# Contents

## 1

**Introduction** .......................................................... 5  
The Drink Machine Example ............................................... 7  
For More Information .................................................... 9  

## 2

**Getting Started** ....................................................... 11  
Copying Tutorial Data .................................................. 11  
Starting NCLaunch ......................................................... 12  
Compiling the Design Source Files .................................... 14  
Starting the Simulator .................................................. 18  
Exiting from NCLaunch .................................................. 20  
For More Information .................................................... 21  

## 3

**Simulating the Design** ............................................... 23  
Selecting the Simulation Data to Save ................................ 23  
For More Information .................................................... 27  

## 4

**Working with Simulation Data as Waveforms** ....................... 29  
Selecting the Signals to Display ...................................... 29  
Displaying Data in the Waveform Window ............................. 31  
Controlling the Appearance of Waveform Data ....................... 33  
For More Information .................................................... 36  

## 5

**Debugging a Design** .................................................. 39  
Searching for Conditions in the Waveform ......................... 39
# NC-Verilog Simulator Tutorial

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzing the Waveforms to Find an Error</td>
<td>41</td>
</tr>
<tr>
<td>Creating Custom Views of Simulation Data</td>
<td>42</td>
</tr>
<tr>
<td>Fixing an Error in the Source Code</td>
<td>44</td>
</tr>
<tr>
<td>Ending a SimVision Session</td>
<td>47</td>
</tr>
<tr>
<td>For More Information</td>
<td>48</td>
</tr>
</tbody>
</table>
Introduction

This tutorial introduces you to the Cadence NC-Verilog simulator and SimVision.

The example used in the tutorial is a design for a drink dispensing machine written in the Verilog hardware description language. Using this example, you will learn how to:

- Compile Verilog source files, elaborate the design, and run the simulation using NCLaunch, a graphical user interface that helps you manage large design projects. NCLaunch helps you configure and launch the compiler, elaborator, and simulator. You can also run other tools from NCLaunch, such as the SDF Compiler, HDL Analysis and Lint, Code Coverage Analyzer, NCBrowse, and Comparescan.

- Debug a problem in the design using the SimVision analysis environment.

SimVision is a unified graphical debugging environment for Cadence simulators. You can use SimVision to debug digital, analog, or mixed-signal designs written in Verilog, VHDL, SystemC, or mixed-language.

You can run SimVision in either of the following modes:

- Simulation mode

  In simulation mode, you view “live” simulation data. That is, you analyze the data while the simulation is running. You can control the simulation by setting breakpoints and stepping through the design.

  SimVision provides several tools to help you track the progress of the simulation:

  - Console Window
  - Source Browser
  - Design Browser
  - Cycle Viewer
  - Schematic Tracer
  - Signal Flow Browser
Waveform Window

Register Window

All of these windows are linked so that when you select an object in one window, it is selected in the other windows as well.

Post-processing environment (PPE) mode

In PPE mode, you analyze simulation data after simulation has completed. You have access to all of the SimVision tools, except for the simulator. As in simulation mode, all of these windows are linked so that, when you select an object in one window, it is selected in the other windows as well.

To run in PPE mode, you must first simulate the design and save the simulation data to a file. You can switch from simulation mode to PPE mode at any time, but you cannot switch from PPE mode to simulation mode.

This tutorial introduces you to some of the major features of the following SimVision tools:

Console window

The Console window lets you enter Tcl simulator commands or SimVision commands.

Design Browser

The Design Browser lets you access the design hierarchy and the signals and variables in the design database.

Register window

The Register window lets you use a free-form graphics editor to define any number of register pages, each containing a custom view of the simulation data.

Signal Flow Browser

The Signal Flow Browser lets you trace the drivers of a signal.

Source Browser

The Source Browser gives you access to the design source code.
The Drink Machine Example

The drink machine design is made up of three modules:

- `drink_machine` counts the amount of change that the user has entered, dispenses a drink, and returns any change that is due.
- `coin_counter` loads the machine with coins and determines when the machine is out of change.
- `can_counter` loads the machine with drinks and determines when the machine is empty.

The example also includes a testbench, which initializes the machine and buys drinks by depositing different combinations of nickels (five cent coins), dimes (ten cent coins), and quarters (twenty-five cent coins).

