Make CCS Noise User Guide

Version D-2009.12, December 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

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

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

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

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

Trademarks (™)

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

Service Marks (^{SM})

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

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

Contents

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* and the *PrimeTime SI Release Notes* in SolvNet.

To see the *Liberty NCX Release Notes* and the *PrimeTime SI Release Notes*,

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

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select Liberty NCX or PrimeTime Suite, and then select a release in the list that appears.

About This Guide

This user guide describes Make CCS Noise, a library characterization utility that characterizes library cells for noise response and adds CCS Noise modeling information to a technology library.

Audience

This user guide is for engineers who create cell libraries for use by products such as the PrimeTime and PrimeTime SI tools.

Conventions

The following conventions are used in Synopsys documentation.

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

Customer Support

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

Accessing SolvNet

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

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

https://solvnet.synopsys.com

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

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

Contacting the Synopsys Technical Support Center

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

- Open a call to your local support center from the Web by going to 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/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

CCS Noise Model Generation

Make CCS Noise is a standalone program that characterizes library cells for noise response and adds CCS Noise modeling information to a technology library. A library with CCS Noise models can be used with the PrimeTime SI tool to check for timing and logic failures caused by noise.

This chapter contains the following sections:

Overview of CCS Noise Library Generation

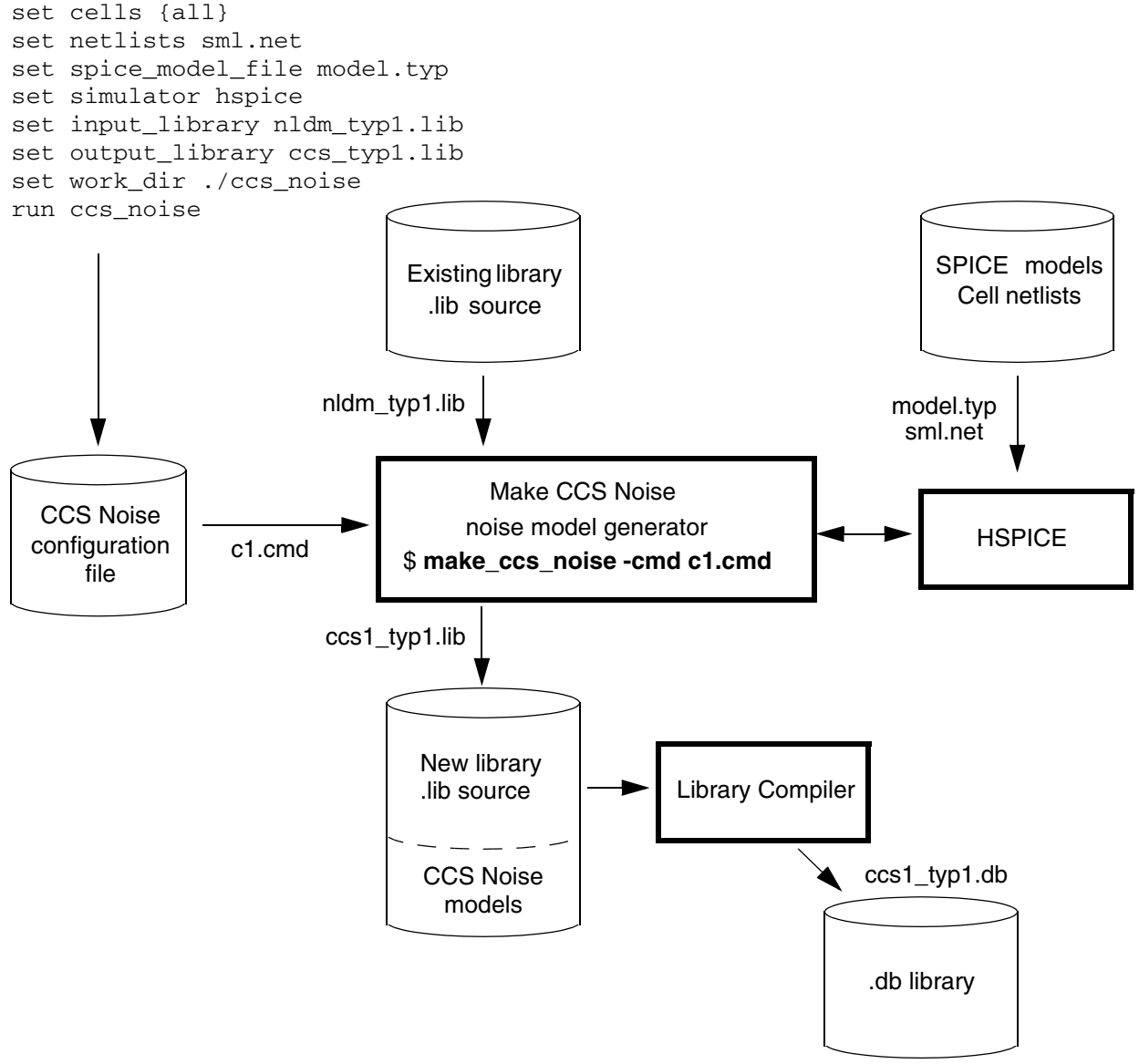
Make CCS Noise adds CCS Noise modeling information to a technology library in Liberty (.lib) format. CCS Noise models in a library enable PrimeTime SI to accurately analyze the delay and logic effects of crosstalk noise on the characterized cells. Noise characterization using Make CCS Noise is orders of magnitude faster than previous-generation characterization tools.

To characterize a library cell for noise, Make CCS Noise builds a model of each cell, using SPICE circuit elements, and simulates the effects of different slews and noise bumps at the cell inputs. Based on the simulation results, Make CCS Noise generates a set of equivalent time-dependent current-source models that represent the behavior at the cell outputs. PrimeTime SI uses these models to accurately predict the response of the cell to noise.

Make CCS Noise is a standalone program that runs under the UNIX or Linux operating system. To use Make CCS Noise, you need an existing technology library in Liberty (.lib) source format, SPICE netlists for the cells, and SPICE models for the circuit elements in the cell netlists. You also need Library Compiler and the HSPICE circuit simulator. A configuration file specifies the parameters for noise characterization such as the input file names, output file name, and list of cells to be characterized. Larger noise characterization jobs can be run on multiple processors. A configuration file setting specifies the list of host CPUs.

Figure 1-1 shows the data flow in a typical CCS Noise characterization session.

Figure 1-1 Correlation Data Flow



Required Licenses and Setup

To invoke the Make CCS Noise program, you need either a Liberty NCX license or a PrimeTime SI license to be available. After you invoke Make CCS Noise, you need a Library Compiler license and an HSPICE license. Make CCS Noise uses a single Library Compiler license even for distributed processing on multiple hosts, plus one HSPICE license for each host used during characterization.

The Make CCS Noise program is installed automatically with Liberty NCX. However, before running Make CCS Noise, you must set the `SYNOPSYS_NCX_ROOT` environment variable to the specific path where Liberty NCX was installed, as shown:

```
% setenv SYNOPSYS_NCX_ROOT $synopsys
```

where *`$synopsys`* is the directory where Liberty NCX was installed, as specified in the Synopsys installer script.

You must also set the `path` environment variable to the appropriate platform, as shown:

```
% set path = ($synopsys/ccsn/platform $path)
```

where *`platform`* is the name of the platform on which `make_ccs_noise` is being invoked. The supported platforms are `linux`, `sparcOS5`, `sparc64`, `suse32`, `suse64`, or `amd64`.

