cadence®

Simulating Your Design

Product Version 14.2
December 2014

Printed in the United States of America.

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

Contents

1

Simulating Your Design with ncsim

After you have compiled and elaborated your design, you can invoke the simulator, *ncsim*.

See *Simulation Command-Line Options* for a description of the `ncsim` command syntax, and for a description of all `ncsim` command-line options.

This manual discusses topics related to simulation with *ncsim*:

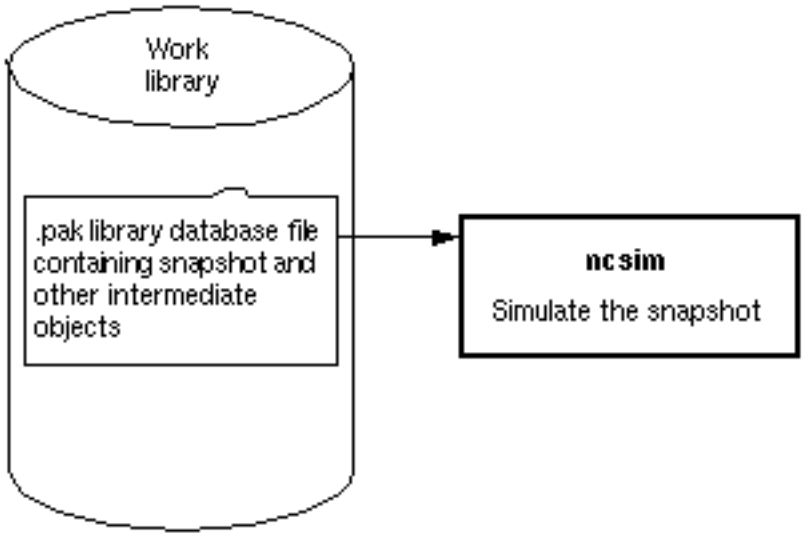
- Overview
- hdl.var Variables
- Invoking the Simulator
- Including User Defined Plus Options on the Command Line
- Starting a Simulation
- Saving, Restarting, Resetting, and Reinvoking a Simulation
- Updating Design Changes When You Invoke the Simulator
- Providing Interactive Commands from a File
- Exiting the Simulation

Overview

ncsim loads the snapshot as its primary input. It then loads other intermediate objects referenced by the snapshot. In the case of interactive debugging, HDL source files and script files also may be loaded. Other data files may be loaded as demanded by the model being simulated (via `$read*` tasks or Textio).

The outputs of simulation are controlled by the model or debugger. These outputs can include result files generated by the model, Simulation History Manager (SHM) databases, Value Change Dump (VCD) files, and so on.

The following figure illustrates the *ncsim* process flow.



Invoke *ncsim* with options and a snapshot name specified in Lib.Cell:View notation. The options and the snapshot argument can occur in any order except that parameters to options must immediately follow the option they modify. Only one snapshot can be specified.

The syntax for invoking the simulator is:

```
% ncsim [options] [Lib.]Cell[:View]
```

- You must specify the cell.
- If a snapshot with the same name exists in more than one library, the easiest (and recommended) thing to do is to specify the library on the command line.
- If there are multiple views that contain snapshots, the easiest (and recommended) thing to do is to specify the view on the command line.

See "Elaborating the Design" for information on how *ncelab* names snapshots.

Rules for Resolving the Snapshot Reference

If you do not specify a library or a view, *ncsim* uses the following rules to resolve the snapshot reference on the command line (assuming that the command line is `% ncsim top`):

1. Is the `WORK` variable set in the `hdl.var` file?

```
YES => Does WORK.top exist?
    YES => How many views of WORK.top have snapshots?
        1 => Simulate this snapshot.
        More than 1 => Error message
          (More than one snapshot matches "top")
    NO => Go to Step 2.
NO => Go to Step 2.
```

2. Search all libraries in the `cds.lib` file.

```
Does LIB*.top exist?
    YES => How many views of LIB*.top have snapshots?
        1 => Simulate this snapshot.
        More than 1 => Error message
          (More than one snapshot matches "top".)
    NO => Error message
          (Snapshot "top" does not exist in the libraries.)
```

hdl.var Variables

The following variables are used by *ncsim*:

- `NCSIMOPTS`

  This variable lets you specify `ncsim` command-line options. For example:

  ```
  DEFINE NCSIMOPTS -messages -errormax 10
  ```

  A snapshot name can also be included.

