Liberty™ NCX
User Guide

Version B-2008.12, December 2008

Comments?
Send comments on the documentation by going to http://solvnet.synopsys.com, then clicking "Enter a Call to the Support Center."

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

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

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

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

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

Trademarks (™)

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

Service Marks (^{SM})

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

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

Contents

7. Troubleshooting

Preface

This preface includes the following sections:

What's New in This Release

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

   http://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. Click 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

Liberty NCX works with HSPICE, Library Compiler, and similar tools. For additional information about HSPICE and Library Compiler, see Documentation on the Web, which is available through SolvNet at the following address:

http://solvnet.synopsys.com/DocsOnWeb

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

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

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

http://solvnet.synopsys.com

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

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

Contacting the Synopsys Technical Support Center

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

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr

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.

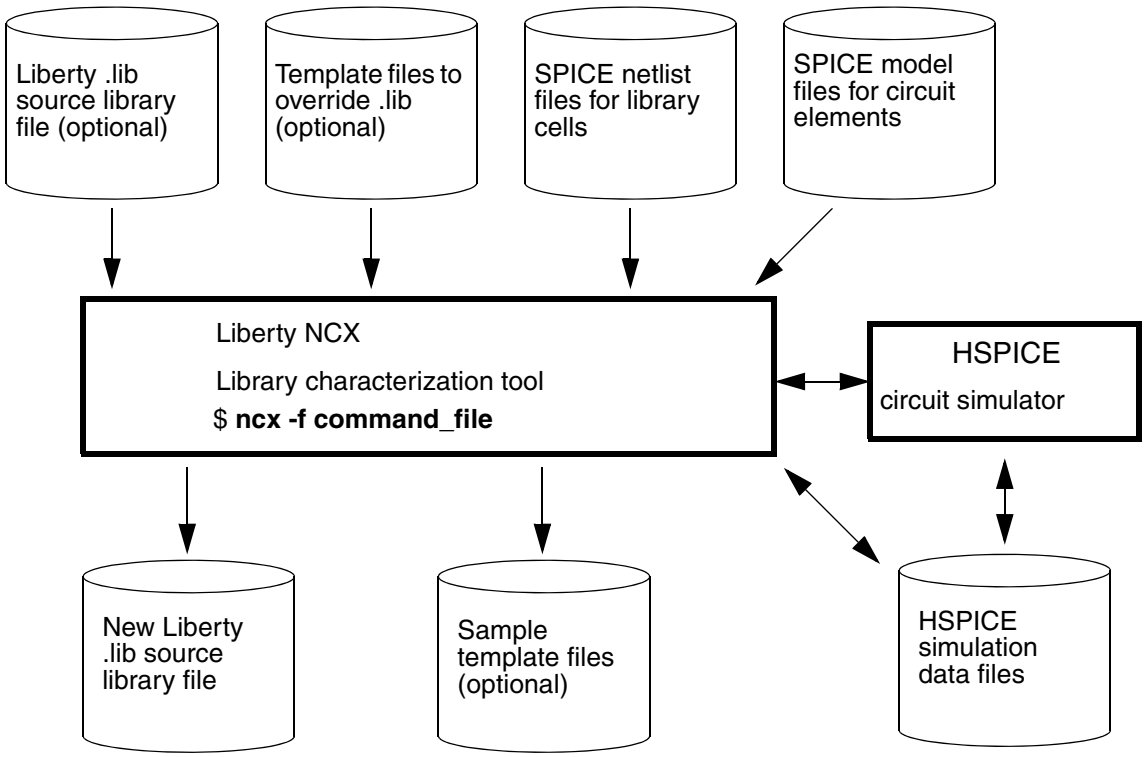
This chapter contains the following sections:

Overview of Liberty NCX

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 command file that sets a number of options for the characterization run. Then you invoke Liberty NCX and execute the command file. For example, you might create a command file called “mycmd” containing the following commands:

```
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 these option settings:

```
% ncx -f mycmd
```

For a characterization task, the command file must specify the SPICE model file name, the SPICE netlist directory, and the SPICE simulator executable. The foregoing 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` command file must specify the input library name, the output library name, and parameters for the formatting or conversion operation.

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

```
% ncx -help
...
Command-file options/syntax:

Files/folders:
 set input_library  : (string) input seed library file ...
 set output_library : (string) output library file name
 set log_file       : (string) output log file ...
 ...
```

The following tables briefly describe the characterization settings:

The following tables briefly describe the Model Adaptation System settings:

Table 1-1 File/Folder Settings

Command 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.	
`output_library`	The name of the new Liberty (.lib) library file created by Liberty NCX.	
`log_file`	The name of the file in which Liberty NCX stores the session log.	`ncx.log`
`work_dir`	The name of the working directory used by Liberty NCX to write all working files.	`work`

Table 1-2 Characterization Flow Settings

Command Setting	Description	Default
`timing`	Causes acquisition of NLDM/CCS timing models.	`false`
`power`	Causes acquisition of NLPM/CCS power models.	`false`
`noise`	If CCS models are being generated, also causes generation of CCS noise models (future enhancement, not implemented in this release).	`false`
`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 `output_template_dir`.	`false`
`prechar`	Allows you to review and modify the templates before performing characterization. When you set `prechar` to true, Liberty NCX writes the information from your seed library to the template files and stops after the optimization is complete.	`false`

Table 1-2 Characterization Flow Settings (Continued)

Command Setting	Description	Default
`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 the `preanalysis_opt` command is set to `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 to rerun optimizations whether or not optimization information is included in the template files, set `preanalysis_opt` to `force`. If you want Liberty NCX to ignore the existing optimization information during characterization, set `preanalysis_opt` to `false`.	`auto`

Table 1-3 Simulation Settings

Command Setting	Description	Default
`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. To use the HSPICE client/server mode, enable the `hspice_server` command in the configuration file, as shown: `set hspice_server true` The `hspice_server` command is off (set to false) by default.	`false`
`simulator_exec`	Specifies the absolute path to the SPICE circuit simulator executable. Supported simulators are `hspice`, `eldo`, and `spectre`.	

Table 1-3 Simulation Settings (Continued)

Command Setting	Description	Default
`simulator_type`	Specifies the type of circuit simulator: `hspice`, `eldo`, or `spectre` The default simulator is `hspice`. You can also invoke the Eldo simulator by setting the `simulator_type` to `eldo` or use the Spectre simulator by setting `simulator_type` to `spectre`. For example, `set simulator_type eldo`	`hspice`
`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: `set leakage_model_file` *`leakage_model_path`*	
`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_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 `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.	
`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_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.	.

Table 1-4 Compute Farm Settings

Command Setting	Description	Default
farm_type	Specifies the multiprocessor farm system. Can be set to LSF (Platform Computing LSF), SGE (Sun Grid Engine), or NoFarm (local processor usage only). For LSF or SGE, you must run Liberty NCX on a submit machine that can accept bsub (LSF) or qsub (SGE) job submission commands.	LSF
farm_update_file	Set farm_update_file to point to a specified file, and you can update the file at any point during the NCX run. For more information, see "Distributed Processing" on page 2-13.	none
queue_name	Specifies the compute farm queue name.	normal
resource	Specifies the compute farm resource requirements such as memory, swap space, CPU load, or host type. LSF resources can be displayed with the lsinfo command. For example, set farm_type LSF and set resource opteron will cause Liberty NCX to run AMD Opteron Linux systems in the LSF farm by submitting jobs using bsub -R operteron script_file.bat. Possible SGE resource names include arch (solaris64, glinux), cputype (sparc, ia32, amd64), os_bit (64, 32), and os_distribution (SunOS, SuSE, redhat). For example, set farm_type SGE and set resource arch=solaris64 sets the architecture to Solaris.	
project_name	Specifies the compute farm project name.	ncxtest
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
farm_retry_limit	Specifies the maximum number of farm job re-submission attempts for the same job. See also the update_interval variable, which specifies the time interval between the compute farm status checks.	3
max_jobs	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

Table 1-4 Compute Farm Settings (Continued)

Command Setting	Description	Default
`max_job_time`	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`
`model_extraction_to_farm`	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, set `model_extraction_to_farm` to true. Note: Distributed model extraction is currently only supported for the HSPICE simulator. For more information about distributed model extraction, see "Distributed Model Extraction" on page 2-13.	`false`
`update_interval`	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 Timing, Power, and Noise Acquisition Settings

Command Setting	Description	Default
`capacitance`	Causes acquisition of capacitance and CCS receiver models.	`true`
`ccs_power`	Causes acquisition of CCS power models. When the `ccs_power` command is set to true (the default), Liberty NCX enables CCS power compaction.	`true`
`ccs_timing`	Causes acquisition of CCS timing models.	`true`
`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 save disk space. When the `compact` command is set to true (the default), Liberty NCX enables CCS power compaction.	`true`

Table 1-5 Timing, Power, and Noise Acquisition Settings (Continued)

Command Setting	Description	Default
`compact_power`	Set the `compact_power` command to true to enable both compact CCS timing and compact CCS power. If you set `compact_power` to false, Liberty NCX enables compact CCS timing but enables expanded CCS power.	`true`
`compact_timing`	Set the `compact_timing` command to true to enable both compact CCS power and compact CCS timing. If you set `compact_timing` to false, Liberty NCX enables expanded CCS timing and compact CCS power.	`true`
`constraint`	Causes acquisition of timing constraint/violation arcs: setup, hold, recovery, removal, and minimum pulse width. If you are only interested in delay and slew, set this option to false 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`
`nldm`	Causes acquisition of NLDM timing models.	`true`
`nlpm`	Causes acquisition of NLPM power models.	`false`
`shpr_constraint`	Causes characterization of interdependent setup/hold models in addition to conventional setup/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`
`ccs_noise`	Causes acquisition of CCS noise models (future enhancement, not available in current release).	`false`

Table 1-6 Template Usage Settings

Command Setting	Description	Default
`input_ template_dir`	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`	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`	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`	The name of the library.	
`template_suffix`	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`

Table 1-7 Output Format Settings

Command Setting	Description	Default
`precision`	The number of digits of precision used to represent floating-point values in the generated library.	`7`
`only_active_ cells_to_ library`	Specifies whether to write only the cells that have been characterized by Liberty NCX to the output library.	`false`
`sensitization_ to_library`	Specifies whether to write cell sensitization information to the output library.	`false`

Table 1-7 Output Format Settings (Continued)

Command Setting	Description	Default
`driver_waveform_ to_library`	Specifies whether to save the predriver information into the output library. The saved description provides enough information to reproduce the exact predriver waveform used for characterization.	`true`
`use_driver_ waveform_ from_library`	Specifies whether to use the predriver information in the seed library for characterization.	`false`

Table 1-8 Program Control Settings

Command Setting	Description	Default
`autofix`	Specifies whether to attempt 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 runlvl parameter set to 6, for higher accuracy at the cost of additional runtime.	`true`
`fix_nldm_timing`	Ensures monotonically increasing cell delays with increasing output load capacitance in the generated NLDM timing models.	`true`
`cleanup`	Specifies the level of cleanup done by deletion of simulation result files after characterization. There are four possible settings: 0, 1, 2, and 3. 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. For any of these settings, the cell simulation directories are preserved if there is a characterization failure of any kind.	`1`
`reuse`	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 good results.	`false`
`test_simulator`	Generates and simulates a small test circuit prior to the start of characterization, in order to ensure valid simulator and model settings.	`true`

Table 1-9 Library Format Translation

Command Setting	Description	Default
`expand_ccs`	Converts a compact CCS library into an expanded CCS library.	`false`
`compact_ccs`	Converts an expanded CCS library into a compact CCS library.	`false`
`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.	`false`
`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.	`false`
`ccs2ecsm`	Converts a CCS library into a Cadence ECSM library.	`false`
`ccs2nldm`	Converts a CCS library into an NLDM library.	`false`
`input_library`	The name of the existing library file used as input to the formatting operation.	
`output_library`	The name of the output library file produced by the formatting operation.	
`nldm_cap`	Uses linear interpolation to calculate NLDM input pin capacitance for a scaled library. The valid values are `linear`, `c1`, and `avg`.	`c1`
`linear`	When `nldm_cap` is set to `linear`, Liberty NCX linearly interpolates the NLDM input capacitance found in the two corner libraries.	none
`c1`	When `nldm_cap` is set to `c1`, Liberty NCX 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`.	none
`avg`	When `nldm_cap` is set to `avg`, Liberty NCX 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.	none

Table 1-10 Compact/Expand CCS Options

Command Setting	Description	Default
`input_library`	The name of the existing library file used as input to the formatting operation.	
`output_library`	The name of the output library file produced by the formatting operation.	

Table 1-11 Merge to One Variation-Aware Library Options

Command Setting	Description	Default
`nominal_library`	The name of the nominal variation-aware library, followed by the parameter names and nominal values: *`lib_name par1 val1 par2 val2 ...`*	
`va_library_list`	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: {*`lib1_name val1 val2 ... lib2_name val1 val2 ... lib2N val1 val2 ...`* }	
`output_library`	The name of the unified variation-aware library created by merging the input libraries.	
`slew_indexes`	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 `all` to retain all the slew index values. This setting affects only the libraries in the `library_list`. All index values in the nominal library are always retained.	`all`
`load_indexes`	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 `all` to retain all the load index values. This setting affects only the libraries in the `library_list`. All index values in the nominal library are always retained.	`all`

Table 1-12 Merge to One CCS Noise Library Options

Command Setting	Description	Default
`timing_library`	The existing library containing CCS timing models.	
`noise_library`	The existing library containing CCS noise models.	
`output_library`	The new library containing both CCS timing and CCS noise models.	

Table 1-13 Format CCS to ECSM Options

Command Setting	Description	Default
`library_list`	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`	The new ECSM library file name.	
`process`	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`	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`	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
`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`	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}`

Table 1-14 Format CCS to NLDM Library Options

Command Setting	Description	Default
`library_list`	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`	The new NLDM library file name.	
`process`	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`	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`	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

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 characterization 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.”

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, and CCS-to-NLDM conversion. These operations are described in Chapter 4, "Library Conversion Support."

You can also review and modify the templates before performing characterization by setting the `prechar` command 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` command is set to `false` by default.

In addition, you can save the pre-analysis 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 command is set to `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` command to `force`. If you want Liberty NCX to ignore the existing optimization information during characterization, set `preanalysis_opt` 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_templates_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 command in the configuration file:

```
set input_templates_dir ./output_template
```

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
- 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 command 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_predriver ;
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.

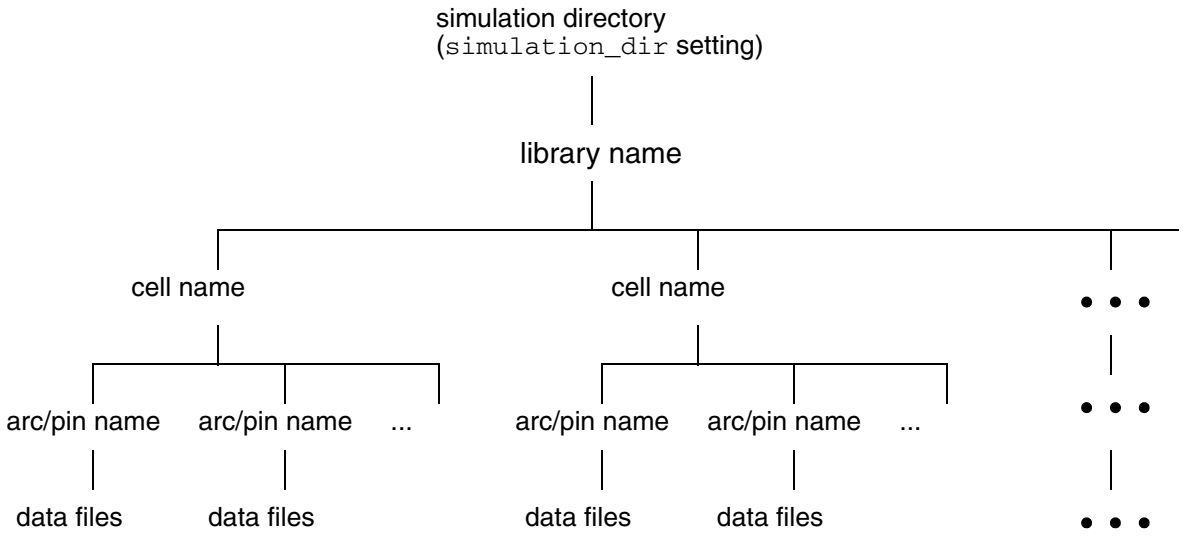
For more information, see Chapter 3, “Template Files”.

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` setting of Liberty NCX.

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.

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/pin directories, which contain the simulation data for individual timing arcs and other characterization data.

The arc/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/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 2-7 and Figure 2-3 on page 2-8 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 command 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, Build Phase

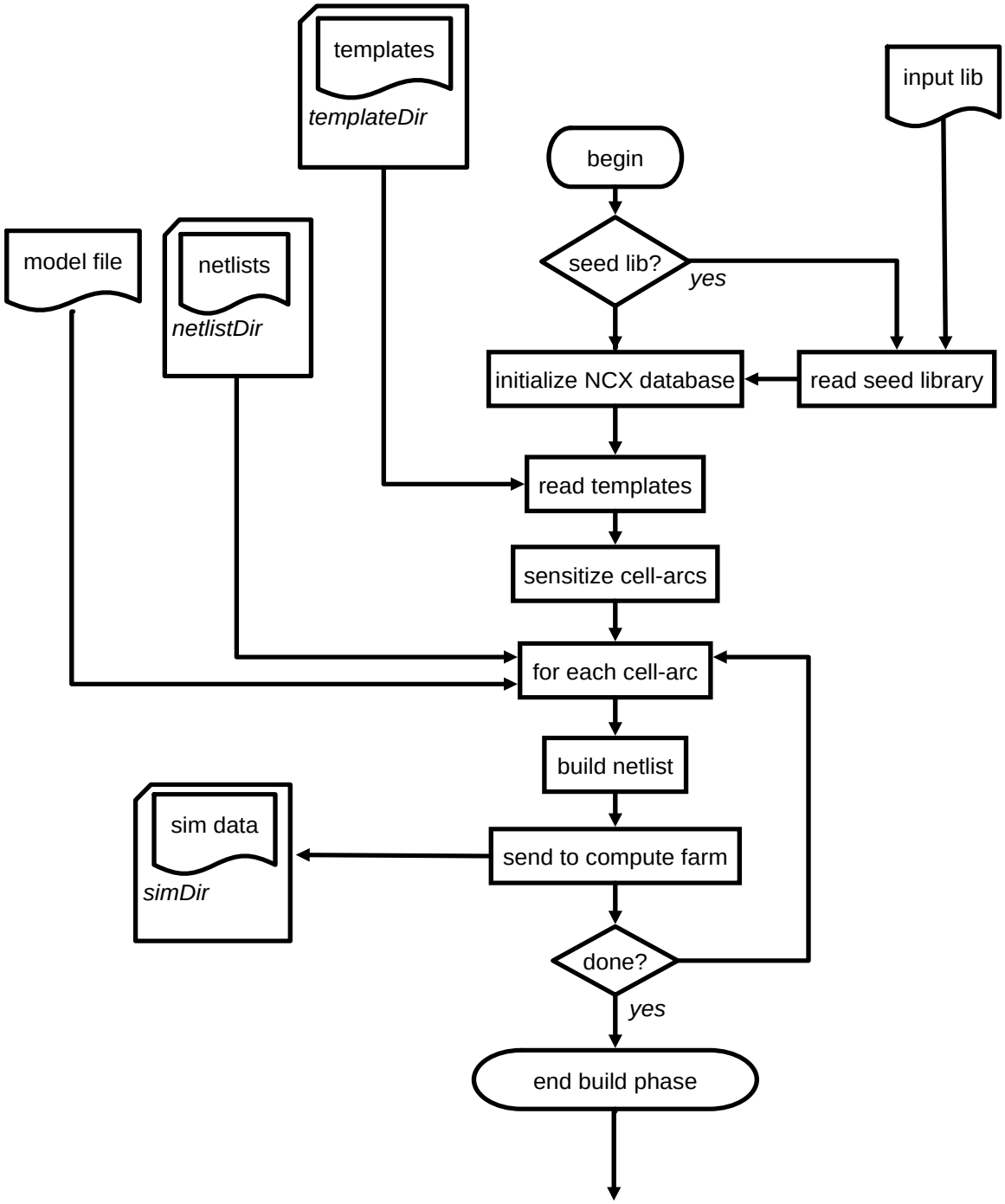
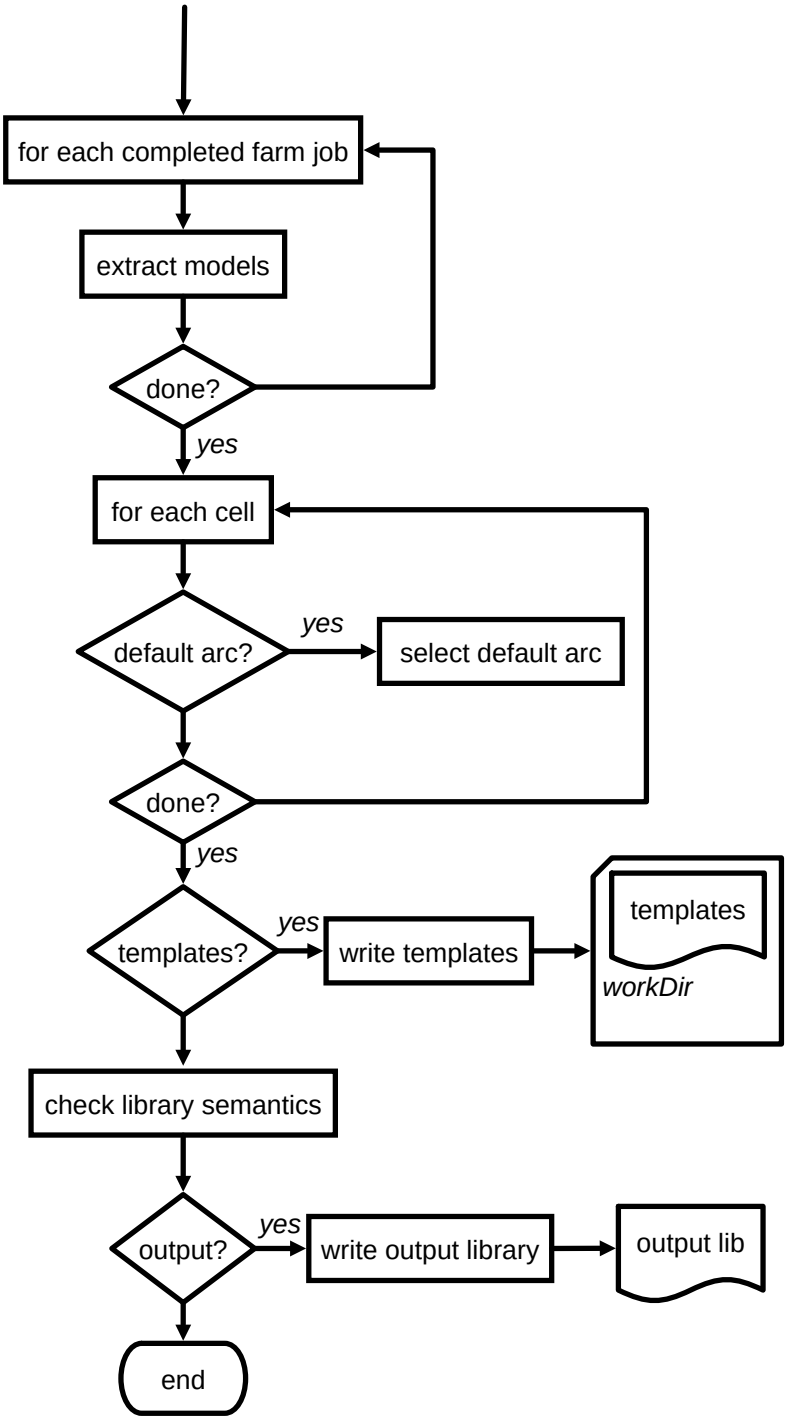


Figure 2-3 Liberty NCX Operation, Output Phase



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
- 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 command 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) v2007.06
            ...  (copyright notice)  ...

Build date: May  7 2007
 ...
Program configuration:
  Files/folders:
    output_library      : ncx.lib
    log_file            : /remote/techp5/gra/p1/ncx.log
    work_dir            : /remote/techp5/gra/p1/work
  Flows:
    timing              : true
    power               : true
    noise               : false
    templates           : false
  Simulation settings:
    simulator_exec      : /global/apps3/hspice/bin/hspice
    simulator_type      : hspice
    model_file          : /remote/techp5/gra/p1/hss_ttrc
    netlist_dir         : /remote/techp5/gra/p1/netlists
    netlist_suffix      : .inc
    simulation_dir      : /remote/techp5/gra/p1/ncx_sim
  Compute farm:
    farm_type           : LSF
    queue_name          : normal
```

```
    project_name        : ncxtest
    bundle_size         : 50
    max_jobs            : 100
    max_job_time        : 600
    update_interval     : 30
  Output format control:
    precision           : 7
    only_active_cells_to_library: false
    sensitization_to_library: false
  Program control:
   ...

Mon May  7 10:08:28 2007

begin flow...
```

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, and flip-flop 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 3-50.