The behavior for accepting coins and dispensing drinks is modeled as a state machine. The amount of money that the user has deposited so far defines the current state. The type of coin that the user deposits determines the machine’s next state.

For example, when no money has been deposited, the machine is in the `idle` state. When the user adds a nickel, the machine transitions to the next state, `five`. When the current state is `five` and the user adds a quarter, the machine transitions to the next state, `thirty`. When the user has added exactly 50 cents, the machine dispenses a drink and transitions back to the `idle` state. When the user adds more than 50 cents, the machine dispenses a drink, returns the correct change, and transitions back to the `idle` state.
### Table 1-1 Drink Machine State Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Transition Value</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle 4’d0</td>
<td>nickel_in</td>
<td>five 4’d1</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>ten 4’d2</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>twenty_five 4’d5</td>
</tr>
<tr>
<td>five 4’d1</td>
<td>nickel_in</td>
<td>ten 4’d2</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>fifteen 4’d3</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>thirty 4’d6</td>
</tr>
<tr>
<td>ten 4’d2</td>
<td>nickel_in</td>
<td>fifteen 4’d3</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>twenty 4’d4</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>thirty_five 4’d7</td>
</tr>
<tr>
<td>fifteen 4’d3</td>
<td>nickel_in</td>
<td>twenty 4’d4</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>twenty_five 4’d5</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>forty 4’d8</td>
</tr>
<tr>
<td>twenty 4’d4</td>
<td>nickel_in</td>
<td>twenty_five 4’d5</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>thirty 4’d6</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>forty_five 4’d9</td>
</tr>
<tr>
<td>twenty_five 4’d5</td>
<td>nickel_in</td>
<td>thirty 4’d6</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>thirty_five 4’d7</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>fifty 4’d10</td>
</tr>
<tr>
<td>thirty 4’d6</td>
<td>nickel_in</td>
<td>thirty_five 4’d7</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>forty 4’d8</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>nickel_out 4’d11</td>
</tr>
<tr>
<td>thirty_five 4’d7</td>
<td>nickel_in</td>
<td>forty 4’d8</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>forty_five 4’d9</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>dime_out 4’d12</td>
</tr>
<tr>
<td>forty 4’d8</td>
<td>nickel_in</td>
<td>forty_five 4’d9</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>fifty 4’d10</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>nickel_dime_out 4’d14</td>
</tr>
<tr>
<td>forty_five 4’d9</td>
<td>nickel_in</td>
<td>fifty 4’d10</td>
</tr>
<tr>
<td></td>
<td>dime_in</td>
<td>nickel_out 4’d11</td>
</tr>
<tr>
<td></td>
<td>quarter_in</td>
<td>two_dime_out 4’d14</td>
</tr>
<tr>
<td>fifty 4’d10</td>
<td></td>
<td>idle</td>
</tr>
<tr>
<td>nickel_out 4’d11</td>
<td></td>
<td>idle</td>
</tr>
<tr>
<td>dime_out 4’d12</td>
<td></td>
<td>idle</td>
</tr>
<tr>
<td>nickel_dime_out 4’d13</td>
<td></td>
<td>idle</td>
</tr>
<tr>
<td>two_dime_out 4’d14</td>
<td></td>
<td>idle</td>
</tr>
</tbody>
</table>
For More Information

SimVision provides other tools not used in this tutorial.

<table>
<thead>
<tr>
<th>Tool/Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Schematic Tracer Icon" /></td>
<td>The Schematic Tracer displays a Verilog or VHDL design as a schematic diagram and lets you trace a signal through the design. See Chapter 12, “Viewing a Design Schematic,” in the SimVision User Guide.</td>
</tr>
<tr>
<td>Simulation Cycle Debugger</td>
<td>The Simulation Cycle Debugger lets you step through a simulation cycle, stopping at each time point, delta cycle, simulation phase, or scheduled process. It is not available for Verilog-XL or AMS Designer. See Chapter 11, “Debugging at the Delta Cycle Level,” in the SimVision User Guide.</td>
</tr>
</tbody>
</table>
Getting Started

Before you can simulate your design, you must compile and elaborate it. Compiling the design produces an internal representation for each HDL design unit in the source files. Elaborating the design constructs a design hierarchy based on the instantiation and configuration information in the design, establishes signal connectivity, and computes initial values for all objects in the design.