  The command-line options that you specify with the `NCSIMOPTS` variable are appended to the `ncsim` command. For example, if you have defined the `NCSIMOPTS` variable as shown above, and then enter the following command:

  ```
  % ncsim -update worklib.top
  ```

  the actual command line is as follows:

  ```
  % ncsim -update worklib.top -messages -errormax 10
  ```

  If an option is specified in the `hdl.var` file and is also included on the command line, the option specified in the `hdl.var` file overrides the option entered directly on the

command line because the last option on the command line is used. For example, suppose that the `hdl.var` file includes the following:

```
DEFINE NCSIMOPTS -errormax 10
```

You then enter the following `ncsim` command:

```
% ncsim -errormax 5 worklib.top
```

The `-errormax 10` option in the `hdl.var` file is appended to this command line, and this overrides the `-errormax 5` option.

```
% ncsim -errormax 5 worklib.top -errormax 10
```

Note: Not all `ncsim` command-line options can be included in the definition of the `NCSIMOPTS` variable. The description of the option in this chapter notes any such restrictions.

- `WORK`

  Specifies the default library in which to look for the snapshot. If the snapshot is not found in this library, the rest of the libraries in the `cds.lib` file are searched.

See "The hdl.var File" for more information on the `hdl.var` file.

Invoking the Simulator

You can invoke the simulator (*ncsim*) in two modes:

- Noninteractive mode

  Automatically starts the simulation or the processing of commands from `-input` options without prompting you for command input. See "Invoking the Simulator in Noninteractive Mode" on page 9.

- Interactive mode

  Stops the simulation at time 0 and returns the *ncsim>* prompt. See "Invoking the Simulator in Interactive Mode" on page 10.

In either mode, you can invoke the simulator with or without the SimVision analysis environment.

The syntax for invoking the simulator is:

```
% ncsim [options] snapshot_name
```

The options and the snapshot argument can occur in any order except that parameters to options must immediately follow the option they modify.

The *snapshot_name* argument is a Lib.Cell:View specification. You must specify the cell.

- If a snapshot with the same name exists in more than one library, the easiest (and recommended) thing to do is to specify the library on the command line.
- If there are multiple views that contain snapshots, the easiest (and recommended) thing to do is to specify the view on the command line.

If you do not specify a library or a view, *ncsim* uses a set of rules to resolve the snapshot reference on the command line. See "Rules for Resolving the Snapshot Reference" on page 7.

Only one snapshot can be specified.

Invoking the Simulator in Noninteractive Mode

Invoking *ncsim* in noninteractive mode automatically starts the simulation or processing of commands from `-input` options without prompting you for input.

- To invoke *ncsim* in noninteractive mode without the SimVision analysis environment, type:

  ```
  % ncsim [-run] [other_options] snapshot_name
  ```

  Examples:

  ```
  % ncsim worklib.top:module
  % ncsim -run worklib.top:module
  ```

  The `-run` option is not required.

- To invoke *ncsim* in noninteractive mode with the SimVision analysis environment, type:

  ```
  % ncsim -gui -run [other_options] snapshot_name
  ```

  Example:

  ```
  % ncsim -gui -run worklib.top:module
  ```

  The `-run` option is required.

If you have included the `-tcl` option in your `NCSIMOPTS` variable in the `hdl.var` file because you invoke the simulator in interactive mode most of the time, use `-batch` to override `-tcl` and simulate in noninteractive mode.

Example:

```
% ncsim -batch worklib.top:module
```

If you want to run in noninteractive mode, but do not want the simulator to exit at the end of the simulation, use both the `-run` and `-tcl` options. The following combination of options runs *ncsim* in noninteractive mode, but returns the *ncsim>* prompt instead of exiting:

```
% ncsim -run -tcl snapshot_name
```

Include the `-exit` option on the command line if you want the simulator to exit under conditions that would normally stop the simulation and put it in interactive mode.

