitions, slews, and loads. It then converts these measurements to CCS library data. The CCS models are written into the output library as delay models for each delay arc between input and output pins, and as capacitance models for each input pin.

The data representation for CCS timing has been incorporated into the Liberty open library standard. CCS builds on the comprehensive library modeling capabilities of Liberty. For more information on CCS as part of Liberty format, please refer to the CCS Liberty Format specification.

To invoke CCS timing acquisition, set the `timing` and `ccs_timing` options to `true` in the Liberty NCX configuration file. [Table 1-6 on page 1-13](#) lists and describes the options available in the configuration file for controlling the acquisition of capacitance, receiver models, delay arcs, CCS driver models, and constraint arcs. In most cases, you must also provide library and cell template files to direct the characterization process as described in [Chapter 3, “Template Files”](#).

If you provide an input library with the `input_library` setting in the configuration file, by default, Liberty NCX recharacterizes all cells in the input library. You can specify which cells are to be characterized or not characterized in the current run of Liberty NCX with the `do` and `dont` attributes in the library template file. To characterize a library incrementally, a specified number of cells at a time, see [“Incremental Characterization” on page 2-28](#).

CCS Noise

Liberty NCX can characterize library cells for noise response and create CCS Noise modeling information. A library with CCS Noise models can be used with PrimeTime SI and other tools to check for timing and logic failures caused by noise.

Liberty NCX performs CCS Noise characterization when you set the `noise` parameter to `true` in the configuration file. For details, see [Chapter 7, “CCS Noise Characterization.”](#)

If you already have two separate libraries containing CCS timing and CCS noise models, you can merge them into a single library containing both types of CCS models by using the `ccsn_merge` formatting option of the Model Adaptation System.

CCS Power

The CCS power model is a current-based power model that has been developed for advanced dynamic rail analysis. It provides a single library format suitable for a wide range of applications, including dynamic power analysis, rail voltage analysis, leakage analysis, and power optimization. It includes current waveforms for power and ground pins, and effective resistance and capacitance models for each cell’s power rails.

CCS power is a newer, more advanced power modeling technology, which can be used by PrimeRail, PrimeTime PX, and other power analysis tools. For compatibility with tools that do not support CCS power, you can use NLPM models. CCS power characterization is usually done at the same time as CCS timing characterization, and can be done as the same time as NLDM and NLPM characterization as well.

For the best possible accuracy using CCS power models, the SPICE netlists or subcircuits used for characterization should include parasitics of power-ground network in standard cells from metal-1 down to the circuit devices. Star-RCXT is the recommended tool for extraction of the SPICE subcircuit in the transistor level.

To invoke CCS power acquisition, set the `power` and `ccs_power` options to `true` in the Liberty NCX configuration file. In most cases, you must also provide library and cell template files to direct the power characterization process as described in Chapter 3, “Template Files.”

To generate CCS power models, Liberty NCX measures dynamic current, power and ground leakage current, gate leakage current, and intrinsic parasitics during simulation. It converts these measurements into CCS power library data. This model allows power analysis with a high degree of accuracy and time resolution for both single-output and multiple-output cells. The model also captures equivalent parasitics necessary to perform rail analysis. If timing and power characterization are performed at the same time, Liberty NCX uses the same input transition and output load index values for both types of characterization.

In the library and cell template files used by Liberty NCX, the attributes `ncx_internal_power_mode` and `ncx_leakage_power_mode` specify whether to exhaustively simulate all possible states for a cell (`all`) or only the first possible state (`first`, the default). Simulating all possible states might result in a more accurate model at the cost of more characterization runtime.

For more information on CCS power as part of the Liberty format, please refer to the chapter called “Composite Current Source Power Modeling” in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

Compact CCS Models

By default, generation of compact models for both timing and power are enabled in Liberty NCX. If the `compact` variable is set to `true` in the configuration file (the default setting), then Liberty NCX enables both compact CCS timing and CCS power, greatly reducing the size of the characterized library.

To control the generation of compact models for CCS Timing and Power individually, use the `compact_timing` and `compact_power` variables in the configuration file. Both these variables have a default value of `true`.

If you set `compact_power` to `false`, Liberty NCX does not reduce the size of compact CCS power libraries. In the configuration file, you can set the `compact_power` variable to `true` to enable only CCS power compaction or set it to `false` to disable CCS power compaction.

Similarly, if you set `compact_timing` to `false`, Liberty NCX disables compact CCS timing. In the configuration file, you can set the `compact_timing` variable to `true` to enable or `false` to disable only CCS timing compaction.

For more information about CCS power compaction, see “Compact CCS Power Modeling” in the “Composite Current Source Power Modeling” chapter in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

NLDM and NLPM Models

For compatibility with specific analysis tools, you might want the library to include NLDM models in addition to CCS timing models. Similarly, you might want the library cells to include NLPM models in addition to CCS power models.

To invoke acquisition of CCS and NLDM timing models in a single characterization run, set `timing` to `true`, `ccs_timing` to `true`, the `nldm` to `true`.

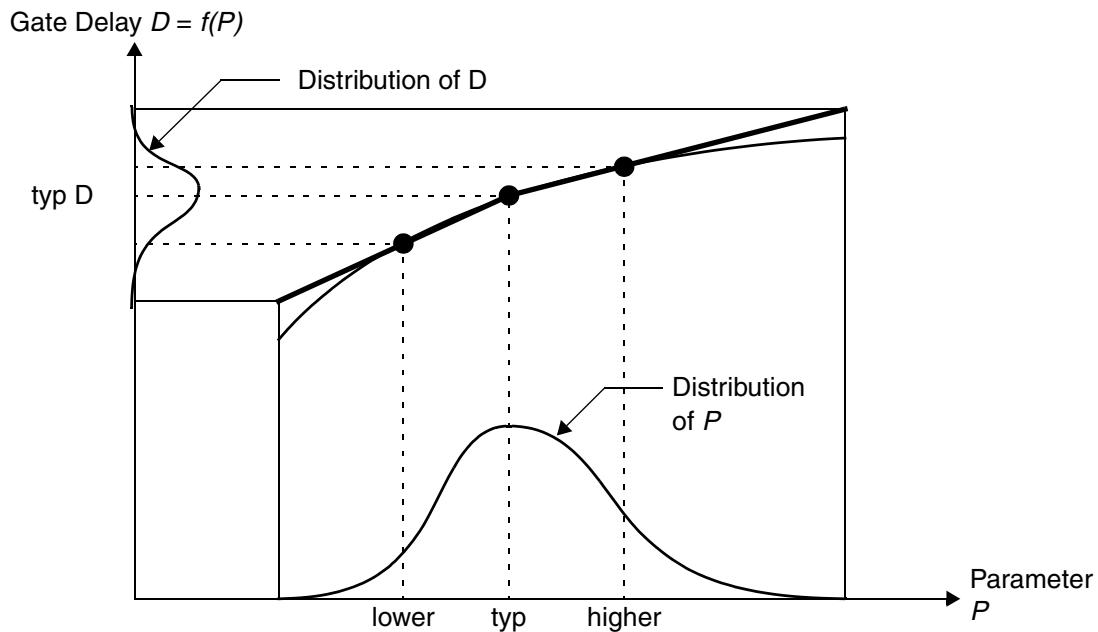
Similarly, to invoke acquisition of CCS and NLPM power models in a single characterization run, set `power` to `true`, `ccs_power` to `true`, and `nlpmp` to `true`.

Variation-Aware Models

PrimeTime VX adds variation-aware analysis capabilities to PrimeTime. A variation-aware analysis increases the accuracy of timing analysis by considering the statistical distribution of characterized parameters. Running a variation-aware timing analysis requires one or more variation-aware libraries, which can be created by Liberty NCX.

To specify the functional dependence of delay or slew on a parameter, the library cells are characterized at the typical parameter value and at two nearby values. An example is shown in [Figure 2-5](#).

Figure 2-5 Delay as a Function of Parameter P



Given the distribution of the parameter P , PrimeTime VX calculates the distribution of delays continuously throughout the range of values, using linear interpolation and extrapolation of the library-defined functional operating points.

The surrounding data points should be close enough to the typical operating point to ensure good accuracy at the middle of the distribution, where the actual parameter value is most likely to occur, but also far enough away to get good accuracy at the more extreme parameter values, where timing violations are most likely to occur.

For on-chip variation, characterization is recommended at one standard deviation (1s) away from the typical value to get the best probable average accuracy along the curve. For die-to-die variations, characterization is recommended at three standard deviations (3s) away from the typical value to get better accuracy at the tail ends of the distribution where violations are more likely to occur.

To generate the libraries that are to be used for variation-aware analysis in PrimeTime VX, you can generate a single CCS-based library that combines the nominal characterization data with the characterization data at each individual parameter value higher or lower than the nominal value. This type of library is called a merged library because it contains variation-aware characterization data at multiple sets of parameter values. Liberty NCX employs base curve technology to minimize the size of the merged library.

To create a single variation-aware CCS library at the same time as characterization (without using `va_merge`), you need to specify the parameter names, nominal values, and offset values in the template file. When you invoke Liberty NCX, set the `variation` option to `true`. Liberty NCX then performs characterization at the specified parameter data points and merges all the characterization data into a single library.

These are the library template attributes that specify the variation-aware parameters and values:

- `va_parameters`: A list of variation parameter names.
- `nominal_va_values`: A list of absolute nominal values of the parameters, in order corresponding to `va_parameters`. These are the values that represent the nominal conditions.
- `va_variation_values`: A list of variation offset values, in order corresponding to `va_parameters`. These values are added to and subtracted from the nominal values to generate the off-nominal conditions. The cells are characterized with each parameter's nominal value changed by the specified amount while the other parameters are kept at their nominal values.

For example, the library template file could contain the following attribute settings:

```
va_parameters : len vt ;
nominal_va_values : 100.0 0.24 ;
va_variation_values : 5.0 0.02 ;
```

In this example, there are two parameters, `len` and `vt`. Their nominal values are `len=100` and `vt=0.24`. Their corresponding off-nominal values are `len=105`, `len=95`, `vt=0.26`, and `vt=0.22`. Liberty NCX characterizes the cells with all the parameters set to their nominal values and also at each of the off-nominal values (while the remaining parameter is set to its nominal value), and combines all the characterization data into a single variation-aware CCS library.

You can also control the usage of index values in the off-nominal delay and slew tables with the following library template parameters:

```
ncx_va_input_net_transition_index : ordinal_list
ncx_va_total_output_net_capacitance_index : ordinal_list
```

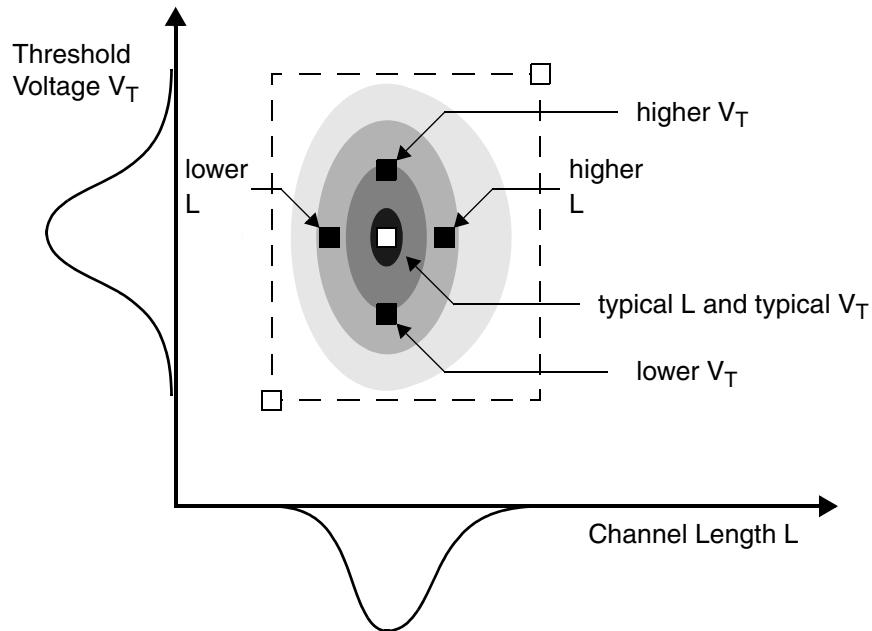
For details, see “[Variation-Aware Index Values](#)” on page [3-89](#).

Variation-aware characterization at multiple sets of parameter values requires more runtime than characterization of a single set of parameter values. Due to longer runtimes, you might consider running the characterization incrementally as described in the section, “[Incremental Characterization](#).”

An alternative to generating a single merged library is to generate a nominal library and one additional library for each individual parameter value higher or lower than the nominal value, resulting in a total of $2N+1$ separate libraries, where N is the number of parameters. This method is not as convenient to use in PrimeTime VX because you must keep track of the individual libraries and specify the parameter values for each respective library in PrimeTime VX.

To use a set of $2N+1$ separate libraries, you must create a separate library for each of the desired parameter characterization points. For example, for variation-aware analysis with two parameters, you would characterize the timing behavior at five operating points: the nominal data point, plus two surrounding data points for the first parameter (with the second parameter at its nominal value), plus two more surrounding data points for the second parameter (with the first parameter at its nominal value). For two parameters “len” and “vt,” you would characterize the behavior at the five data points shown in [Figure 2-6 on page 2-28](#).

Figure 2-6 Five Characterization Points for Two Parameters



You can later combine a set of such libraries into a single merged variation-aware library by using the `va_merge` library formatting option of the Model Adaptation System. For details, see “[Variation-Aware Library Merging](#)” on page 5-3.

Incremental Characterization

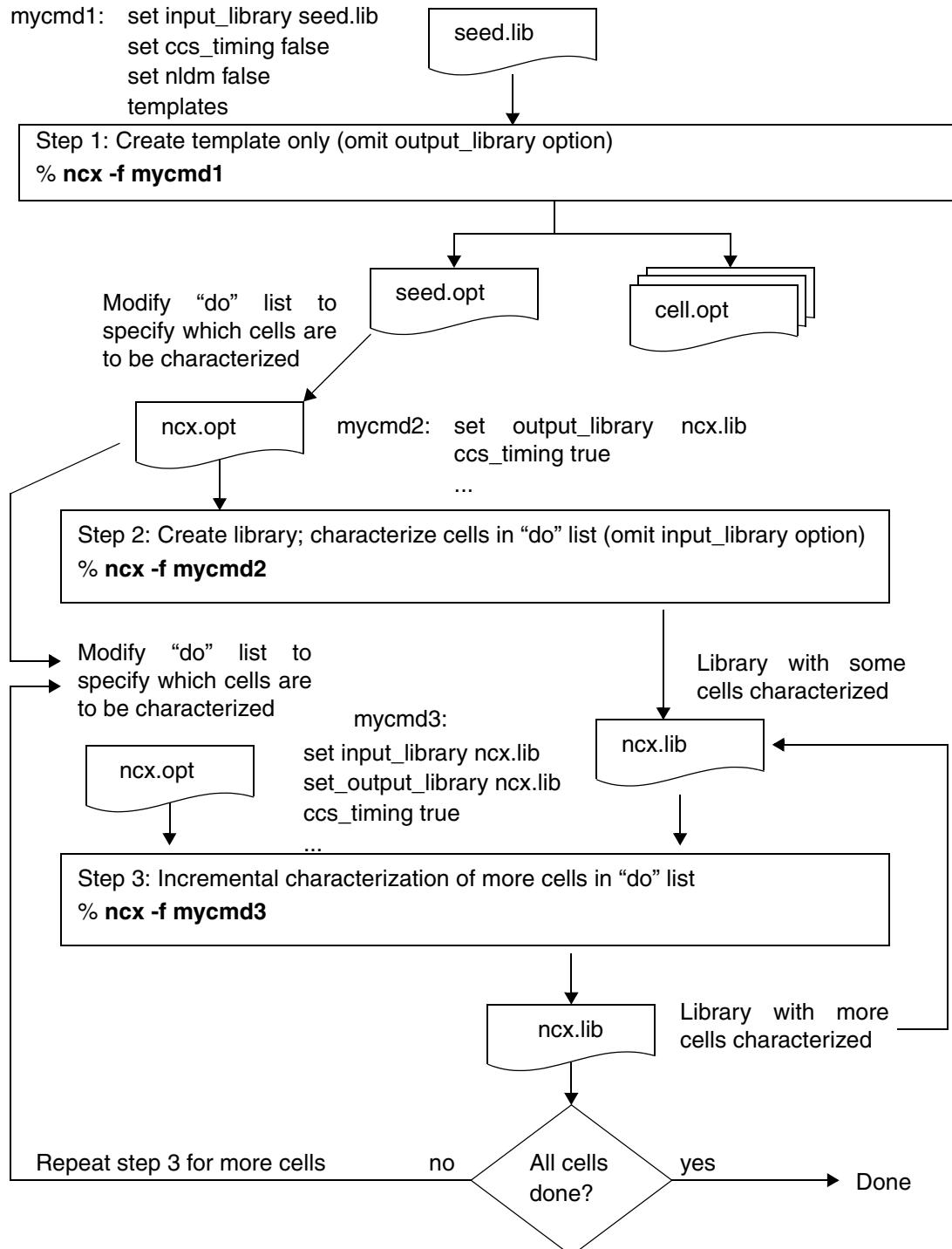
When you need to characterize a large library such as a variation-aware CCS library or a library with a very large number of cells, you can perform the task incrementally, one cell at a time or a few cells at a time. To perform characterization incrementally, you edit the `do` and `dont` list members in the library template file to explicitly specify the cells that are to be included or not included in the current characterization session.

Breaking up one large characterization task into a number of smaller tasks makes it easier to control computer resource usage and to recover from errors. You can examine and use a partially characterized library before you commit the time and resources for full characterization of the library. You can also use incremental characterization to complete the characterization of a library that has already succeeded for some cells, but failed for others.

Incremental characterization consists of three basic steps:

1. Use Liberty NCX with the `templates` option set to `true` to create the library and cell templates for the library if they do not already exist.
2. Modify the `do` and `dont` lists in the library template to specify the cells to be characterized in the first iteration. Run Liberty NCX to generate the partial library.
3. Modify the `do` and `dont` lists in the library template to specify additional cells to be characterized in the next iteration. Run Liberty NCX again to characterize the additional cells and add them to the library.

At each stage of the process, you can examine or test the partially generated library. You can repeat step 3 until the whole library is fully characterized. The steps in the iterative characterization process are summarized in [Figure 2-7 on page 2-30](#).

Figure 2-7 Incremental Characterization Flow

Template Creation for Incremental Characterization

To prepare for incremental characterization, you need to generate the library and cell templates if they do not already exist. You can do this easily by using the `templates` option in the configuration file. Use `set ccs_timing false` and `set nldm false` to prevent characterization from being performed and omit the `output_library` option so that no library is generated. For example,

```
mycmd1 file contents:  
set input_library seed.lib  
set ccs_timing false  
set nldm false  
templates  
...  
  
% ncx -f mycmd1
```

Using the input seed library `seed.lib`, Liberty NCX creates a set of template files in the working directory, where the library template file is named `seed.opt`. Rename this file, choosing a name such as `ncx.opt` if you want the final library to be named `ncx.lib`. If you are creating a variation-aware library, add the variation-aware parameters to the file. Rename the working directory using the desired template directory name so that you can use the edited file as input to Liberty NCX for initial library creation.

Library Creation

Modify the “do” list in the new library template to select a small, initial set of cells to characterize. Then run Liberty NCX to create the initial library having only those cells, using the cell templates created earlier. Omit the `input_library` option to create an entirely new library. For example,

```
mycmd2 file contents:  
set output_library ncx.lib  
set ccs_timing true  
templates  
...  
  
% ncx -f mycmd2
```

If you are creating a variation-aware library, be sure to set the `variation` option to `true`. Specify the output library file name using the same base name as the library template, for example, `ncx.lib`.

Liberty NCX generates a new library containing only the cells in the “do” list. Check for errors during library creation. You can examine and test the generated library for errors using Liberty NCX library validation tools. If the library works as expected, you can then proceed to incrementally add more characterized cells to the library.

Incremental Characterization of More Cells

When you are ready to add more cells to the library, modify the “do” list in the library template file to remove cells already characterized and specify the additional cells to characterize in the next run. Use the `input_library` option to specify the name of the library created in the previous run. The output library name specified with the `output_library` option can be the same as the input library name. For example,

```
mycmd3 file contents:  
set input_library ncx.lib  
set output_library ncx.lib  
set ccs_timing true  
...  
% ncx -f mycmd3
```

If you are creating a variation-aware library, be sure to set the `variation` option to `true`.

Liberty NCX takes the input library, characterizes the cells in the “do” list in the template, adds those cells to the library, and writes out the new, larger library. You should again check for errors during characterization and test the newly generated library for correct operation.

You can repeat incremental characterization as many times as needed until the whole library is characterized.

3

Template Files

You can create or edit template files to provide information about the library and cells being characterized. This chapter describes the template files in the following sections:

- [Template File Overview](#)
- [Characterization Attributes](#)
- [Sensitization](#)
- [Arc Generation Control](#)
- [Characterization Index Values](#)
- [Conditional Characterization](#)
- [Timing Margins](#)
- [Complex Cell Features](#)
- [Power and Ground Pin Connections](#)
- [Low-Power Modeling](#)
- [Transient Leakage Characterization](#)
- [Constraint Methodologies](#)
- [User-Defined Attributes](#)

Template File Overview

You can customize the characterization parameters by creating or modifying the Liberty NCX template files. You can also use template files to specify the cells and models that are required to be in the library. Template files are generally optional. If you do not use them, the default parameter settings and the values obtained from the seed input library are used for characterization. The template directory may contain no more than one library template file and no more than one template file for each cell.

If you are generating a new library “from scratch,” without an input library, you must provide the template files. There must be one library template file and one cell template file for each cell. The `output_library` setting determines the name of the new library produced and `input_template_dir` must be set to the name of the template directory. If the base name of the template file is different from that of the output library, you specify the template file name with the `library_template_file` setting. The default file name extension for template files is “.opt”.

If you are starting with an existing “seed” library and producing a new library, the `input_library` setting determines the name of the input library, and `input_template_dir` must be set to the name of the template directory being used. If you want the new library to be written to a new file, use `output_library` to specify its name.

If you want to generate new template files that can be edited and used as source templates in a future session, set the `prechar` and `templates` configuration variables to `true`. The template files are written to the directory specified by `output_template_dir`, and no timing, power, or noise characterization is performed.

The template file format closely follows that of the Liberty syntax. Each line of the template file defines a characterization attribute and the attribute’s value, or a group and a list of group members:

- A simple attribute is specified by the name of the attribute followed by a separating colon, the value of the attribute, and a terminating semicolon.
- A complex attribute is specified by the name of the attribute followed by a separating colon, the multiple values of the complex attribute separated by spaces, and a terminating semicolon.
- Spaces are optional before the name of the attribute, around the separating colon, and before the terminating semicolon.
- An attribute value which contains spaces must be enclosed in double quotes.
- Continuation lines are allowed with the use of a backslash (\).
- A group is specified by the type of the group, the name of the group if applicable, an opening brace, the applicable values or attributes of the group, and a closing brace.

Parameter names are the same as those used in Liberty syntax, wherever possible, unless the name is user-specific. Reserved names that are not part of the Liberty syntax are specified later in this chapter for each template type.

Attributes whose names begin with `ncx_` are intended to be used only by NCX during the characterization process and are not written to the output library file.

You can include an external file within a template file by using the `include` statement, as in the following example:

```
include my_include_file ;
```

The file name can be an absolute name or a name relative to the location of the template file in which the `include` statement is used.

Characterization Attributes

Some of the characterization attributes that you can control in template files can be used only in the library template, others can be used only in cell template files, and some can be used in both. [Table 3-1](#) describes the attributes that are common to both library and cell template files, [Table 3-2 on page 3-30](#) describes the attributes that can be used only in library template files, and [Table 3-3 on page 3-39](#) describes the attributes that can be used only in cell template files.

Table 3-1 Attributes in Library and Cell Template Files

Attribute	Description	Default
<code>ncx_output_threshold_pct_fall</code>	Specifies new delay falling trip point for recalculating NLDM delay value without simulation.	
<code>ncx_output_threshold_pct_rise</code>	Specifies new delay rising trip point for recalculating NLDM delay value without simulation.	
<code>active_drv_pwl</code>	Specifies whether the active driver should be buffered from the cell being characterized with a PWL voltage source. When set to <code>true</code> , the active driver response is applied as a voltage source PWL waveform. When set to <code>false</code> , the active driver directly drives the cell being characterized.	<code>true</code>

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
capacitance_lower_threshold_pct_fall	Specifies the lower threshold of a falling voltage waveform, expressed as a percentage of the supply voltage, for the calculation of NLDM input pin capacitance. By default, this occurs at the end time of the measurement transition.	
capacitance_lower_threshold_pct_rise	Specifies the lower threshold of a rising voltage waveform, expressed as a percentage of the supply voltage, for the calculation of NLDM input pin capacitance. By default, this occurs at the start time of the measurement transition.	
capacitance_upper_threshold_pct_fall	Specifies the upper threshold of a falling voltage waveform, expressed as a percentage of the supply voltage, for the calculation of NLDM input pin capacitance.	
capacitance_upper_threshold_pct_rise	Specifies the upper threshold of a rising voltage waveform, expressed as a percentage of the supply voltage, for the calculation of NLDM input pin capacitance.	
ccs_delay_tol	Specifies the acceptable difference between measured delay from simulation and delay obtained from the CCS waveform, expressed as a fraction of the measured delay from simulation.	0.05
ccs_margin_tol	Specifies voltage tolerance used to determine whether a signal has reached the rail voltage.	0.005
ccs_segment_tol	Specifies maximum allowed voltage difference between the simulation waveform and the CCS waveform, used for selecting the CCS model point.	0.005
clk_tran	Specifies clock transition time, in seconds, used for generating the input stimulus for a cell being characterized. This also represents the transition time for input signal edges.	10e-12
clk_width	Specifies clock width, in seconds, used for generating the input stimulus for a cell being characterized. This represents the minimum time between input signal toggles. This attribute might need to be set to a larger number in the case of failing constraint acquisitions for a cell, at the cost of more runtime.	1e-9

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
constraint_monitor_node	<p>Identifies an internal node in the cell .subckt for monitoring during constraint arc acquisition. Load capacitance is not attached to the node. To use constraint_monitor_node, you must:</p> <ol style="list-style-type: none"> 1) Specify constraint_monitor_node in the template file. The node name should be an internal node located in the SPICE .subckt. For example, if the internal monitor node name is l1_tp217, specify the template file as follows: <pre data-bbox="556 677 1073 705">constraint_monitor_node : l1_tp217 ;</pre> <p>Liberty NCX uses NCX_MONITOR as an alias for the internal node to be monitored.</p> <ol style="list-style-type: none"> 2) You must add a pin group in the cell as follows: <pre data-bbox="556 825 882 973">pin NCX_MONITOR { direction : output ; internal_node : IQ; }</pre> <p>where IQ is the name of pin in the state table that captures the internal node's (l1_tp217) functionality. If the original state table does not have functionality for the internal node that is specified, a separate column called NCX_MONITOR should be created in the output section.</p>	
constraint_total_output_net_capacitance	<p>Sets output load capacitance on a pin-by-pin basis for violation modeling. The attribute accepts a value or multiple pin name pair values. Consequently, the number of values of the attribute is 1 or an even number. You can use wildcards for the pin name for a cell with many output pins.</p> <p>The following example shows the use of this attribute when 10 and 15 defined in the library in cap units and the value index indicates that the load of that pin tracks the indexes.</p> <pre data-bbox="556 1486 1225 1543">constraint_total_output_net_capacitance : Q 10 Q1 15 Q3 index</pre>	50.0 fF
constraint_total_output_net_capacitance_pct	<p>A relative load capacitance value used for violation modeling expressed as a percentage of the max_capacitance value defined in the library. The smallest allowed value is 10.0e-18.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
default_arc_mode	<p>Determines the selection of side-pin conditions to use for characterization. This includes determining which state-dependent arc is chosen as the default state-independent arc from a set of timing arcs between two pins with valid <code>when</code> conditions on the side-input pins. The selection of side-pin conditions to use for characterization depends on the <code>default_arc_mode</code> attribute, which can be set to <code>first</code>, <code>best</code>, or <code>worst</code>.</p> <p>The <code>first</code> mode selects the first state-dependent arc encountered in the pin group, whereas the <code>best</code> or <code>worst</code> mode selects the state-dependent arc with the best or worst NLDM delay, respectively.</p> <p>The <code>worst_delays</code> and <code>best_delays</code> options specify that the default arc is a composite of all state arcs so that the NLDM delay values are the best or worst of all states at each index point, and all other models (transition, capacitance, CCS, and so on) represent the corresponding models at the worst delay points.</p> <p>The <code>worst_points</code> and <code>best_points</code> options specify the delay, transition, or receiver models that are the worst or best points (independent of each other) for the respective states at each index point.</p> <p>The <code>worst_edges</code> and <code>best_edges</code> options specify that the side input pin sensitization information for the rise constraint and fall constraint can differ from each other. Specifically, the options specify the worst and best sensitization, respectively, for each type of output edge.</p> <p>If you are recharacterizing a library and the pin has only a single state-independent arc, then the behavior is also determined by <code>default_arc_mode</code> as described for a state-dependent arc. However, if the pin has state-dependent arcs in addition to a state-independent arc, the state-independent arc is just a copy of one of the state-dependent arcs. The decision about which one to copy depends on the <code>default_arc_mode</code> setting, which also can be <code>first</code>, <code>best</code>, or <code>worst</code>.</p>	first

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
default_constraint_arc_mode	<p>Determines which state-dependent arc is chosen as the default arc from a set of timing arcs between two pins, with valid when conditions on the side-input pins.</p>	first
	<p>The first option selects the first state-dependent arc encountered in the pin group, whereas the best and worst options select the state-dependent arc with the best or worst constraint values, respectively.</p>	
	<p>The <code>worst_points</code> and <code>best_points</code> options specify the constraint values that are the worst or best points (independent of each other) from all states at each index point.</p>	
default_spice_option	<p>Uses the SPICE option in the netlist if <code>true</code>. If it is <code>false</code>, it uses or no option</p>	true
fast_mode	<p>Enables performance enhancement mode for sequential cell characterization that uses saved initial conditions to minimize multicycle cell initialization at every characterization point. This mode is automatically turned off for a cell if the cell subcircuit netlist is encrypted or if the cell initialization is less than three clock widths. If Liberty NCX reports a failure for a cell, even with accurate simulator options set with <code>sim_opt</code>, you can try turning off the fast mode by setting this attribute to false. This might help characterization succeed, at the cost of more runtime.</p>	true
init_cycle	<p>Sets the number of initialization cycles required for timing measurement. This attribute can also be specified in the <code>ncx_condition</code> group to limit the initialization cycles to a specific arc.</p> <p>The <code>init_cycle</code> attribute provides support at the cell level and the arc level. When you specify the <code>init_cycle</code> attribute at the arc level, Liberty NCX overrides any cell-level specification. For example, if you specify the following conditions in the template file:</p>	delay
	<pre>ncx_condition LPR_arc { ncx_condition_related_pin : LPR ; ncx_condition_arc_type : delay ; init_cycle : 3 ; }</pre>	
	<p>then the <code>LPR</code> pin will have three initialization cycles during the simulation of all propagating arcs that start from the <code>LPR</code> pin.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
input_cap_mode	Determines the mode of calculation of the NLDM input pin capacitance on a pin: <code>average</code> , <code>min</code> , or <code>max</code> . The value is calculated from the rise or fall capacitance values obtained for the various input slew values used for timing characterization. See the descriptions of the following attributes for more information.	average
input_cap_mode_fall	Determines the calculation of the NLDM input pin capacitance on a pin for a falling waveform transition: <code>average</code> , <code>min</code> , or <code>max</code> . The value is calculated from the capacitance values obtained for the various input slew values used for timing characterization.	average
input_cap_mode_rise	Determines the calculation of the NLDM input pin capacitance on a pin for a rising waveform transition: <code>average</code> , <code>min</code> , or <code>max</code> . The value is calculated from the capacitance values obtained for the various input slew values used for timing characterization.	average
leakage_sim_opt	Allows you to specify different simulator options specifically for leakage power acquisition. The option setting that you specify appears last in the simulation netlist for leakage power acquisition and overrides any other simulator option settings.	
min_pulse_width_mode	Extracts the best or worst pin-based <code>min_pulse_width</code> values from the simulated results with side pin combinations. Set the attribute to <code>min</code> or <code>max</code> to select the value for the relevant pin-based attribute. For pin-based minimum pulse width, the default slew is 0.1 ps and the default load value is 5e-14F. You can alter these values by setting the <code>min_pulse_width_slew</code> , <code>min_pulse_width_slew_high</code> , <code>min_pulse_width_slew_low</code> , and <code>constraint_total_output_net_capacitance</code> attributes in the library or cell template file.	
min_pulse_width_slew	Allows you to assign the input slew for pin-based minimum pulse width acquisition. You can define the value in library time units. The default is 0.1 ps. This attribute affects only pin-based minimum pulse width, not arc-based minimum pulse width.	0.1 ps

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
min_pulse_width_slew_high	Allows you to specify the input slew value used in the characterization of the <code>min_pulse_width_high</code> and <code>min_pulse_width_low</code> pin attributes. If this attribute is used, it overwrites the <code>min_pulse_width_slew</code> value.	0.1 ps
	This attribute affects only pin-based minimum pulse width, not arc-based minimum pulse width.	
min_pulse_width_slew_low	Allows you to specify the input slew value used in the characterization of the <code>min_pulse_width_high</code> and <code>min_pulse_width_low</code> pin attributes. If this attribute is used, it overwrites the <code>min_pulse_width_slew</code> value.	0.1 ps
	This attribute affects only pin-based minimum pulse width, not arc-based minimum pulse width.	
ncx_add_default_conditional_receiver_model	Controls the addition of default receiver capacitance arcs (arcs that do not contain a <code>when</code> condition) when Liberty NCX synthesizes receiver capacitance arcs from a template description of a cell. If set to <code>true</code> , Liberty NCX checks to see if there are default arcs and adds them for each valid receiver capacitance type in the cell. This is done even if all the valid states of an arc are synthesized. The <code>ncx_add_default_conditional_receiver_model</code> attribute can be specified on an entire library if it is specified in the library template file, or it can be specified on a cell or pin if specified appropriately in the cell template file. The default value of this attribute is <code>false</code> .	<code>false</code>
ncx_defaultindex_notoggle_ccs_rcv	Specifies to use table indexes for generating CCS receiver models for input pin transitions that do not toggle output pins.	<code>true</code>
	The default value of this attribute is <code>true</code> . The <code>ncx_defaultindex_notoggle_ccs_rcv</code> attribute is used when Liberty NCX generates CCS receiver models for delay arcs that are sensitized by either a rise or fall transition on the related input pin.	
	If set to <code>true</code> , and if no appropriate user-specified index is found, then the first available set of index values for input net transition and total output net capacitance are used.	
	If set to <code>false</code> , the index set used for the opposite input edge, (that edge which sensitizes the delay arc), is used. This is the preferred setting to enable a uniform library structure.	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_add_default_constraint_arcs	<p>Controls the addition of default timing arcs (arcs that do not contain a <code>when</code> condition) when Liberty NCX synthesizes timing arcs from a template description of a cell. If set to <code>true</code>, Liberty NCX checks to see if there are default arcs and adds them for each valid constraint arc type in the cell. This is done even if all the valid states of an arc are synthesized.</p> <p>The <code>ncx_add_default_constraint_arcs</code> attribute can be specified on an entire library if it is specified in the library template file, or it can be specified on a cell or pin if specified appropriately in the cell template file. The default value of this attribute is <code>false</code>.</p>	<code>false</code>
ncx_add_default_delay_arcs	<p>Controls the addition of default timing arcs (arcs that do not contain a <code>when</code> condition) when Liberty NCX synthesizes timing arcs from a template description of a cell. If set to <code>true</code>, Liberty NCX checks to see if there are default arcs and adds them for each valid delay arc type in the cell. This is done even if all the valid states of an arc are synthesized.</p> <p>The <code>ncx_add_default_delay_arcs</code> attribute can be specified on an entire library if it is specified in the library template file, or it can be specified on a cell or pin if specified appropriately in the cell template file. The default value of this attribute is <code>false</code>.</p>	<code>false</code>
ncx_add_default_internal_power	<p>Specifies that default <code>internal_power</code> arcs should be created for any arcs that have multiple state-dependent propagating or nonpropagating <code>internal_power</code> arcs. Valid values for this attribute are <code>min</code>, <code>max</code>, or <code>avg</code>. The value determines how the power values in the default power arc are computed. Leave this attribute unset if you do not wish to create default internal power arcs.</p>	
ncx_auto_function_pin_pattern	<p>Specifies name patterns to guide automatic function recognition. For more information, see “Automatic Function Recognition” on page 2-16”.</p>	<code>q_pin Q</code> <code>qn_pin QN</code>
ncx_skip_auto_function	<p>Skips the <code>auto_function</code> configuration file variable analysis for selected cells. Autofunction recognition skips acquisition for the cell when the variable is set to <code>true</code>, regardless of the cell template content and the <code>auto_function</code> configuration file variable configuration setting.</p>	<code>false</code>

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_cell_leakage_power_mode	Determines how the <code>cell_leakage_power</code> attribute value in a cell is calculated from the state-dependent values in the <code>leakage_power</code> models. Valid values are <code>min</code> , <code>max</code> (the default) or <code>avg</code> .	max
ncx_condition	Allows you to specify an NCX condition, which consists of the following components:	
	<ul style="list-style-type: none"> • A set of characterization criteria, with a string <code>ncx_condition_prefix</code>, that represent the conditions to match when Liberty NCX determines NCX condition usage. • An optional set of Liberty or NCX characterization timing arc attributes that represent your custom characterization. • An optional harness group, specified using <code>ncx_harness</code> syntax, which is used when Liberty NCX characterizes an arc that matches the NCX condition characterization criteria. • An optional <code>design_rule</code> extraction attribute specified by using the <code>ncx_update_max_capacitance</code> and <code>ncx_update_max_transition</code> attributes to enable extraction or preserve the specified <code>design_rule</code> values. <p>For more information, see “Conditional Characterization” on page 3-90.</p>	
ncx_condition_output_toggle	Restricts conditional violation definition to only constraints with toggled or non-toggled outputs. Valid values are <code>true</code> and <code>false</code> .	true
	For more information, see “ Toggled and Non-Toggled Output Constraint Arcs ” on page 3-95.	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_conditional_receiver_model	<p>Specifies whether to generate conditional pin receiver capacitance models. If the attribute is set to true, Liberty NCX generates multiple pin receiver models based on different <code>when</code> conditions, as specified by the <code>ncx_conditional_receiver_model_states</code> attribute. If it is set to false, Liberty NCX generates a single pin receiver model.</p>	false
See also the ncx_conditional_receiver_model_states attribute in Table 3-3.		
ncx_constraint_accuracy	<p>Specifies constraint arc acquisition accuracy. This value controls the threshold at which the constraint search stops.</p>	2 ps
ncx_constraint_outlier_abs_tol	<p>Performs polynomial fitting on SPICE results when the specified tolerance is greater than zero (0). The tolerance is specified in library time units.</p> <p>When both absolute and relative errors are greater than the given tolerances specified by the <code>ncx_constraint_outlier_abs_tol</code> and <code>ncx_constraint_outlier_rel_tol</code> attributes, the tool finds an outlier.</p>	0.0
	<p>If you enable autofix, a new simulation netlist is resimulated with high accuracy settings. If one outlier exists in the row or column after resimulation, Liberty NCX replaces that outlier with the latest SPICE results and prints a warning message. Otherwise, the tool provides an informative message and ends with an error.</p> <p>Take care to specify the tolerance properly because a tight tolerance might produce a false outlier.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_constraint_outlier_rel_tol	<p>Performs polynomial fitting on SPICE results for outlier checking in the constraint table. The tolerance is specified in library time units.</p> <p>When both absolute and relative errors are greater than the given tolerances specified by the <code>ncx_constraint_outlier_abs_tol</code> and <code>ncx_constraint_outlier_rel_tol</code> attributes, the tool finds an outlier.</p> <p>If you enable autofix, a new simulation netlist is resimulated with high accuracy settings. If one outlier exists in the row or column after resimulation, Liberty NCX replaces that outlier with the latest SPICE results and prints a warning message. Otherwise, the tool provides an informative message and ends with an error.</p> <p>Take care to specify the tolerance properly because a tight tolerance might produce a false outlier.</p>	0.3
ncx_constraint_search_interval	Specifies the width of the search interval used by the constraint arc bisection search.	2 ns
ncx_default_constraint_arcs_only	<p>The <code>ncx_default_constraint_arcs_only</code> attribute has been replaced by the <code>ncx_create_arcs</code> attribute. Instead of setting <code>ncx_default_constraint_arcs_only</code> to false to determine whether conditional constraint (setup and hold) arcs are created between input and output pin pairs, you should set <code>ncx_create_arcs</code>, as shown:</p> <pre>ncx_create_arcs : * * * constraint states all</pre> <p>If you use <code>ncx_default_constraint_arcs_only</code>, Liberty NCX issues a warning message saying that <code>ncx_default_constraint_arcs_only</code> is deprecated in favor of <code>ncx_create_arcs</code>.</p> <p>For more information, see “Arc Generation Control” on page 3-65.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_default_delay_arcs_only	<p>The <code>ncx_default_delay_arcs_only</code> attribute has been replaced by the <code>ncx_create_arcs</code> attribute. Instead of setting <code>ncx_default_delay_arcs_only</code> to false to determine whether conditional delay arcs are created between input and output pin pairs, you should set <code>ncx_create_arcs</code>, as shown:</p>	
	<pre>ncx_create_arcs : * * * delay states all</pre> <p>If you use <code>ncx_default_delay_arcs_only</code>, Liberty NCX issues a warning message saying that <code>ncx_default_delay_arcs_only</code> is deprecated in favor of <code>ncx_create_arcs</code>.</p> <p>For more information, see “Arc Generation Control” on page 3-65.</p>	
ncx_default_clear_arcs_only	<p>The <code>ncx_default_clear_arcs_only</code> attribute has been replaced by the <code>ncx_create_arcs</code> attribute. Instead of setting <code>ncx_default_clear_arcs_only</code> to false to separate clear arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set <code>ncx_create_arcs</code>, as shown:</p>	false
	<pre>ncx_create_arcs : * * * clear states all</pre> <p>If you use <code>ncx_default_clear_arcs_only</code>, Liberty NCX issues a warning message saying that <code>ncx_default_clear_arcs_only</code> is deprecated in favor of <code>ncx_create_arcs</code>.</p> <p>For more information, see “Arc Generation Control” on page 3-65.</p>	
ncx_default_conditional_receiver_mode	<p>Determines which state-dependent arc is chosen as the default (state-independent) arc, from a set of receiver capacitance arcs between two pins with valid <code>when</code> conditions on the side-input pins.</p> <p>The <code>first</code> value selects the first valid conditional receiver as the default model.</p> <p>The <code>best</code> and <code>worst</code> values select the state-dependent arc with the <code>best</code> or <code>worst</code> receiver capacitance, respectively.</p> <p>The <code>best_points</code> and <code>worst_points</code> values specify the receiver capacitance with the best or worst points (independent of each other) for the respective states at each index point.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_default_min_pulse_width_arcs_only	<p>The <code>ncx_default_min_pulse_width_arcs_only</code> attribute has been replaced by the <code>ncx_create_arcs</code> attribute. Instead of setting <code>ncx_default_min_pulse_width_arcs_only</code> to false to separate minimum pulse width arcs from constraint arcs (setup, hold, recovery, and removal arcs) and to obtain the arc states, you should set <code>ncx_create_arcs</code>, as shown:</p> <pre data-bbox="556 650 1204 703">ncx_create_arcs : * * * min_pulse_width states all</pre> <p>If you use <code>ncx_default_min_pulse_width_arcs_only</code>, Liberty NCX issues a warning message saying that <code>ncx_default_min_pulse_width_arcs_only</code> is deprecated in favor of <code>ncx_create_arcs</code>.</p>	
ncx_default_non_sequential_arcs_only	<p>The <code>ncx_default_non_sequential_arcs_only</code> attribute has been replaced by the <code>ncx_create_arcs</code> attribute. Instead of setting <code>ncx_default_non_sequential_arcs_only</code> to false to separate nonsequential arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set <code>ncx_create_arcs</code>, as shown:</p> <pre data-bbox="556 1157 1204 1189">ncx_create_arcs : * * * non_seq* states all</pre> <p>If you use <code>ncx_default_non_sequential_arcs_only</code>, Liberty NCX issues a warning message saying that <code>ncx_default_non_sequential_arcs_only</code> is deprecated in favor of <code>ncx_create_arcs</code>.</p> <p>For more information, see “Arc Generation Control” on page 3-65.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_default_preset_arcs_only	<p>The <code>ncx_default_preset_arcs_only</code> attribute has been replaced by the <code>ncx_create_arcs</code> attribute. Instead of setting <code>ncx_default_preset_arcs_only</code> to false to separate preset arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set <code>ncx_create_arcs</code>, as shown:</p>	
	<pre>ncx_create_arcs : * * * preset states all</pre> <p>If you use <code>ncx_default_preset_arcs_only</code>, Liberty NCX issues a warning message saying that <code>ncx_default_preset_arcs_only</code> is deprecated in favor of <code>ncx_create_arcs</code>.</p> <p>For more information, see “Arc Generation Control” on page 3-65.</p>	
ncx_delay_sensitization_mode	<p>Controls the states of unspecified pins while customizing the side-pin states using the <code>ncx_create_arcs</code> or <code>ncx_when</code> attributes. The syntax is as follows:</p>	first
	<pre>ncx_delay_sensitization_mode : worst best [first]</pre> <p>The <code>first</code> mode selects the first state-dependent arc encountered in the pin group, whereas the <code>best</code> and <code>worst</code> options select the state-dependent arc with the best or worst NLDM delay, respectively.</p>	
ncx_prop_power_sensitization_mode	<p>Controls the states of unspecified pins while customizing the side-pin states using the <code>ncx_create_arcs ... prop_power</code>. The syntax is as follows:</p>	first
	<pre>ncx_prop_power_sensitization_mode : worst best [first]</pre> <p>The <code>first</code> mode selects the first state-dependent arc encountered in the pin group, whereas the <code>best</code> and <code>worst</code> options select the state-dependent arc with the best or worst NLPM propagating power energy, <code>nc_int_energy</code>.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_internal_pin_nodeset	<p>Allows you to direct Liberty NCX to initialize an internal node to a specific voltage level in the SPICE simulation deck. Internal nodes are defined as a pin group in the template by using a function statement, and the <code>ncx_internal_pin_signal_level_high</code> and <code>ncx_internal_pin_signal_level_low</code> attributes can also be defined for them. Based on the function statement, an internal node is preset by using the HSPICE .nodeset feature. The valid values are <code>true</code> and <code>false</code>.</p> <p>.</p>	false
ncx_internal_pin_signal_level_high	<p>Specifies the signal levels in volts for internal nodes in order to preset the internal nodes to a specific voltage before simulation occurs. If both <code>ncx_internal_pin_signal_level_high</code> and <code>ncx_internal_pin_signal_level_low</code> are not specified, then the default nominal voltage is used as <code>ncx_internal_pin_signal_level_high</code>.</p> <p>This attribute allows you to preset internal nodes prior to characterization and can accept a voltage source name or an integer as a value. The following example specifies the name VDD3 as the voltage name:</p> <pre data-bbox="556 1115 1160 1248">pin NTCNIE{ function : !IE; ncx_internal_pin_signal_level_high : VDD3; ncx_internal_pin_signal_level_low : 0; }</pre> <p>If the voltage name is specified incorrectly, Liberty NCX issues a warning similar to the following message:</p>	nom_voltage of the library

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_internal_pin_signal_level_low	<p>Specifies signal levels in volts for internal nodes in order to preset the internal nodes to a specific voltage before simulation occurs. If both <code>ncx_internal_pin_signal_level_high</code> and <code>ncx_internal_pin_signal_level_low</code> are not specified, then 0 (zero) is used as <code>ncx_internal_pin_signal_level_low</code>.</p> <p>This attribute allows you to preset internal nodes prior to characterization and can accept a voltage source name or an integer as a value. For an example of using a voltage source name, see <code>ncx_internal_pin_signal_level_high</code>.</p>	0
ncx_input_power_period_scale	<p>Applies a scalar factor on the simulation stop time so that the integration period for dynamic power is adjusted for nonpropagating scaled arcs. See also the <code>ncx_output_power_period_scale</code> attribute.</p> <p>If the <code>ncx_input_power_period_scale</code> and <code>ncx_output_power_period_scale</code> attributes are defined anywhere in the template, they override the <code>ncx_power_period_scale</code> attribute.</p>	
ncx_internal_power_mode	Specifies whether to exhaustively simulate all possible nonpropagating states for a cell (<code>all</code>) or just the first possible state (<code>first</code>). This can be specified at the library, cell, or pin level.	first
ncx_leakage_power_mode	This attribute has been replaced by <code>ncx_create_arcs</code> . For more information, see “ Arc Generation Control ” on page 3-65.	
ncx_leakage_adj_mode	<p>Specifies whether transient or direct current (DC) leakage is to be used for power adjustment.</p> <p><code>ncx_leakage_adj_mode [post_process sim_based]</code></p>	DC
ncx_leakage_meas_mode	<p>Specifies whether to perform leakage subtraction in the after-processing stage or use transient-based simulation adjustment.</p> <p><code>ncx_leakage_meas_mode [tran DC]</code></p>	DC
ncx_input_power_leakage_meas_interval	<p>Specifies the start and stop time for leakage measurement during nonpropagation power measurement.</p> <p><code>ncx_input_power_leakage_meas_interval : start_time stop_time</code></p>	1.1 1.2

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_output_power_leakage_meas_interval	<p>Specifies the start and stop time for leakage measurement during propagation power measurement.</p> <pre>ncx_output_power_leakage_meas_interval : start_time stop_time</pre>	1.1 1.2
ncx_leakage_meas_interval	<p>Specifies the start and stop time for leakage measurement during leakage power measurement.</p> <pre>ncx_leakage_meas_interval start_factor end_factor</pre>	1.1 1.2
ncx_min_setup_based_delay	<p>Measures minimum clock-to-output delay for sequential cells. When you set the <code>ncx_min_setup_based_delay</code> attribute to true, Liberty NCX begins by finding the clock to output delay with the minimum setup time between the clock and the data. Then, it applies the minimum setup time to characterize the clock-to-output delay in a regular transition simulation.</p> <p>For more information, see “Minimum Setup Time for Pessimistic Delay” on page 3-113.</p>	false
ncx_min_setup_based_delay_constraint_slew	<p>Specifies data slew value for minimum clock-to-output delay measurement. When you use <code>ncx_min_setup_based_delay</code> to find the minimum setup time of a clock slew, the data slew must be fixed. You can select the data slew value by using the <code>ncx_min_setup_based_delay_constraint_slew</code> attribute. The attribute specifies the constraint pin slew value from the delay table. The values are <code>middle</code>, <code>min</code>, and <code>max</code>.</p> <p>For more information, see “Minimum Setup Time for Pessimistic Delay” on page 3-113.</p>	middle
ncx_min_setup_based_delay_setup_time_offset	<p>Adds margin to minimum clock-to-output delay measurement. When you set the <code>ncx_min_setup_based_delay_setup_time_offset</code> attribute, Liberty NCX adds a margin to the setup time used for characterizing the delay and output transition time. The default value is <code>1e-12</code> seconds.</p> <p>For more information, see “Minimum Setup Time for Pessimistic Delay” on page 3-113.</p>	1e-12 seconds
ncx_mpw_rail_to_rail	WControls whether minimum pulse width characterization considers only full rail-to-rail voltage swings.	true

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_nlpm_distribution_method	<p>ncx_nlpm_distribution_method : cap_ratio equal</p> <p>This attribute applies to multiple output switching in multiple power and ground pins. It provides you with the control to choose between two dynamic energy methodologies.</p> <p>The <code>equal</code> option specifies equal distribution for multiple output switching. That is, all the arcs from the switching input to the multiple switching outputs are assumed to consume equal energy.</p> <p>The tool does not consider there to be any specific association of the output node and the power supply driving the output during energy computations.</p> <p>Switching energy adjustment, or CV^2 subtraction, is performed without considering the power supply charging the capacitor.</p> <p>The <code>cap_ratio</code> option uses a new methodology to distribute energy for multiple output switching, based on the loads the outputs are driving instead of assuming equal distribution.</p> <p>The tool considers only the power supplies driving the switching output nodes in the computation of energy for those outputs.</p> <p>The tool performs power-supply specific adjustment on the power supply that is actually charging the load capacitor.</p>	cap_ratio
ncx_three_state_disable_res	<p>Specifies the resistor values of a voltage divider for tristate disable arcs. The following example specifies a resistor value of 10:</p> <pre>ncx_three_state_disable_res : 10 ;</pre>	1
ncx_tristate_disable_arc_subtract_cvv	Specifies the subtraction of CV^2 in NLPM calculation for tristate disable arc.	false
ncx_node_set	<p>Defines internal node or output node sets. Instead of specifying <code>violation_mode</code> for each internal node and output node, use <code>ncx_node_set</code> to combine the nodes into sets, and then specify <code>violation_mode</code> for the sets.</p> <p>For more information, see “Constraint Methodologies” on page 3-141.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_optimize_default_arc_state	<p>Controls the determination of the side-pin sensitization for default arcs.</p> <p>When state and default arcs exist between two pins in a cell, the default arcs are typically extracted from the state arcs at the end of the characterization flow and not during the pre-analysis stage for reasons of efficiency. However, if you want the determination of the default arc states to be done during the pre-analysis stage, set the <code>ncx_optimize_default_arc_state</code> to <code>true</code>. You can specify this for the entire library, a specific cell, a pin, or an arc.</p> <p>This temperature attribute has no effect if the <code>default_arc_mode</code> is set to <code>best-points</code> or <code>worst-points</code>, because no explicit model state exists to be determined and each point of the table might correspond to a different state.</p> <p>The valid values are of Boolean: <code>true</code> and <code>false</code>.</p>	<code>false</code>
ncx_optimize_state_power_arcs	<p>Merges similar nonpropagating power arcs, which reduces the output library file size. The <code>ncx_optimize_state_power_arcs</code> tolerance is set by the <code>ncx_optimize_state_power_arcs_tol_pct</code> template attribute, which is set to 10 percent by default. States with overlapping behaviors, where the model values are very close to each other as determined by the value of <code>ncx_optimize_state_power_arcs_tol_pct</code>, are merged into single power models with when conditions that capture all the states. The <code>ncx_optimize_state_power_arcs</code> attribute is set to <code>false</code> by default.</p>	<code>false</code>
ncx_optimize_state_timing_arcs	<p>Optimizes the characterization and modeling of multiple-state timing arcs between two pins. If the value of <code>ncx_optimize_state_timing_arcs</code> is set to <code>merge</code>, states with overlapping behaviors (where the model values are very close to each other as determined by the value of <code>ncx_optimize_state_timing_arcs_tol_pct</code>) are merged into single timing models with when conditions that capture all the states. This reduces the output library file size.</p> <p>Set <code>ncx_optimize_state_timing_arcs</code> to <code>true</code> if you are characterizing MUXs.</p>	<code>false</code>
ncx_optimize_state_timing_arcs_tol_pct	Determines the percentage tolerance in model values that are used for detecting overlapping states for a timing arc. The default value is <code>1.0</code> .	<code>1.0</code>

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_optimize_va_constraint_arcs_compact	<p>Uses a reduced table size for the variation-aware constraint table. The variation-aware constraint table (5x5) is characterized with a reduced table size (3x3) to produce better runtime. Set <code>ncx_optimize_va_constraint_arcs_compact</code> to <code>true</code> to publish the actual table from simulation (3x3). Otherwise, the table (5x5) is obtained by interpolating the simulation result. The valid values are Boolean: <code>true</code> or <code>false</code>.</p>	<code>false</code>
ncx_output_power_period_scale	<p>Applies a scalar factor on the simulation stop time so that the integration period for dynamic power is adjusted for propagating scaled arcs. See also the <code>ncx_input_power_period_scale</code> attribute.</p> <p>If the <code>ncx_input_power_period_scale</code> and <code>ncx_output_power_period_scale</code> attributes are defined anywhere in the template, they override the <code>ncx_power_period_scale</code> attribute.</p>	
ncx_output_threshold_pct_fall	<p>Specifies the delay falling trip point for recalculating NLDM delay value without simulation on a pin-by-pin basis because the Liberty <code>output_threshold_pct_fall</code> attribute cannot be specified on a pin. This attribute can be specified on a pin level in a cell template.</p> <p>Using the postprocess process, based on the CCS timing library (for example, one created with 50%/50% thresholds), a new NLDM timing library can be created for different thresholds (for example, it can be created for 90%/10% thresholds) without resimulation. For example,</p> <pre>pin IO { ncx_output_threshold_pct_rise : 90 ; ncx_output_threshold_pct_fall : 10 ; }</pre> <p>In the postprocess flow, Liberty NCX is able to adjust the NLDM delay according to its CCS timing model and the new delay trip point that is specified by the <code>ncx_output_threshold_pct_fall</code> attribute. The valid values are of type float.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_output_threshold_pct_rise	<p>Specifies the delay rising trip point for recalculating NLDM delay value without simulation on a pin-by-pin basis because the Liberty output_threshold_pct_rise attribute cannot be specified on a pin. This attribute can be specified at the pin level in a cell template.</p> <p>Using the postprocess process, based on the CCS timing library (for example, one created with 50%/50% thresholds), a new NLDM timing library can be created for different thresholds (for example, it can be created for 90%/10% thresholds) without resimulation. For example,</p> <pre>pin IO { ncx_output_threshold_pct_rise : 90 ncx_output_threshold_pct_fall : 10; }</pre> <p>In the postprocess flow, Liberty NCX is able to adjust the NLDM delay according to its CCS timing model and the new delay trip point that is specified by the ncx_output_threshold_pct_rise attribute. The valid values are of type float.</p>	
ncx_output_transition_lower_bound_pct	Sets up the lower boundary of the full swing output transition time expressed as a percentage. The tool measures output swing until it reaches that percentage of the output voltage. The valid values are of type double.	0.5
ncx_output_transition_upper_bound_pct	Sets the upper boundary of the full-swing output transition time expressed as a percentage. The tool measures output swing until it reaches that percentage of the voltage. The valid values are of type double.	99.5
ncx_pin_capacitance_ranges	Controls the generation of NLDM pin capacitance ranges in the output library. The default value is false, which suppresses the generation of NLDM pin capacitance ranges. If the attribute is set to true, the pin capacitance ranges are published.	false

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_retention_pin	<p>Specifies the pin class and disable value for retention pins and is required whenever the power_gating_pin is specified on a retention pin. The syntax is</p> <pre>ncx_retention_pin : "pin_class disable_value"</pre> <p>where the valid values for the pin_class option are the following:</p> <ul style="list-style-type: none"> restore, which restores the state of the cell. save, which saves the state of the cell. save_restore, which saves and restores the state of the cell. <p>The valid values for disable_value, which defines the value of the retention pin when the cell works in normal mode, are 0 and 1.</p> <p>Specify the values within quotation marks (""). For example:</p> <pre>ncx_retention_pin : "save_restore 0.0"</pre>	
ncx_shpr_hold_positive	<p>Changes the default behavior of using a hold time of 0 ps during SHPR measurement when the conventional hold time is negative. To change this behavior to use the negative hold time, set the ncx_shpr_hold_positive attribute to false. The valid values are true or false.</p>	true
ncx_update_max_capacitance	<p>Updates the maximum capacitance values if the design rule model acquisition is active. Allows you to selectively control the acquisition of the pin-based max_capacitance design rule. You can specify this attribute at a library, cell, or pin level to achieve the appropriate control over all pins, all pins of a cell, or a specific pin, respectively.</p> <p>If set to false, any max_capacitance values that might be present in the seed library or cell template are maintained in the output library generated by Liberty NCX. This option is relevant only if the design_rule configuration variable is active. The valid values are Boolean: true or false. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	true

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_update_max_transition	<p>Updates the maximum transition values if the design rule model acquisition is active. Allows you to selectively control the acquisition of the pin-based <code>max_transition</code> design rule. You can specify this attribute at a library, cell, or pin level to achieve the appropriate control over all pins, all pins of a cell, or a specific pin, respectively.</p> <p>If set to false, any <code>max_transition</code> values that might be present in the seed library or cell template are maintained in the output library generated by Liberty NCX. This option is relevant only if the <code>design_rule</code> configuration variable is active. The valid values are Boolean: true or false. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	true
ncx_input_delay_reference_mode	<p>Specifies the delay reference of a pair of input differential signals when measuring the delay from either one of the input signals to output signals. The syntax is</p> <pre>ncx_input_delay_reference_mode differential signal_threshold</pre> <p>For more information, see “Differential Pin Delay Measurement Modes” on page 3-115</p>	differential for differential pins
ncx_output_delay_reference_mode	<p>Specifies the delay reference of a pair of output differential signals when measuring the delay from one or more input signals to either one of the output signals. The syntax is</p> <pre>ncx_output_delay_reference_mode differential signal_threshold</pre> <p>For more information, see “Differential Pin Delay Measurement Modes” on page 3-115</p>	differential
ncx_input_delay_differential_value	<p>Specifies that the differential input delay reference should use a delayed differential signal crossing, specified in library voltage units.</p> <p>For more information, see “Delayed Differential Measurements” on page 3-117</p>	0

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
ncx_output_delay_differential_value	<p>Specifies that the differential output delay reference should use a delayed differential signal crossing, specified in library voltage units.</p> <p>For more information, see “Delayed Differential Measurements” on page 3-117”</p>	0
ncx_input_delay_differential_pct	<p>Specifies that the differential input delay reference should use a delayed differential signal crossing, specified as a percentage of the rail-to-rail voltage swing.</p> <p>For more information, see “Delayed Differential Measurements” on page 3-117”</p>	
ncx_output_delay_differential_pct	<p>Specifies that the differential output delay reference should use a delayed differential signal crossing, specified as a percentage of the rail-to-rail voltage swing.</p> <p>For more information, see “Delayed Differential Measurements” on page 3-117”</p>	0
ncx_use_leakage_model_file_for_dc_only	Specifies whether the leakage model file specified with the <code>leakage_model_file</code> template attribute should be used for DC leakage simulation only, or for both DC and transient leakage simulation.	true
ncx_use_library_sensitization	Specifies whether to use or ignore the sensitization information in the input library. If the attribute is set to <code>true</code> , Liberty NCX sensitizes the cell according to what is contained in the input library. If it is set to <code>false</code> , Liberty NCX determines its own sensitization for the cell.	true
nlpm_gate_leakage	Includes gate leakage information in NLPM leakage power calculation. This attribute is set to true by default, meaning that gate leakage is considered in NLPM leakage power calculation.	true
power_exclude_pin	<p>Specifies external power pins to exclude from power characterization. Set the <code>power_exclude_pin</code> attribute to exclude the power of an external power pin. Specify the name of any power pin or power node that is included in the library or cell subcircuit netlist, such as VDD, VSS, VDDC, or VSSC. To specify multiple pins, use quotation marks as shown:</p> <pre>power_exclude_pin : "VDDC VSSC" ;</pre> <p>Setting the attribute changes the NLPM leakage power and internal power calculation. There is no change in CCS power.</p>	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
setup_hold_method	<p>Selects a setup and hold pessimism reduction (SHPR) setup and hold pair to specify the conventional setup and hold time in a library.</p> <p>When you set <code>setup_hold_method</code> to <code>max_setup</code> or <code>max_hold</code>, Liberty NCX enables the SHPR flow automatically and acquires the data. However, Liberty NCX requires additional simulation in order to get the SHPR result and replace the conventional setup and hold value. The <code>setup_hold_method</code> attribute is set to <code>minimum</code> by default.</p>	minimum
side_out_total_output_net_capacitance	<p>Sets the output load of side output pins on a pin-by-pin basis. You can also use <code>side_out_total_output_net_capacitance</code> with the <code>ncx_condition</code> structures to specify side loads for different acquisitions. The values are specified in library units for capacitance.</p> <p>The attribute accepts a value or multiple pin name pair values. Consequently, the number of values of the attribute is 1 or an even number. You can use wildcards for the pin name for a cell with many output pins.</p> <p>Note: The <code>side_out_total_output_net_capacitance</code> attribute overlaps with the <code>constraint_total_output_net_capacitance</code> attribute for constraint arcs. Therefore, if both attributes are specified for constraint arcs, <code>constraint_total_output_net_capacitance</code> overrides <code>side_out_total_output_net_capacitance</code>.</p> <p>When you specify a value for the <code>side_out_total_output_net_capacitance</code> attribute, it is used as the load for the side output pins in the simulation SPICE deck. If you do not specify a value, the tool uses the middle index value in the load table indexes as a default. If no indexes are specified, as is the case with constraint arcs, the tool uses 50.0 fF as the side output load.</p>	50.0 fF
sim_inc_file	Specifies a simulation “include” file in the netlist. The string of this option is included by using the simulator command <code>.inc</code> .	