You no longer need to set the `MTB_TPATH` or the `CCSN_TPATH` environment variables to `$synopsys/ccsn/`*`platform`*. The `MTB_TPATH` and `CCSN_TPATH` environment variables are not required.

Note:

You no longer need to set the Perl environment to run Perl scripts. Liberty NCX infers the Perl path automatically.

Running Make CCS Noise

After Make CCS Noise is installed, you run the program by entering `make_ccs_noise` at the operating system prompt:

```
$ make_ccs_noise -cmd configuration_filename
```

Here is a typical `make_ccs_noise` session:

```
$ make_ccs_noise -cmd nc.config
Noise Library Generation utility for PrimeTimeSI
  Version    Current R&D version
  Pid        19986  Fri Feb 17 14:02:17 2006
  Started as "/remote/sg/ww/Y-2006.03/project/bin/
    linux/REL/si_lib_gen.pl
 -lib_in nldm_typ1.lib -lib_out ccs_typ1.lib
    -config nanochar.cfg"
  Host       "nclnx03"
  CWD        "/remote/wws2/all/Lab9/Solution"


No processing mode specified or recognised in config

Using Single-Processing Mode on "nclnx03"

Using installation in : /remote/synopsys/bin/linux/REL
 1) Initialising environment                  ...Succeeded
 2) Compiling user .lib file                  ...Succeeded
 3) Extracting attributes from user .lib file ...Succeeded
 4) Parsing user .lib file                    ...Succeeded
 5) Generating verilog file                   ...Succeeded
 6) Pre-driver characterisation               ...Succeeded
 7) Generate Spice Decks                      ...
    ELAPSED TIME to sensitize cell BUFFERD1
       was   0 seconds  [66.9 MB] - Fri Feb 17 14:02:18 2006
    ELAPSED TIME to sensitize cell BUFFERD15
       was   4 seconds  [67.8 MB] - Fri Feb 17 14:02:22 2006
    ELAPSED TIME to sensitize cell DFF
       was  19 seconds  [68.0 MB] - Fri Feb 17 14:02:41 2006
    ELAPSED TIME to sensitize cell INVERTERD1
       was   0 seconds  [68.2 MB] - Fri Feb 17 14:02:41 2006
    ELAPSED TIME to sensitize cell INVERTERD8
        was   1 seconds  [68.7 MB] - Fri Feb 17 14:02:42 2006
Succeeded
 8) Characterising cells                      ...
 8) Characterising cells                      ...
    ELAPSED TIME to characterize BUFFERD1/A_2_1_10001-0
     was 3 seconds  [68.8 MB] - Tue Feb 21 13:32:13 2006
    ELAPSED TIME to characterize BUFFERD1/1_2_Y_10002-1
     was 2 seconds  [68.8 MB] - Tue Feb 21 13:32:15 2006
    ELAPSED TIME to characterize BUFFERD15/A_2_15_10003-0
     was 6 seconds  [68.8 MB] - Tue Feb 21 13:32:21 2006
```

```
    ELAPSED TIME to characterize BUFFERD15/15_2_Y_10004-1
     was 4 seconds  [68.8 MB] - Tue Feb 21 13:32:25 2006
    ELAPSED TIME to characterize INVERTERD1/A_2_Y_10005-2
     was 2 seconds  [68.8 MB] - Tue Feb 21 13:32:27 2006
    ELAPSED TIME to characterize INVERTERD8/A_2_Y_10006-1
     was 3 seconds  [68.8 MB] - Tue Feb 21 13:32:30 2006
Succeeded
 9) Add Noise Data to Library                 ...Succeeded
10) Write noise .lib file                     ...Succeeded
11) Compiling noise .lib file                 ...Succeeded
12) Cleanup working environment               ...Succeeded

Messages generated during characterisation
None.

Detection of CCS-N coverage
  Library has 5 cells.
  Characterization done on 5 cells, checking those cells.

Library Characterisation Complete
Total Spice Decks Generated       : 6
Total Cells Processed             : 5
Total messages generated          : 0

Breakdown of time taken (seconds)
  Initialisation                  : 0
  Compiling user .lib to .db      : 0
  Extracting attributes from .lib : 0
  Reading .lib                    : 0
  Generating Verilog              : 0
  Pre-driver characterisation     : 0
  Generating spice decks          : 5
  Simulating                      : 22
  Adding noise data to library    : 0
  Write Noise Library             : 0
  Compiling noise .lib to .db     : 2

  Total Time Taken                : 29

$
```

The `make_ccs_noise` command must have at least a `-cmd` *`filename`* parameter to specify the configuration file for the command. It can also have any of several options:

- The `-h` or `-help` option displays a summary of the command options and configuration file settings.
- The `-library` *`libname`* option specifies the name of the input library (same as using `set input_library libname` in the configuration file).
- The `-olibrary` *`libname`* option specifies the name of the output library (same as using `set output_library libname` in the configuration file).

- The `-work_dir` *`dirname`* option specifies the name of the working directory (same as using `set work_dir dirname` in the configuration file).
- The `-log` *`filename`* option specifies the name of the log file produced by the command. By default, the log file name is based on the release number and date of usage.

CCS Noise Configuration File

You must prepare the configuration file before you use `make_ccs_noise`. The configuration file is an ASCII-format script containing a sequence of commands to be executed by the `make_ccs_noise` command.

There are two basic configuration file commands, `set` and `run`. A `set` command sets a parameter for CCS Noise model generation. A `run` command invokes `ccs_noise` to generate CCS Noise models.

Here is a typical CCS Noise configuration file:

```
set cells {all}
set netlists sml.net
set spice_model_file model.typ
set simulator hspice
set input_library nldm_library.lib
set output_library ccs_noise_library.lib
run ccs_noise
```

The configuration file can contain the following commands:

```
set allowed_bit                 bit_name_pattern
set allowed_bus_pattern         \[num\]$
set bit_blasted_bus_pattern     bit_name_pattern
set cells                       {all}|{cell_list}|{filename}
set cleanup                     yes | no
set ccr_max_fanout              number
set ccsn_statedep               no | yes
set ccsn_use_status_sel         no | yes
set dc_min                      float
set dc_max                      float
set device_model                {macro_name1:device_type ...}
set exclude_part                part_name_pattern
set grd_options                 user-defined_options
set ground_rail                 {rail_node_name=rail_value}
set ignore_small_cap            float
set ignore_small_res            float
set inc_file                    filename
set input_library               filename
set job_timeout                 time_in_seconds
set lc_shell_path               {lc_shell_exec [arguments]}
set lsf_options                 user-defined_options
```

```
set max_arcs_per_pin              number
set model_unbuffered_output       yes | no
set model_pass_gates              yes | no
set nbq_options                   user-defined_options
set netlists                      path | filename
set netlist_filter                filter_pattern
set node_short_to_vdd             node_name
set output_library                filename
set override_input_voltage_file   filename
set perl_path                     path
set power_rail                    {rail_node_name=rail_value}
set process_host                  host_list
set pt_shell_path                 {pt_shell_exec [arguments]}
set rsh_exec                     path_to_rsh/ssh executable
set run_post_process              no | yes
set simulator                     hspice
set simulator_path                {simulator_exec [arguments]}
set skip_pin_list                 {cell1=pin1,pin2,...
                                   cell2=pin1,pin2,... ...}
set spice_model_file              filename
set tran_time                     float
set work_dir                      dir_name
run ccs_noise
```

Table 1-1 lists and briefly describes the CCS Noise configuration file commands.