You compile, elaborate, and simulate your design by running the following tools:

- `ncvlog` Compiles the Verilog source files.
- `ncelab` Elaborates the design and generates a simulation snapshot.
- `ncsim` Simulates the snapshot.

You can also run NC-Verilog in single-step invocation mode with the `ncverilog` command.

You can also use a single tool, NCLaunch, a graphical user interface that helps you manage large design projects. NCLaunch helps you configure and launch the simulation tools. You can run the tools in multi-step invocation mode or in single-step invocation mode. It can also give you access to Comparescan, SDF Compiler, HDL Analysis and Lint, Code Coverage Analyzer, NCBrowse, and other simulation tools.

This tutorial shows you how to use NCLaunch in multi-step invocation mode.

Copying Tutorial Data

All of the source files for this design are included in your Cadence installation hierarchy. The source files are in the following location:

```
install_dir/doc/ncvlogtut/examples
```

Create a directory (for example, `tutorial`), and then copy the tutorial files to this directory. For example:

```
mkdir tutorial
```
cd tutorial
cp install_dir/doc/ncvlogtut/examples/* .

Starting NCLaunch

1. Start NCLaunch from the directory into which you have copied the source files for the tutorial.

   nclaunch -new &

   The -new option specifies that you want to work on a new design.

   NCLaunch displays a list of modes in which you can run the tool, as shown in Figure 2-1 on page 13.
Multiple Step mode uses the `ncvlog` and `ncelab` commands to compile and elaborate your design; Single Step mode uses the `ncverilog` command.

2. Click *Multiple Step*.

NCLaunch displays the main window, as shown in Figure 2-2 on page 14.
Figure 2-2  NCLaunch Main Window

The left side of the window shows all of the files in the current directory. The right side will show the design libraries, after you have compiled the source files and elaborated the design. The top of the window contains menus and buttons for starting the tools.

Compiling the Design Source Files

Before you can simulate your design, you must compile the source files using the Verilog compiler and elaborate the design into a snapshot using the elaborator. A snapshot is the representation of your design that the simulator uses. It is stored in the work library along with the other intermediate objects generated by the compiler and elaborator.
The NCLaunch main window gives you access to the tools you need to compile and elaborate the design, as well as access to several utilities. You access the tools and utilities by using the *Tools* or *Utilities* menu, or by clicking the appropriate button on the toolbar. Not all tools and utilities listed in the menus are available on the toolbar, but you can customize the toolbar to add any tools or utilities that you like.

Table 2-1 on page 15 shows the buttons that NCLaunch includes in the toolbar.

### Table 2-1 NCLaunch Toolbar Buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="VHDL" /></td>
<td>VHDL Compiler</td>
</tr>
<tr>
<td><img src="image" alt="Verilog" /></td>
<td>Verilog Compiler</td>
</tr>
<tr>
<td><img src="image" alt="Elaborator" /></td>
<td>Elaborator</td>
</tr>
<tr>
<td><img src="image" alt="Simulator" /></td>
<td>Simulator</td>
</tr>
<tr>
<td><img src="image" alt="NCBrowse" /></td>
<td>NCBrowse</td>
</tr>
<tr>
<td><img src="image" alt="Waveform" /></td>
<td>Waveform window</td>
</tr>
</tbody>
</table>

To compile the design:

1. Select the Verilog files that make up the design:
   
   ```
   can_counter.v
   coin_counter.v
   drink_machine.v
   drink_machine_top.v
   test_drink.v
   ```
To select multiple files, hold down the Control key and click on each filename.

2. Click the Verilog Compiler button.

The I/O area at the bottom of the window displays the `ncvlog` command that runs as a result of your selections, and it displays the messages that NC-Verilog generates as it compiles the design files.

By default, NC-Verilog creates a directory called `INCA_libs` and a subdirectory called `worklib`. All modules in the design are compiled into the `worklib` directory. Notice that the `INCA_libs` directory now appears in the file browser (left side) of the NCLaunch window, and that the design library `worklib` has been added to the library browser (right side) of the window.