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
sim_opt	Specifies any extra simulation conditions. The conditions are copied verbatim into the characterization netlist, on a line prefixed by the appropriate command for the HSPICE simulator. The value of this option is appended to a line beginning with the .OPTION keyword. This option is useful for dealing with cells that are difficult to characterize. For example, the following HSPICE simulator settings increase accuracy at the cost of more runtime: ACCURATE=1 RMAX=0.1 RUNLVL=6	
slew_lower_threshold_pct_fall	Specifies the lower threshold for determining the slew of a falling signal, expressed as a percentage of the rail voltage; typically used only in the library template.	20.0
slew_lower_threshold_pct_rise	Specifies the lower threshold for determining the slew of a rising signal, expressed as a percentage of the rail voltage; typically used only in the library template.	20.0
slew_upper_threshold_pct_fall	Specifies the upper threshold for determining the slew of a falling signal, expressed as a percentage of the rail voltage; typically used only in the library template.	80.0
slew_upper_threshold_pct_rise	Specifies the upper threshold for determining the slew of a rising signal, expressed as a percentage of the rail voltage; typically used only in the library template.	80.0
snps_pedriver_ratio	Specifies the relative weighting of the ramp waveform component in the Synopsys predriver waveform, as a floating-point number between 0 and 1. A value of 1.0 results in a linear waveform for the driver, whereas a value of 0 results in an exponential waveform.	0.5
switching_power_split_model	Controls the switching energy calculations associated with the output load capacitor when Liberty NCX is generating the internal energy model. If the attribute is set to <code>true</code> , the switching energy is calculated as $0.5 \times C \times V^2$ and subtracted from both the rise and fall measured energy to obtain the rise or fall internal energy of the cell. If it is set to <code>false</code> , the switching power is calculated as $C \times V^2$ and subtracted from the rise measured energy to get the rise internal energy.	false

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
three_state_pullup_res	Specifies the pull-up and pull-down resistor values that Liberty NCX uses when characterizing a three-state cell.	10000 ohm
three_state_pulldn_res	If the timing arc has an output from Z to 0, a pull-up resistor is used to ensure the output voltage is close to VDD before the Z-to-0 transition. Similarly, if the timing arc has an output from Z to 1, a pull-down resistor is used to ensure the output voltage is close to VSS before the Z-to-1 transition.	
tran_timestep	Specifies transient simulation time step, in seconds.	100e-12
violation_delay_degrade_pct	Specifies the maximum allowable CLK-to-output delay degradation permitted for constraint arc characterization, as a percentage of nominal delay.	10
violation_delay_degrade	Specifies the maximum allowable CLK-to-output delay degradation permitted for constraint arc characterization, in library time units.	
violation_delay_degrade_check	Specifies whether Liberty NCX should check for unexpected output delay behavior during constraint characterization.	false
violation_slew_degrade_pct	Specifies the maximum allowable output pin slew degradation permitted for constraint arc characterization, as a percentage of nominal slew.	10
violation_slew_degrade	Specifies the maximum allowable output pin slew degradation permitted for constraint arc characterization, in library time units.	
violation_glitch_lower_pct	Specifies the lower threshold for non-toggled output for glitch detection. You can specify a value from 0 - 100. This feature is only available with the HSPICE and Eldo simulators.	10
violation_glitch_mode	The <code>violation_glitch_mode</code> attribute has been replaced by the <code>violation_mode</code> attribute. For more information, see “ Constraint Methodologies ” on page 3-141 .	false
violation_glitch_upper_pct	Specifies the upper threshold for non-toggled output for glitch detection. You can specify a value from 0 - 100. This feature is only available with the HSPICE and Eldo simulators.	90

Table 3-1 Attributes in Library and Cell Template Files (Continued)

Attribute	Description	Default
violation_mode	Specifies an advanced mode violation definition that determines how constraint arcs are acquired. Available methods are delay degradation, slew degradation, glitch detection, and capture pass-fail.	
violation_mode_fall		
violation_mode_rise	<code>violation_mode_rise</code> and <code>violation_mode_fall</code> can be used to specify different advanced mode violation definitions for rising and falling arcs, respectively.	
	For more information, see “ Constraint Methodologies ” on page 3-141 .	
violation_pass_fail_mode	The <code>violation_pass_fail_mode</code> attribute has been replaced by the <code>violation_mode</code> attribute. For more information, see “ Constraint Methodologies ” on page 3-141 .	false
violation_delay_degrade_min_value	The <code>violation_delay_degrade_min_value</code> attribute sets the lower bound value to prevent small delay degradations.	5 ps
zstate_leak_threshold_pct	Specifies leakage current threshold, expressed as a percentage of pretransition leakage current, that signifies the onset of a “Z” or high-impedance state at a pin. This is used primarily during acquisition of three-state cells.	10.0

Table 3-2 Attributes in Library Template Files

Attribute	Description	Default
active_drv_ground_node	Specifies the name of the ground node used in the active driver subcircuit.	
active_drv_ground_voltage	Specifies the value of ground voltage used for the active driver subcircuit.	
active_drv_input	Specifies the name of the input node of the active driver, as defined in the subcircuit interface.	
active_drv_inwav	Specifies the driver type at the input of the active driver subcircuit, either <code>snps_predriver</code> (the standard Synopsys predriver waveform) or <code>ramp</code> (an ideal linear ramp).	snps_predriver

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
active_drv_netlist	Specifies the file name of the netlist file containing the subcircuit description of the active driver cell.	
active_drv_output	Specifies the name of the output node of the active driver, as defined in the subcircuit interface.	
active_drv_slew	Specifies the input slew value (rail to rail) to be used to drive the input of the active driver subcircuit. The output slew of the active driver, which drives the cell under test, is controlled by the capacitive load on the output pin of the active driver.	
active_drv_slew_tol_pct	Controls the merging of slews that are close to each other. The default for <code>active_drv_slew_tol_pct</code> is 10.	10
active_drv_supply_node	Specifies the name of the supply node that powers the active driver subcircuit.	
active_drv_supply_voltage	Specifies the value of the supply voltage used to power the active driver subcircuit.	
active_drv_unate	Specifies whether the active driver cell is an inverting cell. This attribute should be set to <code>negative</code> if the active driver subcircuit inverts the input waveform, or <code>positive</code> otherwise.	
active_drv_voltage_map	Specifies the voltage levels of the rails of the active driver. For example, <code>active_drv_voltage_map : VDDZ 1.2 VSSZ 0.0 VDDY 1.0 VSSY 0.0</code>	
do	Specifies a list of cells to include in the characterization flow. This option, when used with the <code>dont</code> option, allows incremental characterization of cells. When you start with a seed library, the <code>do</code> list can be used to specify additional cells to be added to the output library. When you create a library from scratch, the <code>do</code> list specifies all the constituent cells of the new library. A cell template file must exist for each of the cells in the <code>do</code> list.	
dont	Specifies a list of cells to exclude from the characterization flow. This option allows incremental characterization of cells.	

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
driver_model	<p>Specifies the type of driver to be used at the cell inputs for characterization: <code>ramp</code>, <code>snps_predriver</code>, or <code>active_driver</code>. <code>ramp</code> is an ideal, linear ramp with a specified transition time. <code>snps_predriver</code> is the standard Synopsys predriver waveform generated from the specified transition time, which represents a more realistic driving waveform. <code>active_driver</code> is a specified driver circuit that Liberty NCX automatically sets up to generate waveforms of different transition times. The parameters of the active driver circuit must be specified in the library template file.</p>	snps_predriver
global_vdd	<p>Specifies the name of the global power pin used in the netlist.</p>	
global_vss	<p>Specifies the name of the global ground pin used in the netlist.</p>	

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
<code>ibis_condition_min</code> <code>ibis_condition_typ</code> <code>ibis_condition_max</code>	<p>Specifies the parameters for the minimum, typical, and maximum operating conditions of the IBIS model generation. These are required for IBIS model generation. The following options can be set for these attributes.</p> <p><code>temperature</code> float specifies the corner temperature.</p> <p><code>pull_down_voltage</code> float specifies the pull-up voltage (often referred to as power).</p> <p><code>pull_up_voltage</code> float specifies the pull-down voltage (often referred to as ground).</p> <p><code>ibis_model_file</code> string overrides the model file specified in the configuration file for this corner.</p> <p><code>ibis_netlist_dir</code> string overrides the <code>netlist_dir</code> variable specified in the configuration file for this corner.</p> <p><code>ibis_netlist_suffix</code> string overrides the <code>netlist_suffix</code> variable specified in the configuration file for this corner.</p> <p>Of these options, only <code>temperature</code>, <code>pull_down_voltage</code>, and <code>pull_up_voltage</code> are required. You can use the other parameters to override values in the configuration file. The valid values are of type <code>string</code>. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	
<code>ibis_model_file</code>	<p>Overrides the model file specified in the configuration file. IBIS requires three operating conditions to be specified: minimum, typical, and maximum. These operating conditions are specified by using the following attributes:</p> <pre>ibis_condition_min ibis_condition_typ ibis_condition_max</pre> <p>Using <code>ibis_model_file</code>, overrides the model file used for one or all operating conditions. This allows you to specify various SPICE models for different operating conditions. The valid values are of type <code>string</code>. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
ibis_netlist_dir	<p>Overrides the netlist directory specified in the configuration file. IBIS requires 3 operating conditions to be specified: minimum, typical, and maximum. These operating conditions are specified by using the following attributes:</p> <pre data-bbox="621 551 882 656">ibis_condition_min ibis_condition_typ ibis_condition_max</pre>	
	<p>Using <code>ibis_netlist_dir</code> overrides the netlist directory used for one or all operating conditions. This allows you to specify various SPICE netlists for different operating conditions. The valid values are of type string. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	
ibis_netlist_suffix	<p>Overrides the netlist suffix specified in the configuration file. IBIS requires three operating conditions to be specified: minimum, typical, and maximum. These operating conditions are specified by using the following attributes:</p> <pre data-bbox="621 1007 882 1113">ibis_condition_min ibis_condition_typ ibis_condition_max</pre>	
	<p>Using <code>ibis_netlist_suffix</code> overrides the netlist suffix used for one or all operating conditions. This allows you to specify various SPICE netlists for different operating conditions. The valid values are of type string. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	
ncx_check_nominal_va_constraint	<p>Checks to determine if both plus and minus variation curves are on the same side of the nominal curve. When <code>ncx_check_nominal_va_constraint</code> is set to <code>false</code>, the tool does not issue a warning message, although both variation curves might be out of bounds. When it is set to true, if either one is out of bounds, the tool issues a warning message that describes where the variation curves are located and how many curves have been found within the cell.</p>	<code>false</code>

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
ncx_create_3d_table	Specifies whether three-dimensional arcs should automatically be created where possible. By default, three-dimensional NLDM timing, NLPM power, and constraint arcs are only created when three index types are specified for an arc. When this attribute is set to <code>true</code> , Liberty NCX will look for opportunities to identify a related output, and will automatically create three-dimensional lookup tables.	<code>false</code>
ncx_create_arcs	<p>Enables manual vector reduction for leakage and power arcs. You can manually control the nonpropagating internal power arcs on input pins to specify which arcs are synthesized for timing models and for the instances of power models. Allows side-input pin conditions to be used for capacitance acquisition.</p> <p>Note that you can use the * and ? wildcard characters to denote a family of cells, pins, or models. This allows more flexibility and minimizes repetition in the template file. When a specification is not applicable, such as when you are setting an attribute for minimum pulse width, the pin names can both be the same since they are associated with a single pin.</p> <p>Note: The <code>ncx_leakage_power_when</code> attribute is no longer supported. The functionality of <code>ncx_leakage_power_when</code> is now available in <code>ncx_create_arcs</code>.</p> <p>For more information, see “Arc Generation Control” on page 3-65.</p>	<code>default</code> <code>arcs only</code>
ncx_create_half_unate_arcs	<p>Sensitizes half-unate arcs when used with the <code>timing_type</code> attribute.</p> <p>The default value of this attribute is <code>false</code>.</p> <p>For more information, see “Half-Unate Arcs” on page 3-76.</p>	<code>false</code>
ncx_default_power_arcs_only	Specifies that default <code>internal_power</code> arcs should be created for any arcs that have multiple state-dependent propagating or nonpropagating <code>internal_power</code> arcs. A setting of <code>true</code> causes default power arcs to be generated; <code>false</code> prevents them from being generated. To prevent default internal power arc creation, the <code>ncx_add_default_internal_power</code> attribute must also remain undefined.	<code>false</code>

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
ncx_force_create_arcs	Activates the arc creation phase of the Liberty NCX flow when in the presence of user-specified timing arcs in the cell template files. It is used primarily to augment Liberty NCX arcs with custom arcs.	false
ncx_force_create_leakage	Causes Liberty NCX to use existing simulation results from previous runs, if available.	false
ncx_input_tristate_mode	<p>Specifies the sensitization of the control pins of a tristate inout pin when generating state-dependent arcs that are driven by the inout pin. This attribute is used to avoid undesirable feedback and interaction from the inout pin when it is used as input.</p> <p>The default value of this attribute is <code>disable</code>.</p> <p>For more information, see “Bidirectional Cell Arcs” on page 3-74.</p>	disable
ncx_nlpm_leakage_subtraction_mode	<p>Specifies how state-dependent leakage energy is subtracted from total energy during NLPM power characterization. Liberty NCX has two methods to subtract leakage energy from total measured energy when computing internal energy.</p> <p>If you specify <code>init</code>, the tool tries to match the pre-switching leakage power state, if it exists.</p> <p>If you specify <code>final</code>, then the tool matches the post-switching leakage power state, if it exists.</p>	init
ncx_nlpm_mode	<p>Models the power group per power supply, using the <code>rail_connection</code> format, or per power and ground pin, using the <code>pg_pin</code> format.</p> <p>The <code>ncx_nlpm_mode</code> values are <code>aggregate</code> and <code>split</code>. In <code>split</code> mode, the NLPM internal and leakage power table is split into multiple tables for each power and ground pin. The <code>split</code> mode is not valid for a library with a single power supply. Use the <code>aggregate</code> mode for libraries with a single power supply. In <code>aggregate</code> mode, the power tables are not modeled for each supply pin.</p> <p>For more information, see “Power and Ground Pin Connections” on page 3-131.</p>	aggregate

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
ncx_use_pg_pins	<p>Adds <code>pg_pin</code> and <code>voltage_map</code> syntax to all libraries during characterization for IEEE 1801 compliance, also known as Unified Power Format (UPF). If you do not want to add <code>pg_pin</code> and <code>voltage_map</code> syntax to libraries during characterization, you must set <code>ncx_use_pg_pins</code> to <code>false</code>.</p> <p>Note: CCS power characterization requires a power format based on the power and ground pin syntax. Therefore, if you set the <code>ccs_power</code> variable in the configuration file to <code>true</code>, enabling <code>ccs_power</code>, Liberty NCX ignores the <code>ncx_use_pg_pins</code> setting and uses the <code>pg_pin</code> format for power modeling.</p> <p>For more information, see “Power and Ground Pin Connections” on page 3-131.</p>	true
pull_down_voltage	<p>Specifies the pull-down voltage for an IBIS operating condition. IBIS requires three operating conditions to be specified: minimum, typical, and maximum. These operating conditions are specified by using the following attributes.</p> <pre data-bbox="584 1015 850 1121">ibis_condition_min ibis_condition_typ ibis_condition_max</pre> <p>Within each of these statements, you must use the <code>pull_down_voltage</code> statement to specify a ground rail for the operating condition. The valid values are of type <code>string</code>. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	
pull_up_voltage	<p>Specifies the pull-up voltage for an IBIS operating condition. IBIS requires three operating conditions to be specified: minimum, typical, and maximum. These operating conditions are specified by using the following attributes.</p> <pre data-bbox="584 1480 850 1586">ibis_condition_min ibis_condition_typ ibis_condition_max</pre> <p>Within each of these statements, you must use the <code>pull_up_voltage</code> statement to specify a power rail for the operating condition. The valid values are of type <code>string</code>. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	

Table 3-2 Attributes in Library Template Files (Continued)

Attribute	Description	Default
side_pin_driver_model	<p>Specifies the type of driver to be used at the side input pins for characterization. The values are <code>ramp</code> or <code>active_driver</code>, where <code>ramp</code> is an ideal, linear ramp with a specified transition time and <code>active_driver</code> is a specified driver circuit that Liberty NCX automatically sets up to generate waveforms of different transition times. The active driver circuit parameters must be specified in the library template file.</p> <p>The <code>side_pin_driver_model</code> attribute is supported with the HSPICE simulator only.</p>	<code>ramp</code>
temperature	<p>Specifies the temperature for an IBIS operating condition. IBIS requires three operating conditions to be specified: minimum, typical, and maximum. These operating conditions are specified by using the following attributes.</p> <ul style="list-style-type: none"> <code>ibis_condition_min</code> <code>ibis_condition_typ</code> <code>ibis_condition_max</code> <p>Within each of these statements, you must use the <code>temperature</code> statement to specify an operating condition temperature. The valid values are of type <code>string</code>. For more information, see Chapter 4, “I/O Cell Characterization.”</p>	
va_nominal_values	Specifies a list of nominal values corresponding to the parameter names specified by <code>va_parameters</code> .	
va_parameters	Specifies a list of variation-aware library parameter names. The generated library contains cells characterized at different values of these parameters.	
va_variation_values	Specifies a list of absolute parameter values corresponding to the parameter names specified by <code>va_parameters</code> . These are the parameter values at which cells are characterized, either higher or lower than the nominal values specified by <code>va_nominal_values</code> .	

Table 3-3 Attributes in Cell Template Files

Attribute	Description	Default
	<p>The following attributes control NLDM capacitance extraction:</p> <pre>ncx_capacitance_input_net_transition_index ncx_capacitance_total_output_net_capacitance_index ncx_capacitance_rise_input_net_transition_index ncx_capacitance_rise_total_output_net_capacitance_index ncx_capacitance_fall_input_net_transition_index ncx_capacitance_fall_total_output_net_capacitance_index</pre> <p>The values of the attributes correspond to explicit indexes set in the library template, and they represent the indexes to be used for calculating NLDM pin capacitance. The method of calculation is determined by the <code>input_cap_mode</code> attribute, which can specify <code>max</code>, <code>min</code>, or <code>average</code>. For more information, see “Input NLDM Capacitance Index Values” on page 3-88.</p>	
<code>differential_pair</code>	Defines two output pins to be differential. The attribute should list the names of the two pins, separated by a space character. The delay trip points are measured differentially; the trip point is where the two output voltages cross each other.	
<code>input_fall_threshold</code>	Specifies the timing threshold on input falling edge, in volts.	1.5
<code>input_rise_threshold</code>	Specifies the timing threshold on input rising edge, in volts.	1.5
<code>input_signal_level_high</code> <code>input_signal_level_low</code>	Specifies a nonrail input signal levels, in volts. Input pins are driven directly by a nonrail PWL source that swings between the specified high and low voltage levels.	
<code>input_signal_level</code>	Specifies the constant voltage applied to an input pin.	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_active_edge	<p>Describes the active edge of clock pins or enable signals. Required when a specified state table or truth table exists for a given cell with multiple clock pins and enable signals.</p> <p>The valid values are the following:</p> <ul style="list-style-type: none"> R for an active edge that is falling F for an active edge that is rising. H for an active edge that is active high. L for an active edge that is active low. B for an active edge that is both rising and falling. 	
ncx_clear	<p>Specifies a clear pin. Set ncx_clear to L (active low) or H (active high) in the pin group. In the following example, pin R is an active low clear pin:</p>	
	<pre>pin R { direction : input; ncx_clear : L;</pre>	
ncx_condition_arc_type	<p>Specifies the use of the condition for a specific arc type and can assume one of the following values:</p> <ul style="list-style-type: none"> • constraint • delay • tristate • prop_power <p>Note that you can use the asterisk (*) and question mark (?) wildcard characters to denote a family of pins to gain more flexibility and to minimize repetition in the template file. This will accept more than one pin name if the pin names do not have pattern matching, you can specify the pins explicitly. The keywords are fixed. For example, you could use:</p>	
	<pre>ncx_condition_arc_type : delay;</pre>	
ncx_condition_cell_name	<p>Specifies the use of the condition for a specific cell. The value is the name of the cell.</p> <p>Note that you can use the asterisk (*) and question mark (?) wildcard characters to denote a family of pins to gain more flexibility and to minimize repetition in the template file. This attribute accepts more than one pin name, so if the pin names do not have pattern matching, you can specify the pins explicitly. For example, you could use</p>	
	<pre>ncx_condition_cell_name : inverter;</pre>	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_condition_cell_type	<p>Specifies the use of the condition for a specific cell type and can assume one of the following values:</p> <ul style="list-style-type: none"> • sequential • combinational • flipflop • latch • clock_gating <p>Specifies the use of the condition for a specific cell. The value is the name of the cell.</p> <p>Note that you can use the asterisk (*) and question mark (?) wildcard characters to denote a family of pins to gain more flexibility and to minimize repetition in the template file. This attribute accepts more than one pin name, so if the pin names do not have pattern matching, you can specify the pins explicitly. The keywords are fixed. For example, you could use:</p> <pre>ncx_condition_cell_type : combinational;</pre>	
ncx_condition_pin	<p>Specifies the use of the condition for all arcs that terminate at the named pin.</p> <p>Note that you can use the asterisk (*) and question mark (?) wildcard characters to denote a family of pins to gain more flexibility and to minimize repetition in the template file. This attribute accepts more than one pin name, so if the pin names do not have pattern matching, you can specify the pins explicitly.</p> <p>For example, you could use:</p> <pre>ncx_condition_pin : out;</pre>	
ncx_condition_related_pin	<p>Specifies the use of the condition for all arcs that originate from the named pin.</p> <p>Note that you can use the asterisk (*) and question mark (?) wildcard characters to denote a family of pins to gain more flexibility and to minimize repetition in the template file. This attribute accepts more than one pin name, so if the pin names do not have pattern matching, you can specify the pins explicitly. For example, you could use:</p> <pre>ncx_condition_related_pin : clock;</pre>	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_conditional_receiver_model_states	<p>Specifies the pin states for which to generate conditional pin receiver capacitance models. This attribute is used in conjunction with the ncx_conditional_receiver_model attribute set to true. The attribute must be specified in the cell template file in the pin group.</p> <p>If you set the following,</p> <pre>ncx_conditional_receiver_model_states : "CK" " !CK";</pre> <p>Liberty NCX generates pin receiver models for CK equal to 0 and CK equal to 1.</p> <p>See also the ncx_conditional_receiver_model attribute in Table 3-1.</p>	
ncx_internal_leakage_when	This attribute has been replaced by the ncx_create_arcs attribute. For more information, see “Arc Generation Control” on page 3-65 .	
ncx_internal_power_when	<p>Specifies explicitly the “state” conditions for which to acquire the internal input power for a specified pin. For example, consider the following definition under an input pin in a cell attribute file:</p> <pre>ncx_internal_power_when : (A*B) (A*B')</pre> <p>In this example, Liberty NCX acquires two internal power arcs for the pin, using only the two specified “when” conditions on inputs A and B. If this attribute is used, it overrides the ncx_internal_power_mode attribute setting.</p>	
ncx_max_transition_mode	Selects the maximum or minimum as the max_transition design rule of a pin from the values defined by any or all of the ncx_max_transition_related_pin_transition_pt, ncx_max_transition_constrained_pin_transition_pt, and ncx_max_transition_input_pin_transition_pt template attributes.	min
ncx_max_transition_related_pin_transition_pt	Selects a specified related pin slew index point as the maximum transition design rule of a pin.	
ncx_max_output_transition_time	Specifies the maximum output transition time (in seconds) desired for an output pin.	
ncx_max_transition_constrained_pin_transition_pt	Selects a specified constrained pin slew index point as the maximum transition design rule of a pin.	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_max_transition_input_net_transition_pt	Selects a specified input slew index point as the maximum transition design rule of a pin.	
ncx_min_total_output_net_capacitance	Specifies the minimum load capacitance (in farads) for an output pin.	
ncx_mode_total_output_net_capacitance	Specifies the variation of load capacitance between the specified minimum value and the acquired maximum value. This may be one of log or linear.	
ncx_optimize_state_leakage_power	Specifies whether merging optimization is applied to state-dependent leakage power. When you set the attribute to true, Liberty NCX retains only the when condition of the chosen leakage group. When you set the attribute to merge, the when condition of the merged leakage groups is concatenated with an OR operator. The valid values are true, false, and merge.	false
ncx_optimize_state_leakage_power_mode	Controls the merging behavior when leakage power merging optimization is enabled. The attribute defines which value in the merged groups will be published. The valid values are max, min, and average.	max
ncx_optimize_state_leakage_power_tol	Controls the merging tolerance when leakage power merging optimization is enabled, in leakage power library units. When you set the attribute, all values less than the ncx_optimize_state_leakage_power_tol tolerance value are merged.	
ncx_optimize_state_leakage_power_tol_pct	Controls the merging tolerance when leakage power merging optimization is enabled, as a percentage of the leakage power table values. If the leakage power values are within a ncx_optimize_state_leakage_power_tol_pct tolerance percentage of each other, Liberty NCX merges them.	5%
ncx_out_to_out_arcs	Enables output-to-output combinational delay arcs for test output pins. Liberty NCX supports output-to-output arc characterization, where the timing of an output pin takes into consideration the load of the first output pin. You can generate and characterize combinational delay arcs between a cell's two output pins if both output pins have the same function and the destination output pin is a test_output_only pin as indicated in the cell's test_cell group. Set ncx_out_to_out_arcs to true to enable this feature. You must specify the loads for the test_output_only pin.	false

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_preset	Specifies a preset pin. Set <code>ncx_preset</code> to H (active high) or L (active low) in the pin group. In the following example, pin S is an active high preset pin: pin S { direction : input; ncx_preset : H;	
ncx_pulse_clock	Specifies whether a particular clock pin uses pulsed clock.	false
ncx_signal_skew	Specifies that one input pin should be delayed by the specified time in library time units when characterizing cells with simultaneous switching inputs or differential input pin pairs.	
ncx_size_total_output_net_capacitance	Specifies the number of load capacitance index points desired.	
ncx_skip	Skips characterization for the specified timing arc. When <code>ncx_skip</code> is set to <code>true</code> on a timing arc, Liberty NCX copies the timing arc unchanged (as is) into the library. No checks are done, except Library Compiler rule checking, and the arc is not considered for default arc construction. No variation-aware data is generated.	true
ncx_translate_msff	Specifies whether to translate the ff group with <code>clocked_on</code> and <code>clocked_on_also</code> to a cell with two latches in series, rather than being translated as two flip-flops in series. The default value is <code>false</code> , which means any ff group with <code>clocked_on</code> and <code>clocked_on_also</code> is translated to two flip-flops in series.	false
ncx_vary_related_input_slew	Varies the slew for both the pin driving the timing arc and the side pin. If two input pins of a MUX are simultaneously switching, then the slew varies for both pins. This attribute can take a value of <code>true</code> or <code>false</code> . The default is <code>true</code> .	true
ncx_wave_fall	Specifies timing-arc sensitization vectors for falling arcs. The value is a user-specified string. For more information, see “ Sensitization Vectors ” on page 3-51.	
ncx_wave_rise	Specifies timing-arc sensitization vectors for rising arcs. The value is a user-specified string. For more information, see “ Sensitization Vectors ” on page 3-51.	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_when	<p>Specifies a side-pin sensitization during library characterization. This condition is not published in the library that is generated. In addition, if a <code>when</code> condition is present, <code>ncx_when</code> is appended to it during characterization but will not modify it in the library.</p> <p>In the following syntax, Liberty NCX matched all arcs whose <code>when</code> condition is "A." And to those arcs, it adds the condition "<code>!scan</code>" for characterization.</p> <pre>ncx_condition set_when { ncx_condition_when : "A" ; ncx_when : "!scan" ; }</pre>	
ncx_when_fall	Specifies side-pin sensitization during library characterization for falling arcs. The condition is not published in the library that is generated. In addition, if a <code>when</code> condition is present, <code>ncx_when_fall</code> is appended to it during characterization but does not modify it in the library.	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
ncx_when_ignore_function_pins	<p>Specifies that pins referenced in the function definition should be filtered from the ncx_when value. When you set the ncx_when_ignore_function_pins attribute to true in the library or cell template file and specify an ncx_when condition for an arc that conflicts with the function statement, the tool overrides the ncx_when condition with the function statement and continues with library characterization.</p>	false
	<p>For example, if you specify the following conditions in the template file:</p>	
	<pre>ncx_condition Delay_Side_Pins_all { ncx_condition_pin : * ; ncx_condition_arc_type : delay ; ncx_when : "A & !TMODE & JTAGAS & !JTAGA & !HSE & !FILE" ; } pin JTAGAZ { direction : output ; function: (A&!JTAGAS) (JTAGA&JTAGAS) ; related_ground_pin : VSS ; related_power_pin : VDD ; }</pre>	
	<p>the arcs to JTAGAZ cannot be realized with the condition JTAGA=0, A=1, JTAGAS=1. As a result, Liberty NCX sensitization fails. When you set the ncx_when_ignore_function_pins attribute to true, the tool overrides the ncx_when statement with the function statement. To realize the arc in the previous example during sensitization, the tool ignores the pins A, JTAGA, and JTAGAS in the ncx_when statement and sets the other side pins to the states specified in the ncx_when statement.</p>	
ncx_when_rise	<p>Specifies side-pin sensitization during library characterization for rising arcs. The condition is not published in the library that is generated. In addition, if a when condition is present, ncx_when_rise is appended to it during characterization but does not modify it in the library.</p>	
non_rail_output_signal_level	<p>Specifies that the output swing is not rail to rail, either true (output swing is not rail-to-rail) or false (output swing is rail-to-rail). If true, the minimum and maximum values of the voltage swing are not default (0.0 to VDD) but should be measured; the measured swing is used to determine the output slew.</p>	
output_fall_threshold	<p>Specifies the timing threshold on output falling edge.</p>	1.5

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
output_rise_threshold	Specifies the timing threshold on output rising edge.	1.5
output_signal_level_high output_signal_level_low	Specifies a nonrail output signal levels, in volts. Liberty NCX applies the measurement threshold percentages to the specified low-to-high voltage range rather than the range from 0.0 to the rail voltage.	
pin_opposite	Defines two input pins to be differential. The attribute should list the names of the two pins, separated by a space character. The input signals should always be inverted with respect to each other.	
ref_state	Specifies the constant state of a side pin, either 0 or 1.	
related_output_pin	Specifies the related output pin for a cell arc which references multiple output pins. For an example using an n -bit shift register, see “ Conditional Characterization ” on page 3-90.	
sensitization	Specifies the sensitization of the cell, using the sensitization syntax.	
simple_termination_netlist	Specifies the path of a termination subcircuit netlist.	
simple_termination_pin	Specifies the cell pin that will be attached to a termination circuit.	

Table 3-3 Attributes in Cell Template Files (Continued)

Attribute	Description	Default
simultaneous_switching	<p>Allows you to specify simultaneous switching inputs for a cell. For example, in addition to specifying the function for a 1-hot MUX with three select pins, S1, S2, and S3, you can model simultaneous switching behavior. You do this by specifying the contention_condition function in the .lib file to establish the direction of the switching inputs and by specifying simultaneous_switching in the cell template file.</p> <p>Note: The simultaneous_switching attribute is not recorded in the output library.</p> <p>For more information, see “Simultaneous Switching” on page 3-124.</p>	
simultaneous_switching	<p>Specifies simultaneous switching inputs for a cell. Multiple attributes should be used to describe groups of inputs that switch independently of each other. For example, pins A0 and A1 switching together, which are independent of pins B0 and B1 switching together, are described as follows:</p> <pre>simultaneous_switching : 2 A0 A1 ; contention_condition : A0&!A1 !A0&A1 ; simultaneous_switching : 2 B0 B1 ; contention_condition : B0&!B1 !B0&B1 ;</pre>	

Table 3-4 is a summary listing of margin attributes in template files.

Table 3-4 Margin Attributes in Template Files

Attribute	Description	Default
margin_applied	Indicates, at the arc level, whether or not the current library data has had margins added to the simulation results. For more information, see “ Timing Margins ” on page 3-96 .	false
margin_exp	Specifies a margin expression with multiple margin definitions. The resulting margin values can be combined according to the margin_exp_mode attribute. For more information, see “ Timing Margins ” on page 3-96 .	
margin_exp_mode	Specifies how multiple margin definitions in a margin expression defined with the margin_exp attribute should be combined. Available methods are the numerical max (max), the numerical min (min), the average (avg), or the sum (sum). For more information, see “ Timing Margins ” on page 3-96 .	

Table 3-4 Margin Attributes in Template Files (Continued)

Attribute	Description	Default
margin_mode	Specifies how the margins computed by the margin_value, margin_percent, and margin_exp margin methods should be combined. Available methods are the numerical max (<code>max</code>), the numerical min (<code>min</code>), or the sum (<code>sum</code>). By default margin_value takes precedence over margin_percent. For more information, see “Timing Margins” on page 3-96 .	
margin_percent	Specifies margin as a function of nominal value. Use this attribute to increase the characterized timing data by a specified percentage. Valid floating point values in the range of 0 to 100 can be specified. For more information, see “Timing Margins” on page 3-96 .	
margin_value	Specifies margin as an absolute value in library time units. This attribute increases the characterized timing data by a specific value. The valid values are of type <code>float</code> . For more information, see “Timing Margins” on page 3-96 .	
ncx_margin_max	Applies an upper bound to a margin change (delta). This value is in library time units. The valid values are of type <code>float</code> . For more information, see “Timing Margins” on page 3-96 .	
ncx_margin_min	Applies a lower bound to a margin change (delta). This value is in library time units. The valid values are of type <code>float</code> . For more information, see “Timing Margins” on page 3-96 .	
ncx_margin_model	Specifies whether to add or remove a margin or do nothing (preserve) a library, cell, pin, or arc attribute. This option is not published in the output library. It changes the margin_applied arc attribute to <code>true</code> after adding a margin or to <code>false</code> when no margins are added or when margins are removed. The valid values are of type <code>string</code> and are <code>add</code> , <code>remove</code> , and <code>preserve</code> . For more information, see “Timing Margins” on page 3-96 .	

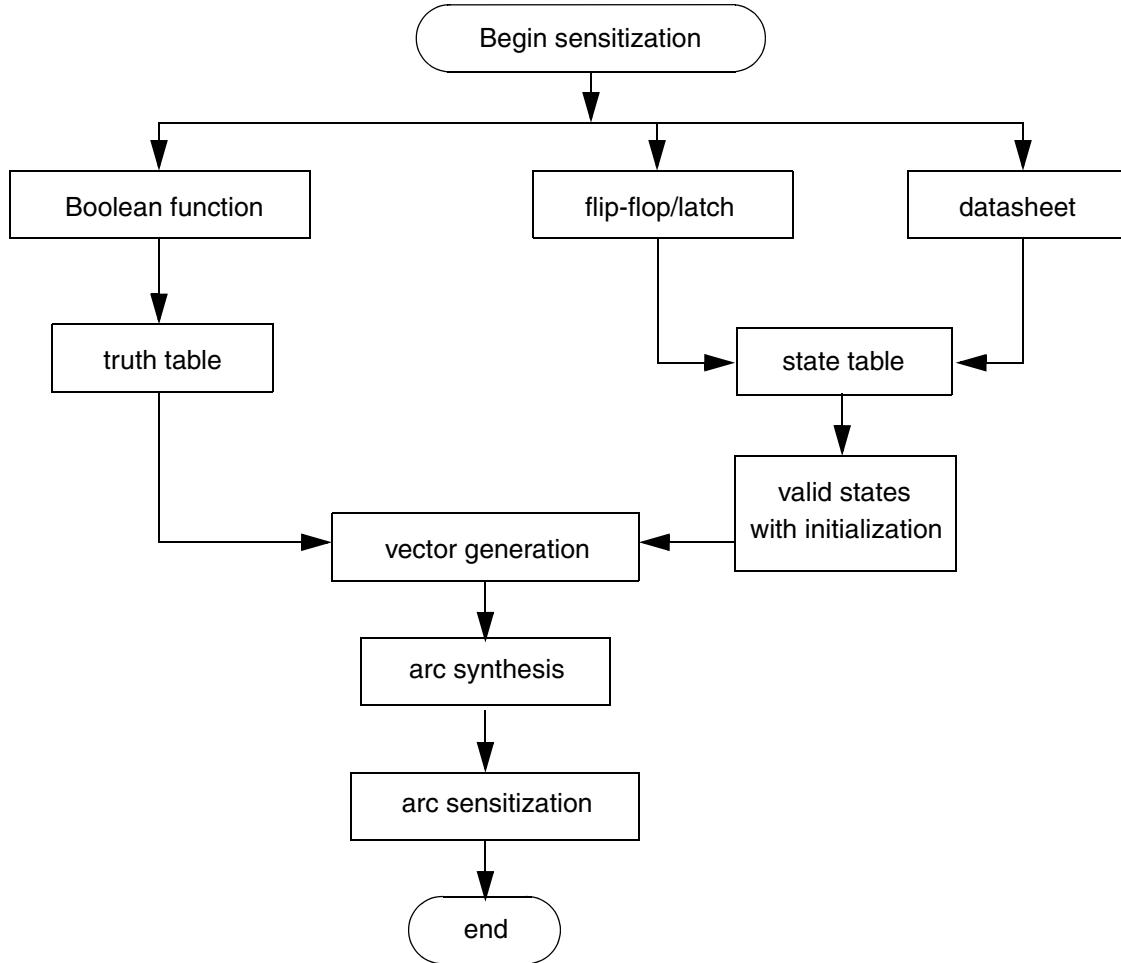
Sensitization

To set up each circuit simulation that is used for characterization, Liberty NCX must consider the logic conditions leading up to the transition being characterized. This logic condition setup process is called sensitization.

Cell sensitization generates valid sensitization vectors for the cell from the functional descriptors of the cell, such as the Boolean function on the output pin, the flip-flop and latch group information, and the information in the state table or truth table.

[Figure 3-1 on page 3-50](#) shows the sensitization flow in Liberty NCX.

Figure 3-1 Sensitization Flow for Flip-Flop and Latch Groups



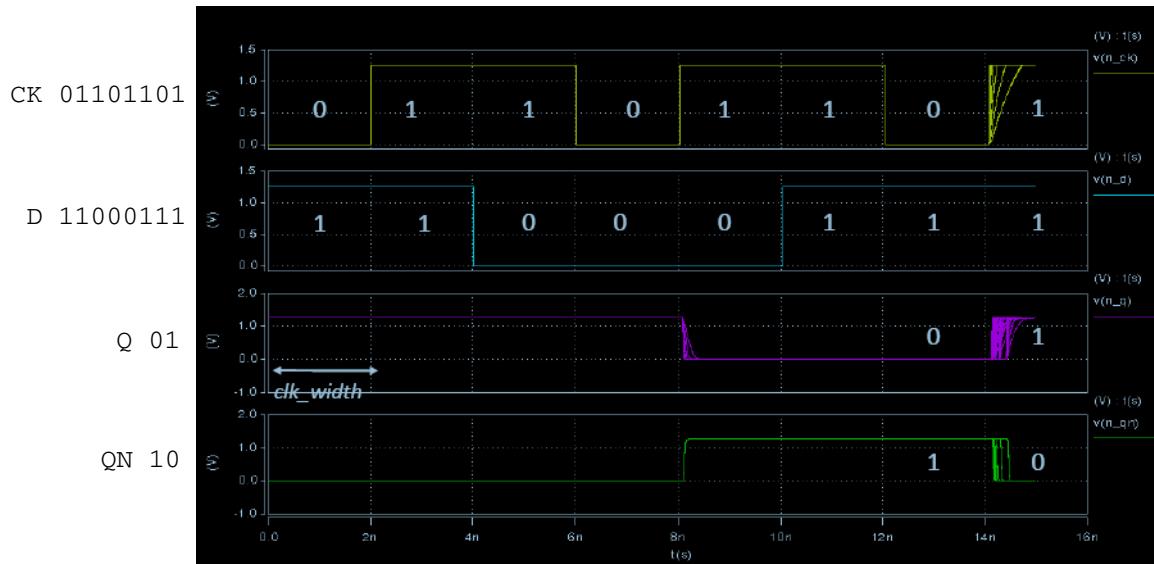
If an input seed library is used, it might contain explicit cell sensitization information using the Liberty `wave_rise` and `wave_fall` attributes. By default, Liberty NCX uses any such information to sensitize each cell. If you prefer to have Liberty NCX determine its own sensitization instead of using sensitization from the input seed library, set the `ncx_use_library_sensitization` attribute to `false` in the cell or library template file.

Liberty NCX can determine the sensitization of simple logic cells. However, more complex cells might require you to enable the AutoFunction feature, or to specify the sensitization explicitly in the cell template file. The template file can contain one or more sensitization vector tables, or a truth table, but not both.

Sensitization Vectors

A sensitization vector is a list of input pins with associated bit streams representing the input stimulus for each pin and a list of output pins with associated bit streams representing the output response of each pin. A complete cell sensitization description consists of several sensitization vectors, some representing propagating states and others representing nonpropagating states of the cell. [Figure 3-2](#) shows the sensitization vectors for a clock-to-output D flip-flop.

Figure 3-2 Sensitization Vectors for Clock-to-Output D Flip-Flop



A sensitization vector must follow these conventions:

- The bit streams of input pins should consist of only 0 or 1 characters.
- The bit streams of output pins should consist of only 0,1, or Z characters.
- Vectors that sensitize delay (propagating) arcs should have at least one input pin and at least one output pin with bit streams lengths that are less than 1 and that represent an input toggle and an associated output toggle, as shown in the following example:

```
D 00111100 CK 01101001 Q 10
```

- Vectors that sensitize delay arcs should have bit streams of exactly a length of 2 for the toggling output pins, representing the toggle of interest, as shown in the following example:

```
D 00111100 CK 01101001 Q 10
```

- Vectors used to sensitize constraint models, such as setup and hold or recovery and removal, should have two associated inputs toggling successively in the last three bits, as shown in the following example:

```
D 11000011 CK 01101001 Q 10
```

- Vectors that sensitize constraint models can have output pins with bit streams lengths greater than 2.
- Vectors used for acquiring minimum pulse width on an input pin require the bit stream for that pin to have a trailing 101 or 010 sequence, as shown in the following example:

```
D 001110000 CK 011011010 Q 10
```

You can create minimum pulse width vectors by using the toggling arc vectors.

- If the bit stream of an input pin has a fewer number of bits than the maximum length of any input pin stream, the last bit is held constant for the duration of the sensitization.

Specifying Sensitization for Timing Arcs

You can use the `ncx_wave_rise` and `ncx_wave_fall` attributes to specify a user sensitization vector sequence for timing arcs. The following syntax specifies a sensitization vector for a timing arc:

```
ncx_wave_rise : inpin1 bits inpin2 bits ... outpin1 bits ;
ncx_wave_fall : inpin1 bits inpin2 bits ... outpin1 bits ;
```

The `ncx_wave_rise` and `ncx_wave_fall` attributes are placed inside the `timing` group that represents a timing arc. During timing arc characterization, the vector sequence is applied to the cell under characterization. The following example shows a few timing arcs of a D flip-flop in the cell template.

```
pin D {
    direction : input ;
    timing {
        related_pin : CK ;
        timing_type : setup_rising ;
        ncx_wave_rise : \
            D 11000011 \
            CK 01101001 \
            Q 01 ;
        ncx_wave_fall : \
            D 00111100 \
```

```

        CK 01101001 \
        Q 10 ;
    }
pin Q {
    direction : output ;
    timing {
        related_pin : CK ;
        timing_type : rising_edge ;
        timing_sense : non_unate ;
        ncx_wave_rise : D 11000111 CK 01101101 Q 01 ;
        ncx_wave_fall : D 00111000 CK 01101101 Q 10 ;
    }
}

```

Liberty NCX checks for compliance with the following rules when you specify a vector inside of an arc group definition.

- Vectors used to sensitize constraint models, such as setup and hold or recovery and removal, should have two associated inputs toggling successively in the last three bits or after the reference edge, as shown in the following example:

D 11000011 CK 01101001 Q 10

- For a setup or recovery arc, the related pin should be toggling last, and the constraint pin should be toggling just before the related pin.
- For a hold or removal arc, the constraint pin should be toggling last, and the related pin should be toggling just before constraint pin.
- For minimum pulse width arc vectors, the last three bits of a related pin should be toggling as 010 or 101. All other toggling inputs should be stable in the same time period. For example,

D 001110000 CK 011011010 Q 10

- For delay arcs, the `ncx_wave_fall` output should be falling, and the `ncx_wave_rise` output should be rising.
- Check for the `complementary_pin` condition of the related pin or constraint pin; and, if present, ensure that the complementary pin vector is inverted.
- If a `when` condition is specified inside a timing arc, the `when` condition should be consistent with the vectors specified.
- Valid combinations should exist between timing type, timing sense, and the output transition, as shown in [Table 3-5 on page 3-54](#).

Table 3-5 Valid Combinations Between Timing Type, Timing Sense, and the Output Transition

Timing type	Timing sense		
	positive_unate	negative_unate	non_unate
combinational	R->R,F->F	R->F,F->R	{R,F}->{R,F}
combinational_rise	R->R	F->R	{R,F}->R
combinational_fall	F->F	R->F	{R,F}->F
three_state_disable	R->{0Z,1Z}	F->{0Z,1Z}	{R,F}->{0Z,1Z}
three_state_enable	R->{Z0,Z1}	F->{Z0,Z1}	{R,F}->{Z0,Z1}
three_state_disable_rise	R->0Z	F->0Z	{R,F}->0Z
three_state_disable_fall	R->1Z	F->1Z	{R,F}->1Z
three_state_enable_rise	R->Z1	F->Z1	{R,F}->Z1
three_state_enable_fall	R->Z0	F->Z0	{R,F}->Z0

- Consistency checks are done between the output vector, `timing_type` attribute, and `timing_sense` attribute in the arc to make sure the vector is a valid combination, as shown in [Table 3-5](#).
- For sequential arcs, the following consistency checks are done between the `timing_type` attribute and the specified vectors:
 - `rising_edge`, if the reference edge on the related input is a rising edge.
 - `falling_edge`, if the reference edge on the related input is a falling edge.
 - `preset`, if the arc has an `ncx_wave_rise` group and does not contain only an `ncx_wave_fall` group.
 - `clear`, if the arc has an `ncx_wave_fall` group and does not contain only an `ncx_wave_rise` group.
 - `hold_rising` and `removal_rising`, if the related pin is rising and it is transitioning before a constraint pin.
 - `hold_falling` and `removal_falling`, if the related pin is falling and it is transitioning before a constraint pin.

- `setup_rising` and `recovery_rising`, if a setup check is to be done on clocked elements, and the related pin is a rising edge.
- `setup_falling` and `recovery_falling`, if a setup check is to be done on clocked elements, and the related pin is a falling edge.
- No syntax checking is done for vectors specified inside the sensitization block.

Some complex sequential cells might require lengthy sensitization vector sequences to set certain internal state elements to required values. If needed, you can guide or specify the initial conditions of the arc simulation by using the `ncx_ic` and `ncx_nodeset` attributes. Liberty NCX converts these attribute values to HSPICE `.IC` and `.NODESET` commands, and places them inside the simulation deck. These directives affect the simulation as follows.

- `.NODESET` sets the seed value of a node for DC convergence, but DC convergence may change the node value if needed.
- `.IC` holds a node to a fixed value throughout the entire DC convergence process.

By setting the initial conditions of internal state elements to known values, it might be possible to use shorter sensitization vector sequences.

Each directive contains a list of one or more pairs of node names and node values:

```
ncx_ic :      node1 value1 [node2 value2 [...]]
ncx_nodeset : node1 value1 [node2 value2 [...]]
}
```

The node name specifies the name of the node inside the cell subcircuit. Node names with special characters, such as colons, should be enclosed in double quotes. The value must be a floating-point number between 0 and 1 inclusive, and specifies the initial condition voltage value as a function of the voltage swing for that node. A value of 0 results in a zero voltage initial condition, and a value of 1 represents a full-rail voltage initial condition.

Place the `ncx_ic` and `ncx_nodeset` directives inside the timing arc group. These directives affect both the rising and falling arc directions. Alternatively, you can specify separate rising and falling arc conditions with the `ncx_ic_rise`, `ncx_ic_fall`, `ncx_nodeset_rise`, and `ncx_nodeset_fall` attributes. For example,

```
timing {
    related_pin : CK ;
    timing_type : rising_edge ;
    timing_sense : non_unate ;
    ncx_wave_rise : D 11 CK 01 Q 01 ;
    ncx_wave_fall : D 00 CK 01 Q 10 ;
    ncx_ic_rise : "int_node:1" 0 "int_node:2" 1 ;
    ncx_ic_fall : "int_node:1" 1 "int_node:2" 0 ;
}
```

In the previous example, the resulting HSPICE .NODESET directives set the specified nodes to their required initial values, then the sensitization vectors defined with the `ncx_wave_rise` and `ncx_wave_fall` attributes are applied.

It is also possible to specify initial condition values without sensitization vectors. In this case, the arc is sensitized using automatic sensitization vector generation. However, the automatic sensitization does not make use of any information from the initial condition attributes. The simulation simply applies the resulting initial condition directives for DC convergence, then applies the automatically-generated sensitization vector sequence.

Note:

When the `ncx_ic*` and `ncx_nodeset*` attributes are used with shortened initialization vector sequences, make sure that downstream tools are compatible with the shortened `wave_rise` and `wave_fall` vector sequences published in the output library. If needed, set the `sensitization_to_library` configuration variable to `false`.

Specifying Sensitization for Non-Timing Arcs

Sensitization can also be provided for nonpropagating internal power arcs, leakage power arcs, and pin capacitance arcs. These are the supported sensitization attributes:

- nonpropagating internal power arcs
 - `ncx_wave_*` sensitization vector attributes
 - `ncx_ic*` and `ncx_nodeset*` initial condition attributes
- capacitance arcs
 - `ncx_wave_*` sensitization vector attributes
 - `ncx_ic*` and `ncx_nodeset*` initial condition attributes
- leakage power arcs
 - `ncx_ic` and `ncx_nodeset` initial condition attributes

For nonpropagating internal power arcs, you can place sensitization attributes inside `internal_power` arcs defined within an input pin group:

```
pin D {
    direction : input ;
    internal_power {
        when : "!SE" ;
        ncx_wave_rise : D 1100001 CK 0110100 Q 0 ;
        ncx_wave_fall : D 0011110 CK 0110100 Q 0 ;
    }
}
```

For pin capacitance arcs, you can place sensitization attributes inside `ncx_capacitance` arcs defined within an input pin group:

```
pin D {
    direction : input ;
    ncx_capacitance {
        when : "!SE" ;
        ncx_wave_rise : D 01 CK 11 Q 00 ;
        ncx_wave_fall : D 10 CK 11 Q 11 ;
        ncx_ic_rise : "int_node:1" 0 "int_node:2" 1 ;
        ncx_ic_fall : "int_node:1" 1 "int_node:2" 0 ;
    }
}
```

For leakage power arcs, you can place sensitization attributes inside `leakage_power` arcs defined within a cell group:

```
cell DFF {
    leakage_power {
        when : "!SE&!CLK&!Q" ;
        ncx_ic : "int_node:1" 0 "int_node:2" 1 ;
    }
    leakage_power {
        when : "!SE&!CLK&Q" ;
        ncx_ic : "int_node:1" 1 "int_node:2" 0 ;
    }
}
```

In the previous example, separate rise and fall initial conditions are not needed because a leakage power arc definition represents a single static state with no switching activity.

Propagating internal power arcs are tied to timing arcs. The timing arc sensitization is also used for the propagating internal power arc sensitization.

Sensitization Vector Tables

A sensitization vector table consists of a series of lines where each line specifies a digital signal vector for each pin of the cell. For example,

```
sensitization {
    InP1,      InP2,      ... ,   InPm      : OutP1,      ... ,   OutPn ;
    InS11,     InS21,     ... ,   InSm1     : OutS11,     ... ,   OutSn1 ;
    InS12,     InS22,     ... ,   InSm2     : OutS12,     ... ,   OutSn2 ;
    ...
    ...
    ...
    ...
    InS1k,     InS2k,     . . . . . ,   InSmk     : OutS1k,     . . . . . ,   OutSnk ;
}
```

If there are no bidirectional pins, there is one sensitization table. If there is one bidirectional pin, there are two sensitization tables, one with the bidirectional pin as an input and the other with the bidirectional pin as an output.

The first line of a vector table contains the pin names. In the foregoing example, *InP1*, ..., *InPm* are the input pin names and *OutP1*, ..., *OutPn* are the output pin names. Input/output pin names or values are delimited by commas. Input pin names or values are separated from the output pin names or values by a colon. Each pin name line is terminated by a semicolon.

Following the initial pin definition line are one or more sensitization lines that specify the logic values on the pins. In the foregoing example, *InS11*, *InS21*, ..., *InSm1* are the input pin characterization character strings and *OutS11*, ..., *OutSn1* are the output pin character strings.

The order of the input pin character strings is correlated with the order of the input pin names. Similarly, the order of the output pin character strings is correlated with the order of the output pin names. Continuation lines are allowed with the use of a backslash (\).

Each input pin character string is a sequence of 0 and 1 characters that represent the sequence of logic values used to sensitize that input. Similarly, each output pin character string is a sequence of characters that represent the expected output value: 0, 1, X, and so on. The allowed characters are listed and described in [Table 3-6](#).

Table 3-6 Output Pin Value Characters

Characters	Description
r	Output rising from 0 state to 1 state
f	Output falling from 1 state to 0 state
1	Output constant in 1 state
0	Output constant in 0 state
x	Unknown state
z	Three-state disable state
0z	Output being disabled from 0 state
1z	Output being disabled from 1 state
z0	Output being enabled to 0 state
z1	Output being enabled to 1 state

Sensitization vectors that are specifically intended for acquisition of constraint parameters such as setup and hold, recovery and removal, or minimum pulse width are specified after the sensitization vectors for timing acquisition. The beginning of the constraint sensitization vector section is indicated by the following line:

```
violation;
```

The following example is a sensitization vector table for a 2-input NAND gate:

```
sensitization {
    A ,      B   :      Z ;
    01 ,     1   :      f ;
    10 ,     1   :      r ;
    1 ,      01  :      f ;
    1 ,      10  :      r ;
}
```

The following example is a sensitization vector table for a D-type flip-flop:

```
sensitization {
    D, CK : Q ;
    110001, 011011 : 00 ;
    110000, 011010 : 00 ;
    001110, 011011 : 11 ;
    001111, 011010 : 11 ;
    1100001, 0110100 : 00 ;
    1100000, 0110101 : 00 ;
    00111001, 01101100 : 11 ;
    00111000, 01101101 : f ;
    0011101, 0110111 : 11 ;
    0011100, 0110110 : 11 ;
    11000110, 01101100 : 00 ;
    11000111, 01101101 : r ;
    0011110, 0110100 : 11 ;
    0011111, 0110101 : 11 ;
    1100010, 0110111 : 00 ;
    1100011, 0110110 : 00 ;
    violation;
    11000011, 01101001 : r ;
    110000110, 011010011 : r ;
    001110000, 011011010 : f ;
    00111000, 01101101 : f ;
    110001111, 011011010 : r ;
    11000111, 01101101 : r ;
    00111100, 01101001 : f ;
    001111001, 011010011 : f ;
}
```

Timing Vector Types

Timing vectors are categorized by the following types:

- Delay vectors

Only one input pin switches during measurement unless you specify simultaneous switching. At least one output must switch. Note that the initialization sequence is optional for combinational cells.

- Constraint vectors

Constraint vectors are characterized by two inputs, related and constraint pins, that switch in succession. The output pins switch, depending on the following constraint types:

- Setup vectors require at least one switching output.
- Latch hold vectors require outputs that do not switch.

Power Vectors

Propagating power vectors follow delay vector specifications. Nonpropagating power vectors require one switching input pin and no switching output pins. Leakage power vectors have no output switching pins. All input pin states must be specified.

Nonpropagating vectors are specified in a single sensitization block format when conditions are derived from last bit state, as shown:

```
sensitization {
    D, CK : Q, QN ;
    110001, 011011 : 00, 11 ;
    110000, 011010 : 00, 11 ;
    001110, 011011 : 11, 00 ;
    001111, 011010 : 11, 00 ;
    1100001, 0110100 : 00, 11 ;
    1100000, 0110101 : 00, 11 ;
    00111001, 01101100 : 11, 00 ;
    0011101, 0110111 : 11, 00 ;
    0011100, 0110110 : 11, 00 ;
    11000110, 01101100 : 00, 11 ;
    0011110, 0110100 : 11, 00 ;
    0011111, 0110101 : 11, 00 ;
    1100010, 0110111 : 00, 11 ;
    1100011, 0110110 : 00, 11 ;
}
```

Validity Checks

Liberty NCX can autosensitize cells based on a function statement group, flip-flop group, latch group, truth table, or state table. It checks the validity of these constructs in the templates you provide. Liberty NCX performs the validity checks described in the following subsections:

- [Function Statement Validity Check](#)
- [Flip-Flop Group Validity Check](#)
- [Latch Group Validity Check](#)
- [Truth Table Validity Check](#)
- [State Table Validity Check](#)

Function Statement Validity Check

Liberty NCX performs the following checks on all function statements:

- Checks that the function is defined only for output or inout pins.
- Checks that the function uses only input pins or inout pins as terms.

Flip-Flop Group Validity Check

Liberty NCX performs the following checks on all flip-flop groups:

- Checks that the output variables, `variable1` and `variable2`, are specified, where `variable1` is the noninverting output of the flip-flop and `variable2` is the inverting output. Both of these variables must be assigned, even if one of them is not connected to a primary output pin. For example,

Syntax

```
library (lib_name) {
    cell (cell_name) {
        ...
        ff ( variable1, variable2 ) {
            clocked_on : "Boolean_expression" ;
            next_state : "Boolean_expression" ;
            clear : "Boolean_expression" ;
            preset : "Boolean_expression" ;
            clear_preset_var1 : value ;
            clear_preset_var2 : value ;
            clocked_on_also : "Boolean_expression" ;
        }
    }
}
```

- Checks that values for `variable1` and `variable2` do not match a pin name used in the cell being described.
- Checks that the `clocked_on` attribute, which identifies the active edge of the clock signal, is specified.
- Checks to that the flip-flop group has a valid `next_state` attribute whose value is a logic equation written in terms of the cell's input pins or the first state variable.
- Checks that if the `clear` and `preset` attributes are both included in the group, then `clear_preset_var1`, `clear_preset_var2`, or both attributes are defined. If `clear_preset_var1`, `clear_preset_var2`, or both attributes are included, the `clear` and `preset` attributes must be defined.

Latch Group Validity Check

Liberty NCX performs the following checks on all latch groups.

- Checks that the output variables, `variable1` and `variable2`, are specified, where `variable1` is the noninverting output of the flip-flop and `variable2` is the inverting output. Both of these variables must be assigned, even if one of them is not connected to a primary output pin. For example,
- Checks that values for `variable1` and `variable2` do not match a pin name used in the cell being described.
- Checks that the `data_in` and `enable` attributes are both defined.

Syntax

```
library (lib_name) {
  cell (cell_name) {
    ...
    latch (variable1, variable2) {
      enable : "Boolean_expression" ;
      data_in : "Boolean_expression" ;
      clear : "Boolean_expression" ;
      preset : "Boolean_expression" ;
      clear_preset_var1 : value ;
      clear_preset_var2 : value ;
    }
  }
}
```

Truth Table Validity Check

Liberty NCX performs the following checks on all truth tables.

- Checks that each truth table row has the correct number of input and output values by using a basic syntax check.
- Checks that pin names are unique and correspond to valid cell ports.
- Checks that the truth table is complete; that is, the number of rows represents all possible permutations of the input.

State Table Validity Check

Liberty NCX checks on all state tables and issues either a warning or error, as appropriate, if the following conditions are not met:

- The basic state table syntax is correct, as shown in the following example:

```
statetable( "input node names", "internal node names" )
{
  table : "input node values : current internal values : next internal
  values,
  input node values : current internal values : next internal values";
}
```

As part of this check, Liberty NCX verifies that the number of values per row matches the number of node names, the current internal values are specified, and white spaces exist around each token.

- Each input and internal node has a valid token value. Input nodes have valid values in the following set:

L	Low
H	High
-	Don't care
L/H	Expands to both L and H
H/L	Expands to both H and L
R	Rising edge
F	Falling edge
~R	Not rising edge
~F	Not falling edge

Internal nodes have valid values in the following set:

L	Low
H	High

- Output is not specified
 - L/H Expands to both L and H
 - H/L Expands to both H and L
 - X Unknown
 - N No event from the current value: hold. You should only use N when all asynchronous inputs and clocks are inactive.
- Input and internal node names are unique.
 - A row of an internal node has value N; otherwise, the node is purely combinational.
 - An output has L/H or H/L values without any L/H or H/L input values.
 - An output port in the cell has either of the following attribute combinations:
an `internal_node` attribute, an optional `input_map` attribute and an optional `three_state` attribute or a `state_function` attribute and an optional `three_state` attribute.
 - The `internal_node` value is a valid internal node of the state table.
 - The `state_function` attribute value is a Boolean expression of inputs, inouts, or outputs with only an `internal_node` attribute.
 - The input map does not have more names than the state table.
 - The `internal_node` name in the input map corresponds to the current port name.
 - An input port is not specified as an internal node in the input map.
 - The `three_state` attribute value is not a function of a sequential output.
 - An internal node does not have a `three_state` or `state_function` attribute.
 - The expanded state table contains every permutation of L and H for the all inputs, and warns of overlapping or missing entries.

Sensitization Truth Tables

For combinational cells, sensitization information can be entered in the form of a truth table. This is the truth table syntax:

```
truthtable {
    InP1,      InP2,      ...,      InPm      :  OutP1,      ...,      OutPn ;
    InV11,     InV21,     ...,      InVm1     :  OutV11,     ...,      OutVn1 ;
    InV12,     InV22,     ...,      InVm2     :  OutV12,     ...,      OutVn2 ;
    ...
    ...
    ...
}
```

```

    ...
    InV1k,     InV2k,     . . . . ,     InVmk :     OutV1k,     . . . . ,     OutVnk ;
}
```

The syntax and meaning of the input pin names and output pin names are identical to those of sensitization tables. The allowed characters for input pin values are -, 0, and 1. The allowed characters for output pin values are -, 0, 1, and Z.

Liberty NCX also supports sensitization with multiple truth tables. Multiple truth tables are generally used to represent I/Os. Create separate truth tables to represent the bidirectional pins as an input pin in one and an output pin in the other. In the following example, PAD is a bidirectional pin; therefore, two truth tables are entered in the template file:

```

truthtable {
    A,   EN,   B,   C,   BH : PAD, Z, Z0 ; // PAD as an output pin
    0,   1,   0,   0,   0 :   0,   -,   - ;
}
...
    -,   0,   0,   0,   0 :   Z,   -,   - ;
}

truthtable {
    A,   EN,   B,   C,   BH,   PAD : Z, Z0 ; // PAD as an input pin
    -,   0,   0,   0,   0 :   0,   0 ;
}
...
```

Note that all input and output pins must be specified in both truth tables even if they do not affect each other.

Arc Generation Control

The occurrence of a characterized event is called an arc. For example, for an inverter cell, Liberty NCX characterizes the delay from a rising-edge transition at the input to the falling-edge transition at the output. This characterized event is called a delay arc. Liberty NCX can also determine the amount of energy consumed by this same event, in what is called a propagating power arc.

If an energy-consuming event does not cause an output transition to occur, the event is called a nonpropagating power arc. For example, for a two-input NAND gate, a rising edge on input A while input B is held constant at logic zero does not cause a transition at the output. However, this event can still result in energy consumption as the charge on an internal node of the cell is discharged.

For a given cell, many different types of arcs can be characterized. An arc defined in a library without “when” conditions is called a default arc. For example, for a two-input NAND gate, Liberty NCX could characterize the delay arc from input A to Z while holding B constant at

logic 1, and it could characterize the delay arc from B to Z while holding A constant at logic 1. Since the required side input states are unambiguous, no “when” conditions are needed and default arcs can be used.

An arc defined in a library with “when” conditions is called a state arc because the arc has been captured with the cell in the specified state. For example, for a three-input XOR gate, Liberty NCX could characterize the non-inverting delay arc from input A to Z while holding B and C constant at logic 0, or it could characterize the same delay arc while holding B and C constant at logic 1, or it could characterize both arcs. If both of the arcs between input A and Z are characterized, each arc will have a “when” condition that identifies the state of side input pins B and C for that timing arc behavior. When performing timing analysis, a tool can use the “when” conditions to perform a more accurate analysis when constant values are present.

If an input seed library is provided, Liberty NCX characterizes the arcs as defined in the seed library. If no input library is provided, Liberty NCX characterizes the arcs according to the sensitization information provided in the template files, as explained in [“Sensitization” on page 3-49](#). If no sensitization information is provided in the template file, Liberty NCX determines its own arc sensitization by considering the functional operation of the cell.

ncx_create_arcs Attribute

Instead of allowing NCX to follow its default choices of arc generation, you can explicitly specify which arcs to characterize by using the `ncx_create_arcs` attribute in the library or cell template file. This is the syntax:

```
ncx_create_arcs : cellName fromPin toPin arcType
                  [states "state1" ... "staten"]
                  [states all [pin1 ... pinn]
                  [states default]
                  [ignore "state1" ... "staten"] ;
```

The `cellName` setting is the name of the cell where the attribute is applied. You can specify multiple related cells by using wildcard characters. The `fromPin` setting is the name of one or more source pins for the arc. The `toPin` setting is one or more destination pins of the arc. If the arc involves just a single pin, then the `fromPin` and `toPin` are the same. After characterization, the arcs will appear in the `toPin` pin groups in the library.

The `arcType` setting is the name of the timing or power arc generated. This can be any valid `timing_type` value in Liberty syntax, such as `setup_rising`, `rising_edge`, `preset`, or `clear`, or any one of the following arc-category keywords:

- `delay` – All propagating arcs between the specified pins, including propagating power arcs if the `prop_power` arc type is not specified separately.
- `constraint` – All constraint arcs between the specified pins, including setup, hold, recovery, and removal arcs.
- `leakage_power` – All leakage power models for the specified cell. The “from” and “to” pin specifications should be set to the wildcard character as in the following example:

```
ncx_create_arcs DFF10SL * * leakage_power all ;
```

- `internal_power` – All nonpropagating power models of the input pins. The “from” and “to” pins must be set to the same pin, as in the following example:

```
ncx_create_arcs :MD10SL IN1 IN1 internal_power all ;
```

- `prop_power` – All propagating power models that require sensitization different from models generated by the `delay` arc type.
- `capacitance` – A pin-based or arc-based CCS receiver model that models the capacitance of the input pin. The “from” and “to” pins must be set to the same pin, as in the following example:

```
ncx_create_arcs : MD18P IN1 IN1 capacitance states !EN EN;
```

The `states` parameter specifies a state or a list of states for creating the arcs. Each state is either an explicit list of logical “when” conditions or a state name. A state name is associated with a set of logical “when” conditions defined separately in an `ncx_when`, `ncx_when_rise`, or `ncx_when_fall` statement. Liberty NCX characterizes an arc for each of the listed states.

The `states all` syntax creates all possible arcs, or in other words, it creates a separate arc for each valid sensitization state for the specified arc type, including all possible combinations of side-input pins of the cell. A side-input pin is an input pin that is not the “from” in and is held constant during the characterized event.

If a list of pins is specified with the `all` keyword, Liberty NCX performs characterization at all possible combinations of logic states for the listed pins. For example, the following statement:

```
ncx_create_arcs : cellName fromPin toPin arctype states all pin1 pin2 ;
```

results in the following sensitization states for the two specified side pins:

```
pin1 and pin2
!pin1 and pin2
pin1 and !pin2
!pin1 and !pin2
```

The `states default` syntax creates only a single default arc without “when” conditions in the library. The sensitization used for characterizing the default arc is determined by the `default_arc_mode` attribute setting as described in [Table 3-1](#). For example, if you specify:

```
ncx_create_arcs : my_cell IN1 OUT2 delay states default ;
```

and `default_arc_mode` is set to `worst`, Liberty NCX finds the sensitization that produces the worst delay from the pin `IN1` to `OUT2` in `my_cell`, and writes that arc to the library as the default arc, without specifying any “when” conditions in the library.

If the `default_arc_mode` attribute is not set, Liberty NCX uses the `first` setting and characterizes the first valid sensitization that it finds and ignores other possible sensitizations for the arc. A valid sensitization is an input vector that toggles only the “from” pin and causes a transition on the “to” pin of the cell.

The `ignore` syntax causes characterization to be skipped for the specified list of states.

The conditional characterization `ncx_condition` group includes the `arcType` capacitance. This extension to the conditional group for capacitance is made only to allow a specification of the `ncx_when` condition. Other attributes cannot be specified in an `ncx_condition` group for `arcType` capacitance.

```
ncx_condition cap_cond {
    ncx_condition_pin : D0 ;
    ncx_condition_arc_type : capacitance ;
    ncx_when : !D1 ;
}
```

The following examples demonstrate usage of the `ncx_create_arcs` attribute.

To generate delay arcs for all states, between all pins, for all cells named `MX3*`:

```
ncx_create_arcs : MX3* * * delay states all ;
```

To synthesize all internal power arcs for the input pin `A`, for all cells named `MC*`:

```
ncx_create_arcs : MC* A A internal_power states all ;
```

To model `non_seq_setup` and `non_seq_hold` arcs for all states, between all pins, for all cells named `MD*`:

```
ncx_create_arcs : MD* * * non_seq* states all ;
```

To ignore all delay arcs terminating at the pin NQ of the cell MD10SL:

```
ncx_create_arcs : MD10SL * NQ delay ignore all ;
```

To model only a single default delay arc for all paths terminating at the pin Q of cell MD10SL:

```
ncx_create_arcs : MD10SL * Q delay states default ;
```

To acquire the minimum pulse width only for two specified states for the pin CK, in all cells named MD1*:

```
ncx_create_arcs : MD1* CK CK min_pulse_width states \
                  "S & R & NT & DT & D" \
                  "S & R & !NT & DT & D";
```

Partial Sensitization

The `default_arc_mode` attribute specifies how to sensitize a cell to generate a default arc, which is the arc that is written to the library without “when” conditions and is used by default by the tool that is performing timing or power analysis. The `default_arc_mode` attribute can be set to any of the following:

```
first    best    worst
worst_delays best_delays
worst_points best_points
worst_edges  best_edges
```

The default behavior is `first`, which means to use the first valid sensitization found, which could be any valid sensitization. The setting `worst` causes Liberty NCX to find the characterization that produces the worst (longest) delay, which is the most conservative result for maximum-delay analysis. For more information about the available settings, see [Table 3-1 on page 3-3](#).