Table 1-1 Configuration File Commands

Command	Description
`set allowed_bit` *bit_name_expression*	Specifies the bits that Make CCS Noise acquires for characterization of a bus. Make CCS Noise acquires the bits whose names match the specified pattern (a regular expression). It applies the CCS noise characterization data to all bits in the same bus, as specified by the parameter `bit_blasted_bus_pattern`.
`set allowed_bus_pattern` *string*	Specifies which bus pins are to be characterized, using naming conventions in Perl format: `\[`*num*`\]$` . The default is `\[0\]$` .
`set bit_blasted_bus_pattern` *bit_name_expression*	Specifies the names of pins that belong to a bus. If a bus pin in a library is bit-blasted (no bus group is defined), Make CCS Noise regards all pins that match the specified pattern (a regular expression) as belonging to a bus. This option can be used with the `allowed_bit` setting to choose the bits for characterization.
`set cells { all } | { cell1 cell2 ... } | { filename }`	The cells for which to generate CCS Noise models: all of the cells in the library, a list of cell names separated by spaces, or the name of a file containing a list of cell names.

Table 1-1 Configuration File Commands (Continued)

Command	Description
`set cleanup` `  yes | no`	Specifies whether to delete (`yes`) or retain (`no`) the working directory containing the intermediate files generated by the characterization process. The default is `yes`.
`set ccr_max_fanout` `  number`	Specifies the maximum fanout from a channel-connected block being cut from the cell's netlist. The default is 6.
`set ccsn_statedep` `  no | yes`	Specifies whether to use the sdf_when condition in a timing group to perform CCS noise characterization for the "from" pin and "to" pin of that timing arc. If `yes`, the sdf_when condition is used; otherwise, it is ignored.
`set ccsn_use_status_sel` `  no | yes`	Specifies whether to choose any channel-connected block from multiple blocks connected to the same port pin (`yes`) or to choose the block having the smallest rank signature (`no`).
`set dc_min` *float*	Specifies the DC table range in the scale of VDD. The value is any floating-point number from -0.5 to -1.0. The default value is -1.
`set dc_max` *float*	Specifies the DC table range in the scale of VDD. The value is any floating-point number from 1.5 to 2. The default value is 2.

`set device_model`
`{`*macro_name1*`:`*device_type*
macro_name2`:`*device_type*
`...}`
	Specifies how to translate devices into their corresponding macros for CCS Noise simulation.

`set exclude_part` `  part_name_pattern`	Causes Make CCS Noise to ignore all parts in the SPICE netlist whose names match the specified expression written as a Perl text parsing pattern. For example, the expression ^ab.* causes all parts beginning with "ab" to be ignored. This option can speed up the parsing of a cell's SPICE netlist, but the excluded parts must not be needed for CCS noise characterization.
`set ground_rail` `{`*rail_node_name= rail_value*`}`	Specifies the name and voltage of a ground rail.
`set grd_options` `  user-defined_options`	Overrides the default Global Resource Director (GRD) options for distributed processing.
`set ignore_small_cap` *float*	Specifies a capacitance value under which the capacitance in the circuit will be ignored during channel-connected block generation. The value is a floating-point number in farads. The default value is 1e-18.

Table 1-1 Configuration File Commands (Continued)

Command	Description
`set ignore_small_res` *`float`*	Specifies a resistance value under which the resistance in the circuit will be ignored during channel-connected block generation. The value is a floating-point number in ohms. The default value is 1.
`set input_library` *`filename`*	The name of the Liberty (.lib) file to be modified by the addition of CCS noise models. This setting serves the same purpose as the `-library` option for the `make_ccs_noise` command. The configuration file setting has priority over the `make_ccs_noise` command option.
`set inc_file` *`filename`*	The name of the SPICE “include” file, which is attached to the SPICE deck used for circuit simulation.
`set job_timeout` *`time_in_seconds`*	The timeout period for each job submitted for distributed processing. The timeout period starts when each simulation job is assigned for execution on a host. When the specified number of seconds has elapsed, characterization stops and the data characterized so far is added to the noise library. The default is `10800`.
`set lc_shell_path` *`lc_shell_exec`* `[` *`options`* `]`	Specifies the full absolute path to the Library Compiler (lc_shell) executable, plus any options to the `lc_shell` command, to be used during CCS noise characterization. In the absence of this configuration setting, Make CCS Noise uses the path *$NC_install_dir*/bin_utils/*$arch*/lc_shell, where *$NC_install_dir* is the Make CCS Noise installation path and *arch* is the machine architecture name.
`set lsf_options` *`user-defined_options`*	Overrides the default Load Sharing Facility (LSF) options for distributed processing.
`set max_arcs_per_pin` *`number`*	Specifies the maximum allowable number of generated timing arcs from a port pin to internal nodes, or from internal nodes to a port pin. This limit prevents an excessive number of arcs from being characterized. The default is 3.
`set model_unbuffered_output` `yes | no`	Specifies whether Make CCS Noise models latches with unbuffered output. The default is yes.
`set model_pass_gates` `yes | no`	Specifies whether Make CCS Noise models cells with pass gates. The default is yes.
`set nbq_options` *`user-defined_options`*	Overrides the default NetBatch queue options for distributed processing.

Table 1-1 Configuration File Commands (Continued)

Command	Description
`set netlists` *path* | *filename*	The path to the directory containing SPICE netlists of the cells being analyzed, or the name of file containing a list of SPICE netlist files for all cells. If a path is specified, there must be one netlist file named *cellname*.* per cell (where * represents any string).
`set netlist_filter` *filter_pattern*	Specifies a Perl filtering pattern for converting netlist names from one naming format to another, such as from pin names D<0>, D<1>, ... to pin names D[0], D[1], ... by using the filter pattern `tr/<>/\[\]/` .
`set node_short_to_vdd` *node_name*	Causes Make CCS Noise to discard the specified .subckt node and connects that node internally to Vdd. This feature supports the definition of an output signal level using an analog output, for example, `set node_short_to_vdd OUT_LEVEL` .
`set output_library` *filename*	The name of the new Liberty (.lib) file produced by adding CCS noise models. This setting serves the same purpose as the `-olibrary` option of the `make_ccs_noise` command. The configuration file setting has priority over the `make_ccs_noise` command option.
`set override_input_` `voltage_file` *filename*	The name of a file containing information about overriding CCS noise model generation for analog or differential input pins of the cells being characterized.
`set perl_path` *path*	The path to the Perl interpreter used by `make_ccs_noise`. The Perl interpreter must be 64-bit compiled version 5.8.3 or later.
`set power_rail` {*rail_node_name= rail_value*}	Specifies the name and voltage of a power rail.
`set process_host` *host_list*	The host list used for distributed processing of `make_ccs_noise`. The format of the host list looks like this: *dis_type*(*sub_machine*):*max_jobs*:*cpu_type*,...
`set pt_shell_path` *pt_shell_exec* [*options*]	Specifies the full absolute path to the PrimeTime (pt_shell) executable, plus any options to the `pt_shell` command, to be used during CCS noise characterization. In the absence of this configuration setting, Make CCS Noise uses the path *$NC_install_dir*/bin_utils/*$arch*/pt_shell, where *$NC_install_dir* is the Make CCS Noise installation path and *arch* is the machine architecture name.