**Elaborating the Design**

To elaborate a design, you typically expand the work library (`worklib`), select the top-level design unit, and click on the Elaborator button.

However, for this tutorial, you must set some options before you run the elaborator. Perform the following steps to set the options and elaborate the design:

1. Expand the work library (`worklib`) by clicking on the plus sign next to the hardhat icon.

2. Expand the top-level design unit. In this example, the top-level unit is the Verilog testbench, `test_drink`.

3. Select module.

4. Choose Tools – Elaborator to open the Elaborate form, shown in Figure 2-3 on page 17.
Notice that the *Access Visibility* button is selected and that the value is set to *All*. This option provides full access (read, write, and connectivity access) to simulation objects so that you can probe objects and scopes to a simulation database and debug the design.

**Note:** Access to simulation objects is on by default when you are using NCLaunch. When you are using the command-line interface, access is off by default and you must start the elaborator with the `-access` option. For example:

```
ncelab -access +rwc worklib.test_drink:module
```
5. Only one module in the drink machine contains the `timescale compiler directive. To prevent the elaborator from issuing errors because the other modules do not have a timescale set, enable the Other Options button and enter the following option in the text field, as shown in Figure 2-3 on page 17:

```
-timescale 1ns/1ns
```

6. Click OK to elaborate the design.

The I/O area at the bottom of the window displays the ncelab command that runs as a result of your selections, and it displays the messages that the elaborator generates.

**Tip**

If you receive elaborator error messages, you may have made a mistake when running these steps. For example:

- Did you select the correct design unit name?
- Did you remember to include the -timescale option?

Perform these steps again if you get errors during elaboration.

**Starting the Simulator**

To start the simulator:

1. Expand the Snapshots folder to display the snapshots that are available in your library.
2. Select the snapshot you want to simulate, as shown in Figure 2-4 on page 19.
3. Click the Simulator button.

The Design Browser and the Console window appear. You can access your design hierarchy in the Design Browser and enter SimVision and Tcl simulator commands in the Console window.

Figure 2-5 on page 19 shows the Design Browser at start-up. SimVision places the simulation at the top of the hierarchy and assigns it the name simulator. The top-level of the design hierarchy is placed below the simulation. In this example, it is named test_drink.

At start-up, the Console window has two tabs, as shown in Figure 2-6 on page 20. The SimVision tab lets you enter SimVision commands and the simulator tab lets you enter Tcl simulator commands. As you run the simulation, the Console window also displays messages from SimVision and the simulator.
Exiting from NCLaunch

After invoking the simulator, you can exit NCLaunch.

To exit NCLaunch:

➤ Bring the NCLaunch main window to the foreground and choose File – Exit from the menu bar.
For More Information

This chapter describes how to run NC-Verilog in multi-step invocation mode using NCLaunch. There are other ways to prepare your design for simulation.

<table>
<thead>
<tr>
<th>Tool/Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCLaunch</td>
<td>You can compile the source files, elaborate the design, and start the simulator in a single step using the NCLaunch <code>ncverilog</code> support as described in the <em>NCLaunch User Guide</em>.</td>
</tr>
<tr>
<td>NC-Verilog, Multiple Step mode</td>
<td>You can compile the source files, elaborate the design, and start the simulator using the <code>ncvlog</code>, <code>ncelab</code>, and <code>ncsim</code> commands, as described in <em>Multi-Step Invocation (Library-Based Mode)</em>, in the <em>NC-Verilog Simulator Help</em>.</td>
</tr>
<tr>
<td>NC-Verilog, Single Step mode</td>
<td>You can compile and elaborate the design in a single step using the <code>ncverilog</code> command, as described in <em>Single-Step Invocation with ncverilog</em> in the <em>NC-Verilog Simulator Help</em>.</td>
</tr>
</tbody>
</table>
Simulating the Design

SimVision lets you control and query the simulation of your design. It can help you locate the lines of code associated with a particular simulation event. If you find an error in the design, you can edit the code directly, and compile and elaborate the design again, without leaving the SimVision environment.

SimVision lets you choose the simulation data that you want to save for particular objects or scopes, which can help keep the size of simulation data files as small as possible. At a later time, you can load a simulation data file back into the Waveform window and reexamine the simulation results.