To create a state arc, which characterizes the cell behavior with the inputs in a specified state, use `ncx_create_arcs` with the `states` attribute, and specify the exact states.

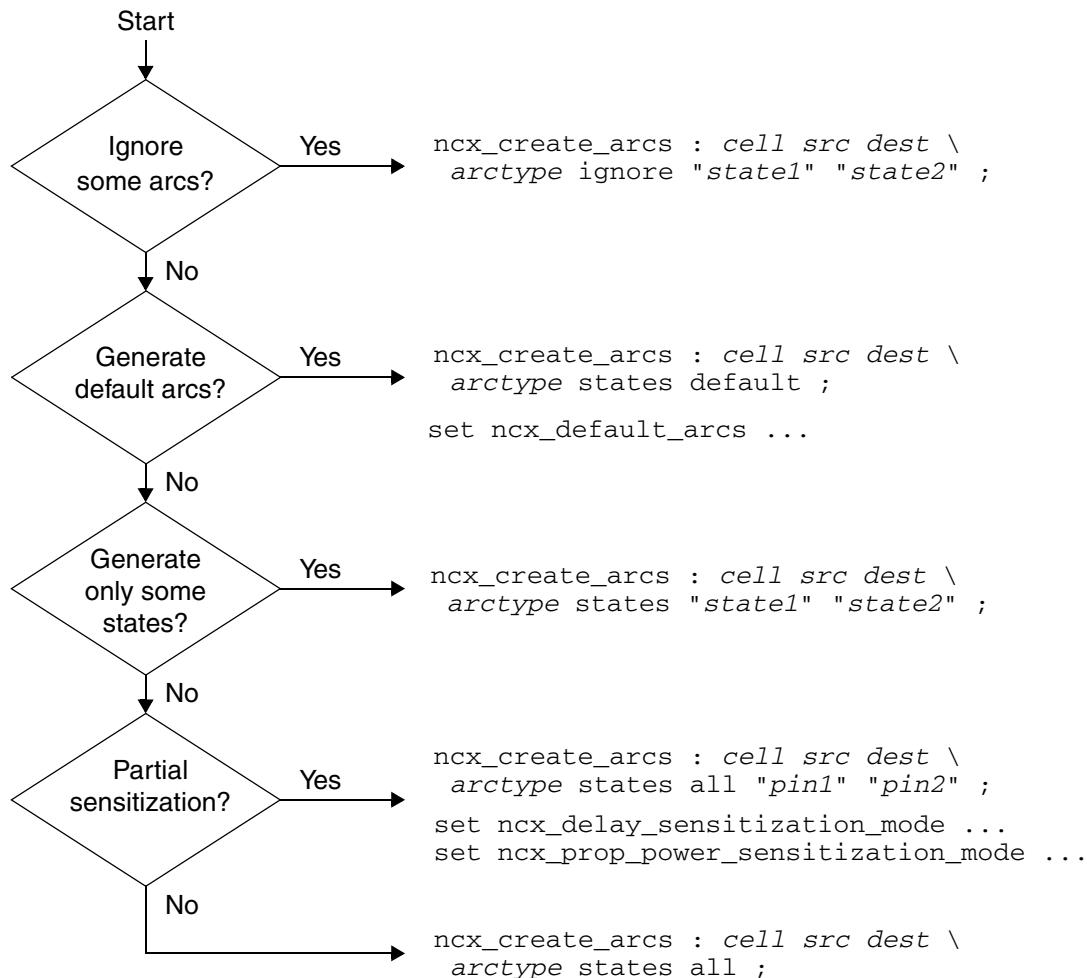
If you specify the states of some input pins but not others, it is called partial sensitization. In that case, Liberty NCX must decide the states of the unspecified input pins. By default, it chooses the first valid set of states it finds for the unspecified input pins and ignores any other possible states.

You can control the selection of the remaining unspecified input pins by using the `ncx_delay_sensitization_mode` attribute, which can be set to `first`, `best`, or `worst`. The default is `first`, which uses first valid sensitization found, which could be any valid sensitization. A setting of `best` chooses the sensitization that results in the least delay, or a setting of `worst` chooses the sensitization the results in the longest delay.

A similar attribute, `ncx_prop_power_sensitization_mode`, lets you control the sensitization for propagated power characterization. It can also be set to `first`, `best`, or `worst` to select the first valid sensitization found, the sensitization that results in the least energy consumption, or the sensitization the results in the most energy consumption.

Figure 3-3 summarizes the `ncx_create_arcs` usage flow.

Figure 3-3 ncx_create_arcs Usage Flow



Delay Arcs

The `delay` arc type setting applies the attribute to all delay timing arcs between the specified pins and cells.

To generate all delay arcs for all cells and all pins:

```
ncx_create_arcs : * * * delay states all ;
```

To model all delay arc states between all pins, for all cells named MX3*:

```
ncx_create_arcs : MX3* * * delay states all ;
```

To generate delay arcs under the “when1” and “when2” conditions:

```
ncx_create_arcs : * * * delay states "when1" "when2" ;
```

Propagating Power Arcs

Internal power is any power dissipated within the boundary of a cell, including both propagating power and nonpropagating power. Liberty NCX performs separate simulations to calculate the propagating and nonpropagating power of a cell.

Nonpropagating power is the power dissipated in the circuit when the switching of one or more inputs does not cause any output to change. For example, in a two-input AND gate, when one of the inputs is set to logic zero, any transition at the other input does not cause a transition at the output, but there is switching activity at the internal nodes of the circuit, thereby causing power to be dissipated in the cell. This energy is captured and represented in the Liberty file as `internal_power` under the specific input pin group.

Propagating power is the power dissipated in the circuit during the time when one or more inputs are switching and one or more outputs also switch as a result. The power dissipated in this situation is written in the Liberty file as `internal_power` under the output pins group.

By default, the simulations used to determine delay timing are also used to determine propagating power. Accordingly, the characteristics of the timing arcs characterized also influence the propagating power characterization. However, you can use the `prop_power` arc type to separately specify the arcs to be generated for propagating power. A `prop_power` arc specification takes precedence over the delay arc states for the specified `cellName`, `fromPin`, and `toPin` values.

For example, the following statement generates arcs under the “when1” and “when2” conditions for both timing and propagating power:

```
ncx_create_arcs : * * * delay states "when1" "when2" ;
```

The following two statements generate delay arcs only under “when1” conditions and propagating power arcs only under “when2” conditions:

```
ncx_create_arcs : * * * delay states "when1" ;
ncx_create_arcs : * * * prop_power states "when2" ;
```

The following two statements generate delay and propagating power arcs only under “when1” conditions, except for CK-to-Q propagating power arcs, which are generated under “when2” conditions:

```
ncx_create_arcs : * * * delay states "when1" ;
ncx_create_arcs : * CK Q prop_power states "when2" ;
```

The following statement, by itself, generates propagating power arcs only under “when2” conditions:

```
ncx_create_arcs : * * * prop_power states "when2" ;
```

Because the `delay` arc type is not specified, the default delay arcs are generated.

The following example ignores timing arcs with the “when1” condition but generates power arcs only under “when1” condition:

```
ncx_create_arcs : * * * delay ignore "when1" ;
ncx_create_arcs : * * * prop_power states "when1" ;
```

The following example creates all timing arcs but no power arcs at all.

```
ncx_create_arcs : * * * delay states all ;
ncx_create_arcs : * * * prop_power ignore all ;
```

The following example creates all timing arcs but only the default power arcs.

```
ncx_create_arcs : * * * delay states all ;
ncx_create_arcs : * * * prop_power states default ;
```

Constraint Arcs

The following examples demonstrate how to specify the generation of constraint arcs using the `ncx_create_arcs` attribute. Constraint arcs are setup, hold, recovery, and removal timing arcs.

To generate the main (setup and hold) constraint arcs between all input and output pin pairs:

```
ncx_create_arcs : * * * constraint states all ;
```

To separately configure preset arcs and constraint arcs:

```
ncx_create_arcs : * * * preset states all ;
```

To separately configure clear arcs and constraint arcs:

```
ncx_create_arcs : * * * clear states all ;
```

To separately configure minimum pulse width arcs and constraint arcs:

```
ncx_create_arcs : * * * min_pulse_width states all ;
```

To generate all `non_seq_setup` and `non_seq_hold` arc states between all pins for all cells:

```
ncx_create_arcs : * * * non_seq* states all ;
```

Internal Leakage Power Arcs

To characterize cells for internal leakage power using the `ncx_create_arcs` attribute, set the `fromPin` and `toPin` options to the asterisk (*) wildcard character. The following examples demonstrate how to specify the states that are characterized.

To exhaustively simulate all possible leakage power states for a cell:

```
ncx_create_arcs : cell_name * * leakage_power states all ;
```

To simulate only the first valid state found for internal leakage power for a cell:

```
ncx_create_arcs : cell_name * * leakage_power states default ;
```

To explicitly specify the states for which to acquire the internal leakage power for a cell:

```
ncx_create_arcs : cell_name * * leakage_power states \
    "A1&A2&B&C" "A1&A2&!B&C" ;
```

Internal Power Arcs

The `internal_power` arc type setting applies the attribute to all nonpropagating power models of the input pins. Internal power is any power dissipated within the boundary of a cell, consisting of propagating power and nonpropagating power.

For example, to synthesize all internal power arcs for input pin A, for all cells named MC*:

```
ncx_create_arcs : MC* A A internal_power states all ;
```

Capacitance Extraction

A capacitance state that does not overlap a timing state is published as a pin-based CCS receiver model. A capacitance state that overlaps a timing state is published as an arc-based CCS receiver model in the timing section.

The following example creates two user-specified capacitance states – one that does not overlap with a timing state, and one that does overlap with a timing state:

```
ncx_create_arcs : * A A capacitance states !EN EN ;
```

The corresponding library data is as follows:

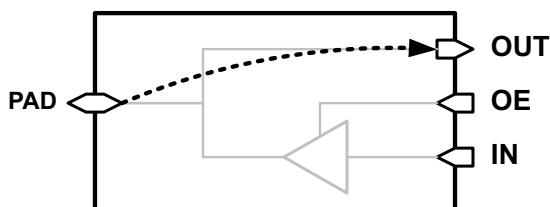
```
pin (A) {
    fall_capacitance : value;
    fall_capacitance_range(value,value);
    capacitance : value;
    rise_capacitance : value;
    rise_capacitance_range(value,value);
    receiver_capacitance () {
        when : "!EN"
        receiver_capacitance1_rise
        receiver_capacitance2_rise
        receiver_capacitance1_fall
        receiver_capacitance2_fall
    }
}
pin (Z) {
    timing () {
        related_pin : A;
        when : "EN";
        .....
        .....
        receiver_capacitance1_rise
        receiver_capacitance2_rise
        receiver_capacitance1_fall
        receiver_capacitance2_fall
    }
}
```

Bidirectional Cell Arcs

Bidirectional cells have tristate inout pins, with cell delay arcs leading to and from the tristate pin. The inout pin can either be actively driven to a logic value by the bidirectional cell so that the cell is a net driver, or placed in the high impedance state (tristate condition) so that the pin is a net receiver.

Consider the typical bidirectional cell shown in [Figure 3-4](#), with an inout pin PAD and an active-high control pin OE.

Figure 3-4 Typical Bidirectional Cell



When OE is driven high, the inout pin is actively driven by the cell's IN-to-PAD internal driver. When OE is driven low, the inout pin is placed in the tristate condition by the cell. The PAD pin declaration in this cell's template file is as follows:

```
pin PAD {
    direction : inout ;
    function : IN ;
    three_state : "OE'" ;
    ...
}
```

When the PAD-to-OUT arc is sensitized, the arc sensitization could

- allow data to come from the cell's IN pin by setting OE high
- allow data to come from external net drivers by setting OE low

The `ncx_input_tristate_mode` template attribute specifies which driver condition should be used for arcs driven by such inout pins. This attribute is used to control whether internal feedback paths through the bidirectional cell are considered for the inout pin when it is used as an input.

The valid values of the `ncx_input_tristate_mode` template attribute are `disable` (the default) and `enable`. The `disable` value disallows the tristate condition during sensitization, so that the inout pin is always actively driven by the cell. The `enable` value requires sensitization of the tristate condition, so that the inout pin is always driven by external net drivers.

By default, Liberty NCX considers the following state-dependent arcs during sensitization.

```
...
creating delay arcs...
combinational (positive_unate) PAD -> OUT when !IN&OE
combinational (positive_unate) PAD -> OUT when IN&OE
...
```

Note that the tristate control pin OE is driven high, so that the PAD pin is driven by the cell's IN-to-PAD internal driver.

If the cell template is updated to use the following attribute setting:

```
ncx_input_tristate_mode : enable ;
```

then the following arcs are created so that the PAD pin is placed in the tristate condition by the cell, and the PAD-to-OUT arc is driven by an external net driver.

```
...
creating delay arcs...
combinational (positive_unate) PAD -> OUT when !IN&!OE
combinational (positive_unate) PAD -> OUT when IN&!OE
...
```

The `ncx_input_tristate_mode` template attribute affects arcs where Liberty NCX must determine the sensitization. It does not affect arcs that have explicitly supplied sensitization or when conditions in the seed library or input template files.

Half-Unate Arcs

A half-unate arc is defined by the `timing_type` attribute to have an output pin that is exclusively rising or falling. You can specify to sensitize a half-unate arc by using the `ncx_create_half_unate_arcs` template attribute. You can use the `ncx_create_half_unate_arcs` template attribute to generate default power arcs during power characterization. When this attribute is set to `true`, power arcs are generated, and when it is set to `false`, it prevents them from being generated. The default is `false`.

By default, the arc definition created from the first sensitization phase cannot be modified. Setting the `ncx_create_half_unate_arcs` template attribute to `true` activates this sensitization, which is indicated by messages in the log file as shown in the following example. Liberty NCX only changes arc definitions if the sensitization indicates that the arc behavior is half-unate.

The power arc control provided by the `ncx_create_half_unate_arcs` template attribute is applicable only to arcs that are created by Liberty NCX during sensitization; it is not applicable to arcs that are input through a seed library or are user-specified in an input template file.

For arcs that are full unate and are not sensitized for both transitions on the output pin, simulation generates either rise or fall delay and constraint subgroups. The missing timing subgroups of these arcs are then populated by copying the generated group during the Liberty NCX library assembly phase to enable correct compilation by Library Compiler and use by downstream tools. The copied groups in the output library are identified by annotated comments. This control enables you to generate libraries that can be successfully used by downstream tools, some of which might also be able to handle half-unate arc definitions, backward-compatibility with older libraries, and the prevention of copied data for invalid output transitions.

The `ncx_create_half_unate_arcs` template attribute can be applied at the library, cell, pin, or arc level (by using `ncx_condition` groups) in the template file.

Example

The following shows examples from the Liberty NCX log file of a cell that is half-unate:

```
...
--- generating sensitizations...
cell lib1::cell1 (1 of 1) (5in/1out/1io/6int pins)
...
 80 sensitization vectors created
```

```

creating delay arcs...
    three_state_enable (negative_unate) IN -> IO when !CHDRV
    ...
10 delay arcs created
    ...
sensitizing cell1::negative_unate three_state_enable arc 0: IN -> IO
(!CHDRV):
Warning: could not sensitize cell for a rise transition on arc
fall: CFO&!CHDRV&!IE&NRST (T11ZNRBN1C1F4ZSE_wave_0_2)
    ...

```

The output library has the following arc structure:

```

library (lib1) {
    ...
cell (cell1) {
    ...
pin (IO) {
    ...
        timing () {
            related_pin : "IN";
            timing_type : "three_state_enable";
            timing_sense : "negative_unate";
            when : "!CHDRV";
            sdf_cond : "CHDRV==1'b0";
            cell_fall ("del_1_4_2") {
                index_1("0.05, 2.5");
                index_2("2.90093, 10.9009");
                values("1.8639266, 3.6666938", \\
                    "2.0545514, 3.8577550");
            }
            fall_transition ("del_1_4_2") {
                index_1("0.05, 2.5");
                index_2("2.90093, 10.9009");
                values("0.8176734, 2.7833569", \\
                    "0.8176241, 2.7825988");
            }
            cell_rise ("del_1_4_2") {
                index_1("0.05, 2.5");
                index_2("2.90093, 10.9009");
                values("1.8639266, 3.6666938", \\
                    "2.0545514, 3.8577550");
            }
            rise_transition ("del_1_4_2") {
                index_1("0.05, 2.5");
                index_2("2.90093, 10.9009");
                values("0.8176734, 2.7833569", \\
                    "0.8176241, 2.7825988");
            }
        }
    ...
}
    ...
}

```

```

    }
...
}

```

If Liberty NCX is run on the same cell with the `ncx_create_half_unate_arcs` template attribute set to `true`, the following results occur:

```

...
--- generating sensitizations...
cell lib1::cell1 (1 of 1) (5in/1out/1io/6int pins)
...
    80 sensitization vectors created
creating delay arcs...
    three_state_enable (negative_unate) IN -> IO when !CHDRV
...
10 delay arcs created
...
sensitizing cell1::negative_unate three_state_enable arc 0: IN -> IO
(!CHDRV):
Warning: could not sensitize cell for a rise transition on arc fall:
CFO&!CHDRV&!IE&NRST (T11ZNRBN1C1F4ZSE_wave_0_2)
changing arc definition to three_state_enable_fall
...

```

The output library now has the following structure:

```

library (lib1) {
...
cell (cell1) {
...
pin (IO) {
...
    timing () {
        related_pin : "IN";
        timing_sense : "negative_unate";
        when : "!CHDRV";
        sdf_cond : "CHDRV==1'b0";
        timing_type : "three_state_enable_fall";
        cell_fall ("del_1_4_2") {
            index_1("0.05, 2.5");
            index_2("2.90093, 10.9009");
            values("1.8639266, 3.6666938", \\
                   "2.0545514, 3.8577550");
        }
        fall_transition ("del_1_4_2") {
            index_1("0.05, 2.5");
            index_2("2.90093, 10.9009");
            values("0.8176734, 2.7833569", \\
                   "0.8176241, 2.7825988");
        }
    }
}
...

```

```

    }
    ...
}
...
}
```

Characterization Index Values

The result of characterization is a library containing tables of values that specify the behavior of the cell. Each table specifies a cell parameter as a function of certain variables, such as delay as a function of output load capacitance and input transition time. The library and cell template files can explicitly specify the index values of the variables used in the characterization tables.

In the library template, one or more sets of index values can be specified for each variable that might form the axis of a lookup table or a CCS model. The syntax for specifying these sets is

```
ncx_variable_index : value1 value2 value3 ... ;
```

The name of the variable used is the same as that used in the Liberty format. For example, to specify a set of input net transition times to be used as `input_net_transition` values in Liberty syntax, the entry is

```
ncx_input_net_transition_index : 0.01 0.040019 0.145393 \
                                0.34662 0.660122 1.1 ;
```

Multiple sets can be specified by with a numerical suffix after the set name, as in the following example:

```
ncx_input_net_transition_index : \
    0.009 0.015 0.0255 0.0465 0.0885 0.174 0.345 ;
ncx_total_output_net_capacitance_index : \
    1.165 2.796 6.291 13.048 26.795 54.289 109.51 ;
ncx_total_output_net_capacitance_index_1 : \
    2.33 5.592 12.582 26.096 53.59 108.578 219.02 ;
ncx_total_output_net_capacitance_index_2 : \
    4.66 11.184 25.164 52.192 107.18 217.156 438.04 ;
```

Each cell template can then reference these sets as needed for the specific model, as shown in the following example:

```

...
pin A {
    direction : input ;
    fanout_load : 0.1990000 ;
    rise_capacitance : 4.1654300 ;
    fall_capacitance : 3.7821700 ;
```

```

}
pin Y {
    direction : output ;
    function : (A)' ;
    timing {
        related_pin : A ;
        timing_sense : negative_unate ;
        ncx_rise_input_net_transition_index : 0 ;
        ncx_fall_input_net_transition_index : 0 ;
        ncx_rise_total_output_net_capacitance_index : 1 ;
        ncx_fall_total_output_net_capacitance_index : 1 ;
    }
}
...

```

The base list with no suffix is treated as having the suffix zero. The suffix “_0” can be optionally used on the base name.

Liberty NCX supports the specification of index values for the following variables:

```

input_net_transition
total_output_net_capacitance
related_pin_transition
constrained_pin_transition

```

The meaning and application of each of these variables can be found in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

The cell template might contain references to sets of indexes to be used in characterizing the cell. The syntax for specifying the set of index values follows this general format:

```
ncx_[timing_type]_[rise|fall]_variable_index : set_id ;
```

where the optional terms (shown in square brackets) allow fine-grained control over the index values to be used for arcs of different timing types and transition types.

The *set_id* refers to the appropriate index set of the variable type specified at the library level. The *set_id* can be 0 to specify the default variable set or nonzero to specify a different defined set.

The index can be specified at the cell level, pin level, or arc level, with the lowest level overriding any higher-level specification. For example, arc-level index values override cell-level or pin-level attributes.

Similarly, the specification of the index can be as specific or general as desired, with the optional use of the timing type and transition type in the parameter name. For example, among the following attributes specified within the same group (for example, at the cell level):

```
ncx_setup_rise_constrained_pin_transition_index : 1 ;  
ncx_setup_constrained_pin_transition_index : 0 ;  
ncx_constrained_pin_transition_index : 0 ;
```

The more specific set of index values has higher priority. Thus, the `setup_rise` index values override the `setup` index values, and the `setup` values have priority over the general `constrained_pin` index values.

However, if a general specification such as

```
ncx_constrained_pin_transition_index : 0 ;
```

is used for a specific pin or timing arc within a cell, it overrides any specification made at the cell level.

If the cell template does not contain a specification for a particular table variable, the default index is used (the value corresponding to the zero-valued index set).

When Liberty NCX generates templates from a seed library, it writes the library-level characterization index sets into a separate file with the extension .indexes and includes that file in the library template file. This enables easy viewing and modification of the index values for further characterization flows.

Percentage-of-Maximum Index Values

You can specify the index values as a percentage of the maximum parameter value, rather than specify explicit values. This lets you easily adjust index values automatically for different cells, based on the desired range, for example, the maximum capacitance value of `total_output_net_capacitance`:

```
ncx_pct_total_output_net_capacitance : 10 30 50 90 100
```

This specifies the index values as percentages of the maximum capacitance of the cell. The maximum capacitance must either exist in the cell already, or sufficient information must exist to allow Liberty NCX to extract the value, as described in “[Maximum Capacitance Acquisition](#)” on page 3-87.

Number and Spacing of Index Values

Liberty NCX can generate a list of index values when you specify the minimum index, maximum index, total number of index values, and spacing mode (`linear`, `log`, or `auto`). It generates the list of values automatically when you provide these four parameters.

For example, you can specify the output net capacitance and input net transition index values as follows:

```

...
ncx_min_total_output_net_capacitance : 0.0005 ;
ncx_max_total_output_net_capacitance : 0.205942 ;
ncx_size_total_output_net_capacitance : 7 ;
ncx_mode_total_output_net_capacitance : log ;

ncx_min_input_net_transition : 0.010 ;
ncx_max_input_net_transition : 1.1 ;
ncx_size_input_net_transition : 7 ;
ncx_mode_input_net_transition : log ;

ncx_max_output_transition_time : 0.8 ;
...

```

Liberty NCX expands these template examples into the following sets of index values:

```

ncx_total_output_net_capacitance_index : \
0.0005 0.00136385 0.00372016 0.0101475 \
0.0276792 0.0755005 0.205942 ;
ncx_input_net_transition_index : \
0.01 0.0218893 0.0479142 0.104881 0.229577 0.502528 1.1 ;

```

It expands each specified range of capacitance or transition times into a list of the specified number of values, spaced linearly or logarithmically from the minimum to the maximum value. The minimum, maximum, size (number of index values), and the expansion mode must all be available for each parameter.

Adaptive Selection of Index Values

If you select `auto` (rather than `linear` or `log`) as the type of index value spacing for output capacitance or input transition time, Liberty NCX adaptively selects the index values to achieve higher accuracy while using fewer index values. It uses more index values, spaced more closely, where the delay is a nonlinear function of output capacitance or input transition time. Likewise, it uses fewer index values, spaced farther apart, where the delay is nearly a linear function of output capacitance or input transition time.

For example, the following template settings invoke adaptive selection of both output capacitance and input transition index values.

```

...
ncx_min_total_output_net_capacitance : 0.0005 ;
ncx_max_total_output_net_capacitance : 0.205942 ;
ncx_size_total_output_net_capacitance : 12 ;
ncx_mode_total_output_net_capacitance : auto ;

```

```

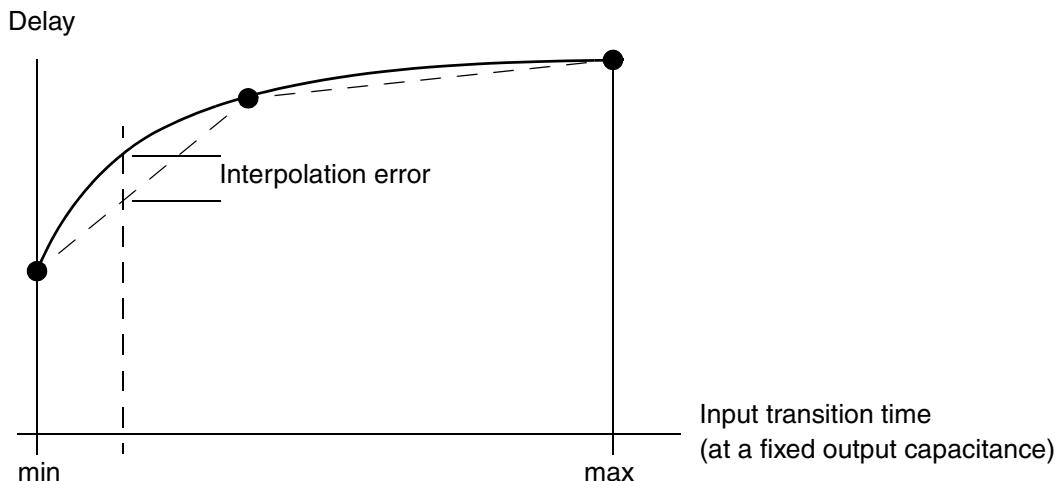
ncx_min_input_net_transition : 0.010 ;
ncx_max_input_net_transition : 1.1 ;
ncx_size_input_net_transition : 12 ;
ncx_mode_input_net_transition : auto ;

ncx_autoindex_tol_pct : 2
...

```

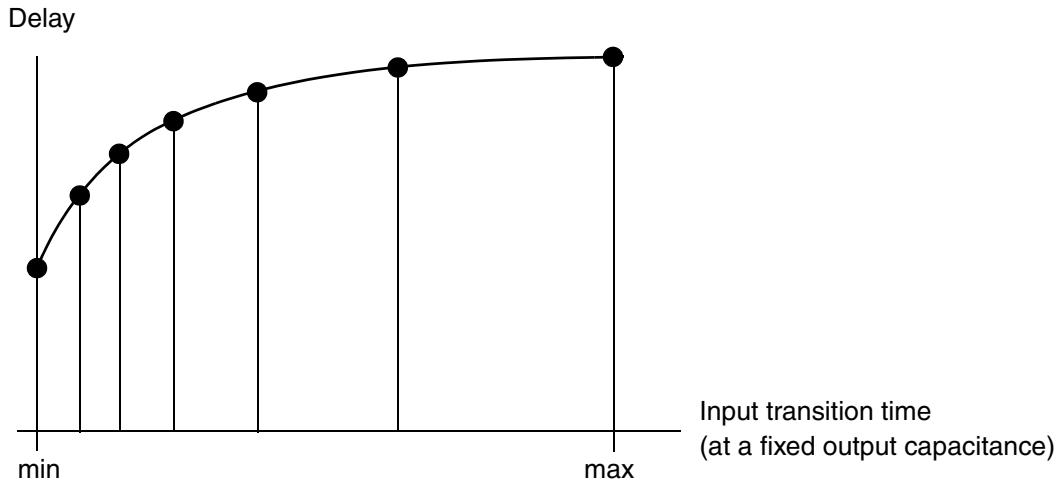
In this example, Liberty NCX selects a set of output capacitance and input transition index values that result in an interpolation error of no more than 2 percent throughout the characterization range, using as few index values as possible, but with a maximum limit of 12 output capacitance index values and 12 input transition values. Interpolation error is the difference between the delay values calculated by curve-fitting between index values and by linear interpolation between index values, as illustrated in [Figure 3-5](#).

Figure 3-5 Interpolation Error



In this example, Liberty NCX might select eight input transition time index values to achieve an interpolation error of no more than 2 percent throughout the characterization range, spaced more closely where the curve is least linear, as shown in [Figure 3-6](#).

Figure 3-6 Adaptively Selected Input Transition Index Values



Liberty NCX similarly selects an appropriate number of output capacitance index values to achieve no more than 2 percent interpolation error throughout the characterization range.

Instead of specifying a relative percentage tolerance, you can specify an absolute tolerance in time units. These are the attributes you can set:

```
ncx_autoindex_tol : absolute_tolerance_in_time_units
ncx_autoindex_tol_pct : relative_tolerance_in_percent
```

If you specify both absolute and relative requirements, Liberty NCX attempts to satisfy both requirements. The absolute tolerance can be specified in scientific notation, for example, `1e2` or `4.4e02`.

For adaptive index value selection, you must explicitly specify the minimum value and the maximum value for the input transition time and for the output capacitance, with the exception that the maximum output capacitance can be determined automatically as described in the next section, “[Maximum Capacitance Acquisition](#).”

For adaptive index value selection, specifying the total number of index values and specifying the accuracy tolerance are both optional:

- If you specify both a size limit and a tolerance, Liberty NCX selects as few index values as possible while meeting the tolerance constraints, up to the specified size limit.
- If you specify only a tolerance without a size limit, Liberty NCX selects as few index values as possible while meeting the tolerance constraints, up to a maximum of 20 index values. In the absence of explicit limits, you could have up to 20 by 20 (400) characterization points.

- If you specify only a size limit without a tolerance, Liberty NCX uses exactly the specified number of index values and spaces them to minimize the interpolation error.
- If you specify neither a size limit nor a tolerance, Liberty NCX uses seven index values and spaces them to minimize the interpolation error.

For a propagating power arc, Liberty NCX uses the same input slew and output load indexes that were used for delay characterization if the “when” conditions of the delay arc match those of the power arc. If there is no delay arc, or if the “when” conditions of the delay arc do not match those of the power arc, Liberty NCX runs a delay simulation to get the power data.

For a nonpropagating power arc, which involves only a single input pin, Liberty NCX chooses the same input slew index values that were used for one of the delay arcs originating at the same input pin, if available.

Adaptive selection of index values can be used only for characterization of delay as a function of input transition time and output capacitance. For other characterized parameters, select either `linear` or `log` for the spacing mode.

Three-Dimensional Lookup Tables

Liberty NCX supports three-dimensional lookup tables for delay and constraint arcs, in both the seed library and template file flows.

Three-dimensional NLDM timing arc tables are supported when the following three index types are specified, in any index order:

- `ncx_input_net_transition_index`
- `ncx_total_output_net_capacitance_index`
- `ncx_related_out_total_output_net_capacitance_index` (related output)

Three-dimensional NLPM internal propagating power arc tables are supported when the following three index types are specified, in any index order:

- `ncx_input_transition_time_index` or `ncx_input_net_transition_index`
- `ncx_total_output_net_capacitance_index`
- `ncx_related_out_total_output_net_capacitance_index` (related output)

Three-dimensional constraint arc tables are supported when the following three index types are specified, in any index order:

- `ncx_constrained_pin_transition_index`
- `ncx_related_pin_transition_index`
- `ncx_total_output_net_capacitance_index` (related output)

When a three-dimensional delay arc table is created for a cell with two outputs, the remaining output pin is automatically used for the related output pin. If a cell has more than two outputs, the related output pin used for the third index dimension can be explicitly specified with the `related_output_pin` attribute. If a related output is not specified for such a cell, the related output is chosen at random.

Similarly, when a three-dimensional constraint arc table is created for a cell with one output, that output pin is automatically used for the related output pin. If a cell has more than one output, the related output pin used for the third index dimension can be explicitly specified with the `related_output_pin` attribute. If a related output is not specified for such a cell, the related output is chosen at random.

Note:

When a table includes an additional related output pin, the index values for that related output pin's index values must be explicitly defined, either with a discrete list of index values, or by using the `linear` or `log` index generation methods. They cannot be auto-generated on a per-arc basis using the `auto` adaptive index value selection method.

Automatic Creation of Three-Dimensional Lookup Tables

By default, three-dimensional lookup tables are only created when you explicitly specify a third index in the cell template definition. However, if you set the `ncx_create_3d_table` template attribute to `true`, a third index type does not need to be specified. Instead, Liberty NCX automatically creates three-dimensional tables where possible for delay, power, and constraint arcs. The related output pin is identified, and the index values used for its arcs are also used for its role as a third index value.

When the `ncx_create_3d_table` attribute is set to `true`, single-index minimum pulse width constraint arcs are also promoted to two-dimensional constraint arcs as follows:

- `ncx_constrained_pin_transition_index`
- `ncx_total_output_net_capacitance_index` (related output)

Maximum Capacitance Acquisition

Liberty NCX can automatically extract the value of the maximum capacitance that can be driven by an output pin of a cell without violating a specified maximum output signal transition time. Liberty NCX performs this acquisition if the following attributes are set in the cell or library template file:

```
ncx_max_output_transition_time
ncx_min_total_output_net_capacitance
```

and the following attribute is not present in the cell template file:

```
ncx_max_total_output_net_capacitance
```

The `ncx_max_output_transition_time` attribute establishes the bound on the transition time of the signal at the output pin. The `ncx_min_total_output_net_capacitance` attribute establishes the lower bound on the load capacitance of the output pin.

If the `ncx_max_total_output_net_capacitance` attribute is present in the cell template file, Liberty NCX assumes that the maximum output net capacitance value has been previously acquired or specified. In that case, it uses the specified value and does not attempt to acquire a new maximum capacitance value.

For example, the following set of attribute settings triggers automatic acquisition of the maximum total output net capacitance:

```
...
ncx_min_total_output_net_capacitance : 0.0005 ;
ncx_size_total_output_net_capacitance : 7 ;
ncx_mode_total_output_net_capacitance : log ;

ncx_min_input_net_transition : 0.010 ;
ncx_max_input_net_transition : 1.1 ;
ncx_size_input_net_transition : 7 ;
ncx_mode_input_net_transition : log ;

ncx_max_output_transition_time : 0.8 ;
...
```

Liberty NCX finds the maximum capacitance value, sets the `ncx_max_total_output_net_capacitance` attribute to that value, and generates the list of index values as described in the previous section.

By default, Liberty NCX determines the maximum capacitance value by considering only the first arc state characterized at each output. For increased accuracy at the cost of more runtime, you can have Liberty NCX determine the maximum capacitance for all timing arc states. To do so, set the `ncx_max_cap_mode` attribute to `all` in the library or cell template file. If you set this attribute to `first`, you get the default behavior.

The scope of the maximum capacitance acquisition value acquired by Liberty NCX depends on the scope of the maximum output transition time attribute and minimum output capacitance attribute. If either of these attributes is specified in the output pin group in the cell template, the maximum capacitance is acquired as the minimum value of the maximum output load capacitance over all the timing arcs associated with the output pin, driven by the maximum input transition time, such that the maximum output transition time is never exceeded. The resulting capacitance value is then annotated on the output pin's maximum total capacitance attribute.

If neither of these two attributes is set in the output pin group in the cell template, and if both attributes are specified at the cell level or library level, the maximum capacitance is acquired as the minimum value of the maximum load capacitance of all the output pins of the cell, where the maximum capacitance value of each output pin is acquired as described earlier. The resulting capacitance value is annotated on the cell's maximum total capacitance attribute.

Liberty NCX uses a bisection-based optimization method and runs a set of simulations to determine the value of maximum load capacitance that would ensure that the specified maximum value of output transition time is not exceeded. The starting point of this optimization is the value of the minimum output capacitance attribute setting, and the initial maximum value is determined by a set of heuristics inside Liberty NCX that considers an estimate of the driving impedance of the output pin, the clock width, and so on. Consequently, it is essential that the specified minimum capacitance value be less than the expected value of the maximum capacitance. Otherwise, the optimization will fail.

The optimization simulations are run on the compute farm, if specified, or on the local machine, depending on the value of the Liberty NCX configuration file settings.

Input NLDM Capacitance Index Values

Set the following attributes in the cell template file to specify the index used for capacitance models and to control NLDM capacitance extraction:

```
ncx_capacitance_input_net_transition_index : index ;  
ncx_capacitance_total_output_net_capacitance_index : index ;  
ncx_capacitance_rise_input_net_transition_index : index ;  
ncx_capacitance_rise_total_output_net_capacitance_index : index ;  
ncx_capacitance_fall_input_net_transition_index : index ;  
ncx_capacitance_fall_total_output_net_capacitance_index : index ;
```

The values of the attributes correspond to explicit indexes set in the library template, and they represent the indexes to be used for calculating NLDM pin capacitance. The method of calculation is determined by the `input_cap_mode` attribute, which can specify `max`, `min`, or `average`.

To specify indexes for capacitance models that are different from delay models, you must specify both the `ncx_capacitance_input_net_transition` and `ncx_capacitance_total_output_net_capacitance` indexes for pins associated with propagating arcs. Otherwise, the delay model indexes are used. For pins not associated with propagating arcs, it is necessary that you specify the `input_net_transition` index only. If you want to use the delay model indexes for capacitance acquisition also, do not specify the `ncx_capacitance*` attributes. In this case, Liberty NCX simultaneously acquires delay and capacitance models, as shown in the following example.

Example

If the library template contains the following definitions,

```
ncx_input_net_transition_index : 0.012 0.02 0.034 \
    0.062 0.118 0.232 0.46 ;
ncx_input_net_transition_index_1 : 0.03 0.04 0.06 0.1 0.2 0.4 ;
ncx_total_output_net_capacitance_index : 0.001165 0.002796 0.006291 \
    0.013048 0.026795 0.054289 0.10951 ;
ncx_total_output_net_capacitance_index_1 : 0.00233 0.005592 0.012582 \
    0.026096 0.05359 0.108578 0.21902 ;
```

and the cell template contains the following settings,

```
pin Q {
    direction : output ;
    function : IQ ;
    ncx_rising_edge_rise_input_net_transition_index : 0 ;
    ncx_rising_edge_fall_input_net_transition_index : 0 ;
    ncx_rising_edge_rise_total_output_net_capacitance_index : 0 ;
    ncx_rising_edge_fall_total_output_net_capacitance_index : 0 ;
    ncx_capacitance_input_net_transition_index : 1 ;
}
```

The capacitance model tables use the second set of input transition values, and the delay tables use the first set.

Note:

By default, the NLDM pin capacitance ranges are not published in the output library.

Variation-Aware Index Values

In a merged variation-aware library characterization using different sets of parameter values, you can optionally specify fewer index values for the off-nominal timing data tables, thereby reducing the library size. The tables specify the cell delay and slew as a function of input transition time and output capacitance.

By default, the off-nominal tables use the same number of index values as the nominal tables. To reduce the index values of the off-nominal tables, set the following attributes in the library template file:

```
ncx_va_input_net_transition_index : ordinal_list
ncx_va_total_output_net_capacitance_index : ordinal_list
```

For example, suppose that the nominal tables use seven input transition time index values and seven output capacitance index values, a 7-by-7 data table. To generate off-nominal tables that use every other index value of the nominal tables, you would use the following lines in the library template file:

```
ncx_va_input_net_transition_index : 1 3 5 7
ncx_va_total_output_net_capacitance_index : 1 3 5 7
```

These lines cause only the first, third, fifth, and seventh index values in the nominal tables to be used in the off-nominal tables, resulting in 4-by-4 rather than 7-by-7 data tables. Be sure to specify a reasonable range and spacing for the index values. Liberty NCX uses the specified subset of index values without checking to see whether they adequately represent the timing data.

Conditional Characterization

The `ncx_condition` template file group allows you to customize the characterization of certain cells, pins, and arcs in a library. You can vary the conditions by specifying certain required criteria in the condition definition.

Conditional characterization is specified by placing an `ncx_condition` group in the library or cell template file. A conditional characterization definition consists of three key components: a user-defined condition name, the conditions to be met, and the characterization attributes which should be applied when the conditions are met.

The required conditions are defined by one or more Liberty NCX required condition attributes, specified with `ncx_condition_attribute` attributes. The attribute can be any valid Liberty arc keyword, such as `when` or `timing_type`. Liberty NCX also provides additional required condition attribute types for greater flexibility.

The following components specify what attributes should be applied when the required conditions are met:

- An optional set of Liberty NCX cell or arc attributes that should be applied. When a cell, pin, or arc matches the characterization conditions of the `ncx_condition` attribute, these attributes are copied over from the `ncx_condition` attribute, overwriting any existing definitions.
- An optional harness group, specified by using `ncx_harness` syntax, which is used when Liberty NCX characterizes an arc that matches the conditional characterization criteria. If an arc matches the characterization criteria of the `ncx_condition` attribute, then the entire harness group is copied over from the `ncx_condition` attribute as if the group were entered in the arc group in the cell template file.
- An optional `design_rule` extraction attribute specified by using the `ncx_update_max_capacitance` and `ncx_update_max_transition` attributes to enable extraction or preserve the specified `design_rule` values.

Syntax

```
ncx_condition condition_name {
    [ncx_condition_attribute : value ;]
    [attribute : value ;]
    [ncx_harness harness_name { ... }]
}
```

`ncx_condition` template file groups allow you to specify cell-level characterization attributes for entire sets of cells from the library template file, rather than specifying the attributes repeatedly across many cell template files.

Specifying Required Conditions

Required conditions are specified with `ncx_condition_attribute` attributes. The attribute suffix can be any valid Liberty attribute that can be specified in an arc group. For example, the following required condition attributes are available:

`ncx_condition_timing_type`

Specifies the use of the condition for all matching arc types. Wildcards are supported. Some of the possible values are:

- `combinational`, `combinational_rise`, `combinational_fall`
- `setup_rising`, `setup_falling`, `hold_rising`, `hold_falling`
- `non_seq_setup_rising`, `non_seq_setup_falling`, `non_seq_hold_rising`, `non_seq_hold_falling`
- `min_pulse_width`

- preset, clear
- rising_edge, falling_edge

For a complete list of Liberty timing_type attributes, refer to the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

ncx_condition_timing_sense

Specifies the use of the condition for all matching arc sense types. The possible values are:

- positive_unate
- negative_unate
- non_unate

In addition to ncx_condition_attribute required condition attributes based on standard Liberty arc attributes, Liberty NCX also provides the following required condition attributes:

ncx_condition_cell_type

Specifies the use of the condition for all matching cell types. It can assume one of the following values:

- sequential
- combinational
- flipflop
- latch
- clock_gating

ncx_condition_cell_name

Specifies the use of the condition for all matching cell names. Wildcards are supported.

ncx_condition_pin

Specifies the use of the condition for all arcs that terminate at the named pin. Wildcards are supported.

ncx_condition_related_pin

Specifies the use of the condition for all arcs that originate from the named pin. Wildcards are supported.

ncx_condition_arc_type

Specifies the use of the condition for a specific arc type and can assume one of the following values:

- constraint
- delay
- tristate
- prop_power

Wildcards are supported.

```
ncx_condition_output_toggle
```

For constraint arcs, differentiates between arcs which apply to toggled and non-toggled outputs. For more information, see “[Toggled and Non-Toggled Output Constraint Arcs](#)” on [page 3-95](#).

You can use wildcards with many of the required condition attributes to apply characterization attributes to a range of cells or pins without editing each individual cell template. For example,

```
ncx_condition scan_margin {
    ncx_condition_cell_name : SDFF* ;
    ncx_condition_timing_type : setup* ;
    margin_pct : 5 ;
}
```

In Liberty NCX, the determination of condition usage is made just before the generation of characterization indexes. Explicit log messages indicate any application of conditions, as shown in the following example:

```
...
checking indices and signal levels...
arm_12cells condition constrAccuracy matches DFFQX2 setup_rising arc [0] CK -> D
arm_12cells condition constrAccuracy matches DFFQX2 hold_rising arc [1] CK -> D
XOR3X1 condition useHarness matches combinational arc [2] A -> Y
XOR3X1 condition useHarness matches combinational arc [3] A -> Y
total indices checking time: 0.003 seconds (elapsed)
...
```

Specifying Conditional Attributes

When the required conditions of an `ncx_condition` group are met for the specified cells, pins, or arcs, the attribute definitions inside the conditional group are applied. Any valid Liberty NCX cell template attributes can be specified inside the conditional group.

`ncx_condition` groups are additive. Multiple condition groups can apply to the same cell, pin, or arc, such that each condition construct can define additional characterization attributes. When multiple `ncx_condition` groups define the same attribute for a given timing arc, the last defined condition group takes precedence.

To set the HSPICE `runlvl=5` option for all constraint arcs, define the following `ncx_condition` group in the library template file:

```
ncx_condition constrAccuracy {
    ncx_condition_cell_type : sequential ;
    ncx_condition_arc_type : constraint ;
    sim_opt : runlvl=5 ;
}
```

To configure some characterization parameters for non-scan and scan flip-flops, define the following `ncx_condition` groups in the library template file:

```
ncx_condition {
    ncx_condition_cell_name : *DFF* ;
    ncx_condition_model : cell ;
    ncx_auto_function_pin_pattern : q_pin Q qn_pin QB
    default_arc_mode : worst;
}

ncx_condition {
    ncx_condition_cell_name : *SDFF* ;
    ncx_condition_model : cell ;
    ncx_auto_function_pin_pattern : q_pin Q qn_pin QB \
        test_in_pin SD test_en_pin SE ;
    ncx_auto_function_add_test_cell : true ;
}
```

In this example, the first `ncx_condition` group defines the attribute definitions for all non-scan and scan flip-flops, since the `*DFF*` cell name pattern matches both `*DFF*` and `*SDFF*` flip-flop names. The second `ncx_condition` group matches only the scan flip-flops, and defines only the additional attribute definitions needed for the scan flip-flop subset. Because the pin list specified for the `ncx_auto_function_pin_pattern` attribute is an attribute value, the entire pin list must be provided when the attribute is redefined for the scan flip-flops.

To attach a harness for all arcs originating from pin A that have a “when” condition of B for the cell XOR3X1, define the following `ncx_condition` group in the cell template file:

```
ncx_condition useHarness {
    ncx_condition_when : B ;
    ncx_condition_related_pin : A ;
    ncx_harness H1 {
        harness_netlist_file : term.spc ;
        pin Y {
            harness_connect_pin : T ;
            harness_load_pin : 1 ;
        }
    }
}
```

To determine which output pin should be monitored for arcs that potentially have multiple output pins, such as an *n*-bit shift register, specify the `related_output_pin` attribute in the cell template file and then use it inside the `ncx_condition` group. For example, to specify that measurements for all recovery and removal arcs in a 2-bit shift register with outputs Q1 and Q2 be done on output pin Q2 only, use the following condition:

```
ncx_condition RecRemOut {
    ncx_condition_arc_type : constraint ;
    ncx_condition_related_pin : CK ;
    ncx_condition_pin : R ;
    related_output_pin : Q1 ;
}
```

Toggled and Non-Toggled Output Constraint Arcs

In constraint arc characterization, outputs may be toggled or non-toggled. A toggled output is expected to change its output state during the constraint arc characterization period. A non-toggled output is expected to hold its output state constant during this period.

Use the `ncx_condition_output_toggle` attribute to differentiate between toggled and non-toggled constraint arcs when applying violation definitions. For example:

```
ncx_condition vio_toggle_arc {
    ncx_condition_arc_type : constraint ;
    ncx_condition_output_toggle : true ;
    violation_mode : delay_degrade_pct glitch ;
}
ncx_condition vio_nontoggle_arc {
    ncx_condition_arc_type : constraint ;
    ncx_condition_output_toggle : false ;
    violation_mode : glitch "int_node:0" ;
}
```

The `ncx_condition_output_toggle` attribute is only supported for advanced violation mode definitions. For more information about violation definitions, and toggled and non-toggled outputs, see “[Constraint Methodologies](#)” on page 3-141.

Conditional Attributes for Nonpropagating Arcs

The `ncx_condition` template file group lets you control the simulation parameters for the following types of nonpropagating, single-pin arcs:

- Pin-based minimum pulse width

Pin-Based Minimum Pulse Width

This is the syntax for customizing pin-based minimum pulse width characterization:

```
ncx_condition cconst {
    ncx_condition_arc_type : min_pulse_width ;
    variable : value ;
}
```

For example,

```
ncx_condition mconst {
    ncx_condition_arc_type : min_pulse_width
    sim_opt : "accurate runlvl=2" ;
}
```

These are the variables you can set:

```
sim_inc_file
clk_width
tran_timestep
sim_opt
ncx_when
ncx_when_rise
ncx_when_fall
min_pulse_width_slew
min_pulse_width_slew_high
min_pulse_width_slew_low
ncx_mpw_rail_to_rail
violation_mode (and all related variables)
```

Timing Margins

You can direct Liberty NCX to add a specified margin to the characterized delay, slew, and constraint values. Margin refers to the addition of user-specified values to timing table models in order to compensate for uncertainties in the design or process. This section describes several ways in which you can add custom margins to library models in Liberty NCX.

Delay and constraint margins are supported for both NLDM and CCS timing models. The specified amount of margin delay time is applied to the delay or constraint value. For constraint arcs and NLDM timing models, the margin value is added to the table values. For expanded CCS timing models, the margin value is applied as a time-shift to the CCS current points. For compact CCS timing models, the Tpeak values are time-shifted by the margin value.

Slew margins are supported only for NLDM libraries. The margin value is added to the slew table values. Slew data is not stored in a CCS library; slew is derived during delay calculation by applying the current response. As a result, there is no straightforward transformation that allows a slew margin to be applied to a CCS timing model.

Liberty NCX does not add margins for an arc containing variation-aware models, characterized when either the `mismatch` or `variation` configuration variables are set to true.

Margin Types

Margin can be specified using any of the following methods:

- As a constant value across all table points.
- As a percentage of the original value at each table point.
- As a complex expression which can evaluate the min, max, average, or sum of one or more of the following margin components:
 - A percentage of one or more table point index values of compatible unit type.
 - A constant value that varies with each table index value.
 - As a weighted value of a measured simulation quantity, typically a circuit delay.
- As a sum, maximum, or minimum of the constant, percentage, and complex expression margin types described in this list.

The margin attributes corresponding to these margin types are summarized in [Table 3-7](#).

Table 3-7 Margin Attributes

Margin Attribute	Description
<code>margin_value</code>	Specifies a constant margin value
<code>margin_percent</code>	Specifies margin as a percentage of the original value
<code>margin_exp</code>	Specifies margin using a complex margin expression

These margin attributes are supported anywhere in the template files, at the library, cell, pin, or arc level. Since margins are usually specified for particular cells, cell families, or arc types, they are typically defined using `ncx_condition` groups within library or cell template files to control the scope of each margin definition.

The `ncx_condition` group is typically used to apply margin as shown:

```
ncx_condition condition_name {
    [ncx_condition_cell_name : cell_name_pattern ;]
    [ncx_condition_pin_name : pin_name_pattern ;]
    [ncx_condition_timing_type : timing_type_pattern ;]
    [ncx_condition_arc_type : arc_type_pattern ;]

    [margin_value : value ;]
    [margin_percent : value ;]
    [margin_exp : margin_expression ;]
}
```

Other required condition attributes are also available. For more information on the `ncx_condition` group, see “[Conditional Characterization](#)” on page 3-90.

Simple Margin Values

You can use the `margin_value` and `margin_percent` template attributes to apply simple margin values to timing and constraint tables during library processing.

The `margin_value` template attribute can be used to add a fixed value in library time units to each table value. The value can be positive or negative, with negative values representing a decrease in value from the original value. [Figure 3-7](#) shows the resulting margin application from the following hold margin template example. The table on the left shows the original table values. The table on the right shows the table values with the margin applied.

```
ncx_condition margin1 {
    ncx_condition_cell_name : DFF* ;
    ncx_condition_timing_type : hold* ;
    margin_value : 10 ;
}
```

Figure 3-7 margin_value Example

Constraint table before specifying margin

related_pin_slew				
constraint d_pin_slew		10	20	30
	10	75	92	100
	20	52	45	32
	30	100	102	132

Constraint table after adding 10 ps

related_pin_slew				
constraint d_pin_slew		10	20	30
	10	85	102	110
	20	62	55	42
	30	110	112	142

$75 + 10 = 85$

The `margin_percent` template attribute can be used to add margin to each table value as a percentage of the original table value. The percentage value can be positive or negative, with negative values representing a decrease in value from the original value. [Figure 3-8](#) shows the resulting margin application from the following setup margin template example.

```
ncx_condition margin2 {
    ncx_condition_cell_name : DFF* ;
    ncx_condition_timing_type : setup* ;
    margin_percent : 10 ;
}
```

Figure 3-8 margin_percent Example

Constraint table before specifying margin

		related_pin_slew		
		10	20	30
constraint d_pin_slew	10	75	92	100
	20	52	45	32
	30	100	102	132

$75 + 10\% = 7.5$

Constraint table after applying 10%

		related_pin_slew		
		10	20	30
constraint d_pin_slew	10	82.5	101.2	110
	20	57.2	49.5	35.2
	30	110	112.2	145.2

$75 + 7.5 = 82.5$

If both the `margin_value` and `margin_percent` attributes apply to a given delay or constraint table, the margin values are applied according to the `margin_mode` attribute. For more information, see “[Combining Margin Types](#)” on page 3-105.”

Margin Expressions

You can specify complex margin expressions with the `margin_exp` template attribute. Complex margin expressions consist of one or more margin definitions:

```
margin_exp : margin_def1 [margin_def2 [...]]] ;
```

Each margin definition consists of a keyword indicating the margin definition type, followed by one or more related values which pertain to that margin definition type. The valid margin definition templates are as follows:

- `constant value`

The related value is a constant value that is applied to all table points.

- `percent value`

The related value is a percentage value of the original value that is added to all points.

- `input_net_transition index1_val ... indexN_val`
`constrained_pin_transition index1_val ... indexN_val`
`related_pin_transition index1_val ... indexN_val`

The related values are a set of absolute margin values that correspond to each index in the table. The first related number is the margin value applied to table entries for the first index number, and so on.

- `input_net_transition_pct index1_val ... indexN_val`
`constrained_pin_transition_pct index1_val ... indexN_val`
`related_pin_transition_pct index1_val ... indexN_val`

The related values are a set of margin percentage values that represent the percentage of the actual index values themselves that should be computed and applied as margin.

- `measurement_name value`

This is a special type of margin definition that allows you to compute margin values as a function of delay values measured directly from each table value's simulation. To use this definition type, define a list of one or more named measurements using the `ncx_measure_internal_delay` attribute:

```
ncx_measure_internal_delay : \
    measure_name from_pin to_pin \
    [measure_name from_pin to_pin \
    ...] ;
```

Now, you can refer to each named measurement in the `margin_exp` list, along with a related value that specifies a scaling factor to use for computing the margin value.

```
margin_exp : measure_name value ;
```

For all margin definition types that pertain to table indexes, the following rules apply.

- An index-based margin specification is applicable only to tables that use an index of that type.
 - However, the `input_net_transition` and `input_net_transition_pct` margin definition types apply to tables which use either the `input_net_transition` or `input_transition_time` table index types.
- If the number of related values is less than the number of table index values, the last related value is applied for the remaining table index values. Specifying a single related value will apply that value to all table index values.

If you set `margin_exp` to `constrained_pin_transition_pct` with a single related value of 10, as shown in the following example, Liberty NCX adds 10 percent of each constrained pin slew index value to the corresponding values in the table. The resulting table is shown in [Figure 3-9](#).

```
ncx_condition margin_exp1 {
  ncx_condition_cell_name : DFF* ;
  ncx_condition_timing_type : setup* ;
  margin_exp : constrained_pin_transition_pct 10 ;
}
```

Figure 3-9 Example for margin_exp Set to constrained_pin_transition_pct

Constraint table before specifying margin
Constraint table after applying margin

$75 + 1 = 76$

		related_pin_slew		
		10	20	30
constraint_d_pin_slew	10	75	92	100
	20	52	45	32
	30	100	102	132

		related_pin_slew		
		10	20	30
constraint_d_pin_slew	10	76	93	101
	20	54	47	34
	30	103	105	135

You can also add an absolute margin value in time units, specific for each constrained pin slew index, by setting `margin_exp` to `constrained_pin_transition` and providing the list of index-specific related values. For example, if you set `margin_exp` to `constrained_pin_transition` and specify 25, 12, and 45 as a set of margin values, as shown in the following example, Liberty NCX adds a constant value specific to each constrained pin slew to all values in the table. The resulting table is shown in [Figure 3-10](#).

```
ncx_condition margin_exp2 {
  ncx_condition_cell_name : DFF* ;
  ncx_condition_timing_type : setup* ;
  margin_exp : constrained_pin_transition 25 12 45 ;
}
```

Figure 3-10 Example for margin_exp Set to constrained_pin_transition

Constraint table before specifying margin

		related_pin_slew		
		10	20	30
constraint_d_pin_slew	10	75	92	100
	20	52	45	32
	30	100	102	132

Constraint table after applying margin

		related_pin_slew		
		10	20	30
constraint_d_pin_slew	10	100	117	125
	20	64	57	44
	30	145	114	177

$75 + 25 = 100$

The constrained_pin_transition margin value 25 corresponds to constrained_pin_slew table value 10, so 25 is added to the values in the first row of the table, making the new values 100, 117, and 125. Similarly, 12 is added to the second row values in the table, and 45 is added to the third row values.

To apply margin based on actual measurements from each table value's simulation, first define the needed measurements using the ncx_measure_internal_delay attribute. Then apply a margin expression definition which references these named measurements. For example,

```
ncx_condition margin_exp3 {
    ncx_condition_cell_name : DFF* ;
    ncx_condition_timing_type : setup* ;

    ncx_measure_internal_delay : \
        CK_to_node1 CK "node:1" \
        D_to_node2 D "node:2" ;

    margin_exp : CK_to_node1 0.05 ;
}
```

In the previous example, the delay from pin CK to internal node node:1 is measured, then 5 percent of this value is computed and applied as margin to the resulting table value.

The margin expression defined with the margin_exp template attribute may contain multiple margin definitions, using any of the margin definition types previously described. When multiple margin definitions are supplied, the values are combined to determine the final margin_exp value according to the margin_exp_mode attribute. This attribute can be set to

the values shown in [Table 3-8](#). The default value of the `margin_exp_mode` attribute is `sum`, which adds the values across all margin definitions to determine the final margin expression value.

Table 3-8 Supported Values for margin_exp_mode Attribute

Attribute value	Description
<code>min</code>	Use numerical minimum of all defined <code>margin_exp</code> margin definitions
<code>max</code>	Use numerical maximum of all defined <code>margin_exp</code> margin definitions
<code>avg</code>	Use numerical average of all defined <code>margin_exp</code> margin definitions
<code>sum</code>	Compute sum of all defined <code>margin_exp</code> margin definitions

For example, the following template definition computes 5 percent of measurement `CK_to_node1`, 5 percent of measurement `D_to_node2`, then applies the larger of the two measurement values as margin to each table value.

```
ncx_condition margin_exp3 {
    ncx_condition_cell_name : DFF* ;
    ncx_condition_timing_type : setup* ;

    ncx_measure_internal_delay : \
        CK_to_node1 CK "node:1" \
        D_to_node2 D "node:2" ;

    margin_exp : CK_to_node1 0.05  D_to_node2 0.05 ;
    margin_exp_mode : max ;
}
```

In the following example, to add constant values specific to each constraint pin slew and related pin slew to the table, set `margin_exp` to provide a list of related values for both the `constrained_pin_transition` and `related_pin_transition` indexes. For example, if you set `margin_exp` to `constrained_pin_transition` and specify 25, 12, and 45 as a set of margin values, and specify `related_pin_transition` with the values of 10, 15, and 10, Liberty NCX adds a constant value specific to each constraint pin slew and related pin slew at each table value:

```

ncx_condition margin_exp4 {
    ncx_condition_cell_name : DFF* ;
    ncx_condition_timing_type : setup* ;
    margin_exp : constrained_pin_transition 25 12 45 \
                  related_pin_transition 10 15 10 ;
    # margin_exp_mode not set; default is sum
}

```

Liberty NCX adds the absolute margin that corresponds to the constrained pin slew and the related pin slew to the table value corresponding to the same constrained and related pin slew. [Figure 3-12](#) shows the original table values on the left, and the margined table values on the right after the `constrained_pin_transition` and `related_pin_transition` margin values have been applied.

Figure 3-11 Example for margin_exp Set to constrained_pin_transition and related_pin_transition

related_pin_slew				
constraint d.pin_slew	10	20	30	
10	75	92	100	
20	52	45	32	
30	100	102	132	

related_pin_slew				
constraint d.pin_slew	10	20	30	
10	110	132	135	
20	74	72	54	
30	155	162	187	

$75 + 25 + 10 = 110$

The `constrained_pin_transition` margin value 25 corresponds to `constrained_pin_slew` table value 10, and `related_pin_transition` margin value 10 corresponds to `related_pin_slew` table value 10. Therefore, the margin calculated for value 75, as shown in [Figure 3-11](#), is 25 plus 10, which equals 35. Liberty NCX adds 35 to table value 75, making the new value 110.

Similarly, the `constrained_pin_transition` margin value 25 corresponds to `constrained_pin_slew` table value 10, and `related_pin_transition` margin value 15 corresponds to `related_pin_slew` table value 20. Therefore, the margin calculated for table value 92 is 25 plus 15, which equals 40. Liberty NCX adds 40 to table value 92, making the new value 132.

Combining Margin Types

The three margin type attributes, `margin_value`, `margin_percent`, and `margin_exp`, can be used to compute and apply margin in different ways. When more than one of these three margin types applies to a table, the `margin_mode` attribute is used to determine how to select from or combine the computed margin values. The supported values for the `margin_mode` attribute are shown in [Table 3-9](#).

Table 3-9 Supported Values for margin_mode Attribute

Attribute value	Description
<undefined>	default behavior, <code>margin_value</code> takes precedence over <code>margin_percent</code>
min	Use numerical minimum value of any defined <code>margin_value</code> , <code>margin_percent</code> , and <code>margin_exp</code> values
max	Use numerical maximum value of any defined <code>margin_value</code> , <code>margin_percent</code> , and <code>margin_exp</code> values
sum	Compute sum of all defined <code>margin_value</code> , <code>margin_percent</code> , and <code>margin_exp</code> values

For example, if you set `margin_mode` to `max` and specify both the `margin_value` and `margin_percent` margin types, as shown in the following example, Liberty NCX calculates whether 10 percent or 10 ps is a larger adjustment for each value in the table, and then adds the larger margin to each value:

```
ncx_condition margin4 {
    ncx_condition_cell_name : DFF* ;
    ncx_condition_timing_type : setup* ;
    margin_percent : 10 ;
    margin_value : 10 ;
    margin_mode : max ;
}
```

[Figure 3-12](#) shows the constraint table before the `max` value is specified, and the table after the largest margin values are applied.

Figure 3-12 Example for margin_mode Set to max

Constraint table before specifying `max`

		related_pin_slew		
		10	20	30
constraint d_pin_slew	10	75	92	100
	20	52	45	32
	30	100	102	132

Constraint table after applying largest margins

		related_pin_slew		
		10	20	30
constraint d_pin_slew	10	85	102	110
	20	62	55	42
	30	110	112.2	145.2

$75 + 10 \text{ or } 10\% = 85 \text{ or } 82.5$

85 is the larger adjustment

When multiple margin definitions are defined to form a margin expression using the `margin_exp` attribute, these margin expression values are first combined using the method specified by the `margin_exp_mode` attribute, then further combined with other margin types using the method specified by the `margin_mode` attribute. This allows for complex margins to be defined, such as using `margin_exp_mode` to determine the worst margin across a number of `margin_exp` expressions, then using `margin_mode` and `margin_value` to specify a minimum fallback value.

Adding Margin Data To a Library As a Postprocess

Adding margin data to a library as a postprocess allows Liberty NCX to add margin to a library without recharacterization. To enable this flow, set the following variable in the configuration file:

```
set postprocess true
```

As a prerequisite to the postprocess flow, you must have completed the following steps during the characterization of the library.

- Specify margin variables, such as `margin_value`, `margin_percent`, and `margin_exp`, as described in “[Timing Margins](#)” on page 3-96. By default, when margin variables are specified during characterization, the calculated margins are added to the simulated data.
- Set `margin_data_to_library` to `true` in the configuration file. This results in the following library structure example.

Note:

If the `margin_data_to_library` variable has not been set during the initial characterization flow, postprocessing does not work.

Library Structure Example

```

pin (clrz) {
    timing () {
        related_pin : "clk";
        timing_type : "recovery_rising";
        margin_exp : "m1_clk_q 0.3 related_pin_transition_pct 1
                      constrained_pin_transition_pct 20";
        margin_rise_constraint: "84.974400 86.272100 \
                                  85.792900 87.350100";
        margin_measure_name_rise : "m1_clk_q";
        margin_measure_value_rise:"281.148 280.807 \
                                  283.643 284.167";

        margin_applied : true ;
        rise_constraint ("constraint_slewref_2slewdata_2") {
            index_1("3.0000000, 10.0000000");
            index_2("3.0000000, 10.0000000");
            values("93.7639273, 99.2021936", \
                   "91.3392481, 96.1483694");
        }
    }
}

```

[Table 3-10](#) explains the information in “Library Structure Example” on page 3-107 that is written by Liberty NCX during characterization.

Table 3-10 Library Structure Descriptions

Attribute name	Description
margin_exp	Comes from setting <code>margin_exp</code> in the template during characterization.
margin_rise_constraint	Comes from calculating the margin delta values during characterization.
margin_measure_name_rise	Is the name of the delay path specified by <code>margin_exp</code> .
margin_measure_value_rise	Is the actual simulation data for the delay path specified by <code>margin_exp</code> .
margin_applied	Is added by Liberty NCX to indicate that margin values have been added to table values.
rise_constraint	Is library data.

The following list explains the postprocess manipulations that can be done to the margin data:

- `ncx_margin_model [add | remove | preserve];`

`ncx_margin_model` is a master switch to specify whether to add, remove, or preserve a margin that is defined by either or both of the following combinations of attributes:

- The `margin_exp` attribute
- Any or all of the `margin_mode`, `margin_value`, and `margin_percent` attributes

The `ncx_margin_model` switch specifies a library, cell, pin, or arc attribute to add or remove a margin or do nothing (preserve). It changes the `margin_applied` arc attribute to `true` after adding a margin or to `false` after removing a margin.

Liberty NCX generates an error or a warning and does not change any values when either of the following conditions apply:

- The `ncx_margin_model` attribute is specified as `add`, but the `margin_applied` attribute is `true`.
- The `ncx_margin_model` attribute is set to `remove`, but the `margin_applied` attribute is `false` or is not available.

You can use `ncx_margin_model` in conjunction with the postprocessing flow to change or add margins in the library without any resimulation or characterization; it becomes a postprocessing step.

For the postprocess flow, your input seed library must be generated with `margin_data_to_library = true` and contain `margin_data` as a user-defined attribute. Otherwise, the tool is not able to add, remove, or recalculate margins.

- `ncx_margin_min`

Applies a lower bound to a margin change (delta). This value is in library time units.

- `ncx_margin_max`

Applies an upper bound to a margin change (delta). This value is in library time units.

Note:

Make sure that the prerequisite conditions are met during characterization in order for the postprocessing to work.

Liberty NCX issues a warning (without changing the value) when either of the following conditions apply:

- You add a margin to a library that already has a margin (that is, when `margin_applied` is set to `true`).
- You remove margins from an unmargined library (that is, when `margin_applied` is set to `false`).