Table 1-1 Configuration File Commands (Continued)

Command	Description
`set rsh_exec` *`path_to_rsh/ssh executable`*	Overwrites the path to the rsh executable and uses ssh (secure shell) instead of rsh (remote shell) to log in remotely to the host machine when distributed processing is enabled. The host machine is defined by the `process_host` command.
`set run_post_process`	Specifies whether to run a post-process analysis to generate a CCS noise model when initial characterization fails.
`set simulator hspice`	The type of SPICE simulator to be used. Currently only `hspice` is supported.

`set simulator_path`
`simulator_exec`
`[` *`options`* `]`
	Specifies the full absolute path to the HSPICE simulator executable, plus any options to the simulator command, to be used during CCS noise characterization. In the absence of this configuration setting, Make CCS Noise uses the path *$NC_install_dir*/bin_utils/*$arch*/hspice, where *$NC_install_dir* is the Make CCS Noise installation path and *arch* is the machine architecture name.

`set skip_pin_list` `{`*`cell1=pin1,pin2,...`* *`cell2=pin1,pin2,... ...`*`}`	Skips noise characterization of specified pins of specified cells (for example, analog input pins). Use one or more spaces between each cell or each pin in the list. Do not use any spaces between the equal sign or names within a cell list or a pin list. For example: `set skip_pin_list {acom1=a1,a2 acom2=aa,ab}` The CCS noise model defines each skipped pin with the attribute `is_needed: false` in the Liberty CCS noise model.
`set spice_model_file` *`filename`*	The name of the file containing the SPICE models for the circuit elements used in the cell netlists.
`set sync_timeout` *`time_in_seconds`*	The total timeout period for distributed processing. The timeout period starts when the first simulation job is assigned for execution on a host. When the specified number of seconds has elapsed, characterization stops and the data characterized up to that point is added to the noise library. The default is 86400.
`set tran_time` *`float`*	Specifies the transient analysis time in nanoseconds. The value is a floating-point number. The default value is 10.

Table 1-1 Configuration File Commands (Continued)

Command	Description
`set work_dir` *`dir_name`*	The name of the working directory used for writing files used for CCS Noise characterization. This setting serves the same purpose as the `-work_dir` option of the `make_ccs_noise` command. The configuration file setting has priority over the `make_ccs_noise` command option.
`run ccs_noise`	Runs CCS Noise characterization.

Required Configuration Commands

The configuration file must contain the following commands:

- `set cells`
- `set netlists`
- `set spice_model_file`
- `set input_library`
- `set output_library`
- `set perl_path`

The other commands in Table 1-1 are optional. The configuration file must end with a `run ccs_noise` command, which starts the noise characterization process.

The `set cells` command specifies the cells in the library that are to have CCS Noise information added: all the cells in the library, a list of cells, or the name of a separate file that lists the cells.

There must be a SPICE netlist for each cell to be characterized for noise. The `set netlists` command specifies either the path to a directory containing the SPICE netlist or the name of a file containing the SPICE netlists. There must also be a file containing the SPICE models for the circuit elements used in the SPICE netlists. The `set spice_model_file` command specifies the name of this file.

The `set input_library` command specifies the name of the Liberty (.lib) file that is to be modified by the addition of CCS noise models. The `set output_library` command specifies the name of the new Liberty (.lib) file produced by adding the new models.

Make CCS Noise uses some Perl code internally, which requires a Perl interpreter, 64-bit compiled version 5.8.3 or later. The `set perl_path` command specifies the path to the Perl interpreter.

Bus Patterns

The `set allowed_bus_pattern` command specifies which bus pins are to be characterized, using naming conventions in Perl format: `\[num\]$` . For example, to characterize bit number 3 of a bus pin, if the bus naming style is `%s[%d]`, set the parameter to the string `\[3\]$`. If the input library uses a different bus naming style, use the appropriate string. For example, if the bus naming style is `s(%d)`, set the parameter to `\(3\)$` . The default is `\[0\]$` .

The `set bit_blasted_bus_pattern` command specifies the names of pins that belong to a bus. By default, if a bus pin in a library is bit-blasted (no bus group is defined), each bit is treated as a single pin. However, Make CCS Noise considers all pins that match the specified pattern (a regular expression) to belong to a bus. For example, the pattern `[^\d]\d+$` causes pin names ending in one or more digits to be considered bus bits. This option can be used together with the `allowed_bit` setting to choose the bits for acquisition.

The `set allowed_bit` command specifies the bits that Make CCS Noise acquires for characterization of a bus. Make CCS Noise acquires the bits whose names match the specified pattern and applies the CCS noise characterization data to all bits in the same bus, as specified by the `bit_blasted_bus_pattern` parameter. For example, if `allowed_bit` is set to the string `[^\d]0$`, only the bus pin names ending with a zero character are acquired for characterization.

Here is another example. Suppose that a cell has pins A1 and A2, bit-blasted bus B0 through B7, and bit-blasted bus C0 through C5. Bits B4 and C3 have typical electrical characteristics that can be acquired and characterized, and can have their CCS noise modeling applied to the remaining bits of bus B and bus C, respectively. To carry out characterization in this manner, use the following configuration file commands:

```
set bit_blasted_bus_pattern  '^[BC]\d+$'
set allowed_bit              '^(B4|C3)$'
```

For a bus pin written with square brackets, such as `%s[%d]`, Make CCS Noise writes the data into a `bus()` group. For a bit-blasted bus pin, Make CCS Noise writes the data separately for each bus pin. For example, for a bus pin named `D[31:0]`, the CCS noise information is written to the library in the following form:

```
bus (D) {
   ...
   pin(D[31:0]) {
     ...
   }
```

```
    ccsn_first_stage() {
      ...
    }
  }
```

For a bit-blasted bus pin, the CCS noise information is written to the library as follows:

```
pin(D31) {
  ccsn_first_stage() {
    ...
  }
  ...
}
pin(D30) {
  ccsn_first_stage() {
    ...
  }
}
...
pin(D0) {
  ccsn_first_stage() {
    ...
  }
}
```

The latter method requires more runtime.

Channel-Connected Region Maximum Fanout

The `set ccr_max_fanout` command specifies the maximum fanout from a channel-connected region being cut from the cell's netlist. The fanout of a node is the number of transistors whose source or drain is connected to the node directly or through resistors. If the fanout exceeds the specified number, Make CCS Noise stops path tracing at that node and includes all the fanout transistors in the off state. The default maximum fanout is 6.

Channel-Connected Block Selection

The `set ccsn_use_status_sel` command specifies whether to choose the channel-connected block from multiple blocks connected to the same port pin based on matching the simulation data to the current-model data (`yes`) or based on the block with the smallest "rank signature" (`no`). The default is `no`.

If there are multiple channel-connected blocks for the same port pin, Make CCS Noise uses the data from only one of those channel-connected blocks. If the `ccsn_use_status_sel` parameter is set to `no` (the default), Make CCS Noise chooses the block with the smallest rank signature. The rank signature is a string that describes the topology of a channel-connected block and is unique for each such block, having the following form:

<num_passgate>_<num_input>_<num_mos_in_1stage>_
<num_mos_in_2stage>_<distance_from_to>_
<input_node>_<output_node>

The string contains the number of pass gates in the block, the number of input nodes of the block, the number of transistors in the first stage of the block, the number of transistors in first two stages of the block, the number of transistors traced from the input node to the output node, the input node name, and the output node name. This method selects the same channel-connected block for multiple CCS Noise characterization runs for different corners of the same library, which is necessary for generating scalable CCS Noise data in the libraries.