Selecting the Simulation Data to Save

You can save simulation data by executing simulator commands at run time. Simulator commands probe the design during simulation and save the values of the probed objects to a database.

There are two types of probe commands:

- Probe a specific object or objects. The values of the specified objects are saved in the database.
- Probe a scope or scopes. You can choose the type of information you want to save, such as the inputs to that scope, and you can choose whether to probe some or all subscopes.

To probe all objects in all scopes, beginning at the top module:

1. In the Design Browser, click on the + icon next to test_drink to expand the hierarchy.
2. Select the top scope.

The signal list on the right side of the window displays the signals for the top scope, as shown in Figure 3-1 on page 24. The signal list indicates the type of each signal—input, output, inout, internal signal, or transaction.
Figure 3-1 Choose the Top Scope

3. Choose *Simulation – Create Probe* from the menu bar.

SimVision opens the Set Probe form, as shown in Figure 3-2 on page 25. This form lets you probe one or more levels of subscope, choose the type of signals you want to probe, and write the probed information to any database.
Figure 3-2 Set Probe Form

[Image of SimVision: Set Probe dialog box]

- **Probe Name:** (optional)
- **Probe these signals and scopes:** test_drink.top
- **Signal/Scope:**
- **Within each scope, include:**
  - Scope
  - Sub-scope
  - ...all levels...
  - (inputs)
  - (outputs)
  - (inouts)
  - (internals)
- **Include sub-scopes:**
- **Include within each scope:**
- **Store in database:** (default)
- **Add to waveform display**
For this probe:

- Select *Include sub-scopes* and choose *all* from the drop-down list to include all
  the subscopes in the design.

- Select *Include within each scope* and choose *all* from the drop-down list to
  include all inputs, outputs, and ports.

- Deselect *Add to waveform display*.

4. Click *OK* to set the probe and close the form.

**Tip**

If you see the following message in the Console window, you have successfully
created the probe:

```bash
ncsim> database -open waves -into waves.shm -default
Created default SHM database waves
ncsim> probe -create test_drink.top -depth all -all -shm
Created probe 1
```

5. From the Console window, choose *Simulation – Run*. SimVision simulates the design
and saves the simulation data in a default database. As it runs, it displays the following
messages:

```bash
ncsim> run
  400 loading machine with 5 cans
  400 *** machine empty! ***
  700 enter nickel
  900 enter dime
 1100 enter quarter
 1300 enter dime
 1500 enter quarter
 1500 -> drink dispensed

-------------------------------
  1800 enter nickel
 2100 enter nickel
 2300 enter dime
 2500 enter dime
 2600 *** machine empty! ***
 2700 enter quarter
 2700 nickel changed
 3000 enter nickel
 3200 enter dime
 3400 enter quarter
 3600 enter dime
 3800 enter quarter
 3900 -> drink dispensed

-------------------------------
 68400 enter quarter
```
NC-Verilog Simulator Tutorial
Simulating the Design

68700 enter nickel
69000 enter nickel
69200 enter dime
69400 enter dime
Simulation complete via $finish(1) at time 69600 NS + 0
./test_drink.v:48 $finish;
ncsim>

Tip

When you have completed these steps, your working directory should contain a new
directory named waves.shm. The waves.shm directory should contain two files:
waves.dsn and waves.trn. If these files are significantly smaller than 65,500 and
360 bytes, respectively, you did not probe all of the necessary objects during
simulation.

To correct any problems, restart the simulator by choosing Simulation – Reinvoke
Simulator from the Console window or by exiting from the simulator and restarting it with
the following command:

ncsim -gui worklib.test_drink:module

Then rerun the steps in this chapter.

For More Information

This chapter describes how to set a probe and use the Console window to run the simulation.
However, there are other ways to perform these steps, as follows:

<table>
<thead>
<tr>
<th>Tool or Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$shm_open</td>
<td>You can include calls to these system tasks in your Verilog source files.</td>
</tr>
<tr>
<td>$shm_probe</td>
<td>When you simulate the design, these system tasks open your database, set</td>
</tr>
<tr>
<td></td>
<td>the objects that you want to probe, and close the database after the</td>
</tr>
<tr>
<td></td>
<td>simulation has completed.</td>
</tr>
<tr>
<td>$shm_close</td>
<td>See “Displaying Waveforms with the SimVision Waveform Viewer” in the NC-Verilog</td>
</tr>
<tr>
<td></td>
<td>Simulator Help for details on these system tasks.</td>
</tr>
<tr>
<td>database -open</td>
<td>You can call these simulator commands from the Tcl command-line interface,</td>
</tr>
<tr>
<td>probe -create run</td>
<td>or include them in a command file to simulate the design in batch mode.</td>
</tr>
<tr>
<td></td>
<td>See the probe, database, and run commands in the NC-Verilog Simulator Help.</td>
</tr>
</tbody>
</table>
### Tool or Feature

<table>
<thead>
<tr>
<th>Tool or Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation database</td>
<td>You can create different simulation databases for individual components of a design to help with debugging, or you can open simulation databases from other tools.</td>
</tr>
<tr>
<td>management</td>
<td></td>
</tr>
<tr>
<td>Probes</td>
<td>You can enable, disable, and delete probes, or create new probes from the Properties window. You can also create a probe for a signal in the Waveform window without using the Set Probe form.</td>
</tr>
<tr>
<td>Breakpoints</td>
<td>You might want to run the simulation up to a point, then stop so that you can examine the state of the design. To control where simulation stops, you can set breakpoints.</td>
</tr>
<tr>
<td>Simulation control</td>
<td>You can run a simulation, stop at breakpoints, and step into and over subprogram calls. You can also save the state of a simulation, or checkpoint, and restart the simulation from that point.</td>
</tr>
</tbody>
</table>

Working with Simulation Data as Waveforms

Waveforms show the values of signals at any time during simulation. They can help you to understand your design behavior.

The Waveform window lets you choose which signals you want to view and the radix in which you want to display signal values. You can even create a custom format, called a mnemonic map, to display signal values in the form that is most meaningful to you.

Selecting the Signals to Display

In the Design Browser, you can select objects and send them to the Waveform window.

To select the signals that you want to display in the Waveform window:

1. Click the *Show Edit Buffer* button to open the edit buffer, as shown in Figure 4-1 on page 30.

   The edit buffer lets you select objects from several scopes, arrange them in the order that you want, and send them to the Waveform window all at once.
2. From the top module, select the signals that you want to display in the Waveform window. For the sample session, select nickel_in, dime_in, quarter_in, dispense, nickel_out, dime_out, two_dime_out, and clk.

   As you click each signal, SimVision adds it to the edit buffer.

3. Expand top, and select current_state from the vending module to add it to the edit buffer.

4. Click the Waveform button to display these signals in the Waveform window.

5. Click the Hide Edit Buffer button to remove the edit buffer in the Design Browser window.
Displaying Data in the Waveform Window

In the Waveform window, signal names and their current values are displayed on the left; their waveforms are displayed on the right, as shown in Figure 4-2 on page 31.

Figure 4-2 Waveform Window with Waveform Data

Tip
You can access the Design Browser or Design Search window as a sidebar within a Waveform window. For more information, see “Using the Design Browser Sidebar” and “Using the Design Search Sidebar” in the SimVision User Guide.
In the Waveform window, above the waveform data, you can see the beginning and ending times for the simulation data currently displayed. Below the waveform data, the scroll bar shows the entire simulation time. You can adjust the amount of waveform data displayed in the window by entering a new time range.

To enter a new time range:

1. Enter a time range in the *Time Range* text field.
   
   For this example, enter `0:3000`, as shown in Figure 4-3 on page 32.

![Figure 4-3 Entering a New Time Range](image)

2. Press Return to apply the new time range.

3. Save these zoom settings by selecting *Keep this view* from the drop-down list above the waveform data, as shown in Figure 4-4 on page 32.
   
   At any time, you can quickly return to a view by selecting it from the drop-down list.

![Figure 4-4 Saving a View of the Waveform Data](image)

The Waveform window contains two cursors, named *TimeA* and *Baseline*. You can move these cursors to any point in simulation time and use them as reference points.

To enter a new simulation time:

- Either drag the cursor to the desired time or enter a simulation time in the cursor time text field.

  For this example, change the simulation time of *TimeA* to 16,700 ns, as shown in Figure 4-5 on page 33.
Controlling the Appearance of Waveform Data

The Waveform window lets you control the appearance of the waveform data, such as the radix in which you want to view signal values.