Complex Cell Features

Liberty NCX can often determine all of the cell operating characteristics from the netlist alone. However, for complex cells, Liberty NCX needs additional information about the functionality of the cell so that it can properly characterize those features.

The following sections describe the features that require additional specifications in the template files:

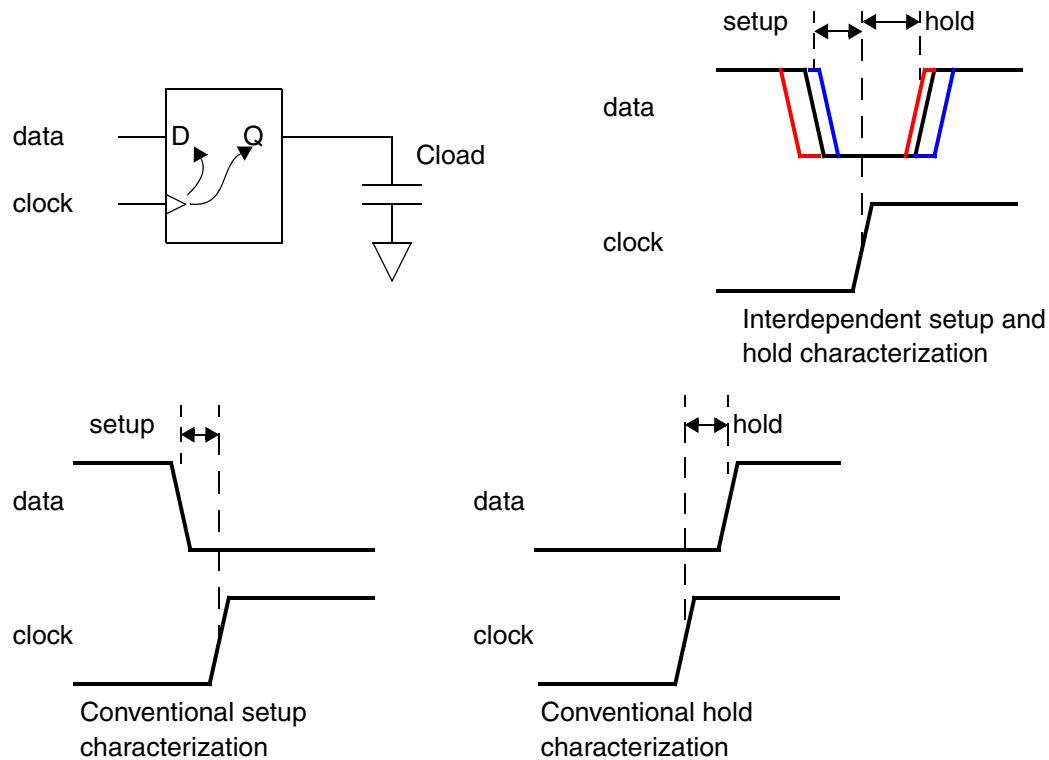
- [Interdependent Setup and Hold](#)
- [Minimum Setup Time for Pessimistic Delay](#)
- [Differential Inputs and Outputs](#)
- [Nonrail Voltage Swing](#)
- [Constant Logic States on Inputs](#)
- [Pin-Specific Timing Thresholds](#)
- [Initialization Cycle](#)
- [Synchronizer Circuit](#)
- [Complex Sequential Cells](#)
- [Simultaneous Switching](#)
- [Custom Harness](#)

Interdependent Setup and Hold

For a sequential cell such as a flip-flop or transparent latch, Liberty NCX performs characterization of the setup and hold constraints. The setup time is the minimum amount of time that the data must be valid before the clock edge. The hold time is the minimum amount of time that the data must be held valid after the clock edge.

Liberty NCX performs conventional characterization of setup and hold times separately. It performs setup analysis with the data left constant for a long time after the clock edge, and it performs hold analysis with the data held constant for a long time before the clock edge. See the conventional setup and hold characterization timing diagrams at the bottom of [Figure 3-13](#).

Figure 3-13 Interdependent Setup and Hold



However, the conventional setup and hold times can be optimistic because they ignore the negative impact of the interdependence between setup and hold. A very small or minimum setup time might require a longer hold time, and conversely, a very small or minimum hold time might require a longer setup time. This effect becomes increasingly significant in deep submicron technologies.

You can select a setup and hold pessimism reduction (SHPR) setup and hold pair to specify the conventional setup and hold time in a library. To do this, set the `setup_hold_method` attribute to `minimum` in the library or cell template file.

When you set `setup_hold_method` to `max_setup` or `max_hold`, Liberty NCX enables the SHPR flow automatically and acquires the data. However, Liberty NCX requires additional simulation in order to get the SHPR result and replace the conventional setup and hold value. The `setup_hold_method` attribute is set to `minimum` by default.

An interdependent analysis finds the worst-case setup and hold times simultaneously, to fully account for the interdependence between setup and hold, resulting in various combinations of setup and hold values. See the interdependent setup and hold characterization timing diagram at the top of [Figure 3-13 on page 3-110](#).

If the library description of a cell has interdependent setup and hold information, an optimization or analysis tool can get better results by using the interdependent values in place of the fixed values generated the conventional way.

To invoke interdependent setup and hold characterization of sequential cells, set the constraint and shpr_constraint options to true in the configuration file. The constraint option invokes timing arc constraint acquisition, including conventional setup and hold. The shpr_constraint option invokes interdependent setup and hold acquisition in addition to conventional setup and hold.

The characterized model must contain conventional setup and hold information together with the interdependent information. If the optimization or analysis tool cannot use the interdependent data in the library, it uses the conventional data instead.

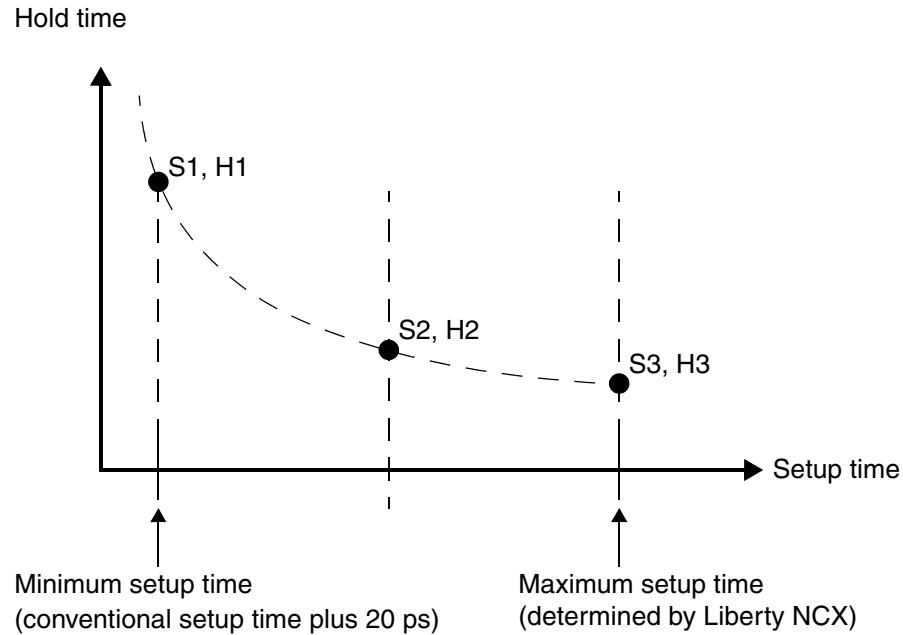
By default, Liberty NCX generates three setup and hold pairs of values. Experiments have shown that at least three pairs of values are required for an accurate representation, and three pairs typically give good results.

For even higher accuracy, you can specify a larger number of data points, at the cost of some additional memory usage and runtime. To specify a larger number of setup and hold pairs of values, set the ncx_shpr_setup_points attribute in the cell or library template file to the desired number. For example,

```
ncx_shpr_setup_points : 4 ;
```

Liberty NCX chooses setup values that are equally spaced and ranging from just above the conventional setup time to the maximum setup time. For example, to find three setup and hold data points, it divides the range from the minimum setup time to the maximum setup time into equal parts. (The minimum setup time is the conventional setup time plus an offset of 20 picoseconds, and the maximum setup time is a value determined by Liberty NCX.) Then it finds the interdependent setup and hold times at the minimum, maximum, and intermediate points, as illustrated in [Figure 3-14 on page 3-112](#).

Figure 3-14 Default Choice of Interdependent Setup Times



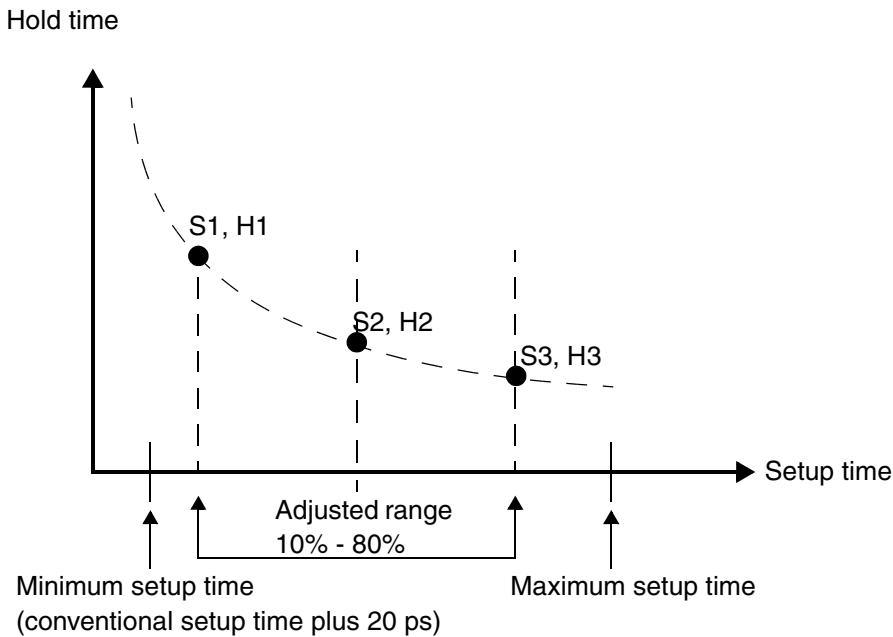
User Customization

You can also specify a different range by setting the upper and lower boundary attributes in the cell or library template, as in the following example:

```
ncx_shpr_setup_lower_bound_pct : 10;
ncx_shpr_setup_upper_bound_pct : 80;
```

The lower bound is raised by 10 percent of the default range and the upper bound is lowered to 80 percent of the default range. The resulting setup points are shown in [Figure 3-15 on page 3-113](#).

Figure 3-15 Modified Choice of Interdependent Setup Times



You can also specify the offset from the conventional setup time used to obtain the minimum setup time. The default offset is 20 ps. You can specify a different value such as 30 ps:

```
ncx_shpr_setup_offset : 30.0e-12;
```

Liberty NCX always uses a ramp-type waveform for interdependent setup and hold acquisition. If you specify any other type of waveform for characterization, Liberty NCX uses that type of waveform for all characterization, including conventional setup and hold acquisition, but uses a ramp waveform for interdependent setup and hold.

In the generated library description of the cell in Liberty format, the interdependent setup and hold information immediately follows the conventional data. The Liberty keyword `interdependence_id` identifies the data point number, ranging from 1 to 3 by default.

Minimum Setup Time for Pessimistic Delay

You can use the `ncx_min_setup_based_delay` attribute to find the clock to output delay with the minimum setup time between the clock and the data. When you set the `ncx_min_setup_based_delay` attribute to `true` in the library or cell template file, Liberty NCX begins by finding the corresponding minimum setup time for each clock slew. Then, it applies the minimum setup time to characterize the clock-to-output delay arc in a regular

transition simulation. The result is a more pessimistic delay value. The `ncx_min_setup_based_delay` attribute only applies to the delay arc of sequential cells that have a setup constraint arc from a data-to-clock or a data-to-enable pin.

The data slew must be fixed when Liberty NCX finds the setup time of the specified clock slew. You can select the data slew value by using the `ncx_min_setup_based_constraint_slew` attribute. The `ncx_min_setup_based_delay_constraint_slew` attribute specifies the constraint pin slew value from the delay table. The values are `middle` (the default), `min`, and `max`.

The output load must also be fixed when Liberty NCX finds the setup time. It is recommended that you use a `zero` load. Use the `ncx_min_setup_based_delay_setup_time_offset` attribute to add a margin to the setup time used for characterizing the delay and output transition time. The offset margin value must be a positive number. The default value for `ncx_min_setup_based_delay_setup_time_offset` is `1e-12` seconds.

Caution:

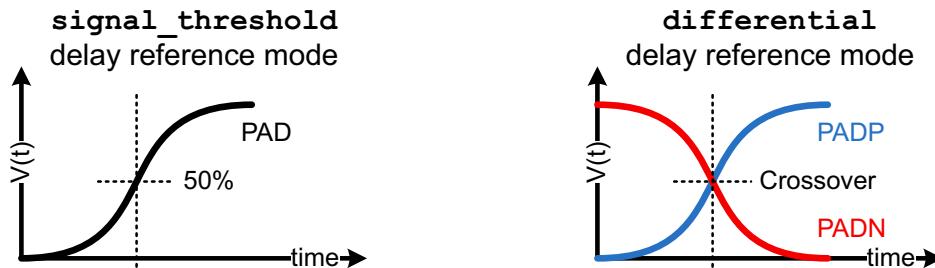
A negative offset value causes a characterization failure.

Differential Inputs and Outputs

Liberty NCX supports differential inputs and outputs defined with the `complementary_pin` Liberty attribute.

Normally, delay measurement involving a pin uses the delay trip point crossing of the pin's voltage response, usually 50 percent, as the reference time for the cell delay measurement. This standard method is called the *signal threshold* delay reference mode. With a differential pin pair, the pins are always logically inverted with respect each other, and the delay reference is measured according to the crossover point of the differential signals. This is called the *differential* delay reference mode.

Figure 3-16 Signal Threshold Pins and Differential Pins



Differential signals are used for more robust on-chip or off-chip signal transport in noisy or low-voltage environments. Careful routing of the differential signals can ensure that both signals are affected similarly by noise and crosstalk effects. When the signals arrive at the differential receiver, subtracting the two voltage responses to determine the difference removes the shared interference effects.

If a cell has differential input or output pin pairs, use the `complementary_pin` Liberty attribute in the template pin group description to declare the differential relationship. A pin group definition is required only for the primary signal pin in the differential pin pair, as the complementary differential pin is automatically created when the `complementary_pin` attribute is defined. For example,

```
pin PADP {
    direction : input ;
    complementary_pin : PADN ;
}
```

For differential inputs defined with the `complementary_pin` attribute, Liberty NCX automatically drives the pin pair with a logically inverted waveform during characterization.

Differential Pin Delay Measurement Modes

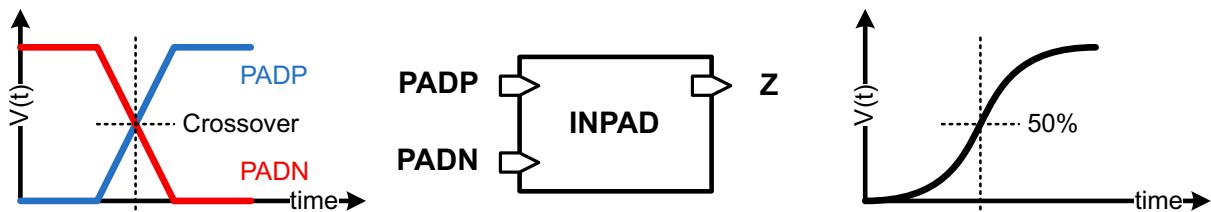
In addition to defining a pin pair using the `complementary_pin` attribute, you should also specify the desired delay reference mode for any differential pin pairs. By default, all pins use the signal threshold reference mode, including differential pins. You can specify the delay reference mode for input and output differential pin pairs using the following template attributes:

```
ncx_input_delay_reference_mode: signal_threshold | differential ;
ncx_output_delay_reference_mode: signal_threshold | differential ;
```

The template description for the simple differential input pad shown in [Figure 3-17](#) is as follows.

```
# template definition for differential INPAD cell
pin PADP {
    direction : input ;
    complementary_pin : PADN ;
    ...
}
pin Z {
    direction : output ;
    function : PADP ;
}
ncx_input_delay_reference_mode : differential ;
```

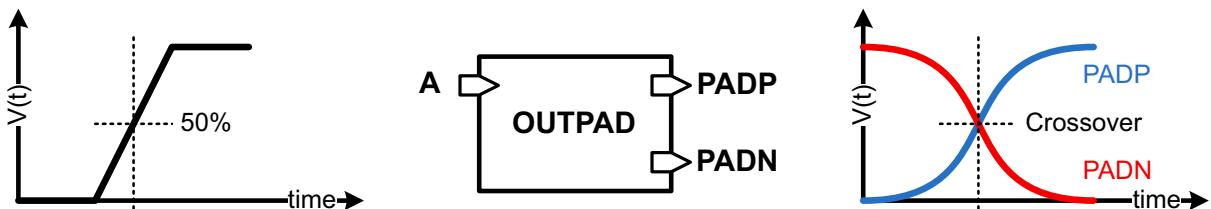
Figure 3-17 Simple Differential Input Pad Cell



The template code for the simple differential output pad shown in [Figure 3-18](#) is as follows.

```
# template definition for differential OUTPAD cell
pin A {
    direction : input ;
}
pin PADP {
    direction : output ;
    complementary_pin : PADN ;
    function : A ;
    ...
}
ncx_output_delay_reference_mode : differential ;
```

Figure 3-18 Simple Differential Output Pad Cell



All differential pin measurement configuration attributes, including those defined in the following sections, can be controlled at the timing arc level using the `ncx_condition` group. For example, there may be a test operating mode of a differential pad cell where only the primary pin varies, and the `signal_threshold` method should be used for these test-mode delay arcs. Such a case could be described using the following template description:

```
# template definition for differential INPAD cell with test mode
pin PADP {
    direction : input ;
    complementary_pin : PADN ;
    ...
}
```

```

pin TESTMODE {...}
pin Z {
    direction : output ;
    function : PADP ;
}
ncx_input_delay_reference_mode : differential ;

# use single-ended PADP signal only for test mode
ncx_condition test_mode {
    ncx_condition_when : "TESTMODE" ;
    ncx_condition_related_pin : PADP ;
    ncx_input_delay_reference_mode : signal_threshold ;
}

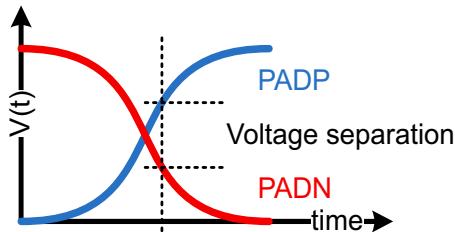
```

For more information on conditional groups, see “[Conditional Characterization](#)” on [page 3-90](#).

Delayed Differential Measurements

In some cases, you might want to measure the delay of differential pins according to the point where their voltage responses have crossed and separated by some amount. This measurement methodology can be useful in situations that require robustness against inadvertent early crossing events. The voltage separation results in a delayed differential reference time, as shown in [Figure 3-19](#).

Figure 3-19 Delayed Differential Measurements



The amount of voltage separation for input and output differential pin pairs can be defined as a positive non-zero voltage using the following absolute voltage separation template attributes:

```

ncx_input_delay_differential_value voltage_value ;
ncx_output_delay_differential_value voltage_value ;

```

The absolute voltage separation values are specified in library voltage units. The default value is zero, which uses the exact crossing point as the delay reference.

The amount of voltage separation can also be defined as a percentage of pin rail voltage using the following relative voltage separation template attributes:

```
ncx_input_delay_differential_value_pct value ;
```

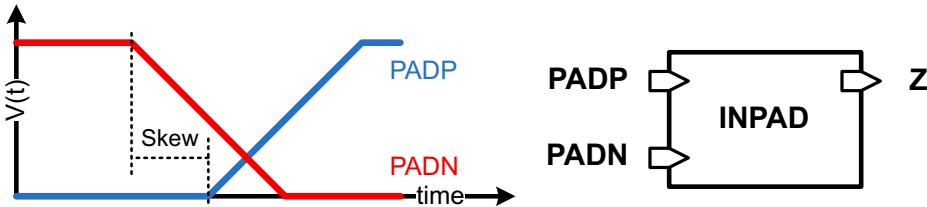
```
ncx_output_delay_differential_value_pct value ;
```

These relative voltage separation attributes are undefined by default. When defined, they take precedence over the absolute voltage separation attributes.

Skewed Differential Input Signals

In some cases, the complementary signals of a differential input pin pair might not be expected to transition at the same time. For example, the primary PADP input signal in [Figure 3-20](#) is skewed later in time relative to the complementary PADN input signal.

Figure 3-20 Skewed Differential Input Pin Signals



You can specify that skewed differential signals should be applied to differential input pin pairs during characterization with the `ncx_signal_skew` attribute:

```
ncx_signal_skew : reference_pin skewed_pin skew_time_value ;
```

Specify the reference pin name first, followed by the skewed, or delayed, pin name. Specify the amount of skew as a positive non-zero value in library time units. During characterization, the skewed pin input stimulus will be delayed by this time value relative to the reference pin stimulus. The reference and complementary pins can be specified in either order in the skew definition to achieve the desired relative relationship.

The `ncx_signal_skew` attribute is specified at the cell template level. The skew definition will apply to any differential input pin pairs which match the pin names in the skew definition.

For the previous example, PADP can be delayed from PADN using the following attribute definition:

```
# delay PADP by 100ps
ncx_signal_skew : PADN PADP 100 ;
```

Nonrail Voltage Swing

If a cell has any input pins or output pins that do not swing fully between 0.0 and the rail voltage, this information should be declared in the template. For an input pin, you should explicitly specify the low and high voltages reached by the pin. For an output pin, you should

explicitly specify the low and high voltages reached by the pin, if known. Otherwise, you should specify that the output swings to nonrail voltages so that Liberty NCX measures the low and high output voltages.

To specify the nonrail voltages of an input pin, use

```
input_signal_level_high : voltage ;  
input_signal_level_low : voltage ;
```

For example,

```
input_signal_level_high : 1.65 ;  
input_signal_level_low : 0.85 ;
```

When Liberty NCX generates the input waveforms, it drives the input pin directly with a nonrail PWL source that swings between the specified high and low voltages.

To specify the nonrail voltages of an output pin, use

```
output_signal_level_high : voltage ;  
output_signal_level_low : voltage ;
```

For example,

```
output_signal_level_high : 1.65 ;  
output_signal_level_low : 0.85 ;
```

When Liberty NCX measures the delay or output slew, it applies the measurement threshold percentages to the specified low-to-high voltage range rather than the range from 0.0 to the rail voltage.

To specify whether an output swings to nonrail voltages, use

```
non_rail_output_signal_level : true | false ;
```

For example,

```
non_rail_output_signal_level : true ;
```

Setting this attribute to true indicates that the output swings to unknown nonrail voltages. Liberty NCX determines the high and low voltage levels attained at the output and applies the measurement threshold percentages to this measured range rather than the range from 0.0 to the rail voltage.

Constant Logic States on Inputs

If there is a side input of a cell that is held at a fixed logic state, you must declare the fixed state in the template file. Use the following syntax:

```
ref_state : 0|1
```

Example:

```
ref_state : 1 ;
```

Liberty NCX holds the input at the specified logic state during characterization.

If the voltage of the logic state is not the default VDD or VSS voltage, you can specify the voltage with the following syntax:

```
input_signal_level : voltage
```

Example:

```
input_signal_level : 0.88 ;  
ref_state : 1 ;
```

Pin-Specific Timing Thresholds

The voltage thresholds that define the logic transition points are defined in the library. By default, Liberty NCX uses these library-defined thresholds to measure the delay times for all characterization of a library.

However, there can be cases in which a specific pin of a cell uses threshold voltages that are different from the library-defined thresholds. For proper characterization, you need to specify these pin-specific thresholds so that Liberty NCX can use them for delay measurements.

This is the syntax for specifying pin-specific threshold voltages:

```
input_rise_threshold : voltage  
input_fall_threshold : voltage  
output_rise_threshold : voltage  
output_fall_threshold : voltage
```

Example:

```
input_rise_threshold : 1.5 ;  
input_fall_threshold : 1.5 ;  
output_rise_threshold : 1.5 ;  
output_fall_threshold : 1.5 ;
```

These attributes specify the voltages at which timing measurements are made at the applicable input pin or output pin for rising-edge and falling-edge transitions.

Initialization Cycle

Some special-purpose cells require repeated input transitions to put the cell into a known stable state for timing measurement. The `init_cycle` attribute defines the number of input transitions required for initialization of a combinational cell:

```
init_cycle : N
```

Example:

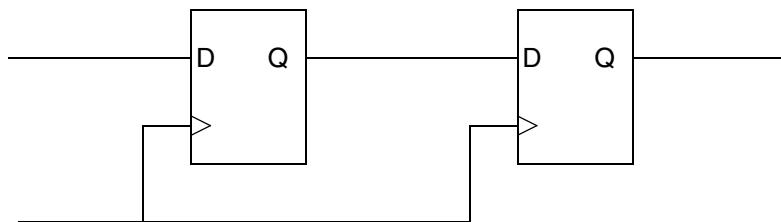
```
init_cycle : 2 ;
```

Liberty NCX performs the specified number of initialization transitions before it applies the measured input transition.

Synchronizer Circuit

A synchronizer is a circuit consisting of two back-to-back D flip-flops clocked by the same clock signal, as shown in [Figure 3-21 on page 3-121](#). There is no specific Liberty syntax to define a synchronizer because a synchronizer can be defined as a single D flip-flop in Liberty. However, to sensitize the timing arcs of a synchronizer, Liberty NCX needs information about the presence of a synchronizer implemented as two D flip-flops.

Figure 3-21 Synchronizer Circuit



Liberty NCX uses the Liberty keyword `clocked_on_also` to indicate that a cell is a synchronizer. If the original Liberty library defines a synchronizer as a D flip-flop, you must define the cell function again in the Liberty NCX template file by using `clocked_on_also`. For example,

```
ff "IQ" "IQN" {
    clocked_on : CK ;
```

```

    clocked_on_also : CK ;
    next_state : D ;
}

```

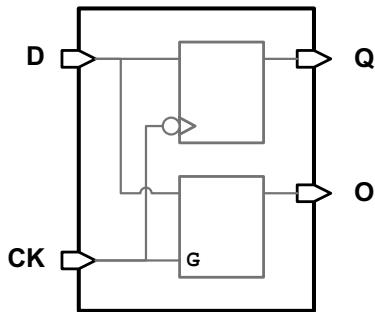
Complex Sequential Cells

Sequential elements are defined using `statetable`, `ff`, and `latch` groups. Normally, when a cell template description contains multiple sequential group definitions, a single sequential element is kept according to the following precedence order:

- `statetable` (highest priority)
- `ff`
- `latch` (lowest priority)

However, some complex sequential cells may have multiple internal sequential elements. An example of such a cell is shown in [Figure 3-22](#).

Figure 3-22 Complex Flip-Flop and Latch Cell



You can model multiple sequential elements in a cell's template description by using the `ncx_multiple_descriptor` group. This group informs Liberty NCX that multiple sequential groups, provided inside the multiple descriptor group, are needed to describe the cell's functional behavior.

The specified sequential groups do not have to exactly match the underlying transistor-level cell structure. It is sufficient to simply describe the functional behavior of the cell so that the proper sensitizations can be determined for characterization.

The following template description correctly describes the example cell in [Figure 3-22](#).

Example 3-1 Complex Flip-Flop and Latch Cell Template Description

```
ncx_multiple_descriptor {
    # flip-flop path to Q:
    ff "IQ1" "IQN1" {
        clocked_on : !CK ;
        next_state : D ;
        clear : !R ;
    }

    # latch path to O:
    latch "IQ2" "IQN2" {
        enable : CK ;
        data_in : D ;
        clear : !R ;
    }
}
```

Each sequential element may launch data to a separate output pin, or multiple sequential elements may launch data to the same output pin. For example, the following template describes a double-data-rate cell where the rising edge of CK captures and launches data from input pin DR, while the falling edge of CK captures and launches data from input pin DF.

Example 3-2 Double-Data-Rate Flip-Flop Template Description

```
ncx_multiple_descriptor {
    # flip-flop capturing and launching DR when CK rises:
    ff "IQ1" "IQN1" {
        clocked_on : CK ;
        next_state : DR ;
    }

    # flip-flop capturing and launching DF when CK falls:
    ff "IQ2" "IQN2" {
        clocked_on : CK' ;
        next_state : DF ;
    }
}

pin Q {
    direction : output ;
    function : IQ1 ;
    function : IQ2 ;
}
```

Note:

Only parallel sequential elements are supported. Sequential elements cannot be connected in series, such that the output of one sequential element drives the input of another sequential element.

Publishing Complex Sequential Cell Descriptions

Liberty NCX uses the information in `ncx_multiple_descriptor` groups to determine the needed sensitizations for cell characterization. However, it is possible to describe complex sequential cell functionality using `ncx_multiple_descriptor` groups that cannot be fully described in the resulting Liberty output library file.

For example, the flip-flop and latch cell described in [Example 3-1](#) can be fully represented in the output library because Liberty supports multiple sequential group elements driving separate output pins. However, the double-data-rate flip-flop cell described in [Example 3-2](#) cannot be fully represented in the output library because Liberty does not support multiple sequential group elements driving the same output pin. For this cell, the needed timing arcs will be present in the output library, but the sequential functionality of the cell cannot be fully described.

When the `ncx_multiple_descriptor` group is used, Liberty NCX uses the following rules to determine what functional representation is published in the output library, listed in order of highest precedence first.

- If a sequential group element is specified outside the `ncx_multiple_descriptor` group, it is published to the output library.
- If a `statetable` sequential group element is mixed with `ff` or `latch` sequential group elements inside the `ncx_multiple_descriptor` group, the cell will be published as a black box cell with no functional information.
- If multiple `statetable` sequential group elements are specified inside the `ncx_multiple_descriptor` group, only the first `statetable` definition is published because Liberty does not support multiple `statetable` definitions within a cell.
- If multiple `ff` and/or `latch` sequential group elements are specified inside the `ncx_multiple_descriptor` group, and they drive separate output pins, each sequential group element is published for its respective output pin.
- If multiple `ff` and/or `latch` sequential group elements are specified inside the `ncx_multiple_descriptor` group, and one or more drive the same output pin, only the first sequential group element is published for that output pin.

You can use these publishing precedence rules to fine-tune the functional description published to the output library.

Simultaneous Switching

You can specify simultaneous switching inputs for a cell by using the `simultaneous_switching` attribute as shown:

```
simultaneous_switching : "switching_pin_count pin1 pin2 ... ";
```

For example, in addition to specifying the function for a 1-hot MUX with three select pins, S1, S2, and S3, you can model simultaneous switching behavior. To do this, specify the `contention_condition` function in the .lib file or a cell-level template file to establish the direction of the switching inputs, as shown in the following example:

```
contention_condition : "s1*s2 + s1*s3 + s2*s3 + s1*s2*s3 + !s1!*s2!*s3"
```

Specify `simultaneous_switching` in the cell template file to model simultaneous switching behavior, as shown in the following example where 2 is the number of pins that are switching:

```
simultaneous_switching : "2 S1 S2 S3";
```

Custom Harness

Liberty NCX supports the use of one or more custom harnesses when characterizing a test cell. A custom harness is any circuit netlist, separate from the test cell netlist, that is attached to one or more terminals of the test cell. The harness can be used as a loading network, as a driver conduit through which the test cell is stimulated, or as a source of measure points for characterization waveforms.

A custom harness is defined in the cell template file. The scope of harness usage depends on the location of the harness definition. For example, if the harness is defined at the cell level, it is used for every acquisition of that cell, or if it is defined at the timing arc level, it is used only for that arc.

This is the syntax for defining a harness:

```
ncx_harness harness_name {
    harness_netlist_file : file_name ;
    pin (cell_pin_name) {
        harness_connect_pin : harness_pin_name ;
        harness_drive_pin : harness_pin_name ;
        harness_measure_pin : harness_pin_name ;
    }
    pin (cell_pin_name) {
        harness_connect_pin : harness_pin_name ;
        harness_load_pin : harness_pin_name ;
        harness_measure_pin : harness_pin_name ;
    }
    ...
}
```

Each harness has a name and a `harness_netlist_file` specification. Multiple harnesses of a cell must have different names. Each specified `cell_pin_name` must be a valid pin name of the test cell. It is not necessary to connect every pin to a harness. Any unconnected harness terminals are left open.

Every cell pin group must have a `harness_connect_pin` specification. The harness pin name must be a valid pin name of the harness subcircuit. The `harness_drive_pin`, `harness_load_pin`, and `harness_measure_pin` specifications are optional. The `harness_load_pin` specification is ignored if the parent cell pin is an input pin. The `harness_drive_pin` specification is ignored if the parent cell pin is an output pin.

For an input pin, the `harness_measure_pin`, if specified, should be the same as either the cell pin or the `harness_drive_pin`. For an output pin, the `harness_measure_pin`, if specified, should be the same as either the cell pin or the `harness_load_pin`. These requirements prevent invalid connections of the meter elements that are used to monitor the response.

The following examples of harness specifications in the cell template file apply to a test cell, DFFQX2, for various measurement configurations. Each template entry is followed by a schematic diagram showing the circuit used. The various circuit elements of interest that influence the model extraction are:

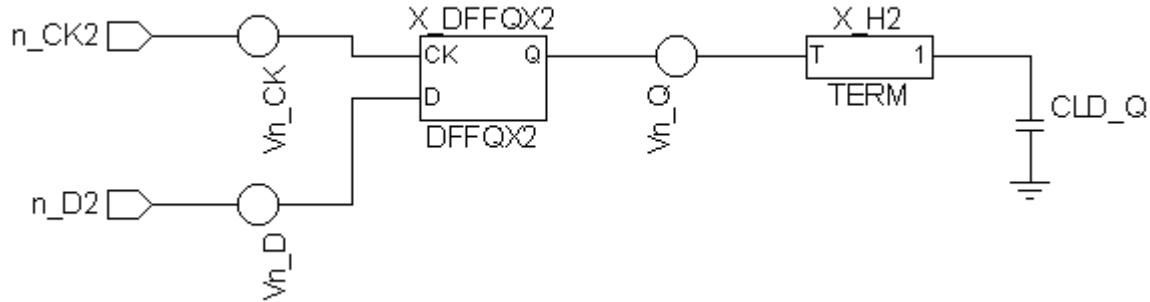
- CLD_Q : output load capacitor at node Q
- Vn_CK : response monitor (ammeter) for input node CK
- Vn_D : response monitor (ammeter) for input node D
- Vn_Q : response monitor (ammeter) for output node Q

External Termination Harness

The following harness attaches an external termination to the output pin Q of the test cell. It is shown in [Figure 3-23 on page 3-127](#).

```
ncx_harness H2 {
    harness_netlist_file : term.spc ;
    pin (Q) {
        harness_connect_pin : T ;
        harness_load_pin : 1 ;
    }
}
```

Figure 3-23 External Termination Harness



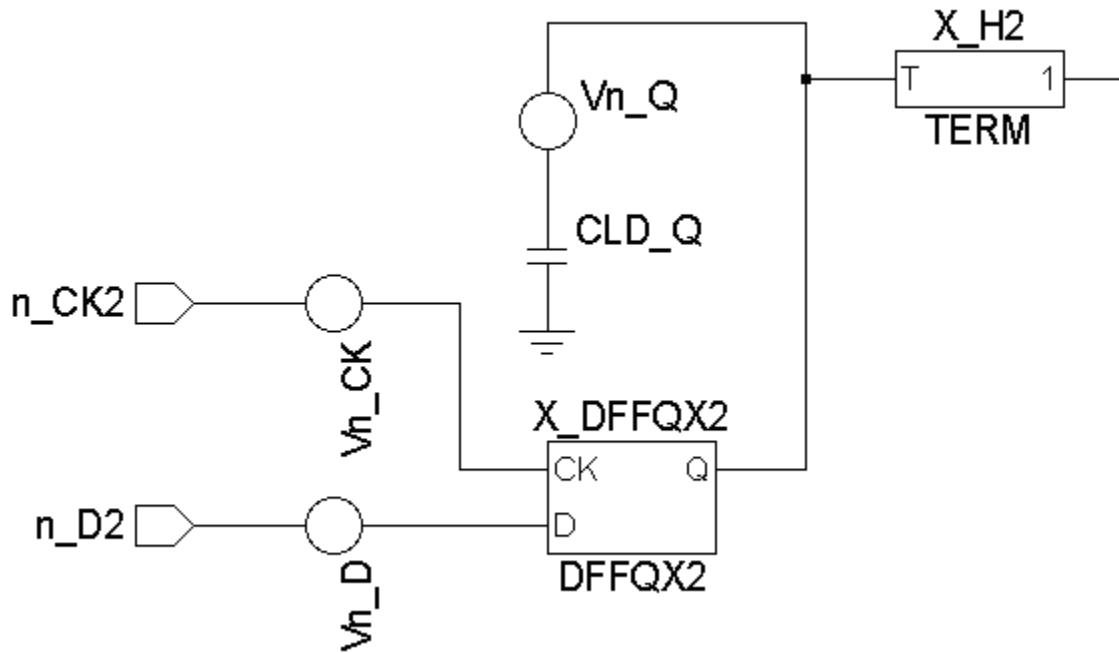
Termination and Measurement Harness

The following harness attaches an external termination to the output pin 1 of the harness cell and measures the response at the output pin Q of the test circuit. It is shown in [Figure 3-24 on page 3-128](#).

```

ncxHarness H2 {
    harnessNetlistFile : term.spc ;
    pin (Q) {
        harnessConnectPin : T ;
    }
}
    
```

Figure 3-24 External Termination and Measurement Harness



Driver, Termination, and Measurement Harness

The following harness has the output pin termination of the previous example. It is shown in [Figure 3-25 on page 3-129](#). It also drives the test cell inputs and measures the input response at the input pin T1 of the harness for the cell pin CK, and also at the cell pin itself for the second cell pin D.

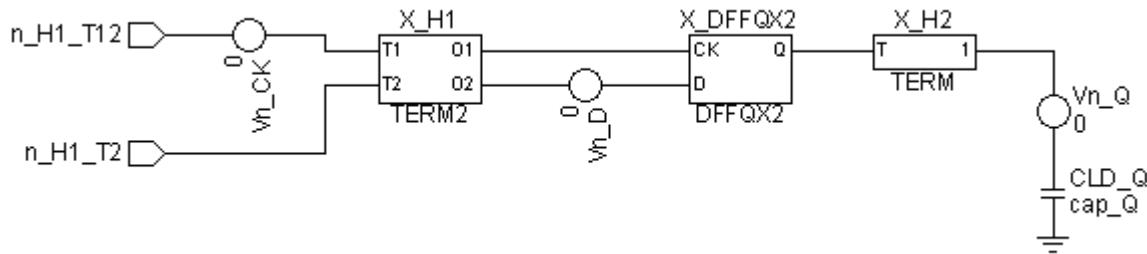
```
ncx_harness H1 {
    harness_netlist_file : term2.spc ;
    pin (CK) {
        harness_connect_pin : O1 ;
        harness_drive_pin : T1 ;
        harness_measure_pin : T1 ;
    }
    pin (D) {
        harness_connect_pin : O2 ;
        harness_drive_pin : T2 ;
    }
}
```

```

ncx_harness H2 {
    harness_netlist_file : term.spc ;
    pin (Q) {
        harness_connect_pin : T ;
        harness_load_pin : 1 ;
        harness_measure_pin : 1 ;
    }
}

```

Figure 3-25 Driver, Termination, and Measurement Harness



Bidirectional I/O Harness

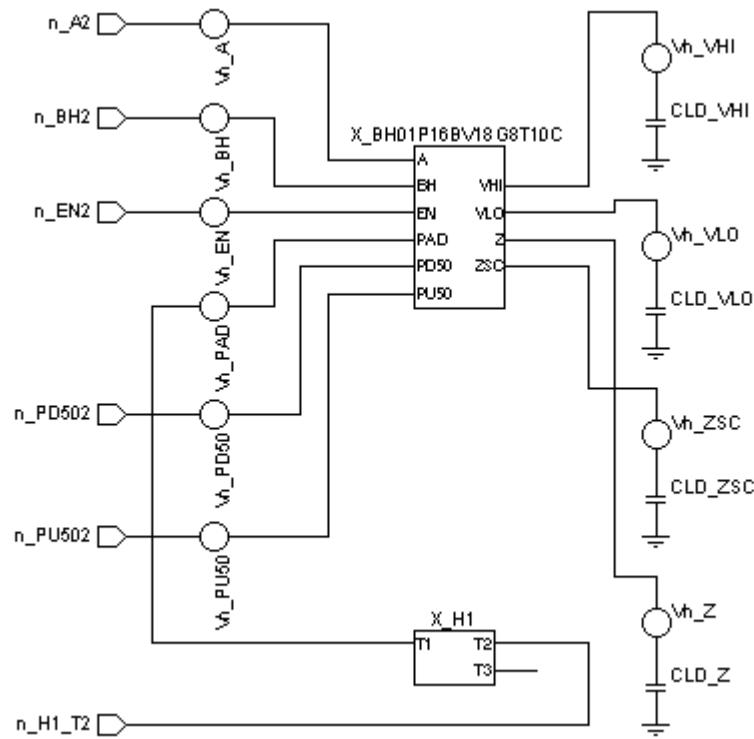
The following harness is attached to a bidirectional pin, using different connections that depend on the pin direction. In the template definition, when the PAD pin is an input pin, it is driven through the harness terminal T2. When the pin is an output pin, the load is connected to the harness terminal T3. The circuit schematics corresponding to the two setups are shown in [Figure 3-26 on page 3-130](#) and [Figure 3-27 on page 3-131](#) for a complex cell containing such a pin.

```

ncx_harness H1 {
    harness_netlist_file : term.spc ;
    pin PAD {
        harness_connect_pin : T1 ;
        # when PAD is in input mode
        harness_drive_pin : T2 ;
        # when PAD is in output mode
        harness_load_pin : T3 ;
    }
}

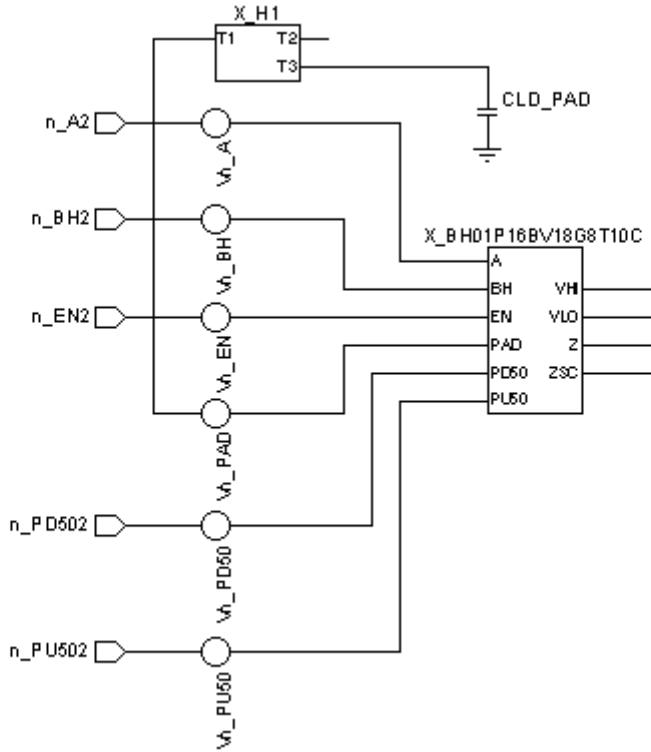
```

Figure 3-26 PAD as an Input Pin



When attaching a harness with a power pin in the interface, if the power pin is global, then the power pin name will be the power node name in the harness instance that is defined in the Liberty NCX-generated SPICE deck. If it is not global, that is, if it is expected to be defined in the cell .subckt interface, then the power node name in the SPICE deck has an “n_” prefix.

Figure 3-27 PAD as an Output Pin



Power and Ground Pin Connections

Liberty NCX adds `pg_pin` and `voltage_map` syntax to all libraries during characterization, by using the `ncx_use_pg_pins` attribute. The `ncx_use_pg_pins` attribute is enabled (set to true) by default. If the seed library or cell template files are defined using the old `rail_connection` virtual power and ground pins, Liberty NCX converts the old `rail_connection` syntax to the new `pg_pin` syntax and issues the following warning message:

"Convert rail_connection format to pg_pin by default"

If you do not want to add `pg_pin` and `voltage_map` syntax to libraries during characterization, you must set `ncx_use_pg_pins` to false.

CCS power characterization requires a power format based on the power and ground pin syntax. Therefore, if you set the `ccs_power` variable in the configuration file to true, enabling `ccs_power`, Liberty NCX ignores the `ncx_use_pg_pins` setting and uses the `pg_pin` format for power modeling.

NLPM modeling supports both the `rail_connection` and `pg_pin` power formats. You can use the `ncx_nlpm_mode` attribute to model the power group per power supply, using the `rail_connection` format, or per power and ground pin, using the `pg_pin` format. The `ncx_nlpm_mode` values are `aggregate` and `split`.

In `split` mode, the NLPM internal and leakage power table is split into multiple tables for each power and ground pin. The sum of the power values under a specific arc or `when` condition in each table should be equal to the corresponding value in the aggregated table. The `split` mode is not valid for a library with a single power supply. Use the `aggregate` mode for libraries with a single power supply. In `aggregate` mode, the default mode, power tables are not modeled for each supply pin.

You can also specify `pg_pin` groups in template files using Liberty syntax, and thereby define multiple supply and ground names that can be used for transistor substrate connections.

Examples

The following example uses the `ncx_use_pg_pins` attribute to generate a `pg_pin` based library:

```
/* lib level */
voltage_map ("VDD",1.100000);
voltage_map ("BIAS1",1.213600);
voltage_map ("BIAS2",1.213600);
voltage_map ("BIASFET",1.800000);
voltage_map ("BUS_BIAS3",0.000000);
voltage_map ("BUS_VSSS",0.000000);
voltage_map ("VDDS",1.800000);
voltage_map ("VSS",0.000000);
operating_conditions ("N_25_1.1_1.8") {
    process : 2.000000;
    temperature : 25.000000;
    voltage : 1.100000;
    tree_type : "balanced_tree";
}
...
...
/* cell level */
cell (OLVDS18G) {
    ...
    cell_leakage_power : 2.372464e+10;
    pg_pin (BIAS1) {
        voltage_name : "BIAS1";
        pg_type : "backup_power";
    }
    pg_pin (BIAS2) {
        voltage_name : "BIAS2";
        pg_type : "backup_power";
    }
}
```

```

        }
        pg_pin (BIASFET) {
            voltage_name : "BIASFET";
            pg_type : "backup_power";
        }
        pg_pin ("BUS_BIAS3") {
            voltage_name : "BUS_BIAS3";
            pg_type : "backup_ground";
        }
        pg_pin ("BUS_VSSS") {
            voltage_name : "BUS_VSSS";
            pg_type : "backup_ground";
        }
        pg_pin (VDD) {
            voltage_name : "VDD";
            pg_type : "primary_power";
        }
        pg_pin (VDDS) {
            voltage_name : "VDDS";
            pg_type : "backup_power";
        }
        pg_pin (VSS) {
            voltage_name : "VSS";
            pg_type : "primary_ground";
        }
    }
    leakage_power () {
        when : "A&!LOPWRA&!LOPWRB&!LPSEL&!PWRDN";
        value : 1.805228e+10;
    }
    leakage_power () {
        when : "!A&!LOPWRA&!LOPWRB&!LPSEL&!PWRDN";
        value : 1.805228e+10;
    }
    /* pin level */
pin (A) {
    ...
    related_power_pin : "VDD";
    related_ground_pin : "VSS";
    internal_power () {
        when : "(!LOPWRA & !LOPWRB & !LPSEL & PWRDN)";
        rise_power ("power_inputs_1") {
            ...
        }
        fall_power ("power_inputs_1") {
            ...
        }
    }
}

```

The following example specifies pg_pin groups and defines multiple supply and ground names:

```
pg_pin VDD1 {  
    voltage_name : VDD ;  
    pg_type : primary_power ;  
}  
pg_pin VSS1 {  
    voltage_name : VSS ;  
    pg_type : primary_ground ;  
}  
pg_pin VDD2 {  
    voltage_name : VNW ;  
    pg_type : backup_power ;  
}  
pg_pin VSS2 {  
    voltage_name : VPW ;  
    pg_type : backup_ground ;  
}
```

The pg_pin group name must match the one in the voltage_map group, which is not a requirement in the Liberty syntax.

Low-Power Modeling

Advanced low-power design methodologies such as multivoltage and multithreshold-CMOS (MTCMOS) require library cells that can support power and ground (PG) pin syntax and switch cells. This section provides an overview of the Liberty NCX support for power and ground pins and coarse-grain multithreshold-CMOS cells.

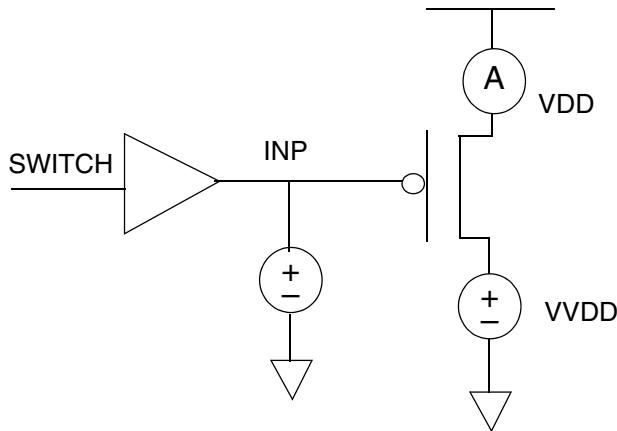
Multithreshold-CMOS Characterization

Coarse-grain multithreshold-CMOS cells are used to switch a block of standard cells on and off. There are two types of coarse-grain switch cells: header switch cells, which control power nets based on a PMOS transistor, and footer switch cells, which control ground nets based on a NMOS transistor.

Liberty NCX supports DC currents for header and footer cells. The dc_current group specifies the DC current through the output pin of the cell (generally the related_internal_pg_pin) in the current units that are specified at the library level by the current_unit attribute. To characterize DC currents for header and footer cells, set the power variable to true in the configuration file.

Timing information must be captured for the internal logic for multithreshold-CMOS cells with internal logic in front of PMOS or NMOS switch transistors. In [Figure 3-28](#), the NLPM timing is captured for the timing arc from the switch pin (SWITCH) to the internal pin (INP). The information is required for rush current calculation. To ensure that timing information is included in your output file, set the `timing` variable to `true` in the configuration file.

Figure 3-28 Switch Cell With Internal Pin



The following information must be specified in the template file or in the seed library:

```
pin "internal_pin_name" {
    direction : internal;
    timing {
        related_pin : switch_pin_name;
        timing_sense : positive_unate / negative_unate;
        ncx_rise_input_net_transition_index : index_number;
        ncx_fall_input_net_transition_index : index_number;
    }
}
```

An example template definition is as follows:

```
pin "I_30:1" {
    direction : internal;
    timing {
        related_pin : SWITCH;
        timing_sense : negative_unate;
        ncx_rise_input_net_transition_index : 0;
        ncx_fall_input_net_transition_index : 0;
    }
}
```

To generate an internal pin and its timing arc information, you must set the following attributes:

`direction`

Specifies the pin type. The `direction` attribute tells Liberty NCX to generate timing information.

`related_pin`

Liberty NCX requires that you set the `related_pin` attribute for internal pins in multithreshold-CMOS cells.

`timing_sense`

Liberty NCX requires that you set the `timing_sense` attribute for internal pins in multithreshold-CMOS cells. You can specify either the `positive_unate` or `negative_unate` values.

`index`

The `input_net_transition` index is required for characterizing 1-d NLDM tables.

Note:

If the pin name includes a colon (:), you must use double quotation marks (" ") in the syntax in the template file, as in the following example:

```
pin "I_30:1" {
```

Transient Leakage Characterization

Transient leakage characterization provides a method for you to specify subtraction of transient or DC leakage from propagating or nonpropagating power and control how transient leakage is measured. The leakage measurements are performed from transient analysis and published in the library. Liberty NCX uses the transient leakage measurement from propagating and nonpropagating power arcs as described in the next subsection.

If you request exhaustive states for leakage, and there is no corresponding propagating or nonpropagating power arc, Liberty NCX generates additional vectors to characterize that state for leakage power using transient analysis.

You can specify the methodology that Liberty NCX uses to perform leakage current measurements (DC or transient) by using the following library, cell, or arc level template attribute setting:

```
ncx_leakage_meas_mode : [ DC | tran ]
```

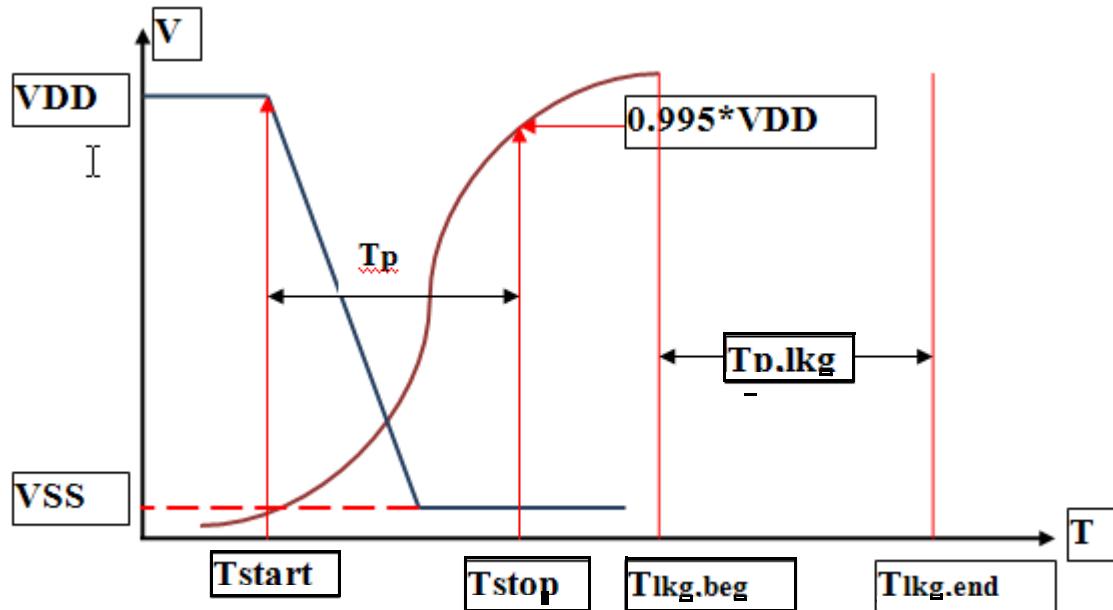
The following template attributes apply to this process:

- ncx_leakage_adj_mode
- ncx_output_power_leakage_meas_interval
- ncx_input_power_leakage_meas_interval
- ncx_leakage_meas_mode
- ncx_leakage_meas_interval

Transient Leakage Subtraction Implementation

Calculating and subtracting transient leakage can be described using propagation power characterization. Liberty NCX calculates the propagation energy as shown in [Figure 3-29](#), which shows a cell with only one power supply VDD, ground VSS, and input and output switching.

Figure 3-29 Propagation Power Characterization



$$\text{Propagation energy} = \int_0^{T_p} VDD * I_{VDD}(t) dt$$

where

- T_{start} = the start of input transition time
- T_{stop} = the time at which the latest rising or falling waveform attains 99.5 percent or 0.5 percent of the VDD or VSS, respectively.
- T_p = the time over which energy is calculated = $T_{stop} - T_{start}$

In order to enable leakage measurement during transient simulation and adjust the propagating and nonpropagating energy published in the output library, the following template attribute is provided:

```
ncx_leakage_adj_mode : [ post_process | sim_based ]
```

The `post_process` setting is the default. It causes the adjustment, or leakage subtraction, to be made in the after-processing stage, using the leakage model from the separate simulation in the `d*` directory. This mode enables both initial and final subtractions. The usage of the DC or transient leakage measurement is based on the `ncx_leakage_meas_mode` setting.

The `sim_based` setting uses the same simulation setup used for power and supports only transient-based simulation adjustment, since the power simulation can only be transient. The same model file is used in the simulation for the adjustment. This mode supports only the “final” leakage state.

For transient leakage adjustment, Liberty NCX measures leakage power during transient simulation and subtracts it from the propagation power calculated in the same simulation. To calculate leakage power, Liberty NCX waits until all nodes have settled to their final value and then calculates leakage power as $VDD * I(VDD)$. You can control the start ($T_{1kg,beg}$) and end time ($T_{1kg,end}$) for leakage power measurement by using the following complex attribute:

```
ncx_output_power_leakage_meas_interval : start_time stop_time
```

where `start_time` and `stop_time` are specified as numbers by which the `Tstop` time is scaled. The default values of `start_time` and `stop_time` are 1.1 and 1.2, respectively, which means

$$T_{1kg,beg} = 1.1 * T_{stop}$$

and

$$T_{1kg,end} = 1.2 * T_{stop}$$

For example, if you specify the following:

```
ncx_output_power_leakage_meas_interval : 1.1 1.5
```

then

$T_{lkg,beg} = 1.1 * T_{stop}$

and

$T_{lkg,end} = 1.5 * T_{stop}$.

Leakage power is calculated by multiplying the average current between $T_{lkg,beg}$ and $T_{lkg,end}$ by the corresponding power supply. This leakage power is subtracted from the dynamic power, and then the adjusted dynamic power is converted to energy and published in the library.

HSPICE Measurement Statements

The HSPICE measure statements used to calculate adjusted dynamic energy for power supply VDD (`nc_true_dyn_energy`) are as follows:

- Average Leakage Current Between $T_{lkg,beg}$ and $T_{lkg,end}$

```
.MEASURE TRAN nc_itran_avg_VDD AVG I(Vn_VDD) FROM = '1.1 * t_simstop'
TO = '1.2 * t_simstop'
```

- Leakage Power Calculation

```
.MEASURE TRAN nc_LKGPWR_VDD PARAM = 'ABS(+nc_itran_avg_VDD*V(n_VDD))'
```

- Dynamic Power Calculation

```
.MEASURE TRAN nc_dyn_pwr_VDD PARAM = 'nc_dyn_eng_VDD/(t_simstop-nc_td)'
```

- Dynamic Power Minus Leakage Power Calculation

```
.MEASURE TRAN nc_pwr_minus_lkg_VDD PARAM = 'nc_dyn_pwr_VDD -
nc_LKGPWR_VDD'
```

- Adjusted Dynamic Energy Calculation

```
.MEASURE TRAN nc_true_dyn_energy_VDD PARAM =
'nc_pwr_minus_lkg_VDD*(t_simstop-nc_td)'
```

Nonpropagation Power Adjustment

Similar to propagation power calculation, nonpropagation power can also be adjusted with transient leakage. You can do this by setting `ncx_leakage_adj_mode : tran`, which controls both propagation and nonpropagation power adjustment. You can control the leakage measurement start and stop time by using a separate attribute:

```
ncx_input_power_leakage_meas_interval : start_time stop_time
```

The values for `start_time` and `stop_time` are numbers that scale the `Tstop` time for nonpropagation power measurement. The default values of `start_time` and `stop_time` are 1.1 and 1.2, respectively. For example, if you specify the following:

```
ncx_input_power_leakage_meas_interval : 1.0 1.5
```

then

$$T_{1kg,beg} = Tstop$$

and

$$T_{1kg,end} = 1.5 * Tstop$$

The `Tstop` time for nonpropagation power is calculated as follows:

```
Tstop = max (nc_td+5.0*iptr, nc_td+500ps)
```

where

`nc_td` = start of inputs switching

and

`iptr` = input transition time

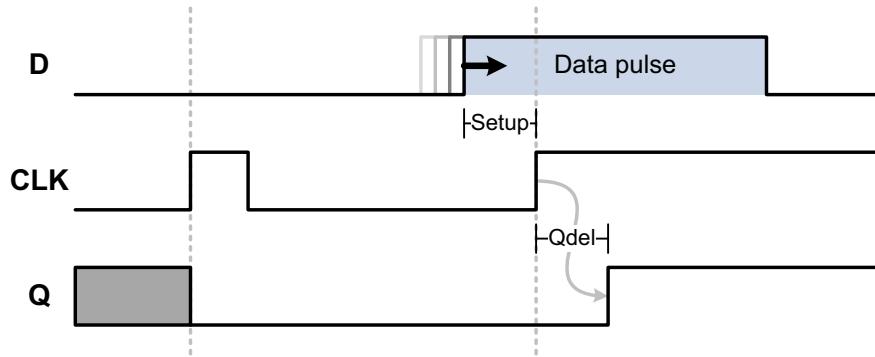
Leakage power is calculated by multiplying the average current between $T_{1kg,beg}$ and $T_{1kg,end}$ by the corresponding power supply. This leakage power is subtracted from the nonpropagation power and then the adjusted nonpropagation power is converted to energy and published in library.

Constraint Methodologies

The setup and hold times of a constraint arc are characterized by capturing a new data value into a sequential cell, and searching for the minimum setup and hold time relationships between the data and clock signals that do not trigger a violation event.

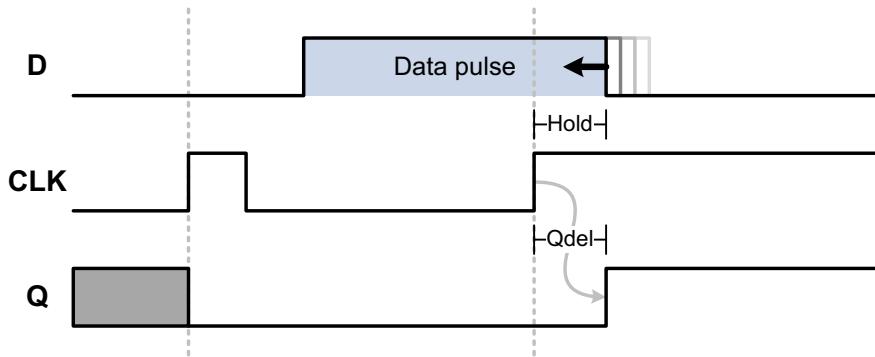
A setup constraint arc is characterized by pushing the leading data edge (before the active clock edge) successively later towards the clock edge, until a violation is detected. This process is shown in [Figure 3-30](#).

Figure 3-30 Characterization of Setup Constraint Arc



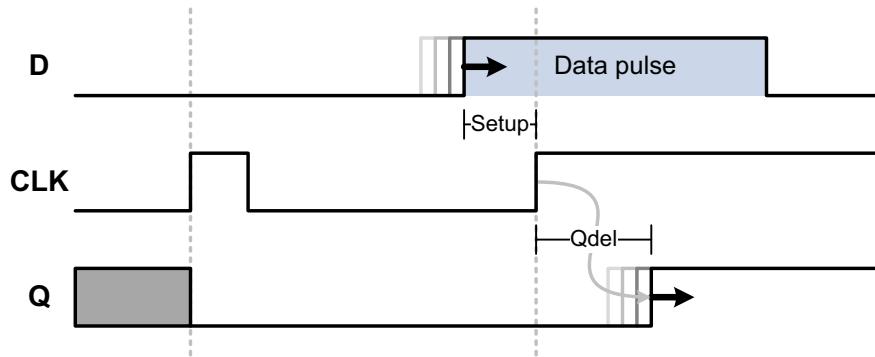
A hold constraint arc is characterized by pushing the trailing data edge (after the active clock edge) successively earlier towards the clock edge, until a violation is detected. This process is shown in [Figure 3-31](#).

Figure 3-31 Characterization of Hold Constraint Arc



For constraint arc characterization, a violation condition represents a condition where the sequential cell is capturing outside its intended range of behavior. One type of violation is *delay degradation*. As the data edge is moved closer to the active clock edge, the clock-to-output delay might deviate (or *degrade*) from its nominal value. When the delay deviates from nominal in either direction by more than a predetermined amount, a delay degradation violation occurs. An example of a delay degradation is shown in [Figure 3-32](#).

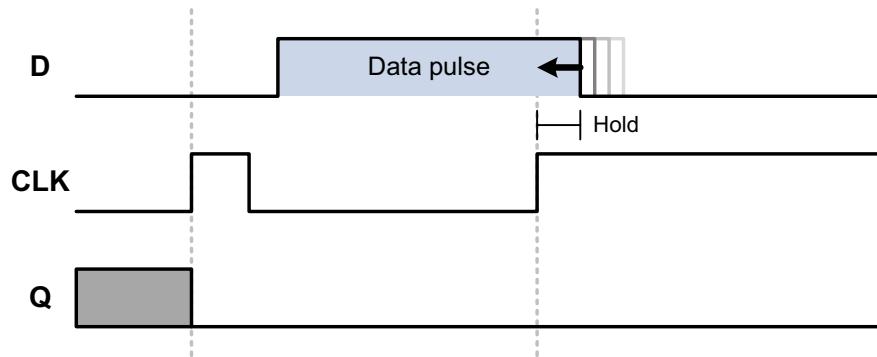
Figure 3-32 Delay Degradation Violation Mode



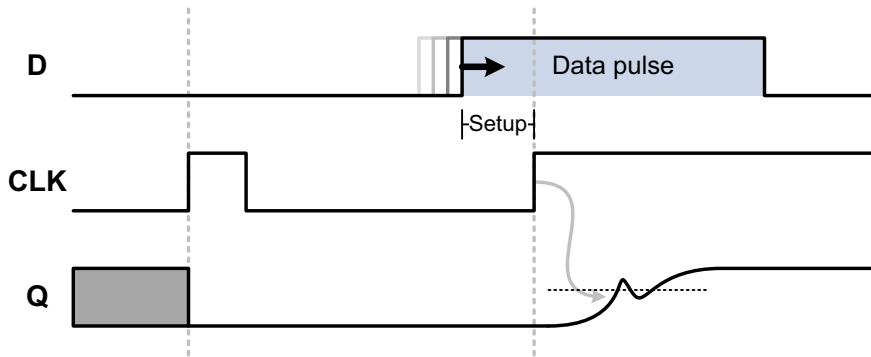
In addition to the delay degradation violation type, there is also a *slew degradation* violation type.

The *pass-fail* violation represents the case where the data edge has been pushed too close to the active clock edge, and the sequential cell can no longer capture the data. An example of a pass-fail violation is shown in [Figure 3-33](#).

Figure 3-33 Pass-Fail Violation Mode for Toggled Output



The *glitch* violation represents the case where the output pin transition is not clean, and exhibits multiple crossings of certain voltage thresholds in its voltage response. An example of a glitch violation is shown in [Figure 3-34](#).

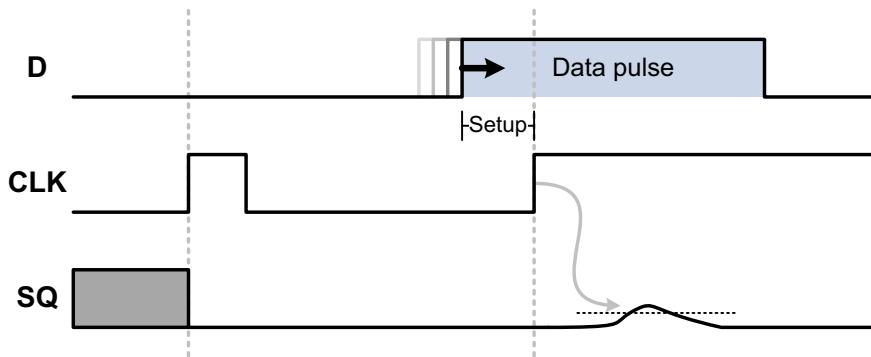
Figure 3-34 Glitch Violation Mode for Toggled Output

In constraint arc characterization, outputs may be toggled or non-toggled. A toggled output is expected to change its output state during the constraint arc characterization period.