If `ccsn_use_status_sel` is set to `yes`, Make CCS Noise selects the channel-connected block whose simulation data most closely matches the calculated data generated by the internal gate-current model.

Device Models

The `device_model` parameter specifies the device type associated with each device or macro used in the SPICE netlist, so that Make CCS Noise can correctly recognize and use those devices or macros during netlist parsing.

Some foundries use macros with additional parameters to represent transistors, resistors, or other devices. These macros are defined in the SPICE model file. When Make CCS Noise parses the cell's netlist and encounters these macros, it needs to recognize each macro as a device. Make CCS Noise treats each macro as a PMOS or NMOS transistor, resistor, or other device, according to the device type specified by the `device_model` setting; or by the second character used in the instance name if no device type is specified.

For example, an NMOS transistor is defined in the SPICE model file as follows:

```
.subckt nmos_lvt drain gate source sub
+ w=...
.ends nmos_lvt
```

The transistor is instantiated in the SPICE netlist as follows:

```
xM001 net001 net001 GND GND nmos_lvt
```

You can define the macro `nmos_lvt` as a device wrapper macro by setting the `device_model` parameter:

```
set device_model {nmos_lvt:nmos}
```

During netlist parsing, Make CCS Noise treats the instance as a normal NMOS transistor. For CCS Noise simulation, the instance is still used as a macro.

The device type should be a valid SPICE model name listed in Table 1-2.

Table 1-2 SPICE Model Names

Model name	Description
AMP	Operational amplifier model
C	Capacitor model
CORE	Magnetic core model
D	Diode model
JFET	Junction field-effect transistor model
L	Magnetic core mutual inductor model
NJF	N-channel JFET model
NMOS	N-channel MOSFET model
NPN	NPN bipolar junction transistor model
OPT	Optimization model
PJF	P-channel JFET model
PLOT	Plot model for the .GRAPH statement
PMOS	P-channel MOSFET model
PNP	PNP bipolar junction transistor model
R	Resistor model
SP	S parameter
U	Lossy transmission-line model (lumped)
W	Lossy transmission-line model

You must define any PMOS, NMOS, and macro devices in a cell netlist. You can ignore other devices because they have no impact on the Make CCS Noise netlist parser.

If you do not define the device_model, Make CCS Noise attempts to get the device model from the SPICE model file. If the SPICE model is encrypted or the device model cannot be found in the SPICE model file, Make CCS Noise determines the device type by its instance name. For example, if the cell netlist has the following statement:

```
m001 net001 net001 GND GND nmos_lvt
```

Make CCS Noise considers the device a transistor (the instance name starts with the letter *m*) and considers it an n-channel device (the reference name contains the letter *n*).

For the statement

```
r001 net001 net002 0.1
```

Make CCS Noise considers the device a resistor (the instance name stars with the letter *r*).

If the device_model option is defined, Make CCS Noise does not parse the SPICE model file. It is recommended that you provide the device_model definition so that Make CCS Noise does not spend time parsing the SPICE model file to get the device model definitions.

The device_model parameter replaces the device_macro parameter previously used in the NanoChar CCS Noise product. However, the older device_macro keyword is still accepted. If you update a script to use the newer device_model syntax, be aware that you must specify the transistor channel type, such as pmos or nmos, rather than the m used with device_macro. For example, if you previously defined a transistor as

```
set device_macro {nch:m pch:m}
```

the tool treats nch and pch as transistors and determines the channel type from the model name (NMOS for a model name containing the letter *n* or PMOS for a model name containing the letter *p*). If you use the newer device_model syntax, you must specify the channel type explicitly, for example,

```
set device_model {nch:nmos pch:pmos}
```

State-Dependent Noise Characterization

The ccsn_statedep parameter in the configuration file specifies whether to use the sdf_when condition in a timing group to perform CCS noise characterization for the “from” pin and “to” pin of that timing arc. If ccsn_statedep is set to yes, the sdf_when condition is used; otherwise, it is ignored.

If the ccsn_statedep parameter is set to yes, you can also specify additional logic conditions under which CCS noise characterization is performed. To do so, set the NLG_ADD_WHEN_*cellname* environment variable to a logic condition string before you invoke make_ccs_noise. Then, the CCS noise characterization will be performed for the specified conditions.

For example, suppose you have a NAND cell named “nand1a” that has a logic function Y=!(A&B), and you want to perform noise characterization for the input logic conditions (!A&!B) and (A&!B). To specify these conditions, use:

```
% setenv NLG_ADD_WHEN_NAND1A "\!A&\!B; A&\!B"
% make_ccs_noise
```

or

```
% setenv NLG_ADD_WHEN_NAND1A '!A&!B; A&!B'
% make_ccs_noise
```

The environment variable name must be all uppercase. The exclamation point (!) symbol is a special character and must be preceded by a backslash (\) if you use double quotation marks (" "). The list of logic conditions must be separated by a semicolon (;).

Override Input Voltage File

Analog or differential input pins of cells are not supported by the current CCS Noise Model technology. If you run Make CCS Noise on cells containing such input pins, it reports error or warning messages for those pins and no CCS noise information is generated for them. For these pins, you need to provide a template file that specifies the contents of the input_voltage attribute for each of the applicable pins. Make CCS Noise overrides the input_voltage attribute to make the final CCS noise library for those pins, instead of generating CCS noise data for those pins.

To invoke usage of an input voltage override template file, set the override_input_voltage_file parameter in the configuration file:

```
set override_input_voltage_file filename
```

The file specifies the name of the cell, the names of the pins in the cell to override, and the input voltage information for each named pin. This is the general format of the file:

```
# comment
cell: <cell_name_1>;
pin : <pin_name_1>, <pin_name_2>, ... ;
input_voltage (<template_name>) {
    vil : float | expression;
    vih: float | expression;
```

```
    vimin : float | expression;
    vimax : float | expression;
}
# comment
cell: <cell_name_2>;
pin : <pin_name_1>, <pin_name_2>, ... ;
input_voltage (<template_name>) {
    vil : float | expression;
    vih: float | expression;
    vimin : float | expression;
    vimax : float | expression;
}
```

The keywords in the syntax are case-sensitive, but the cell names and pin names are case-insensitive. The opening brace character ({) must be on the same line as the template name.

Here is an example of an override input voltage file:

```
# Synopsys std_cell_a
cell: <std_cell_a>;
pin : <a>, <b>;
input_voltage (templ_vdd) {
    vil : 0.3 * VDD ;
    vih: 0.7 * VDD ;
    vimin : -0.5 ;
    vimax : VDD + 0.5;
}
# Synopsys std_cell_b
cell: < std_cell_b>;
pin : <I>, <OE>,...;
input_voltage (templ_vcc) {
    vil : 0.3 * VDD ;
    vih: 0.7 * VDD ;
    vimin : -0.5 ;
    vimax : VDD + 0.5;
}
```

The file specifies that there are two cells with analog or differential pins. The first cell is “std_cell_a” with pins “a” and “b,” and the input voltage template file name is “templ_vdd.” The other cell is “std_cell_b” with pins “I” and “OE,” and the input voltage template file name is templ_vcc.

Rail Voltages

Make CCS Noise accepts the name `VDD` or `VCC` as the power rail name and `VSS` or `GND` as the ground rail name, in uppercase or lowercase letters (case-insensitive). The value of the power rail is the `nom_voltage` value extracted from input library. The value of the ground rail is 0.0.