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

```
Mon May  7 10:08:33 2007

generating sensitizations...

cell SLOW_125_0P9_NCX::and2x1w (1 of 5)
Generating truth-table for 2 inputs and 1 outputs

assigning arc sensitization templates...
done
creating timing arcs...
adding arc: A->Z combinational positive_unate
adding arc: B->Z combinational positive_unate
2 timing arcs created

sensitizing combinational positive_unate arc 0: A -> Z:
  rise: and2x1g1t10w_wave_0_6
  fall: and2x1g1t10w_wave_0_8
sensitizing combinational positive_unate arc 1: B -> Z:
  rise: and2x1g1t10w_wave_0_4
```

```
  fall: and2x1g1t10w_wave_0_7

cell SLOW_125_0P9_NCX::invx12w (2 of 5)
  ...
```

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:

```
Mon May  7 10:10:01 2007

building netlists...

cell SLOW_125_0P9_NCX::and2x1w (1 of 5)
  pin Z (1 of 3) (2 arcs)
    A -> Z
    B -> Z
  building non-propagated power netlists...
    [4 netlists] done
  building dc leakage power netlists...
    [4 netlists] done

cell SLOW_125_0P9_NCX::invx12w (2 of 5)
  ...
```

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:

```
Mon May  7 10:10:03 2007

checking simulation status...
checking compute farm status in 28 seconds at 10:10:31
checking compute farm status in 3 seconds at 10:10:31
checking job status...0 queued, 1 running, 0 pending, 2
done...done

[1/52] or2x1g1t10w::A
  netlist:
/remote/techp5/gra/p1/ncx_sim/SLOW_125_0P9_NCX/or2x1w/
  dor2x1w_0003/dor2x1_0003.sp
  extracting NLDM models...done
  extracting CCS models...done
done
\gzip -qf /remote/techp5/gra/p1/ncx_sim/SLOW_125 ...

[2/52] or2x1w::B
  ...

Mon May  7 10:13:38 2007

extracting default arcs and design rules...

cell SLOW_125_0P9_NCX::and2x1w (1 of 5)

cell SLOW_125_0P9_NCX::invx12w (2 of 5)

 ...

done

CCS statistics:
  380 curves processed
```

```
total model extraction time : 1.376 seconds (elapsed)
total default arc extraction time: 0.034 seconds (elapsed)
```

Distributed Model Extraction

Liberty NCX performs 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 file command 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` command is set to false by default.

Note:
Distributed model extraction is currently only supported for the HSPICE simulator.

Distributed Processing

You can use the `farm_update_file` command to control the LSF job submission based on the time of the day, the load on the grid, and so on. Set the `farm_update_file` command in the configuration file to point to a specified file, and then update the file at any point during the Liberty NCX run. The contents of the file can contain one or more of the following configuration file commands:

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

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

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:

```
Mon May  7 10:13:38 2007

merging library...done

Mon May  7 10:13:38 2007

writing
/remote/techp5/gra/p1/work/SLOW_125_0P9_NCX.opt...done
total template write time: 0.004 seconds (elapsed)

Mon May  7 10:13:38 2007

deleting sensitization templates...done
total template deletion time: 0.000 seconds (elapsed)

Mon May  7 10:13:38 2007

Checking semantics...1...2...done
writing SLOW_125_0P9_NCX to ncx.lib...done
total library write time: 0.121 seconds (elapsed)

Mon May  7 10:13:38 2007

done

total farm time          : 259.000 seconds (average 64.750)
  total farm wait time   : 22.000 seconds (average 5.500)
  total farm run time    : 237.000 seconds (average 59.250)
local simulation time    : 4.609 seconds (average 4.609)
total program time       : 309.960 seconds (5.17 min.)
```

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 PrimePower.

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 4-8.

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 4-6.

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 command file. Table 1-5 on page 1-9 lists and describes the options available in the command 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 command 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 2-21.

CCS Noise

The current release of Liberty NCX does not directly support the acquisition of CCS noise models. To generate CCS noise models, first perform CCS timing characterization (with `ccs_timing` set to `true` in the Liberty NCX command file) to generate the CCS timing library. Then use the `make_ccs_noise` characterization utility to add CCS models to the library. This utility has been distributed with each PrimeTime SI product installation package since the Z-2007.03 release. It is documented in a separate manual, the *Make CCS Noise User Guide*, available in the PrimeTime Suite of documentation in SolvNet, at http://solvnet.synopsys.com.

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 command 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 User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries, Version Z-2007.03*.

Compact CCS Power

By default, Liberty NCX enables CCS power compaction. If the `compact` or `ccs_power` commands are set to true in the command file (the default setting), Liberty NCX reduce the library size of CCS power libraries. If the `compact_power` command is set to true in the configuration file (the default setting), Liberty NCX enables both compact CCS timing and compact CCS power.

If you set `compact_power` to false, Liberty NCX disables compact CCS power but enables compact CCS timing. You can set the `compact_timing` command to true in the configuration file to enable both compact CCS power and compact CCS timing.

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

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`.

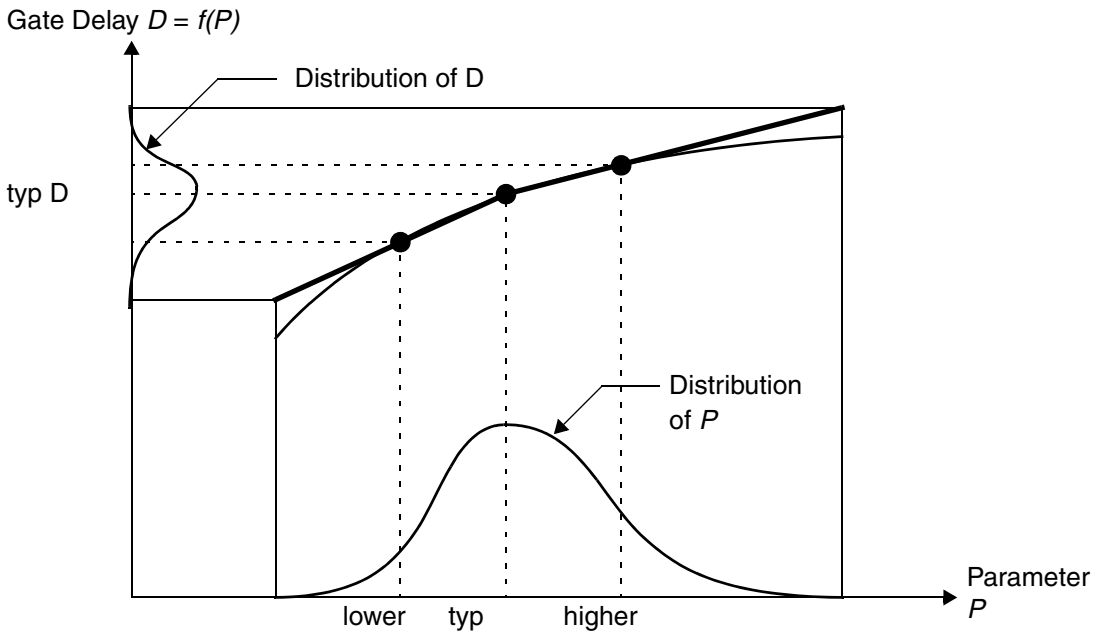
Similarly, to invoke acquisition of CCS and NLPM power models in a single characterization run, set `power` to `true`, `ccs_power` to `true`, and `nlpm` to `true`.

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 2-4.

Figure 2-4 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
```

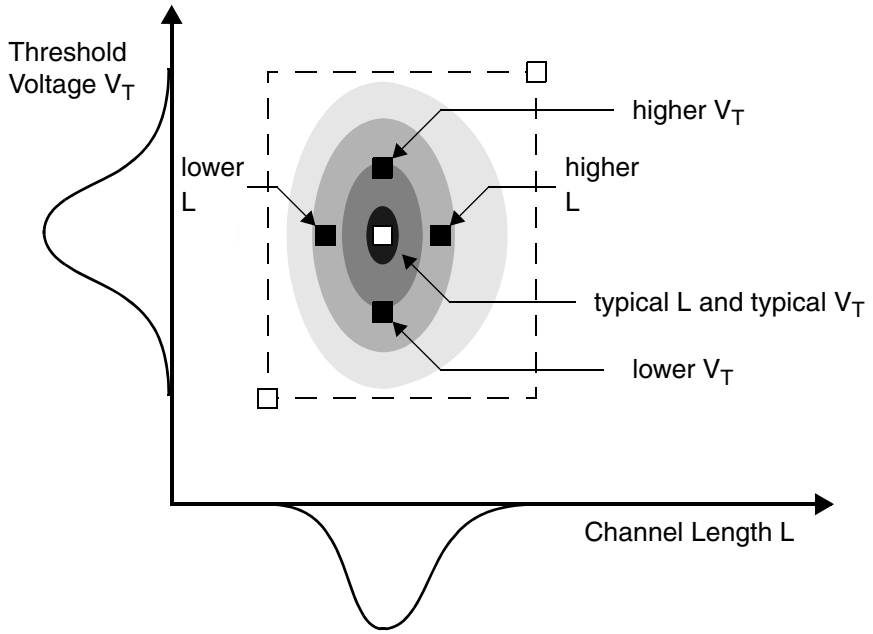
For details, see “Variation-Aware Index Values” on page 3-44.

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 2*N*+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 2*N*+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 2-5.

Figure 2-5 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 4-4.

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.

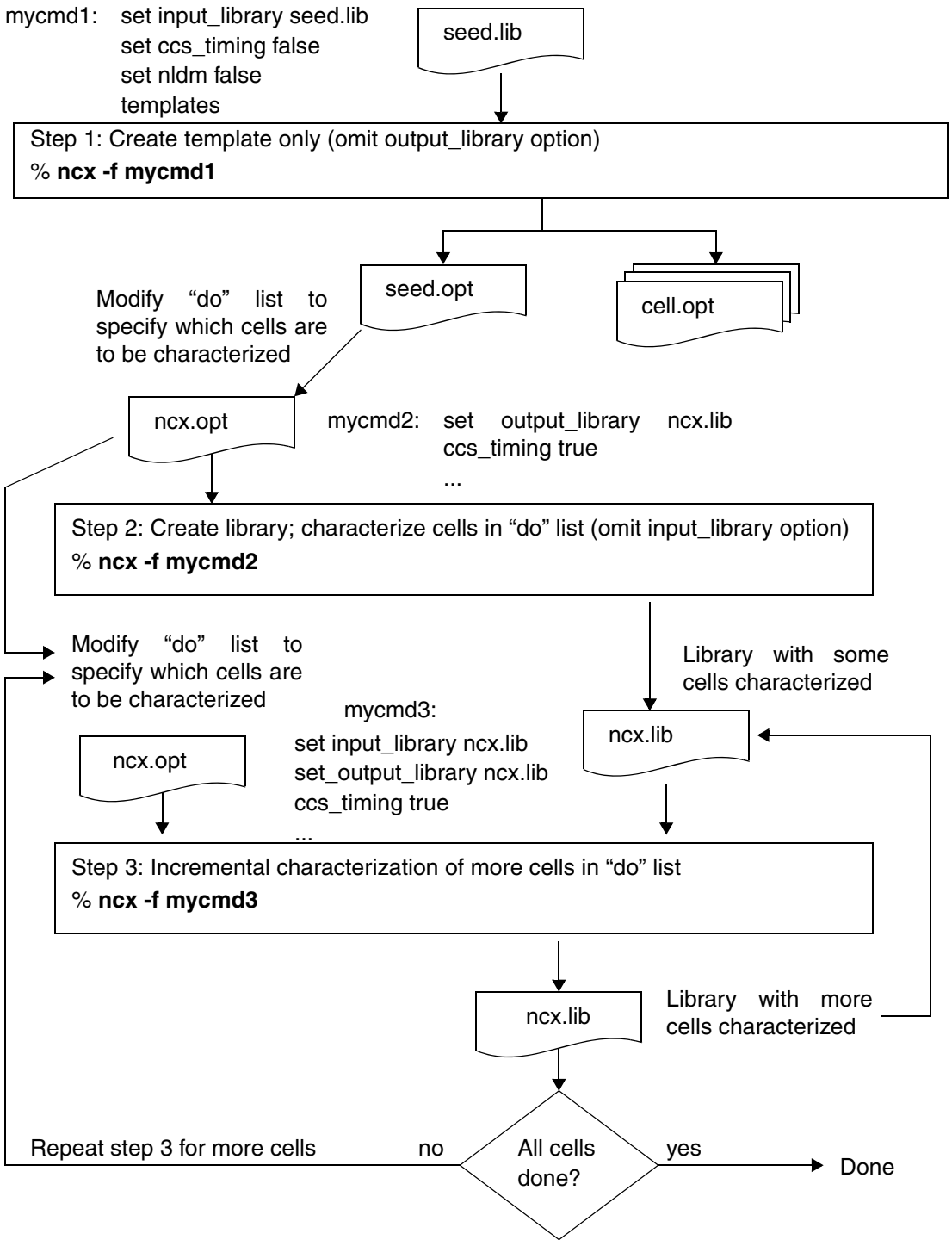
Incremental characterization consists of three basic steps:

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 2-6.

Figure 2-6 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 command 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

You can customize the characterization parameters by creating or modifying the 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, so you can choose whether to use them. 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 a new template file that can be edited and used in a future session, set `tempates` to `true`. The new template files are written to the directory specified by `output_template_dir`.

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 colon, the value of the attribute, and a terminating semicolon.
- A complex attribute is specified by the name of the attribute followed by a colon, the multiple values of the complex attribute separated by spaces, and a terminating semicolon.
- 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 ;
```

In this syntax, there is no colon. 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.

Arc Synthesis Control

When Liberty NCX generates a library using a template-based flow that does not specify timing arcs in the constituent cell templates, it synthesizes timing arcs using the respective functional descriptions. The `ncx_create_arcs` attribute is provided to control this behavior.

The `ncx_create_arcs` attribute also supports manual vector reduction for leakage and power arcs. You can manually control the nonpropagating internal power arcs on input pins. Specify `ncx_create_arcs` in the library template file to control which arcs are synthesized for timing models and for power models instances.

Note:

The `ncx_create_arcs` and `ncx_leakage_power_when` attributes have been consolidated to support user-defined leakage `when` conditions. The `ncx_leakage_power_when` attribute is no longer supported. Its functionality has been transferred to `ncx_create_arcs`.

The `ncx_create_arcs` syntax is as follows:

```
ncx_create_arcs : cellName fromPin toPin arcType ignore|states state1... ;
```

where

- `cellName` is the name of the cell or cell family to which the attribute applies.
- `fromPin` is the name of the arc's source pin or pins. Where applicable, this is the value of the `related_pin` attribute of the arc.

  For leakage definition, set the `fromPin` value to the asterisk (*) wildcard character.
- `toPin` is the name of the arc's destination pin or pins. The arcs appear in the pin groups of these pins in the library.

  For leakage definition, set the `toPin` value to the asterisk (*) wildcard character.

- `arcType` is the name of the timing or power models to which the attribute applies. The value can be any valid `timing_type` value in Liberty syntax or one of the following special values:
  - `delay` applies the attribute to all propagating arcs between the specified pins.
  - `constraint` applies the attribute to all constraint arcs between the specified pins.
  - `leakage_power` applies the attribute to all leakage power models for the specified cell. The pin specifications for this model should be set to the asterisk (*) wildcard character. Setting `arcType` to `leakage_power` applies the `ncx_create_arcs` definition to all leakage power models for the specified cell.
  - `internal_power` applies the attribute to all nonpropagating power models of the input pins. The `fromPin` and `toPin` names should be the same.

  Note:

  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.
- After you specify the `arcType` value, specify one of the following options:
  - `ignore` specifies that the model be ignored completely.
  - `states` specifies a list of states (`when` conditions) for which arcs will be created. Enclose each `when` condition in double quotation marks (" ").

    After you specify either `ignore` or `states`, specify one of the following:
    - `all` specifies that separate arcs should be created for each valid sensitization state.
    - `default` specifies that only a single default arc be created for the arc. The sensitization for the arc is determined by the `default_arc_mode` parameter setting.
    - `state ...` is a list of states (`when` conditions) for which arcs are to be created. Each `when` condition can be enclosed in double quotation marks.

Migrating to ncx_create_arcs

The ncx_create_arcs attribute has replaced the following attributes. If you use any attribute listed in this section, Liberty NCX issues a warning message saying that the attribute is deprecated in favor of ncx_create_arcs.

- ncx_default_delay_arcs_only

  Instead of setting ncx_default_delay_arcs_only to false to determine whether conditional delay arcs are created between input and output pin pairs, you should set ncx_create_arcs, as shown:

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

- ncx_default_preset_arcs_only

  Instead of setting ncx_default_preset_arcs_only to false to separate preset arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set ncx_create_arcs, as shown:

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

  When you set this attribute, all state arcs for all arcs are generated and optimized.

- ncx_default_clear_arcs_only

  The ncx_default_clear_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_clear_arcs_only to false to separate clear arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set ncx_create_arcs, as shown:

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

  When you set this attribute, all state arcs for all arcs are generated and optimized.

- ncx_default_constraint_arcs_only

  Instead of setting ncx_default_constraint_arcs_only to false to determine whether conditional constraint (setup and hold) arcs are created between input and output pin pairs, you should set ncx_create_arcs, as shown:

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

- ncx_default_min_pulse_width_arcs_only

  Instead of setting ncx_default_min_pulse_width_arcs_only to false to separate minimum pulse width arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set ncx_create_arcs, as shown:

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

When you set this attribute, all state arcs for all arcs are generated and optimized.

- `ncx_default_non_sequential_arcs_only`

  Instead of setting `ncx_default_non_sequential_arcs_only` to false to separate nonsequential arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set `ncx_create_arcs`, as shown:

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

  When you set this attribute, all state arcs for all arcs are generated and optimized.

- `ncx_leakage_power_mode`

  Instead of setting `ncx_leakage_power_mode` to specify whether to exhaustively simulate all possible leakage power states for a cell or to simulate just the first possible state, you should set `ncx_create_arcs`, as shown:

  For all possible leakage power states, specify

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

  For the first possible state, specify

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

- `ncx_leakage_power_when`

  Instead of setting `ncx_leakage_power_when` to explicitly specify the “state” conditions for which to acquire the internal leakage power for a specified cell, you should set `ncx_create_arcs`, as shown:

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

Examples

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

This command specifies that all internal power arcs for input pin A be synthesized for all cells with a name that begins with MC.

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

This specifies that all delay arc states be modeled between all pins for all cells with a name that begins with MX3.

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

This specifies that all `non_seq_setup` and `non_seq_hold` arc states be modeled between all pins for all cells with a name that begins with MD.

```
ncx_create_arcs : MD10SLL * NQ delay ignore all ;
```

This specifies that all delay arcs terminating in the pin NQ of the cell MD10SLL should be ignored.

```
ncx_create_arcs : MD10SLL * Q delay states default ;
```

This specifies that only a default delay arc is to be modeled for all paths terminating at the pin Q of the cell MD10SLL.

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

This specifies that the minimum pulse width only be acquired for the two specified states for the pin CK in all cells with a name that begins with MD1.

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 3-26 describes the attributes that can be used only in library template files, and Table 3-3 on page 3-31 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
`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`	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`	Voltage tolerance used to determine whether a signal has reached the rail voltage.	`0.005`
`ccs_segment_tol`	Maximum allowed voltage difference between the simulation waveform and the CCS waveform, used for selecting the CCS model point.	`0.005`

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

Attribute	Description	Default
clk_tran	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	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

constraint_monitor_node	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:
	
	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:
	
	`constraint_monitor_node : l1_tp217 ;`
	
	Liberty NCX uses NCX_MONITOR as an alias for the internal node to be monitored.
	