Examples of toggled outputs are the Q pins of flip-flops for setup and hold constraint arcs, and the Q pin of a latch for setup constraint arcs. A non-toggled output is expected to hold its output state during the constraint arc characterization period. Examples of non-toggled output are the Q pin of a latch for hold constraint arcs, and some clock gating cell setup and hold arcs.

The delay and slew degradation violations pertain only to toggled outputs, because they are defined according to a measurable output response. The pass-fail and glitch violation types apply to both toggled and non-toggled arcs. A pass-fail violation for a non-toggled arc represents the case where an output toggle unexpectedly occurs. A glitch violation for a non-toggled arc represents the case where the waveform exceeds a voltage threshold before returning back to its original rail value. By default, a non-toggled glitch is one where a low response rises above 10 percent of ground rail voltage, or where a high response falls below 90 percent of rail voltage.

An example of a glitch violation for a non-toggled output is shown in [Figure 3-35](#).

Figure 3-35 Glitch Violation Mode for Non-Toggled Output

Simple Violation Definitions

When the `violation_mode` attribute is not set, a simple violation specification mode is used. In this simple mode, a violation for a toggled arc can be defined using one of the following methods:

- a relative amount of delay degradation for toggled arcs
- an absolute amount of delay degradation for toggled arcs

By default, a violation is defined as a 10 percent delay degradation in CLK-to-output delay. You can set the `violation_delay_degrade_pct` attribute to change the relative delay degradation limit. Alternatively, if an absolute delay degradation limit is desired, you can set the `violation_delay_degrade` attribute, which takes priority over the relative limit. If a cell has multiple output pins, the delay degradation is checked against the pin with the largest output delay.

In the simple violation mode, non-toggled arcs are always checked using the pass-fail violation method.

This simple violation specification mode is supported by all simulators.

Advanced Violation Definitions

A more advanced violation specification mode is available by setting the `violation_mode` attribute to a violation definition value. This advanced violation mode is supported by HSPICE. It is also supported by Eldo and Spectre when the Generic Simulator Interface is used.

In the advanced violation mode, a violation can be defined using one or more of the following methods shown in [Table 3-11](#).

Table 3-11 `violation_mode` Methods

Method	Description	Optional parameter
<code>delay_degrade_pct</code>	Check for delay degradation using relative delay percentage value	Percentage value (default value is the <code>violation_delay_degrade_pct</code> attribute value)
<code>delay_degrade</code>	Check for delay degradation using absolute delay value in library time units	Absolute delay in library time units (default value is the <code>violation_delay_degrade</code> attribute value)

Table 3-11 violation_mode Methods (Continued)

Method	Description	Optional parameter
slew_degrade_pct	Check for slew degradation using relative slew percentage value	Percentage value (default value is the violation_slew_degrade_pct attribute value)
slew_degrade	Check for slew degradation using absolute slew value in library time units	Absolute slew in library time units (default value is the violation_slew_degrade attribute value)
glitch	Check for glitch behaviors at specified glitch thresholds	Lower, upper threshold percentage values (default value is the violation_glitch_lower_pct and violation_glitch_upper_pct attribute values)
passfail	Check for data capture success or failure	

By default, when a violation method is specified in a violation definition, it applies to all cell output pins. For example, the following cell template definition considers the relative delay degradation of both the Q and QN output pins.

```
pin Q {...}
pin QN {...}
violation_mode : delay_degrade_pct ;
```

When multiple violation types are specified, all violation criteria are considered simultaneously during constraint arc characterization. The constraint arc is characterized so that none of the violation events occur. For example, the following cell template definition considers both the relative and absolute delay degradation of the Q and QN output pins.

```
pin Q {...}
pin QN {...}
violation_mode : delay_degrade_pct delay_degrade ;
```

You can limit each violation method to a specified pin or internal node by specifying the pin or node name after the violation method keyword. Multiple pins or nodes can be specified using multiple methods, each with a pin or node. For example, the following cell template definition considers relative delay degradation only on the Q pin, and glitch detection on two internal nodes:

```
pin Q {...}
pin QN {...}
violation_mode : delay_degrade_pct Q glitch Int1 glitch "Int:2" ;
```

Double quotation marks are required if the internal node name contains a colon.

Some violation methods have optional parameters, as shown in the “Optional parameter” column of [Table 3-11](#). When not specified, these parameters are set to the default attribute values shown in the table. However, these parameters can also be set directly as a part of the violation definition. This syntax requires the pin or node name to be specified. For example,

```
violation_mode : delay_degrade_pct Q 5 glitch Int1 10 90 ;
```

If needed, you can use the `violation_mode_rise` and `violation_mode_fall` attributes to specify different violation definitions for rising and falling output data edges, respectively.

The following examples are cell template examples that demonstrate more complex `violation_mode` definitions:

Example 1

```
# Change relative delay degradation default to 20%
violation_delay_degrade_pct : 20 ;

# Change absolute delay degradation default to 0.03 (lib time unit)
violation_delay_degrade : 0.03 ;

# Change glitch detection upper, lower threshold defaults to 25%, 75%
violation_glitch_lower_pct : 25 ;
violation_glitch_upper_pct : 75 ;

# Define a node set of interest
ncx_node_set : set1 Q1 int1 ;

# Now:
# * Apply 20% default relative delay degradation to all external
#   output pins
# * Apply 0.03 default absolute delay degradation to pin Q
# * Apply passfail to all external pins
# * Apply 25/75 lower/upper default glitch thresholds to internal
#   pin int2
# * Apply 10 (lower) and 90 (upper) glitch thresholds to all nodes
#   in set1
violation_mode : \
  delay_degrade_pct \
  delay_degrade Q \
  passfail \
  glitch int2 \
  glitch set1 10 90 ;
```

Example 2

```
# Apply glitch to all hold_rising arcs of latch cells
ncx_condition latch_hold_arc {
```

```

ncx_condition_cell_type    : latch ;
ncx_condition_arc_type     : constraint ;
ncx_condition_timing_type  : hold_rising ;
violation_mode : glitch;
}

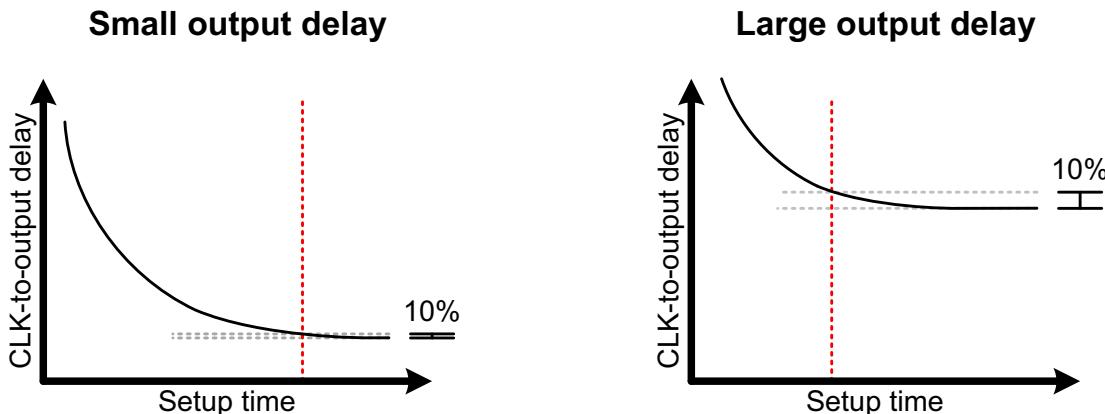
# For setup_rising arcs of all latch cells, apply different violation
# definitions to rise_constraint and fall_constraint respectively
ncx_condition latch_setup_arc {
    ncx_condition_cell_type    : latch ;
    ncx_condition_arc_type     : constraint ;
    ncx_condition_timing_type  : setup_rising ;
    violation_mode_rise : delay_degrade_pct    delay_degrade      glitch;
    violation_mode_fall : delay_degrade      glitch;
}

```

Configuring Delay Degradation Violations

When a delay degradation violation is defined using the `delay_degrade_pct` violation type, the allowed change in output delay is specified as a percentage of the nominal delay. For some cell types or table points, the output delay may already be a small delay value, and taking a percentage of that results in an even smaller delay value. Such a small delay change trigger value can cause the constraint search to end early, resulting in an unreasonably large constraint value, as shown in [Figure 3-36](#).

Figure 3-36 Delay Degradation as a Function of Output Delay



The `violation_delay_degrade_min_value` template attribute specifies an alternative minimum delay change value used to avoid such sensitive small-value delay degradation violations. The default value of this attribute is 5ps, converted to library time units. When a

delay degradation search is performed, if the computed delay degradation amount is less than the minimum value specified by the `violation_delay_degrade_min_value` attribute, the minimum value will be used instead.

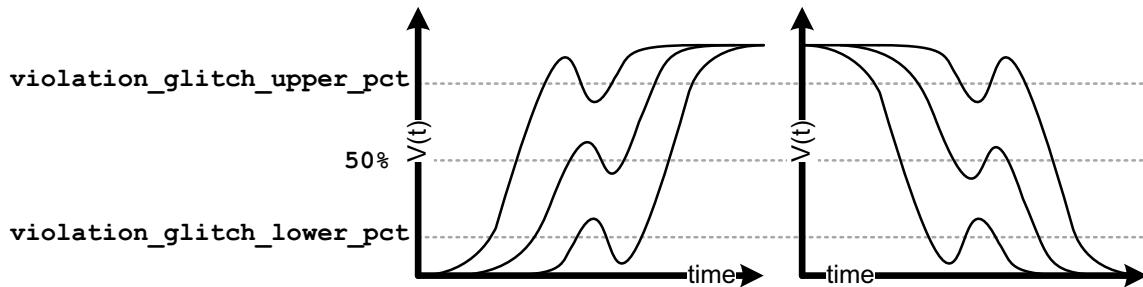
For delay degradation, a constraint arc is characterized such that the output delay does not vary from nominal – in either direction – by more than the degradation value as the data edge approaches the clock edge. By default, the sign of the specified degradation value is ignored. To ensure that the output delay moves in the expected direction, you can set the `violation_delay_degrade_check` attribute to `true`. In this case, a positive delay degradation value specifies that the output delay is expected to increase as the data edge approaches the clock edge, and a negative value specifies that the output delay is expected to decrease. If the output delay does not move in the expected direction, Liberty NCX issues an error message.

Configuring Glitch Violations

By default, the glitch violation method looks for crossings at three voltage thresholds - the lower, middle, and upper voltage thresholds. The default thresholds are 10, 50, and 90 percent of rail voltage. You can configure the upper and lower thresholds using the `violation_glitch_upper_pct` and `violation_glitch_lower_pct` attributes, respectively.

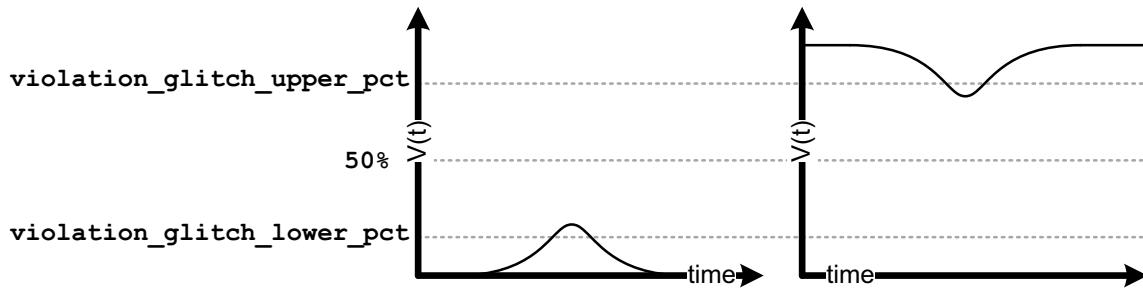
For toggled outputs, a glitch violation occurs when the signal exhibits multiple crossings at any of the glitch voltage thresholds. See [Figure 3-37](#). A glitch violation also occurs if the signal does not cross a voltage threshold at all. As a result, a glitch violation check for a toggled output includes a pass-fail check.

Figure 3-37 Glitch Thresholds for Toggled Outputs and Nodes



For non-toggled outputs, a glitch violation occurs if a signal reaches any of the glitch voltage thresholds. Because a non-toggled output cannot reach the middle threshold without crossing either the lower or upper threshold first, non-toggled glitch violations use only the lower and upper voltage thresholds. See [Figure 3-38](#).

Figure 3-38 Glitch Thresholds for Non-Toggled Outputs and Nodes



Alternatively, you can specify a list of multiple threshold levels that are used to detect failures caused by glitches. Use the `violation_glitch_thresholds_pct` attribute in the library or cell template to specify the list of desired percentage thresholds. An output waveform that crosses any given threshold more than once is considered a failure. For example,

```
violation_mode : delay_degrade_pct Q glitch Int1 glitch "Int:2" ;
violation_glitch_thresholds_pct : 10 20 30 40 50 60 70 80 90 ;
```

In this example, if the output waveform crosses any one of the nine specified percentage threshold levels more than once, it is considered an output glitch failure.

When a delay degradation violation check is applied to a pin or node that never toggles, Liberty NCX converts the check to a glitch check.

Defining Node Sets

You can use the `ncx_node_set` attribute to define internal node or output node sets. Instead of specifying `violation_mode` for each internal node and output node, use `ncx_node_set` to combine the nodes into sets, as shown in the following example:

```
ncx_node_set : set1 Q1 int1 ;
ncx_node_set : set2 Q2 "c:3" ;
```

After you combine the nodes into sets, you can then reference the sets in the `violation_mode` definition. As the following example shows, you can set `violation_mode` to `glitch` to enable glitch detection mode and set `violation_mode` to `passfail` to perform pass and fail violations:

```
violation_mode: glitch set1 20 80 \
    passfail set2 ;
```

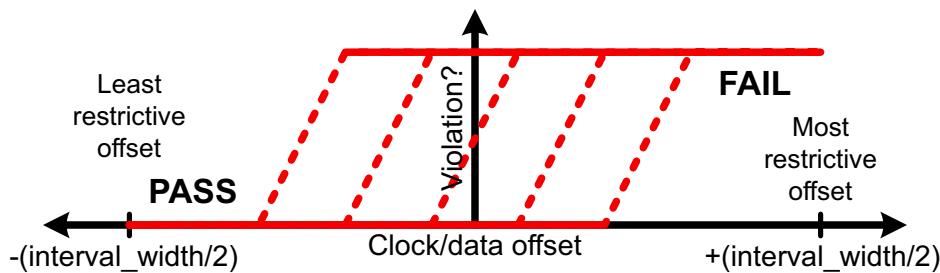
In the example, `violation_mode` set to `glitch` enables glitch detection mode for all nodes in `set1`, and `violation_mode` set to `passfail` performs pass and fail violations for all nodes in `set2`.

Accuracy Settings for Constraint Acquisition

For each constraint violation type, Liberty NCX varies the data input across a range of possible timing offset values relative to the clock input, and determines the timing offset where the cell behavior switches from non-violating to violating. The total width of this search interval is controlled by the `ncx_constraint_search_interval` attribute.

The default search interval width is 2ns, converted to library time units. The search interval for most constraint arcs is centered around the clock and data edges coinciding. This means that the default search interval results in the data edge varying within $\pm 1\text{ns}$ from the clock edge, as shown in [Figure 3-39](#). For minimum pulse width checks, the search interval varies between a zero pulse width and a pulse width equivalent to the search interval width.

Figure 3-39 Search Interval for Constraint Violation Search



For all violation types, it is assumed that the violation condition does not occur at the least restrictive end of the search interval (data leading for setup, data trailing for hold), and does occur at the most restrictive end of the search interval (data trailing for setup, data leading for hold). For the delay and slew degradation violation types, the reference nominal delay or slew is measured at the least aggressive end of the search interval.

Liberty NCX checks to ensure that a violation occurs at the most restrictive constraint value and does not occur at the least restrictive value. If not, it doubles the interval width and checks again, using the new most-restrictive and least-restrictive constraint values. It repeats this doubling process, if necessary, until it finds an appropriate interval width.

If you know that the search interval will be larger than 2ns for some cells, you can specify the initial interval width with the `ncx_constraint_search_interval` attribute. Specifying a larger initial search interval can reduce the runtime for cells which require the larger interval, but it can increase runtime if the interval is too large for most cells. If only a few special or complex cells need a wider search interval, specify the wider interval in the appropriate cell template file, or inside an `ncx_condition` group in the library template file.

Once the search interval is established, the constraint violation search uses the bisection method to determine the transition point between the passing and failing violation condition. The bisection method determines the violation status at three points along an interval (lower

value, midpoint value, and upper value), determines which side of the midpoint the violation transition occurs, and iteratively repeats the process on this new halved interval until the precise violation transition point is determined. By default, this process repeats until the the interval value decreases to 2ps, at which point the search ends. This final simulation could represent either a passing or failing condition, within an error range of $\pm 2\text{ps}$.

To ensure conservatism, Liberty NCX adds the 2ps minimum interval value to the final simulated constraint value. If finer accuracy is needed (or faster runtime is needed), a smaller (or larger) minimum search interval width can be specified with the `ncx_constraint_accuracy` attribute. The final adjustment ensures that the final constraint value is always conservative.

The attributes described in this section affect both simple and advanced violation definitions.

User-Defined Attributes

In addition to the standard Liberty library attributes, the Liberty format also supports user-defined attributes. User attributes allow a library to be augmented with library-level, cell-level, and pin-level information supplied by the user.

In a Liberty library file, user-defined attributes must be predefined at the top-level library scope using the `define` construct:

```
library ("my_lib") {  
    ...  
    define(my_lib_attr,library,string);  
    define(my_cell_attr,cell,string);  
    define(my_pin_attr,pin,string);
```

Each `define` construct defines the name of the user attribute, the object type, and the data type of the attribute. Once the attribute has been predefined in at the library level, it can be set to a value just as with any other Liberty attribute.

You can provide user attribute information in library and cell template files, and Liberty NCX will include the user attribute definitions in the output library. User attributes are predefined at the top-level library scope as needed.

To define a user attribute in a template file, prefix the attribute definition with the `define` template keyword:

```
define user_attr_name: user_attr_value;
```

To define a library user attribute, place the definition in a library template file. To define a cell user attribute, place the definition in a cell template file. To define a pin user attribute, place the definition inside a pin definition group in a cell template file.

Note:

All user attributes are defined as `string` data type attributes in the output library.

4

I/O Cell Characterization

You can customize features in Liberty NCX for I/O cell characterization. The features in I/O cell characterization include handling high pin counts, complex functionality, dynamic slew control and output drive strength, and variable voltage levels. You can also capture and model special cell behaviors such as large delays, large capacitance drive capability as reflected by maximum capacitance, non rail-to-rail swings, and power models.

The I/O Cell characterization process is described in the following sections:

- [I/O Cell Overview](#)
- [Examples of Special Cases](#)
- [Voltage Group Support](#)
- [I/O Cell .nodeset Feature](#)
- [IBIS Model Generation](#)

I/O Cell Overview

I/O cell characterization has the following customization capabilities:

- Arc synthesis for one or more states of a user-specified, subset of input pins.
- Support for arc-specific, signal level specifications on a pin and automatic detection of signal levels on an arc basis.
- Pin-based control over design rules (maximum capacitance and maximum transition) acquisition.

Use the `ncx_create_arcs` attribute to direct Liberty NCX to select the first valid, best, or worst side-pins whose states are not specified.

The following attributes can be specified only at the pin level and do not appear in the output library if specified at the arc level:

```
input_signal_level_high
output_signal_level_high
input_signal_level_low
output_signal_level_low
```

Use the `non_rail_output_signal_level` attribute at the arc level to enable auto signal-level acquisition at the arc-level.

Use the `output_signal_level_z` attribute to specify the output voltage level of a three-state pin when it is in the high-impedance state. It is used in conjunction with the signal level attributes described for the other `signal_level` attributes to characterize three-state enable arcs. An example is

```
output_signal_level_z : value ;
```

Use the following pin-based attributes to control extraction of design rules for that pin:

```
ncx_update_max_capacitance : [true]|false
ncx_update_max_transition : [true]|false ;
```

By default, if Liberty NCX is set up to extract design rules, they will be extracted for every relevant pin. If either of these attributes is set to false, the appropriate design rules are not modified, and any values that you provide are retained in the output library generated by Liberty NCX.

Use the `ncx_condition` group to specify these parameters for a range of cells or pins without having to edit each individual cell template, as shown in the following example:

```
ncx_condition keep_dr {
  ncx_condition_cell_name : cell_name ;
```

```

ncx_condition_pin_name : pin_name ;
ncx_update_max_capacitance : false ;
ncx_update_max_transition : false ;
}

```

This example preserves the maximum capacitance and maximum transition values on the specified pin.

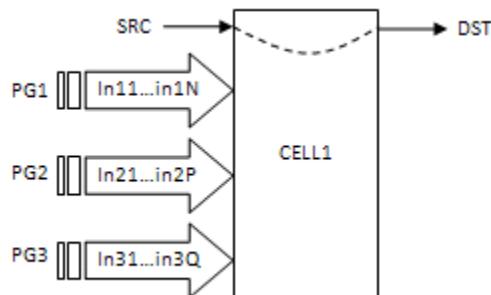
Examples of Special Cases

The following examples describe the design flexibility available with I/O cell customization. They show the following special cases for creation of the arc (SRC to DST), along with the template syntax.

- Default arc: worst (PG1/PG2), user (PG3)
- Default arc: first (PG1), worst (PG2), user (PG3)
- State arc: all (PG1), unpublished-worst (PG2/PG3)
- State arc: user (PG1), unpublished-first (PG2), unpublished-user (PG3)
- State arc: user (PG1), unpublished-worst (PG2), unpublished-user (PG3)

Figure 4-1 shows for a single arc from an input pin named SRC to an output pin named DST in a cell named CELL1. All other side-input pins are classified into three groups, PG1, PG2, and PG3.

Figure 4-1 Illustration of a Single Arc Input Pin (SRC) to Output Pin (DST)



Default arc: worst (PG1/PG2), user (PG3)

This example selects the state with the worst delay from all states of PG1 and PG2, and user-specified states for PG3.

```

# Specify default arc creation
ncx_create_arcs : CELL1 SRC DST delay states default ;
# Specify worst default arc mode
default_arc_mode : worst ;
# Set states of pins in PG3 group
ncx_condition def_arc_condn {
    ncx_condition_cell_name : CELL1 ;
    ncx_condition_pin : SRC ;
    ncx_condition_related_pin : DST ;
    ncx_when : in31&...&in3Q ;
}

```

The determination of the worst state is done in the pre-analysis stage, and is stored in the template in the `ncx_optimization` construct as the `ncx_opt_default_arc` attribute.

Default arc: first (PG1), worst (PG2), user (PG3)

This example selects the state with the worst delay from the first valid state of PG1 pins, all states of PG2 pins, and user-specified states of PG3 pins.

```

# Specify desired state arc creation
ncx_create_arcs : CELL1 SRC DST delay states all in11 in12 ... in1N;

```

The `when` conditions for these arcs are constructed only from PG1 pins. These arcs are characterized during the regular characterization phase.

State arc: all (PG1), unpublished-worst (PG2/PG3)

This example creates arcs for all states of PG1 pins, using the worst valid condition for PG2 and PG3 pins. List pin names after the `all` token in the `ncx_create_arcs` attribute.

```

# Specify desired state arc creation
ncx_create_arcs : CELL1 SRC DST delay states all in11 in12 ... in1N;
# Select worst delay of remaining pin states
ncx_delay_sensitization_mode : worst ;

```

These arcs are analyzed during the sensitization phase to find the worst state of the PG2 and PG3 pins for each arc. They are then assigned the sensitization that is used during the regular characterization phase. The `when` conditions for these arcs are constructed only from PG1 pins.

State arc: user (PG1), unpublished-first (PG2), unpublished-user (PG3)

This example creates a state arc with a `when` condition constructed from the user-specified state of PG1 pins, the first unpublished valid state of PG2 pins, and the unpublished user-specified state of PG3 pins.

```
# Specify desired state arc creation
ncx_create_arcs : CELL1 SRC DST delay states "in11&...&in1N";
# Set states of pins in PG3 group
ncx_condition def_arc_condn {
    ncx_condition_cell_name : CELL1 ;
    ncx_condition_pin : SRC ;
    ncx_condition_related_pin : DST ;
    ncx_when : in31&...&in3Q ;
}
```

This arc is characterized during the regular characterization phase.

State arc: user (PG1), unpublished-worst (PG2), unpublished-user (PG3)

This example creates a state-arc with a `when` condition constructed from the user-specified state for PG1 pins, the worst of all valid states of PG2 pins, and the unpublished user-specified state of PG3 pins

```
# Specify desired state arc creation
ncx_create_arcs : CELL1 SRC DST delay states "in11&...&in1N";
# Set states of pins in PG3 group
ncx_condition def_arc_condn {
    ncx_condition_cell_name : CELL1 ;
    ncx_condition_pin : SRC ;
    ncx_condition_related_pin : DST ;
    ncx_when : in31&...&in3Q ;
}
# Select worst delay of remaining pin states
ncx_delay_sensitization_mode : worst ;
```

This arc is analyzed during the sensitization phase to find the worst state of the PG2 pins. It is then assigned the sensitization that will be used during the regular characterization phase.

Voltage Group Support

I/O pads are cells at the chip boundaries that communicate with devices external to the chip. Because of this fact, I/O cells behave differently than other cells in the chip. One of these differences is the voltage level at which input pads transfer logic 0 or 1 into or out of the chip. Liberty NCX uses voltage groups to specify these voltage levels.

Voltage group support affects how Liberty NCX generates input signals and where it makes delay measurements. Input and output voltage groups are supported in the Liberty NCX characterization flow. You can use these voltage groups instead of trip-points and thresholds in netlist model generation and netlist measurements.

These voltage groups are described in the following sections:

- [Voltage Groups Overview](#)
 - [Voltage Groups in the Library Group](#)
 - [Netlist Changes](#)
 - [Delay Calculation with Voltage Groups](#)
 - [Voltage Group Limitations](#)
-

Voltage Groups Overview

Two types of voltage groups can be defined for a PAD pin. To use them, you should set the `is_pad` property to true for the given pin. An input voltage group is used when the pad is an input pad and is specified as an `input_voltage_group` in Liberty NCX syntax. An output voltage group is used when the pad is an output pad and is specified as an `output_voltage_group`. If the pad is of inout type, then you can define both input and output voltage groups for it.

When you specify a valid voltage group, then its values overwrite the default minimum and maximum voltage and delay measurement thresholds that would normally be used.

If `output_threshold_rise` and `output_threshold_fall` library values are applicable, then the voltage groups do not take precedence. This allows you to change thresholds on a per-pin basis if necessary.

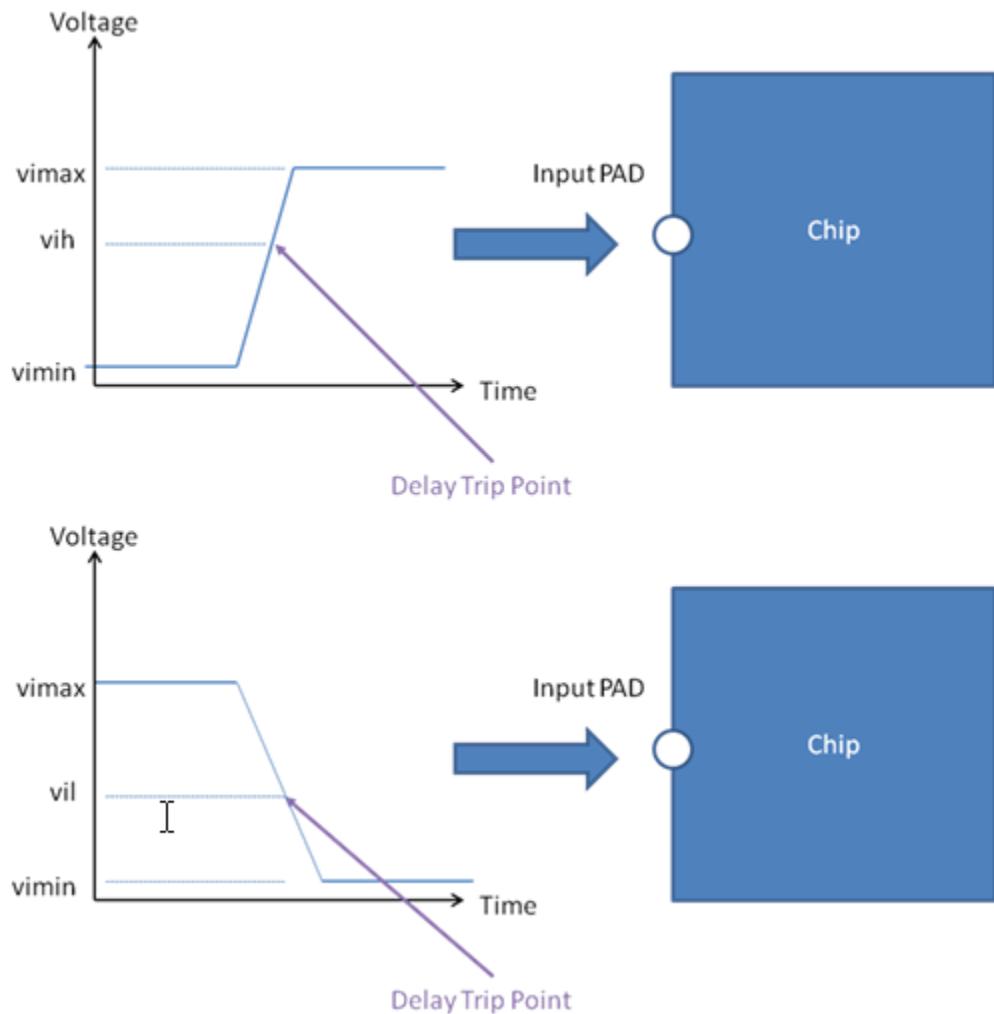
Input Voltage Group Elements

An input voltage group contains the following four elements:

- v_{il} - The maximum input voltage for which the input pad is guaranteed to be logic 0.
- v_{ih} - The minimum input voltage for which the input pad is guaranteed to be logic 1.
- v_{imin} - The minimum acceptable input voltage.
- v_{imax} - The maximum acceptable input voltage.

The diagrams in [Figure 4-2](#) indicate how these values are applied to a PAD pin for a rising and falling signal, respectively.

Figure 4-2 Example of the Relationship Between Voltage Groups and Input Signals to I/O Pads



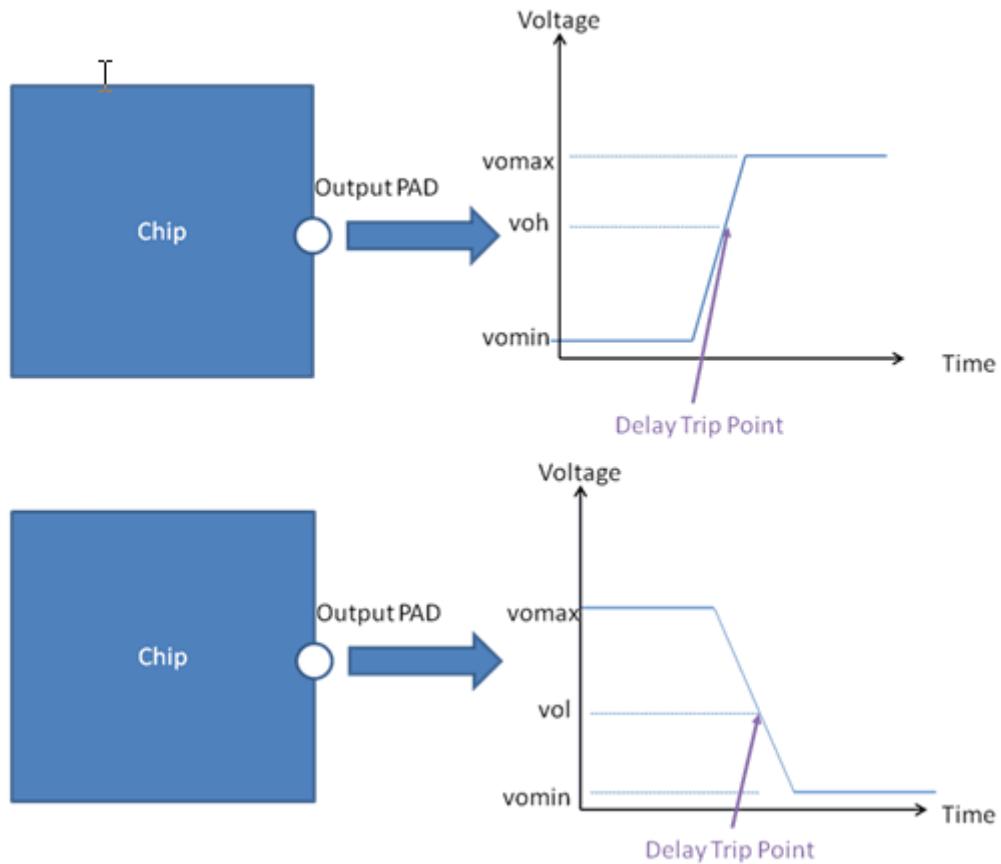
Output Voltage Group Elements

An output voltage group contains the following four elements:

- v_{ol} - The maximum voltage that represents a logic 0.
- v_{oh} - The minimum output voltage that represents a logic 1 or the maximum input voltage for which the input to the core is guaranteed to be a logic 1.
- v_{omin} - The minimum output voltage a pad can generate.
- v_{omax} - The maximum output voltage a pad can generate.

The diagrams in [Figure 4-3](#) indicate how these values would be applied for an output pad pin for a rising and falling signal, respectively.

Figure 4-3 Example of the Relationship Between Voltage Groups and Input Signals to I/O Pads



Voltage Groups in the Library Group

Voltage groups are part of the library group and are defined by the following syntax:

```
library (namestring) {
    ...
    input_voltage (namestring) {
        vil : float ;
        vih : float ;
        vimin : float ;
        vimax : float ;
    }
    ...
    output_voltage(namestring) {
        vol : float ;
        voh : float ;
        vomin : float ;
        vomax : float ;
    }
    output_voltage (namestring) {
        ... output_voltage description ...
    }
}
```

Library and Cell Template

The following example shows a Liberty NCX library template.

```
library (myLib) {
    input_voltage(CMOS) {
        vil : 0.3 ;
        vih : 0.7 ;
        vimin : -0.5 ;
        vimax : 1.5 ;
    }
    input_voltage(TTL_5V) {
        vil : 0.8 ;
        vih : 2.0 ;
        vimin : -0.5 ;
        vimax : 1.5 ;
    }
    output_voltage(GENERAL) {
        vol : 0.4 ;
        voh : 2.4 ;
        vomin : -0.3 ;
        vomax : 1.3 ;
    }
}
```

Once the voltage groups have been defined in the library group you can associate them with each pin as shown in the following cell template.

```

pin(PAD) {
    is_pad : true;
    direction : output;
    output_voltage : GENERAL;
    slew_control : high;
    ...
}

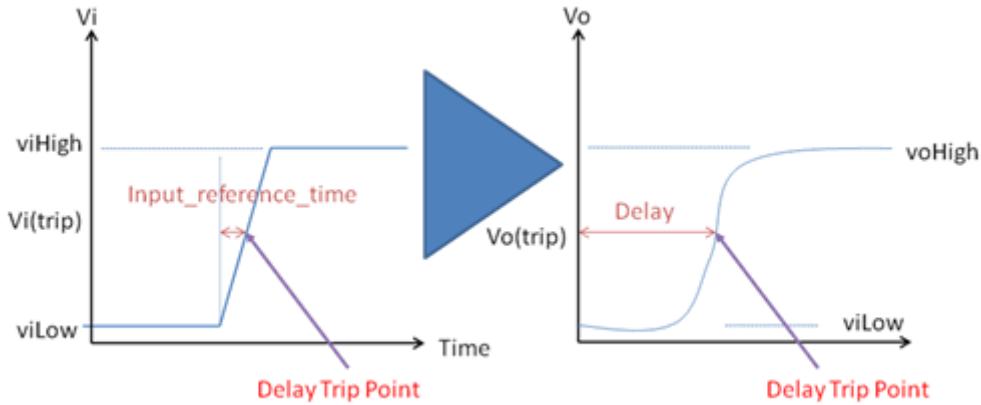
```

Netlist Changes

When a voltage group has been defined for a pin, Liberty NCX modifies the way in which the signals used in SPICE netlists are generated. It also affects where certain delay measurements are taken. The impact of voltage groups on input and output I/O pads is illustrated in [Figure 4-2 on page 4-7](#) and [Figure 4-3 on page 4-8](#). Changes are made to netlists to generate signals and perform the measurements illustrated by using voltage groups instead of delay trip points and thresholds.

In order to clarify how voltage groups are used, see [Figure 4-4](#) to review how voltages, delay trip points, and thresholds are used in Liberty NCX generated netlists. It highlights an input signal driving a buffer with an exponential output.

Figure 4-4 PWL Input Signal Driving a Buffer with an Exponential Output



Delay Calculation with Voltage Groups

When voltage groups are specified, the delay calculation shown in [Figure 4-4 on page 4-10](#) actually becomes simpler. Instead of using thresholds to specify delay trip points, actual voltages are used.

```

input_voltage (namestring) {
    vil : float ;

```

```

        vih : float ;
        vimin : float ;
        vimax : float ;
    }
...
output_voltage(namestring) {
    vol : float ;
    voh : float ;
    vomin : float ;
    vomax : float ;
}

```

The voltage groups specify the voltage values to be used. In [Figure 4-4 on page 4-10](#), the specified input voltages for rising signals correspond to the following voltage group values.

- `vimax = viHigh`
- `vimin = viLow`
- `vih = Vi(trip)`

Likewise, the specified output voltages correspond to the following voltage group values.

- `vomax = viHigh`
- `vomin = viLow`
- `voh = Vo(trip)`

The only difference between the specified voltages for rising signals and for falling signals is that for falling signals, `vil` and `vol` are used instead of `viH` and `voh`, respectively.

Voltage Group Limitations

Liberty syntax allows for voltage group values to be specified by using expressions; for example,

```
vil = VDD*0.3
```

However, Liberty NCX does not actually support this syntax.

I/O Cell .nodeset Feature

Some types of cells (primarily I/O cells) have level shifters connected to the input pins. Such level-shifter cells have internal nodes that must be set to logic `high` or logic `low` at time `t=0` for SPICE simulation.

The internal nodes value at time $t=0$ is determined by the condition of the input pin for all types of simulations (as in, delay, power, capacitance, leakage, and so forth). Use the .nodeset feature to preset such internal nodes prior to the characterization.

Liberty NCX provides the following attributes to use the nodeset feature:

- ncx_internal_pin_nodeset
- ncx_internal_pin_signal_level_high
- ncx_internal_pin_signal_level_low

.nodeset is an I/O cell characterization feature. It provides the following attributes:

- ncx_internal_pin_nodeset Allows you to direct Liberty NCX to initialize an internal node to a specific voltage level in the SPICE simulation deck. Internal nodes are defined as a pin group in the template by using a function statement, and the ncx_internal_pin_signal_level_high and ncx_internal_pin_signal_level_low attributes can also be defined for them. Based on the function statement, an internal node is preset by using the HSPICE .nodeset feature.

- ncx_internal_pin_signal_level_high
ncx_internal_pin_signal_level_high : voltage

Specifies signal levels in volts for internal nodes in order to preset the internal nodes to a specific voltage before simulation occurs. The valid value is a floating point number that represents the voltage. If both ncx_internal_pin_signal_level_high and ncx_internal_pin_signal_level_low are not specified, then the default nominal voltage is used as ncx_internal_pin_signal_level_high.

- ncx_internal_pin_signal_level_low
ncx_internal_pin_signal_level_low : voltage

Specifies signal levels in volts for internal nodes in order to preset the internal nodes to a specific voltage before simulation occurs. The valid value is a floating point number that represents the voltage. If both ncx_internal_pin_signal_level_high and ncx_internal_pin_signal_level_low are not specified, then 0 (zero) is used as ncx_internal_pin_signal_level_low.

This example shows the use of the .nodeset command.

```
pin "INTNODE:3" {      //comment : Need quotation mark if internal node
contains a column ':'
    direction : internal ;
    function : !IE&IN ;
    ncx_internal_pin_signal_level_high : 3.3;
    ncx_internal_pin_signal_level_low   : 0; }

pin "NTCNIE/F736" {
```

```

direction : internal ;
function : !IE ;
ncx_internal_pin_signal_level_high : 3.3;
ncx_internal_pin_signal_level_low : 0; } \

pin NTCNIE {
    direction : internal ;
    function : !IE ;
    ncx_internal_pin_signal_level_high : 3.3;
    ncx_internal_pin_signal_level_low : 0; }

pin NTCNIN {
    direction : internal ;
    function : !IN ;
    ncx_internal_pin_signal_level_high : 3.3;
    ncx_internal_pin_signal_level_low : 0; }

pin TCNIE {
    direction : internal ;
    function : IE ;
    ncx_internal_pin_signal_level_high : 3.3;
    ncx_internal_pin_signal_level_low : 0; }

pin TCNIN {
    direction : internal ;
    function : IN ;
    ncx_internal_pin_signal_level_high : 3.3;
    ncx_internal_pin_signal_level_low : 0; }

```

In the SPICE deck the .nodeset command is added like it is in the following example:

```

* SensInputs= IN:      1
* SensInputs= IE:      01
* SensInputs= NRST:    0
* SensInputs= IO:      1
* SensInputs= CFO:     1
* SensInputs= CHDRV:   0
* SensOuputs= OUT:    01

* Set .NODESET for internal pin according to its function .NODESET
V(X_T11ZNRBN1C1F4ZSE.INTNODE:3) = 3.3 .NODESET
V(X_T11ZNRBN1C1F4ZSE.NTCNIE) =
3.3 .NODESET V(X_T11ZNRBN1C1F4ZSE.NTCNIE/F736) = 3.3 .NODESET
V(X_T11ZNRBN1C1F4ZSE.NTCNIN) = 0 .NODESET V(X_T11ZNRBN1C1F4ZSE.TCNIE) = 0
.NODESET V(X_T11ZNRBN1C1F4ZSE.TCNIN) = 3.3

```

IBIS Model Generation

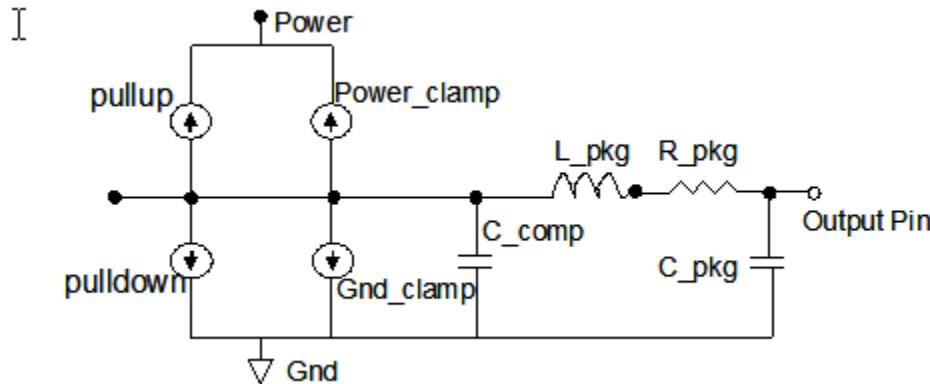
The Input/Output Buffer Information Specification (IBIS) provides a standardized methodology for representing the electrical characteristics of a digital device pin (input, output, or I/O pin) behaviorally. This allows for rapid simulation and the hiding of proprietary information.

An IBIS model is an ASCII file that contains all the data required to model a digital device behaviorally. The data is constructed from measurements made at each pin and contains

- I-V characteristics
- Switching characteristics (output voltage versus time)
- Device capacitance

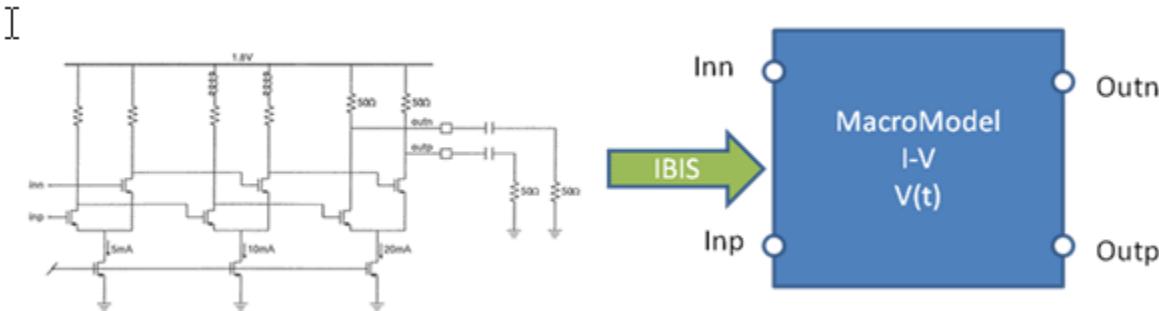
In an IBIS model, each pin of the device is replaced with a behavioral model like that in [Figure 4-5](#). In this case, it highlights the behavioral model for an output pin.

Figure 4-5 Macromodel for an Output Pin



You can obtain the properties of this behavioral model either by direct measurement or by circuit simulation. Different behavioral models exist for input pins, output pins, I/O pins, inverting pins, non-inverting pins, and so forth. [Figure 4-6 on page 4-15](#) shows an example of the IBIS conversion process.

Figure 4-6 IBIS Conversion Process



An IBIS model allows designers to simulate digital devices without any knowledge of the subblock's internal details. Likewise, IP vendors include IBIS models to reduce the need to release any proprietary information.

Liberty NCX can generate IBIS models for an arbitrary set of cells in its characterization flow.

You must specify the following settings in order to generate an IBIS model:

- Set the `ibis` variable to `true` in the configuration file.
- Set the `ibis_dir` variable to the location where the IBIS model files are to be created.
- Specify the IBIS operating conditions in the library template file.
- Specify a valid simulator.
- Ensure that valid SPICE model files exist.
- Ensure that valid SPICE netlists exist.
- Ensure that a library or template file exists that indicates which cells are to be characterized.
- Specify the IBIS version by using the `ibis_version` variable. Valid values are `3.2`, `4.0`, or greater.

Liberty NCX has the following capabilities for IBIS model generation:

- Detects the appropriate set of cells.
- Categorizes each pin in each cell in the relevant behavioral model.
- Uses HSPICE to perform the simulations for minimum, typical, and maximum operating conditions necessary to generate the data required for IBIS models.
- Gathers the relevant data required for generating the IBIS models.
- Uses the relevant data to create the IBIS files.

The `ibis` variable is described under the [IBIS Model Generation Flow](#) subsection. The following IBIS variables are described in [Table 1-5 on page 1-12](#).

```
ibis
ibis_dir
ibis_version
```

The following library template IBIS variables are described in [Table 3-2 on page 3-30](#).

```
ibis_condition_max
ibis_condition_min
ibis_condition_typ
temperature
pull_down_voltage
pull_up_voltage
ibis_model_file
ibis_netlist_dir
ibis_netlist_suffix
```

IBIS Model Generation Flow

The `ibis` variable runs the Input/Output Buffer Information Specification (IBIS) model generation flow in Liberty NCX.

```
set ibis (true|false)
```

The IBIS flow is enabled if this variable is set to `true`. The IBIS flow generates IBIS models for an arbitrary set of cells. You can specify the selection of cells by using either a seed library or templates (`do {}`).

An IBIS model is generated for each cell, and this model is placed in the location specified by the `ibis_dir` variable. The default behavior for Liberty NCX is to generate IBIS models only for PAD pins. The PAD pins should have the `is_pad` property set to `true`. In addition to specifying the `true` or `false` option, IBIS also requires that you specify three operating conditions (minimum, typical, maximum) in the library template file. You can specify the operating conditions by using the following `ncx` condition statements:

`ibis_condition_min` - Specify minimum condition parameters.

`ibis_condition_typ` - Specify typical condition parameters.

`ibis_condition_max` - Specify maximum condition parameters.

IBIS Operating Condition Options

The `ibis` variable operating conditions have the following options:

`temperature float` - Specify corner temperature.

`pull_down_voltage float` - Specify pull up voltage (more commonly referred to as power).

`pull_up_voltage float` - Specify pull down voltage (more commonly referred to as ground).

`ibis_model_file string` - Override the model file specified in the configuration file for this operating condition.

`ibis_netlist_dir string` - Override the `netlist_dir` specified in the configuration file for this operating condition.

`ibis_netlist_suffix string` - Override the `netlist_suffix` specified in the configuration file for this operating condition.

The `temperature`, `pull_down_voltage`, and `pull_up_voltage` options are required. All others are optional

The Liberty NCX IBIS model generation is a separate flow to characterization and cannot be run simultaneously with it.

Time Step and Range

IBIS specification versions 2.1 through 3.2 limit V-T and I-V data tables to 100 points. In IBIS version 4.0, the limit is 1000 points. By default, the simulations generate the full number of allowable time steps, either 100 or 1000, using a fixed step size of 0.1 in either case, and using a fixed voltage step size of 0.1 volts.

To explicitly specify the transient simulation duration and the DC simulation step size, use the following template file syntax:

```
ncx_ibis_transient_sim_duration : value ;  
ncx_ibis_dc_voltage_step : value ;
```

These template file attributes specify the duration of the transient simulation and the voltage step size of the DC simulation, respectively.

To explicitly specify the transient simulation time step, use the existing `tran_timestep` attribute:

```
tran_timestep : value ;
```

For example, if you specify the following:

```
tran_timestep : 0.05 ;
```

```
ncx_ibis_transient_sim_duration : 2.0 ;
ncx_ibis_dc_voltage_step : 0.20 ;
```

Liberty NCX generates the following statement for the transient simulation:

```
.TRAN 0.05 2.0 START = 0.0
```

and the following statement for the DC simulations:

```
.DC Vn_Pin 0 1.2 0.20
```

5

Library Conversion Support

Liberty NCX allows you to convert existing libraries from one format to another by using the Model Adaptation System, which converts or merges one or more existing library files in Liberty (.lib) format to create a new library that is written out in Liberty format. You can also convert Liberty files to datasheets and Verilog models for use in characterization.

This chapter includes the following sections:

- [Model Adaptation System](#)
- [Generating Datasheets](#)
- [Generating Verilog Models](#)

Model Adaptation System

The Model Adaptation System converts existing libraries from one format to another. The specific formatting operations are described in the following sections:

- [Library Formatting Overview](#)
- [CCS Model Compaction and Expansion](#)
- [Variation-Aware Library Merging](#)
- [CCS Noise Merging](#)
- [CCS to ECSM Conversion](#)
- [CCS Noise to ECSM Noise Conversion](#)
- [CCS to NLDM Conversion](#)
- [VA-CCS to S-ECSM Conversion](#)

Library Formatting Overview

Library Compiler can perform various types of library formatting operations. Each operation converts or merges one or more existing library files in Liberty (.lib) format to create a new library, which is written out in Liberty format. The set of formatting capabilities is called the Model Adaptation System.

To perform a formatting operation, you specify the type of operation, the input library names, the name of the new library file, and any options related to the operation in a configuration file. For example,

myfile contents:

```
set compact_ccs true
set input_library a90iz4.lib
set output_library a90iz4c.lib
```

Then, from the Library Compiler prompt,

```
lc_shell-xg-t> format_lib -f myfile
```

To view a list of the formatting options from Library Compiler, use the **-help** option:

```
lc_shell-xg-t> format_lib -help
```

If you need to perform multiple library formatting options, you must do so one at a time. You cannot perform multiple operations simultaneously.

The following library formatting operations are supported: CCS model compaction and expansion, variation-aware library merging, CCS Noise merging, CCS to ECSV conversion, CCS Noise to ECSV Noise conversion, CCS to NLDM conversion, and VA-CCS to S-ECMV conversion.

CCS Model Compaction and Expansion

The `compact_ccs` operation converts library cell models from expanded CCS format to compact CCS format. The compact format offers the same high accuracy as conventional expanded CCS data, but uses much less space in the library files. The theory of CCS data compaction and the Liberty syntax for CCS data are described in the section “Advanced Compact CCS Timing Model Support” in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*.

To convert conventional CCS data to compact CCS data, use the `compact_ccs` option, set the input library containing expanded CCS data, and set the new library to contain compact CCS data. For example,

```
set compact_ccs true
set input_library xlib82.lib
set output_library xlib82c.lib
```

The `expand_ccs` option performs the opposite function. It expands a library containing compact CCS modeling information into a library containing the original CCS modeling information. For example,

```
set expand_ccs true
set input_library xlib82c.lib
set output_library xlib82exp.lib
```

Variation-Aware Library Merging

You can use the Model Adaptation System to merge multiple CCS libraries characterized at different variation parameter values into a single variation-aware CCS library. A single library is easier to use for variation analysis than multiple libraries.

To combine multiple libraries, use the following configuration file syntax:

```
set va_merge true
set nominal_library {nom_lib_name par1 val1 par2 val2 ...}
set va_library_list {lib1_name val1 val2 ... \
                    lib2_name val1 val2 ... \
                    lib3_name val1 val2 ... \
                    lib4_name val1 val2 ... \
                    ...
                    lib2N_name val1 val2 ... }
```

```
set output_library out_lib_name
[set slew_indexes {index1 index2 ...}]
[set load_indexes {index1 index2 ...}]
```

For example, suppose that you have created a set of five libraries with variation in two parameters, `len` and `vt`. There is a single nominal library called `nom.lib`, libraries characterized at lower and higher `len` values called `lenlow.lib` and `lenhi.lib`, and libraries characterized at lower and higher `vt` values called `vtlow.lib` and `vthi.lib`. To merge these five libraries into a single variation-aware CCS library, you would use a configuration file similar to the following:

```
set va_merge true
set nominal_library { nom.lib      len 100.0  vt 0.24 }
set va_library_list { lenlow.lib    95.0     0.24 \
                      lenhi.lib    105.0    0.24 \
                      vtlow.lib   100.0    0.22 \
                      vthigh.lib  100.0    0.26 }
set output_library va_lvt.lib
```

You specify the nominal library using `nominal_library` and the off-nominal libraries using `va_library_list`. With the nominal library name, you specify all the parameter names and their respective nominal values. With each off-nominal library in the library list, you specify the full list of parameter values for that library in exactly the same order as for the nominal library.

The libraries in the library list can be specified in any order. However, for clarity, it is suggested that you use the order shown in the foregoing example: the low and high libraries for the first variable, followed by the low and high libraries for the second variable, and so on.

You can optionally specify a list of slew index values that are to be retained from the slew data tables in the off-nominal input libraries. For example, to retain only the first, second, third, and fifth slew index values, use `slew_indexes {1 2 3 5}`. This setting affects only the libraries specified with `library_list`. All index values in the nominal library are always retained.

Similarly, by using the `load_indexes` option, you can specify a list of load index values that are to be retained from the load data tables in the off-nominal input libraries.

When you run the formatting operation, the Model Adaptation System reads in the nominal and off-nominal variation libraries in Liberty format and combines them into a single variation-aware CCS library. It writes out the new library in Liberty format, using compact CCS models. You can then compile the new library using Library Compiler.

In PrimeTime VX, when you use a single merged variation-aware library instead of a set of libraries, the `set_variation_library` command is not required because the merged library already contains information about the variation relationships between the original libraries. You only need to link in the single variation-aware library and use the

`create_variation` and `set_variation` commands in the usual manner to specify the distribution of parameter values for timing analysis. For more information, see “Using a Single CCS-Based Variation-Aware Library” in the *PrimeTime VX User Guide*.

CCS Noise Merging

The current release of Liberty NCX does not support direct generation of CCS noise models. Instead, you can add CCS noise models to an existing library by using the Make CCS Noise utility. For more information, see the *Make CCS Noise User Guide*.

If you already have two different libraries containing CCS timing and CCS noise models, you can use the `ccsn_merge` operation to merge the two libraries into a single library containing both CCS timing and CCS noise models. The two input libraries must be in Liberty format and must contain the same cells.

To merge the two libraries, use `ccsn_merge` and specify the names of the CCS timing library file, the CCS noise library file, and the new merged library to be generated, as in the following example:

```
set ccsn_merge true
set timing_library x182c_tim.lib
set noise_library x182c_noise.lib
set output_library x182c.lib
```

When you run the formatting operation, the Model Adaptation System reads in the CCS timing and CCS noise libraries in Liberty format and combines them into a single new library in Liberty format. You can then compile the new library with Library Compiler.

CCS to ECSM Conversion

Effective Current Source Model (ECSM) is a library modeling standard developed at Cadence Design Systems, Inc. for representing cell behavior as current-source elements. You can convert a library containing CCS timing models to a new library containing ECSM models by using the `ccs2ecsm` option.

To specify whether to convert CCS timing data, CCS noise data, or both, set the `mode` option. The valid values for `mode` are `timing` and `noise`. If you specify `timing`, the CCS timing data will be converted but not the noise data. If you specify `noise`, the CCS noise data will be converted but not the timing data. You can convert both the timing and noise data by specifying the `timing` and `noise` values separated by a space, as shown:

```
set mode {timing noise}
```

When mode is set to timing, CCS timing data is converted to ECSM, and any CCS noise or power data is stripped out. Similarly, when mode is set to noise, CCS noise is converted to ECSM noise and any CCS timing or power data is stripped out from the output library.

If you do not specify the mode option, ccs2ecsm will convert the timing data only.

This is the general syntax for performing a CCS to ECSM conversion:

```
set ccs2ecsm true
set library_list { lib1 lib2 lib3 ... }
set output_library out_lib_name
set process value
set voltage value
set temperature value
set capacitance_mode first|second|ave|min|max]
set sample { v1 v2 v3 v4 ... }
```

To generate a new ECSM library, you can specify a single CCS library or multiple CCS libraries as input. If you specify a single library, the Model Adaptation System performs a straightforward conversion of the library. The operating conditions of the output library match those of the input library. For example,

```
set ccs2ecsm true
set library_list t52_nom.lib
set output_library t52ecsm_nom.lib
```

If you specify multiple libraries, the Model Adaptation System performs process, voltage, and temperature (PVT) scaling between the provided CCS libraries to obtain the cell behavior at a specified set of PVT conditions. Then it generates a single output library in ECSM format for that set of operating conditions.

For example, to generate a single NLDM library for voltage = 1.15 and temperature = 70 by scaling between four provided CCS libraries:

```
set ccs2ecsm true
set library_list { tv2c_p0v0t0.lib \
                  tv5c_p0v0t1.lib \
                  tv5c_p0v1t0.lib \
                  tv5c_p0v1t1.lib }
set output_library tv5c_ecsm.lib
set voltage 1.15
set temperature 70.0
```

When you run the formatting operation, the Model Adaptation System reads in the CCS library or libraries in Liberty format. If you use any of the process, temperature, or voltage options, you must provide multiple CCS libraries so that the Model Adaptation System can perform scaling between them to obtain models at the target operating conditions. The Model Adaptation System generates equivalent ECSM models and writes them out into the new library in Liberty format.

The CCS cell receiver model uses two capacitance values to more accurately model the Miller effect. The two values, called the first and second capacitance values, are used at the beginning and end of each logic transition, respectively. The ECSM standard, however, supports only a single receiver capacitance value. By default, the conversion process uses only the first CCS capacitance value and ignores the second value.

You can optionally specify a different conversion convention by using the `capacitance_mode` option. It can be set to `first`, `second`, `ave`, `min`, or `max`, causing the conversion process to use the first value, second value, the average of both values, the smaller of the two values, or the larger of the two values, respectively. For example,

```
set ccs2ecsm true
set library_list t52_nom.lib
set output_library t52ecsm_nom.lib
set capacitance_mode ave
```

This example causes all ECSM receiver models in the generated library to use the average of the corresponding first and second CCS capacitance values in the source library.

The conversion process generates ECSM data tables having discrete voltage sample points between the full voltage swing between 0.0 and the rail voltage. By default, the following voltage sample points are used to generate the data tables:

```
0.05 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 0.95
```

To specify a different set of voltage sample points, use the `sample` option. For example,

```
set ccs2ecsm true
set library_list t52_nom.lib
set output_library t52ecsm_nom.lib
set sample {0.02 0.05 0.10 0.20 0.30 0.40 0.50 \
            0.60 0.70 0.80 0.90 0.95 0.97}
```

CCS Noise to ECSM Noise Conversion

You can convert a library containing CCS noise models to a new library containing Effective Current Source Model (ECSM) noise models by using the `ccs2ecsm` option. To generate a new ECSM library, you must specify a single CCS noise input library only. If you specify multiple input libraries, `ccs2ecsm` issues a warning message and deletes the noise information.

Set the following variables in a configuration file to convert a library with CCS noise models to a library with ECSM noise models:

```
set ccs2ecsm true
set library_list library_name.lib
set output_library out.lib
```

```
set mode {timing noise}
set log_file out.log
```

As `mode` is set to `timing` and `noise`, it will convert both CCS timing and CCS noise models to ECSV.

To specify whether to convert CCS timing data, CCS noise data, or both, set the `mode` option in the configuration file. The valid values for `mode` are `timing` and `noise`. If you specify `timing`, the CCS timing data will be converted but not the noise data. If you specify `noise`, the CCS noise data will be converted but not the timing data. You can convert both the timing and noise data by specifying the `timing` and `noise` values separated by a space, as shown:

```
set mode {timing noise}
```

If you do not specify the `mode` option, `ccs2ecsv` will convert the timing data only.

To specify the voltage variation, set the following options in the configuration file:

- `set ecsv_vhtolerance value`

The value is an absolute floating-point number that specifies the variation in voltage that is considered to be negligible when the signal is high. The default value is 0.

- `set ecsv_vltolerance value`

The value is an absolute floating-point number that specifies the variation in voltage that is considered to be negligible when the signal is low. The default value is 0.

CCS to NLDM Conversion

Nonlinear Delay Model (NLDM) is an older, well-established modeling standard for representing cell behavior by using voltage-supply elements. You can convert a library containing CCS timing models to a new library containing NLDM models by using the `ccs2nldm` option.

To generate a new NLDM library, you can specify a single CCS library or multiple CCS libraries as input. If you specify a single library, the Model Adaptation System performs a straightforward conversion of the library. In such a case, the operating conditions of the output library match those of the input library. For example,

```
set ccs2nldm true
set library_list t52_nom.lib
set output_library t52nldm_nom.lib
```

If you specify multiple libraries, the Model Adaptation System performs process, voltage, and temperature (PVT) scaling between the provided CCS libraries to obtain the cell behavior at a given set of PVT conditions. Then it generates a single output library in NLDM format for that set of operating conditions.

For example, to generate a single NLDM library at process = 1.08, voltage = 1.15, and temperature = 70 by scaling between eight provided CCS libraries, use this syntax:

```
set ccs2nldm true
set library_list { tv2c_p0v0t0.lib \
                    tv5c_p0v0t1.lib \
                    tv5c_p0v1t0.lib \
                    tv5c_p0v1t1.lib }
set output_library tv5c_nldm.lib
set voltage 1.15
set temperature 70.0
```

Liberty NCX provides the following `nldm_cap` values to generate the input pin capacitance for a scaled NLDM library:

- `linear`

The `linear` value linearly interpolates the NLDM input capacitance found in the two corner libraries.

- `c1`

The `c1` value linearly scales the receiver model from the two corner libraries and uses the average of all `receiver_capacitance1` values in the scaled receiver model. The `c1` value is the default value for `nldm_cap`.

- `avg`

The `avg` value linearly scales the receiver model from the two corner libraries. The `avg` value then calculates the average of all `receiver_capacitance1` values and all `receiver_capacitance2` values in the scaled receiver model and uses the average of those values.

To use linear interpolation to calculate NLDM input pin capacitance for a scaled library, set the `nldm_cap` variable in the configuration file, as shown in the following example:

```
set nldm_cap linear
```

When you run the formatting operation, the Model Adaptation System reads in the CCS library or libraries in Liberty format. If you use any of the `process`, `temperature`, or `voltage` options, you must provide multiple CCS libraries so that the Model Adaptation System can perform scaling between them to obtain models at the target operating conditions. The Model Adaptation System generates equivalent NLDM models and writes them out into the new library in Liberty format.

VA-CCS to S-ECSM Conversion

You can use the Model Adaptation System to convert VA-CCS (variation-aware CCS) libraries to S-ECSM (Statistical Effective Current Source Model) libraries. ECSM is a library modeling standard developed by Cadence Design Systems, Inc. for representing cell behavior as current-source models. S-ECSM is an extension to the ECSM timing modeling format.

Liberty NCX can convert a VA-CCS library to S-ECSM library by using the `format_lib` command. Set the following options in the `format_lib` configuration file:

```
set vaccs2seccsm true
set input_library ncx.lib
set output_library out.lib
set s_ecsm_param_class_unit_list "param1 class1 unit1...paramn
                                  classn unitn"
set capacitance_mode [first|second|max|min|ave]
set sample "value"
```

The options are defined as follows:

`vaccs2seccsm`

Converts a variation-aware CCS library to an S-ECSM library. If `vaccs2seccsm` is set to `true`, `format_lib` generates an S-ECSM library from a variation-aware CCS library. The option is set to `false` by default.

`s_ecsm_param_class_unit_list`

Sets the S-ECSM parameters, class, and units. The valid values are `param1, class1`, and `unit1` through `paramn, classn`, and `unitn`. The values are user-defined. The `unitn` value can include a multiplier of 1, 10, or 100. Valid values for `classn` are global, local, random, and environmental.

Valid values for `unitn` are `nan, nm, um, A, mV, V, and C`. The `nan` value indicates no unit for the `paramn` parameter.

`capacitance_mode`

Specifies the conversion mode of the receiver capacitance. The following values are valid:

- `first`

Specifies that `ecsm_capacitance_sensitivity` is derived from the `va_receiver_capacitance1_rise` and `va_receiver_capacitance1_fall` values. The `first` value is the default for `capacitance_mode`.

- second

Specifies that `ecsm_capacitance_sensitivity` **is derived from the** `va_receiver_capacitance2_rise` **and** `va_receiver_capacitance2_fall` **values.**

- max

Specifies that `ecsm_capacitance_sensitivity` **is the maximum value between** `va_receiver_capacitance1_rise` **and** `va_receiver_capacitance2_rise` **or** `va_receiver_capacitance1_fall` **and** `va_receiver_capacitance2_fall`.

- min

Specifies that `ecsm_capacitance_sensitivity` **is the minimum value between** `va_receiver_capacitance1_rise` **and** `va_receiver_capacitance2_rise` **or** `va_receiver_capacitance1_fall` **and** `va_receiver_capacitance2_fall`.

- ave

Specifies that `ecsm_capacitance_sensitivity` **is the average value between** `va_receiver_capacitance1_rise` **and** `va_receiver_capacitance2_rise` **or** `va_receiver_capacitance1_fall` **and** `va_receiver_capacitance2_fall`.

sample

Specifies the `ecsm_waveform` **and** `ecsm_waveform_sensitivity` **sample points with a** space-separated value. The sample values are normalized and must be in the range of 0 to 1. The default is “0.05 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 0.95”.

When you run the formatting operation, the Model Adaptation System reads in the variation-aware CCS library in Liberty format and generates equivalent S-ECSM models and writes them out into the new library in Liberty format.

Generating Datasheets

Characterization tools support datasheets, which provide detailed information about a cell, in addition to Liberty models. Generating datasheets from Liberty .lib files is helpful if you are new to characterization or if you are creating new libraries rather than recharacterizing existing libraries with new process models. This section describes how to generate datasheets from Liberty .lib files for use in characterization. It includes the following sections:

- [Datasheet Generation Overview](#)
- [Datasheets in HTML Format](#)
- [Datasheets in Text Format](#)
- [HTML and Text Datasheet Details](#)

Datasheet Generation Overview

The characterization flow requires that you first run Liberty NCX and generate a .lib file. Next, you generate a datasheet from the .lib file using Library Compiler. This ensures that the datasheet matches the .lib file exactly. Although datasheet generation is performed in Library Compiler, it requires a Liberty NCX license and will fail if it is not available.