The following examples illustrate the various options for invoking the simulator in noninteractive mode. Other `ncsim` command options are not shown.

`% ncsim [-run] top`	Invokes *ncsim* and runs the simulation.
`% ncsim -gui -run top`	Invokes *ncsim* with the SimVision analysis environment and runs the simulation.
`% ncsim -run -tcl top`	Invokes *ncsim* and runs the simulation. When the simulation is completed or interrupted, returns the *ncsim>* prompt instead of exiting.
`% ncsim -batch top`	Use `-batch` if you want to simulate in noninteractive mode, but have the `-tcl` option included with the `NCSIMOPTS` variable in the `hdl.var` file.

Invoking the Simulator in Interactive Mode

You can interact with your design throughout a simulation by instructing *ncsim* to stop at the beginning of the simulation so that you can enter interactive mode at simulation time 0.

If you are using the command-line interface, use the `-tcl` option when you invoke the simulator.

```
% ncsim -tcl [other_options] snapshot_name
```

Example:

```
% ncsim -tcl worklib.top
```

If you are using the SimVision analysis environment, *ncsim* automatically stops at the beginning of the simulation.

```
% ncsim -gui [-tcl] [other_options] snapshot_name
```

Examples:

```
% ncsim -gui worklib.top
```

```
% ncsim -gui -tcl worklib.top
```

Note: The `-tcl` option is not required.

Including User Defined Plus Options on the Command Line

Use the `$test$plusargs` system function in your Verilog code to check for the presence of a plus run-time option. For example:

```
initial
  if ($test$plusargs("dumpon"))
  begin
    $dumpfile("results.vcd");
    $dumpvars;
  end
```

In this example, you can enable dumping by including the `+dumpon` option on the `ncsim` command line. No recompilation is necessary.

```
% ncsim +dumpon snapshot_name
```

The following example illustrates how to use `$test$plusargs` to control which stimulus file is used for a particular simulation.

```
initial
  if ($test$plusargs("test01"))
  $readmemh("test01.dat", stim_mem);

  if ($test$plusargs("test02"))
  $readmemh("test02.dat", stim_mem);

  if ($test$plusargs("test03"))
  $readmemh("test03.dat", stim_mem);

  if ($test$plusargs("test04"))
  $readmemh("test04.dat", stim_mem);
```

By using the `$test$plusargs` system function, you can change the stimulus without rebuilding the design. This method is also an easy way to run multiple simulations in parallel. For example:

```
% ncsim snapshot_name +test01 &
% ncsim snapshot_name +test02 &
% ncsim snapshot_name +test03 &
% ncsim snapshot_name +test04 &
```

Support for Compressed Files

Compressed files save storage space when managing larger files. Both *ncsim* and *irun* allow you to use compressed files with user defined pus options.

The simulator will recognize and extract the following archive formats:

- Gnu zip compression (`.gz`)
- Standard compression (`.z`)

For example:

```
% ncsim snapshot_name +TESTFILE=VER.FULLSCAN.test.verilog.gz
```

Multiple compressed files are supported. If multiple compressed files are processed together, then *ncsim* will extract each file into a `/tmp` directory. Once every file is decompressed and the simulation is complete, the `/tmp` directory is deleted.

Starting a Simulation

To start or resume a simulation:

- If you are using the Tcl command-line interface, use the `run` command. This command has several options that let you control when the simulation is to stop:
  - `-delta`—Run to the beginning of the next delta cycle or to a specified delta cycle.
  - `-next`—Run one behavioral statement, stepping over any subprogram calls.
  - `-return`—Run until the current subprogram (task, function, procedure) returns.
  - `-step`—Run one behavioral statement, stepping into subprogram calls.
  - `-timepoint`—Run for a specified number of time units.
  - `-phase`—Run to the beginning of the next phase of the simulation cycle. The two phases of a simulation cycle are signal evaluation and process execution.
  - `-process`—Run until the beginning of the next scheduled process or to the beginning of the next delta cycle, whichever comes first. In VHDL, a process is a

process statement. In Verilog, it is an `always` block, `initial` block, or one of several kinds of anonymous behavior that can be scheduled to run.