	2) You must add a pin group in the cell as follows:
	
	`pin NCX_MONITOR {`
	`  direction : output ;`
	`  internal_node : IQ;`
	`}`
	
	where IQ is the name of pin in the statetable that captures the internal node's (l1_tp217) functionality. If the original statetable does not have functionality for the internal node specified, a separate column called NCX_MONITOR should be created in the output section.	

constraint_total_output_net_capacitance	Allows you to set the 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.	50.0 ff

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

Attribute	Description	Default
constraint_total_ output_net_ capacitance_pct	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.	
side_out_total_ output_net_ capacitance	Allows you to set the output load of side output pins on a pin-by-pin basis. You can also use side_out_total_output_net_capacitance with the ncx_condition structures to specify side loads for different acquisitions. The values are specified in library units for capacitance. 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. Note: The side_out_total_output_net_capacitance attribute overlaps with the constraint_total_output_net_capacitance attribute for constraint arcs. Therefore, if both attributes are specified for constraint arcs, constraint_total_output_net_capacitance overrides side_out_total_output_net_capacitance.	50.0 ff

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

Attribute	Description	Default
default_arc_mode	Determines which state-dependent arc is chosen as the default (state-independent) arc, from a set of timing arcs between two pins with valid when conditions on the side-input pins. The first mode selects the first state-dependent arc encountered in the pin group, whereas the best or worst mode selects the state-dependent arc with the best or worst NLDM delay, respectively. The worst_delays and best_delays 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. The worst_points and best_points 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. The worst_edges and best_edges 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.	first
default_constraint_ arc_mode	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. 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. The worst_points and best_points options specify the constraint values that are the worst or best points (independent of each other) from all states at each index point.	first
default_spice_ option	Uses the SPICE option in the netlist for yes, or no such option for no.	yes

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

Attribute	Description	Default
fast_mode	Performance enhancement mode for sequential cell characterization that uses saved initial conditions to minimize multicyle 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 sim_opt, 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.	true
init_cycle	The number of initialization cycles required for timing measurement.	0
input_cap_mode	Determines the mode of calculation of the NLDM input pin capacitance on a pin: average, min, or max. The value is calculated from the rise/fall capacitance values obtained for the various input slew values used for timing characterization (see the descriptions of the following attributes).	min
input_cap_mode_ fall	Determines the calculation of the NLDM input pin capacitance on a pin for a falling waveform transition: average, min, or max. 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: average, min, or max. The value is calculated from the capacitance values obtained for the various input slew values used for timing characterization.	average
min_pulse_width_ mode	Extracts the best or worst pin-based min_pulse_width values from the simulated results with side pin combinations. Set the attribute to min or max 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 min_pulse_width_slew, min_pulse_width_slew_high, min_pulse_width_slew_low, and constraint_total_output_net_capacitance attributes in the library or cell template file.	

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

Attribute	Description	Default
`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`
`min_pulse_width_slew_low`	Allows you to specify the input slew value used in the characterization of the `min_pulse_width_high` and `min_pulse_width_low` pin attributes. If this attribute is used, it overwrites the `min_pulse_width_slew` value. This attribute affects only pin-based minimum pulse width, not arc-based minimum pulse width.	`0.1 ps`
`min_pulse_width_slew_high`	Allows you to specify the input slew value used in the characterization of the `min_pulse_width_high` and `min_pulse_width_low` pin attributes. If this attribute is used, it overwrites the `min_pulse_width_slew` value. This attribute affects only pin-based minimum pulse width, not arc-based minimum pulse width.	`0.1 ps`
`ncx_cell_leakage_power_mode`	Determines how the `cell_leakage_power` attribute value in a cell is calculated from the state-dependent values in the `leakage_power` models. Valid values are `min`, `max` (the default) or `avg`.	`max`
`ncx_condition`	Allows you to specify an NCX condition, which consists of the following components: – A set of characterization criteria, with a string `ncx_condition_`*`prefix`*, 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 `ncx_harness` syntax, which is used when Liberty NCX characterizes an arc that matches the NCX condition characterization criteria. For more information, see “Conditional Characterization” on page 3-74.	none

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

Attribute	Description	Default

ncx_condition_output_toggle	Use ncx_condition_output_toggle to look for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for toggled and non-toggled arcs. The values are true and false. Use ncx_condition_output_toggle with ncx_condition, for example:
	
	`ncx_condition vio_toggle_arc {`
	`  ncx_condition_arc_type : constraint ;`
	`  ncx_condition_output_toggle : true ;`
	`  violation_mode : delay_degrade_pct delay_degrade ;`
	`}`
	
	For more information, see "Toggled And Non-toggled Arcs" on page 3-77.
		true

ncx_constraint_search_interval	If characterization provides constraint numbers that seem too large or if there are bisection errors during constraint characterization, use ncx_constraint_search_interval to control the accuracy of the constraint numbers. Liberty NCX runs a simulation to determine the nominal or the reference delay. The setup time used to find the nominal delay is 1ns, which is half of the ncx_constraint_search_interval default value. If a 1ns setup time is too small to determine the nominal delay, you can set the ncx_constraint_search_interval value as needed. The value should be a floating-point number in library time units.	2 ns
ncx_constraint_accuracy	If characterization provides constraint numbers that seem too large or if there are bisection errors during constraint characterization, use ncx_constraint_accuracy to control the accuracy of the constraint numbers. The ncx_constraint_accuracy attribute determines the end of the bisection search during constraint characterization. Liberty NCX stops the bisection search when the bisection search interval is less than 2 ps. You can define the attribute as needed.	2 ps

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

Attribute	Description	Default
ncx_default_ conditional_ receiver_mode	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 when conditions on the side-input pins. The best and worst mode selects the state-dependent arc with the best or worst receiver capacitance, respectively. The best_points and worst_points options specify the receiver capacitance with the best or worst points (independent of each other) for the respective states at each index point.	none
ncx_default_ constraint_arcs_only	The ncx_default_constraint_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_constraint_arcs_only to false to determine whether conditional constraint (setup and hold) arcs are created between input and output pin pairs, you should set ncx_create_arcs, as shown: `ncx_create_arcs :  * * * constraint states all` If you use ncx_default_constraint_arcs_only, Liberty NCX issues a warning message saying that ncx_default_constraint_arcs_only is deprecated in favor of ncx_create_arcs. For more information, see “Arc Synthesis Control” on page 3-3.	
ncx_default_ delay_arcs_only	The ncx_default_delay_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_delay_arcs_only to false to determine whether conditional delay arcs are created between input and output pin pairs, you should set ncx_create_arcs, as shown: `ncx_create_arcs : * * * delay states all` If you use ncx_default_delay_arcs_only, Liberty NCX issues a warning message saying that ncx_default_delay_arcs_only is deprecated in favor of ncx_create_arcs. For more information, see “Arc Synthesis Control” on page 3-3.	

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

Attribute	Description	Default
ncx_default_arc_mode	If you are generating a new library from a set of cell templates, by default, Liberty NCX creates state-independent arcs (without a when condition) between pins, where applicable. The selection of side-pin conditions to use for characterization depends on ncx_default_arc_mode, which can be set to first, best, or worst. The first mode selects the first set of conditions encountered. The best or worst mode selects the arc having the best or worst maximum delay value. If you are recharacterizing a library, if the pin has only a single state-independent arc, then the behavior is also determined by ncx_default_arc_mode as described above. 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 default_arc_mode setting, which also can be first, best, or worst.	first
ncx_default_non_sequential_arcs_only	The ncx_default_non_sequential_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_non_sequential_arcs_only to false to separate nonsequential arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set ncx_create_arcs, as shown: `ncx_create_arcs :  * * * non_seq* states all` If you use ncx_default_non_sequential_arcs_only, Liberty NCX issues a warning message saying that ncx_default_non_sequential_arcs_only is deprecated in favor of ncx_create_arcs. For more information, see “Arc Synthesis Control” on page 3-3.	

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

Attribute	Description	Default
ncx_default_min_ pulse_width_arcs_ only	The ncx_default_min_pulse_width_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_min_pulse_width_arcs_only 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 ncx_create_arcs, as shown: `ncx_create_arcs : * * * min_pulse_width states all` If you use ncx_default_min_pulse_width_arcs_only, Liberty NCX issues a warning message saying that ncx_default_min_pulse_width_arcs_only is deprecated in favor of ncx_create_arcs. For more information, see “Arc Synthesis Control” on page 3-3.	
ncx_default_preset_ arcs_only	The ncx_default_preset_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_preset_arcs_only to false to separate preset arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set ncx_create_arcs, as shown: `ncx_create_arcs : * * * preset states all` If you use ncx_default_preset_arcs_only, Liberty NCX issues a warning message saying that ncx_default_preset_arcs_only is deprecated in favor of ncx_create_arcs. For more information, see “Arc Synthesis Control” on page 3-3.	

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

Attribute	Description	Default
ncx_default_clear_arcs_only	The ncx_default_clear_arcs_only attribute has been replaced by the ncx_create_arcs attribute. Instead of setting ncx_default_clear_arcs_only to false to separate clear arcs from constraint arcs (setup, hold, recovery, and removal arcs) and obtain the arc states, you should set ncx_create_arcs, as shown: `ncx_create_arcs :  * * * clear states all` If you use ncx_default_clear_arcs_only, Liberty NCX issues a warning message saying that ncx_default_clear_arcs_only is deprecated in favor of ncx_create_arcs. For more information, see "Arc Synthesis Control" on page 3-3.	false
ncx_add_default_conditional_receiver_model	Controls the addition of default receiver capacitance arcs (arcs that do not contain a when condition) when Liberty NCX synthesizes receiver capacitance arcs from a template description of a cell. If set to true, 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 ncx_add_default_conditional_receiver_model 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 false.	false
ncx_add_default_internal_power	Specifies the addition of default internal_power arcs on output pins that have multiple state-dependent internal_power groups. Valid values for this attribute are min, max, or avg (the default). The values determine how the table points for the default power arc are selected.	avg
ncx_internal_power_mode	Specifies whether to exhaustively simulate all possible nonpropagating states for a cell (all) or just the first possible state (first). This can be specified at the library, cell, or pin level.	first

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

Attribute	Description	Default
`ncx_add_default_constraint_arcs`	Controls the addition of default timing arcs (arcs that do not contain a “when” condition) when Liberty NCX synthesizes timing arcs from a template description of a cell. If set to true, 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. The `ncx_add_default_constraint_arcs` 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 false.	`false`
`ncx_add_default_delay_arcs`	Controls the addition of default timing arcs (arcs that do not contain a “when” condition) when Liberty NCX synthesizes timing arcs from a template description of a cell. If set to true, 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. The `ncx_add_default_delay_arcs` 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 false.	`false`
`ncx_leakage_power_mode`	This attribute has been replaced by `ncx_create_arcs`. For more information, see “Arc Synthesis Control” on page 3-3.	
`ncx_min_setup_based_delay`	When you set the `ncx_min_setup_based_delay` 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. For more information, see “Minimum Setup Time for Pessimistic Delay” on page 3-63.	`false`

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

Attribute	Description	Default
ncx_min_setup_based_delay_constraint_slew	When you use ncx_min_setup_based_delay 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 ncx_min_setup_based_delay_constraint_slew attribute. The attribute specifies the constraint pin slew value from the delay table. The values are middle, min, and max. For more information, see "Minimum Setup Time for Pessimistic Delay" on page 3-63.	middle
ncx_min_setup_based_delay_setup_time_offset	When you set the ncx_min_setup_based_delay_setup_time_offset attribute, Liberty NCX adds a margin to the setup time used for characterizing the delay and output transition time. The default value is 1e-12 seconds. For more information, see "Minimum Setup Time for Pessimistic Delay" on page 3-63.	1e-12 seconds
ncx_mpw_rail_to_rail	When you set the ncx_mpw_rail_to_rail attribute to false, Liberty NCX reports input pulses that might not swing between rails. To ensure that Liberty NCX writes out minimum pulse width numbers that correspond to full swing pulses, set the ncx_mpw_rail_to_rail attribute to true, the default. When you do this, Liberty NCX bounds the minimum pulse width value to where the pulse represents a full rail swing pulse.	true
ncx_pin_capacitance_ranges	Suppresses the generation of NLDM pin capacitance ranges in the output library. To suppress the generation of NLDM pin capacitance ranges, set ncx_pin_capacitance_ranges to false. If the attribute is not set, or if it is set to true, the pin capacitance ranges are published.	true
ncx_input_power_period_scale	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 ncx_output_power_period_scale attribute. If the ncx_input_power_period_scale and ncx_output_power_period_scale attributes are defined anywhere in the template, they override the ncx_power_period_scale attribute.	

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

Attribute	Description	Default
`ncx_output_power_period_scale`	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 `ncx_input_power_period_scale` attribute. If the `ncx_input_power_period_scale` and `ncx_output_power_period_scale` attributes are defined anywhere in the template, they override the `ncx_power_period_scale` attribute.	
`ncx_conditional_receiver_model`	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 `when` conditions, as specified by the `ncx_conditional_receiver_model_states` attribute. If it is set to false, Liberty NCX generates a single pin receiver model. See also the `ncx_conditional_receiver_model_states` attribute in Table 3-3.	`false`
`ncx_node_set`	Defines 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, and then specify `violation_mode` for the sets. For more information, see "Constraint Methodologies" on page 3-86.	
`ncx_use_library_sensitization`	Specifies whether to use or ignore the sensitization information in the input library. If the attribute is set to `true`, Liberty NCX sensitizes the cell according to what is contained in the input library. If it is set to `false`, Liberty NCX determines its own sensitization for the cell.	`true`

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

Attribute	Description	Default
ncx_optimize_ state_power_arcs	Merges nonpropagated power arcs. This reduces the output library file size. The ncx_optimize_state_power_arcs tolerance is set by the ncx_optimize_state_power_arcs_tol_pct 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 ncx_optimize_state_power_arcs_tol_pct, are merged into single power models with when conditions that capture all the states. The ncx_optimize_state_power_arcs attribute is set to false by default.	false
ncx_optimize_state_ timing_arcs	Optimizes the characterization and modeling of multiple-state timing arcs between two pins. If the value of ncx_optimize_state_timing_arcs is set to merge, states with overlapping behaviors (where the model values are very close to each other as determined by the value of ncx_optimize_state_timing_arcs_tol_pct) are merged into single timing models with “when” conditions that capture all the states. This reduces the output library file size. Set ncx_optimize_state_timing_arcs to true if you are characterizing MUXs.	false
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
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 1.0.	1.0
power_exclude_pin	Set the power_exclude_pin 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: power_exclude_pin : "VDDC VSSC" ; Setting the attribute changes the NLPM leakage power and internal power calculation. There is no change in CCS power.	none

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

Attribute	Description	Default
setup_hold_method	Selects a setup and hold pessimism reduction (SHPR) setup and hold pair to specify the conventional setup and hold time in a library. 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.	minimum
sim_inc_file	Allows you to specify an “include” file in the netlist. The string of this option is included by using the command .inc.	
sim_opt	Allows you to specify 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	
three_state_pullup_res three_state_pulldn_res	Use the three_state_pullup_res and three_state_pulldn_res attributes to specify the pull-up and pull-down resistor values that Liberty NCX uses when characterizing a three-state cell. 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.	10000 ohm
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.	none

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

Attribute	Description	Default
`slew_lower_ threshold_pct_fall`	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_upper_ threshold_pct_fall`	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_lower_ threshold_pct_rise`	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_rise`	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_predriver_ ratio`	A floating-point number between 0 and 1 that specifies the relative weighting of the ramp waveform component in the Synopsys predriver waveform. 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`
`tran_timestep`	Transient simulation time step, in seconds.	`100e-12`
`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 true, the switching energy is calculated as `0.5 *C*V2` and subtracted from the total measured energy to obtain the internal energy of the cell. If it is set to false, the switching power is calculated simply as `C*V2` and subtracted from the total measured energy.	`true`
`violation_delay_ degrade_pct`	Used for delay-sensitive violation (violation_pass_fail_mode = false). Liberty NCX finds the setup/hold time such that the output delay is degraded by at least the specified amount (10 percent by default).	`10`
`violation_delay_ degrade`	This attribute specifies an absolute delay degradation, in seconds, rather than a percentage change. For example, setting `violation_delay_degrade` to 25e-12 causes Liberty NCX to use a delay degradation of 25 picoseconds to determine setup and hold violations. A `violation_delay_degrade` setting overrides any `violation_delay_degrade_pct` setting. There is no default setting.	none

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

Attribute	Description	Default
violation_glitch_ mode	The violation_glitch_mode attribute has been replaced by the violation_mode attribute. For more information, see "Constraint Methodologies" on page 3-86.	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.	none
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.	none
violation_pass_ fail_mode	The violation_pass_fail_mode attribute has been replaced by the violation_mode attribute. For more information, see "Constraint Methodologies" on page 3-86.	false
violation_mode	The violation_mode attribute considers an absolute and a relative degradation simultaneously. When you set violation_mode to delay_degrade_pct delay_degrade, Liberty NCX monitors all output pins with both relative and absolute delay degradation. Setting violation_mode to glitch enables glitch detection mode. Setting violation_mode to passfail performs pass/fail violations. You can also use violation_mode to monitor single node or multiple nodes. For more information, see "Constraint Methodologies" on page 3-86.	
violation_mode_rise	Use violation_mode_rise to look for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for rising arcs only.	

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

Attribute	Description	Default
violation_mode_fall	Use violation_mode_fall to look for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for falling arcs only.	
zstate_leak_ threshold_pct	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	The exact name of the ground node used in the active driver subcircuit.	
active_drv_input	The exact 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 snps_predriver (the standard Synopsys predriver waveform) or ramp (an ideal linear ramp).	snps_ predriver
active_drv_netlist	The full name of the netlist containing the subcircuit description of the active driver cell.	
active_drv_output	The exact name of the output node of the active driver, as defined in the subcircuit interface.	
active_drv_supply_ node	The exact name of the supply node that powers the active driver subcircuit.	
active_drv_slew	This is the value of input slew (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 active_drv_slew_tol_pct is 10.	10

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

Attribute	Description	Default
active_drv_unate	This option should be set to negative if the active driver subcircuit inverts the input waveform, or positive otherwise.	
active_drv_supply_voltage	The value of the supply voltage used to power the active driver subcircuit.	
active_drv_ground_voltage	The value of ground voltage used for the active driver subcircuit.	
do	Specifies a list of cells to include in the characterization flow. This option, when used with the dont option, allows incremental characterization of cells. When you start with a seed library, the do list can be used to specify additional cells to be added to the output library. When you create a library from scratch, the do list specifies all the constituent cells of the new library. A cell template file must exist for each of the cells in the do list.	
dont	Specifies a list of cells to exclude from the characterization flow. This option allows incremental characterization of cells.	
driver_model	Specifies the type of driver to be used at the cell inputs for characterization: ramp, snps_predriver, or active_driver. ramp is an ideal, linear ramp with a specified transition time. snps_predriver is the standard Synopsys predriver waveform generated from the specified transition time, which represents a more realistic driving waveform. active_driver 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. For more information, see the following attributes: active_drv_unate active_drv_supply_voltage active_drv_ground_voltage	snps_predriver
global_vdd	Specifies the name of the global power pin used in the netlist.	
global_vss	Specifies the name of the global ground pin used in the netlist.	

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

Attribute	Description	Default
`ncx_create_arcs`	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. 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. Note: The `ncx_leakage_power_when` attribute is no longer supported. The functionality of `ncx_leakage_power_when` is now available in `ncx_create_arcs`. For more information, see "Arc Synthesis Control" on page 3-3.	default arcs only
`ncx_default_` `power_arcs_only`	Specifies whether to generate default power arcs during power characterization. A setting of `true` cases default power arcs to be generated; `false` prevents them from being generated. A default power arc is a power group without any "when" conditions, which specifies the power value when all the "when" conditions for power are false (see the description of the cell-level attribute `ncx_internal_power_when`). In the absence of a default power arc, the power value is assumed to be zero when all specified "when" conditions are false.	`false`

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

Attribute	Description	Default
ncx_internal_power_ calculation	If the leakage_model_file attribute is defined, the ncx_internal_power_calculation attribute lets you specify whether the leakage used in the internal power adjustment should be characterized using a timing SPICE model. Valid values are default_model and leakage_power_model. If you specify default_model, Liberty NCX characterizes an extra leakage SPICE deck using a timing SPICE model for the internal_power adjustment. If you specify leakage_power_model, the internal power adjustment uses leakage power that is characterized based on a leakage SPICE model. Note: If leakage_model_file is not defined, the ncx_internal_power_calculation attribute is ignored.	default_ model
ncx_nlpm_mode	Models 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 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 power tables are not modeled for each supply pin. For more information, see “Power and Ground Pin Connections” on page 3-45.	aggregate

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

Attribute	Description	Default
ncx_use_pg_pins	The ncx_use_pg_pins attribute supports the UPF format. The ncx_use_pg_pins attribute is enabled (set to true) by default, meaning that Liberty NCX adds pg_pin and voltage_map syntax to all libraries during characterization. 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. Note: CCS power characterization requires a power format based on the power and ground pin syntax. Therefore, if you set the ccs_power command in the command file to true, enabling ccs_power, Liberty NCX ignores the ncx_use_pg_pins setting and uses the pg_pin format for power modeling. For more information, see "Power and Ground Pin Connections" on page 3-45.	true
side_pin_driver_model	Specifies the type of driver to be used at the side input pins for characterization. The values are ramp or active_driver, where ramp is an ideal, linear ramp with a specified transition time and active_driver 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. The side_pin_driver_model attribute is supported with the HSPICE simulator only.	ramp
va_parameters	Specifies a list of variation-aware library parameter names. The generated library contains cells characterized at different values of these parameters.	
va_nominal_values	Specifies a list of nominal values corresponding to the parameter names specified by va_parameters.	
va_variation_values	Specifies a list of absolute parameter values corresponding to the parameter names specified by va_parameters. These are the parameter values at which cells are characterized, either higher or lower than the nominal values specified by va_nominal_values.	

Table 3-3 Attributes in Cell Template Files

Attribute	Description	Default
differential_pair	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.	
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.	
input_fall_ threshold	The timing threshold on input falling edge, in volts.	1.5
input_rise_ threshold	The timing threshold on input rising edge, in volts.	1.5
input_signal_ level_high input_signal_ level_low	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.	
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.	

ncx_clear	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:
	`pin R {`
	`direction : input;`
	`ncx_clear : L;`
		none

ncx_preset	Specifies a preset pin. Set ncx_preset 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;`
		none

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

Attribute	Description	Default
simultaneous_ switching	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. Note: The simultaneous_switching attribute is not recorded in the output library. For more information, see "Simultaneous Switching" on page 3-48.	none
ncx_internal_ power_when	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: `ncx_internal_power_when : (A*B) (A*B')` 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.	
ncx_internal_ leakage_when	This attribute has been replaced by the ncx_create_arcs attribute. For more information, see "Arc Synthesis Control" on page 3-3.	