You can generate a datasheet in HTML format or text format. When you generate a datasheet, Library Compiler generates a datasheet from a .lib file and creates a *cell.html* file for each cell in the library. You must set `datasheet_enable` in order to generate a datasheet, and depending on what this is set to, it generates text or HTML.

`datasheet_enable: text` generates a text datasheet.

`datasheet_enable: html` generates an HTML datasheet.

`datasheet_enable: true` generates a text datasheet for backward compatibility.

Use the following commands to generate datasheets:

1. To enable datasheet generation, set the `datasheet_enable` variable to `text` or `html`. By default, the variable is set to `html`, and a datasheet is generated in HTML format.

```
set datasheet_enable [text | html]
```

2. To create a datasheet from a .lib file, run the `write_lib` command in `lc_shell` and specify the .lib library name as shown:

```
write_lib -format datasheet library_name
```

3. By default, datasheets are written to *library_name_datasheet* for a text datasheet, and *library_name_datasheethtml* for an HTML datasheet. If you want the datasheet to be written to a directory other than the default, use the one of the following commands:

For a text datasheet,

```
set datasheet_output_dir directory_name
```

For an HTML datasheet,

```
set datasheet_html_output_dir directory_name
```

Table 5-1 provides an overview of the commands that are available to specify datasheet sections, including the file format and default settings. Each row in the table, except the first three rows, represents a section in the datasheet. You can set each command directly in `lc_shell` or include them in a configuration file and source the file by running the `source config_file_name` command in `lc_shell`.

To generate datasheets in HTML format, you must provide an HTML datasheet template. A default HTML template is available if you do not provide a custom template.

Table 5-1 Commands Used to Specify Datasheet Details

Command or section	Text format	HTML format
Enabling datasheet generation	set datasheet_enable text (Enabled by default)	set datasheet_enable html
Output directory location	Default output directory <code>library_name_datasheet</code> . To specify a directory, use set datasheet_output_dir <code>directory_name</code>	Default output directory: <code>library_name_datasheet_html</code> To specify a directory, use set datasheet_html_output_dir <code>directory_name</code>
HTML template location	n/a	Default template location: <code>\$synopsys_root/auxx/template/datasheet_template.html</code> To override the default template, use set datasheet_html_template <i>file_name</i>
Logo	n/a	To include a logo, use set datasheet_html_logo_file <i>file_name</i> (No default setting)
Cell name	Derived from .lib file	Derived from .lib file
Cell description file	n/a	By default, HTML datasheet generation leaves Cell Description empty, unless it finds the following directory: <code>library_name_cdesc</code> To override this, use set datasheet_html_cell_desc_dir <code>directory_name</code> If this attribute described is defined, by default, it looks for the .cdesc file extension. To override the default suffix, use set datasheet_html_cell_desc_suffix <i>string</i>

Table 5-1 Commands Used to Specify Datasheet Details (Continued)

Command or section	Text format	HTML format
Function description file	Derived from .lib file	<p>Default behavior: Derived from .lib file</p> <p>To specify a directory, use <code>set datasheet_html_func_desc_dir directory_name</code></p> <p>Default file suffix: .fdesc</p> <p>To specify a file suffix, use <code>set datasheet_html_func_desc_suffix string</code></p>
Symbol	n/a	<p>Default file location: library_name_sym directory in the current working directory</p> <p>To specify a directory, use <code>set datasheet_html_symbol_dir directory_name</code></p> <p>Default file suffix: .sym</p> <p>To specify a file suffix, use <code>set datasheet_html_symbol_suffix string</code></p>
Schematic	n/a	<p>Default file location: library_name_sch directory in the current working directory</p> <p>To specify a directory, use <code>set datasheet_html_schematic_dir directory_name</code></p> <p>Default file suffix: .sch</p> <p>To specify a file suffix, use <code>set datasheet_html_schematic_suffix string</code></p>
Port names	Derived from .lib file	Derived from .lib file
Cell area	Derived from .lib file	Derived from .lib file
Pin capacitance	Derived from .lib file	Derived from .lib file
Delay data	Derived from .lib file	Derived from .lib file

Table 5-1 Commands Used to Specify Datasheet Details (Continued)

Command or section	Text format	HTML format
Power data	n/a	Derived from .lib file

Datasheets in HTML Format

Library Compiler creates *cell.html* files in the user-defined *library_name_datasheet_html* directory under the current working directory. If the *library_name_datasheet_html* directory already exists under the current working directory, the newly generated *cell.html* files overwrite the existing files in the directory, and Library Compiler issues a warning message.

Each *cell_name.html* datasheet provides the following information for a cell:

- Logo: *logo_file*
See “[Logo \(HTML\)](#)” on page 5-17.
- Cell: *cell_name*
See “[Cell \(HTML and Text\)](#)” on page 5-17.
- Cell Description File: *directory_name*
See “[Cell Description File \(HTML only\)](#)” on page 5-17.
- Function Description
See “[Function Description \(HTML and Text\)](#)” on page 5-18.
- Symbol: *symbol_file*
See “[Symbol \(HTML Only\)](#)” on page 5-23.
- Schematic: *schematic_file*
See “[Schematic \(HTML Only\)](#)” on page 5-23.
- Port Names
See “[Port Names \(HTML and Text\)](#)” on page 5-24.
- Cell Area: *cell area*
See “[Cell Area \(HTML and Text\)](#)” on page 5-25.
- Pin Capacitance
See “[Pin Capacitance \(HTML and Text\)](#)” on page 5-25.

- Delay Data
See “[Delay Data \(HTML and Text\)](#)” on page 5-27.
 - Constraint Data
See “[Constraint Data \(HTML and Text\)](#)” on page 5-29.
 - Power Data
See “[Power Data \(HTML Only\)](#)” on page 5-31.
-

Datasheets in Text Format

Library Compiler creates *cell.txt* files in the user-defined library_name_datasheet directory under the current working directory. If the library_name_datasheet directory already exists under the current working directory, the newly generated *cell.txt* files overwrite the existing files in the directory, and Library Compiler issues a warning message.

Each *cell_name.txt* datasheet provides the following information for a cell:

- Cell: *cell_name*
See “[Cell \(HTML and Text\)](#)” on page 5-17.
- Function Description
See “[Function Description \(HTML and Text\)](#)” on page 5-18.
- Port Names
See “[Port Names \(HTML and Text\)](#)” on page 5-24.
- Cell Area: *cell area*
See “[Cell Area \(HTML and Text\)](#)” on page 5-25.
- Pin Capacitance
See “[Pin Capacitance \(HTML and Text\)](#)” on page 5-25.
- Delay Data
See “[Delay Data \(HTML and Text\)](#)” on page 5-27.
- Constraint Data
See “[Constraint Data \(HTML and Text\)](#)” on page 5-29.

HTML and Text Datasheet Details

The valid arguments to use for the *cell.html* and *cell_name.txt* files are described in the following subsections.

Logo (HTML)

If you want a logo included in the datasheet, you must specify it in a logo file. There is no default setting. To include a logo, use the following variable:

```
set datasheet_html_logo_file file_name
```

There is no default setting.

Syntax

Logo: *logo_file*

Example

```
set datasheet_html_logo_file logo_URL or path to file
```

This option is available only for HTML datasheets.

Cell (HTML and Text)

The datasheet generator takes the current library cell name from the input Liberty (.lib) file and writes it to the first line of the datasheet file, as shown:

Syntax

Cell: *cell_name*

Example

Cell: AND2X2

This option is available for both HTML and text datasheets.

Cell Description File (HTML only)

An optional cell description section can be added to the HTML Datasheet. If nothing is specified, this section is says “None”. However, you can provide information for the cell description for each cell. The default file location is the *library_name_cdesc_dir* directory in the current working directory.

To specify a directory, use:

```
set datasheet_html_cell_desc_dir directory_name
```

The default file suffix is .cdesc. To specify a different file suffix, use:
set datasheet_html_cell_desc_suffix string.

The datasheet generator relies on user input in the configuration file to indicate the location of the cell description file. You can locate the cell description file for the current cell in either of the following two ways:

- If there is a map.\$datasheet_html_cell_desc_suffix under the \$datasheet_html_cell_desc_dir directory, then the datasheet generator checks the cell's name in the mapping file, include wildcard. If a mapped file to this cell is found, the datasheet generator uses this file as the cell description file for this cell.
- If no \$datasheet_html_cell_desc_dir or map.\$datasheet_html_cell_desc_suffix exists, or the datasheet generator does not find a mapped cell description file to this cell, then it searches the cell_name.\$datasheet_html_cell_desc_suffix under the \$datasheet_html_cell_desc_dir directory, using it as the cell description file for this cell.

If the datasheet generator cannot find the cell description file by using one of these methods, it replaces this section with None.

Syntax

```
None [This is the default if nothing is specified]
```

Example

```
Cell Description:  
Inverter
```

This option is available only for HTML datasheets.

Function Description (HTML and Text)

If the function description of the cell is included in the input Liberty file, it is written out in the Function Description section of the datasheet. For black box cells that have no function description in the .lib file, the datasheet generator issues a warning and adds a comment in the Function Description section of the datasheet.

Function Description for HTML Datasheets

By default the function description from the input .lib is used. However, for HTML datasheets, you can override this by specifying the location and file names of the custom function description.

For a function description file, the default file location is the library_name_fdesc directory in the current working directory. To specify a directory, use

```
set datasheet_html_func_desc_dir directory_name
```

The default file suffix is .fdesc. To specify a different file suffix, use

```
set datasheet_html_func_desc_suffix string
```

You can locate the function description file for the current cell in either of the following two ways:

- If there is a map.\$datasheet_html_func_desc_suffix under the \$datasheet_html_func_desc_dir directory, then the datasheet generator checks the cell's name in the mapping file `include` wildcard. If a mapped file to this cell is found, the datasheet generator uses this file as the function description file for this cell.
- If no \$datasheet_html_function_desc_dir or map.\$datasheet_html_func_desc_suffix exists, or the datasheet generator does not find a mapped function description file to this cell, then it searches the `cell_name.$datasheet_html_func_desc_suffix` under the \$datasheet_html_func_desc_dir directory, using it as the function description file for this cell.

If the datasheet generator cannot find the function description file by using one of these methods, it uses the information from the function statement in the input .lib.

Example

```
set datasheet_html_func_desc_dir ./function_description_dir  
set datasheet_html_func_desc_suffix .txt
```

This option is available for both HTML and text datasheets.

Default Function Description Arguments

The following list shows the function description types.

- Combinational function

The combinational function format provides the Boolean function for a gate and has the following syntax:

```
Function: output_pin = Boolean_expression  
          output_pin = Boolean_expression  
          ...
```

The arguments are as follows:

output_pin

The output pin is the corresponding output or bidirectional pin name found in the input Liberty file.

Boolean_expression

The Boolean expression is retrieved directly from the `function` attribute in the input Liberty file.

The following examples shows the .lib syntax followed by the datasheet equivalent:

.lib Example

```
pin (Y)
function: A ^ B' & C;
}
```

Datasheet Example

Function: $Y = A \wedge B' \wedge C;$

- Three-State Function

The three-state function format is similar to the combinational function. Three-state functions are generated in the datasheet when a bidirectional or output pin has a `three_state` attribute specified in input Liberty file. The three-state function has the following syntax:

Function: *output_pin* = *input_pin*
 Three state: *output_pin* = *Boolean_expression*

output_pin

The output pin is the name of the output or bidirectional pin with a three-state function in the input Liberty file.

Boolean_expression

The Boolean expression is retrieved directly from the corresponding `three_state_function` attribute value in the input Liberty file.

The following examples shows the .lib syntax followed by the datasheet equivalent:

.lib Example

```
pin (O) {
    direction : "output";
    function : "I";
    three_state : "EN!";
    ...
}
```

Datasheet Example

Function: $O = (I)$

Three state: $O = (EN')$

- Sequential Function

The sequential function format is a state table and has the following syntax:

statetable:
state_table_pin_list
table_content

```

statetable:
  state_table_pin_list
  table_content

statetable:
  output_pin = Boolean_expression
  ...

state_table_pin_list

```

The state table pin list is a comma-separated list and contains input pins followed by one output pin with a current state and a next state. It has the following characteristics:

- The output pin name is the real output pin name.
- If the current state pin name is *output*, the next state pin name is *output+*. For example, if the current state is *Q*, the next state is *Q+*.
- One state table has exactly one output pin.
- The order of the input pin list determines the order of the pins in the state table.
- Generally, there is a state table for each output of the cell; however, in some cases, a state table might not exist. For example, *QN* might be a logic function of *Q* (for instance, inverted). The datasheet generator writes a Boolean expression for *QN*, using *Q*. The state table has the following syntax:

```

  input_state [input_state...] current_state next_state
  ...
  input_state

```

The input state list is a space character (“ ”) separated list of input state values. Valid characters for the input values are 0, 1, r, f, b, ?, x, and *.

current_state

The current state is the output current state value, and the valid characters are 0, 1, x, ?, and b.

next_state

The next state is the next state value, and the valid characters are 0, 1, x, and -.

These state table values have the following definitions:

- 0 -A low state.
- 1 -A high state.
- x -unknown.

- ? -an iteration of 0, 1 and x.
- b -an iteration of 0 and 1.
- r -The same as 01.
- f -The same as 10.
- * -The same as ??.
- - -No change.

Each row defines the output, based on the current state and the particular combinations of input values.

The following examples show a latch function with Q and QN outputs and a flip-flop with asynchronous inputs and with Q and QN outputs.

Latch Function Example

```
statetable:
  CDN  D   E   SDN  Q   Q+
  0    ?   ?   1    ?   0
  1    1   1   ?    ?   1
  1    ?   0   1    ?   -
  ?    0   1   1    ?   0
  ?    ?   ?   0    ?   1

  statetable:
  SDN  D   E   CDN  QN  QN+
  0    ?   ?   1    ?   0
  1    0   1   ?    ?   1
  1    ?   0   1    ?   -
  ?    1   1   1    ?   0
  ?    ?   ?   0    ?   1
```

Flip-Flop Example

```
statetable:
  CDN  CP   D   Q   Q+
  0    ?   ?   ?   0
  1    (01) 1   ?   1
  1    (1?)  ?   ?   -
  1    (?0)  ?   ?   -
  ?    (01) 0   ?   0

  statetable:
  QN = (not Q)
```

Symbol (HTML Only)

For a symbol file, the default file location is the library_name_sym directory in the current working directory.

To specify a directory, use

```
set datasheet_html_symbol_dir directory_name
```

Default file suffix: .sym

To specify a file suffix, use

```
set datasheet_html_symbol_suffix string
```

The datasheet generator relies on user input in the configuration file to indicate the location of the symbol file. You can locate the symbol file for the current cell in either of the following two ways:

- If there is a map.\$datasheet_html_symbol_suffix file under the \$datasheet_html_symbol_dir directory, then the datasheet generator checks the cell's name using the mapping-file include wildcard. If a mapped file to this cell is found, the datasheet generator uses this file as the symbol file for this cell.
- If no map.\$datasheet_html_symbol_suffix exists, or the datasheet generator does not find a mapped symbol file to this cell, then it searches for the cell_name.\$datasheet_html_symbol_suffix file under the \$datasheet_html_symbol_dir directory as the symbol file for this cell.

If it cannot find the symbol file by using one of these methods, the datasheet generator replaces {symbol_file} with None.

The format of the symbol file should be JPEG, BMP, or PNG.

The datasheet generator does not display a symbol picture unless you specify that it do so.

Syntax

Symbol: *symbol_file*

Example

```
set datasheet_html_symbol_suffix symbol_URL
```

This option is available only for HTML datasheets.

Schematic (HTML Only)

For a schematic file, the default file location is the library_name_sch directory in the current working directory.

To specify a directory, use

```
set datasheet_html_schematic_dir directory_name
```

Default file suffix: .sch

To specify a file suffix, use

```
set datasheet_html_schematic_suffix string
```

The datasheet generator relies on user input in the configuration file to indicate the location of the schematic file. You can locate the schematic file for the current cell in either of the following two ways:

- If there is a map.\$datasheet_html_schematic_suffix file under the \$datasheet_html_schematic_dir directory, then the datasheet generator checks the cell's name using the mapping-file include wildcard. If a mapped file to this cell is found, the datasheet generator uses this file as the schematic file for this cell.
- If no \$map.\$datasheet_html_schematic_suffix exists, or the datasheet generator does not find a mapped symbol file to this cell, then it searches for the cell_name.\$datasheet_html_schematic_suffix under the \$datasheet_html_schematic_dir directory as the symbol file for this cell.

If it cannot find the schematic file by using one of these methods, the datasheet generator replaces {schematic_file} with None.

The format of the schematic file should be JPEG, BMP, or PNG.

The datasheet generator does not display a schematic picture unless you specify that it do so.

Syntax

```
schematic: schematic_file
```

Example

```
set datasheet_html_schematic_suffix schematic_URL
```

This option is available only for HTML datasheets.

Port Names (HTML and Text)

Syntax

```
Inputs: pin_list
Outputs : pin_list
InOuts: pin_list
```

pin_list

The pin list is a comma-separated list. It takes its name from the current library pin name in the input Liberty file.

The rules regarding pin placement are determined by the `direction` attribute in the current pin group, as follows:

- If the value is `input`, the pin is written to the Inputs list.
- If the value is `output`, the pin is written to the Outputs list.
- If the value is `inout`, the pin is written to the Inouts list.

This option is available for both HTML and text datasheets.

Cell Area (HTML and Text)

Syntax

```
Cell area : cell_area
```

cell_area

The cell area is a floating-point number with six digits after the decimal point. It is retrieved from the `area` attribute at the cell-level from the input Liberty file. No units are explicitly given for the value, but the datasheet generator uses the same unit for the area of all cells in a library.

.lib Example

```
cell (ACHCINX2) {  
area: 14.818000;  
}
```

Datasheet Example

```
Cell area: 14.818000
```

This option is available for both HTML and text datasheets.

Pin Capacitance (HTML and Text)

The datasheet generator returns pin capacitance data in two tables, one for input pins and one for output pins.

The input pin table includes input and bidirectional pins. The input pin capacitance values are derived from the `capacitance` attribute in the .lib file. If the .lib file does not include a `capacitance` attribute, the value is derived from the larger of the `fall_capacitance` and `rise_capacitance` attribute values. If there are no `rise_capacitance` and `fall_capacitance` attributes in the .lib file, no capacitance value is written for that pin.

The output pin table includes output and bidirectional pins. The output pin capacitance values are derived from the `max_capacitance` attribute in the .lib file.

Syntax

Inputs:

Pin	Cap. [<i>capacitance_unit</i>]
<i>pin_name</i>	<i>capacitance_value</i>
<i>pin name_</i>	<i>capacitance _alue</i>

Outputs:

Pin	Max. Cap. [<i>capacitance_unit</i>]
<i>pin_name</i>	<i>maximum capacitance_value</i>
<i>pin_name</i>	<i>maximum capacitance_value</i>

pin_name

The pin name in the input Liberty file.

capacitance_value

A floating-point number with six digits after the decimal point.

capacitance_unit

The capacitance unit for the capacitance value and the maximum capacitance value, which is derived from the `capacitive_load_unit` attribute in the input Liberty file.

.lib Example (For an Input File)

```
capacitive_load_unit(1.000000, "pf");
cell (sample) {
    pin (A) {
        capacitance: 2.258850;
    }
    pin (B) {
        capacitance: 5.234676;
    }
    pin (CIN) {
        rise_capacitance: 2.225339;
        fall_capacitance: 2.314452;
    }
}
```

```
pin (CO) {
    max_capacitance: 109.509995;
}
}
```

Datasheet Example

Pin Capacitance

Inputs:

Pin	Cap. (pF)
A	2.258850
B	5.234676
CIN	2.314452

Outputs:

Pin	Max. Cap. (pF)
CO	109.509995

This option is available for both HTML and text datasheets.

Delay Data (HTML and Text)

Syntax

Delay Path	Delay [<i>time_unit</i>]
<i>input_pin input_transition => output_pin output_transition</i>	<i>delay_value</i>

The arguments are as follows:

input_pin

The input pin, or related pin, for a timing arc.

output_pin

The output pin for a timing arc.

input_transition

Shows the state transition of the input.

output_transition

Shows the state transition of the output.

time_unit

This value is derived from the `time_unit` attribute in the input Liberty file.

delay_value

A floating-point number with six digits after the decimal points.

This option is available for both HTML and text datasheets.

Input Transition

The input transition characteristics are as follows:

- For `non_unate` delay arcs, no input transition is written out.
- For `positive_unate` and `negative_unate` delays arcs, the input transition is 01 and 10.

Input transition is derived from the `output transition`, `timing_type`, and `timing_sense`.

When `timing_type` is `combinational`, `combinational_fall`, `combinational_rise`, `preset`, or `clear`, there can be up to two input transitions, as follows:

- When `timing_sense` is `positive_unate`, the input transition is consistent with the output transition.
- When `timing_sense` is `negative_unate`, the input transition is opposite to the output transition.

When `timing_type` is `three_state_disable`, `three_state_enable`, `three_state_disable_rise`, `three_state_disable_fall`, `three_state_enable_rise`, or `three_state_enable_fall`, the input transition is as follows:

- When `timing_sense` is `positive_unate`, the input transition is 01.
- When `timing_sense` is `negative_unate`, the input transition is 10.

When `timing_type` is `rising_edge`, the input transition is 01, and when `timing_type` is `falling_edge`, the input transition is 10.

Output Transition

The output transition characteristics are as follows:

- The valid values are: 01, 10, 0Z, Z1, 1Z or Z0
where 0 is low state, 1 is high state, and Z is high-impedance state. For arcs that are not three-state arcs, if the group type is `cell_rise`, the `output_transition` is 01, and if the group type is `cell_fall`, the `output_transition` is 10.

For three-state arcs, the output transition is derived from `timing_type`, as follows:

- When `timing_type` is `three_state_disable`, the output transition is `0Z/1Z`.
- When `timing_type` is `three_state_enable`, the output transition is `Z0/Z1`.
- When `timing_type` is `three_state_enable_rise`, the output transition is `Z1`.
- When `timing_type` is `three_state_enable_fall`, the output transition is `Z0`.
- When `timing_type` is `three_state_disable_rise`, the output transition is `0Z`.
- When `timing_type` is `three_state_disable_fall`, the output transition is `1Z`.

The datasheet generator chooses the last value in the `cell_rise` or `cell_fall` table. This corresponds to the worst delay, as the delay value is monotonically increasing in the `cell_rise` or `cell_fall` table.

If there are duplicate timing arcs and if timing groups in the input Liberty file have multiple path delays with the same transition for the same input to output pin, they are considered duplicate path delays. The datasheet generator avoids generating duplicate path delays. If the datasheet generator finds a potential duplicate path delay, it uses the timing arc with the worst delay value.

Example

The following example shows a delay table for a two-input NAND cell:

Delay Path	Delays [ps]
A 10 => Y 10	123.432424
A 01 => Y 01	221.425345
B 10 => Y 10	121.735747
B 01 => Y 01	121.758758

This option is available for both HTML and text datasheets.

Constraint Data (HTML and Text)

The datasheet contains the following information about constraint data.

Syntax

Check	Constraint [<i>time unit</i>]
<i>constraint_pin constraint_pin_transition check_type related_pin related_pin_transition</i>	<i>constraint_value</i>

The arguments are as follows:

constraint_pin

The parent pin of a timing constraint in an input Liberty file.

constraint_pin_transition

The value is derived from the `rise_constraint` or `fall_constraint` group type in the input Liberty file.

check_type

The timing check type. Currently, only setup, hold, recovery, removal, and skew are supported.

related_pin

The pin with the `related_pin` attribute in the timing constraint.

related_pin_transition

The value is derived from the `timing_type` attribute in the input Liberty file. If `timing_type` is `setup_rising`, `hold_rising`, `recovery_rising`, `removal_rising`, or `skew_rising`, the related pin transition is 01.

time_unit

The value is derived from the `time_unit` attribute in the input Liberty file.

constraint_value

The value is a floating-point number with six digits after the decimal point.

The related pin transition characteristics are as follows:

- If `timing_type` is `setup_rising`, `hold_rising`, `recovery_rising`, `removal_rising`, or `skew_rising`, the related pin transition is 01.
- If `timing_type` is `setup_falling`, `hold_falling`, `recovery_falling`, `removal_falling`, or `skew_falling`, the related pin transition is 10.

The constraint pin transition characteristics are as follows:

- The constraint pin transition is `rise` when the group type is `rise_constraint`.
- The constraint pin transition is `fall` when the group type is `fall_constraint`.

The datasheet generator chooses the last value in the `rise_constraint` or `fall_constraint` table for the constraint value.

If timing groups in Liberty NCX have multiple timing checks with the same transition for the same input and output and the same type, they are considered duplicate timing checks. The duplicate timing checks are as follows:

- The datasheet generator avoids generating duplicate timing checks.
- If a potential duplicate timing check is found, the datasheet generator uses the timing constraint with the worst constraint value and ignores the rest.

Example

The following example shows the constraint data:

Check	Constraints [ps]
D 10 setup CK 01	123.432523
D 10 hold CK 01	221.432523
D 01 setup CK 01	121.432523
D 01 hold CK 01	121.432523

Power Data (HTML Only)

Input Power Data

Input power data shows the internal power corresponding to each input pin. This information is derived from the `.lib`.

Input Syntax

Power Path	Power [<i>power_unit</i>]
<i>input_power_path</i>	<i>input_power</i>

Output Power Data

Output power data provides information about the internal power at the output pin corresponding to a path from each input pin in the cell. This information is derived from the `.lib`.

Output Syntax

Power Path [power_unit]	Load Capacitance [cap_unit]					
	load	load	load	load	load	load
output_power_path	{output_power}	{output_power}	{output_power}	{output_power}	{output_power}	{output_power}

Input and Output Syntax Arguments

input_power_path

Provides information on the input pin and whether it is rise or fall `internal_power`. For example, `I1 01` Indicates a rise `internal_power` for input pin `I1`. The `input_power_path` is a floating-point number with six digits after the decimal point.

input_power

Is the last value in the `internal_power` table corresponding to this path and type. The `input_power` is a floating-point number with six digits after the decimal point.

power_unit

A floating-point number with six digits after the decimal point.

cap_unit

A floating-point number with six digits after the decimal point.

load

A floating-point number with six digits after the decimal point.

output_power_path

Shows which path goes from the input pin to the output pin and whether it is rise or fall. The `load` is the list of output load indices and is derived from the `.lib`. The `output_power_path` is a floating-point number with six digits after the decimal points. The delay value is extracted from NLDM tables only in Liberty NCX version A-2007.12.

output_power

Is the last row of the table of the internal power values corresponding to the `output_power_path`. The `output_power` is a floating-point number with six digits after the decimal points. The delay value is extracted from NLDM tables only in Liberty NCX version A-2007.12.

This option is available only for HTML datasheets.

Generating Verilog Models

In addition to Liberty models, characterization tools support Verilog models, which provide detailed information about a cell. Generating Verilog models from Liberty .lib files is helpful if you are new to characterization or if you are creating new libraries rather than recharacterizing existing libraries with new process models. This section describes how to generate Verilog models in .v file format from Liberty .lib files for use in characterization. It includes the following sections:

- [Overview](#)
 - [Generating Verilog Files](#)
 - [Creating UDPs](#)
 - [Verilog Model Details](#)
-

Overview

The characterization flow requires that you first run Liberty NCX and generate a .lib file. Next, you generate a Verilog file from the .lib file using Library Compiler. This ensures that the Verilog file matches the .lib file exactly. Although you generate the Verilog file in Library Compiler, it requires a Liberty NCX license and will fail if a Liberty NCX license is not available.

Generating Verilog Files

Library Compiler generates a Verilog model from a .lib file and creates a *cell.v* Verilog file for each cell in the library. By default, Library Compiler creates the *cell.v* files in the user-defined *library_name_verilog* directory, under the current working directory. If the *library_name_verilog* directory already exists under the current working directory, the newly generated *cell.v* files overwrite the existing files in the directory, and Library Compiler issues a warning message.

Use the following commands to generate Verilog models:

1. To enable Verilog model generation, set the `veriloglib_enable` variable to `true`. By default, the variable is set to `false`, and a Verilog file is not generated.

```
set veriloglib_enable [true | false]
```

2. To create a Verilog file from a .lib file, run the `write_lib` command in `lc_shell` and specify the .lib library name as shown:

```
write_lib -format verilog library_name
```

3. If you want the Verilog model to be written to a directory other than *library_name_verilog*, set the `veriloglib_output_dir` variable, as shown:

```
set veriloglib_output_dir directory_name
```

Creating UDPs

The Verilog model generator can create two types of User-Defined Primitives (UDPs): independent UDPs and custom UDPs. They are described in the following subsections:

- [Creating Independent UDPs](#)
- [Creating Custom UDPs](#)

Creating Independent UDPs

The Verilog model generator generates a Verilog model with independent user-defined primitives (UDPs) and allows multiple cells to share the same user-defined primitive.

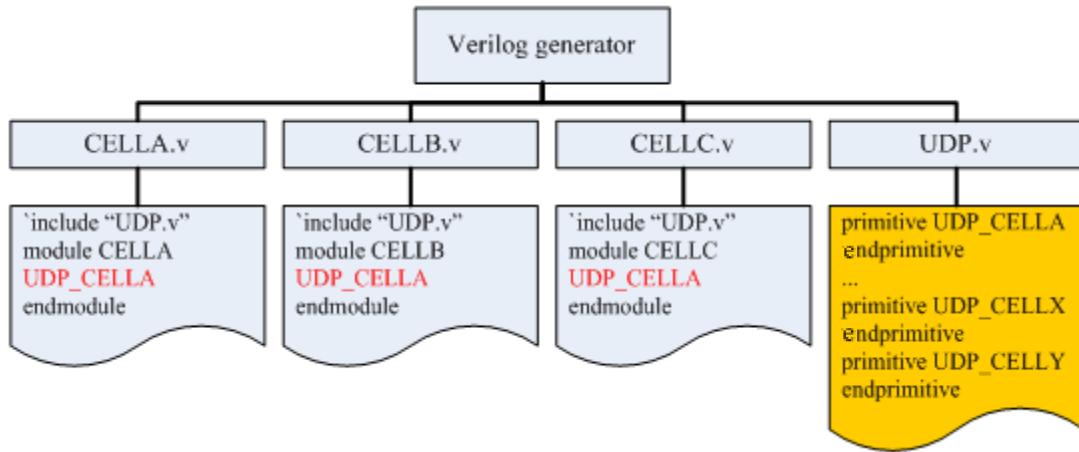
The Verilog model generator generates user-defined primitives and modules in separate Verilog files. The Verilog model generator determines whether the cell function model is the same as the one that was previously output; if the cell function model is the same, the Verilog model generator includes that user-defined primitive file and does not generate a new one. This is called the independent user-defined primitive feature of the Verilog model generator.

To use the independent user-defined primitive flow in the Verilog model generator, set the `veriloglib_independent_udp` variable to `true`. The default is `false`.

In an independent UDP flow, the Verilog file contains only the module part for each cell. The UDP parts are written in the UDP.v file, which is shared by all cells. The Verilog model generator identifies whether or not the cell's function is the same as the one that has already been generated. It uses the existing cell's UDP in the UDP.v file if they are identical; otherwise, it generates a new UDP for this cell and writes it into the UDP.v file.

This flow is shown by the example in [Figure 5-1 on page 5-35](#). Assume that the output cell order is CELLA, CELLB, CELLC, and they have exactly the same function model. The Verilog model generator determines that the function of CELLB and CELLC are the same as that of CELLA, and CELLA's UDP has already been generated in the UDP.v file. CELLB and CELLC simply call CELLA's UDP, so no duplicate UDP has to be generated.

Figure 5-1 Independent Verilog UDP Flow



Flow for Independent UDP

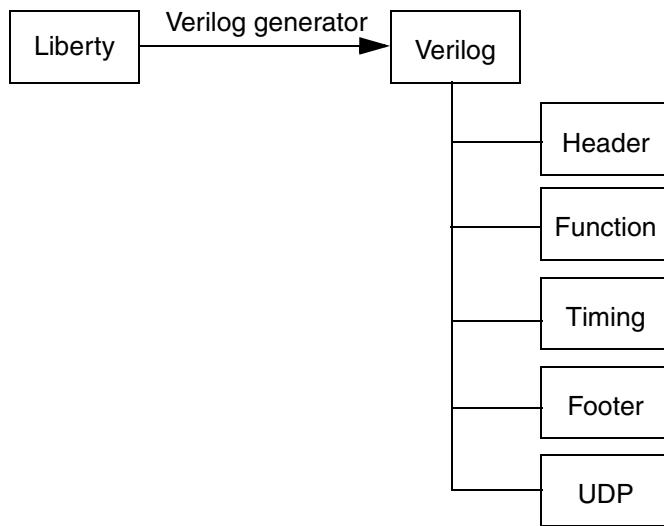
The independent UDP flow includes the following usage characteristics:

- Set the `veriloglib_independent_udp` variable to `true` to use the independent UDP flow in the Verilog model generator.
- Each independent UDP's naming rule is the same as the previously generated Verilog model. The generated independent UDP file is named `UDP.v`.
- The generated independent UDP file (`UDP.v`) is located in the same directory as other generated Verilog files. If the `veriloglib_output_dir` directory is defined, `UDP.v` is output to it; otherwise, it is output to the default directory `library_name_verilog`.

Creating Custom UDPs

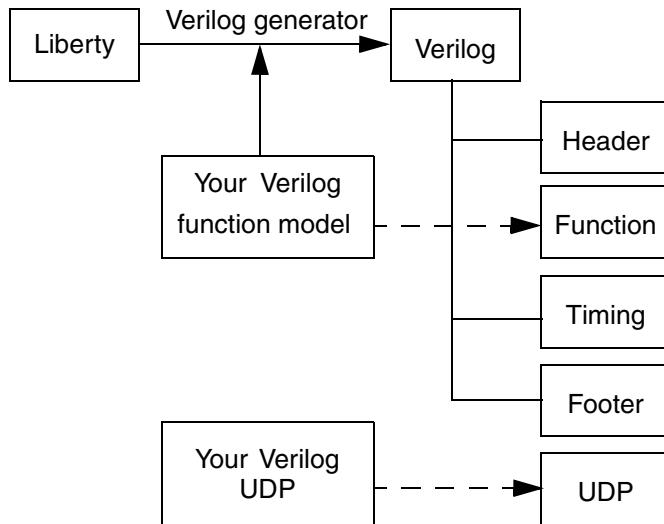
Although the Verilog model generator can generate user-defined primitives (UDPs) automatically, you might prefer to use your own custom UDPs. You can generate Verilog models using your own UDPs by providing a complete Verilog function model and a complete Verilog UDP that matches the cell in the Liberty file. [Figure 5-2 on page 5-36](#) shows the Verilog model generation flow without custom UDPs. The Verilog model is directly generated from the Liberty NCX file.

Figure 5-2 Verilog Model Generation Flow Without Custom UDPs



[Figure 5-3](#) shows the Verilog model generation flow with custom UDPs.

Figure 5-3 Verilog Model Generation Flow With Custom UDPs



In the custom UDP flow, you must define the following:

- Verilog UDP

Normally, the UDP definition is generated automatically in the Verilog model. In the custom UDP flow, you add the UDP definition.

- Verilog function model

The function model is the Verilog code that instantiates the custom UDP and ties it to the logic surrounding it. You must specify this in a function file that is located either in the `library_name_func` directory or in the directory that you specify with the `veriloglib_custom_func_dir` attribute. The Verilog syntax is added to the “Function” section of the generated Verilog.

Enabling the Custom UDP Flow

You can enable the custom UDP flow in one of the following ways:

1. Create a `library_name_func` directory in the same directory as the Liberty file.
2. Define the `veriloglib_custom_func_dir` variable to point to the directory that contains the mapping files, as shown:

```
set veriloglib_custom_func_dir custom_func_dir
```

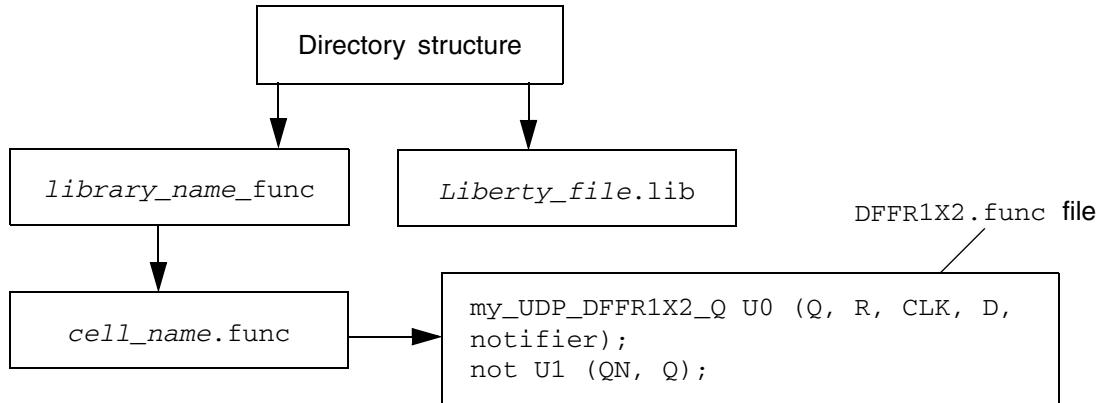
In either case, the Verilog model generator looks for the mapping files in the directory, maps them to the cells in the Liberty file, and uses the custom function file when generating the Verilog model for the mapped cells. The Verilog model generator continues to generate Verilog models as usual for cells that are not mapped.

Defining the Function Section

You can define the function file with a specification of the `cell.func` or `library.fun` file, as described in the following two ways:

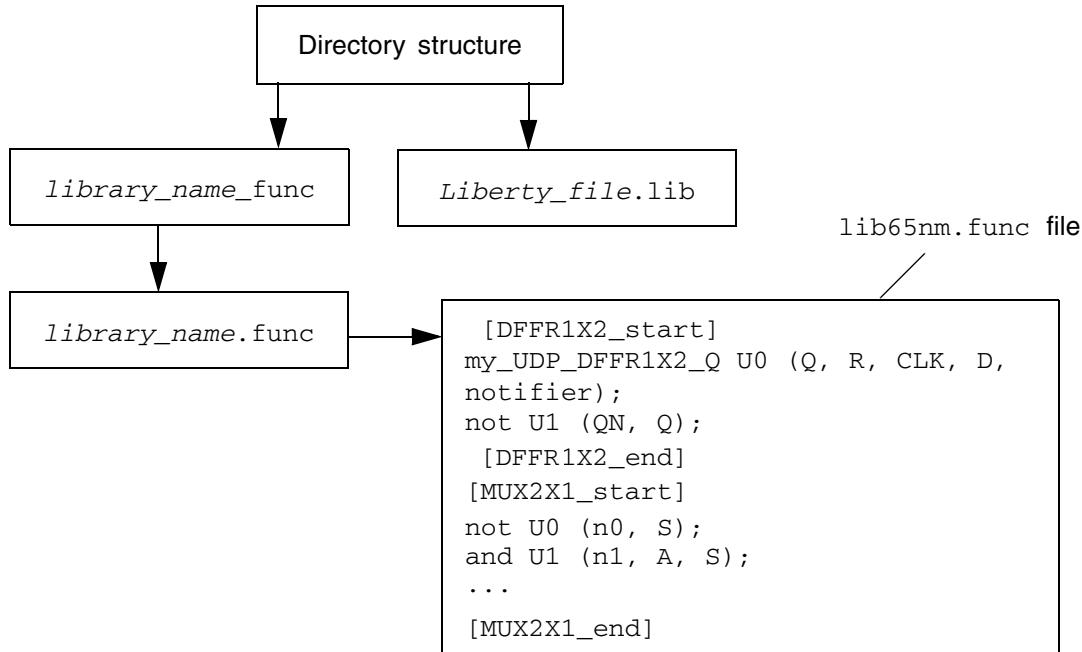
1. [Figure 5-4 on page 5-38](#) shows the specification of the `cell.func` file. Specify the following for the `cell_name.func` file:
 - a. Location: The `cell_name.func` file should be in the `library_name_func` directory or in the directory specified by the `veriloglib_custom_func_dir` attribute.
 - b. Naming convention: There should be a `cell_name.func` file for each cell that requires a custom UDP. The Verilog model generator looks for the `.func` file extension and maps each of the `cell_name` prefixes to a cell in the library. If a cell matching this prefix is found, the contents of the `cell_name.func` file are used in the Verilog model for the cell.
 - c. Content: The function file contains the Verilog instantiation of the custom UDP. The Verilog model generator does not check the syntax or the logical connection of the Verilog specified in the function file. You must make sure that the Verilog description matches the Liberty NCX description.

Figure 5-4 Specifying the `cell_name.func` File



2. [Figure 5-5 on page 5-39](#) shows the specification of the `library.func` file. Specify the following for the `library_name.func` file:
 - a. Location: The `library_name.func` file should be in the `library_name_func` directory or in the directory specified by the `veriloglib_custom_func_dir` attribute.
 - b. Naming convention: The name of this file should be `library_name.func`. The Verilog model generator looks for the `.func` file extension and maps the `library_name` prefix to the name of the library in the Liberty file. If there is a match, the custom UDP is implemented for all cells defined in the `library_name.func` file.
 - c. Content: Reading each `cell_name.func` file can be a burden on the tool; therefore, the Verilog model generator supports the combining of all the custom function files into one file, reducing the number of file reading operations. The name of the file must be `library_name.func`. It includes the function section for each of the cells. The function sections are included in cell separators to uniquely identify them for each cell. For example, `[cell_name_start]` : identifies the start of the cell's function section and `[cell_name_end]` : identifies the end of the cell's function section. The brackets (`[]`) surrounding the `cell_name_start` and `cell_name_end` are required. The information between these separators is similar to what is defined in each `cell_name.func` file.

Figure 5-5 Specifying the library.func File



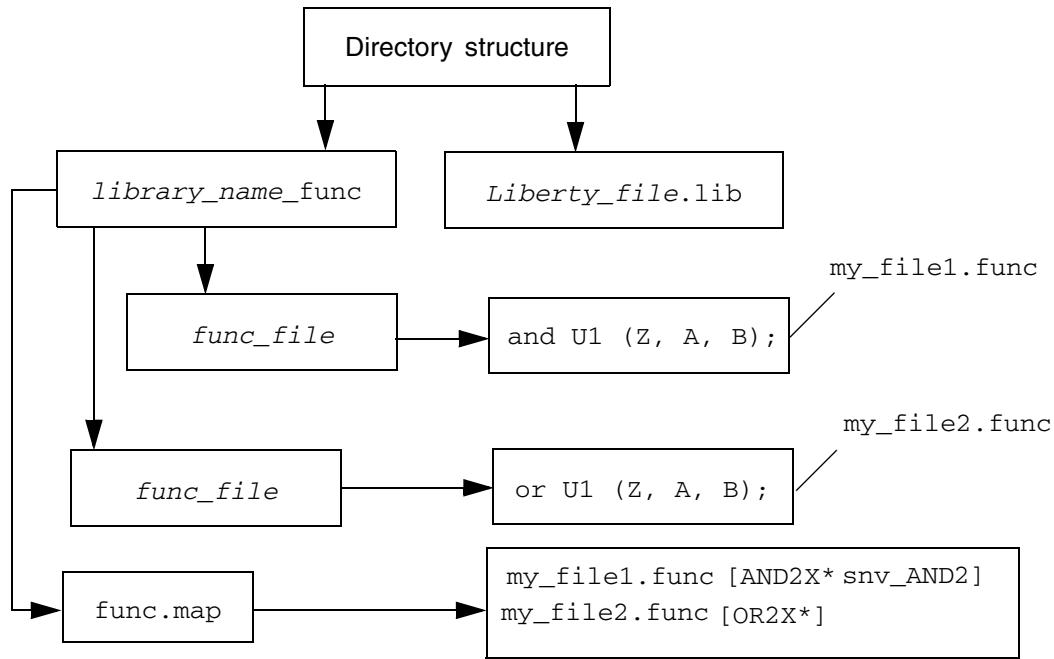
3. [Figure 5-6 on page 5-40](#) shows the specification of the cell.func file. Specify the mapping file: You can use the mapping file to specify the custom function section in the Verilog file. In the general flow, each cell defines its own function section. Specifying the mapping file is helpful if many cells have the same function section or when you are defining the function section for cells that have the same function but different drive strengths.

- Location: The `func.map` mapping file should be in the `library_name_func` directory or in the directory specified by the `veriloglib_custom_func_dir` attribute.
- Naming convention: The name of the mapping file must be `func.map`; however, the files containing the function section can have any name. The mapping file shows how each of the function files map to the cells.
- Content: The mapping file contains multiple entries in the following format:

`func_file_name [cell_name|wildcard name_of_cell ...]`

Where the `func_file_name` is the file that contains the function section for all the cells in the cell list that are enclosed in brackets ([]). The brackets are required. The cell list can include complete cell names or cell names with wildcard characters. Currently, the asterisk (*) character in a suffix is the only wildcard that is supported. You must separate multiple cell names with spaces.

Figure 5-6 Specifying the func.map Mapping File



Search Order for the Custom UDP

There are multiple ways to specify the function section of the custom UDP. The Verilog model generator searches in the following order when looking for custom UDPs for a cell to determine which function section to use:

1. The Verilog model generator looks for custom UDPs in the `func.map` file first. The `func.map` file can be located in the `library_name_func` directory or in the directory that you specify with the `veriloglib_custom_func_dir` attribute. If it finds the file, the Verilog model generator reads the contents of the file and maps the function section to the cells in the corresponding cell list. Once it finds a function section for a cell, it ignores that cell.
2. The second place the Verilog model generator looks for UDP information is the `cell_name.func` file in the `library_name_func` directory or in the directory that you specify with the `veriloglib_custom_func_dir` attribute. The Verilog model generator maps the function section in this file to the cell. If the cell already has a function section defined, for example from a `func.map` definition, the function section in the `cell_name.func` file is ignored.
3. Lastly, the Verilog model generator looks for UDP information in the `library_name.func` file in the `library_name_func` directory or in the directory that you specify with the `veriloglib_custom_func_dir` attribute. Then, the Verilog model generator maps the

function section to the cells that were not mapped in steps 1 and 2. Because this is the last priority in the Verilog model generator's search for UDPs, only the cells that are not previously mapped are mapped at this point.

Guidelines for Specifying the UDP

In the custom UDP flow, the UDP part of the Verilog model is not generated by the Verilog model generator so that the custom UDP can be used. These are the rules for UDP customization:

- Multiple UDPs can be defined for one cell in the custom UDP file.
- The functionality and syntax is not verified by the Verilog model generator, so make sure the UDP function in the function section matches the Liberty definition.

Verilog Model Generator Assumptions

The Verilog model generator assumes the following:

- The Verilog specified in the function section is syntactically correct. No check is done to make sure there are no syntax errors.
- The Verilog model generator does not check that the custom UDP is correctly instantiated, including the names of pins and nets. It does not check that there are no conflicts between the nets or pins in the generated Verilog and the corresponding function section.

For example, if the function section is

and U0 (Z, A, B);

and the Verilog model generated from the .lib file is

```
module AND2X2 (I1, I2, O);
  ...
  // inserted by custom UDP flow
  and U0 (Z, A, B);
  ...
endmodule
```

there is a pin mismatch between what is instantiated [Z, A, B] and what is defined in the module [O, I1, I2]. You must make sure that the custom UDP is instantiated correctly. The Verilog model generator does not check this.

- It is important that you make sure the custom UDP is logically correct and matches the Liberty file used to generate the Verilog.

Verilog Model Details

The outline of a cell's Verilog model is as follows:

```

`timescale 1ns / 1ps
`celldefine
module cell_name ( cell pin list )           // header section
pin declarations
gate instantiation &| UDP instantiation // function description
specify                                         // specify block
specparam definition
path delay
timing checks
endspecify
endmodule
`endcelldefine
[primitive UDP_identifier ( UDP_pin_list ); // UDP definition for
                                             sequential cells
UDP_pin_declaration
UDP_body
endprimitive]
```

Each Verilog model contains the following:

- The ``celldefine` and ``endcelldefine` compiler attributes and the ``timescale` compiler attribute. The Verilog model generator only supports these types of compiler attributes.
- The `module` declaration with all pins in the cell declared in the pin declaration section.
- The cell function, either using instances of Verilog built-in gates or instances of UDPs.
- The `specify` block, where path delay and optional timing checks reside. If the input Liberty file contains timing constraint information, corresponding timing checks appear in the `specify` block.
- The UDP definition if it is instantiated in the cell function.

Header Section

The following sections are included in the header section:

Module Declaration

The module syntax is as follows:

Syntax

```
module cell_name ( cell_pin_list );
```

The arguments are as follows:

- *cell_name* is the name of the cell as specified in the input Liberty file.
- *cell_pin_list* is a list of the pins in the cell. The order of the list follows the pin group orders specified in the input Liberty file.

Example

```
module AND2X2 (A, B, O);
```

Pin Declarations

Pin declarations specify an ordered list of the pins for the module. The order follows the pin order in the input Liberty file. Each pin declaration can be one of the following:

```
inout [range] pin_identifier;
input [range] pin_identifier;
output [range] pin_identifier;
[reg notifier;]
```

The arguments are as follows:

- *range* gives addresses to the individual bits in a multiple-bit bus, depending on the following pin types:
 - Scalar pins: The *range* argument is not used by scalar pins.
 - Liberty bundle: This Liberty construct is ignored and *range* is not used.
 - Liberty bus pins: The *range* value is derived from the bus associated type specification in the input Liberty file.
- *pin_identifier* corresponds to the pin name in the Liberty file and depends on the following pin types:
 - Scalar pins: The *pin_identifier* argument is the corresponding pin name in the input Liberty file.
 - Liberty bundle: The *pin_identifier* argument is each member pin of the bundle group. The *bundle* argument is ignored.
 - Liberty bus pins: The *pin_identifier* is the bus name found in the input Liberty file.

Note:

No bus member is referred here, so no *bus_naming_style* consideration is needed.

- *reg_notifier* is declared as a register in the module where timing check tasks are invoked, and the register notifier is passed as the last argument to a system timing check.

Example

```
input A;
input B;
output O;
```

Function Description

If the function description of the cell is included in the input Liberty file, it is written out in the Verilog model. For black-box cells that have no function description in the .lib file, the Verilog model generator issues a warning and adds a comment in the Verilog function section.

Combinational Function

The combinational function section represents the behavior of the cell using gate instantiations. The Verilog model generator models all combinational cells, including adders, subtracters, multiplexers, and decoder cells with gate instantiation, rather than UDPs.

Syntax

The gate instantiation syntax is as follows:

```
gate_primitive instance_name (terminal_list);
```

The arguments are as follows:

- *gate_primitive*

The *gate_primitive* is the built-in primitive name in Verilog. For each output pin and bidirectional pin, the Verilog model generator converts the Boolean expression in the function attribute directly to Verilog built-in primitives.

Valid primitive names include, and, or, xor, not, nand, nor, xnor, and buf.

- *instance_name*

The *instance_name* is the name of the instance of the gate primitive. The format of the instance name is $U<n>$, where n starts at 0 and increases in steps of 1 for every instance (gate or UDP) that is added.

- *terminal_list*

The *terminal_list* describes how the gate connects to the rest of the model. The characteristics are as follows:

- The terminals are separated by commas.
- The order of the terminal list is determined by the gate primitive definition. Output pins and bidirectional pins are generally specified first, followed by input pins.
- The terminal list can be either pins of the cell or internal connection signals constructed as output from other instances.

- The terminal list determines the internal net-naming convention. In order to avoid possible conflict with existing pin names in a cell, the internal net names are constructed in the format `_net_<n>`, where `n` is an integer starting at 0 and increasing in steps of 1 for every net added. If the format conflicts with the cell's pin names, the Verilog model generator quits with an error. The name does not need to be explicitly declared for a single-bit signal scalar.

The following examples shows the .lib syntax followed by the Verilog equivalent:

.lib Example

```
pin (Y) {
  function: A ^ B' & C;
}
```

Verilog Example

```
not U0(_net_0, B);
xor U1(_net_1, _net_0, A);
and U2(Y, _net_1, C);
```

Sequential Function

Sequential cells are modeled using UDP instances. Each UDP instance models a sequential state. Cells with multiple states are modeled as a netlist of multiple UDP instances. Output logic is modeled using gate instantiation.

Syntax

The UDP instantiation syntax is as follows:

```
udp_name instance_name (terminal_list);
```

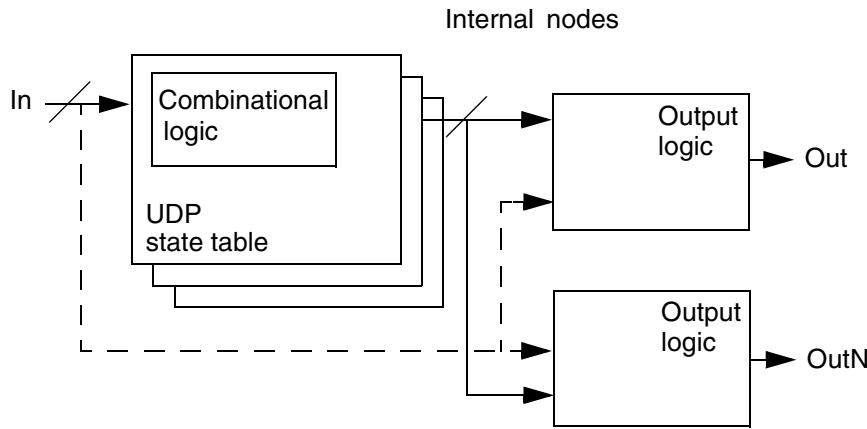
where `udp_name` is formatted as `UDP_cell_port`, and where `cell` specifies the cell name and `port` specifies either the name of the output pin or the internal node.

The `terminal_list` describes how the UDP is connected to the model. The characteristics are as follows:

- The order of the terminal list is determined by the UDP definition. Output pins and bidirectional pins are generally specified first, followed by input pins.
- If a cell has bus or bundle pins, each individual member is modeled independently with its own UDP instantiation of the same UDP primitive.
- Each state of a cell is modeled with a UDP instantiation.

[Figure 5-7 on page 5-46](#) shows the general sequential output model with UDP state tables. Sequential cell functions may contain several sequential “states” and thus can be described with multiple UDP state tables. Each UDP instantiation (state table) describes one sequential state. Cells with multiple sequential states have multiple UDP instantiations.

Figure 5-7 General Sequential Output Model



Input (combinational) logic is included in the UDP state table description. Thus, there is no need to explicitly describe the input logic in the Verilog function description. Output logic is not included in the UDP state table, which requires gate instantiations to be described in Verilog. Output logic is modeled using gate instantiation, similar to combinational functions.

Three-State Function

A three-state function is similar to the combinational function, except that it specifically uses either the `bufif0` or `bufif1` gate primitives. When the `three_state` attribute is specified on an output pin in the input Liberty file, one of the `bufif0` or `bufif1` gate primitives is used to define the output value for the three-state disable state. Whether `bufif0` or `bufif1` is used depends on the active high or low state of the `three_state_signal`.

Syntax

The gate instantiation syntax is as follows:

```
bufif0 instance_name(output_pin, normal_function_signal,  
three_state_signal);  
bufif1 instance_name(output_pin, normal_function_signal,  
three_state_signal);
```

The arguments are as follows:

- `output_pin`

Specifies the output pin name in the .lib file.

- `normal_function_signal`

Specifies the internal connection signal that describes logic in the `function`, `internal_node` or `state_function` attributes in the output pin group.

- *three_state_signal*

Specifies the internal connection signal that describes logic in the `three_state` attribute in the output pin group.

.lib Example

```
pin (Y) {
  function: A&B;
  three_state: EN';
}
```

Verilog Example

```
and U0(_net_0, A, B); //construct function logic;
bufif1 U2(Y, _net_0, EN); //express output value for three state;
```

specparam

The `specparam` syntax is as follows:

```
specparam
delay_param=0.01 [, delay_param=0.01,...];
[constraint_param=0.01 [, constraint_param=0.01,...]];
```

The arguments are as follows:

- *delay_param*
Specifies the parameter name for the path delay.
- *constraint_param*
Specifies the parameter name for the timing checks.
- The unit time (0.01) is used for the path delay and the timing check parameters.

The naming convention for the `delay_param` path is as follows:

`tdelay_input_output_transition[_index]`

where

- *input* is the input pin (`related_pin`) for a timing arc.
- *output* is the output pin for a timing arc.
- *transition* is one of the following: 01, 10, 0z, z1, 1z, z0.
- *index* is an integer value starting at 0 and increasing in steps of 1 when multiple delay parameters with the same input, output, and transition values are specified.

The naming convention for the `constraint_param` timing check parameter is as follows:

`t<check_type>_<start_pin>_[<end_pin>_]<index>`

where

- The *check_type* values include setup, hold, recovery, removal, pulsedwidth, period, and skew.
- *start_pin* is the pin associated with the start event in the related timing check. The start event is determined by the specific timing check type. For more information, see “[Timing Checks](#)” on page 5-52.
- *end_pin* is the pin associated with the end event in the related timing check. The end event is determined by a specific timing check type. For more information, see “[Timing Checks](#)” on page 5-52.

Because *start_pin* and *end_pin* are the same for pulsedwidth and period, only *start_pin* is used in the parameter name.

- *index* is an integer value starting at 0 and increasing in steps of 1 when multiple timing check parameters with the same timing check type, start pin, and end pin values are specified.

Delay Example

```
specify
  specparam
    tdelay_I1_O_01_0=0.01,
    tdelay_I1_O_10_0=0.01;
    (I1 +=> O)=(tdelay_I1_O_01_0, tdelay_I1_O_10_0);
endspecify
```

Timing Check Example

```
specify
  specparam
    tsetup_D_CK_0=0.01,
    thold_CK_D_0=0.01;
    $setuphold(posedge CK , posedge D , tsetup_D_CK_0 , thold_CK_D_0,
    notifier);
endspecify
```

Path Delay

These path delays are described in the following subsections:

- “[Simple Path Delay](#)” on page 5-48
- “[Edge-Sensitive Path Delay](#)” on page 5-49

Simple Path Delay

The simple path delay syntax is as follows:

```
(input [polarity] => output) = (delay_param, delay_param);
(input [polarity] => output) = (delay_param, delay_param, delay_param
delay_param, delay_param, delay_param);
```

The arguments are as follows:

- *input*
Specifies the input pin name.
- *polarity* (optional)
Specifies polarity. Specify "+" for positive unate arcs, "-" for negative unate arcs, or no polarity for `non_unate` arcs or arcs without the `timing_sense` attribute, such as three state.
- *output*
Specifies the output pin name.
- *delay_param*
Specifies the delay parameter name. The number of *delay_param* is either 2 for non-three-state rise and fall delays or 6 (rise, fall, three state) for three-state arc delays. See “[Delay Values](#)” on page 5-52.

The naming convention for the *delay_param* path is as follows:

`tdelay_input_output_transition[_index]`

where

- *input* is the input pin (`related_pin`) for a timing arc.
- *output* is the output pin for a timing arc.
- *transition* is one of the following: 01, 10, 0z, z1, 1z, z0.
- *index* is an integer value starting at 0 and increasing in steps of 1 when multiple delay parameters with the same input, output, and transition values are specified.

The *delay_param* includes timing groups with one of the following values for `timing_type` and no `when` condition, except `default_timing_group`: `combinational`, `combinational_rise`, `combinational_fall`, `three_state_disable`, `three_state_disable_rise`, `three_state_disable_fall`, `three_state_enable`, `three_state_enable_rise`, `three_state_enable_fall`, `preset`, and `clear`.

The default timing group may have no `when` condition but it is modeled using the `ifnone` simple path delay. See “[State-Dependent Path Delay](#)” on page 5-51.

Edge-Sensitive Path Delay

The edge-sensitive path delay syntax is as follows:

```
(edge_identifier input => (output [polarity] : data_source) ) =  
(delay_param, delay_param);
```

The arguments are as follows:

- *edge_identifier*

Specifies `posedge` or `negedge` based on the `timing_type` and `timing_sense` of the timing group in the input Liberty file.

- *data_source*

Describes the flow of data to the path destination. For example, for the DFF cell clock to Q output timing path, the data pin state is actually populated to Q; thus the data pin is *data_source*.

Use the `veriloglib_sdf_edge` variable to control globally whether a timing group is modeled with edge-sensitive path delay, as shown:

```
set veriloglib_sdf_edge [true | false]
```

If set to false, no edge-sensitive path delay is modeled.

If the `veriloglib_sdf_edge` variable is set to true (the default),

- If the timing group has a `timing_type` attribute with a value of `rising_edge` or `falling_edge`, it is modeled with an edge-sensitive path delay. The `rising_edge` value denotes `posedge` on the input. The `falling_edge` value denotes `negedge` on the input.
- If the `timing_type` value in the timing group is `preset`, the input edge is `posedge` if `timing_sense` is `positive_unate`. The input edge is `negedge` if `timing_sense` is `negative_unate`. Otherwise, the path delay is modeled with a simple path delay rather than an edge-sensitive path delay.
- If the `timing_type` value in the timing group is `clear`, the input edge is `negedge` if `timing_sense` is `positive_unate`. The input edge is `posedge` if `timing_sense` is `negative_unate`. Otherwise, the path delay is modeled with a simple path delay rather than an edge-sensitive path delay.

For other cases, the path delay is modeled with a simple path delay.

.lib Example

```
pin (Q) {
    direction : "output";
    function : "IQ";
    timing () {
        related_pin : "CK";
        timing_type : "falling_edge";
    }
}
```

The `falling_edge` value in the .lib example corresponds to `negedge` in the Verilog example.

Verilog Example

```
(negedge CK => Q)=(tdelay_CK_Q_01_0, tdelay_CK_Q_10_0);
```

State-Dependent Path Delay

The state-dependent path delay syntax is as follows:

```
if (sdf_cond) simple path delay
| if (sdf_cond) edge sensitive path delay
| ifnone simple path delay
```

where

sdf_cond specifies the `sdf_cond` attribute in the input Liberty timing group.

If the `sdf_cond` value in the Liberty file is “one bit signal,” meaning that its value is not equal to any pin and does not contain any characters or operators, such as &, |, !, ^, +, *, =, and ~, the Verilog model generator adds a function description for the one-bit signal, with gate instantiation, based on the logic of the `when` condition in the same timing group.

When there are state-dependent timing arcs in the input Liberty file, `ifnone` is used to define the default timing group. It is used only if the input Liberty file has default timing groups.

Either of the following criteria identify default timing groups:

- The `default_timing : true` setting is found in a timing group. In this case, the timing group is used as the default timing group. Besides its `when` condition (`sdf_cond`) being used to model a state-dependent path delay, the timing group is used again to model a default timing arc using `ifnone`, ignoring that the timing group has a `when` condition.
- Among timing groups with the same `related_pin` attribute, if one timing group does not have a `when` condition and all the other timing groups have `when` conditions, the one without the `when` condition is regarded as the default timing group and is used to model a default path delay, using `ifnone`.

.lib Example

```
pin Z {
    timing() {
        related_pin      : "D";
        timing_sense    : negative_unate;
        when            : "(A'*B'*C') ";
        sdf_cond        : "ALBLCL";
    }
}
```

Verilog Example

```
not U2 (_net_1, B);
not U3 (_net_2, A);
not U4 (_net_3, C);
```

```

and U5 (ALBLCL, _net_1, _net_2, _net_3);
...
if (ALBLCL) (D ==> Z)=(tdelay_D_Z_01_0, tdelay_D_Z_10_0);

```

Delay Values

Generally, there is output rise and output fall for a path delay. The Verilog model generator uses two values (t01, t10) for one path delay entry. With three-state timing arcs, six values are modeled in one path delay entry, as follows:

- t01, t10, t0z, tz1, t1z, tz0
- Once a `three_state_enable` timing group is found, the six values (t01, t10, t0z, tz1, t1z, tz0) are used regardless of whether there are corresponding `three_state_disable` rising or falling arcs. The Verilog model generator always declares six path delay `specparams` for these six delay values.

Example

```
(EN+ ==> O)=(tdelay_EN_O_Z1_0, tdelay_EN_O_Z0_0, tdelay_EN_O_Z1_0,
tdelay_EN_O_Z1_0, tdelay_EN_O_Z0_0, tdelay_EN_O_Z0_0);
```

Bus and Bundle Signal

Based on the input Liberty content, if the delay applies to all members of the bundle or bus, then the bundle or bus name is used in the path delay. Otherwise, each bundle or bus member signal is modeled in separate path delay entries. When referring to individual bus members, `bus_naming_style` in Liberty is considered to convert a Liberty bus member name to a Verilog bus member name because Verilog only allows the `%s[%d]` style.

Duplicate Path Delay

Timing groups in Liberty may result in multiple path delays with the same state condition, same edge for the same input and output, and therefore duplicate path delays. In this case, the Verilog model generator avoids generating duplicate path delays. If a potential duplicate path delay is found, the Verilog model generator models the path delay once and skips later timing groups for those duplicate path delays.

Timing Checks

This section includes timing checks for `setup`, `hold`, `setuhold`, `recovery`, `removal`, `recrm`, `pulse width`, `period`, `skew`, and include `edge` statements and conditional timing checks and duplicate timing checks.

Setup

The `setup` syntax is as follows:

Syntax

```
$setup ([edge_identifier_start] start_pin [&&& start_cond],
```

```
[edge_identifier_end] end_pin [&& end_cond], setup_param, notifier);
```

The arguments are as follows:

- *edge_identifier_start* (optional)
If specified, *edge_identifier_start* is the edge associated with the start pin. See “[Edge Statements](#)” on page 5-61.
- *start_pin*
Specifies the data pin.
- *start_cond* (optional) based on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- *edge_identifier_end* is optional, based on the input Liberty file. If specified, it is the edge associated with the end pin. See “[Edge Statements](#)” on page 5-61.
- *end_pin* is the clock pin.
- *end_cond* is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- *setup_param* is the timing limit. For naming conventions, see “[specparam](#)” on page 5-47.
- *notifier* is a `reg` variable used to detect timing check violations behaviorally. Whenever a timing violation occurs, the system task updates the value of the notifier. The Verilog model generator uses notifier to set the UDP output to `x` when a timing check violation occurs. All timing checks share the same notifier.
- The `$setup` timing check is modeled only when there is no matching hold timing check found in the Liberty timing group. See “[Setuphold](#)” on page 5-55 for more information.

You can use the `veriloglib_combine_timingcheck` variable to combine setup and hold timing checks to one `$setuphold` timing check. If a setup timing check and a hold timing check have the same clock and data pin and the same edge identifier and condition for a clock and data pin, respectively, the two timing checks are combined into one `$setuphold` timing check. However, `no_edge` is regarded as a special edge identifier and “no condition” is regarded as a special condition.

Valid values for the `veriloglib_combine_timingcheck` variable include `setuphold` and `recrem`, for example:

```
set veriloglib_combine_timingcheck {setuphold}
```

The Verilog model generator does not combine setup and hold timing checks by default.

You can set the `veriloglib_write_recrem_as_setuphold` variable to `true` to output timing checks in the same format but with the keyword `$recrem` replaced by `$setuphold`, `$recovery` replaced by `$setup`, or `$removal` replaced by `$hold`. The valid values are `true` and `false`. The default is `false`.

Hold

The `hold` syntax is as follows:

Syntax

```
$hold ([edge_identifier_start] start_pin [&& start_cond],  
[edge_identifier_end] end_pin [&& end_cond], hold_param, notifier);
```

The arguments are as follows:

- `edge_identifier_start` (optional)

If specified, `edge_identifier_start` is the edge associated with the start pin. See “[Edge Statements](#)” on page 5-61.

- `start_pin` is the clock pin.

- `start_cond` is optional and is dependent on the Liberty input file. See “[Conditional Timing Checks](#)” on page 5-66.

- `end_pin` is the data pin.

- `end_cond` is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.

- `hold_param`

Specifies the timing limit. For naming conventions, see “[specparam](#)” on page 5-47.

- `notifier` is a `reg` variable used to detect timing check violations behaviorally. For more information, see “[Setup](#)” on page 5-52.