See `run` for details on the `run` command and for examples.

- If you are using the SimVision analysis environment, use the commands on the *Simulation* menu. To simulate for a specified number of time units, set a time breakpoint before starting the simulation. See Setting a Time Breakpoint.

  You can also control the simulation by using the simulation buttons on the simulation toolbar.

  See Controlling the Simulation for details on controlling the simulation using SimVision.

Saving, Restarting, Resetting, and Reinvoking a Simulation

This section tells you how to:

- Save the current state of a simulation in a snapshot.
- Restart a simulation with a saved snapshot.
- Reset a simulation to its original state at time 0.
- Reinvoke a simulation.

Saving and Restarting the Simulation

You can save and restart the simulation state at any time. Creating simulation checkpoints is especially useful for large simulations where you might want to save the simulation state at regular intervals. Another common use is to save the simulation state after the circuit has been initialized so that future simulations can begin at that point rather than from time 0.

When you save the simulation state, the simulator creates a new snapshot. To restart the simulation at a later time, you must load the saved snapshot.

The current simulation state that is saved in the snapshot includes the simulation time and all object values, scheduled events, annotated delays, the contents of the memory allocated for access type values, and file pointers. It does not include aspects of the debugging environment such as breakpoints, probes, Tcl variables, and GUI configuration. PLI/VPI callbacks and handles are saved under certain circumstances. Please refer to the PLI/VPI manuals for details.

You cannot save a snapshot if the simulator is in the process of executing sequential HDL code. If the simulation is in a state that cannot be saved, you must use the `run -clean` command to run the simulation until the currently running sequential behavior (if any) suspends itself at a delay or event control or a VHDL `wait` statement.

- If you are using the Tcl command-line interface, use the `save` command to save the simulation state and the `restart` command to load a saved snapshot.

  See `save` and `restart` for details on these commands. The documentation for the `save` command includes an example of saving and restarting.

- If you are using the SimVision analysis environment, select *Simulation – Save Checkpoint* and fill in the Save Checkpoint form to save the simulation state.

  To restart, select *Simulation – Restart from Checkpoint* and fill in the Restart from Checkpoint form.

  See Saving and Restarting a Simulation for more information on saving and restarting a simulation.

When you restart with a saved snapshot in the same simulation session:

- SHM databases remain open and all probes remain set.
- Breakpoints set at the time that you execute the restart remain set.

  Note: If you set a breakpoint that triggers, for example, every 10 ns (that is, at time 10, 20, 30, and so on) and restart with a snapshot saved at time 15, the breakpoint triggers at 20, 30, and so on, not at time 25, 35, and so on.

- Forces and deposits in effect at the time you issue a `save` command are still in effect when you restart.

If you exit the simulation and then invoke the simulator with a saved snapshot, databases are closed. Any probes and breakpoints are deleted. If you want to restore the full Tcl debug environment when you restart, make sure that you save the environment with the `save -environment` command. This command creates a Tcl script that captures the current breakpoints, databases, probes, aliases, and predefined Tcl variable values. You can then use the Tcl `source` command after restarting or the `-input` option when you invoke the simulator to execute the script.

For example:

```
% ncsim worklib.top
ncsim> (Open a database, set probes, set breakpoints, deposits, forces, etc.)
ncsim> run 100 ns
ncsim> save worklib.top:ckpt1
ncsim> save -environment ckpt1.tcl
```

```
ncsim> exit
% ncsim -tcl worklib.top:ckpt1
ncsim> source ckpt1.tcl
```

Some operating systems may impose a limit on the size of a file. If a library database exceeds this limit, you will not be able to add objects to the database. If you save many snapshot checkpoints to unique views in a single library, this file size limit could be exceeded. If you reach this limit, you can:

- Use `save -overwrite` to overwrite an existing snapshot. For example,

  ```
  ncsim> save -simulation -overwrite snap1
  ```

- Save snapshots to a separate library. For example,

  ```
  % mkdir INCA_libs/snaplib
  % ncsim -f ncsim.args
  ncsim> run 1000 ns
  ncsim> save -simulation snaplib.snap1
  ncsim> run 1000 ns
  ncsim> save -simulation snaplib.snap2
  ```

  Note: In addition to creating a directory for the library, the library must be defined in the `cds.lib` file.