For example, to set the radix of the `current_state` variable to decimal:

1. Select `current_state` in the list of objects in the Waveform window.
2. Enable `Decimal` in the `Format – Radix/Mnemonic` menu.

When looking at a waveform, it is sometimes helpful to display the signal values as ASCII strings. For example, when the value of `current_state` is 1, the user has deposited 5 cents in the machine; when `current_state` is 2, the user has deposited 10 cents. It can be helpful when viewing the waveform to display the `current_state` as `five` and `ten` rather than 1 and 2.

To map signal values to ASCII strings, define a mnemonic map:

1. Choose `Windows – Tools – Mnemonic Maps` to open the Properties window for Mnemonic Maps, as shown in Figure 4-6 on page 34.
When you define a mnemonic map, you can define not only the text that is displayed for a particular signal value, but also the way that value is displayed in the Waveform window, including the shape of the waveform, the color, and icons for any special conditions associated with a value.

2. Click the *New Map* button to create a new mnemonic map.

3. Define the first mnemonic map entry as follows:
   a. To change the default radix, click on the ‘h entry and select ‘d.
   b. Double-click in the *Values Matching* field, enter the value 0, and then press Tab to move to the *Relabel As* field.
   c. In the *Relabel As* field, enter the string `idle` and then press Tab to move to the *Values Matching* field of the next entry.
The Preview field shows what the Waveform window will display when the value of current_state is 0. That is, it displays a box containing the string idle.

4. Define the following states:

<table>
<thead>
<tr>
<th>Values Matching</th>
<th>Relabel As</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>idle</td>
</tr>
<tr>
<td>1</td>
<td>five</td>
</tr>
<tr>
<td>2</td>
<td>ten</td>
</tr>
<tr>
<td>3</td>
<td>fifteen</td>
</tr>
<tr>
<td>4</td>
<td>twenty</td>
</tr>
<tr>
<td>5</td>
<td>twenty_five</td>
</tr>
<tr>
<td>6</td>
<td>thirty</td>
</tr>
<tr>
<td>7</td>
<td>thirty_five</td>
</tr>
<tr>
<td>8</td>
<td>forty</td>
</tr>
<tr>
<td>9</td>
<td>forty_five</td>
</tr>
<tr>
<td>10</td>
<td>fifty</td>
</tr>
<tr>
<td>11</td>
<td>nickel_out</td>
</tr>
<tr>
<td>12</td>
<td>dime_out</td>
</tr>
<tr>
<td>13</td>
<td>nickel_dime_out</td>
</tr>
<tr>
<td>14</td>
<td>two_dime_out</td>
</tr>
</tbody>
</table>

5. In the Name field, change New map to Current State.

6. Click Apply To Selected Signals.

Now the values of current_state are displayed as idle, five, fifty, and so on, rather than as their decimal equivalents, 0, 1, 10, and so on, as shown in Figure 4-7 on page 35.

Figure 4-7 Displaying Signal Values with a Mnemonic Map
7. Choose *File – Close Window* to close the Properties window.

**For More Information**

This chapter describes a few of the ways that you can display and organize information in the Waveform window. Other tools and features are available to help you display waveforms.

<table>
<thead>
<tr>
<th>Tool/Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Search window</td>
<td>You can search for signals, variables, and scopes in multiple databases, without regard for the design hierarchy.</td>
</tr>
<tr>
<td>Groups and expressions</td>
<td>You can collect signals into groups and expressions. These features let you treat a set of signals as a single entity.</td>
</tr>
<tr>
<td>Markers</td>
<td>You can place markers, similar to cursors, on the simulation timeline. You can jump from marker to marker to compare simulation results at different times, but you cannot use markers for measurement purposes.</td>
</tr>
<tr>
<td>Measurement window</td>
<td>You can create a table of measurements to examine the characteristics of variables over a specified simulation time. For example, you can look at their slopes and minimum and maximum values. This is useful for measuring the properties of analog signal waveforms.</td>
</tr>
<tr>
<td>Preferences</td>
<td>You can specify the style of toolbars in SimVision windows, set Waveform window defaults, select the text editor for the Source Browser, and otherwise customize SimVision.</td>
</tr>
</tbody>
</table>
Simulation Cycle Debugger

The Simulation Cycle Debugger lets you step through a simulation cycle, stopping at each time point, delta cycle, simulation phase, or scheduled process. It is not available for Verilog-XL or AMS Designer.