- The `$hold` timing check is modeled only when there is no matching setup timing check found in the Liberty timing group. For more information, see “[Setuphold](#)” on page 5-55.

You can use the `veriloglib_combine_timingcheck` variable to combine setup and hold timing checks to one `$setuphold` timing check. If a setup timing check and a hold timing check have the same clock and data pin and the same edge identifier and condition for a clock and data pin, respectively, the two timing checks are combined into one `$setuphold` timing check. However, `no_edge` is regarded as a special edge identifier and “no condition” is regarded as a special condition.

Valid values for the `veriloglib_combine_timingcheck` variable include `setuphold` and `recrem`, for example:

```
set veriloglib_combine_timingcheck {setuphold}
```

The Verilog model generator does not combine setup and hold timing checks by default.

You can set the `veriloglib_write_recrem_as_setuphold` variable to `true` to output timing checks in the same format but with the keyword `$recrem` replaced by `$setuphold`, `$recovery` replaced by `$setup`, or `$removal` replaced by `$hold`. The valid values are `true` and `false`. The default is `false`.

Setuphold

The `setuphold` syntax is as follows:

Syntax

```
$setuphold ([edge_identifier_clock] clock_pin[&&&clock_cond],  
[edge_identifier_data] data_pin [&&&data_cond], setup_param, hold_param,  
notifier);
```

The arguments are as follows:

- `edge_identifier_clock` is optional and is based on the input Liberty file. If specified, it is the edge associated with the clock pin. See “[Edge Statements](#)” on page 5-61.
- `clock_pin` is the `start_pin` for the associated hold timing check and `end_pin` for the associated setup timing check.
- `clock_cond` is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- `edge_identifier_data` is optional and is based on the input Liberty file. If specified, it is the edge associated with the data pin. See “[Edge Statements](#)” on page 5-61.
- `data_pin` is the `end_pin` for the associated hold timing check and `start_pin` for the associated setup timing check.
- `data_cond` is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- `setup_param` is the timing limit for the associated setup timing check. For naming conventions, see “[specparam](#)” on page 5-47.
- `hold_param` is the timing limit for the associated hold timing check. For naming conventions, see “[specparam](#)” on page 5-47.
- `notifier` is a `reg` variable used to detect timing check violations behaviorally. For more information, see “[Setup](#)” on page 5-52.

- The Verilog model generator tries to combine every setup and hold timing check into a single `$setuphold` timing check. If a setup timing check and a hold timing check have the same clock and data pin and the same edge identifier and condition for a clock and data pin, respectively, the two timing checks are combined into one `$setuphold` timing check. However, `no_edge` is regarded as a special edge identifier and “no condition” is regarded as a special condition.

You can use the `veriloglib_combine_timingcheck` variable to combine setup and hold timing checks to one `$setuphold` timing check, as shown:

```
set veriloglib_combine_timingcheck {setuphold}
```

Valid values for the `veriloglib_combine_timingcheck` variable include `setuphold` and `recrem`. The Verilog model generator does not combine setup and hold timing checks by default.

You can set the `veriloglib_write_recrem_as_setuphold` variable to `true` to output timing checks in the same format but with the keyword `$recrem` replaced by `$setuphold`, `$recovery` replaced by `$setup`, or `$removal` replaced by `$hold`. The valid values are `true` and `false`. The default is `false`.

Recovery

The `recovery` syntax is as follows:

Syntax

```
$recovery ([edge_identifier_control] control_pin [&& control_cond],  
[edge_identifier_clock] clock_pin [&& clock_cond], recovery_param,  
notifier);
```

The arguments are as follows:

- `edge_identifier_control` is optional, based on the input Liberty file. If specified, it is the edge associated with the control pin. See “[Edge Statements](#)” on page 5-61.
- `control_pin` is the asynchronous control pin, such as clear and preset. It is the `start_pin` for the recovery timing check.
- `control_cond` is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- `edge_identifier_clock` is optional, based on the input Liberty file. If specified, it is the edge associated with the clock pin. See “[Edge Statements](#)” on page 5-61.
- `clock_pin` is the clock pin. It is the `end_pin` for the recovery timing check.
- `clock_cond` is optional and is dependent on the input Liberty file.
- `recovery_param` is the timing limit. For naming conventions, see “[specparam](#)” on page 5-47.

- `notifier` is a `reg` variable used to detect timing check violations behaviorally. See “[Setup](#)” on page 5-52 for more information.
- `$recovery` timing check is modeled only when there is no matching removal timing check in the Liberty timing groups. See “[Recrem](#)” on page 5-58.

You can use the `veriloglib_combine_timingcheck` variable to combine recovery and removal timing checks to one `$recrem` timing check. If a recovery timing check and a removal timing check have the same clock and control pin and the same edge identifier and condition for a clock and control pin, respectively, the two timing checks are combined into one `$recrem` timing check.

Valid values for the `veriloglib_combine_timingcheck` variable include `setuphold` and `recrem`, for example:

```
set veriloglib_combine_timingcheck {recrem}
```

The Verilog model generator does not combine recovery and removal timing checks by default.

You can set the `veriloglib_write_recrem_as_setuphold` variable to `true` to output timing checks in the same format but with the keyword `$recrem` replaced by `$setuphold`, `$recovery` replaced by `$setup`, or `$removal` replaced by `$hold`. The valid values are `true` and `false`. The default is `false`.

Removal

The removal syntax is as follows:

Syntax

```
$removal ([edge_identifier_control] control_pin [&& control_cond],  
[edge_identifier_clock] clock_pin [&& clock_cond], removal_param,  
notifier);
```

The arguments are as follows:

- `edge_identifier_control` (optional) based on the input Liberty file. If specified, it is the edge associated with the control pin. See “[Edge Statements](#)” on page 5-61.
- `control_pin`
Specifies an asynchronous control pin, such as clear and preset. It is the end pin of a removal timing check.
- `control_cond` (optional) dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- `edge_identifier_clock` (optional) based on the input Liberty file. If specified, it is the edge associated with the clock pin. See “[Edge Statements](#)” on page 5-61.

- *clock_pin*
Specifies the clock pin. It is the start pin of a removal timing check.
- *clock_cond* is dependent on the input Liberty file.
- *removal_param* is the timing limit. For naming conventions, see “[specparam](#)” on [page 5-47](#).
- *notifier* is a `reg` variable used to detect timing check violations behaviorally. See “[Setup](#)” on [page 5-52](#) for more information.
- The `$removal` timing check is modeled only when there is no matching recovery timing check in the Liberty timing groups. See “[Recrem](#)” on [page 5-58](#) for more information

You can use the `veriloglib_combine_timingcheck` variable to combine recovery and removal timing checks to one `$recrem` timing check. If a recovery timing check and a removal timing check have the same clock and control pin and the same edge identifier and condition for a clock and control pin, respectively, the two timing checks are combined into one `$recrem` timing check.

Valid values for the `veriloglib_combine_timingcheck` variable include `setuphold` and `recrem`, for example:

```
set veriloglib_combine_timingcheck {recrem}
```

The Verilog model generator does not combine recovery and removal timing checks by default.

You can set the `veriloglib_write_recrem_as_setuphold` variable to `true` to output timing checks in the same format but with the keyword `$recrem` replaced by `$setuphold`, `$recovery` replaced by `$setup`, or `$removal` replaced by `$hold`. The valid values are `true` and `false`. The default is `false`.

Recrem

The `recrem` syntax is as follows:

Syntax

```
$recrem ([edge_identifier_control] control_pin[&&control_cond],  
[edge_identifier_clock] clock_pin [&&clock_cond], recovery_param,  
removal_param, notifier);
```

The arguments are as follows:

- *edge_identifier_control* (optional), based on the input Liberty file. If specified, it is the edge associated with the control pin. See “[Edge Statements](#)” on [page 5-61](#).
- *control_pin* is the asynchronous control pin, such as clear and preset.

- *control_cond* (optional) and is dependent on input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- *edge_identifier_clock* is optional, based on the input Liberty file. If specified, it is the edge associated with the clock pin. See “[Edge Statements](#)” on page 5-61.
- *clock_pin*
Specifies the clock pin.
- *clock_cond* is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- *recovery_param* is the timing limit for the associated recovery timing check. For naming conventions, see “[specparam](#)” on page 5-47.
- *removal_param*: timing limit for the associated removal timing check. For the naming convention, see “[specparam](#)” on page 5-47.
- *notifier* is a `reg` variable used to detect timing check violations behaviorally. See the Setup section for more information.

The Verilog model generator tries to combine every recovery and removal timing check to a single `$recrEm` timing check. If a recovery timing check and a removal timing check have the same clock and control pin and the same edge identifier and condition for a clock and control pin, respectively, the two timing checks are combined into one `$recrEm` timing check.

You can use the `veriloglib_combine_timingcheck` variable to combine recovery and removal timing checks to one `$recrEm` timing check, as shown:

```
set veriloglib_combine_timingcheck {recrEm}
```

Valid values for the `veriloglib_combine_timingcheck` variable include `setuphold` and `recrEm`. The Verilog model generator does not combine recovery and removal timing checks by default.

You can set the `veriloglib_write_recrEm_as_setuphold` variable to `true` to output timing checks in the same format but with the keyword `$recrEm` replaced by `$setuphold`, `$recovery` replaced by `$setup`, or `$removal` replaced by `$hold`. The valid values are `true` and `false`. The default is `false`.

Pulse Width

The `pulse width` syntax is as follows:

Syntax

```
$width (edge_identifier pin[&&&cond>], pulsewidth_param, 0 ,  
notifier);
```

The arguments are as follows:

- *edge_identifier*
Specifies the start event edge associated with the constraint pin based on the input Liberty file. See “[Edge Statements](#)” on page 5-61.
- *pin*
Specifies the constraint pin.
- *cond* is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- *pulsewidth_param*
Specifies the timing limit for the width timing check. The default threshold is 0.
- *notifier* is a `reg` variable used to detect timing check violations behaviorally. For more information, see “[Setup](#)” on page 5-52.

Period

The `period` syntax is as follows:

Syntax

```
$period (edge_identifier pin[&&&cond], period_param, notifier);
```

The arguments are as follows:

- *edge_identifier*
Specifies the start event edge associated with the constraint pin, based on the input Liberty file. See “[Edge Statements](#)” on page 5-61.
- *pin*
Specifies the constraint pin.
- *cond* is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.
- *period_param*
Specifies the timing limit for the period timing check.
- *notifier* is a `reg` variable used to detect timing check violations behaviorally. For more information, see “[Setup](#)” on page 5-52.

Skew

The `skew` syntax is as follows:

Syntax

```
$skew ([edge_identifier_start] start_pin [&& start_cond],  
[edge_identifier_end] end_pin [&& end_cond], skew_param, notifier);
```

The arguments are as follows:

- `edge_identifier_start` (optional) based on the input Liberty file.

If specified, `edge_identifier_start` is the edge associated with the start pin. See “[Edge Statements](#)” on page 5-61.

- `start_pin`

Specifies the start clock pin. It is the `related_pin` of the timing group in the input Liberty file.

- `start_cond` is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.

- `edge_identifier_end` is optional and is based on the input Liberty file. If specified, it is the edge associated with the end pin. See “[Edge Statements](#)” on page 5-61.

- `end_pin`

Specifies the end clock pin. It is the `parent_pin` of the timing group in the input Liberty file.

- `end_cond` is optional and is dependent on the input Liberty file. See “[Conditional Timing Checks](#)” on page 5-66.

- `skew_param`

Specifies the timing limit for the skew timing check. For the naming convention, see “[specparam](#)” on page 5-47.

- `notifier` is a `reg` variable used to detect timing check violations behaviorally. See “[Setup](#)” on page 5-62 for more information.

Edge Statements

The following rules apply to setup, hold, recovery, removal, and skew timing checks:

- The `sdf_edges` attribute in the input Liberty file defines the edge specification on the start pin and the end pin. The `sdf_edges` attribute can be one of the following edge types: `noedge`, `start_edge`, `end_edge`, or `both_edges`. The default is `noedge`.

- If the `sdf_edge` attribute does not specify an edge, the exact edge is determined based on the attributes in the timing group (as described in the following sections) for each timing check that must be specified.

Setup

The following rules apply to setup timing checks:

- The `rise_constraint` group or the `intrinsic_rise` attribute denotes `posedge` for the start pin (data pin).
- The `fall_constraint` group or the `intrinsic_fall` attribute denotes `negedge` for the start pin (data pin).
- A value of `setup_rising` for the `timing_type` attribute denotes `posedge` for the end pin (clock pin).
- A value of `setup_falling` for the `timing_type` attribute denotes `negedge` for the end pin (clock pin).

Hold

The following rules apply to hold timing checks:

- A value of `hold_rising` for the `timing_type` attribute denotes `posedge` for the start pin (clock pin).
- A value of `hold_falling` for the `timing_type` attribute denotes `negedge` for the start pin (clock pin).
- The `rise_constraint` group or the `intrinsic_rise` attribute denotes `posedge` for the end pin (data pin).
- The `fall_constraint` group or the `intrinsic_fall` attribute denotes `negedge` for the end pin (data pin).

Edge Statements in \$setup and \$hold Timing Check .lib Example

```
cell (DFFR1X2) {
    ff (IQ,IQN) {
        next_state : "D";
        clocked_on : "CK'";
        clear : "R'";
    }
    pin (D) {
        direction : "input";
        rise_capacitance : 0.012667;
        fall_capacitance : 0.012721;
        timing () {
            related_pin : "CK";
            timing_type : "setup_falling";
            rise_constraint ("vio_3_3_1") {
                index_1("...");
            }
        }
    }
}
```

```

        index_2("...") ;
        values("...", "...", "...");
    }
    fall_constraint ("vio_3_3_1") {
        index_1("...");
        index_2("...");
        values("...", "...", "...");
    }
}
timing () {
    related_pin : "CK";
    timing_type : "hold_falling";
    rise_constraint ("vio_3_3_1") {
        index_1("...");
        index_2("...");
        values("...", "...", "...");
    }
    fall_constraint ("vio_3_3_1") {
        index_1("...");
        index_2("...");
        values("...", "...", "...");
    }
}
}
...
}

```

The `setup_falling` and `hold_falling` values in the .lib example correspond to `negedge CK` in the following Verilog example. The `rise_constraint` value corresponds to `posedge D` in the example, and `fall_constraint` corresponds to `negedge D`.

Edge Statements in \$setup and \$hold Timing Checks Verilog Example

```
$setuphold(negedge CK , posedge D , tsetup_D_CK_0 , thold_CK_D_0 ,
notifier);
$setuphold(negedge CK , negedge D , tsetup_D_CK_1 , thold_CK_D_1,
```

Recovery

The following rules apply to recovery timing checks:

- The `rise_constraint` group or the `intrinsic_rise` attribute denotes `posedge` for the start pin (control pin).
- The `fall_constraint` group or the `intrinsic_fall` attribute denotes `negedge` for the start pin (control pin).
- A value of `recovery_rising` for the `timing_type` attribute denotes `posedge` for the end pin (clock pin).
- A value of `recovery_falling` for the `timing_type` attribute denotes `negedge` for the end pin (clock pin).

Removal

The following rules apply to removal timing checks:

- A value of `removal_rising` for the `timing_type` attribute denotes `posedge` for the start pin (clock pin).
- A value of `recovery_falling` for the `timing_type` attribute denotes `negedge` for the end pin (clock pin).

Edge Statements in \$recover and \$removal Timing Check .lib Example

```
pin (R) {
    direction : "input";
    rise_capacitance : 0.021705;
    rise_capacitance_range(0.020768,0.022129);
    capacitance : 0.021528;
    fall_capacitance : 0.021528;
    fall_capacitance_range(0.019914,0.023608);
    timing () {
        related_pin : "CK";
        timing_type : "recovery_falling";
        rise_constraint ("vio_3_3_1") {
            index_1("...");
            index_2("...");
            values("...", "...", ...");
        }
    }
    timing () {
        related_pin : "CK";
        timing_type : "removal_falling";
        rise_constraint ("vio_3_3_1") {
            index_1("...");
            index_2("...");
            values("...", "...", ...");
        }
    }
    timing () {
        related_pin : "R";
        timing_type : "min_pulse_width";
        fall_constraint ("constraint_3_0_1") {
            index_1("...");
            values("...");
        }
    }
}
```

The `recovery_falling` and `removal_falling` values in the .lib example correspond to `negedge CK` in the following Verilog example and the `rise_constraint` value corresponds to `posedge R`.

Edge Statements in \$recover and \$removal Timing Check Verilog Example

```
$recrem(posedge R, negedge CK, trecover_R_CK_0, tremoval_CK_R_0,
```

Skew

The following rules apply to skew timing checks:

- A value of `skew_rising` for the `timing_type` attribute denotes `posedge` for the start pin, which is the start clock pin, the `related_pin` in the timing group.
- A value of `skew_falling` for the `timing_type` attribute denotes `negedge` for the start pin, which is the start clock pin, the `related_pin` in the timing group.
- The `rise_constraint_group` or `intrinsic_rise` attribute denotes `posedge` for the end pin, which is the end clock pin, the `parent_pin` of timing group.
- The `fall_constraint_group` or `intrinsic_fall` attribute denotes `negedge` for the end pin, which is the end clock, pin, or `parent_pin` of the timing group.

Pulsewidth

Liberty provides the following ways to specify pulse width and to deduce the edge on a pulse width constraint pin.

For a simple attribute in a pin group,

- The `min_pulse_width_high` attribute denotes `posedge` on the constraint pin.
- The `min_pulse_width_low` attribute denotes `negedge` on the constraint pin.

For a `min_pulse_width` group in the pin group,

- The `constraint_high` attribute denotes `posedge` on the constraint pin.
- The `constraint_low` attribute denotes `negedge` on the constraint pin.

For a timing group with a value of `min_pulse_width` for the `timing_type` attribute,

- The `rise_constraint_group` denotes `posedge` on the constraint pin.
- The `fall_constraint_group` denotes `negedge` on the constraint pin.

Period

Liberty provides the following ways to specify period and to deduce the edge on the period constraint pin:

- For the `minimum_period` attribute, which is a simple attribute in a pin group,
 - The edges for period checks cannot be identified from the attribute itself. First, the Verilog model generator checks for a timing group in the output pin groups with a value of `rising_edge` or `falling_edge` for the `timing_type` attribute, where the value of the `related_pin` attribute is the same clock pin as in the period checks.
 - A value of `rising_edge` denotes `posedge`, and a value of `falling_edge` denotes `negedge`.

- If a similar timing group cannot be found, the Verilog model generator issues a warning and assumes a `posedge` for the period check.
- For the `minimum_period` group in the pin group, the Verilog model generator checks for the existence of the constraint attribute.

Note:

For edge determination, the same rules as for the `minimum_period` attribute apply.

- For a timing group with a `minimum_period` value for the `timing_type` attribute,
 - The `rise_constraint` group denotes `posedge` on the constraint pin.
 - The `fall_constraint` group denotes `negedge` on the constraint pin.

Conditional Timing Checks

For setup, hold, recovery, removal, and skew timing checks, the following attributes are related to pin conditions: `when`, `when_start`, `when_end`, `sdf_cond`, `sdf_cond_start`, and `sdf_cond_end`. The `sdf_cond_start` attribute maps to the start pin condition; the `sdf_cond_end` attribute maps to the end pin condition. When none of these attributes exist, no condition is applied to the corresponding pin, whether it is a start pin or an end pin.

For pulse width, the Verilog model generator checks the following conditions for timing:

- If there is a simple attribute in pin group.
- If there is no condition for a constraint pin.
- For a `min_pulse_width` group in pin group, the Verilog model generator checks to see if the `when` and `sdf_cond` attributes are related to the constraint pin condition. It also checks to see if the `sdf_cond` attribute is used directly as the constraint pin condition if it exists.
- When the timing group has a value of `min_pulse_width` for the `timing_type` attribute, the Verilog model generator checks to make sure that the `when` and `sdf_cond` attributes are related to the constraint pin condition. It also checks to make sure that the `sdf_cond` attribute is used directly as the constraint pin condition if it exists.

For period, the Verilog model generator checks the following conditions for timing:

- If there is a simple attribute in pin group.
- If there is no condition for a constraint pin.
- For a `minimum_period` group in pin group, the Verilog model generator checks to see if the `when` and `sdf_cond` attributes are related to the constraint pin condition. It also checks to see if the `sdf_cond` attribute is used directly as the constraint pin condition if it exists.

- When the `timing` group has a value of `minimum_period` for the `timing_type` attribute, the Verilog model generator checks to make sure that the `when` and `sdf_cond` attributes are related to the constraint pin condition. It also checks to make sure that the `sdf_cond` attribute is used directly as the constraint pin condition if it exists.

As with the state dependent path delay, if the `sdf_cond`, `sdf_cond_start`, or `sdf_cond_end` attributes are recognized as “single-bit signal,” the Verilog model generator adds a function description for this one-bit signal by instantiating a gate, based on the corresponding logic of the `when`, `when_start`, or `when_end` attributes. For more information, see [“State-Dependent Path Delay” on page 5-51](#).

Duplicate Timing Checks

If timing groups in Liberty have multiple timing checks with the same type, same edge identifier, pin, and condition for both the start and end pins, these timing checks are considered to be duplicate timing checks. The Verilog model generator avoids generating duplicate timing checks. However, if potential duplicate timing checks occur, the Verilog model generator generates only one entry for the timing check and skips the later duplicate timing checks.

UDP Definition

The UDP definition syntax is as follows:

```
primitive UDP_identifier (UDP_pin_list); //UDP header;
UDP_pin_declaration; //UDP pin declaration;
table
 //UDP body or UDP state
endtable
endprimitive
```

The `UDP_identifier` in the UDP header is in the format `UDP_cell_port`, where the arguments are as follows:

- `cell` and `port` describe the behavior of the UDP. The `port` value can be either output pin or internal node.
- `UDP_pin_list` is a comma-separated list containing the output and input pins. One UDP has exactly one output pin. The output pin is the first pin in the port list.

For timing check violations, a `notifier` pin is explicitly added to the end of the pin list as an input signal. The order of the input pins in the pin list determines the order of the inputs in the UDP state table definition. For more information, see the example at the end of this section.

The UDP pin declaration is as follows:

```
output q;
reg q;
input input_pin[, input_pin...], notifier
```

where the *input_pin* values are named in1, in2, and so on.

The UDP state table is as follows:

```
table
input_state...] : current_state : next_state;
...
Endtable
```

The arguments are as follows:

- *input_state*

Specifies a space-separated list of input values. Valid characters for the input values are 0, 1, r, f, b, ?, x and *.

- *current_state*

Specifies the output current state value. Valid characters are 0, 1, x, ?, and b.

- *next_state*

Specifies the output next state value. Valid characters are 0, 1, x, ?, and -. Each row defines the output, based on the current state, particular combinations of input values, and one input transition at the most. The order of the input state fields of each row of the state table is taken directly from the port list in the UDP definition header. Any event on notifier puts the output in the x state.

Three-valued logic tends to make pessimistic estimates of the output when one or more inputs are unknown. UDPs can be used to reduce this pessimism. The Verilog model generator provides the ability to write pessimism reduction in UDP.

To reduce pessimism in UDP, the general rule is that “x” represents either 0 or 1. If an input pin can be set to either 0 or 1 while having no impact on the output state, even if the state of the input is unknown, there won’t be any impact on the output state. There are other rules for certain conditions, such as ignoring certain edges, which must be added specially.

Example

```
primitive UDP_DFF1X1_Q (q, in1, in2, notifier);
output q;
reg q;
input in1, in2, notifier;
table
CP      D      notifier      : Q      Q+1 ;
(01)    0      ?            :?      : 0  ;
```

```

(01) 1      ?      :?      : 1 ;
(1?) ?      ?      :?      : - ;
(?0) ?      ?      :?      : - ;

?      *      ?      :?      : - ;
?      ?      *      :?      : x ;
//Added as part of pessimism reduction
x1    0      ?      : 0     : 0 ;
0x    0      ?      : 0     : 0 ;
x1    1      ?      : 1     : 1 ;
0x    1      ?      : 1     : 1 ;
endtable

endprimitive

```

Creating the Testbench

Library Compiler can automatically generate testbenches with expected outputs for the entire library during Verilog generation. To automatically generate library-level testbenches with input stimuli and expected outputs for the cell during Verilog generation, set the `veriloglib_tb_compare` variable to a value from 0 to 5, as shown in the following example:

```
set veriloglib_tb_compare = [0 | 1 | 2 | 3 | 4 | 5 ]
```

The values are described as follows:

0 (the default)

If you set the variable to 0, the default setting, no testbench is created.

1

If you set the variable to 1, each time an input changes, Library Compiler checks to ensure that the result from the previous input matches the expected result. You should define enough time between the input vectors for the outputs to propagate and stabilize. This check is useful for unit-delay structural libraries.

2

If you set the variable to 2, each time an expected output changes, Library Compiler checks to ensure that the actual output has changed or is changing at the same time. This check is useful if you are unable to define some of the expected outputs. It can also be used if the expected output signals are timed pessimistically and if changes usually occur later. The actual outputs can have many unpredictable changes that are not detected. This check reports known pins and does not check unknown pins.

3

If you set the variable to 3, each time an expected output changes, Library Compiler checks to ensure that the actual output makes an identical change at the same time, verifying the correct pin state and transition time.

4

If you set the variable to 4, Library Compiler checks each expected output against the corresponding actual output to ensure they are identical at the same time. When the actual output pins are active, Library Compiler checks to ensure that the expected output signals are the same at the same time.

5

If you set the variable to 5, Library Compiler checks each expected output against the corresponding actual output to ensure they are identical at the same time. When the actual output pins are active, Library Compiler checks to ensure that the expected output signals are the same at the same time.

When you set `veriloglib_tb_compare` to a value from 1 to 5, enabling testbench generation, Library Compiler creates the following files:

- Testbench files

The testbench files, generated in Verilog format, specify the SDF file for back annotation and the hierarchical path of the cell instance. They do this by using the `$sdf_annotation` function. Library Compiler reads the input stimulus file, applies the input stimulus, writes out the simulation output changes, compares the output with any expected output, and reports the differences. Library Compiler creates or defines a netlist of the Verilog model instances being tested and writes out a testbench file for each library that is tested and a testbench file for each cell. Testbench files use the contents of the input stimulus file as a test vector.

The testbench file name format is `library_tb.v` for an entire library and `library_cell_tb.v` for a specific cell. The simulation output file name format is `library_tb.out` or `library_cell_tb.out`. The output file format is compatible with the input stimulus file. The SDF file name format is `o_cell.sdf` and the module instantiation name format is `cell_inst`. The time scale is hard-coded as ``timescale 1 ns/10 ps`.

- Input stimulus files

The `library_tb.sen` and `library_cell_tb.sen` input stimulus files contain a pin list with expected outputs in the following format for each cell in the library:

```
Input1; Input2; Input3; Output1; Output2; /*pin order list*/
$
absolute-time1 input-stimulus1 #expected-outputs1
absolute-time2 input-stimulus2 #expected-outputs2
absolute-time3 input-stimulus3 #expected-outputs3
```

where `absolute-time` is expressed in `time_units`, as defined in the testbench file, and the characters used for `input-stimulus` and `expected-outputs` are in IEEE Std 1164.1 format.

The Verilog model generator creates one `.sen` file for an entire library when testbenches are written out. The `library_tb.v` file uses the contents of the `.sen` file as the test vectors. The `.sen` files that The Verilog model generator writes out contain the absolute time and input stimulus only. (You can add expected outputs and comments later.)

The pin list must begin with a blank line and terminate with a dollar sign (`$`). The stimulus vectors are then listed after the dollar sign with the time they are to be applied. The input and output pin list is optional. Testbench generation does not output a pin list. By default, the stimulus file lists pin names in the order they are specified in the `cell.v` file. You can define a different pin order in your individual cell testbenches by including the pin order at the beginning of the input stimulus file before the dollar sign.

- Script files

The Verilog model generator writes out the following script files:

- `library.dc.tcl`, a `dc_shell` script that generates an SDF file for each cell. The `dc_shell` script is as follows:

```
set link_library lib.db
read_verilog -netlist wrapper_library.v
set designs [get_designs *]
foreach_in_collection ds $designs {
    set d_name [get_object_name $ds]
    current_design $d_name
    write_sdf $d_name.sdf
}
```

The Verilog model generator automatically creates a `wrapper_library.v` file. The `dc_shell` script reads the wrapper file, which constructs one module, or design, for each cell. The modules are also packed into one `wrapper_library.v` file. You must create a `.db` file from the input `.lib` file before using the `dc_shell` script. The input `.lib` file generates a `lib.db` file during Verilog generation.

- `library.pt.tcl`, a `pt_shell` script that generates an SDF file for each cell. The `pt_shell` script is as follows:

```
set link_library library.db
set i 0
read_verilog wrapper_library.v
set designs [get_designs *]
foreach_in_collection ds $designs {
    set d_name [get_object_name $ds]
    if {$i==0} {
        set ld $d_name
    } else {
        lappend ld $d_name
    }
    incr i
}
foreach dl_name $ld {
    read_verilog wrapper_library_$dl_name.v
```

```

link_design $dl_name
write_sdf $dl_name.sdf [-include SETUPHOLD] [ -include RECREM]
}
quit

```

The Verilog model generator automatically creates a `wrapper_library.v` file. The `pt_shell` script reads the wrapper file, which constructs one module, or design, for each cell. Each cell also has a specific `wrapper_library_o_cell.v` file. You must create a `.db` file from the input `.lib` file before using the `pt_shell` script. The input `.lib` file generates a `.lib.db` file during Verilog generation.

If the `veriloglib_combine_timingcheck` variable is set to the `setuphold` value, `write_sdf` will include the `-include SETUPHOLD` value. If `veriloglib_combine_timingcheck` is set to `recrem`, `write_sdf` will include the `-include RECREM` value.

- `library .csh`, a C shell script that runs library verification.

The C shell script calls the VCS simulator to run testbench simulation. You can use the following options with the `library .csh` command:

```
library .csh -tb
```

The `-tb` option skips the Verilog model compilation step. If you have already run the `.csh` script once, the library has already been compiled. Using the `-tb` option allows you to skip this step and save time.

```
library .csh -sp
```

The `-sp` option omits the portion of the script that runs the testbench on the entire library. Using the `-sp` option allows you to skip the entire library check. The `-lib` option specifies whether a specific cell is tested. The script includes a cell list. If you want to test only a few cells, you can edit the script and remove the cells you do not want to test.

```
library .csh -lib
```

The `-lib` option omits the portion of the script that runs the testbenches on specific cells. Using the `-lib` option allows you to skip the check on individual cells.

During Verilog testbench generation, Library Compiler creates a `testbench` directory under the Verilog model directory and puts all generated files into that directory. The default directory is `library_verilog` under the current working directory. If a `testbench` directory already exists, the newly generated files overwrite the existing files in the directory and Library Compiler issues a warning message.

When the testbench compares the expected and actual output, an X (unknown) is equal only to another X if the `veriloglib_tb_x_eq_dontcare` variable is set to false, as shown:

```
set veriloglib_tb_x_eq_dontcare = false
```

If `veriloglib_tb_x_eq_dontcare` is set to true, an X is a “don’t care” value and is equal to any logic value. If you do not define `veriloglib_tb_x_eq_dontcare`, the default is false.

6

Transistor Mismatch Characterization

You can use transistor mismatch characterization to model the within-cell effects of transistor parameter variations. The resulting library cell models can be used with PrimeTime VX for accurate variation-aware analysis of transistor mismatch effects.

This chapter contains the following sections:

- [Overview of Transistor Mismatch](#)
- [How Mismatch Characterization is Performed](#)
- [Mismatch Characterization Options](#)
- [Sigma and Mean Support for Mismatch Analysis](#)
- [Defining the Model File to Control Mismatch Parameters](#)

Overview of Transistor Mismatch

A cell model in a library contains information about the timing, power, and noise characteristics of the cell at the cell level. The model does not contain any information about the circuitry or transistors within the cell. To model the effects of within-cell transistor mismatch, it is necessary to capture that information during characterization.

During characterization of a cell, the behavior of each transistor model is assumed by default to be uniform throughout the cell. Even for variation-aware characterization, variations in transistor parameters are assumed to be systematic and uniform throughout the cell.

Liberty NCX offers an option to characterize the effects of random (not uniform) variations between different transistors with the cell. This type of characterization produces a variation-aware model of the cell that takes into account the random mismatch between transistor parameters. This type of model more accurately predicts the delay, slew, and timing constraint characteristics when the cell has significant mismatch variation effects. This modeling capability is called transistor mismatch characterization.

To perform transistor mismatch characterization, you must first properly set up the cell netlists and template files to describe the dependency of transistor parameters on the variation parameters. The `set variation true` and `set mismatch true` commands in the Liberty NCX configuration file invoke transistor mismatch characterization.

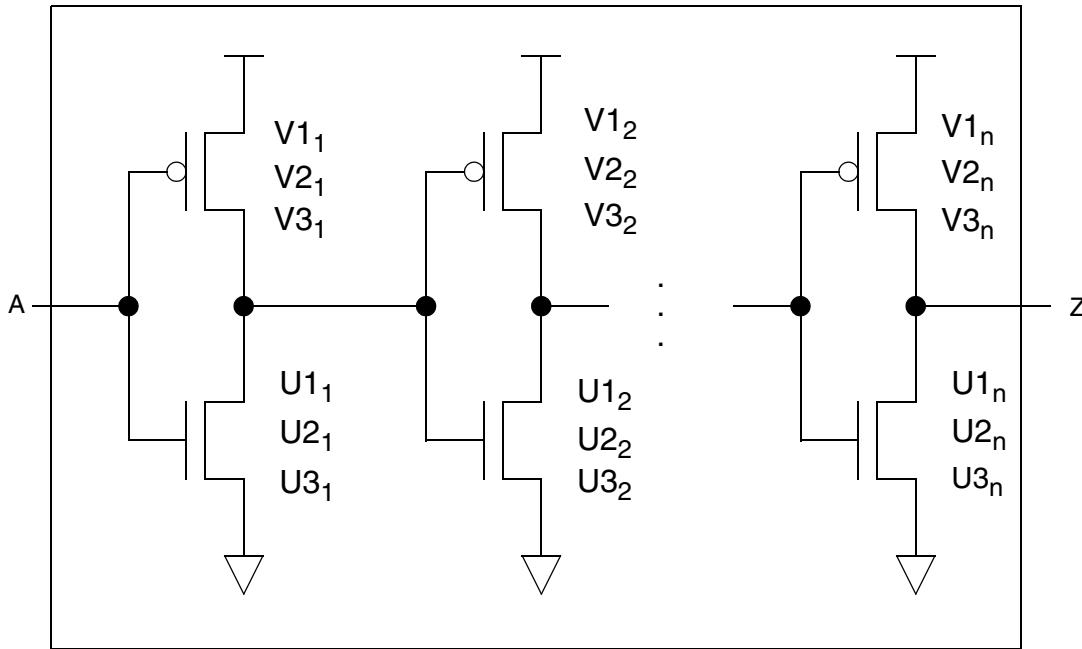
Characterization with transistor mismatch produces at least one new synthetic variation parameter to represent the mismatch variation, in addition to any other global variation parameters being characterized. The resulting library can be used for variation-aware analysis in PrimeTime VX.

Synthetic Variation Parameters

Liberty NCX creates one or more synthetic variation parameters as part of the transistor mismatch characterization process. A synthetic parameter reflects the aggregated effect of multiple mismatch variables applied to the transistors in the cell. The number of synthetic parameters created by Liberty NCX depends on your choice of parameter variation groupings and transistor type grouping. Synopsys recommends using a single synthetic variation parameter for mismatch modeling.

[Figure 6-1 on page 6-3](#) shows an example of the internal circuitry within a cell. The cell has n NMOS-PMOS transistor pairs.

Figure 6-1 Cell-Internal Transistors With Variation Mismatch

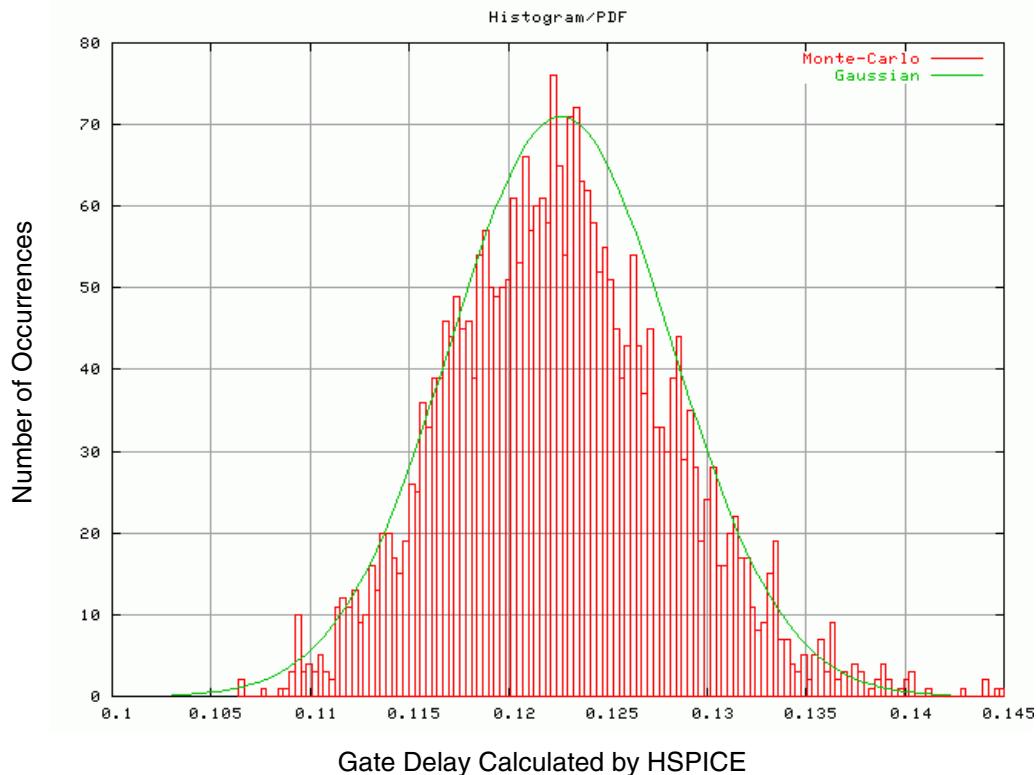


In this example, each transistor has three independent variables that represent process variations at the transistor level. The NMOS transistors have variables U₁, U₂, and U₃; and the PMOS transistors have variables V₁, V₂, and V₃. The timing characteristics of the transistors depend upon these variables. For each NMOS-PMOS pair, there are six variables that affect the stage timing. To the extent that the variables are independent, up to $6n$ variables affect the timing characteristics of the cell.

Each variable has a Gaussian distribution $N(m,s)$, where m is the nominal value and s is the standard deviation. Experiments using Monte Carlo analysis have shown that when the variables are varied independently, the impact on a given timing parameter (delay, slew, or timing constraint) resembles a Gaussian distribution.

For example, a gate circuit was simulated 3,000 times in a Monte Carlo analysis. Each variable of each transistor was assigned a randomly sampled value from the variable's distribution $N(m,s)$. The gate delay for an input-to-output transition was plotted in a histogram. The results are shown in [Figure 6-2 on page 6-4](#).

Figure 6-2 Cell Delay Histogram and Gaussian Curve



You can see that the delay distribution closely follows a Gaussian curve. The combined arc-delay effect of all mismatch variables on all transistors can be modeled as a single Gaussian variable $N(m,s(D))$, where m is the nominal value of the arc delay and $s(D)$ is the standard deviation of the arc delay. The results in this example show that the nominal delay value is 0.123, the delay value at $+s$ is 0.128, and the delay value at $-s$ is 0.118. Experiments have shown that slew and timing constraints (setup and hold) exhibit similar behavior.

Because the delay, slew, and timing constraints exhibit a Gaussian distribution, the effect of all transistor mismatch variables can be combined into a single synthetic random variable M . The synthetic variable can be considered a vector of all the original variables because it represents the overall mismatch behavior of all parameters in all transistors at the cell level.

How Mismatch Characterization is Performed

To find the Gaussian curve that matches the distribution results of random variations, Liberty NCX uses a proprietary technique to efficiently calculate the sensitivities of the delay and slew to the multiple transistors and their individual variation parameters. Experiments have shown very good agreement between the results obtained by this method and by Monte Carlo analysis.

To calculate the timing effects of transistor mismatch, you can use the characterization data from nominal and offset parameter values in a PrimeTime VX variation-aware analysis. You typically generate data at three values for each synthetic parameter variation: the nominal value, the $+s$ offset value, and the $-s$ offset value.

The nominal cell model is characterized by setting all variables to their nominal values. The $+s$ offset data represents the cell behavior with the synthetic variable M at nominal plus one s (or 1.5s, or 2s, or whatever quantile you specify). The $-s$ offset data represents the cell behavior with the synthetic variable M at nominal minus one s (or whatever quantile you specify). You specify the desired offset using `ncx_mm_sigma` in the configuration file. The default setting is 1.0. For example, to set the offset to 1.5s as follows:

```
set ncx_mm_sigma 1.5
```

Characterization at plus and minus one sigma away from nominal is recommended for typical within-cell transistor mismatch variation.

For transistor mismatch characterization, Liberty NCX follows the Liberty standard format and always generates the delay and slew information in CCS timing (not NLDM) format. A given Liberty NCX characterization run generates a single merged CCS library containing the nominal, $+s$, and $-s$ data.

You can perform characterization of both transistor mismatch variations and global variations (variations that apply uniformly across the whole design). Both types of variations should be characterized at the same time. Transistor mismatch characterization adds at least one synthetic variation parameter to the existing global variation parameters. For example, if you have two global parameters Len and Vt characterized at plus and minus 3s, and you also perform transistor mismatch characterization at plus and minus s, then you will have a total of three variation parameters, Len, Vt, and M. This results in $2N+1$ (seven) characterization points: nominal, $+3s$ Len, $-3s$ Len, $+3s$ Vt, $-3s$ Vt, $+s$ M, and $-s$ M.

Mismatch Characterization Options

To invoke transistor mismatch characterization in Liberty NCX, use the following commands in the configuration file:

```
set variation true  
set mismatch true
```

The library template file specifies the SPICE models that are varied for characterization, the number of synthetic parameters created, and the mismatch parameters that are varied for each synthetic parameter.

The following attributes specify the SPICE transistor model whose characteristics are modified as a function of the variation parameter values:

```
ncx_mm_pmos_model: pmos_model_name;  
ncx_mm_nmos_model: nmos_model_name;
```

For example, specify the following to vary the characteristics of PMOS transistor model “pch” and NMOS transistor “nch”:

```
ncx_mm_pmos_model: pch;  
ncx_mm_nmos_model: nch;
```

Liberty NCX then determines the cell characteristics in the presence of parameter mismatch between individual instances of “pch” and “nch” transistors in the cell.

The following template file attribute specifies the mismatch parameters to be analyzed when the parameters are the same for PMOS and NMOS transistors:

```
ncx_mm_parameter: var_list;
```

If the mismatch parameters for PMOS and NMOS transistors are different, use the following pair of attributes instead:

```
ncx_mm_parameter_pmos: p_var_list;  
ncx_mm_parameter_nmos: n_var_list;
```

For example,

```
ncx_mm_parameter_pmos: S_PVTH S_PXL;  
ncx_mm_parameter_nmos: S_NVTH S_NXL;
```

Liberty NCX uses these attributes to parameterize the individual mismatch variables in each transistor, thereby controlling the behavior of each transistor based on the mismatch parameter variations.

You can specify the synthetic variation-aware parameters created by mismatch characterization. By default, Liberty NCX creates a single synthetic parameter named MM to represent all mismatch variables for all transistors. You can optionally specify separate synthetic parameters to represent different mismatch variables or groups of mismatch variables. You can also specify separate synthetic parameters for PMOS and NMOS transistors. Synopsys recommends a single synthetic parameter, the default behavior of Liberty NCX.

The following template file attribute specifies the single synthetic variation-aware attribute created by mismatch characterization:

```
ncx_mm_va_parameter: "synth_va_param";
```

The specified name is used instead of the default (MM) for the generated library's va_parameters attribute, which is the name used to represent the transistor mismatch variation in a PrimeTime VX analysis. The specified name must be different from any global variation names.

To create multiple synthetic variation-aware parameters to represent different mismatch parameters or groups of such parameters, use the following syntax:

```
ncx_mm_va_parameter: "synth_va_param_1 param_list"
                     "synth_va_param_2 param_list"
                     ...
;
```

If the mismatch parameter names for PMOS and NMOS transistors have the same name, as in the following example,

```
ncx_mm_parameter_pmos: param1 param2 ;
ncx_mm_parameter_nmos: param1 param2 ;
```

Specify the synthetic parameters that aggregate their behavior in the published library, as follows:

```
ncx_mm_va_parameter_pmos: "MMp param1 param2" ;
ncx_mm_va_parameter_nmos: "MMn param1 param2" ;
```

It is recommended that you use unique mismatch parameter names for clarity.

Sigma and Mean Support for Mismatch Analysis

Liberty NCX provides the following attributes to allow you to specify the mean and sigma values for each mismatch parameter in the Liberty NCX configuration for characterizing a library for transistor mismatch effects. You must specify the attributes in the library template file.

- `ncx_mm_parameter_sigma`

Set the `ncx_mm_parameter_sigma` attribute, as shown, to specify a list of mismatch parameter names and sigma pairs:

```
ncx_mm_parameter_pmos : SUBC_RANDOM_PVTH SUBC_RANDOM_PXL ;
ncx_mm_parameter_nmos : SUBC_RANDOM_NVTH SUBC_RANDOM_NXL ;
ncx_mm_parameter_sigma : SUBC_RANDOM_PVTH 0.33 SUBC_RANDOM_NXL 0.25 ;
```

The default is 1.0 if a sigma value is not specified for the mismatch parameter.

- `ncx_mm_parameter_mean`

Set the `ncx_mm_parameter_mean` attribute, as shown, to specify a list of mismatch parameter names and mean value pairs.

```
ncx_mm_parameter_pmos : SUBC_RANDOM_PVTH SUBC_RANDOM_PXL ;
ncx_mm_parameter_nmos : SUBC_RANDOM_NVTH SUBC_RANDOM_NXL ;
ncx_mm_parameter_mean : SUBC_RANDOM_PVTH 0.5 SUBC_RANDOM_NXL 0.2 ;
```

The default is 0.0 if a mean value is not specified for the mismatch parameter.

- `ncx_mm_sigma`

Set the `ncx_mm_sigma` attribute to specify the standard deviation of the aggregate mismatch parameter that is published in the library. The default is 1.0.

Defining the Model File to Control Mismatch Parameters

You must define the SPICE model file to include mismatch parameters in the subcircuit instantiation line so Liberty NCX can control the mismatch parameter values for mismatch characterization. You do not need to change the model file if the model does not employ a `.subckt` definition and the mismatch parameters appear directly in the netlist, as shown in the following example:

```
:
MP0 VDD DATA net40 VDD P w=0.48u ad=0.0537p as=0.0528p
+     l='0.06u+2.600n*PMM1'
+     delvt0='-19.00m*PMM2'
+     pd=1.125u ps=1.18u nrd=0.233073 nrs=0.104167 sa=165n sb=210.321n
:
```

In the example, the mismatch parameters are `PMM1` and `PMM2`. Liberty NCX is able to directly control the mismatch parameters in the netlist. If the mismatch parameters do not appear in the netlist, and they are included in the model definition only, the model interface statement must include the mismatch parameters as interface parameters. For example, if the mismatch parameters for a PMOS device model are `PMM1` and `PMM2`, the model interface must include the following information:

```
.SUBCKT P D G S B W=0 L=0 MI=1 AD=1 AS=1 PD=1 PS=1 PMM1=0 PMM2=0
```

This allows Liberty NCX to vary the parameters in the mismatch netlist, as follows:

```
:  
XM3 n1 n2 n7 vbb P ad=0.012p as=0.021p l=0.04u pd=0.331u ps=0.597u  
w=0.18u  
+ PMM1=mmp17_0_0  
+ PMM2=mmp17_0_1  
:
```

Without this control, Liberty NCX cannot perform mismatch characterization for a library.

7

CCS Noise Characterization

Liberty NCX can characterize library cells for noise response and create CCS Noise modeling information. A library with CCS Noise models can be used with PrimeTime SI and other tools to check for timing and logic failures caused by noise.

This chapter contains the following sections:

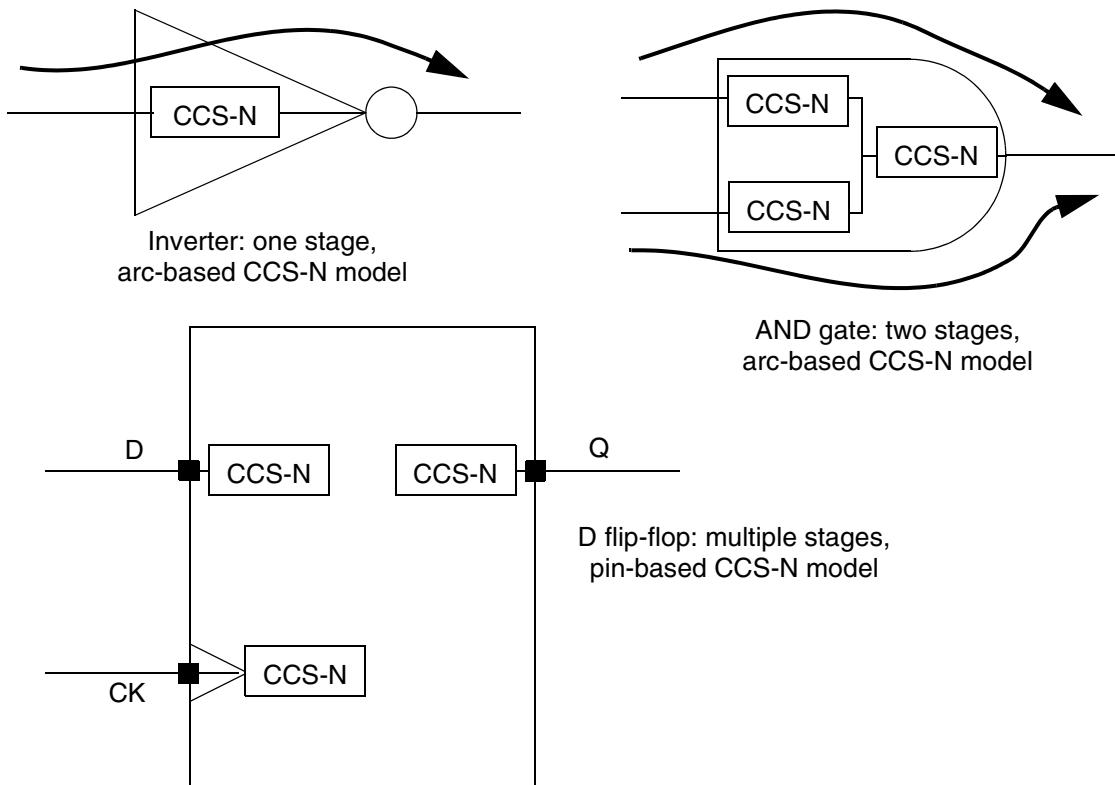
- [Overview of CCS Noise Characterization](#)
- [Requirements for Noise Characterization](#)
- [CCS Noise Characterization Setup](#)

Overview of CCS Noise Characterization

Liberty NCX performs CCS Noise characterization when you set the `noise` parameter to `true` in the configuration file. In that case, Liberty NCX creates CCS Noise modeling information for the technology library. CCS Noise models enable PrimeTime SI and other tools to accurately analyze the delay and logic effects of crosstalk noise.

To characterize a library cell for noise, Liberty NCX builds a model of the cell using SPICE circuit elements and simulates the effects of different slews and noise bumps at the cell inputs. Based on the simulation results, Liberty NCX generates a set of equivalent time-dependent current-source models that represent the behavior at the cell outputs. PrimeTime SI and other tools use these models to accurately predict the response of the cell to noise.

To start characterization for noise, Liberty NCX examines the circuit netlist for the cell. It breaks down the netlist into channel-connected blocks. Each channel-connected block has a potential conducting path through transistor sources and drains. Liberty NCX cuts the netlist into separate stages at channel-connected block boundaries. A simple cell such as an inverter has only a single stage from input to output. An AND gate has two stages. A type D flip-flop has more than two stages. See [Figure 7-1](#).

Figure 7-1 CCS Noise Model Representations

A CCS noise model represents the noise response of each stage. A single-stage or two-stage cell results in an arc-based noise model. A cell with more than two stages results in a pin-based model.

To characterize a stage of a cell, Liberty NCX first simulates the DC response of the stage netlist. It varies the input voltage and measures the output voltage to generate a simple two-dimensional table representing the output current as a function of DC input and output voltages.

Then, to determine the noise response of the stage, it simulates the circuit by using a few different ramps and glitches at the stage input. It indirectly uses the output transition and noise propagation results to derive the dynamic behavior of the stage. The behavior of each stage is then used to make the CCS Noise model for the whole cell.

Requirements for Noise Characterization

The following requirements apply to noise characterization performed by Liberty NCX.

- Each cell in the .lib library must have at least one timing arc to each output port and bidirectional port.
- There must be HSPICE netlists and models for the cells.
- Each flat SPICE netlist must have fewer than 200,000 transistors. (This limit does not apply to hierarchical netlists.)
- Liberty NCX requires a Perl interpreter, 64-bit compiled version 5.8.3 or later.
- The following conditions apply to transistor netlists:
 - Two layers of transistors around boundary must be logic that maps to standard cells. Noise modeling requires clear identification of channel-connected regions at the block boundary.
 - The following types of structures are generally supported: static combinational cells, three-state buffers, buffered latches, flip-flops, and clock-gating cells.
 - The following types of structures are generally not supported: analog outputs, voltage-controlled oscillator inputs, phase-locked loops if the charge pump is exposed, blocks with dynamic body bias, and feedthroughs.

The following requirements apply to noise characterization of memory devices.

- Two common three-state diver structures are supported.
- Feedthrough and gated feedthrough are not supported. These features are not common for standalone memory blocks.
- I/O must be buffered.
- Analog I/O is not supported.

CCS Noise Characterization Setup

Before characterizing cells for noise, you need to prepare the configuration file and template files. To invoke CCS Noise characterization, set the `noise` parameter to `true` in the configuration file as in the following example:

```
set cells {all}
set netlist_dir myhome/netlists/proj1
set netlist_file sml.net
set model_file model.typ
set simulator hspice
set input_library my_library_1.lib
set output_library my_library_2.lib
set noise true
...
```

The following template file attributes control several CCS Noise characterization options:

```
ncx_noise_dc_max : float
ncx_noise_dc_min : float
ncx_noise_max_fanout : integer
ncx_noise_nmos_models : "model_name1 model_name2 ..."
ncx_noise_pmos_models : "model_name1 model_name2 ..."
ncx_noise_normalized_dc_indexes : "index1 index2 ..."
ncx_noise_skip_pin_list : "pin1 pin2 ..."
ncx_noise_three_state_pulldn_res : float
ncx_noise_three_state_pullup_res : float
```

Table 7-1 lists and briefly describes the template file attributes that control CCS Noise characterization.

Table 7-1 CCS Noise Template File Attributes

Template file attribute	Description
ncx_noise_dc_max : float	Specifies the DC table range in the scale of VDD. The value is any floating-point number from 1.5 to 2. The default value is 2.
ncx_noise_dc_min : float	Specifies the DC table range in the scale of VDD. The value is any floating-point number from -0.5 to -1.0. The default value is -1.0.
ncx_noise_max_fanout : integer	Specifies the maximum fanout from a channel-connected block being cut from netlist of a cell. The default is 6.
ncx_noise_nmos_models "model_name1 model_name2 ..."	Specifies the channel model names of NMOS transistors. You can specify one or more model names separated by a space character and enclosed in quotation marks.
ncx_noise_pmos_models "model_name1 model_name2 ..."	Specifies the channel model names of PMOS transistors. You can specify one or more model names separated by a space character and enclosed in quotation marks.
ncx_noise_normalized_dc_indexes "index1 index2 ..."	Specifies the normalized DC table indexes for the noise model. The indexes in the library DC table are scaled according to the actual voltage swing at the channel-connected block input and output. The index values must be monotonically increasing and must contain the values 0.0 and 1.0. The default list of indexes is: -1.0 -0.5 -0.2 -0.1 0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0 1.1 1.2 1.5 2.0

Table 7-1 CCS Noise Template File Attributes (Continued)

Template file attribute	Description
<code>ncx_noise_skip_pin_list : "pin1 pin2 ..."</code>	Skips noise characterization for the specified pins. Use one or more spaces between each pin in the list. The CCS noise model defines each skipped pin with the attribute <code>is_needed: false</code> in the Liberty CCS Noise model.
<code>ncx_noise_three_state_pulldn_res : float</code>	Specifies the pulldown resistor value of the three-state enable arc in noise characterization. The default is <code>1e6</code> .
<code>ncx_noise_three_state_pulley_res : float</code>	Specifies the pullup resistor value of the three-state enable arc in noise characterization. The default is <code>1e6</code> .

Liberty NCX typically can determine whether a transistor model is NMOS or PMOS. However, in case it cannot do this, you can explicitly specify the model types by using the `ncx_noise_nmos_models` and `ncx_noise_pmos_models` template file attributes.

The arc states used for noise characterization must be the same as those used for timing characterization. If you select all states for timing characterization, then you must also use all states for noise characterization.

8

Troubleshooting

This chapter contains the following sections:

- Troubleshooting
- Frequently Asked Questions

Troubleshooting

The main repository of all problem reports, errors, and warnings is the log file, which is the starting point of any troubleshooting effort. Using the information contained in the log file, you can debug errors in the simulation database, input templates, command-line options, and so on. As such, an intimate knowledge of the information contained in the log file is essential to an efficient debug effort. The contents of the log file are explained in “[Liberty NCX Operation and Log File](#)” on page 2-7.

A general approach to troubleshooting characterization problems in Liberty NCX includes the following steps:

1. Check program summary in log file for failing cells.
2. Locate associated error messages for each failing cell or arc in the specified stage of the log file.
3. Apply any appropriate correction to program arguments.
4. Implement any cell property overrides in the cell template. In the working directory, Liberty NCX generates a cell template for each failing cell and a library template that is set up to run only the failing cells.
5. If appropriate, look in the specified arc simulation directory and check the reason for the simulation error. Liberty NCX saves the problem waveforms and generates a troubleshooting circuit for failing model index values. You can manually modify the SPICE netlist to test possible options that can result in success, and you can run the netlist locally to establish the necessary options. Add necessary simulation options to the library/cell template file, if appropriate.
6. Ensure that an incremental library template and the modified cell templates are in the template directory.
7. Rerun Liberty NCX, specifying the output library from the previous run as the input library for the new run. This will preserve the results for the cells that were successfully characterized.

When a cell fails characterization in Liberty NCX:

- The failed cell is not written to the `output_library`, but all other successful cells are written to the `output_library`.
- The library and cell templates are written to the `work_dir` and are set up to rerun only the failed cell.
- The entire cell simulation subdirectory is retained in the `simulation_dir`, and the successful results can be reused.

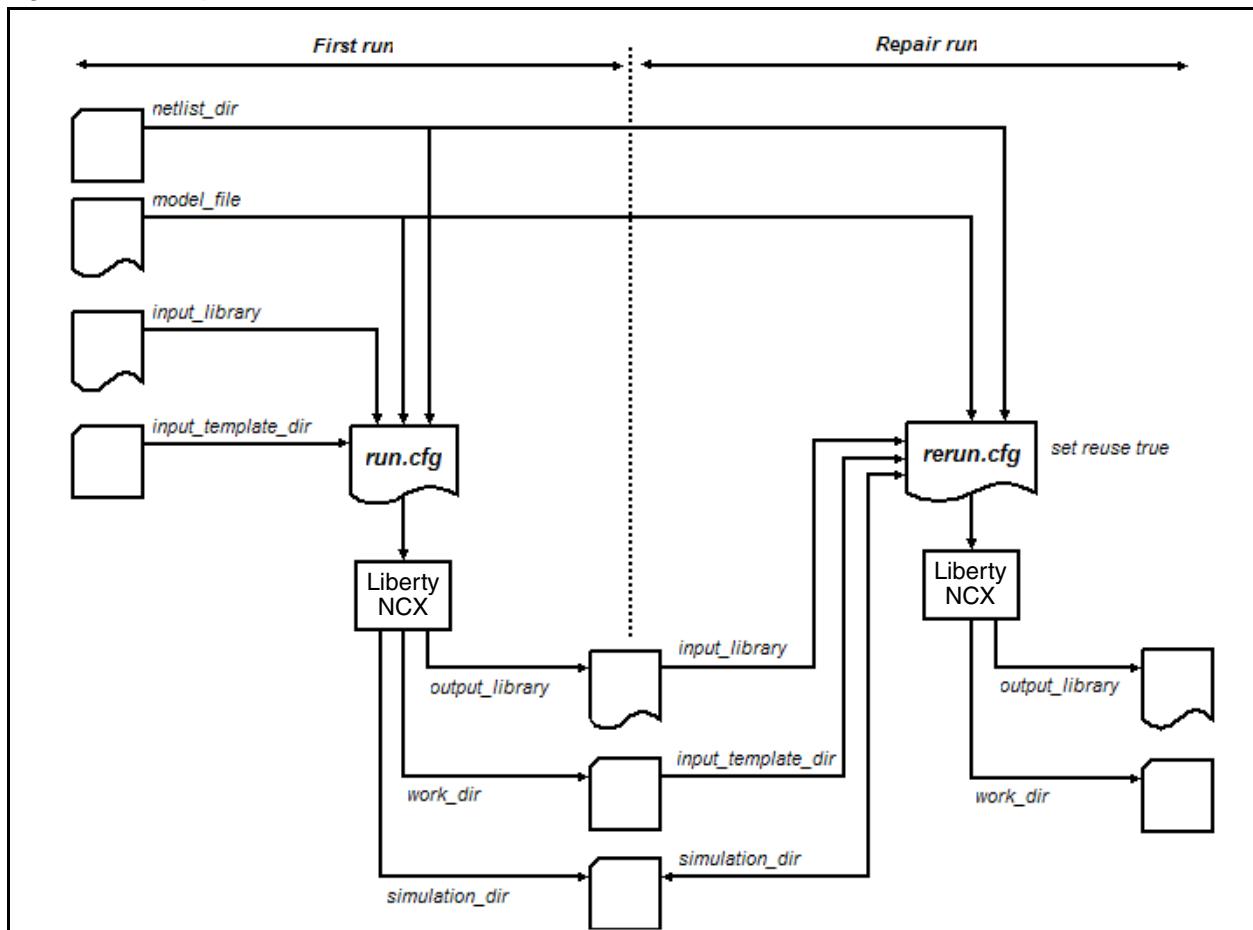
To run the repair flow as shown in [Figure 8-1](#):

1. Specify the output library from the original run as the input library for the repair run in order to retain successful results.
2. Specify the same simulation directory as the original run to reuse the successful results.
3. Specify the work directory of the original run as the input template directory, in order to use the repair templates that are written out.
4. Set the `reuse` option to true.

Note:

Make sure that the `only_active_cells_to_library` variable is set to false.

Figure 8-1 Repair Flow Run



Frequently Asked Questions

Here are the answers to some frequently asked questions.

- On what platforms does Liberty NCX run?

It runs on 32-bit and 64-bit Linux and Solaris platforms.

- What simulators does Liberty NCX support?

Liberty NCX supports HSPICE, Eldo, and Spectre.

- What processes and models does Liberty NCX support?

It supports all processes and device models that are supported by HSPICE.

- Is Liberty NCX case-sensitive?

It supports only the HSPICE simulator, which is not case-sensitive.

- How can I generate a CCS-only library using Liberty NCX?

Use `set nldm false` in the configuration file. Timing constraint models, which are common to both NLDM and CCS model types, are acquired if either CCS or NLDM models are specified.

- Can I generate only NLDM data using Liberty NCX?

Yes. Use `set timing true`, `set ccs_timing false`, and `set nldm true` in the configuration file.

- How do I add a new cell to an existing library?

Create a library template for the input library and add the name of the new cell to the `do` list. See “[Characterization Attributes](#)” on page 3-3. Ensure that a template for the new cell exists in the template directory.

- How can I generate only a selected set of cells?

Use a library template that contains a `do` list.

- How can I skip characterizing certain cells in the library?

Use a library template that contains a `dont` list.

- Can I specify a compressed library as an input argument?

Yes. The file name should end with `.gz` or `.z`.

- Must I specify an input library?

No. If an input library is not specified, Liberty NCX creates a new library. An output library name must be specified.

- Must I specify an output library?
No, but if no output library name is specified, no output library is written.
- Can I run Liberty NCX just to check the sensitization of cells in my library?
Yes. Disable simulation by setting `timing`, `power`, and `noise` to `false` in the configuration file.
- How can I specify a different location for the simulation database?
Use `set simulation_dir` in the configuration file.
- Does Liberty NCX generate templates automatically?
If any cells fail the characterization process, Liberty NCX generates a template for the library and for each of the failing cells. Otherwise, Liberty NCX generates templates only if you use `set templates true` in the configuration file.
- How can I create a set of templates for an existing library?
Use `set templates true` in the configuration file. The templates are written to the working directory or to the directory specified by the `output_template_dir` setting in the configuration file.
- Where should I place my input templates for Liberty NCX?
Templates intended for input to Liberty NCX must be placed in the directory specified by the `input_template_dir` setting in the configuration file, or in the current working directory if this option is not used.
- How can I override delay and slew thresholds in the input library?
Create a library or cell template for input to Liberty NCX, as appropriate, and enter the new value as described in the section "[Characterization Attributes](#)" on page 3-3.
- How do I run Liberty NCX on a local machine?
Use `set farm_type NoFarm` in the configuration file.
- Does Liberty NCX support LSF and Sun Global Resource Director (GRD)?
Yes. Sun GRD support was added in the Z-2007.06-SP1 release.
- Can I run Liberty NCX on a set of specified machines?
Liberty NCX currently does not support execution on a set of named machines.
- How can I limit the number of machines to be used on the compute farm?
Use `set max_jobs` in the configuration file.

A

Generic Simulator Interface

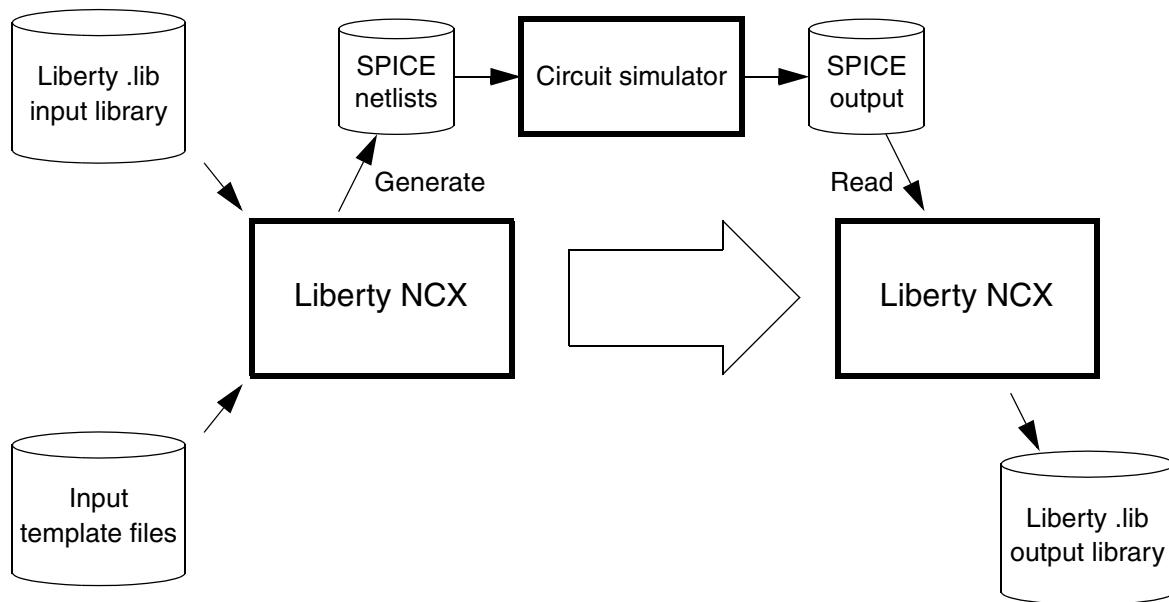
This appendix describes the generic simulator interface in the following sections:

- [Overview](#)
- [Simulator Interface Components](#)
- [Invoking the Third-Party Simulator Interface](#)

Overview

Liberty NCX uses a circuit simulator such as HSPICE to generate timing, power, and noise data used for modeling the behavior of library cells. Liberty NCX provides input to the simulator in the form of ASCII-format SPICE netlists. After simulation execution has been completed, Liberty NCX reads the simulator output, collates the results, and writes the characterization data to a library in Liberty format. The interaction between Liberty NCX and the simulators is summarized in [Figure A-1](#):

Figure A-1 Data Flow Between Liberty NCX and Circuit Simulator



There are two points of interaction between Liberty NCX and the simulator:

- Netlist generation – Liberty NCX generates the ASCII-format input files for the circuit simulator.
- Model reading – Liberty NCX translates the simulator output into an internal format that it can understand and use to write the characterization data.