- Remove snapshots using the *ncrm* utility. For example,

  ```
  % ncrm -snapshot worklib.snap1
  ```

Resetting the Simulation

You can reset the currently loaded model to its original state at time zero.

- If you are using the Tcl command-line interface, use the `reset` command.

  See `reset` for details on using the `reset` command.

  The documentation for the `save` command includes an example of resetting the simulation.

- If you are using the SimVision analysis environment, select *Simulation – Reset to Start*. The time-zero snapshot, created by the elaborator, must still be available.

  You can also click the *Reset* button on the simulation toolbar.

  See Resetting the Simulation for more information on resetting the simulation.

When you reset the simulation to its state at time 0, the debug environment remains the same.

- Tcl variables remain as they were before the reset.
- SHM and VCD databases remain open, and probes remain set.

  Note: VCD databases created with the `$dumpvars` call in Verilog source code are closed when you reset.
- Breakpoints remain set.
- The SimVision waveform viewer window remain the same.

Forces and deposits in effect at the time you issue the `reset` command are removed.

If you exit the simulation instead of resetting, databases are closed, probes and breakpoints are deleted, and Tcl variables are reset to their default values.

Reinvoking a Simulation

If you are using the SimVision analysis environment, you can reinvoke the simulation session at any time, even after the simulation has finished. When you reinvoke, an `exit` command is issued and then *ncsim* is invoked with the `-update` option, which recompiles any changed design units, re-elaborates the design, generates a new snapshot, and loads the updated snapshot.

To reinvoke, select *Simulation – Reinvoke Simulator*.

If you have set the *Prompt before reinvoke* option on the Preferences form, SimVision displays the Reinvoke form, which lists the command-line arguments you used when you invoked *ncsim* originally. If you want to change the *ncsim* arguments, edit the text field. Click *Yes* to reinvoke the simulation. Click *No* to cancel the reinvoke and remain in the current simulation session.

If the *Prompt before reinvoke* option on the Preferences form is not set, the Reinvoke form does not appear, and *ncsim* is invoked with the same original command-line arguments.

Reinvoke works the same way if you are running the simulator with *irun*. If the *Prompt before reinvoke* option on the Preferences form is not set, the Reinvoke form does not appear, and *ncsim* is invoked with the same original *irun* command-line arguments. If you edit the text field to add an option that affects elaboration (`-access`, for example), the design is re-elaborated and a new snapshot is created.

Note: Reinvoke is a SimVision option. You can reinvoke the simulation inside the graphical environment as many times as you want, but if you edit the text field on the Reinvoke form to remove the `-gui` option, it is not possible to reinvoke from the Tcl *ncsim>* prompt.

When you reinvoke, your current setup is automatically saved and restored. This includes:

- SHM databases
- Probes
- Breakpoints
- All signals that were displayed in the SimVision waveform viewer

Forces and deposits in effect at the time you reinvoke are removed.

See Reinvoking the Simulation for details on reinvoking a simulation.

Updating Design Changes When You Invoke the Simulator

When you change design units in the hierarchy, you must recompile them and re-elaborate the design hierarchy.

If you want to update and simulate, use the `-update` option on the `ncsim` command to automatically recompile and re-elaborate all out-of-date design units. This option calls *ncupdate* to recompile any changed design units, re-elaborate the design, and generate a new snapshot. It then invokes the simulator and loads the new snapshot.

If you only want to update the snapshot, run the *ncupdate* utility. See "ncupdate" for details on *ncupdate*.

The purpose of *ncupdate* is to provide quick design change turnaround when you have edited a design unit. The modifications to design units cannot cross file boundaries to modify other files. Do not use *ncupdate* (or `ncsim -update`) after adding a design unit, a source file, or compiler directives to the design. For example, *ncupdate* will not update correctly if you edit a source file to define a new macro, or if you change a design unit in a way that introduces a new cross-file dependency. In these cases, recompile the design with `ncvlog` or `ncvhdl -update`.