	The following attributes control NLDM capacitance extraction:
	
	`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`
	
	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. The subset of indexes must be a subset of the indexes used for timing modeling. For more information, see "Input NLDM Capacitance Index Values" on page 3-43.
		none

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

Attribute	Description	Default
`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`	none
`ncx_condition_cell_name`	Specifies the use of the condition for a specific cell. The value is the name of the cell.	none
`ncx_condition_cell_type`	Specifies the use of the condition for a specific cell type and can assume one of the following values: • `sequential` • `combinational` • `flipflop` • `latch` • `clock_gating`	none
`ncx_condition_pin`	Specifies the use of the condition for all arcs that terminate at the named pin.	none
`ncx_condition_related_pin`	Specifies the use of the condition for all arcs that originate from the named pin.	none
`related_output_pin`	Liberty NCX supports the `related_output_pin` attribute inside the `ncx_condition` block. When you specify `related_output_pin` in the cell template file, you can use it inside the `ncx_condition` block to determine which output pin should be monitored for arcs that potentially have multiple output pins. For an example using an *n*-bit shift register, see "Conditional Characterization" on page 3-74.	none
`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.	none
`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.	none

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

Attribute	Description	Default

ncx_when	Specifies a side-pin sensitization during library characterization. This condition is not published in the library that is generated. In addition, if a when condition is present, ncx_when is appended to it during characterization but will not modify it in the library.
	
	In the following syntax, Liberty NCX matched all arcs whose when condition is "A." And to those arcs, it adds the condition "!scan" for characterization.
	
	`ncx_condition set_when {`
	`  ncx_condition_when : "A" ;`
	`  ncx_when : "!scan" ;`
	`}`
		none

ncx_when_rise	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.	
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 when condition is present, ncx_when_fall is appended to it during characterization but does not modify it in the library.	
ncx_conditional_ receiver_model_ states	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. If you set the following, `ncx_conditional_receiver_model_states : "CK" "!CK";` Liberty NCX generates pin receiver models for CK equal to 0 and CK equal to 1. See also the ncx_conditional_receiver_model attribute in Table 3-1.	

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

Attribute	Description	Default
The following four attributes can be used to determine the maximum output capacitance for a cell. All of these parameters must be specified for automatic acquisition of maximum output capacitance:		
ncx_max_output_ transition_time	The maximum output transition time (in seconds) desired for an output pin.	
ncx_min_total_ output_net_ capacitance	The minimum load capacitance (in farads) for an output pin.	
ncx_mode_total_ output_net_ capacitance	The variation of load capacitance between the specified minimum value and the acquired maximum value. This may be one of log or linear.	
ncx_size_total_ output_net_ capacitance	The number of load capacitance index points desired.	
ncx_optimize_state _leakage_power	Set the ncx_optimize_state_leakage_power attribute to merge leakage power automatically. 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	Set the ncx_optimize_state_leakage_power_mode attribute to merge leakage power automatically. 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	Set the ncx_optimize_state_leakage_power_tol attribute to merge leakage power automatically. When you set the attribute, all values less than the ncx_optimize_state_leakage_power_tol tolerance value are merged.	none
ncx_optimize_state _leakage_power_tol _pct	Set the ncx_optimize_state_leakage_power_tol_pct attribute to merge leakage power automatically. 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%
non_rail_output_ signal_level	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.	

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

Attribute	Description	Default
output_fall_ threshold	The timing threshold on output falling edge.	1.5
output_rise_ threshold	The timing threshold on output rising edge.	1.5
ref_state	The constant state of a side pin, either 0 or 1.	
input_signal_level	The constant voltage applied to an input pin.	
sensitization	Specifies the sensitization of the cell, using the sensitization syntax.	
simple_ termination_ netlist	The path of termination subcircuit netlist.	
simple_ termination_pin	The cell pin that will be attached to a termination circuit.	
test_output_only	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 test_output_only to true to enable this feature. You must specify the loads for the test_output_only pin.	false

Table 3-4 is a summary listing of the attributes that have default settings.

Table 3-4 Attributes With Default Settings

Attribute	Default
clk_tran	1e-11
clk_width	1e-9
default_arc_mode	first
default_spice_option	yes
driver_model	snps_predriver

Table 3-4 Attributes With Default Settings (Continued)

Attribute	Default
fast_mode	true
init_cycle	0
input_cap_mode	min
input_cap_mode_fall	average
input_cap_mode_rise	average
ncx_default_constraint_arcs_only	true
ncx_default_power_arcs_only	false
ncx_default_timing_arcs_only	true
ncx_internal_power_mode	first
ncx_leakage_power_mode	first
ncx_use_library_sensitization	true
slew_lower_threshold_pct_fall	20.0
slew_upper_threshold_pct_fall	80.0
slew_lower_threshold_pct_rise	20.0
slew_upper_threshold_pct_rise	80.0
snps_predriver_ratio	0.5
tran_timestep	100e-12
violation_pass_fail_mode	false
violation_delay_degrade_pct	10
zstate_leak_threshold_pct	10

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, 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 User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries*.

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 3-41.

Minimum/Maximum/Mode 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 or log). 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 the foregoing 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 (linear or log) must all be available for each parameter.

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.

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 command 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`. The subset of indexes must be a subset of the indexes used for timing modeling.

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,

```
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 while 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.

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` command in the command 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.

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 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:

```
simultaneous_switching : 2 S1 S2 S3 ;
```

where 2 is the number of pins that are switching.

The simultaneous_switching attribute is not recorded in the output library.

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.

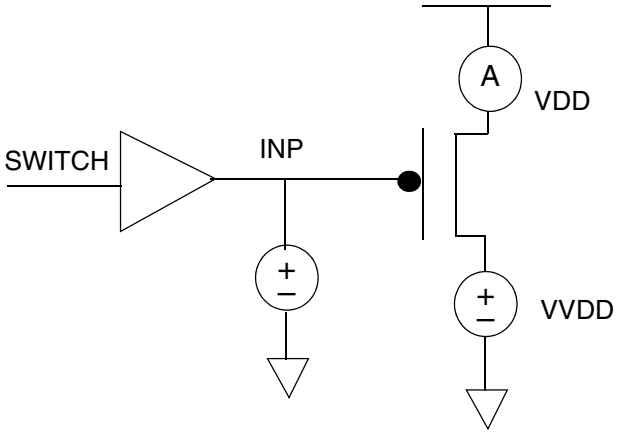
Multithreshold-CMOS Characterization

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 `power` 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 3-1, 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 `timing` to true in the configuration file.

Figure 3-1 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>;
   }
}

 ex :
    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 `negantive_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" {
```

Sensitization

To set up each circuit simulation 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.

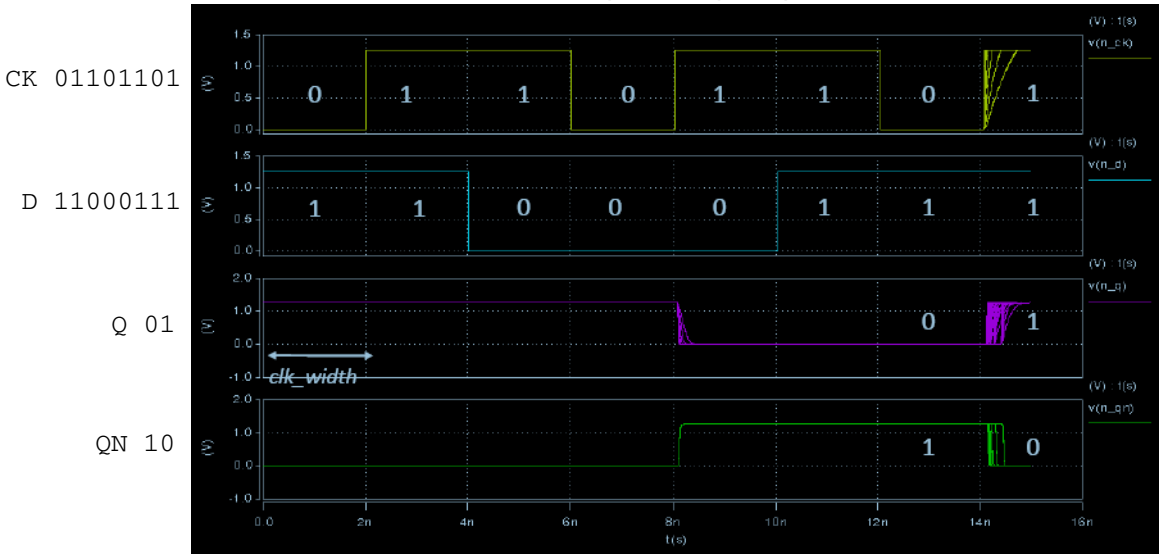
The existing input library might contain cell sensitization information. By default, Liberty NCX uses any such information to sensitize each cell. Cell sensitization generates valid sensitization vectors for the cell from the functional descriptors of the cell, such as the Boolean function on the output pin, the flip-flop latch group information, and the information in the statetable or truth table. If you prefer to have Liberty NCX determine its own sensitization, 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 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.

Use the `ncx_wave_rise` and `ncx_wave_fall` attributes to specify timing-arc sensitization vectors for rising and falling arcs. The value is a user-specified string. 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 ;
```

Example

The following excerpt from an example shows the 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_fall : \
         D 00111100 \
         CK 01101001 \
         Q 10 ;
      ncx_wave_rise : \
         D 11000011 \
      CK 01101001 \
      Q 01 ;
   }
   :
}
pin CK {
   direction : input ;
   clock : true ;
```

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

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 3-5.

Table 3-5 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/hold, recovery/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. Non-propagating 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.

Non-propagating 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 ;
}
```

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, 1, and -. The allowed characters for output pin values are 0, 1, Z and -.

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, 0  ;
…
}
```

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

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:

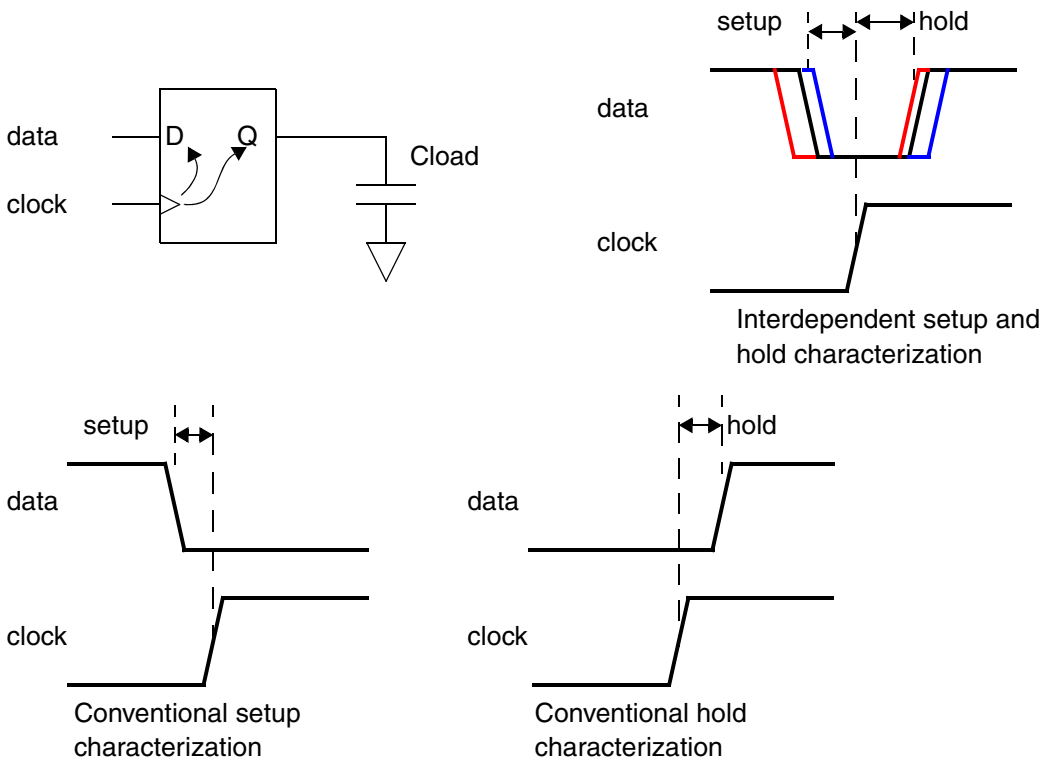
- Interdependent Setup and Hold
- Minimum Setup Time for Pessimistic Delay
- Differential Outputs and Inputs
- Nonrail Voltage Swing
- Constant Logic States on Inputs
- Pin-Specific Timing Thresholds
- Initialization Cycle
- Synchronizer Circuit
- Custom Harness

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 3-3.

Figure 3-3 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 3-3.

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 command 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.

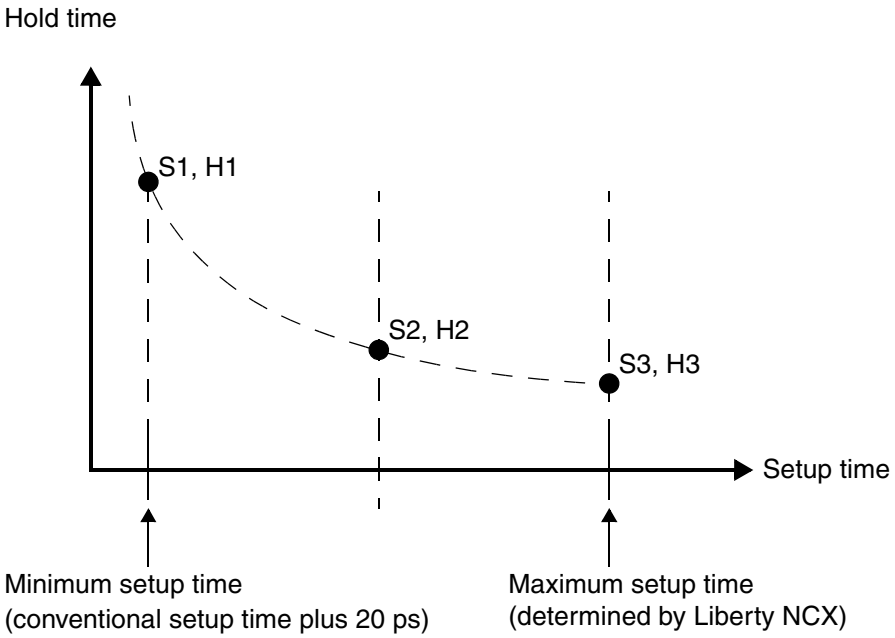
By default, Liberty NCX generates three setup/hold pairs of values. Experiments have shown that at least three pairs of values are required for an accurate representation, and three pairs typically give good results.

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/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/hold data points, it divides the range from the minimum setup time up 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 3-4.

Figure 3-4 Default Choice of Interdependent Setup Times



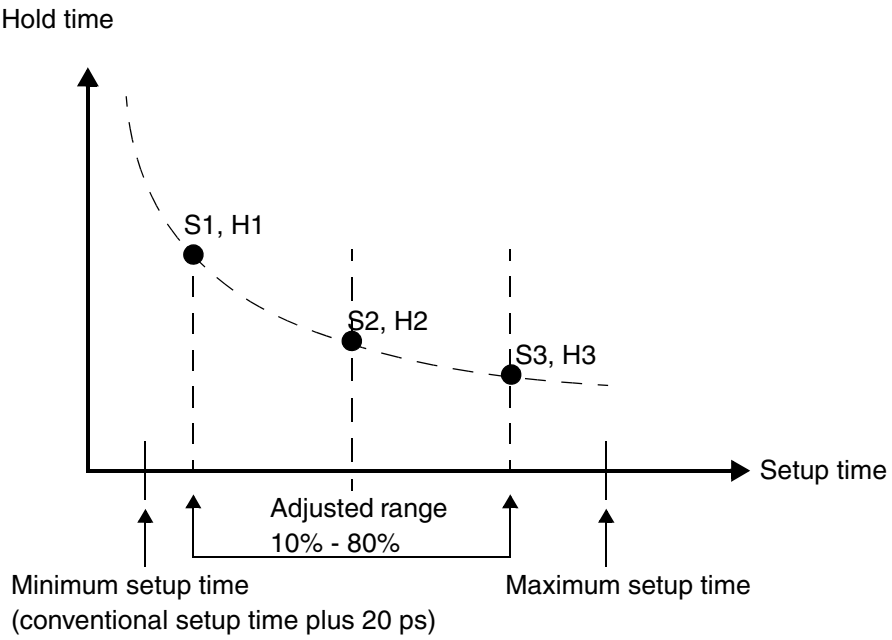
User Customization

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 3-5.

Figure 3-5 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 `ncx_min_setup_based_delay` 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 that is 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_delay_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 will cause a characterization failure.

Differential Outputs and Inputs

If a cell has differential output pins, the functional relationship between the differential pins in each pair should be declared with the `differential_pair` attribute. The definition consists of the keyword followed by the names of the two pins, separated by a space. For example,

```
differential_pair : "PADP PADN" ;
```

The names of the two pins must be enclosed in quotation marks. For a pair of differential output pins, instead of measuring the point where each output crosses a fixed delay threshold, Liberty NCX measures the point at which the two output values cross each other.

If a cell has differential input pins, the functional relationship between the differential pins in each pair should be declared with the `pin_opposite` attribute. For example,

```
pin_opposite : in1 in2 ;
```

This type of declaration does not use quotation marks because it is a complex attribute in Liberty format. For a pair of differential input pins, the two input signals are always logically inverted with respect to each other. This affects the sensitization of the cell during characterization.

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
```

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:

```
output_signal_level_high : voltage
output_signal_level_low : voltage
```

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
```

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 the 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 the cell:

```
init_cycle : N
```

Example:

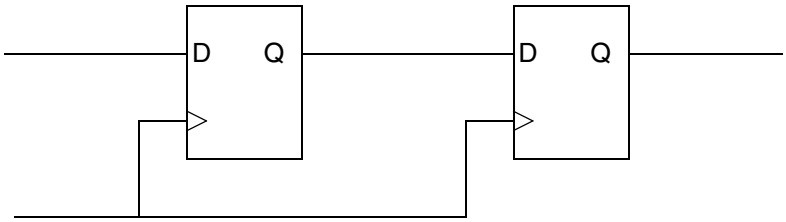
```
init_cycle : 2 ;
```

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 3-6. 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 3-6 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 ;
}
```

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:

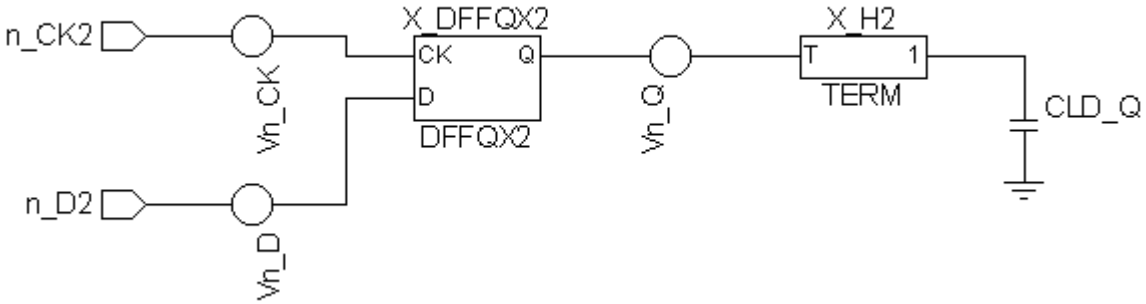
- CLD_Q : output load capacitor at node Q
- Vn_CK : response monitor (ammeter) for input node CK
- Vn_D : response monitor (ammeter) for input node D
- Vn_Q : response monitor (ammeter) for output node Q

External Termination Harness

The following harness attaches an external termination to the output pin Q of the test cell.

```
ncx_harness H2 {
  harness_netlist_file : term.spc ;
  pin (Q) {
    harness_connect_pin : T ;
    harness_load_pin : 1 ;
  }
}
```

Figure 3-7 External Termination Harness

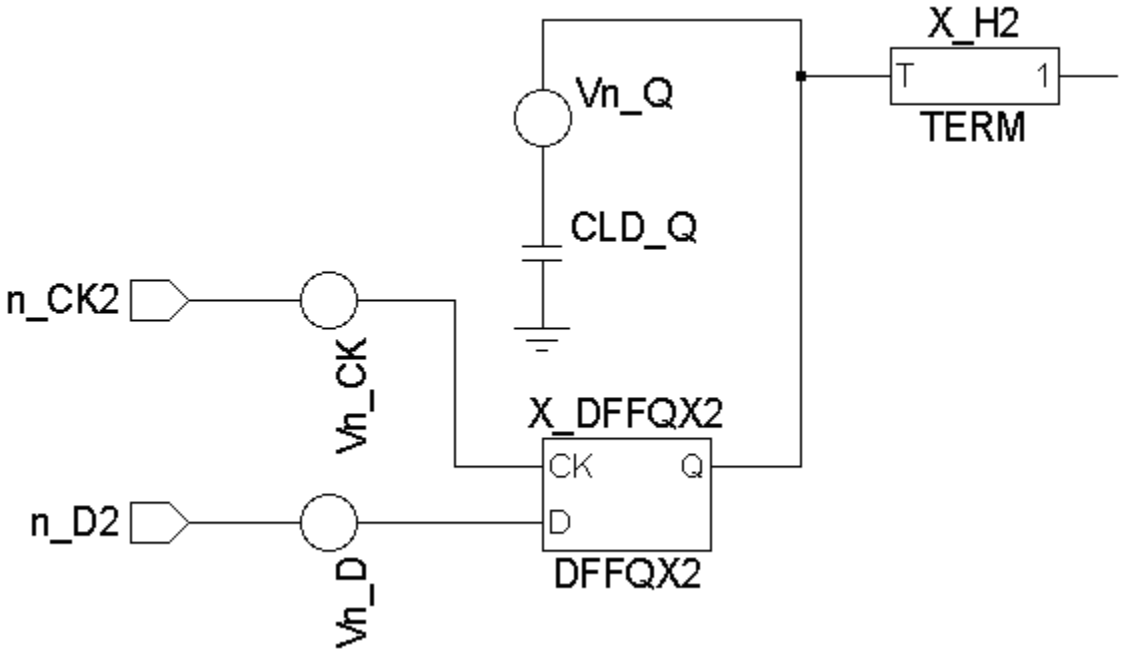


Termination and Measurement Harness

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.