Liberty NCX directly supports the use of certain circuit simulators such as HSPICE, Eldo, and Spectre. However, you also have the option to program a custom interface to any circuit simulator. The custom simulator interface, which is implemented in the Perl programming language, provides a translation layer between Liberty NCX and the external simulator.

The programmable third-party simulator interface offers the following benefits:

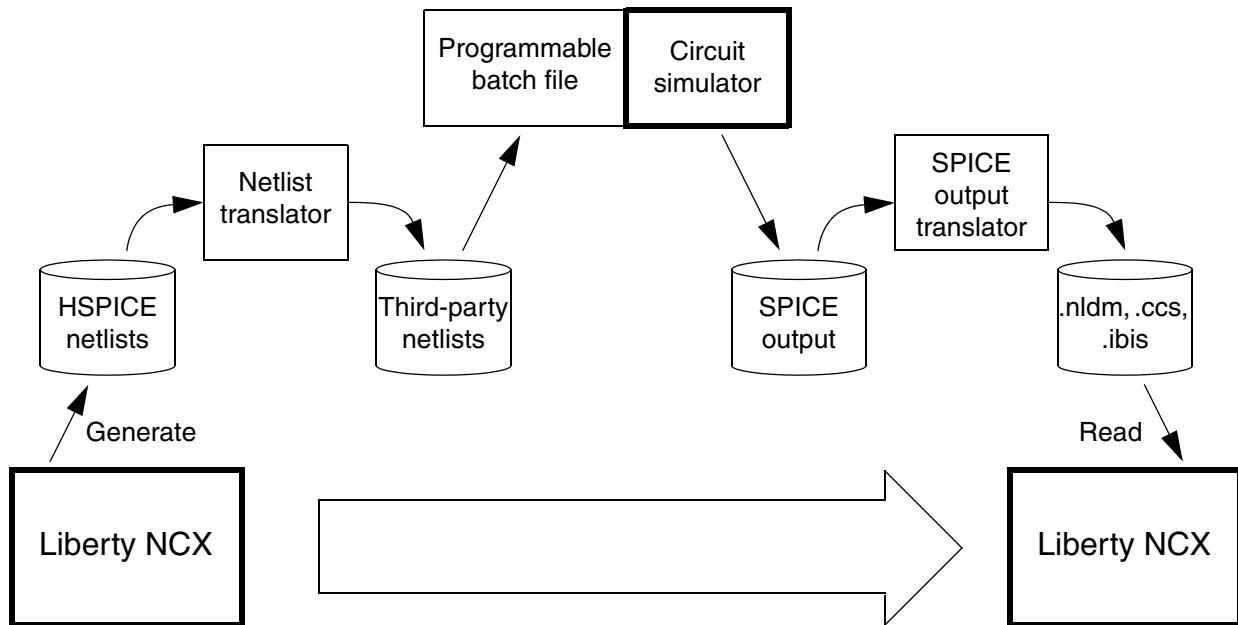
- Allows the use of proprietary simulators
- Allows customization of simulator interfaces to match your requirements or preferences
- Allows some simulator issues to be quickly resolved by editing the programmable interface, without requiring an updated release of the Liberty NCX executable

The programmable interface intercepts each HSPICE netlist generated by Liberty NCX and translates it to the desired format, and it intercepts the simulator output and translates it into a format accepted by Liberty NCX.

Simulator Interface Components

The simulator interface consists of the three components: a netlist translator, a programmable batch file, and a SPICE output translator. The interaction between these components is shown in [Figure A-2](#).

Figure A-2 Simulator Interface Components



The simulator interface components are ASCII files stored in the Liberty NCX installation directory as follows:

- Netlist translator:

```
installation_dir/ncx/utils/third_party/translateHSPICE and  
installation_dir/ncx/utils/third_party/simulator_type.pm
```

- Programmable batch file:

```
installation_dir/ncx/utils/third_party/genericSimulator.pl
```

- SPICE output translator:

```
installation_dir/ncx/utils/third_party/simulator_type_extract.pm
```

where *installation_dir* is the directory in which you installed Liberty NCX and *simulator_name* is the name of the simulator specified by the *simulator_type* configuration file setting. All of these files are user-programmable, although the *translateHSPICE* file is typically left unchanged.

Instead of using the *installation_dir/ncx/utils/third_party* directory to store and maintain the simulator interface files, you can copy those files to a different directory and specify that directory path using the *spice_netlist_translator* setting in the configuration file. However, if you choose to do this, you must keep all the simulator interface files in that directory. The directory you use can be kept under version control, if desired.

[Table A-1](#) lists and describes the contents of a typical third-party simulator interface directory (by default at *installation_dir/ncx/utils/third_party*).

Table A-1 Third-Party Simulator Interface Directory Contents

Directory element	Description
docs	A directory containing documentation about the process of adding a third-party translator. The <i>index.html</i> file under this directory is a good place to get started when you add a new simulator.
example.pm	A Perl package <i>example</i> containing a list of subroutines to be written in order to support a third-party translator.
example_extract.pm	A Perl package <i>example</i> containing a list of subroutines to be written in order to support the extraction portion of a third-party translator.
genericSimulator.pl	The generic batch file interface, a Perl executable. This is required because different simulators have different user interfaces. For example, input files are specified with <i>-i</i> under HSPICE but no keyword is used for Spectre. This behavior must be programmed accordingly.

Table A-1 Third-Party Simulator Interface Directory Contents (Continued)

Directory element	Description
hspiceElem.pm	An internal Perl package intended to store HSPICE elements after they have been parsed.
hspiceParse.pm	An internal Perl package that parses the HSPICE netlist file and stores the contents in hspiceElem.pm.
spectre_extract.pm	A Perl package containing example code for an extractor for the Spectre simulator.
spectre.pm	A Perl package containing example code for the Spectre translator.
translateHSPICE	The wrapper script that executes the translation operation, a Perl executable.

Invoking the Third-Party Simulator Interface

The third-party simulator interface is disabled by default. To enable customization of the interface, use the following commands in the Liberty NCX configuration file:

```
set enable_generic_simulator_interface true
set spice_netlist_translator path_to_translator/translateHSPICE
set simulator_output_extractor true
set model_extraction_to_farm true
```

The latter two options are requirements based on how the SPICE output extractor interacts with Liberty NCX. Using these two options greatly improves the robustness of this portion of the translator.

For usage information, open the following web page in an HTML web browser:

installation_dir/ncx/utils/third_party/docs/

B

CCS Models

Deep submicron designs (90 nm and below) can be difficult to model accurately due to effects such as faster clocks, complex interconnect impedance, and nonlinear device behavior. Composite Current Source (CSS) models to address these issues. CCS models for device driver and receiver elements are highly accurate models for the simulation of these designs.

This chapter contains the following sections:

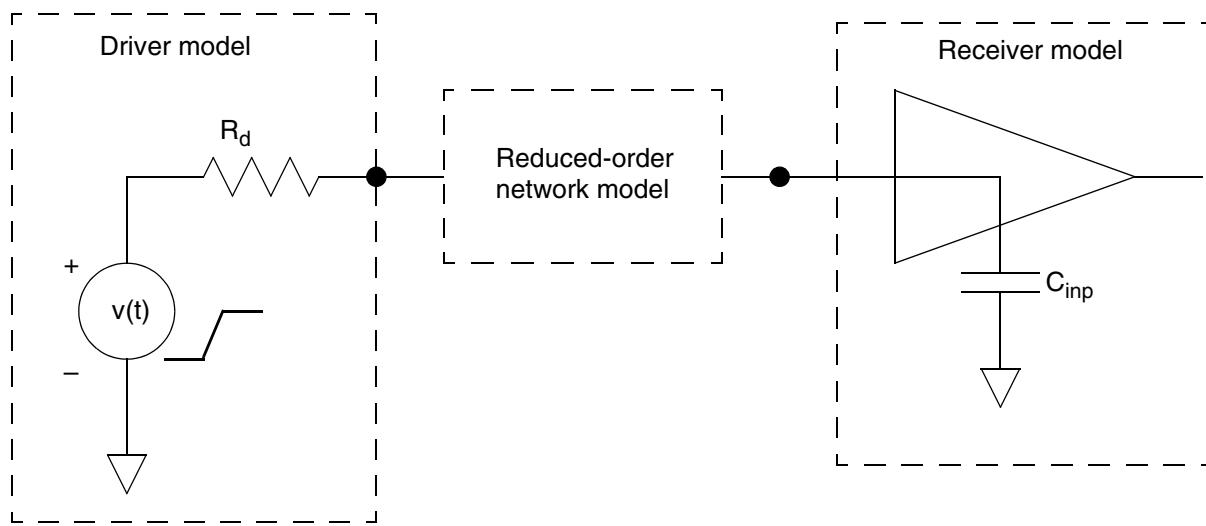
- [Driver Models](#)
- [Receiver Models](#)

Driver Models

Conventional driver models use either a Thevenin voltage source model (a time-dependent voltage source in series with an output resistance) or a Norton current source model (a time-dependent current source in parallel with an output resistance). The output drive resistors are used to analyze timing-arc sensitivity to output capacitance. The time-dependent sources are used to define the driver output waveform shape.

The nonlinear delay model (NLDM) uses a Thevenin voltage source to model the driver. [Figure B-1](#) shows how this model is used to calculate the delay and slew between a driver and receiver. Although this model has been effective and accurate for most submicron designs (1.3 microns and above), significant limitations to the model are observed when the driver output interconnect impedance becomes much greater than that of the drive resistor.

Figure B-1 NLDM Driver/Receiver Model



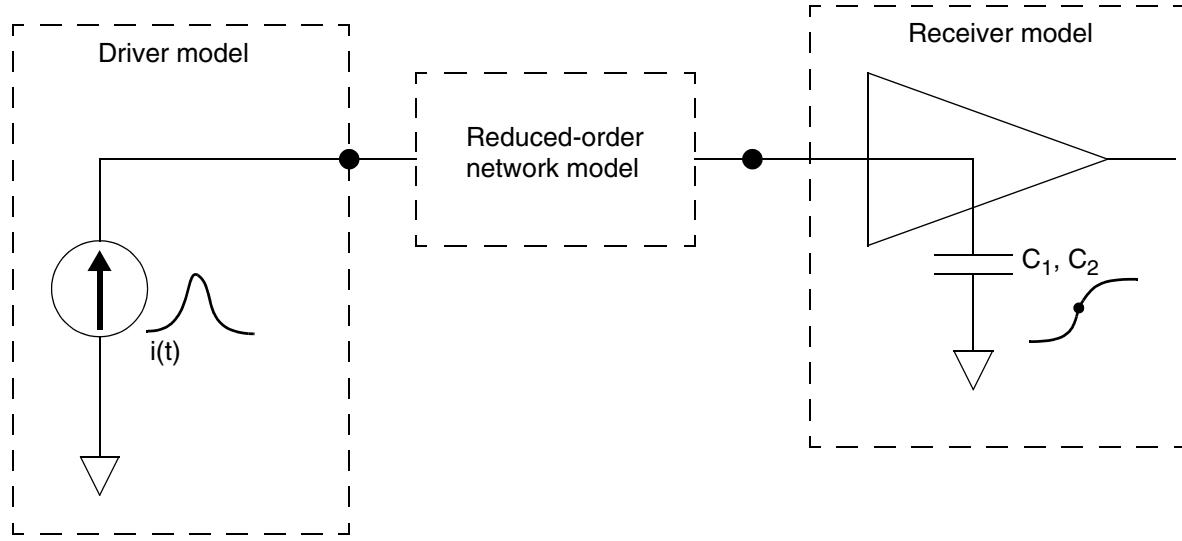
The output voltage can be determined by the following equation:

$$V_{out} = V_{in} \times (Z_{net} / (Z_{net} + R_d))$$

As $Z_{net} > R_d$, V_{out} approaches V_{in} . This results in an incorrect representation of driver behavior.

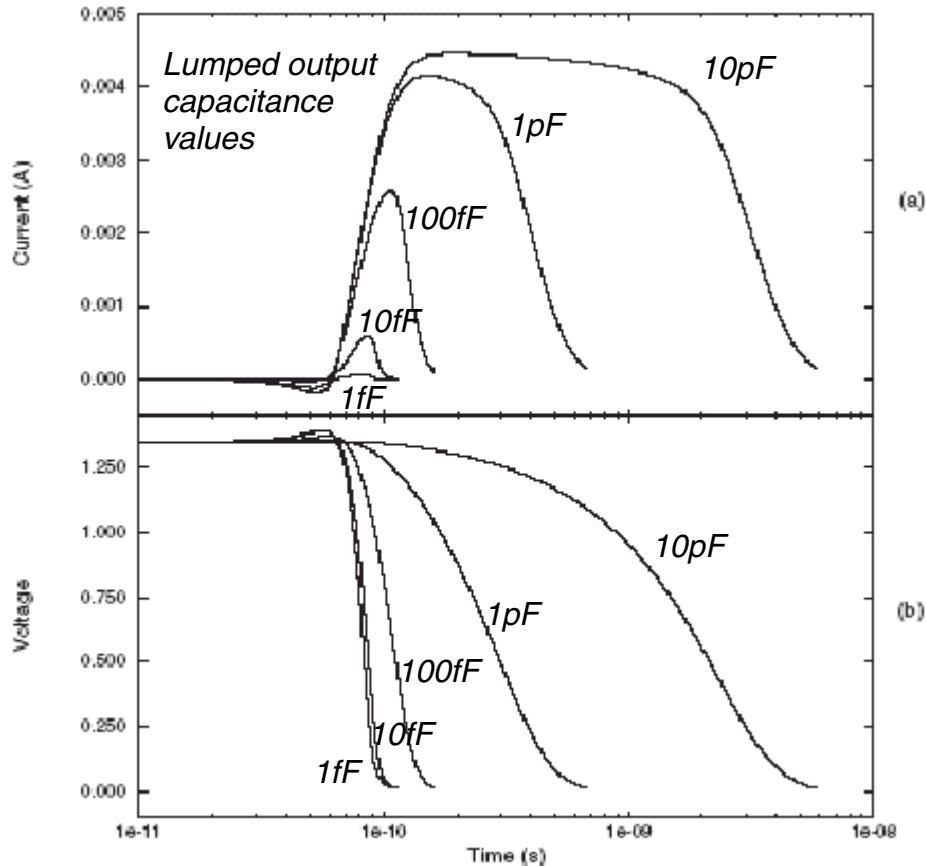
The CCS driver model shown in [Figure B-2](#) uses a time-dependent current source with an essentially infinite drive resistance; hence it does not suffer model inaccuracies introduced when $Z_{\text{net}} \gg R_d$. It achieves high accuracy by not modeling the transistor behavior at all, but instead mapping the arbitrary transistor behavior for lumped loads to that for an arbitrary detailed parasitic network.

Figure B-2 CCS Driver/Receiver Model



The mapping algorithm works in the following manner. Consider a set of characterization measurements of the output current as a function of time for a specific input slew and a set of output capacitance values, as shown in [Figure B-3 on page B-4](#). If these currents are applied to their respective capacitors, the voltage waveforms can be reconstructed. Drive situations that have output capacitance values different from those characterized can have their resulting waveforms derived by interpolation between the characterized currents. Similarly, if a driver has an input slew that was not characterized, the output waveform can be interpolated.

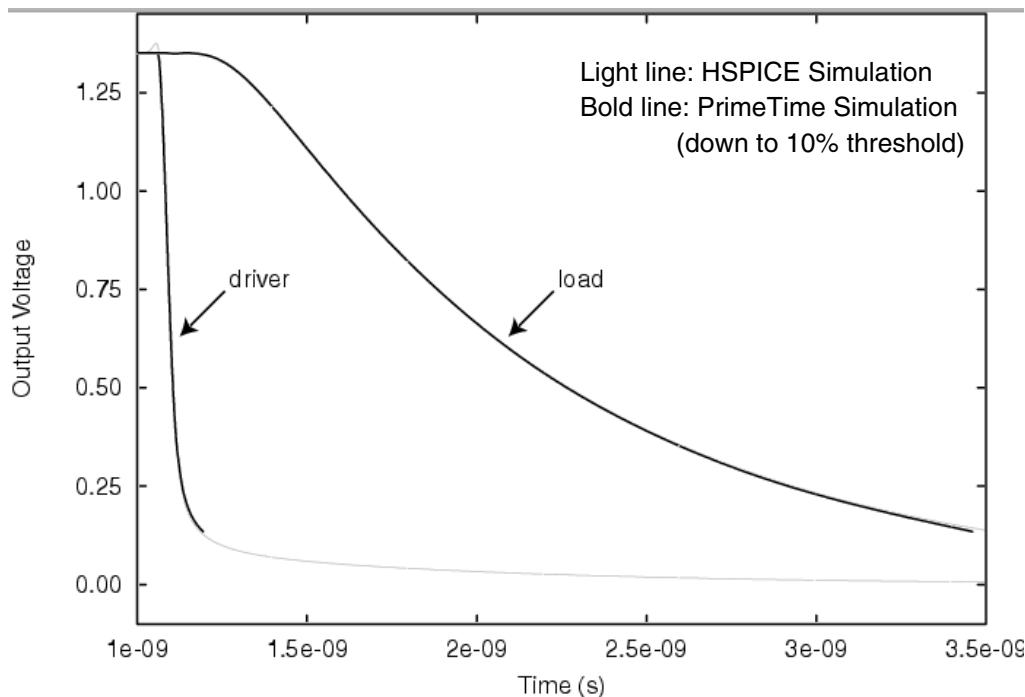
Figure B-3 Output Current and Voltage Versus Time and Load



Now consider driving a detailed parasitic network. At a given time step, the output currents can be applied from the characterization to the network. There is a unique current that produces the same voltage on both a lumped capacitance and the network at the given time step. This current is the chosen value for the given time step. This procedure can be reapplied at every subsequent time step. This process can be expressed by the following term:

$$I_{\{out\}}(V_{\{out\}})(t)$$

An example that illustrates the accuracy attainable with the CCS driver model is shown in [Figure B-4 on page B-5](#). The comparison is between transistors connected to detailed parasitics and the CCS model connected to an Arnoldi reduction of the detailed parasitics.

Figure B-4 CCS Model Accuracy

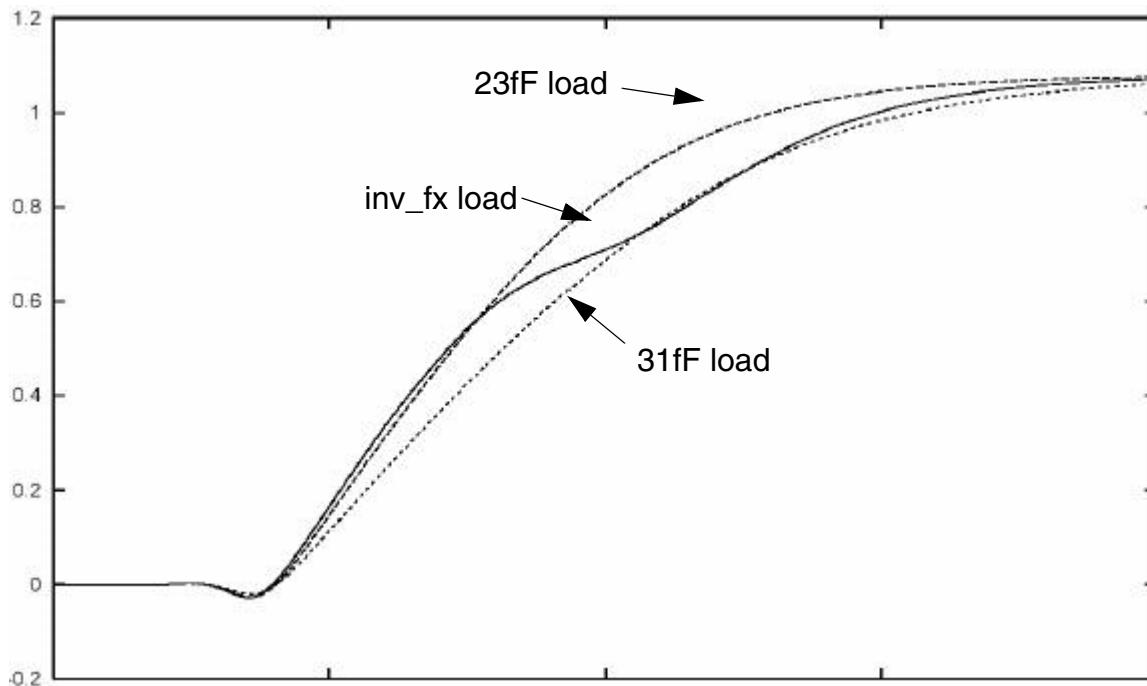
Receiver Models

The conventional gate-level approach of modeling receiver capacitance as a single lumped value has delivered acceptable accuracy for process geometries of 1.3 microns and larger. However, actual transistor simulations at 90 nm and below show that the capacitance at the receiver varies significantly during a receiver cell state transition. A single capacitance value or a min/max rise/fall lumped capacitance approach is no longer sufficient for accurate delay and receiver slew modeling.

The effect of Miller capacitance on the actual switching waveform is illustrated in [Figure B-5 on page B-6](#). Using a single lumped capacitance value (dashed lines), either the lower or upper portion of the waveform can be matched, but not both. The result is that the slew is inaccurate compared to the actual cell load (solid line). It is desirable to match both the upper and lower portions of the switching waveform, so that the measured delay will be accurate, but this is not possible using a single capacitance value.

Figure B-5 Single-Capacitance Model Versus Actual Switching Waveform

Miller Effect. 1 Capacitor model, Sinp = 10p



The CCS receiver model greatly improves the receiver model accuracy. In this approach, the input capacitance of a receiver is dynamically adjusted during the transition using two capacitance values, one before and one after the signal reaches the delay threshold (typically 50 percent of the rail voltage). The capacitance values are acquired during characterization and are based on voltage, input slew, output load, and transition of the cell. A single lumped value is replaced with two capacitance values expressed in the library as tables of input slew and output load.

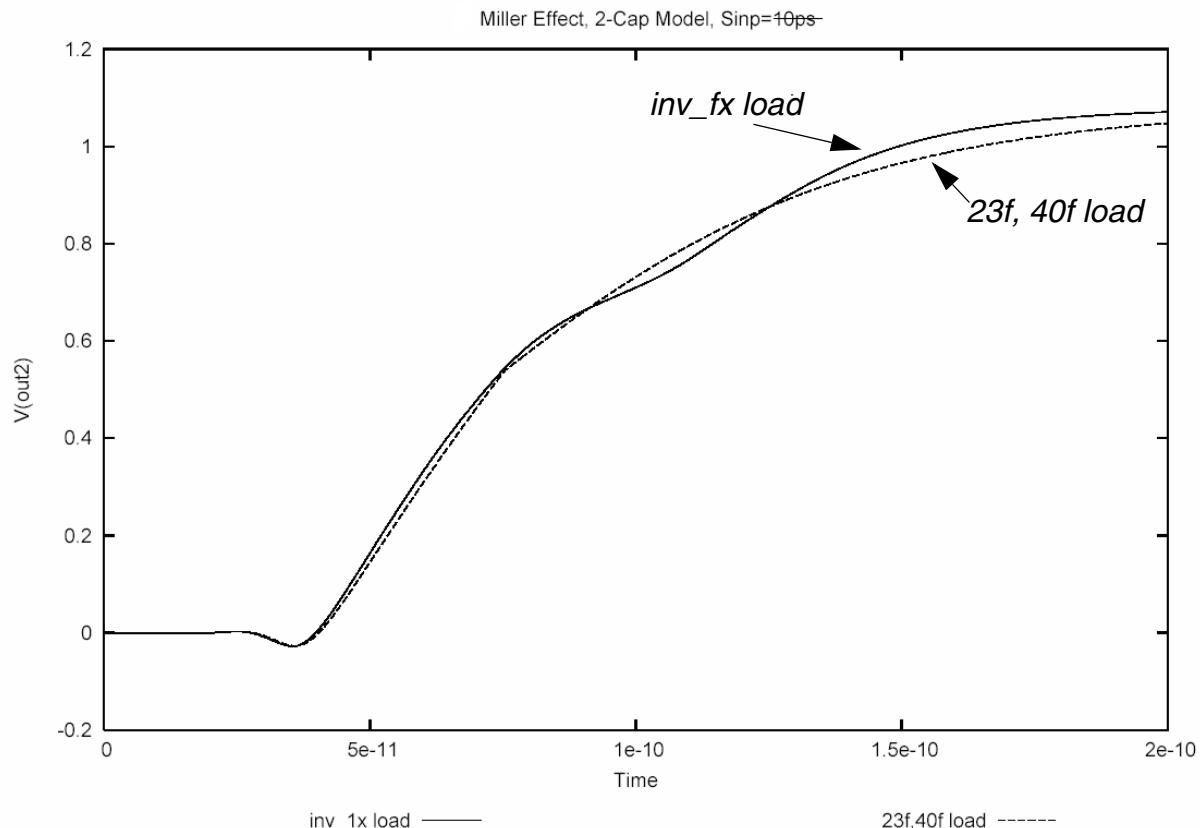
$$C1 = f(Slew_{in}, C_{out})$$

$$C2 = f(Slew_{in}, C_{out})$$

There are separate values for rise and fall. They can be specified on a library arc or on a library pin. The library pin version does not depend on output capacitance and is useful when there are no forward timing arcs (such as at the D pin of a flip-flop). The receiver model dynamically adjusts the capacitance input values during the transition, switching at the delay threshold. The CCS receiver model must be used in conjunction with the CCS driver model.

The result is a better match in slew as well as delay to the actual switching waveform. The same switching waveform with the two-capacitance receiver model approach is shown in [Figure B-6](#).

Figure B-6 CCS Receiver Model Versus Actual Switching Waveform



[Figure B-7 on page B-8](#) and [Figure B-8 on page B-9](#) show the stage delay and slew values calculated for a timing arc by PrimeTime using CCS timing models versus the same parameters calculated by HSPICE. The values calculated by PrimeTime using CCS timing models are within two percent of HSPICE.

Figure B-7 CCS (PrimeTime) Versus HSPICE Delay Calculation

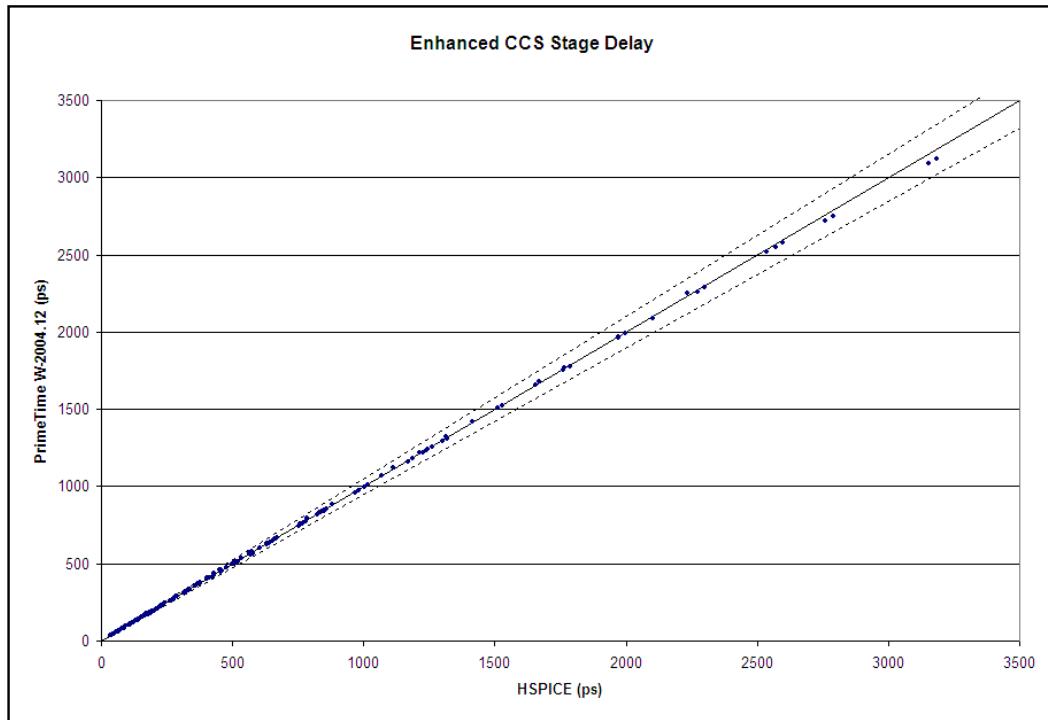
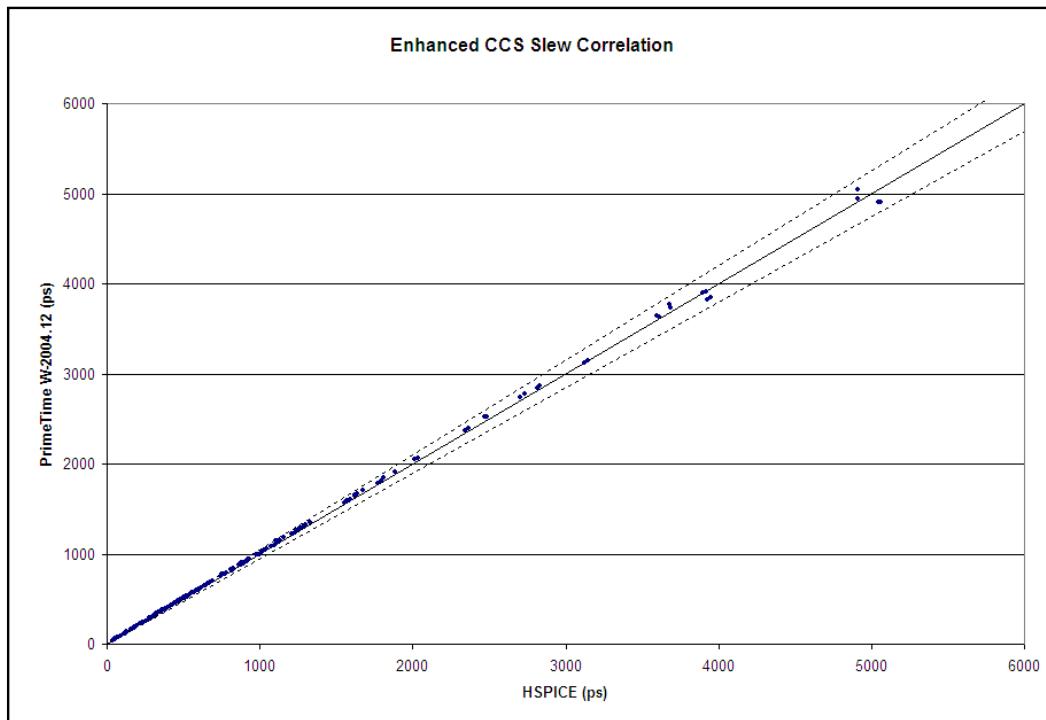


Figure B-8 CCS (PrimeTime) Versus HSPICE Slew Calculation



Index

A

acquisition
 CCS noise models 2-22
 CCS power models 2-23
 CCS timing models 1-13, 2-22
 delay arcs and CCS driver models 1-13
 NLDM and NLPM models 2-24
 NLDM timing models 1-13
 timing constraint/violation arcs 1-13
 variation-aware models 2-25
acquisition settings 1-13
acquisition types 2-21
active_drv_ground_node attribute 3-30
active_drv_ground_voltage attribute 3-30
active_drv_input attribute 3-30
active_drv_inwav attribute 3-30
active_drv_netlist attribute 3-31
active_drv_output attribute 3-31
active_drv_pwl template file attribute 3-3
active_drv_slew attribute 3-31
active_drv_slew_tol attribute 3-31
active_drv_supply_node attribute 3-31
active_drv_supply_voltage attribute 3-31
active_drv_unate attribute 3-31
active_drv_voltage_map attribute 3-31
adjustment flow 1-17

advanced_dp command 1-9
arc generation
 control 3-65
arc synthesis
 control 3-35, 3-36
arc_specs_to_template 1-14
arcs, default and state 3-65
arcs, timing and power 3-65
attributes
 margin_percent 3-49
 margin_applied 3-48
 margin_exp 3-48
 margin_mode 3-49
 margin_value 3-49
 ncx_internal_pin_signal_level_high 4-12
 ncx_margin_max 3-49, 3-108
 ncx_margin_min 3-49, 3-108
 ncx_margin_model 3-49, 3-108
template file
 active_drv_pwl 3-3
 capacitance_lower_threshold_pct_rise 3-4
 capacitance_upper_threshold_pct_fall 3-4
 capacitance_upper_threshold_pct_rise
 3-4
 capacitance_lower_threshold_pct_fall 3-4
 ccs_delay_tol 3-4
 ccs_margin_tol 3-4
 ccs_margin_tol template 3-4

clk_tran 3-4
clk_width 3-4
clocked_on_also 3-121
constraint_monitor_node 3-5
constraint_total_output_net_capacitance 3-5, 3-27
constraint_total_output_net_capacitance_pct 3-5
default_arc_mode 3-6
default_constraint_arc_mode 3-7
default_spice_option 3-7
differential_pair 3-39
driver_model 3-32
fast_mode 3-7
global_vdd 3-32
global_vss 3-32
init_cycle 3-7, 3-121
input_cap_mode 3-8
input_cap_mode_fall 3-8
input_cap_mode_rise 3-8
input_fall_threshold 3-120
input_rise_threshold 3-120
input_signal_level_high 3-39, 3-119
input_signal_level_low 3-39, 3-119
leakage_sim_opt 3-8
min_pulse_width_mode 3-8
min_pulse_width_slew 3-8
min_pulse_width_slew_high 3-9
min_pulse_width_slew_low 3-9
ncx_add_default_conditional_receiver_mode 3-9
ncx_add_default_constraint_arcs 3-10
ncx_add_default_delay_arcs 3-10
ncx_add_default_internal_power 3-10
ncx_auto_function_pin_pattern 3-10
ncx_cell_leakage_power_mode 3-11
ncx_condition 3-11
ncx_condition_output_toggle 3-11
ncx_conditional_receiver_model 3-12
ncx_constraint_accuracy 3-12
ncx_constraint_outlier_abs_tol 3-12
ncx_constraint_outlier_rel_tol 3-13
ncx_constraint_search_interval 3-13
ncx_create_arcs 4-2
ncx_create_half_unate_arcs 3-35
ncx_default_clear_arcs_only 3-14
ncx_default_conditional_receiver_mode 3-14
ncx_default_constraint_arcs_only 3-13
ncx_default_non_sequential_arcs_only 3-15
ncx_default_preset_arcs_only 3-16
ncx_defaultindex_notoggle_ccs_rcv 3-9
ncx_delay_sensitization_mode 3-16
ncx_force_create_arcs 3-36
ncx_force_create_leakage 3-36
ncx_input_tristate_mode 3-36, 3-74
ncx_internal_pin_nodeset 3-17, 4-12
ncx_internal_pin_signal_level_high 3-17, 4-12
ncx_leakage_power_mode 3-18
ncx_min_setup_based_delay 3-19
ncx_min_setup_based_delay_constraint_slew 3-19
ncx_min_setup_based_delay_setup_time_offset 3-19
ncx_mpw_rail_to_rail 3-19
ncx_multiple_descriptor 3-122
ncx_nlpm_distribution_method 3-20
ncx_nlpm_leakage_subtraction_method 3-36
ncx_nlpm_leakage_subtraction_mode 3-36
ncx_node_set 3-20
ncx_optimize_default_arc_state 3-21
ncx_optimize_state_power_arcs 3-21
ncx_optimize_state_timing_arcs 3-21
ncx_optimize_state_timing_arcs_tol_pct 3-21
ncx_optimize_va_constraint_arcs_compact 3-22
ncx_output_power_period_scale 3-22
ncx_output_threshold_pct_fall 3-3, 3-22
ncx_output_threshold_pct_rise 3-3, 3-23

ncx_output_transition_lower_bound_pct
 3-23
 ncx_output_transition_upper_bound_pct
 3-23
 ncx_retention_pin 3-24
 ncx_three_state_disable_res 3-20
 ncx_tristate_disable_arc_subtract_cvv
 3-20
 ncx_update_max_capacitance 3-24
 ncx_update_max_transition 3-25
 ncx_use_library_sensitization 3-26
 nlcx_shpr_hold 3-24
 nlpm_gate_leakage 3-26
 output_fall_threshold 3-46
 output_rise_threshold 3-47
 output_signal_level_high 3-47
 output_signal_level_low 3-47
 power_exclude_pin 3-26
 ref_state 3-47, 3-120
 sensitization 3-47
 setup_hold_method 3-27
 sim_inc_file 3-27
 sim_opt 3-28
 slew_lower_threshold_pct_fall 3-28
 slew_lower_threshold_pct_rise 3-28
 slew_upper_threshold_pct_fall 3-28
 slew_upper_threshold_pct_rise 3-28
 snps_pdriver_ratio 3-28
 switching_power_split_model 3-28
 tran_timestep 3-29
 violation_delay_degrade 3-29
 violation_delay_degrade_pct 3-29
 violation_glitch_lower_pct 3-29
 violation_glitch_mode 3-29
 violation_glitch_upper_pct 3-29
 violation_mode 3-30
 violation_mode_fall 3-30
 violation_mode_rise 3-30
 violation_pass_fail_mode 3-29, 3-30
 violation_slew_degrade_pct 3-29
 zstate_leak_threshold_pct 3-30
 veriloglib_enable 5-33

auto_function variable 1-13
 autofix 1-15
 autofunction recognition 1-13

B

backup_failed_sims command 1-9
 bundle_size command 1-9

C

capacitance and CCS receiver models 1-13
 capacitance command 1-13
 capacitance_lower_threshold_pct_rise
 template file attribute 3-4
 capacitance_upper_threshold_pct_fall
 template file attribute 3-4
 capacitance_upper_threshold_pct_rise
 template file attribute 3-4
 capacitance, maximum acquisition 3-87
 capacitance_lower_threshold_pct_fall
 template file attribute 3-4
 CCS model compaction and expansion 5-3
 CCS Models B-1
 driver models B-2
 receiver models B-5
 CCS noise merging 5-5
 summary table 1-19
 CCS noise models 2-22
 CCS noise to ECSM noise conversion 5-7
 CCS power acquisition 1-13
 CCS power models 2-23
 CCS timing models 2-22
 CCS to ECSM conversion 5-5
 CCS to ECSM settings, summary table 1-19,
 1-20
 CCS to NLDM conversion 5-8
 CCS to NLDM settings, summary table 1-20
 ccs_delay_tol template file attribute 3-4
 ccs_margin_tol attribute 3-4

ccs_margin_tol template file attribute 3-4
ccs_timing 1-13
ccs2ecsm 5-5, 5-7
ccs2nldm 5-8
ccsn_merge, merging CCS noise libraries 5-5
cell template file attribute
 ncx_active edge 3-40
cell template file attributes
 ncx_pulse_clock 3-44
 ncx_translate_msff 3-44
 simultaneous_switching 3-48
characterization
 flow diagram 2-8
 flows 1-6
 I/O cell 4-2
 incremental 2-28
 input data 2-2
 output data 2-5
 requirements 7-3
 template files 2-3
characterization flows 2-1
circuit simulator type, specifying 1-9
cleanup 1-16, 2-6
clear pins, specifying 3-40
clk_tran attribute 3-4
clk_width attribute 3-4
clock pins, active edge 3-40
clocked_on_also template file attribute 3-121
command (configuration) file 1-3
commands
 advanced_dp 1-9
 backup_failed_sims 1-9
 bundle_size 1-9
 capacitance 1-13
 concurrent_ncx 1-10
 constraint_bundle_size 1-10
 enable_generic_simulator_interface 1-8
 farm_mgr_exec 1-10
 farm_retry_interval 1-10
 farm_retry_limit 1-10
farm_type 1-10
farm_update_file 1-11
format_lib 5-2
hspice_server 1-8
input_library 1-6
job_done 1-11
leakage_model_file 1-8
log_file 1-6
man 1-4
max_job_time 1-11
max_jobs 1-11
model_extraction_to_farm 1-11
model_file 1-8
ncx_exec 1-11
netlist_dir 1-8
netlist_files 1-8
netlist_suffix 1-9
noise 1-7
output_library 1-6, 1-15
power 1-7
preanalysis_opt 1-7
prechar 1-7
project_name 1-11
queue_name 1-11
resource 1-12
simulation_dir 1-9
simulator_exec 1-9
simulator_output_extractor 1-9
simulator_type 1-9
spice_netlist_translator 1-9
templates 1-8
timing 1-8
update_interval_time 1-12
work_dir 1-6
compact CCS power 2-24
 enabling 1-13
compact CCS settings, summary table 1-18
compact CCS timing 2-24
 enabling 1-13
compact CCS timing, enabling 1-13
compact_ccs 5-3

compact_power command 1-13, 2-24
compact_timing command 1-13, 2-24
compute farm settings 1-9, 1-12
concurrent_ncx command 1-10
condition
 for arcs that originate from a specified pin 3-41
 for arcs that terminate at a specified pin 3-41
 for specific arc types 3-40, 3-92
 for specific cell types 3-41
 for specific cells 3-40, 3-41
conditional characterization 3-90
conditions, specifying 3-11
configuration file 1-3
 commands 7-5
configuration file variable
 auto_function 1-13
constant logic states on inputs 3-119
constrained_pin_transition 3-80
constraint accuracy, controlling 3-12, 3-13
constraint methodologies 3-141
constraint_bundle_size command 1-10
constraint_monitor_node template file attribute
 internal monitor node, identifying 3-5
constraint_total_output_net_capacitance
 attribute 3-5
constraint_total_output_net_capacitance
 template file attribute 3-27
constraint_total_output_net_capacitance_pct
 attribute 3-5
contention_condition attribute 3-124
conventions for documentation xiii
current_unit attribute 3-134
custom harness 3-125
customer support xiv
customizing characterization 3-11, 3-94

D

datasheets, generating 5-11

default arc 3-65
default_arc_mode template file attribute 3-6
default_constraint_arc_mode template file
 attribute 3-7
default_spice_option template file attribute 3-7
delay and constraint margins 3-96
 specifying one or more requirements 3-99
delay degradation
 relative or absolute 3-30, 3-95
 using ncx_condition_output_toggle 3-11
delay degradation, absolute and relative 3-30
design_rules 1-13
differential outputs and inputs 3-114
differential_pair (template file attribute) 3-114
differential_pair template file attribute 3-39
direction attribute 3-136
distributed model extraction 1-22
distributed processing 1-23
do list (cells to include in characterization) 3-31
dont list (cells to exclude from characterization)
 3-31
driver type at side input pins, specifying 3-38
driver_model template file attribute 3-32
driver_waveform_to_library 1-15

E

ECSM conversion settings, summary table
 1-19, 1-20
enable signals, active edge 3-40
enable_generic_simulator_interface command
 1-8
exclude cells (dont list) 3-31
expand CCS settings, summary table 1-18
expand_ccs 5-3

F

failed_cells_to_library 1-15
FAQs 8-2, 8-4

farm settings 1-9, 1-12
 farm_mgr_exec command 1-10
 farm_retry_interval command 1-10
 farm_retry_limit command 1-10
 farm_type command 1-10
 farm_update_file command 1-11, 1-23
 fast_mode template file attribute 3-7
 file and directory settings 1-6
 files used in Liberty NCX 1-2
 fix_nldm_timing 1-16
 flip-flop group 3-61
 flow diagram 2-8
 format_lib command 5-2
 function statement 3-61

G

gate leakage information in NLPM leakage
 power calculation, adding 3-26
 generating and saving new optimization
 information 1-7
 generation of sample templates 1-8
 glitch detection 3-142
 glitch thresholds, configuring 3-148
 global_vdd template file attribute 3-32
 global_vss file template attribute 3-32

H

harness, custom 3-125
 help (online) 1-4
 hold time, changing 3-24
 HSPICE
 client/server mode 1-8
 requirement 1-3
 simulator settings 1-8
 hspice_server command 1-8

I
 I/O Cell characterization 4-1
 I/O cell customization 4-2
 I/O pads 4-6
 IBIS
 conversion process 4-15
 ibis_condition_max 3-33
 ibis_netlist_dir 3-34
 ibis_netlist_suffix 3-34
 iibis_model_file 3-33
 pull_down_voltage 3-37
 pull_up_voltage 3-37
 temperature 3-38
 ibis 1-13
 ibis_condition_max attribute 3-33
 ibis_dir 1-7, 1-12, 4-16
 ibis_model_file attribute 3-33
 ibis_netlist_dir attribute 3-34
 ibis_netlist_suffix attribute 3-34
 ibis_version 1-12
 include cells (do list) 3-31
 incremental characterization 2-28
 adding cells 2-32
 do list 2-31
 flow diagram 2-30
 library creation 2-31
 template creation 2-31
 index attribute 3-136
 index values 3-79
 maximum capacitance acquisition 3-87
 minimum/maximum/mode 3-81
 percentage of maximum 3-81
 power and ground pin connections 3-132
 variation-aware 3-89
 indexes_to_template 1-14
 init_cycle template file attribute 3-7, 3-121
 initialization cycle 3-121
 input pin capacitance for scaled NLDM
 libraries, generating 5-9

i
 input_cap_mode template file attribute 3-8
 input_cap_mode_fall template file attribute 3-8
 input_cap_mode_rise template file attribute
 3-8
 input_fall_threshold attribute 3-39
 input_fall_threshold template file attribute
 3-120
 input_library command 1-6
 input_net_transition 3-80
 input_rise_threshold attribute 3-39
 input_rise_threshold template file attribute
 3-120
 input_signal_level (template file attribute) 3-39,
 3-120
 input_signal_level_high template file attribute
 3-39, 3-119
 input_signal_level_low template file attribute
 3-39, 3-119
 input_template_dir 1-14
 Interdependent Setup and Hold 3-109
 interdependent setup and hold 3-109
 constraint 3-111
 enabling 1-14
 ncx_shpr_setup_lower_bound_pct 3-112
 ncx_shpr_setup_offset 3-113
 ncx_shpr_setup_points 3-111
 shpr_constraint 3-111
 internal pin and timing arc information,
 generating 3-136
 invoking Liberty NCX 1-3

J

job_done command 1-11

L

latch group 3-62
 leakage power states, simulating 3-18
 leakage power, merging 3-43

leakage_model_file command 1-8
 leakage_sim_opt template file attribute 3-8
 library format (Model Adaptation System)
 settings, summary table 1-17
 library_name 1-14
 library_template_file 1-14
 log file 2-7
 arc netlist creation section 2-12
 cell sensitization section 2-11
 initialization section 2-10
 model extraction section 2-13
 partial sensitization flow 2-14
 program summary section 2-13
 log_file command 1-6
 low power modeling 3-134
 LSF job submission, controlling 1-23

M

macromodel for a pin 4-14
 make_ccs_noise 2-22
 man pages 1-4
 margin applied 3-48
 margin data 1-17
 margin expression 3-48
 margin maximum 3-49, 3-108
 margin minimum 3-49, 3-108
 margin mode 3-49
 margin model 3-49, 3-108
 margin value 3-49
 margin_applied template file margin attribute
 3-48
 margin_exp template file margin attribute 3-48
 margin_mode template file margin attribute
 3-49
 margin_percent template file margin attribute
 3-49
 margin_value template file margin attribute
 3-49
 max_job_time command 1-11

max_jobs command 1-11
maximum capacitance acquisition 3-87
maximum capacitance and transition rules
 1-13
merge to one CCS noise library settings,
 summary table 1-19
merge to one variation-aware library settings,
 summary table 1-18
merging variation-aware libraries 5-3
min_pulse_width_mode template file attribute
 3-8
min_pulse_width_slew template file attribute
 3-8
min_pulse_width_slew_high template file
 attribute 3-9
min_pulse_width_slew_low template file
 attribute 3-9
minimum pulse width
 full swing pulses 3-19
minimum setup time, applying 3-19
mismatch characterization 6-2
Model Adaptation System 5-2
 CCS model compaction and expansion 5-3
 CCS noise merging 5-5
 CCS noise to ECSM noise 5-7
 CCS to ECSM 5-5
 CCS to NLDM 5-8
 general usage 5-2
 VA-CCS to S-ECSM 5-10
 variation-aware library merging 5-3
model_extraction_to_farm command 1-11,
 1-22
model_file command 1-8
modeling the power group per power supply
 3-36
modifying templates before characterization
 1-7
monitoring output pins for arcs with multiple
 outputs 3-47
example 3-95

multiple-state timing arcs between two pins,
 optimizing 3-21
multithreshold-CMOS characterization 3-134

N

ncx command 1-3
NCX conditions 3-11, 3-94
ncx_active_edge attribute 3-40
ncx_active_edge attribute 3-40
ncx_add_default_conditional_receiver_model
 template file attribute 3-9
ncx_add_default_constraint_arcs template file
 attribute 3-10
ncx_add_default_delay_arcs template file
 attribute 3-10
ncx_add_default_internal_power template file
 attribute 3-10
ncx_auto_function_pin_pattern template file
 attribute 3-10
ncx_auto_function_pin_pattern variable 2-18
ncx_cell_leakage_power_mode template file
 attribute 3-11
ncx_check_nominal_va_constraint attribute
 3-34
ncx_clear attribute 3-40
ncx_condition examples 3-95
ncx_condition template file attribute 3-11, 3-90
ncx_condition_arc_type attribute 3-40, 3-92
ncx_condition_cell_name attribute 3-40, 3-41
ncx_condition_cell_type attribute 3-41
ncx_condition_output_toggle attribute 3-95
ncx_condition_output_toggle template file
 attribute 3-11
ncx_condition_pin attribute 3-41
ncx_condition_related_pin attribute 3-41
ncx_conditional_receiver_model template file
 attribute 3-12
ncx_conditional_receiver_model_states
 attribute 3-42

ncx_constraint_accuracy template file attribute
 3-12

ncx_constraint_outlier_abs_tol template file
 attribute 3-12

ncx_constraint_outlier_rel_tol template file
 attribute 3-13

ncx_constraint_search_interval template file
 attribute 3-13

ncx_create_arcs attribute 3-35

ncx_create_arcs template file attribute 4-2

ncx_create_half_unate_arcs template file
 attribute 3-35

ncx_default_clear_arcs_only template file
 attribute 3-14

ncx_default_conditional_receiver_mode
 worst_points and best_points options 3-14

ncx_default_conditional_receiver_mode
 template file attribute 3-14

ncx_default_constraint_arcs_only template file
 attribute 3-13

ncx_default_non_sequential_arcs_only
 template file attribute 3-15

ncx_default_preset_arcs_only template file
 attribute 3-16

ncx_defaultindex_notoggle_ccs_rcv template
 file attribute 3-9

ncx_delay_sensitization_mode 3-69
 first option 3-16
 worst and best_options 3-16

ncx_delay_sensitization_mode template file
 attribute 3-16

ncx_exec command 1-11

ncx_force_create_arcs attribute 3-36

ncx_force_create_leakage attribute 3-36

ncx_harness 3-125

ncx_input_delay_differential_pct template
 variable 3-26

ncx_input_delay_differential_value template
 variable 3-25, 3-117

ncx_input_delay_differential_value_pct
 template variable 3-117

ncx_input_delay_reference_mode template
 variable 3-25, 3-26, 3-115

ncx_input_tristate_mode template file attribute
 3-36, 3-74

ncx_internal_pin_nodeset template file
 attribute 3-17, 4-12

ncx_internal_pin_signal_level_high attribute
 4-12

ncx_internal_pin_signal_level_high template
 file attribute 3-17, 4-12

ncx_internal_power_mode template file
 attribute 3-18

ncx_internal_power_when 3-35, 3-42

ncx_leakage_power_mode template file
 attribute 3-18

ncx_margin_max template file margin attribute
 3-49, 3-108

ncx_margin_min template file margin attribute
 3-49, 3-108

ncx_margin_model template file margin
 attribute 3-49, 3-108

ncx_max_output_transition_time 3-42, 3-43

ncx_max_transition_constrained_pin_transitio
 n_pt 3-42

ncx_max_transition_input_net_transition_pt
 3-43

ncx_maxt_transition_mode 3-42

ncx_min_setup_based_delay attribute 3-113

ncx_min_setup_based_delay template file
 attribute 3-19

ncx_min_setup_based_delay_constraint_slew
 attribute 3-114

ncx_min_setup_based_delay_constraint_slew
 template file attribute 3-19

ncx_min_setup_based_delay_setup_time_off
 set attribute 3-114

ncx_min_setup_based_delay_setup_time_off
 set template file attribute 3-19

ncx_mm_parameter_nmos attribute 6-7

ncx_mm_parameter_pmos attribute 6-7

ncx_mm_sigma 6-5

ncx_mode_total_output_net_capacitance 3-43
ncx_mpw_rail_to_rail template file attribute 3-19
ncx_multiple_descriptor template file attribute 3-122
ncx_nlpm_distribution_method template file attribute 3-20
ncx_nlpm_leakage_subtraction_method template file attribute 3-36
ncx_nlpm_leakage_subtraction_mode template file attribute 3-36
ncx_nlpm_mode attribute 3-36
ncx_node_set attribute 3-149
ncx_node_set template file attribute 3-20
ncx_optimize_default_arc_state template file attribute 3-21
ncx_optimize_state_leakage_power attribute 3-43
ncx_optimize_state_leakage_power_mode attribute 3-43
ncx_optimize_state_leakage_power_tol attribute 3-43
ncx_optimize_state_leakage_power_tol_pct attribute 3-43
ncx_optimize_state_power_arcs template file attribute 3-21
ncx_optimize_state_timing_arcs template file attribute 3-21
ncx_optimize_state_timing_arcs_tol_pct template file attribute 3-21
ncx_optimize_va_constraint_arcs_compact template file attribute 3-22
ncx_out_to_out_arcs attribute 3-43
ncx_output_delay_differential_value template variable 3-26, 3-117
ncx_output_delay_differential_value_pct template variable 3-117
ncx_output_delay_reference_mode template variable 3-115
ncx_output_delay_reference_mode variable 3-25
ncx_output_power_period_scale template file attribute 3-22
ncx_output_threshold_pct_fall template file attribute 3-3, 3-22
ncx_output_threshold_pct_rise template file attribute 3-3, 3-23
ncx_output_transition_lower_bound_pct template file attribute 3-23
ncx_output_transition_upper_bound_pct template file attribute 3-23
ncx_power_sensitization_mode 3-70
ncx_preset attribute 3-44
ncx_pulse_clock 3-44
ncx_retention_pin template file attribute 3-24
ncx_shpr_setup_lower_bound_pct 3-112
ncx_shpr_setup_offset 3-113
ncx_shpr_setup_points 3-111
ncx_signal_skew template variable 3-118
ncx_size_total_output_net_capacitance 3-44
ncx_three_state_disable_res template file attribute 3-20
ncx_tristate_disable_arc_subtract_cvv template file attribute 3-20
ncx_update_max_capacitance template file attribute 3-24
ncx_update_max_transition template file attribute 3-25
ncx_use_leakage_model_file_for_dc_only template variable 3-26
ncx_use_library_sensitization template file attribute 3-26
ncx_use_pg_pins attribute 3-37 example 3-132
ncx_va_input_net_transition_index 3-90
ncx_va_total_output_net_capacitance_index 3-90
ncx_vary_related_input_slew 3-44
ncx_wave_rise attribute 3-52
ncx_wave_fall attribute 3-44, 3-52
ncx_wave_rise attribute 3-44
ncx_when attribute 3-45

ncx_when_fall attribute 3-45
ncx_when_ignore_funtion_pins attribute 3-46
ncx_when_rise attribute 3-46
 netlist file requirements 2-2
netlist_dir command 1-8
netlist_files command 1-8
netlist_suffix command 1-9
nlcx_shpr_hold positive template file attribute 3-24
nldm 1-13
 NLDM and NLPM models 2-24
 NLDM capacitance extraction 3-39
 NLDM capacitance index values 3-88
 NLDM conversion settings, summary table 1-20
nldm_cap command 5-9
 NLPM acquisition 1-13
nlpm_gate_leakage template file attribute 3-26
nncx_translate_msff 3-44
 node sets, defining 3-20, 3-149
 noise acquisition settings 1-13
 noise command 1-7
 noise model acquisition, enabling 1-7
nominal_va_values 2-26
non_rail_output_signal_level 3-46, 3-119
 nonrail voltage swing 3-118

O

online help 1-4
only_active_cells_to_library 1-15
.opt (template) files 2-3
 output format settings 1-15
output_fall_threshold template file attribute 3-46
output_library command 1-6, 1-15
output_rise_threshold template file attribute 3-47

output_signal_level_high template file attribute 3-47
 template file
output_signal_level_high 3-119
output_signal_level_low template file attribute 3-47
 template file
output_signal_level_low 3-119
output_template_dir 1-14
 output-to-output arc characterization 3-43
 overview of Liberty NCX 1-2

P

partial sensitization 3-69
 pessimistic delay, minimum setup time 3-113
 pg_pin and voltage_map syntax, adding to libraries 3-37
pg_pin groups 3-132
pin_opposite (template file attribute) 3-47
 pin-specific timing thresholds 3-120
 postprocessing 1-17
 power acquisition settings 1-13
 power and ground pin connections 3-132 example 3-132
 power arc 3-65
 power command 1-7
 power model acquisition, enabling 1-7
 power vectors, propagating 3-60
power_exclude_pin template file attribute 3-26
 preanalysis_opt command 1-7, 2-5
 prechar command 1-7, 2-5
 preset pins, specifying 3-44
 PrimeTime VX 2-25
 program control settings 1-15
project_name command 1-11
pull_down_voltage attribute 3-37
pull_up_voltage attribute 3-37

Q

queue_name command 1-11

R

ref_state template file attribute 3-47, 3-120
 related_internal_pg_pin attribute 3-134
 related_output_pin attribute 3-47
 example 3-95
 related_pin attribute 3-136
 related_pin_transition 3-80
 resource command 1-12
 reuse 1-16
 running Liberty NCX 1-1

S

saving pre-analysis optimization information 2-5
 sensitization 3-49
 partial 3-69
 truth tables 3-64
 vector tables 3-57
 sensitization flow 2-14
 sensitization template file attribute 3-47
 sensitization vectors 3-51
 sensitization_to_library 1-15
 sensitization_to_template 1-14
 sensitize control pins of a tristate inout pin 3-36, 3-74
 sensitize half-unate arc 3-35
 settings
 CCS to ECSM, summary table 1-19, 1-20
 CCS to NLDM, summary table 1-20
 characterization flows 1-6
 compact/expand CCS, summary table 1-18
 compute farm 1-9, 1-12
 file and directory 1-6
 library format (Model Adaptation System),
 summary table 1-17

library postprocess commands, summary table 1-17
 merge to one CCS noise library, summary table 1-19
 merge to one variation-aware library, summary table 1-18
 model acquisition 1-13
 output format 1-15
 program control 1-15
 simulation 1-8
 template usage 1-14
 settings configuration file) 1-3
 setup time for pessimistic delay 3-113
 setup time, applying margin 3-19
 setup_hold_method template file attribute 3-27
 shpr_constraint 1-14
 side input 3-67
 side_pin_driver_model attribute 3-38
 side-pin sensitization for falling arcs, specifying 3-45, 3-46
 side-pin sensitization for rising arcs, specifying 3-46
 sigma 2-26
 sim_inc_file template file attribute 3-27
 sim_opt template file attribute 3-28
 simple_termination_netlist,
 simple_termination_pin 3-47
 simple_termination_pin attribute 3-47
 simulating leakage power states 3-18
 simulation data files, directory tree 2-6
 simulation data files, file naming conventions 2-7
 simulation settings 1-8
 simulation_dir 2-6
 simulation_dir command 1-9
 simulator_exec command 1-9
 simulator_output_extractor command 1-9
 simulator_type command 1-9
 simultaneous switching 3-124
 simultaneous switching inputs, specifying 3-48

simultaneous_switching 3-48
simultaneous_switching attribute 3-48, 3-124
slew_lower_threshold_pct_fall template file attribute 3-28
slew_lower_threshold_pct_rise template file attribute 3-28
slew_upper_threshold_pct_fall template file attribute 3-28
slew_upper_threshold_pct_rise template file attribute 3-28
snps_pedriver_ratio template file attribute 3-28
SolvNet
 accessing xiv
 documentation xii
 Download Center xii
sort_output_library 1-15
SPICE
 model file 2-2
 model file name, specifying 1-8
 netlist files 2-2
 netlists directory, specifying 1-8
 netlists file name, specifying 1-8
 simulation data files, directory tree 2-6
 simulation data files, file naming conventions 2-7
 simulator executable 1-9
 simulator settings 1-8
spice_netlist_translator command 1-9
standard deviation 2-26
state arc 3-65
state table 3-63
switching_power_split_model template file attribute 3-28
synchronizer circuit 3-121
syntax (online help) 1-4
synthetic variation parameters 6-2

T

temperature attribute 3-38

template
 ncx_va_input_net_transition_index 3-90
 ncx_va_total_output_net_capacitance_index 3-90
nominal_va_values 2-26
va_parameters 2-26
va_variation_values 2-26
template file
 init_cycle 3-7, 3-121
 input_fall_threshold 3-120
 input_rise_threshold 3-120
 zstate_leak_threshold_pct 3-30
template files 2-3, 3-2
 cell attributes 3-39, 3-48
 characterization attributes 3-3
 characterization index values 3-79
 clocked_on_also 3-121
 complex cell features 3-109
 conditional characterization 3-90
 constant logic states on inputs 3-119
 custom harness 3-125
 differential outputs and inputs 3-114
 differential_pair 3-39
 do list 3-31
 dont list 3-31
 driver_model 3-32
 generating 2-4
 global_vdd 3-32
 global_vss 3-32
 include statement 3-3
 initialization cycle 3-121
 input_fall_threshold, input_rise_threshold 3-39
 input_signal_level 3-39, 3-120
 input_signal_level_high 3-39, 3-119
 input_signal_level_low 3-39, 3-119
 input_template_dir 3-2
 interdependent setup and hold 3-109
 library and cell attributes (table) 3-3
 library attributes (table) 3-30
 library_template_file 3-2
 margin_applied 3-48

margin_exp 3-48
 margin_mode 3-49
 margin_percent 3-49
 margin_value 3-49
 ncx_check_nominal_va_constraint 3-34
 ncx_harness 3-125
 ncx_internal_power_when 3-35, 3-42
 ncx_margin_max 3-49, 3-108
 ncx_margin_min 3-49, 3-108
 ncx_margin_model 3-49, 3-108
ncx_max_
 transition_related_pin_transition_pt 3-42
 ncx_max_output_transition_time 3-42, 3-43
 ncx_max_transition_constrained_pin_transit_ion_pt 3-42
 ncx_max_transition_input_net_transition_pt 3-43
 ncx_max_transition_mode 3-42
 ncx_mode_total_output_net_capacitance 3-43
 ncx_multiple_descriptor 3-122
 ncx_pulse_clock 3-44
 ncx_size_total_output_net_capacitance 3-44
 ncx_translate_msff 3-44
 ncx_vary_related_input_slew 3-44
 non_rail_output_signal_level 3-46, 3-119
 nonrail voltage swing 3-118
 output_fall_threshold 3-46
 output_library 3-2
 output_rise_threshold 3-47
 output_signal_level_high 3-47, 3-119
 output_signal_level_low 3-47, 3-119
 pin_opposite 3-47
 pin-specific timing thresholds 3-120
 ref_state 3-47, 3-120
 sensitization 3-47, 3-49
 side_pin_driver_model 3-38
 simple_termination_netlist,
 simple_termination_pin 3-47
 simultaneous_switching 3-48
 synchronizer circuit 3-121
 syntax 3-2
 va_nominal_values 3-38
 va_parameters 3-38
 va_parameters, va_nominal_values 3-38
 template usage settings 1-14
 template_files
 output_template_dir 3-2
 template_suffix 1-14
 templates command 1-8
 test_simulator 1-16
 testbench creation for Verilog models 1-10, 5-69
 timing acquisition settings 1-13
 timing arc 3-65
 timing command 1-8
 timing model acquisition, enabling 1-8
 timing vector types 3-60
 timing_arcs_to_template 1-14
 timing_sense attribute 3-136
 timing-arc sensitization vectors, specifying 3-44, 3-52
 total_output_net_capacitance 3-80
 tran_timestep template file attribute 3-29
 transistor mismatch characterization 6-2
 Gaussian distribution 6-3
 Monte Carlo simulation 6-3
 ncx_mm_parameter_mean 6-8
 ncx_mm_parameter_sigma 6-8
 ncx_mm_sigma 6-5, 6-8
 operation 6-5
 options 6-6
 sigma offset 6-5
 specifying mean and sigma values 6-7
 synthetic variation parameters 6-2
 troubleshooting 8-2, 8-4
 truth table 3-63

U
 update_interval_time command 1-12

usage flows (characterization and library
formatting), summary 1-23
use_driver_waveform_from_library 1-15
user-defined attributes 3-151

V

va_merge 5-3
va_nominal_values attribute 3-38
va_parameters 2-26
va_parameters attribute 3-38
va_variation_values 2-26
va_variation_values attribute 3-38
vaccs2secsm 5-10
validity checks 3-61
 flip-flop group 3-61
 function statement 3-61
latch group 3-62
state table 3-63
truth table 3-63
variables
 ncx_auto_function_pin_pattern 2-18
 ncx_output_delay_reference_mode 3-25
 template
 ncx_input_delay_differential_pct 3-26
 ncx_input_delay_differential_value 3-25,
 3-117
 ncx_input_delay_differential_value_pct
 3-117
 ncx_input_delay_reference_mode 3-25,
 3-26, 3-115
 ncx_output_delay_differential_value 3-26,
 3-117
 ncx_output_delay_differential_value_pct
 3-117
 ncx_output_delay_reference_mode 3-115
 ncx_signal_skew 3-118
 ncx_use_leakage_model_file_for_dc_only
 3-26

variation-aware
 CCS to S-ECSM conversion 5-10
 index values 3-89
 libraries, merged and separate 2-26
 library
 merge settings, summary table 1-18
 library merging 5-3
 models 2-25
Verilog models, generating 5-33
veriloglib_enable 5-33
violation_delay_degrade template file attribute
 3-29
violation_delay_degrade_pct template file
 attribute 3-29
violation_glitch_lower_pct template file
 attribute 3-29
violation_glitch_mode template file attribute
 3-29
violation_glitch_upper_pct template file
 attribute 3-29
violation_mode attribute 3-144
violation_mode template file attribute 3-30
violation_mode_fall template file attribute 3-30
violation_mode_rise template file attribute
 3-30
violation_pass_fail_mode template file
 attribute 3-29, 3-30
violation_slew_degrade_pct template file
 attribute 3-29

W

work_dir command 1-6

Z

zstate_leak_threshold_pct template file
 attribute 3-30