Use the `-nosource` option with `-update` if you recompile selected parts of your design and then want to automatically re-elaborate and load the new snapshot into the simulator. For example, suppose you have edited two source files, `first.v` and `second.v`, and you only want to include the changes in `second.v`. Recompile the file `second.v` and then use `ncsim -update -nosource`.

```
% ncsim -update -nosource snapshot_name
```

You can also use `-nosource` when you have changed one design unit in a file with more than one design unit. You can recompile only the unit you have changed and then use `ncsim -update -nosource` to re-elaborate the design and invoke the simulator.

1. Recompile the changed design unit.

   ```
   % ncupdate -unit [lib.]cell[:view]
   ```

   or

   ```
   % ncvlog -unit [lib.]cell[:view]
   ```

2. Use `ncsim -update -nosource`

   ```
   % ncsim -update -nosource snapshot_name
   ```

If you make a change to any SDF-related file (the SDF source file, the compiled SDF file, the SDF configuration file, or the SDF command file), and then execute an `ncsim -update` command, the elaborator automatically re-annotates the design using the new, up-to-date files. SDF source files that have changed are automatically recompiled. See "SDF Timing Annotation" for details on SDF annotation.

The following example shows how to use `ncsim -update` to automatically recompile, re-elaborate, and load a new snapshot.

```
;# After editing a design unit, use -update to automatically recompile,
;# re-elaborate, and reinvoke the simulator.
;# The -update option calls ncupdate to recompile the file counter.v, which
;# contains the changed module, m16.
% ncsim -nocopyright -messages -update -tcl board
Updating snapshot worklib.board:module (SSS), reason: file `./counter.v' is newer
than expected.
       expected: Mon May  3 15:20:43 1999
       actual: Tue Jun  1 08:52:57 1999
Updating: module worklib.m16:module (VST)
file: ./counter.v
    module worklib.m16:module
        errors: 0, warnings: 0
;# ncupdate re-elaborates the design hierarchy.
Updating: snapshot worklib.board:module (SSS)
Update of snapshot worklib.board:behav (SSS) successful.
;# The -update option automatically reinvokes the simulator.
Loading snapshot worklib.board:module ..................... Done
ncsim>
```

The following example shows how to recompile one module in a file containing multiple modules before using `-update`.

```
;# The source file 16bit_alu.v contains four modules. After editing module
;# logic, use ncupdate-unit to update only that module.
% ncupdate -nocopyright -messages -unit worklib.logic:module
Updating: module worklib.logic:module (VST)
file: ./16bit_alu.v
    module worklib.logic:module
        errors: 0, warnings: 0
;# Then use ncsim -update -nosource to re-elaborate, generate a new snapshot,
;# and reinvoke the simulator.
% ncsim -nocopyright -messages -tcl -update -nosource alu_16
Updating snapshot worklib.alu_16:module (SSS), reason: dependent module
worklib.logic:module (VST) is newer than expected.
        expected: Tue Oct  8 11:19:04 1996
        actual:   Tue Oct  8 11:20:31 1996
Updating: snapshot worklib.alu_16:module (SSS)
Update of snapshot worklib.alu_16:module (SSS) successful.
Loading snapshot worklib.alu_16:module ..................... Done
ncsim: *W,VSCNEW: file './16bit_alu.v' is newer than expected by module
worklib.alu_16:module (VST).
ncsim>
```

Providing Interactive Commands from a File

You can load a file containing simulator commands by specifying an input file. This is useful when you want to load a file of Tcl commands or aliases, or when you want to reproduce an interactive session by re-executing a file of commands saved from a previous simulation run (a key file). When *ncsim* has processed all of the commands in the input file, or if you interrupt processing, input reverts back to the terminal.

There are three ways to execute the commands in an input file:

- Specify the input file with the `-input` option when you invoke the simulator. When you use the `-input` option, commands contained in the input file are executed at the beginning of the simulation session.
- Specify the input file with the Tcl `input` command.
- Execute the Tcl `source` command.

The behavior of the `input` command and the `-input` option is different from the behavior of the `source` command in the following ways:

- With the `source` command, execution of the commands in the script stops if a command generates an error. With the `input` command or with the `-input` option, the contents

of the file are read in place of standard input at the next Tcl prompt, as if you had typed the commands at the command-line prompt. This means that errors do not stop the execution of commands in the script.