```
ncx_harness H2 {
  harness_netlist_file : term.spc ;
  pin (Q) {
    harness_connect_pin : T ;
  }
}
```

Figure 3-8 External Termination and Measurement Harness



Driver, Termination, and Measurement Harness

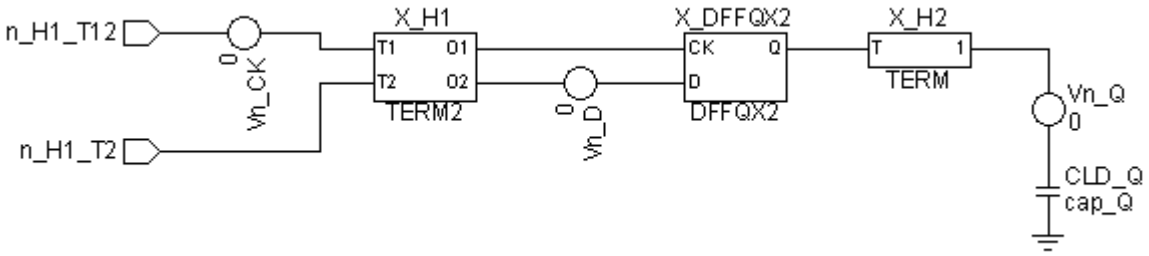
The following harness has the output pin termination of the previous example. It also drives the test cell inputs and measures the input response at the input pin T1 of the harness for the cell pin CK, and also at the cell pin itself for the second cell pin D.

```
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 3-9 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 pin PAD 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 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 3-10 PAD as an Input Pin

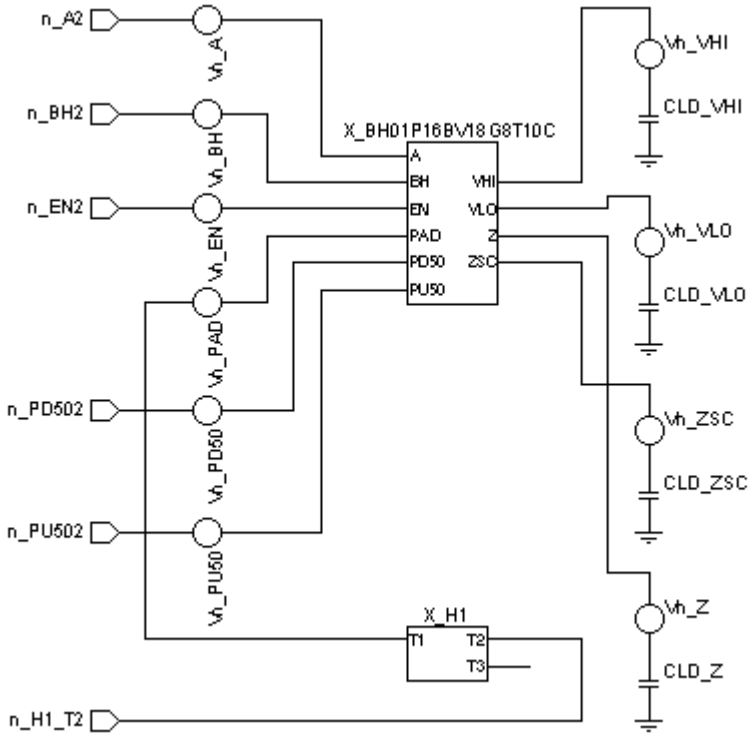
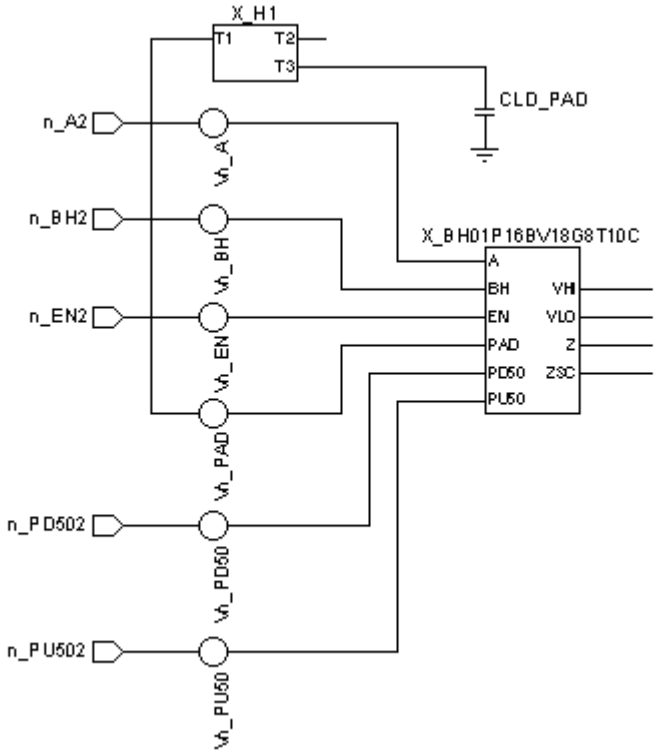


Figure 3-11 PAD as an Output Pin



Conditional Characterization

NCX conditions allow you to customize the characterization of certain arcs and models in a library. You can vary the conditions by specifying certain criteria in the condition definition.

NCX conditions can be specified by setting `ncx_condition` in the library or cell template file. The definition of an NCX condition consists of the following components:

- A set of characterization criteria, with a string `ncx_condition_` *`prefix`*, that represent the conditions to match when Liberty NCX determines NCX condition usage. The scope of the condition is largely determined by the number and type of matching criteria that you specify.
- An optional set of Liberty or NCX characterization timing arc attributes that represent your custom characterization. When an arc matches the characterization conditions of an NCX condition, these attributes are copied over from the NCX condition, overwriting any existing definitions.
- An optional `harness` group, specified using `ncx_harness` syntax, which is used when Liberty NCX characterizes an arc that matches the NCX condition characterization criteria. If an arc matches the characterization criteria of an NCX condition, the entire harness group is copied over from the NCX condition as if the group was entered in the arc group in the cell template file.

Note:
You can use wildcard characters in any `ncx_condition` field.

Syntax

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

The user-defined `attribute` can be any valid Liberty keyword such as `when` or `timing_type`.

In addition to the standard Liberty keywords for use in the condition characterization criteria, Liberty NCX also provides the following additional criteria to enable greater flexibility in condition usage:

`ncx_condition_cell_type`

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

- `sequential`

- `combinational`
- `flipflop`
- `latch`
- `clock_gating`

`ncx_condition_cell_name`

Specifies the use of the condition for a specific cell. The value is the name of the cell.

`ncx_condition_pin`

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

`ncx_condition_related_pin`

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

`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`

`ncx_condition_`*`valid_arc_descriptor`*

Specifies the use of the condition with a specified arc. A valid arc descriptor is any valid Liberty attribute that can be specified in an arc group. For example, you can specify `timing_type` or `timing_sense` to match the condition to a specific arc timing type or timing sense.

`ncx_condition_output_toggle`

Looks for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for toggled and non-toggled arcs. The values are true and false. For more information, see “Toggled And Non-toggled Arcs” on page 3-77.

To set the HSPICE `runlvl=5` option or all constraint arcs, define the following NCX condition in the library template file:

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

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 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` block. 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  ;
}
```

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 below:

```
  :
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)
 :
```

You can also use `violation_mode_rise` and `violation_mode_fall` with `ncx_condition` to apply `violation_mode` specifically to rising or falling arcs, as shown:

```
ncx_condition latch_setup_arc {
  ncx_condition_arc_type : constraint; // Applies to constraint arc
  ncx_condition_timing_type : setup_rising; // Applies to hold_rising
  violation_mode_rise : delay_degrade_pct delay_degrade glitch ; // Applies to rising arc
  violation_mode_fall : delay_degrade glitch ; // Applies to falling arc
}
```

Toggled And Non-toggled Arcs

Use `ncx_condition_output_toggle` to look for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for toggled and non-toggled arcs. The values are `true` and `false`. Use `ncx_condition_output_toggle` with `ncx_condition` to specify the success criteria based on toggled or non-toggled output, as shown:

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

Delay and Constraint Margins

You can direct Liberty NCX to add a specified margin to the characterized delay and constraint values. For example, when Liberty NCX determines the cell delay to be 60 ps under certain conditions, you can have the characterized delay value increased by a specified percentage, such as 10 percent, or by a specified absolute amount, such as 5 ps, before it is written to the output library, thereby producing a more conservative library model.

If you want to specify a margin for all delays, you can specify margin parameters anywhere in the template file, at the library, cell, pin, or arc level, for example:

```
margin_value: value ;
margin_percent: value ;
margin_mode: min|max|sum ;
```

Based on the previous example, the margin is applied to all delay and constraint types.

To specify a margin for each type of constraint, use the `ncx_condition` attribute in the library- or cell-level template to specify the target of the margin that you want to apply. The `ncx_condition` attribute targets the application of any attribute in the `ncx_condition` group. You can specify the additional margin for all delay values, for all constraint values, or for specific types of delay or constraint values in the cell. For each margin, you can specify a percentage change, an absolute change, or both. If you specify both a percentage and an absolute change, another template attribute selects which one to use. You must specify the margin values in library time units.

In general, use the ncx_condition attribute as shown:

```
ncx_condition condition {
  ncx_condition_cell_name : cell_name ;
  ncx_condition_pin_name : pin_name ;
  ncx_condition_timing_type | ncx_condition_arc_type : combinational | constraint_type ;
  margin_value : 10 ;
}
```

where

- The ncx_condition_cell_name and ncx_condition_pin_name attributes are optional.
- You can specify either ncx_condition_timing_type or ncx_condition_arc_type.
- You can set the timing type to combinational for combinational timing arcs. Alternatively, you can set the timing type or arc type to constraint to apply a margin to all delay and tristate values, or you can choose a specific constraint. If you choose to add a margin to a specific constraint, specify one of the following constraint types:
  - setup
  - hold
  - removal
  - recovery
  - non_seq_setup
  - non_seq_hold
  - min_pulse_width

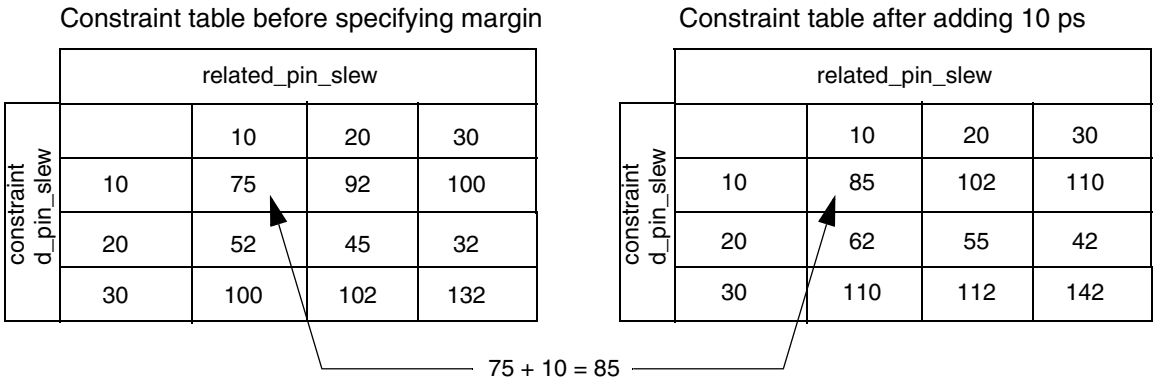
    You can add a wildcard value (*) to specify setup_rising and setup_falling to apply the margin to both rising and falling arcs.
- You can specify a percentage change, an absolute change, or both, by choosing one of the following attribute settings:
  - margin_value

    Set ncx_condition to margin_value to increase calculated delays by a specific number. For example, if you set margin_value to 10, as shown in the following example, Liberty NCX increases calculated delays by 10 library time units:

    ```
    ncx_condition m1 {
      ncx_condition_cell_name : mycell ;
      ncx_condition_pin_name : Y ;
      ncx_condition_timing_type : setup* ;
      margin_value : 10 ;
    }
    ```

In the example, the asterisk wildcard (*) in `setup*` specifies setup for rising and falling arcs. As shown in Figure 3-12, Liberty NCX adds 10 library time units in picoseconds to all values in the constraint table in the specified cell. The table on the left shows the table before the margin values are added. The table on the right shows the new margin values.

Figure 3-12 Example for margin_value



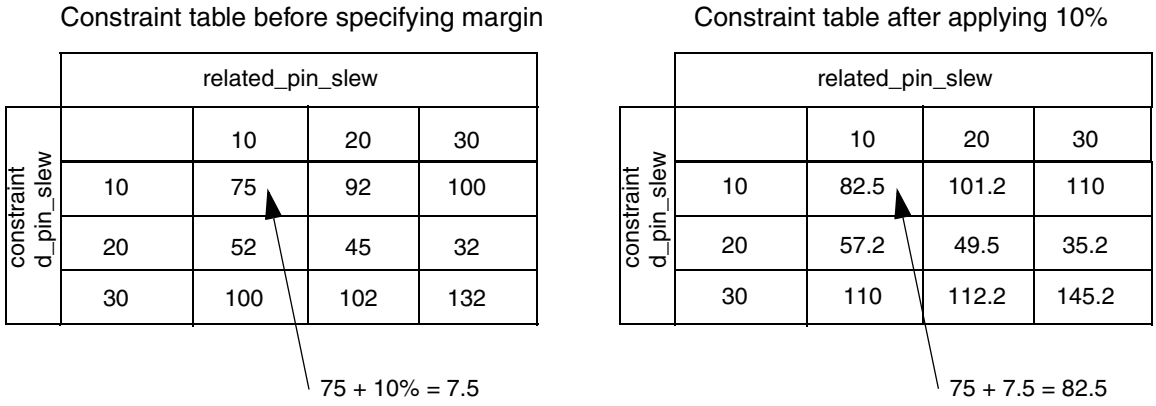
- `margin_percent`

  Set `ncx_condition` to `margin_percent` to increase calculated delays by a specific percentage. For example, if you set `margin_percent` to `10`, as shown in the following example, Liberty NCX increases calculated delays by 10 percent:

```
ncx_condition m1 {
  ncx_condition_cell_name : mycell ;
  ncx_condition_pin_name : Y ;
  ncx_condition_timing_type : setup* ;
  margin_percent : 10 ;
}
```

  As shown in Figure 3-13, Liberty NCX Liberty NCX calculates 10% of each value in the table and then adds that margin to each value. For example, Liberty NCX takes a value of 75, calculates 10%, and then adds 7.5 to it, making the new value 82.5.

Figure 3-13 Example for margin_percent



- `margin_mode`

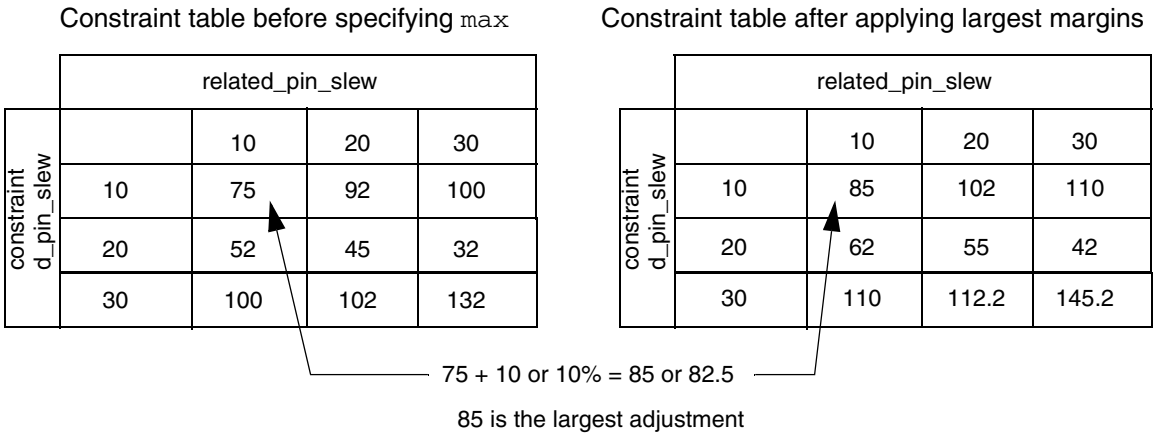
  Set `ncx_condition` to a `margin_mode` value to specify a margin that results in the largest adjustment (`max`), the smallest adjustment (`min`), or the sum of both (`sum`).

  For example, if you set `margin_mode` to `max` and specify a value of `10`, 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 m1 {
  ncx_condition_cell_name : mycell ;
  ncx_condition_pin_name : Y ;
  ncx_condition_timing_type : setup* ;
  margin_percent : 10 ;
  margin_value : 10 ;
  margin_mode : max ;
}
```

  Figure 3-14 shows the constraint table before the `max` value is specified and the table after the largest margin values are applied.

Figure 3-14 Example for margin_mode set to max



- `margin_exp`

  Set `ncx_condition` to `margin_exp` to specify a sum of one or more margin requirements. You specify each margin requirement with keywords and one or more related values, specified in library units. The valid keywords are as follows:

- `constant`

  The related value is a constant value that is applied to all points.

- `percent`

  The related value is a percentage value of the model that is applied to all points.

- `input_net_transition`

  The related values are a set of margin values for each `input_net_transition` index in the table. This specification is applicable only to the models that have `input_net_transition` or `input_transition_time` as one of the table indexes. If the number of values is less than the number of table indexes, the last related value is applied for all the trailing table index values.

- `input_net_transition_pct`

  The related values are a set of margin values calculated as a percentage of each `input_net_transition` index in the table. This specification is applicable only to the models that have `input_net_transition` or `input_transition_time` as one of the table indexes.

- `constrained_pin_transition`

  The related values are a set of margin values for each `constrained_pin_transition` index in the table. This specification is applicable only to the models that have `constrained_pin_transition` as one of the table indexes.

- `constrained_pin_transition_pct`

  The related values are a set of margin values calculated as a percentage of each `constrained_pin_transition` index in the table. This specification is applicable only to the models that have `constrained_pin_transition` as one of the table indexes.

- `related_pin_transition`

  The related values are a set of margin values for each `related_pin_transition` index in the table. This specification is applicable only to the models that have `related_pin_transition` as one of the table indexes.

- `related_pin_transition_pct`

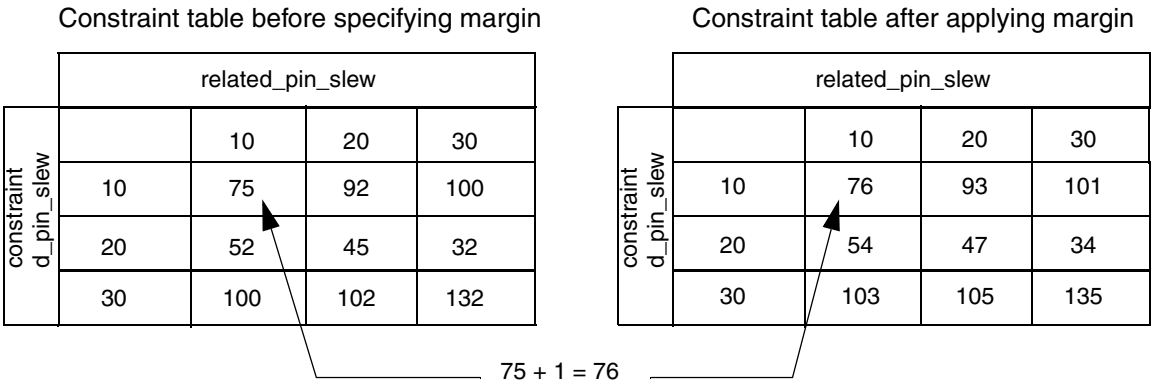
  The related values are a set of margin values calculated as a percentage of each `related_pin_transition` index in the table. This specification is applicable only to the models that have `related_pin_transition` as one of the table indexes.

For example, if you set `margin_exp` to `constrained_pin_transition_pct` and specify `10`, as shown in the following example, Liberty NCX adds 10 percent of the constrained pin slew value to all values in the table:

```
ncx_condition m1 {
  ncx_condition_cell_name : mycell ;
  ncx_condition_pin_name : Y ;
  ncx_condition_timing_type : setup* ;
  margin_exp : constrained_pin_transition_pct 10 ;
}
```

Figure 3-15 shows the constraint table before the `constrained_pin_transition_pct` value is specified and the table after 10 percent of the constrained pin slew value is applied.

Figure 3-15 Example for margin_exp set to constrained_pin_transition_pct

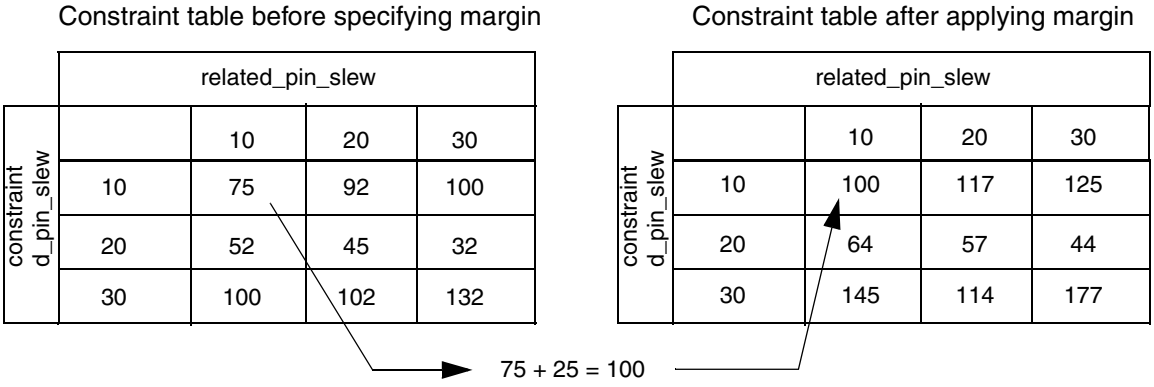


You can also add an absolute value specific to each constrained pin slew to the table by setting `margin_exp` to `constrained_pin_transition`. For example, if you set `margin_exp` to `constrained_pin_transition` and specify `25`, `12`, and `45` as a set of margin values, as shown in the following example, Liberty NCX adds a constant value specific to each constrained pin slew to all values in the table:

```
ncx_condition m1 {
  ncx_condition_cell_name : mycell ;
  ncx_condition_pin_name : Y ;
  ncx_condition_timing_type : setup* ;
  margin_exp : constrained_pin_transition 25 12 45 ;
}
```

Liberty NCX adds the absolute margin that corresponds to the constrained pin slew to the table value corresponding to the same constrained pin slew. Figure 3-16 shows the constraint table before the `constrained_pin_transition` value is specified and the table after the margins are applied.