If a rail name in the netlist is different from these standard names, you can specify the rail information (name and voltage of the rail) in either the configuration file or in the input library.

To specify the rail information in the configuration file, use the following syntax:

```
set power_rail { rail_node_name=rail_value ... }

set ground_rail { rail_node_name=rail_value ... }
```

The *rail_node_name* is the name of the rail node in the SPICE netlist (case-insensitive) and the *rail_value* is the rail voltage, in volts. The information in these statements overrides any rail information in the input library.

If the input library already contains the rail information, you do not need to specify that information again in the configuration file. Make CCS Noise supports two kinds of Library Compiler power syntax: the older `power_rail` syntax and the newer `voltage_map` syntax.

In the older `power_rail` Library Compiler syntax, the library defines the power rail in a `power_supply()` group and `operating_conditions()` group. In the following Library Compiler syntax example, two global rails exist for all cells, vpower=3.3v and vground=0.0v.

```
library(foo_lib_name) {
  ...
  power_supply() {
    ...
    power_rail(vpower,3.3);
    power_rail(vground,0.0);
  }
  operating_conditions() {
    ...
    power_rail(vpower,3.3);
    power_rail(vground,0.0);
  }
  ...
  cell (foo_cell_name) {
    ...
  }
}
```

The rail names used in the library must match those in the cell netlist.

In the following example from a cell netlist, the cell has global rails `VDD` and `VSS` and its own power or ground pins `pg_pin1` and `pg_pin2`:

```
.global vdd vss
.subckt TESTCELL in out pg_pin1 pg_pin2
   ...
*mos's sub connected to pg pins
M0 vdd in out pg_pin1  PMOS ...
```

```
M1 vss  in out pg_pin2  NMOS ...
   ...
.ends TESTCELL
```

In this example, `pg_pin1` and `pg_pin2` are power or ground pins and the values are (for example) pg_pin1=3.3v and pg_pin2=0.0v. In addition, `pg_pin1` and `pg_pin2` are considered to be power or ground rails only in the cell `testcell`. For this example, you should specify the rail information in library as follows:

```
library (foo_lib_name) {
  ...
  power_supply() {
    ...
    power_rail(v1,3.3);
    power_rail(v2,0.0);
  }
  operating_conditions() {
    ...
    power_rail(v1,3.3);
    power_rail(v2,0.0);
  }
  ...
  cell(testcell) {
    rail_connection(pg_pin1, v1); /* should match name */
    rail_connection(pg_pin2, v2); /* in cell netlist */
    ...
    pin(in1) {
      ...
    }
  }
}
```

In the newer `voltage_map` Library Compiler syntax, you define the voltage map at the library level and the power or ground pin at the cell level. To get the same results as in the previous example, define the rail information as follows:

```
library(foo_lib_name) {
  ...
  voltage_map(vpower,3.3);
  voltage_map(vground,0.0);

  ...
  cell (foo_cell_name) {
    pg_pin(pg_name_1) {
      voltage_name : vpower;
      voltage_type : type;
    }
  pg_pin(pg_name_2) {
    voltage_name : vground;
    voltage_type : type;
  }
```

```
  ...
}
```

The rail names used in the `voltage_map` statement must match those in the cell netlist. The voltage types specified in the `voltage_type` statement can be `primary_power`, `primary_ground`, `back_power`, `back_ground`, and so on, according to the function of the power or ground pin.

For power syntax, the `vpower` and `vground` names in the `power_rail` statement and in the `pg_pin1` and `pg_pin2` names in the `rail_connection` statement must match the rail names in the cell netlist. Otherwise, the names `vpower`, `vground`, `pg_pin1`, and `pg_pin2` in the cell netlist are not recognized as rails.

Distributed Processing

When distributed processing is enabled in the CCS noise characterization flow, Make CCS Noise logs in remotely using rsh (remote shell) to the machine defined by the `process_host` command. The `process_host` command specifies the host used for distributed processing of the characterization job, and the LSF job launches from that host machine. The machine defined by `process_host` should be an “LSF submit host” machine that is permitted to launch jobs to the LSF farm. The hosts are specified in the following format:

dis_type(*sub_machine*):*max_jobs*:*cpu_type*,...

This format specifies the distribution type, submachine name, maximum number of jobs per host, and CPU type. For example, for distribution on a SparcOS5 host named “nova” and a Linux host named “ptd1360g,” using LSF and GRD, the format would be:

```
LSF(nova):100:sparcOS5,GRD(ptdl360g):100:linux
```

There is a maximum limit of 10 CPUs for a given job.

By default, Make CCS Noise uses the rsh executable in the /usr/bin/rsh directory to log in remotely to the host machine. However, you can overwrite the path to the rsh executable and use ssh (secure shell) by setting the `set rsh_exec` *`path_to_rsh/ssh executable`* command in the configuration file, as shown in the following example:

```
set rsh_exec /usr/bin/ssh
```

If you do not set `rsh_exec` to the path to the ssh executable, Make CCS Noise points to the /usr/bin/rsh directory and logs in using rsh.

Repair Flow: Approximate Noise Model for Failing Pins

Make CCS Noise can fail to generate a noise model for a given pin or arc for any of several reasons, such as the inability to find a channel-connected block or a simulation failure. In such an occurrence, you can optionally have Make CCS Noise generate an approximate noise model for the pin or arc. To do so, use the following command in the configuration file:

```
set run_post_process yes
```

The default `run_post_process` setting is `no`. In that case, when noise characterization fails for a pin or arc, an error is reported. However, if `run_post_process` is set to `yes`, when noise characterization fails for a pin or arc, Make CCS Noise generates an approximate noise model. It creates an effective inverter stage using the first PMOS and NMOS transistors found connected to the pin, acquires the noise characteristics of the inverter, and uses the acquired model for the pin. It also issues a warning message.

The option should be used carefully because the generated model might not be an accurate representation of the true noise characteristics, depending on the cause of the original characterization failure.

Working Directory

The `set work_dir` command specifies the name of the working directory used for writing intermediate files used for CCS Noise characterization. These files include SPICE netlists and simulations results, organized by cell.

The `set cleanup` command specifies whether to delete or retain the working directory. The default is `yes`, which causes the working directory to be deleted.