- The `input` command and the `-input` option echo commands to the screen as they are executed, along with any command output or error messages.The `source` command, on the other hand, displays the output of only the last command in the file. Output from the model (for example, the output of `$display`, `$monitor`, or `$strobe` tasks, or the output of stop points) is printed to the screen.

See "-input Command Syntax" on page 20 for more information on the `-input` command-line option.

If you are using the SimVision analysis environment:

1. Select *File – Source Command Script*.

   This opens the Source Command Script form.

2. Enter the name of the file that contains your Tcl commands in the *Filename* field.
3. Set the *Send commands to* field to *simulator Console (NC-Sim)*.
4. Click the *OK* button.

See Using Command Scripts for details on sourcing a file of SimVision commands.

-input Command Syntax

The `-input` option allows you to specify a file name or a simulator command as an argument.

Use the `-input` option with a file name to specify a file of commands.

Example:

```
% ncsim -input setup.inp worklib.top:module
```

To specify a simulator command, use the @ symbol before the command, enclosing everything in quotation marks if the command takes an argument, modifier, or option.

Syntax:

```
% ncsim -input @command snapshot_name
```

Examples:

```
% ncsim -input @run worklib.top:module
```

```
% ncsim -input "@stop -line 22" worklib.top:module
```

You can include more than one `-input` option of either form on the command line. The input files (or commands) are processed in the order in which they appear on the command line.

The following example shows you how to use the `-input` option when you invoke the simulator.

```
;# Create the input file using a text editor. Key files can also be used as
;# input files.
;* Command input file: set_break.inp
stop -create -line 27
run
value data
run 50
value data
run 50
value data
run
;# Run ncsim with the -input option to specify the input file. The simulator
;# executes the commands in the file.
% ncsim -nocopyright -messages -input set_break.inp hardrive
Loading snapshot worklib.hardrive:module ..................... Done
ncsim> stop -create -line 27
Created stop 1
ncsim> run
0 FS + 0 (stop 1: ./hardrive.v:27)
./hardrive.v:27    repeat (2)
ncsim> value data
4'hx
ncsim> run 50
Ran until 50 NS + 0
ncsim> value data
4'h0
ncsim> run 50
at time 50 clr =1 data= 0 q= x
Ran until 100 NS + 0
ncsim> value data
4'h0
ncsim> run
at time 150 clr =1 data= 1 q= 0
at time 250 clr =1 data= 2 q= 1
...
```

```
...
...
at time 3250 clr =0 data=15 q= 0
at time 3350 clr =0 data=15 q= 0
Simulation complete via $finish(1) at time 3400 NS + 0
;# Control reverts back to the terminal after ncsim has executed all commands
;# in the file.
ncsim> exit
%
```

Exiting the Simulation

To exit the simulator:

- If you are using the Tcl command-line interface, use either the `exit` command or the `finish` command.

  The `exit` command is a built-in Tcl command. It halts execution and returns control to the operating system. See `exit` for details.

  The `finish` command also halts execution and returns control to the operating system. This command takes an optional argument that determines what type of information is displayed after exiting.

  - 0—Prints nothing (same as executing `finish` without an argument).
  - 1—Prints the simulation time.
  - 2—Prints simulation time and statistics on memory and CPU usage.

  See `finish` for details.

- If you are using the SimVision analysis environment, there are two ways that you can stop simulating with SimVision:

  - Disconnect from the simulation. This leaves the simulation running but makes it unavailable from the SimVision user interface.

    To disconnect from a simulation, click the *Disconnect* button in the simulator tab of the Console window.

    You can reconnect to the simulation by choosing *File – Open Simulation* and selecting the simulation from the Connect to Simulation Session form.

- Terminate the simulation. This stops the simulator. You cannot reconnect to the simulation after you have terminated it.

  To terminate a simulation, click the *Terminate* button in the simulator tab of the Console window.

You can also select *File – Exit SimVision*.

See Disconnecting and Terminating a Simulation for more information.

Index

Symbols

C

D

F

H

I

M

N

P

R

S

U