Figure 3-16 Example for margin_exp set to constrained_pin_transition



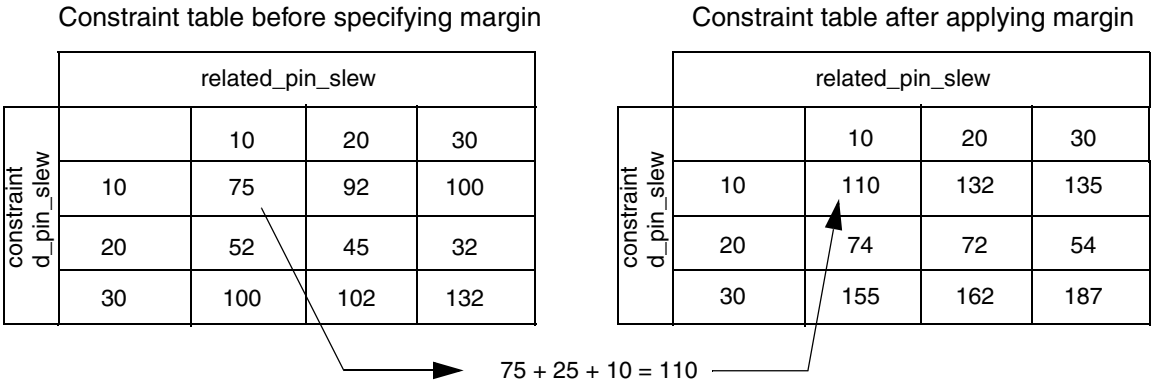
The `constrained_pin_transition` value `25` corresponds to `constrained_pin_slew` table value `10`, so 25 is added to the values in the first row of the table, making the new values `100`, `117`, and `125`. Similarly, 12 is added to the second row values in the table, and 45 is added to the third row values.

To add constant values specific to each constraint pin slew and related pin slew to the table, set `margin_exp` to `constrained_pin_transition` and specify `related_pin_transition` values. 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`, as shown in the following example, Liberty NCX adds a constant value specific to each constraint pin slew and related pin slew to all values in the table:

```
ncx_condition m1 {
  ncx_condition_cell_name : mycell ;
  ncx_condition_pin_name : Y ;
  ncx_condition_timing_type : setup* ;
  margin_exp : constrained_pin_transition 25 12 45 \
               related_pin_transition 10 15 10 ;
}
```

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 3-17 shows the constraint table before the `constrained_pin_transition` value is specified and the table after the margins are applied.

Figure 3-17 Example for margin_exp set to constrained_pin_transition and related_pin_transition



The `constrained_pin_transition` value `25` corresponds to `constrained_pin_slew` table value `10`, and `related_pin_transition` value `10` corresponds to `related_pin_slew` table value `10`. Therefore, the margin calculated for value `75`, as shown in Figure 3-17, 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` value `25` corresponds to `constrained_pin_slew` table value `10`, and `related_pin_transition` 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`.

Note:

If the margin number is less than the constrained and related index number, the last constrained and related value is applied to all the trailing table index values.

You can also include constraint slew and related slew in the margin calculations for constraint data generation, except minimum pulse width, by using `ncx_condition` and setting `margin_exp`. The constraint types can be `constraint`, `setup`, `hold`, `non_seq_setup`, `non_seq_hold`, `recovery`, or `removal`. You can also specify a percent or constant to be added to this margin. When you do this, the margin value is calculated as shown in the following equation:

```
Margin = percent + constant +
related_slew * related_pin_transition_percent +
constraint_slew * constrained_pin_transition_percent
```

For example,

```
ncx_condition m1 {
  ncx_condition_arc_type : constraint ;
```

```
  margin_exp : related_pin_transition_pct 12 \
     constrained_pin_transition_pct 10 constant 0.005 percent 15 ;
}
```

NLDM delay and constraint values are adjusted by the specified percentage or absolute amount. In CCS unexpanded driver models, the timing points are adjusted by the amounts calculated by applying the specified percentage or absolute amounts to the NLDM delay models. CCS receiver models are not affected by the specified margins. For compact CCS timing models, the Tpeak values are shifted based on the specified percentage or absolute amount.

In variation-aware models, the `va_compact_ccs_rise` and `va_compact_ccs_fall` groups are adjusted by the same method as nominal compact CCS models, and the `va_rise_constraint` and `va_fall_constraint` groups are adjusted by the same method as the nominal constraint. Variation-aware receiver capacitance models are not affected.

Constraint Methodologies

The setup and hold time of a constraint arc is characterized by searching the minimum setup and hold time that does not trigger a violation event. There are several ways to define a violation criterion, and each directly affects characterization results in the library. For example, a violation event can be defined by a failed delay measurement, a relative or absolute delay degradation, or a glitch that is greater than a specific threshold.

Liberty NCX supports constraint characterization by allowing you to specify an output node that you can use for initial delay. When you choose delay degradation as a violation criterion, the constraint arc setup and hold time is characterized by searching a specific setup and hold time so that the corresponding delay is equal to a predefined target delay.

Use the `violation_mode` attribute to consider an absolute and a relative degradation simultaneously for rising and falling arcs. When you set `violation_mode` to `delay_degrade_pct delay_degrade`, Liberty NCX monitors all output pins with both relative and absolute delay degradation. To assign absolute delay only, use the `violation_delay_degrade` attribute set to an absolute delay degradation value in library time units. To assign relative delay only, use the `violation_delay_degrade_pct` attribute set to a relative delay degradation percentage value.

Ideally, the corresponding delay of the characterized setup time should be equal to the target delay. However, the setup value reported by HSPICE could come from a failed event where the delay value is greater than the target delay or a passed event where the delay value is less than the target delay. Currently, Liberty NCX adds a margin in the final setup time to satisfy the requirement. For example, HSPICE stops the searching process if the current search interval is less than the `ncx_constraint_accuracy` value. The default is 2 ps.

Because the final setup time in the HSPICE output file is either from the failed event or from the passed event, and their difference is less than the `ncx_constraint_accuracy` value, Liberty NCX adds `ncx_constraint_accuracy` as a margin to guarantee the new setup time is no less than the setup time of the passed event.

You can apply different violation modes depending on whether the arcs are rising or falling. Use `violation_mode_rise` to look for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for rising arcs only, and use `violation_mode_fall` to look for relative or absolute delay degradation, measurement failures (pass/fail), or glitch detection for falling arcs only.

Multiple Output Cells

Cells with multiple outputs have more than one delay path. You can monitor a specific delay path when characterizing a constraint arc. Use the `violation_mode` attribute to monitor multiple nodes, as shown:

```
violation_mode : delay_degrade_pct Q glitch Int1 glitch "Int:2" ;
```

The node names can be external or internal nodes. In the example, Liberty NCX monitors pin `Q` and internal nodes `Int1` and `Int:2`. The quotation marks are required if the internal node name contains a colon.

violation_mode Variables

Table 3-6 lists the `violation_mode` settings that you can use to return the worst case for multiple netlists with different violation criteria.The node in column two can be an internal node or an output node.

Table 3-6 violation_mode Settings

Method	Node	Optional parameter	Optional parameter
`delay_degrade_pct`	*node_name*	Relative delay percentage value	n/a
`delay_degrade`	*node_name*	Absolute delay value in library time units	n/a
`glitch`	*node_name*	Lower threshold percentage value	Upper threshold (percentage)
`passfail`	*node_name*	n/a	n/a

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 specify violation_mode for the sets. 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/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/fail violations for all nodes in set2.

Accuracy Settings for Constraint Acquisition

If Liberty NCX characterization provides constraint numbers that seem too large or if there are bisection errors during constraint characterization, use the following variables to control the accuracy of the constraint numbers:

- ncx_constraint_search_interval

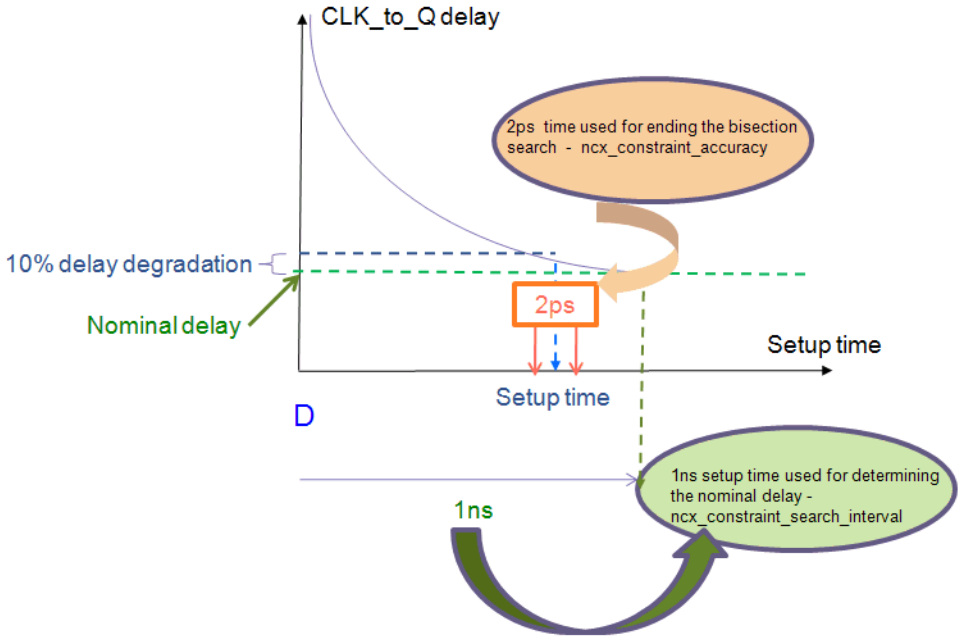
  Currently for delay degradation, Liberty NCX runs a simulation to determine the nominal or the reference delay. The setup time used to find the nominal delay is 1 ns (half of the ncx_constraint_search_interval). If a 1 ns setup time is too small to determine the nominal delay, you can set ncx_constraint_search_interval to the appropriate or desired value, which is a floating-point number in library time units. The default value is 2 ns.

  You can also use ncx_constraint_search_interval as the starting search interval to determine the setup, hold, recover, or removal time. If the search interval is set to 2 ns, the “must pass” for setup, hold, recovery, and removal is obtained at +1ns and the “must fail” is obtained at -1 ns. However, a width of 2 ns is used for the minimum pulse width.

- ncx_constraint_accuracy

The `ncx_constraint_accuracy` attribute determines the end of the bisection search during constraint characterization. Liberty NCX stops the bisection search when the bisection search interval is less than 2 ps. You can define the value, which is a floating-point number in library time units. The default value is 2 ps.

Figure 3-18 Accuracy Settings for Constraint Acquisition



Glitch Detection for Constraint Arcs

Glitch detection is available for setup, hold, recovery, removal, and minimum pulse width for constraint arcs. You can set `violation_mode` to `glitch` to enable glitch detection mode. You can also set `violation_mode` to `passfail` to perform pass/fail violations, and use `violation_mode` to monitor single node or multiple nodes.

To ensure that the output remains stable, use the following attributes to detect any glitches during characterization:

```
violation_mode : glitch ;
```

The `violation_mode` attribute considers absolute and a relative degradation simultaneously. When you set `violation_mode` to `delay_degrade_pct delay_degrade`, Liberty NCX monitors all output pins with both relative and absolute delay degradation. Setting `violation_mode` to glitch enables glitch detection mode. Setting `violation_mode` to `passfail` performs pass/fail violations.

`violation_mode : delay_degrade_pct`

For a specific arc, it is required to monitor two nodes at the same time when characterizing the setup time. You should specify the violation criteria for each monitor node. For example, you can specify node A with delay degradation mode and node B with glitch detection mode, as shown:

```
violation_mode: delay_degrade_pct "intnode:4" glitch Q;
```

Liberty NCX characterizes a setup time that satisfies both criteria. In the example, Liberty NCX monitors relative delay degradation on internal pin `intnode:4` and also monitors glitch detection on external pin `Q`.

Note:

The `violation_mode` attribute is available with the HSPICE simulator only.

`violation_glitch_upper_pct : (0 - 100)`

The `violation_glitch_upper_pct` attribute specifies the upper threshold for non-toggled output. You can specify a value from 0 – 100.

`violation_glitch_lower_pct : (0 - 100)`

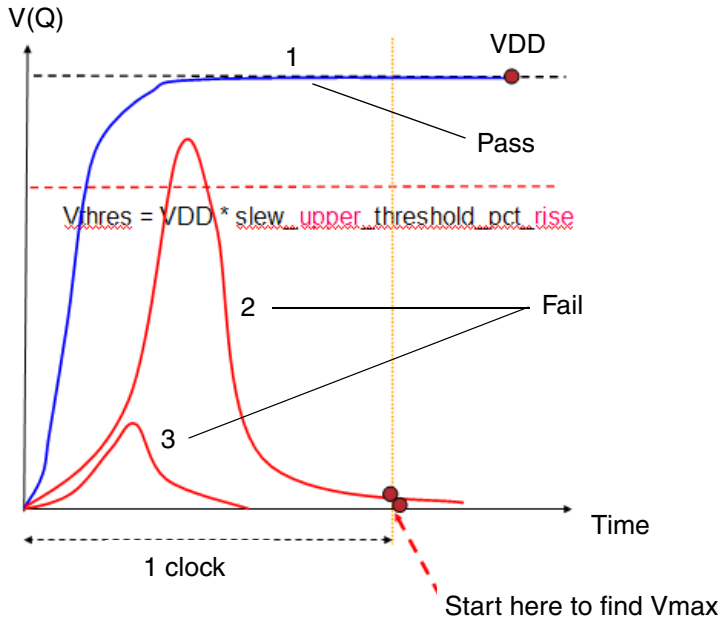
The `violation_glitch_lower_pct` attribute specifies the lower threshold for non-toggled output. You can specify a value from 0 – 100.

Note:

These attributes are available with the HSPICE and Eldo simulators only.

For example, Figure 3-19 represents the results of a setup test with pessimistic results. The output is supposed to rise from zero to VDD. If the cell passes the test, the output should rise and remain steady, as curve 1 does in Figure 3-19. However, some cells show glitches, as shown by curves 2 and 3. Because curves 2 and 3 do not reach the `violation_glitch_*_pct` threshold and remain steady, Liberty NCX reports them as failures.

Figure 3-19 Setup Rising Glitch Detection Test



In Figure 3-20, Liberty NCX reports curves 2 and 3 as failures because the output does not fall below the lower threshold and remain steady, as it does with curve 1.

Figure 3-20 Setup Falling Glitch Detection Test

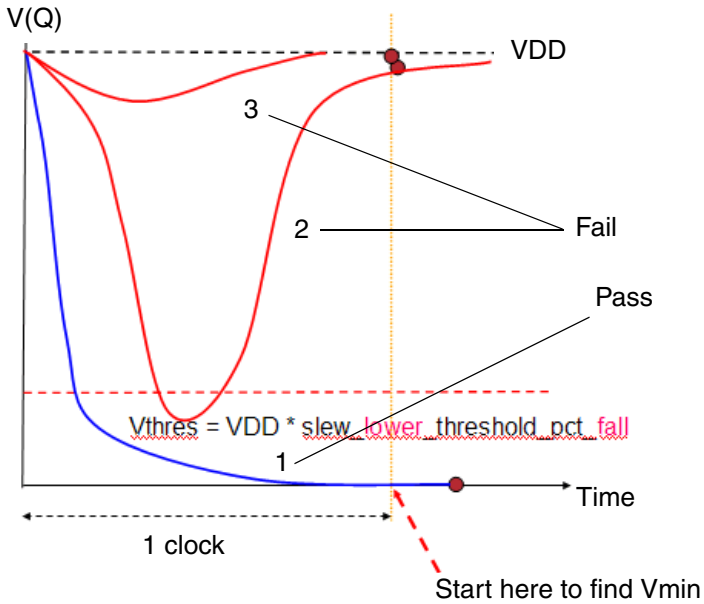
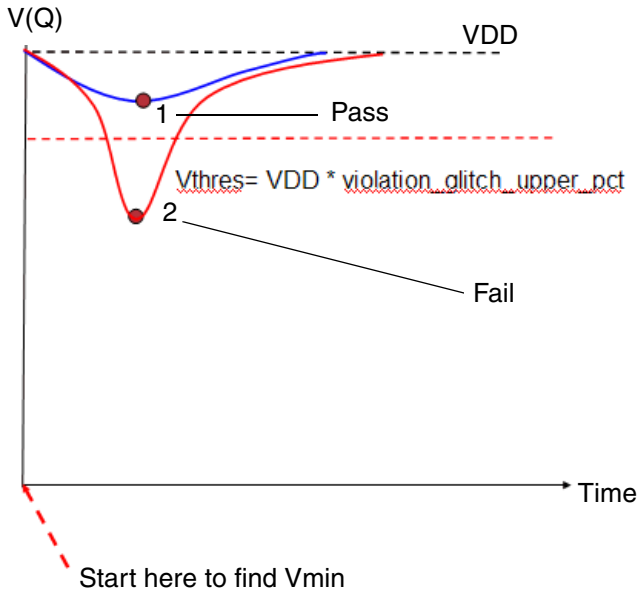


Figure 3-21 represents the results of a hold test with pessimistic results. The output should remain steady above the threshold. Curve 1 stays within the threshold and passes the test. However, curve 2 crosses the threshold and fails.

Figure 3-21 Hold Glitch Detection Test



Template Examples

The following examples are cell template examples for `violation_mode`:

Example 1

```
violation_delay_degrade_pct  :  20 ; //Change relative delay degradation default to 20%
violation_delay_degrade : 0.03 ; // Change absolute delay degradation default to 0.03
                                        (lib time unit)
violation_glitch_lower_pct : 25 ; // Change glitch detection lower threshold default to 25%
violation_glitch_upper_pct : 75 ; // Change glitch detection upper threshold default to 75%
ncx_node_set : set1 Q1  int1 ;

violation_mode : \
delay_degrade_pct  \  // Apply relative delay degradation to all external output pins
                        with current default parameters (20)
delay_degrade   Q \  // Apply absolute delay degradation to pin Q with current default
                      parameters (0.03)
passfail \   // Apply passfail to all external pins
glitch  int2 \  // Apply glitch to internal pin int2 with current default thresholds (25/75)
glitch  set1 10 90 ;  // Apply glitch to all nodes in set1 with thresholds 10 and 90
```

Example 2

```
  // Apply glitch to all hold_rising arc of a latch cell
ncx_condition latch_hold_arc {
```

```
          ncx_condition_cell_type : latch;
          ncx_condition_arc_type : constraint;
          ncx_condition_timing_type : hold_rising;
          violation_mode : glitch;
}

// Apply different violation mode 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;
}
```

4

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

The Model Adaptation System can convert 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-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 command 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.

These are the supported library formatting operations:

- CCS Model Compaction and Expansion
- Variation-Aware Library Merging
- CCS Noise Merging
- CCS-to-ECSM Conversion
- CCS-to-NLDM Conversion
- VA-CCS-to-S-ECSM 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 User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries*.

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 command 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 command 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*, which is available in SolvNet (www.solvnet.com) in the PrimeTime documentation suite.

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 xl82c_noise.lib
set output_library xl82c.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.

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-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 command 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 vaccs2secsm 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 as follows:

vaccs2secsm

  Converts a variation-aware CCS library to an S-ECSM library. If vaccs2secsm 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 valid values are:

- 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:

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 a Liberty NCX license is not available.

Generating Datasheets

Library Compiler generates a datasheet from a .lib file and creates a *cell*.txt file for each cell in the library. By default, Library Compiler creates the *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.

Use the following commands to generate datasheets:

1. To enable datasheet generation, set the `datasheet_enable` variable to true. By default, the variable is set to false, and a datasheet is not generated.

```
set datasheet_enable [true | false]
```

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. If you want the datasheet to be written to a directory other than *library_name*_datasheet, set the `datasheet_output_dir` variable, as shown:

   ```
   set datasheet_output_dir directory_name
   ```

Datasheet Details

Each *cell_name*.txt datasheet provides the following information for a cell:

- Cell : *cell_name*
- Function Description
  - Combinational Function : *function/statetable*
  - Three state : *function*
- Port Names
  - Pin declarations
- Cell Area : *cell area*
- Pin Capacitance
  - Input pins load and output pin max capacitance table
- Delay Data
  - Pin to pin delay table
- Constraint Data
  - Constraint table

Cell Name

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
```

Function Description

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.

Combinational Function. The combinational function format provides the Boolean function for a gate.

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 ^ B' & 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.

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 statetable.

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 statetable pin list is a comma-separated list and contains input pins followed by one output pin with a current state and next state. The characteristics are as follows:

- 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 current state is Q, the next state is Q+.
- One statetable has exactly one output pin.
- The order of the input pin list determines the order of the pins in the statetable.

- Generally, there is a statetable for each output of the cell. However, in some cases, there might not be a statetable. For example, QN might be a logic function of Q (for instance, inverted). The datasheet generator writes a Boolean expression for QN, using Q.

Table Content. The table content syntax is as follows:

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, *

current_state

The current state is the output current state value. Valid characters are: 0, 1, x, ?, b

next_state

The next state is the next state value. Valid characters are: 0, 1, x, -

The values are defined as follows:

0

Specifies a low state.

1

Specifies a high state.

x

Specifies unknown.

?

Is an iteration of 0, 1 and x.

b

Is an iteration of 0 and 1.

r

Is the same as 01.

f

Is the same as 10.

*

Is the same as ??

–

Specifies 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)
```

Port Names

The port names syntax is as follows:

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.

Cell Area

The cell area syntax is as follows:

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
```

Pin Capacitance

The datasheet translator 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. [*capacitance unit]***
pin name	*capacitance value*
pin name	*capacitance value*

Outputs:

Pin	**Max. Cap. [*capacitance unit]***
pin name	*maximum capacitance value*
pin name	*maximum capacitance value*

The arguments are as follows:

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

Delay Data

The delay data syntax is as follows:

Syntax

Delay Path	Delay [*time unit*]
input pin input transition => output pin output transition	*Delay value*

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. The delay value is extracted from NLDM tables only in Liberty NCX version A-2007.12.

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

Constraint Data

The datasheet contains the following information about constraint data.

Syntax

Check	**Constraint [*time unit*]**
constraint_pin constraint_pin_transition check_type related_pin related_pin_transition	*constraint_value*

The arguments are as follows:

constraint_pin

The parent pin of a timing constraint in 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 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

Scripts and Examples

This section provides scripts for generating datasheets and provides Liberty .lib and datasheet examples.

Tcl Script

You can run the following Tcl script in Library Compiler to enable datasheet creation and generate a datasheet from a .lib file.