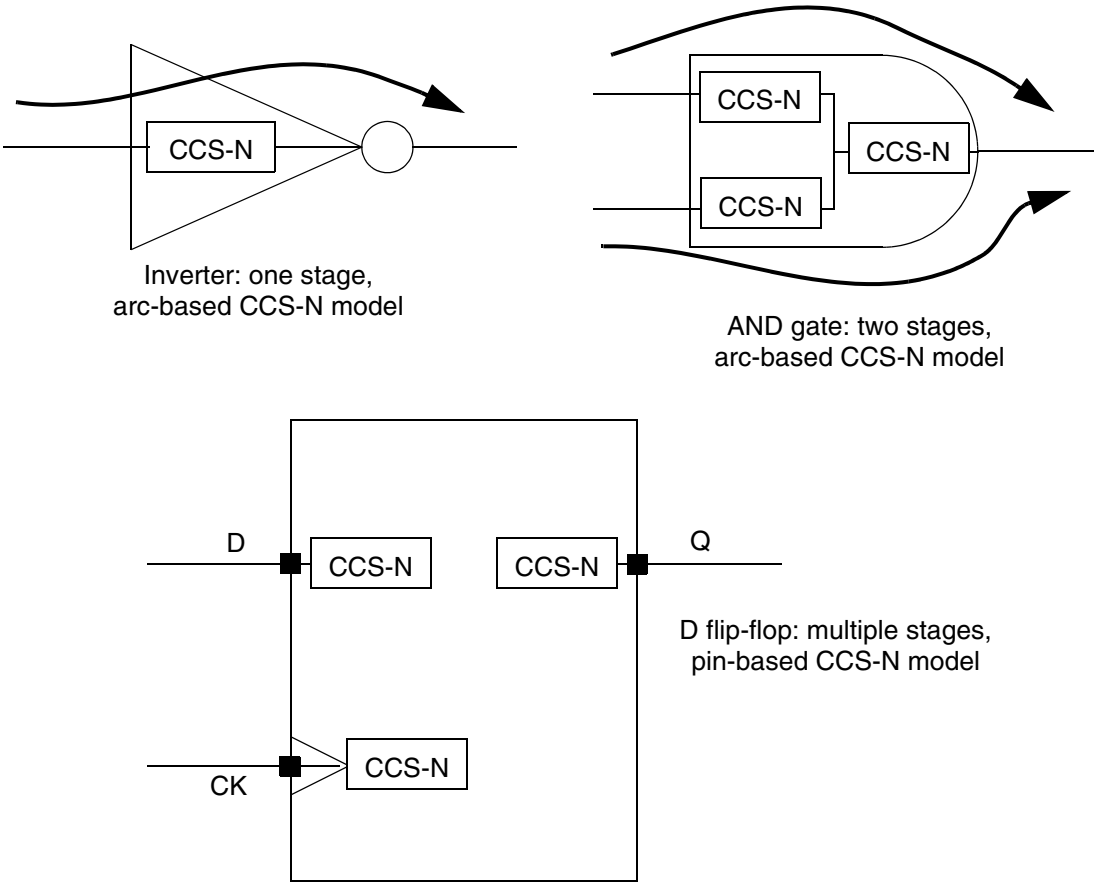
How Make CCS Noise Operates

To characterize a cell, Make CCS Noise examines the circuit netlist for the cell. It breaks down the netlist into channel-connected blocks. Each channel-connected block has a potential conducting path through transistor sources and drains.

Make CCS Noise cuts apart the netlist into separate stages at channel-connected block boundaries. A simple cell such as an inverter has only a single stage from input to output. An AND gate has two stages. A type D flip-flop has more than two stages.

A CCS noise model represents the noise response of each stage. A single-stage or two-stage cell results in an arc-based noise model. A cell with more than two stages results in a pin-based model. See Figure 1-2 on page 1-25.

Figure 1-2 CCS Noise Model Representations



To characterize a stage of a cell, Make CCS Noise first simulates the DC response of the stage netlist. It varies the input voltage and measures the output voltage to generate a simple two-dimensional table representing the output current as a function of DC input and output voltages.

Then, to determine the noise response of the stage, it simulates the circuit by using a few different ramps and glitches at the stage input. It indirectly uses the output transition and noise propagation results to derive the dynamic behavior of the stage. The behavior of each stage is then used to make the CCS Noise model for the whole cell.

During characterization, Make CCS Noise issues reports on its progress in performing a sequence of numbered steps. The steps are listed and described in Table 1-3.

Table 1-3 Characterization Steps

Characterization Step	Description
1. Initializing environment	Make CCS Noise creates the working directories for storing the characterization results.
2. Compiling user .lib file	Make CCS Noise compiles the input library, using Library Compiler.
3. Extracting attributes from user .lib file	Make CCS Noise extracts timing-sense information from arcs in the .lib file.
4. Parsing user .lib file	Make CCS Noise reads the input library and extracts the attributes of the library cells.
5. Generating Verilog file	Make CCS Noise generates a master Verilog file containing instances of all cells in the library.
6. Pre-driver characterization	Make CCS Noise generates the input noise glitch waveform.
7. Generate SPICE Decks	Make CCS Noise generates the SPICE decks that will be used to simulate the cells.
8. Characterizing cells	Make CCS Noise invokes HSPICE to simulate the behavior of cells under varying input slews and input noise bumps.
9. Add noise data to library	Make CCS Noise generates time-dependent current-source models to describe the noise response at the cell outputs and adds the new models to the library description.
10. Write noise .lib file	Make CCS Noise writes out the new .lib library file containing CCS Noise models.
11. Compiling noise .lib file	Make CCS Noise invokes Library Compiler to compile the new .lib library into a new .db library.
12. Cleanup working environment	Make CCS Noise deletes the working directory containing intermediate data files used for characterization.

Requirements for Characterization

The following requirements apply to noise characterization performed by Make CCS Noise.

- Each cell in the .lib library must have at least one timing arc to each output port and bidirectional port.
- There must be HSPICE netlists and models for the cells.
- Each flat SPICE netlist must have fewer than 200,000 transistors. (This limit does not apply to hierarchical netlists.)
- Make CCS Noise requires a Perl interpreter, 64-bit compiled version 5.8.3 or later.
- The following conditions apply to transistor netlists:
  - Two layers of transistors around boundary must be logic that maps to standard cells. Noise modeling requires clear identification of channel-connected regions at the block boundary.
  - The following types of structures are generally supported: static combinational cells, three-state buffers, buffered latches, flip-flops, and clock-gating cells.
  - The following types of structures are generally not supported: analog outputs, voltage-controlled oscillator inputs, phase-locked loops if the charge pump is exposed, blocks with dynamic body bias, and feedthroughs.

The following requirements apply to noise characterization of memory devices.

- Two common three-state diver structures are supported.
- Feedthrough and gated feedthrough are not supported. These features are not common for standalone memory blocks.
- I/O must be buffered.
- Analog I/O is not supported.
- Rambus memory is not supported due to the timing loop that synchronizes operation.

Troubleshooting

The following sections explain how to handle issues involving installation, characterization failure, and validation or correlation with the Library Quality Assurance System utility.

Installation Issues

If Make CCS Noise fails to run properly when you attempt to start it, consider the following installation issues:

- SYNOPSYS_NCX_ROOT environment variable – You must set the SYNOPSYS_NCX_ROOT environment variable to *$synopsys*, where *$synopsys* is the directory where Liberty NCX was installed, as specified in the Synopsys installer script, and *platform* is the name of the platform on which make_ccs_noise is being invoked.
- $path environment variable – You must set the $path environment variable to the appropriate platform, as shown:

  ```
  % set path = ($synopsys/ccsn/platform $path)
  ```

  where *platform* is the name of the platform on which make_ccs_noise is being invoked. The supported platforms are linux, sparcOS5, sparc64, suse32, suse64, or amd64.
- Perl environment – The Perl version must be 5.8.3 or later. You can find out the version number by using the perl -v command. You can get more information about Perl and download the latest version from the Web at http://www.perl.org.

  Note:
  You no longer need to set the Perl path. Liberty NCX infers the path automatically.
- Licensing – Verify that you have a valid and current license for Library Compiler, HSPICE, and either PrimeTime SI or Liberty NCX. When you start Make CCS Noise, it opens PrimeTime SI to test for the existence of a license. If no such license is found, it checks out a Liberty NCX license.
- HSPICE environment – Check for correct output. HSPICE must create “bht.mt0” output files. Standard output (stdout) must be directed to Make CCS Noise if an HSPICE wrapper is used. For example,

  ```
  lis=`echo $1 | sed 's/\.sp/.lis/'`
  hspice $1 -o ; cat $lis

        OR

  hspice <spice_netlist>  >& dc.lis ; cat dc.lis
  ```

- LSF or GRD – Confirm that batch jobs can run. Confirm that the server can start on a simple test case. Check for job timeouts, controlled by user-defined settings. Locate running jobs by using the `bjobs -u username` command.