```
read_lib trial.lib
set datasheet_enable true
set datasheet_output_dir test
write_lib -format datasheet trial
quit
```

You can source the Library Compiler Tcl script by running the following in lc_shell:

```
lc_shell-t -f lc.tcl
```

Datasheet Examples

This section provides datasheet examples for different types of cells.

Inverter Cell. The following example shows an inverter cell in Liberty .lib format:

```
cell (INV1) {
           area : 8.000000;
           pin (I) {
                direction : "input";
                fall_capacitance : 0.008996;
                capacitance : 0.008996;
                rise_capacitance : 0.009715;
           }
           pin (O) {
                direction : "output";
                function : "I'";
                timing () {
                     related_pin : "I";
                     timing_type : "combinational";
                     timing_sense : "negative_unate";
                     cell_rise ("del_1_7_7") {
                          index_1("…");
                          index_2("…");
                          values("…", \
 …
 "…, …, 2.4615774");
                     }
                     rise_transition ("del_1_7_7") {
                          index_1("…");
                          index_2("…");
                          values("…", \
  …
 "…, …, 2.5410982");

                     }
                     cell_fall ("del_1_7_7") {
                          index_1("…");
                          index_2("…");
                          values("…", \
  …
 "…, …, 2.8203990");

                     }
                     fall_transition ("del_1_7_7") {
                          index_1("…");
                          index_2("…");
                          values("…", \
```

```
  …
 "…, …, 2.9987413");

                        }
                  }
            }
      }
```

The following example shows an inverter cell in a datasheet:

```
Cell: INV1

Function Description:
Combinational gate.
Function: O = (I')

Port Names:
        Inputs: I
        Outputs: O

Cell Area: 8.000000

Pin Capacitance:
        Inputs:
        Pin          Cap.(1.000000pf)
        I            0.008996

        Outputs:
        Pin          Max. Cap.(1.000000pf)

Delay Data:
Delay Path                      Delays(1ns)
I 10 => O 01                            2.46158
I 01 => O 10                            2.8204
```

Register Cell. The following example shows a register cell in Liberty .lib format:

```
cell (DFF1X2) {
           area : 35.000000;
           ff (IQ,IQN) {
                next_state : "D";
                clocked_on : "CK";
           }
           pin (D) {
                direction : "input";
                rise_capacitance : 0.013006;
                capacitance : 0.012277;
                fall_capacitance : 0.012277;
                timing () {
                     related_pin : "CK";
                     timing_type : "setup_rising";
                     fall_constraint ("vio_3_3_1") {
```

```
                  index_1("…");
                  index_2("…");
                  values("…", \
                        "…", \
                        "…, 0.5745505");
            }
            rise_constraint ("vio_3_3_1") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        "…", \
                        "…, 0.1776389");
            }
      }
      timing () {
            related_pin : "CK";
            timing_type : "hold_rising";
            rise_constraint ("vio_3_3_1") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        "…", \
                        "…, -0.1370505");
            }
            fall_constraint ("vio_3_3_1") {
                  index_1("…");
                  index_2("…");
                  values("…", \

                        "…", \
                        "…, -0.4651755");
            }

      }
}
pin (CK) {
      direction : "input";
      clock : true;
      rise_capacitance : 0.012356;
      capacitance : 0.011735;
      fall_capacitance : 0.011735;
      timing () {
            related_pin : "CK";
            timing_type : "min_pulse_width";
            fall_constraint ("constraint_3_0_1") {
                  index_1("…");
                  values("…");
            }
            rise_constraint ("constraint_3_0_1") {
                  index_1("…");
                  values("…");
            }
      }
```

```
}
pin (Q) {
      direction : "output";
      function : "IQ";
      timing () {
            related_pin : "CK";
            timing_type : "rising_edge";
            timing_sense : "non_unate";
            cell_rise ("del_1_7_7") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        …

                        "…, 2.3738176");
            }
            rise_transition ("del_1_7_7") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        …
                        "…, 2.6907971");
            }
            cell_fall ("del_1_7_7") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        …
                        "…, 2.6747051");
            }
            fall_transition ("del_1_7_7") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        …
                        "…, 2.9919676");
            }
      }
}
pin (QB) {
      direction : "output";
      function : "IQN";
      timing () {
            related_pin : "CK";
            timing_type : "rising_edge";
            timing_sense : "non_unate";
            cell_rise ("del_1_7_7") {
                  index_1("…");
                  index_2("…");
                  values("…", \
                        …
                        "…, 2.2308171");
            }
```

```
                        rise_transition ("del_1_7_7") {
                              index_1("…");
                              index_2("…");
                              values("…", \
                                    …
                                    "…, 2.2275617");
                        }
                        cell_fall ("del_1_7_7") {
                              index_1("…");
                              index_2("…");
                              values("…", \
                                    …
                                    "…, 2.8469531");
                        }
                        fall_transition ("del_1_7_7") {
                              index_1("…");
                              index_2("…");
                              values("…", \
                                    …
                                    "…, 2.9996052");
                        }
                  }
            }
      }
```

The following example shows a register cell in a datasheet:

```
Cell: DFF1X2

Function Description:
Flip-flop.

Statetable:
        CK      D       Q       Q+
        (01)    0       ?       0
        (01)    1       ?       1
        (1?)    ?       ?       -
        (?0)    ?       ?       -

Statetable:
QB = (not Q)

Port Names:
        Inputs: D, CK
        Outputs: Q, QB

Cell Area: 35.000000

Pin Capacitance:
        Inputs:
        Pin           Cap.(1.000000pf)
        D             0.012277
```

```
        CK              0.011735

        Outputs:
        Pin             Max. Cap.(1.000000pf)


Delay Data:

Delay Path                          Delays(1ns)
CK     => Q 01                              2.37382
CK     => Q 10                              2.67471
CK     => QB 01                             2.23082
CK     => QB 10                             2.84695

Constraint Data:

Check                               Constraint(1ns)
D 01 setup       CK 01              0.177639
D 10 setup       CK 01              0.574551
D 01 hold        CK 01              -0.13705
D 10 hold        CK 01              -0.465176
```

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
- 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 verilog_enable variable to true. By default, the variable is set to false, and a Verilog file is not generated.

   ```
   set verilog_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
   ```

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 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 generator models all combinational cells, including adders, subtractors, 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 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 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);
```

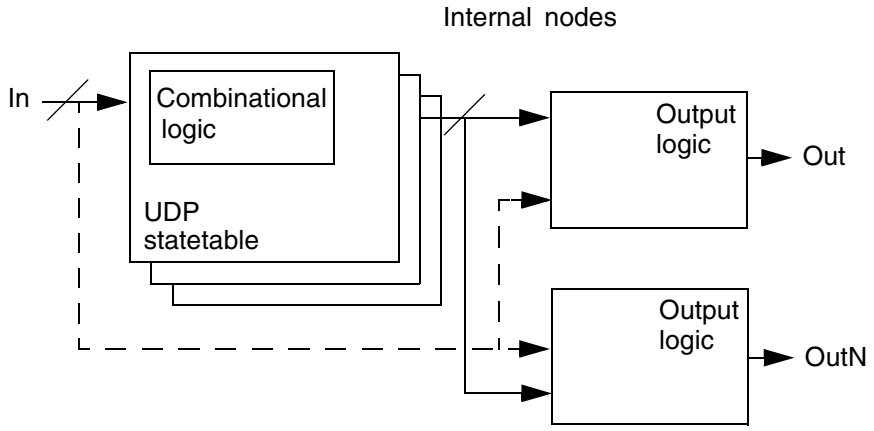
where *udp_name* is formatted as UDP_*cell*_*port*, and where *cell* specifies the cell name and *port* specifies either the name of the output pin or the internal node.

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 4-1 shows the general sequential output model with UDP statetables. Sequential cell functions may contain several sequential "states" and thus can be described with multiple UDP statetables. Each UDP instantiation (statetable) describes one sequential state. Cells with multiple sequential states have multiple UDP instantiations.

Figure 4-1 General Sequential Output Model



Input (combinational) logic is included in the UDP statetable description. Thus, there is no need to explicitly describe them in the Verilog function description. Output logic is not included in the UDP statetable, 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;
```

specify Block

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, pulsewidth, 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 4-41.
- *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 4-41.

  Because *start_pin* and *end_pin* are the same for pulsewidth 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. 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-tristate rise and fall delays or 6 (rise, fall, three state) for tristate arc delays. See “Delay Values” on page 4-40.

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 4-39.

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 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 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 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 generator avoids generating duplicate path delays. If a potential duplicate path delay is found, the Verilog 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, setuphold, recovery, removal, recrem, 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 4-50.

- *`start_pin`*

  Specifies the data pin.

- *start_cond* (optional) based on the input Liberty file.See “Conditional Timing Checks” on page 4-55.
- *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 4-50.
- *end_pin* is the clock pin.
- *end_cond* is optional and is dependent on the input Liberty file. See “Conditional Timing Checks” on page 4-55.
- *setup_param* is the timing limit. For naming conventions, see “specparam” on page 4-35.
- `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 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 4-44 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 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 4-50.

- *start_pin* is the clock pin.
- *start_cond* is optional and is dependent on the Liberty input file. See “Conditional Timing Checks” on page 4-55.
- *end_pin* is the data pin.
- *end_cond* is optional and is dependent on the input Liberty file. See “Conditional Timing Checks” on page 4-55.
- *hold_param*

  Specifies the timing limit. For naming conventions, see “specparam” on page 4-35.

- `notifier` is a `reg` variable used to detect timing check violations behaviorally. For more information, see “Setup” on page 4-41.
- 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 4-44.

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 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 4-50.
- *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 4-55.
- *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 4-50.
- *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 4-55.
- *setup_param* is the timing limit for the associated setup timing check. For naming conventions, see "specparam" on page 4-35.
- *hold_param* is the timing limit for the associated hold timing check. For naming conventions, see "specparam" on page 4-35.
- `notifier` is a `reg` variable used to detect timing check violations behaviorally. For more information, see "Setup" on page 4-41.
- The Verilog 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 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 4-50.
- *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 4-55.
- *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 4-50.
- *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 4-35.
- `notifier` is a `reg` variable used to detect timing check violations behaviorally. See "Setup" on page 4-41 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 4-47.

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 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 4-50.
- *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 4-55.
- *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 4-50.
- *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 the specparam section.
- `notifier` is a `reg` variable used to detect timing check violations behaviorally. See the Setup section for more information.
- The `$removal` timing check is modeled only when there is no matching recovery timing check in the Liberty timing groups. For more information, see "Recrem."

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 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 4-50.
- `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 4-55.
- *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 4-50.
- *clock_pin*

  Specifies the clock pin.
- *clock_cond* is dependent on the input Liberty file. See “Conditional Timing Checks” on page 4-55.
- *recovery_param* is the timing limit for the associated recovery timing check. For naming conventions, see the specparam section.

- *removal_param*: timing limit for the associated removal timing check. For the naming convention, see the specparam section.
- notifier is a reg variable used to detect timing check violations behaviorally. See the Setup section for more information.

The Verilog 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 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 4-50.

- *pin*

  Specifies the constraint pin.

- *cond* is optional and is dependent on the input Liberty file. See “Conditional Timing Checks” on page 4-55.

- *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 4-41.

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 4-50.

- *pin*

  Specifies the constraint pin.

- *cond* is optional and is dependent on the input Liberty file. See “Conditional Timing Checks” on page 4-55.

- *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 4-41.

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.”

- *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 4-55.
- *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.”
- *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 4-55.
- *skew_param*

  Specifies the timing limit for the skew timing check. For the naming convention, see “specparam” on page 4-35.
- `notifier` is a `reg` variable used to detect timing check violations behaviorally. See the “Setup” section 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, notifier);
```

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, trecovery_R_CK_0, tremoval_CK_R_0, notifier);
```

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 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 generator issues a warning and assumes a `posedge` for the period check.
- For the `minimum_period` group in the pin group, the Verilog 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 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 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 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 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 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 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 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 4-39.

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 generator avoids

generating duplicate timing checks. However, if potential duplicate timing checks occur, the Verilog 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
table_content
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 statetable 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 statetable is as follows:

```
table
input_state[ 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 statetable 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 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 for the entire library during Verilog generation. To automatically generate library-level testbenches, 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_annotate` 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 timescale 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 lines in the following format:

```
[pin order list]
$
absolute-time1 input-stimulus1 [expected-outputs]
absolute-time2 input-stimulus2 [expected-outputs]
absolute-time3 input-stimulus3 [expected-outputs]
```

  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 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 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 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 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 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.

5

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

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 command file invoke transistor mismatch characterization.

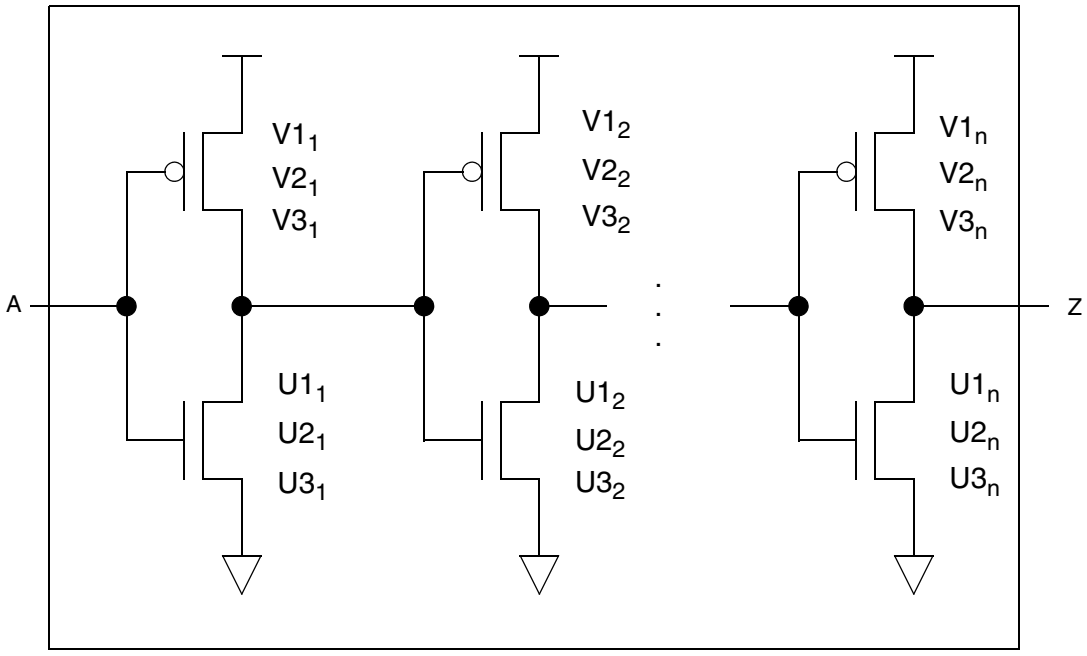
Characterization with transistor mismatch produces at least one new synthetic variation parameter to represent the mismatch variation, in addition to any other global variation parameters being characterized. The resulting library can be used for variation-aware analysis in PrimeTime VX.

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 5-1 shows an example of the internal circuitry within a cell. The cell has *n* NMOS-PMOS transistor pairs.

Figure 5-1 Cell-Internal Transistors With Variation Mismatch

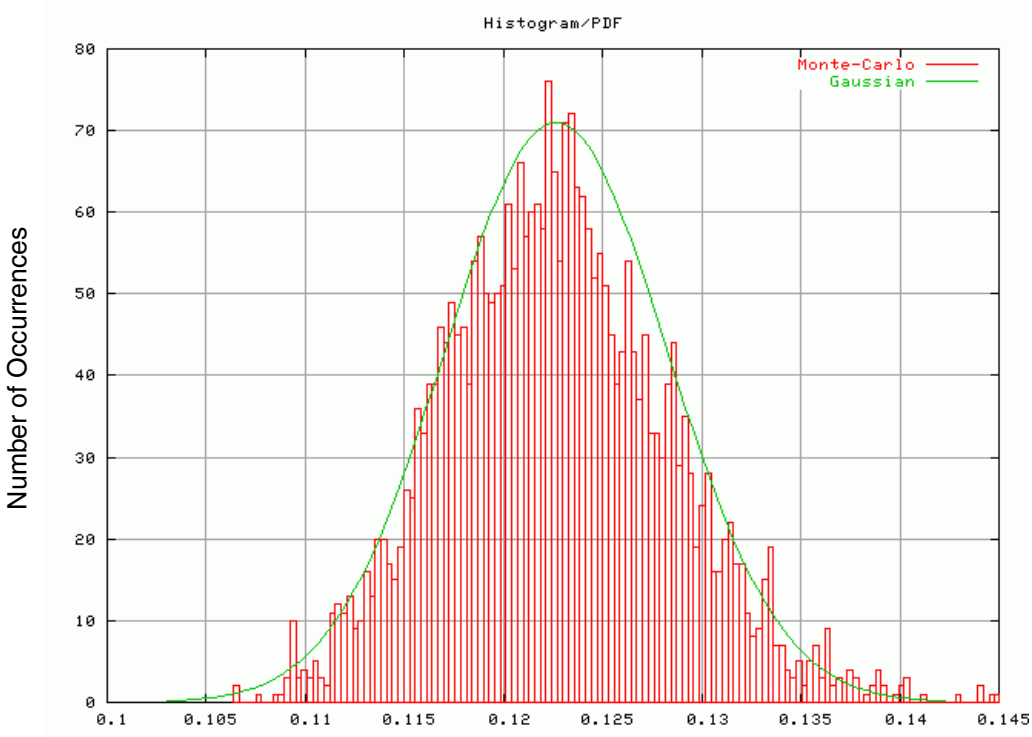


In this example, each transistor has three independent variables that represent process variations at the transistor level. The NMOS transistors have variables U1, U2, and U3; and the PMOS transistors have variables V1, V2, and V3. The timing characteristics of the transistors depend upon these variables. For each NMOS-PMOS pair, there are six variables that affect the stage timing. To the extent that the variables are independent, up to 6*n* variables affect the timing characteristics of the cell.

Each variable has a Gaussian distribution N(μ, σ), where μ is the nominal value and σ 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(μ, σ). The gate delay for an input-to-output transition was plotted in a histogram. The results are shown in Figure 5-2.

Figure 5-2 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 $N(\mu,\sigma(D))$, where μ is the nominal value of the arc delay and $\sigma(D)$ 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 $+\sigma$ is 0.128, and the delay value at $-\sigma$ 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 +σ offset value, and the –σ offset value.

The nominal cell model is characterized by setting all variables to their nominal values. The +σ offset data represents the cell behavior with the synthetic variable M at nominal plus one σ (or 1.5σ, or 2σ, or whatever quantile you specify). The –σ offset data represents the cell behavior with the synthetic variable M at nominal minus one σ (or whatever quantile you specify). You specify the desired offset using `ncx_mm_sigma` in the command file. The default setting is 1.0. For example, to set the offset to 1.5σ 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, +σ, and –σ 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 3σ, and you also perform transistor mismatch characterization at plus and minus σ, then you will have a total of three variation parameters, Len, Vt, and M. This results in 2N+1 (seven) characterization points: nominal, +3σ Len, –3σ Len, +3σ Vt, –3σ Vt, +σ M, and –σ M.

Mismatch Characterization Options

To invoke transistor mismatch characterization in Liberty NCX, use the following commands in the command 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. (You must specify one type of transistor model only):

```
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.

6

CCS Models

Very deep submicron designs (90 nm and below) challenge the accuracy of static timing analysis modeling and simulation due to effects such as faster clocks, complex interconnect impedance, and nonlinear device behavior. Synopsys has developed 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 very deep submicron designs.

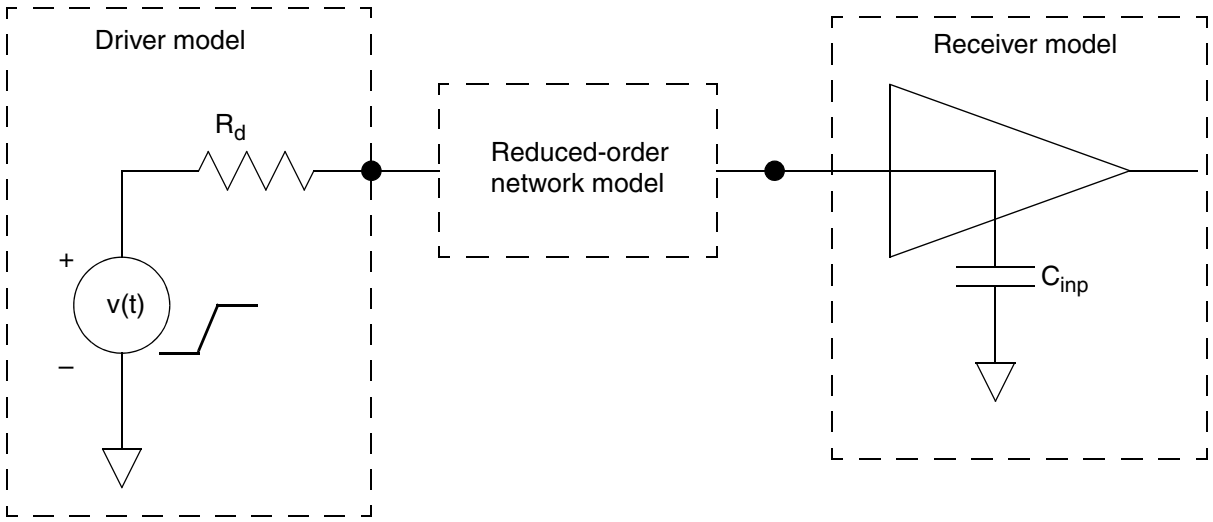
This chapter contains the following sections:

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 6-1 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 6-1 NLDM Driver/Receiver Model



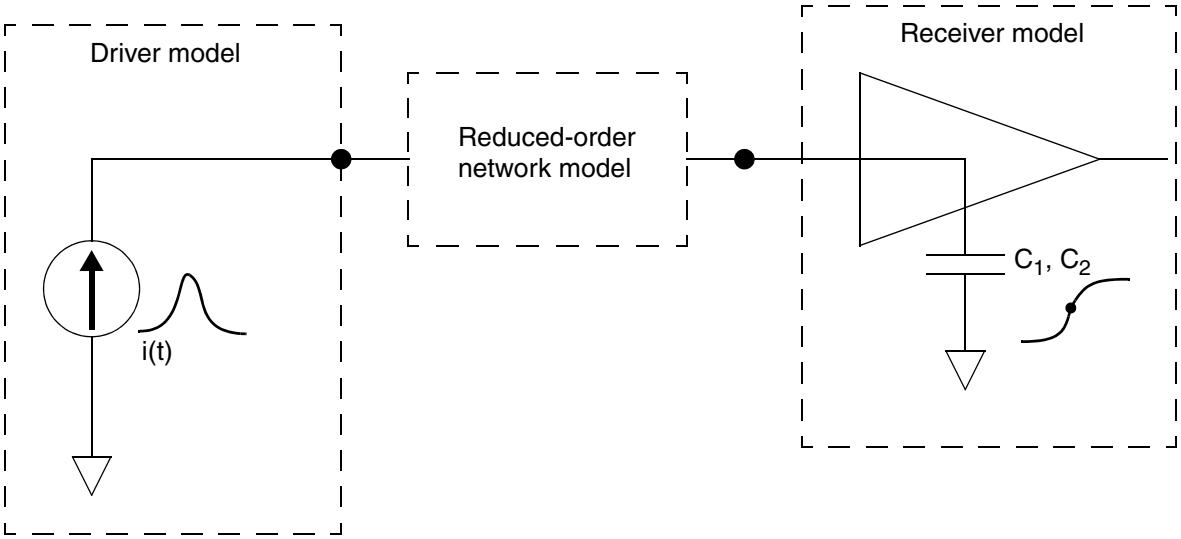
The output voltage can be determined by the following equation:

$$V_{out} = V_{in} \times (Z_{net} / (Z_{net} + R_d))$$

As $Z_{net} >> R_d$, V_{out} approaches V_{in}. This results in an incorrect representation of driver behavior.

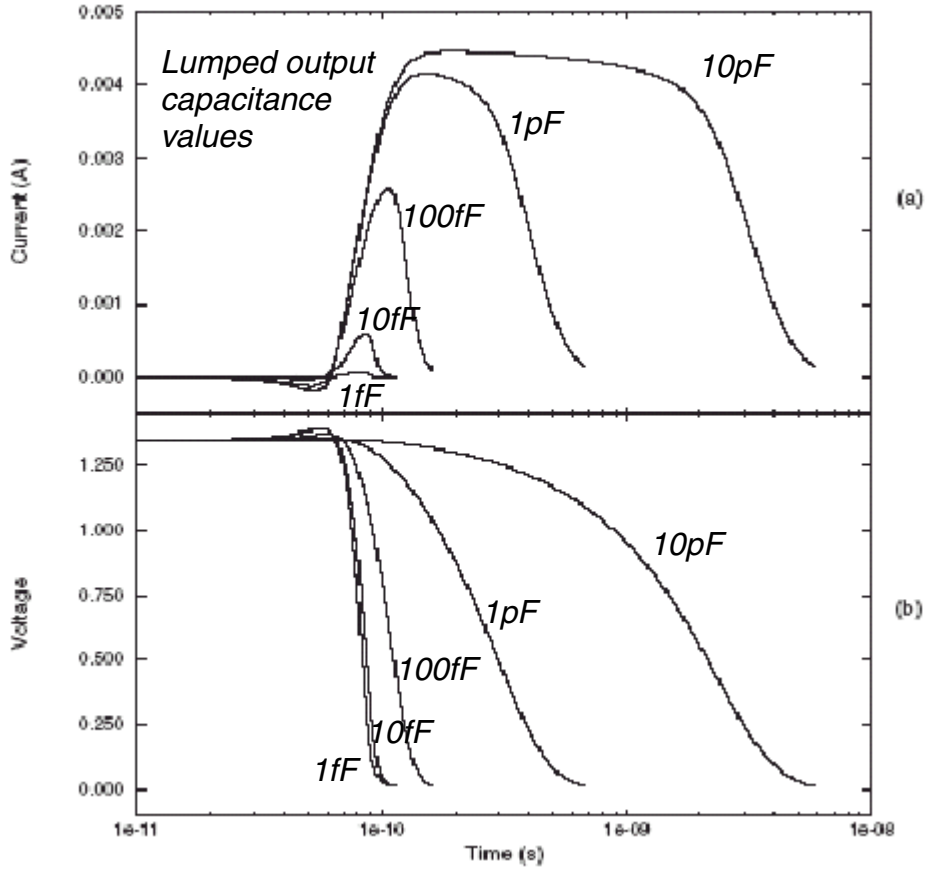
The CCS driver model shown in Figure 6-2 uses a time-dependent current source with an essentially infinite drive resistance; hence it does not suffer model inaccuracies introduced when $Z_{net} >> R_d$. It achieves high accuracy by not modeling the transistor behavior at all, but instead mapping the arbitrary transistor behavior for lumped loads to that for an arbitrary detailed parasitic network.

Figure 6-2 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 6-3. 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 6-3 Output Current and Voltage Versus Time and Load

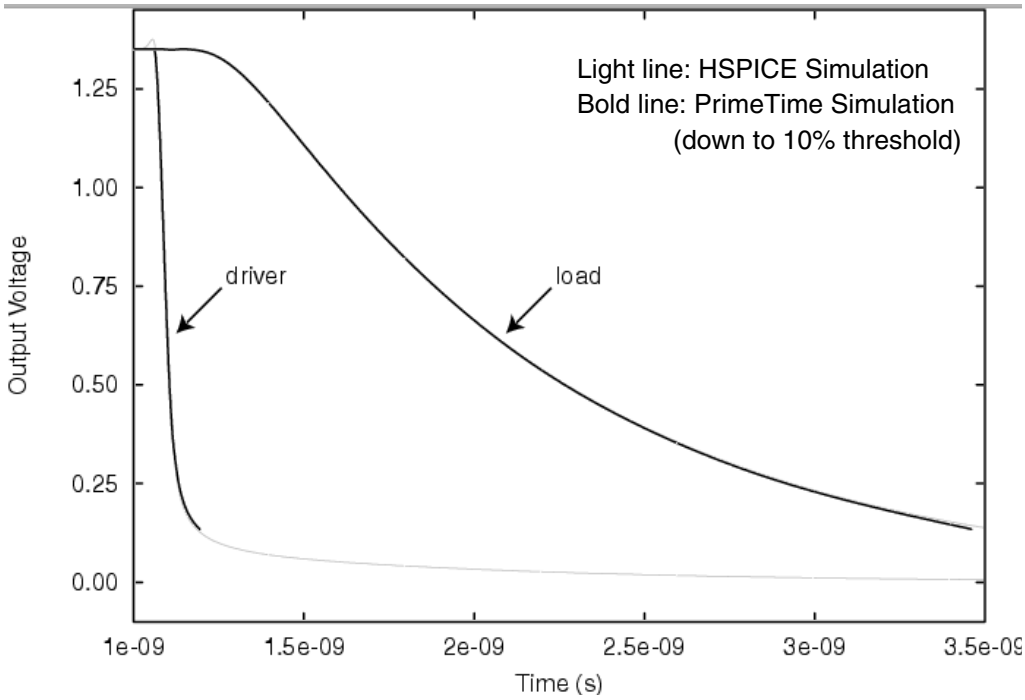


Now consider driving a detailed parasitic network. At a given time step, the output currents can be applied from the characterization to the network. There is a unique current that produces the same voltage on both a lumped capacitance and the network at the given time step. This current is the chosen 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 6-4. The comparison is between transistors connected to detailed parasitics and the CCS model connected to an Arnoldi reduction of the detailed parasitics.

Figure 6-4 CCS Model Accuracy

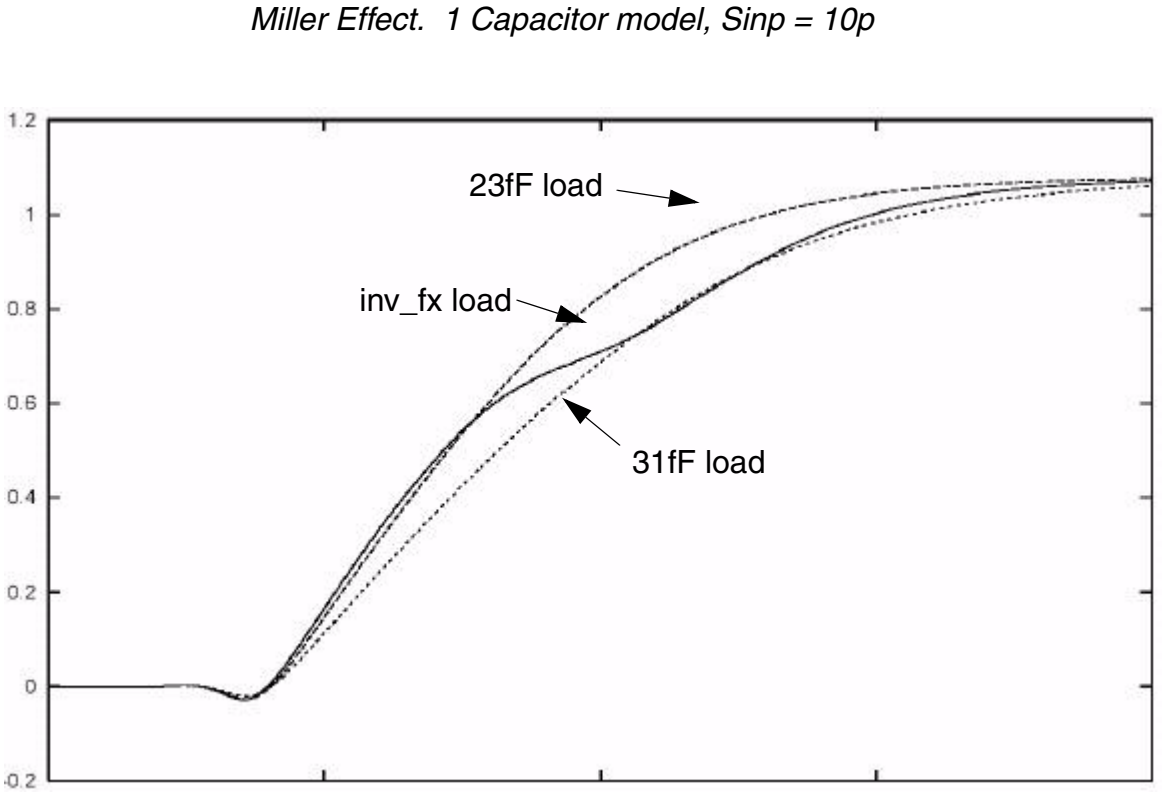


Receiver Models

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 6-5. 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 6-5 Single-Capacitance Model Versus Actual Switching Waveform



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 6-6.

Figure 6-6 CCS Receiver Model Versus Actual Switching Waveform

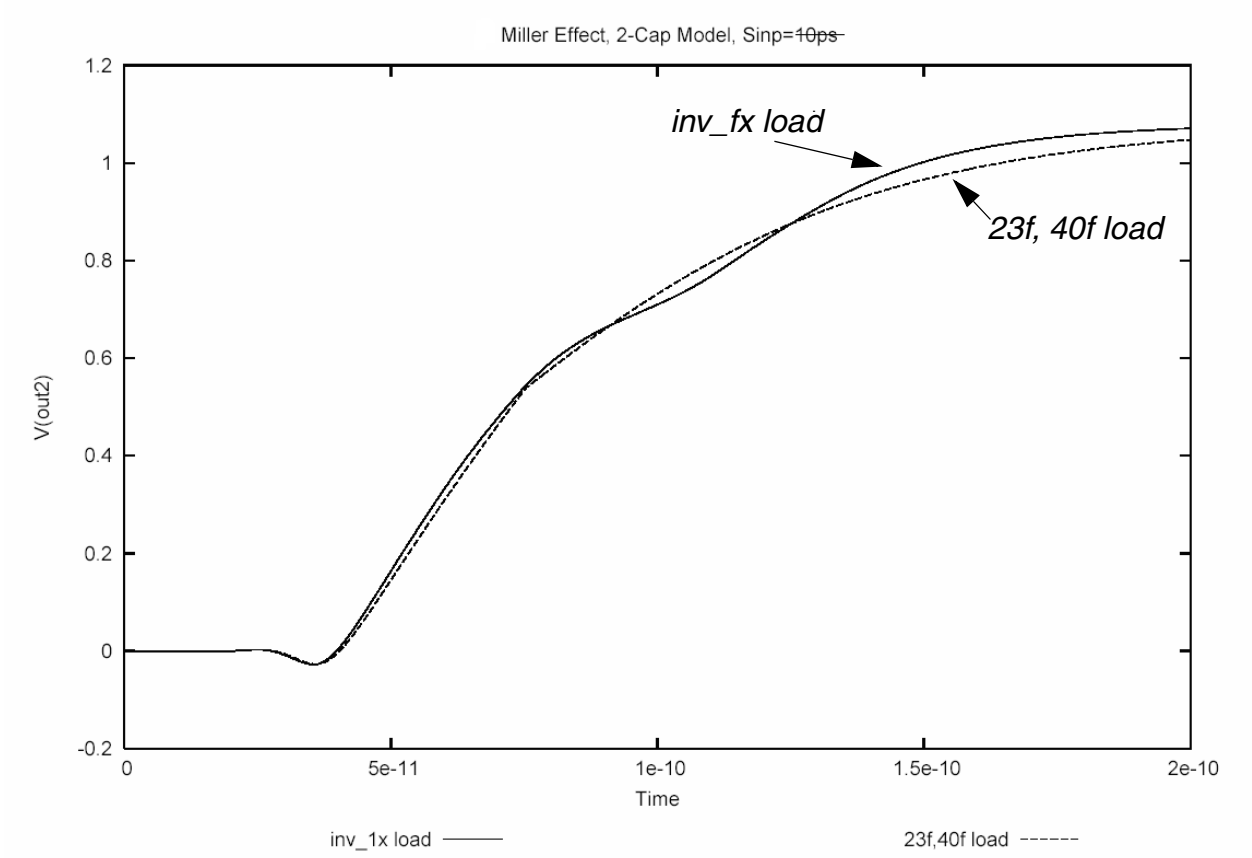


Figure 6-7 and Figure 6-8 show the stage delay and slew values calculated for a timing arc by PrimeTime using CCS timing models versus the same parameters calculated by HSPICE. The values calculated by PrimeTime using CCS timing models are within two percent of HSPICE.

Figure 6-7 CCS (PrimeTime) Versus HSPICE Delay Calculation

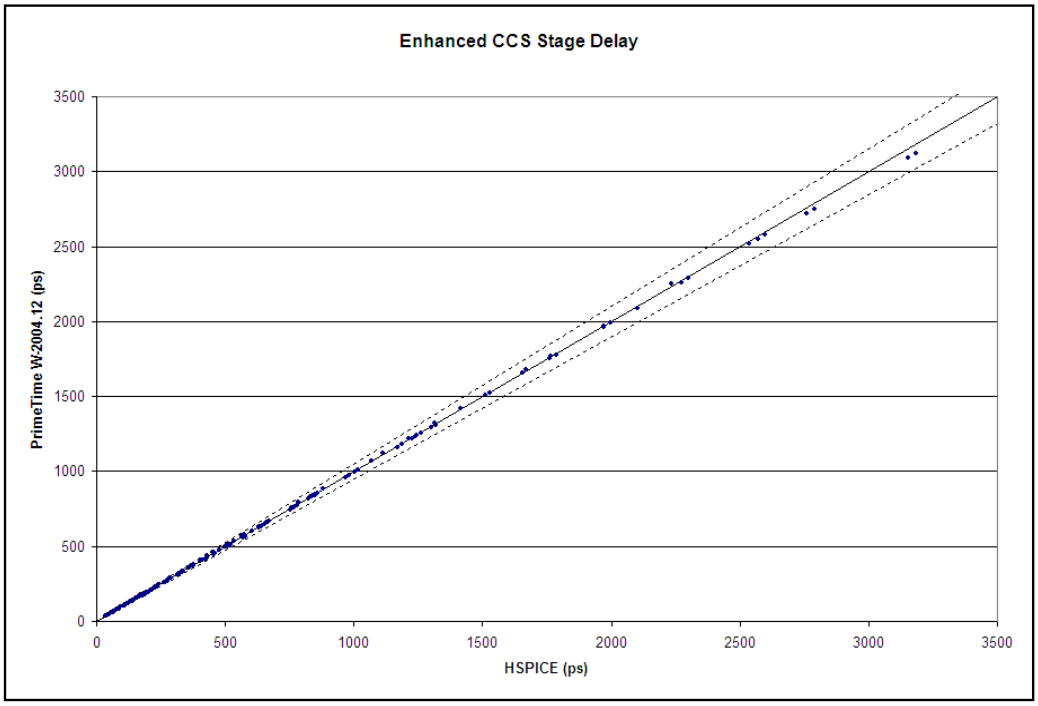
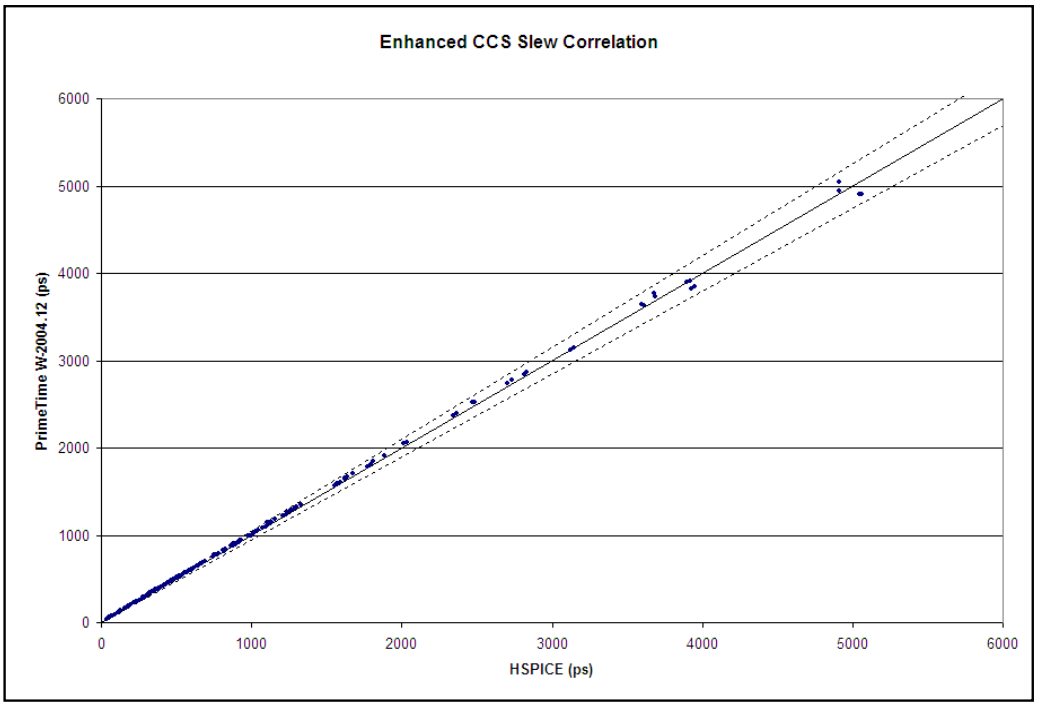


Figure 6-8 CCS (PrimeTime) versus HSPICE Slew Calculation



7

Troubleshooting

This chapter contains the following sections:

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 2-6.

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:

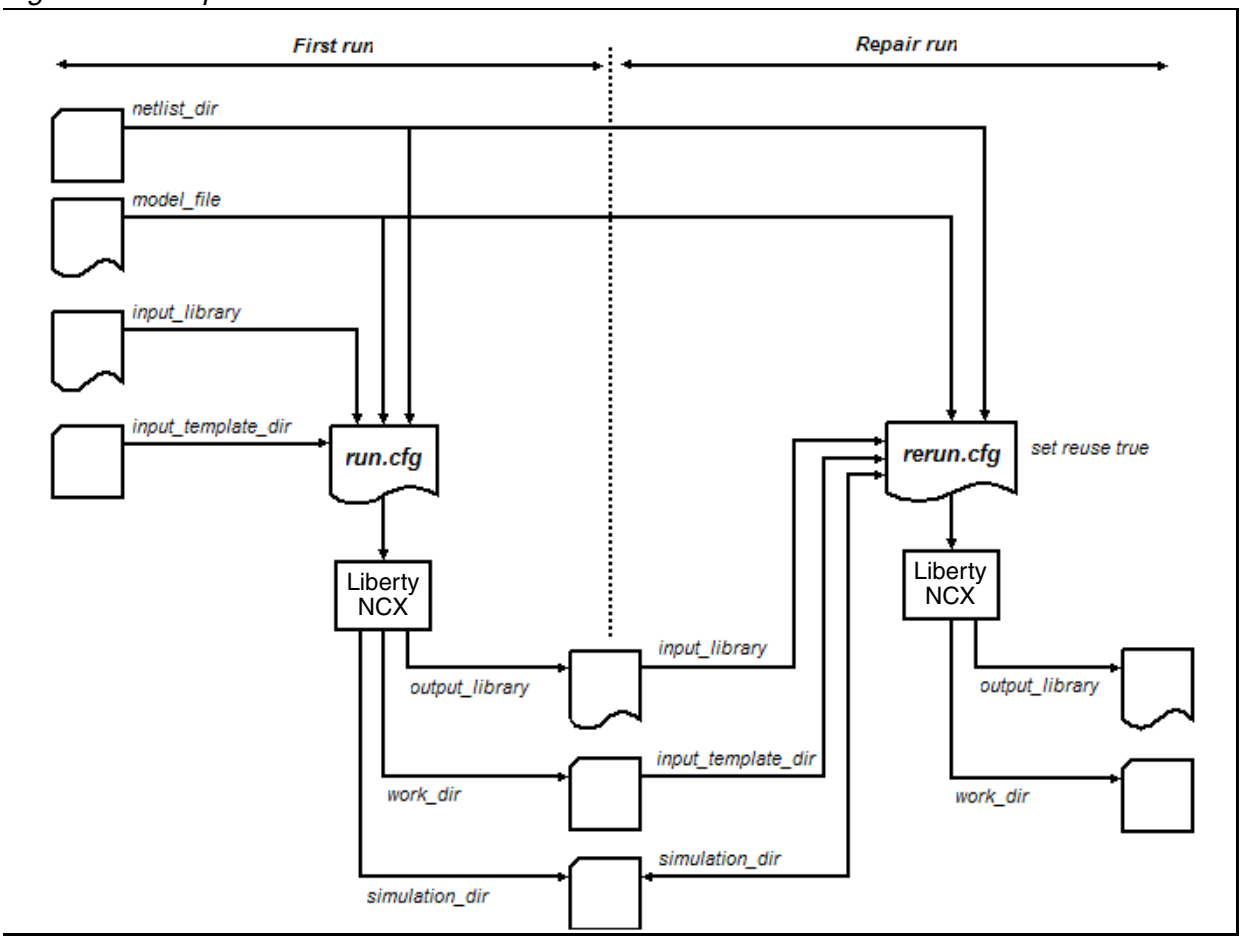
- The failed cell is not written to the `output_library`, but all other successful cells are written to the `output_library`.
- The library and cell templates are written to the `work_dir` and are set up to rerun only the failed cell.
- The entire cell simulation subdirectory is retained in the `simulation_dir`, and the successful results can be reused.

To run the repair flow:

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 7-1 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 process/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 command 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 command 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 3-8. 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 command file.

- How can I specify a different location for the simulation database?

  Use `set simulation_dir` in the command 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 command file.

- How can I create a set of templates for an existing library?

  Use `set templates true` in the command file. The templates are written to the working directory or to the directory specified by the `output_template_dir` setting in the command 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 command file, or in the current working directory if this option is not used.

- How can I override delay/slew thresholds in the input library?

  Create a library/cell template for input to Liberty NCX, as appropriate, and enter the new value as described in the section "Characterization Attributes" on page 3-8.

- How do I run Liberty NCX on a local machine?

  Use `set farm_type NoFarm` in the command 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 command file.

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z