Characterization Failures

If the characterization process does not run successfully to completion, consider the following troubleshooting steps.

- Make sure that the `SYNOPSYS_NCX_ROOT` environment variable is set to the correct location of the `make_ccs_noise` executable. For more information, see "Required Licenses and Setup" on page 1-4.
- Make sure that the `$path` environment variable is set correctly to find the `make_ccs_noise` executable.
- Check for the availability of a license for Library Compiler, HSPICE, and either PrimeTime SI or Liberty NCX.

  If you do not have a Library Compiler license, the compiled .db library will be incomplete. Check the log file in the running directory for errors, such as the following:

```
...
Using installation in : /INSTALL/ccsn/amd64/
 1) Initialising environment                  ...Succeeded
 2) Compiling user .lib file                  ...Succeeded
 3) Extracting attributes from user .lib file ...
Fatal: LIBRARY_COMPILER is not enabled. (DCSH-1)

Failed
```

- Check the log file to ensure proper setup and submission to the batch queuing system. Failing simulations are identified in the log file. Follow the log file information to find the location of the simulation that caused the problem.
- View the HSPICE simulation log files to identify the simulation problems. Correct the netlists and resimulate until the problem is resolved.
- If the environment is stable, verify that the HSPICE simulations ran to completion as expected. Correct any setup issues and rerun the failing cells using the most recently generated CCS noise library as input.
- If you have trouble characterizing I/O pad cells, check the power rail definitions in the cell. All power supplies and ground connections must be properly defined for the cell. The rail names must match in the input .lib file and the SPICE netlist, or the `set power_rail` and `set ground_rail` commands must be used in the configuration file to identify the rail names.

Figure 1-3 shows the hierarchy of content that `make_ccs_noise` creates on disk during a characterization run. If a particular acquisition fails, the failure is noted in the log file that `make_ccs_noise` writes into the working directory. To debug a failure, find the failed simulation netlist inside the directory structure.

Figure 1-3 Simulation File Directory Structure

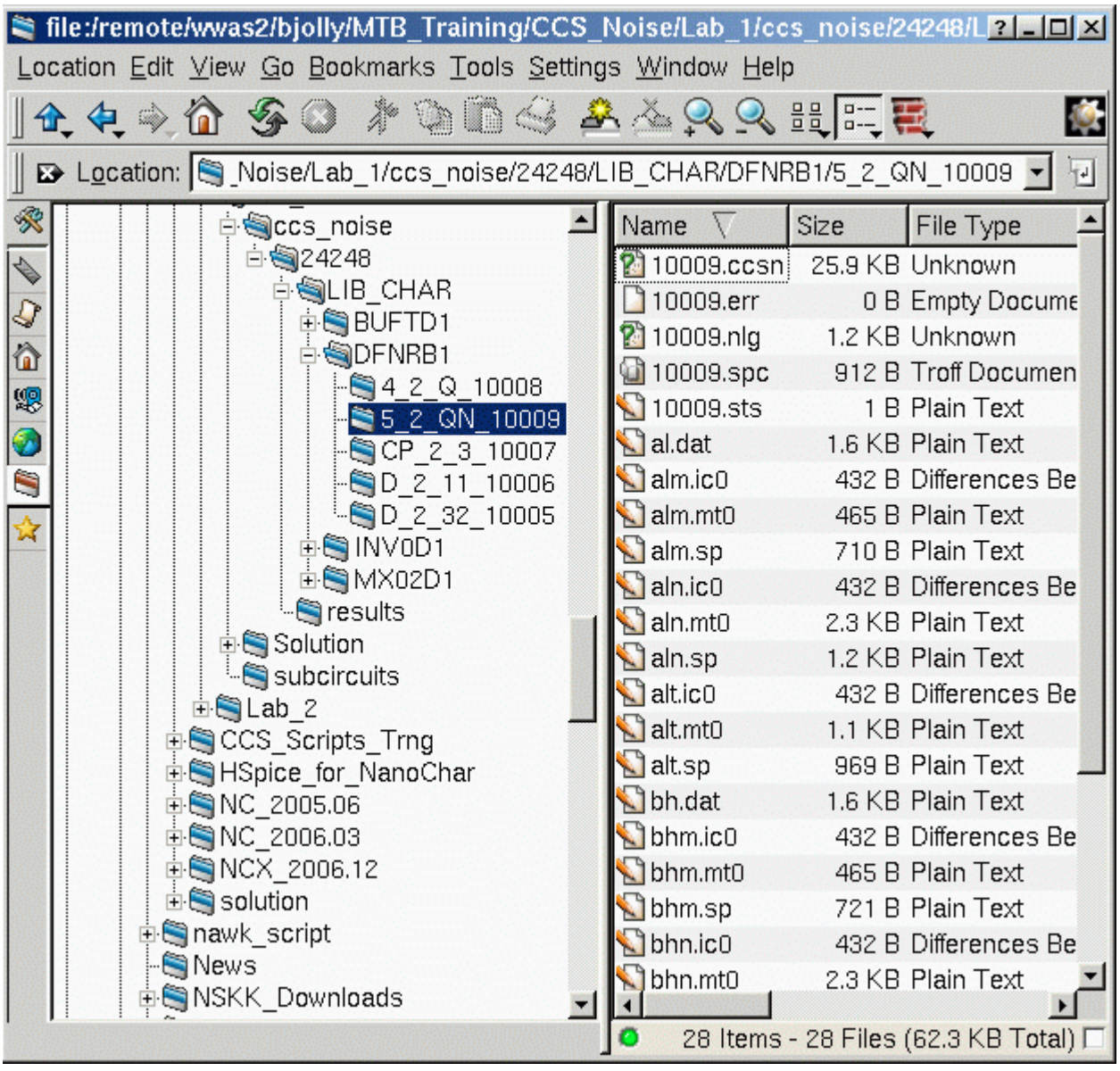


Figure 1-3 shows an example directory in which cell DRNRB1 had a failure on timing arc 5_2_QN. This directory has the data files from seven simulations. Each of the *.sp netlists in the directory was run by HSPICE to generate CCS noise data. The results of these simulations are required for generating the complete noise model.

To debug the problem, you can view the output measurement (*.mt0) files from each simulation. If any of these files are missing or indicate measurement failures, then the corresponding netlist simulation was not successful. Such failures can often be resolved by modifying the HSPICE simulation options or circuit initial conditions.

Library Validation and Correlation

The Library Quality Assurance System is a Library Compiler utility that checks a technology library in Liberty (.lib) format for the completeness, consistency, and accuracy of its cell models, especially composite current source (CCS) models.

The utility is typically used to verify the integrity of new or changed library cell models that have been created or modified by characterization tools such as Make CCS Noise. The utility can perform library validation and CCS noise correlation. You invoke the utility from the Library Compiler shell prompt with the `check_ccs_lib` command.

Library validation tests are performed by the Library Compiler `read_lib` command, which verifies the accuracy and completeness of CCS noise information in the .db library file.

For more information about using the Library Quality Assurance System utility, see the *Library Quality Assurance System User Guide*. The “CCS Noise Validation and Correlation,” section describes the CCS noise validation tests performed by the utility.

Index

E

G

H

I

J

L

M

N

O

P

R

S

T

W