Debugging a Design

The Waveform window can help you find errors in your design. By defining conditions and analyzing the waveform data, you can see where errors occur. When you have located the error, you can edit the design source code and restart the simulator to test your changes.

Searching for Conditions in the Waveform

A condition is a combination of signal values that you want to search for in the waveform data. For example, you can define a condition that occurs whenever the `dispense` and `nickel_out` signals have the same value, as follows:

1. Select the `quarter_in` and `dispense` signals in the Waveform window and choose `Edit – Create – Condition`.

   SimVision opens the Expression Calculator, as shown in Figure 5-1 on page 40.
The Expression Calculator creates a default expression that uses the AND logic. This condition expression is true whenever both signals have the same value. You can edit this expression if you want to look for a different condition.

2. Enter a name for the condition expression in the Name field, such as `quarter_inANDdispense`.

3. Click the Waveform button to add the condition to the Waveform window.

4. Choose File – Close Window to close the Expression Calculator.

5. Select the condition in the Waveform window and use the Next Edge and Previous Edge buttons to locate places where the condition occurs.
Analyzing the Waveforms to Find an Error

Analyzing waveforms can help you find problems in a design. For example:

1. Set the simulation time to 16,700 ns.
   
   At this time, the machine is in the idle state.

   **Tip**
   
   There are several ways to set the simulation time:
   - Enter the desired time in the cursor time field.
   - Drag the primary cursor until it reaches the desired time.
   - Select the dispense signal and click the Next Edge button until you reach the desired time.

2. Select the current_state signal and click the Next Edge button to follow the sequence of events from clock cycle to clock cycle:
   - The user adds a nickel, and the machine transitions to the next state, five.
   - The user adds a dime, and the machine transitions to the state fifteen.
   - The user adds a quarter, and the machine transitions to the state forty.
   - The user adds a dime, the machine transitions to the state fifty.

   During this clock cycle, the user also adds a quarter. You would expect the machine to transition immediately to the state twenty_five, but it does not. The machine transitions to the idle state, and sets the dispense signal to 1; the quarter is ignored. During the next clock cycle, the user deposits a nickel, and the machine transitions to the state five.

   **Figure 5-2** on page 42 shows the waveform at the point where the error occurs. Between the time the machine dispenses a drink and the machine returns to the idle state, the quarter_in signal goes high. The machine transitions to the state five in the next clock cycle, when the user deposits a nickel.
Figure 5-2 Locating the Error in the Design

3. From the Time Range field, choose *Keep this View* from the drop-down list so that you can easily go back to this view later.

[Video]

Analyzing Simulation Results in the Waveform Window

Creating Custom Views of Simulation Data

Another way to analyze simulation results is through the Register window, where you can create custom views of the simulation data, including freeform text and graphical elements. A Register window can have several pages, each with its own view.

To create a page in a Register window:

1. From the Waveform window, select the signals that you want to analyze, such as the *nickel_in, dime_in, quarter_in, dispense, nickel_out, dime_out, two_dime_out, clk, and current_state*.

2. Click the *Register* button [Register] to send these signals to a Register window, as shown in Figure 5-3 on page 43.
Along the right side of the window are buttons that let you draw graphical objects, add text, and manage the layout of the objects in the window. Tool tips pop up when you place the cursor over these buttons, telling you what functions they perform.

3. To apply the mnemonic map to the register window, select the current_state signal and choose Format – Radix/Mnemonic – Current State.

4. Arrange the objects any way you want. For example, the layout in Figure 5-4 on page 44 shows the relationship between the inputs—nickel_in, dime_in, and quarter_in, and clk—the state machine variable, current_state, and the outputs—dispense, nickel_out, dime_out, and two_dime_out.
5. Enter a simulation time, such as 16,700 ns.
   The Register window updates the signals to show their values at that time.

6. Select a signal, such as current_state, and click the Next Edge button.
   The time progresses to the next edge of that signal and the Register window updates all
   of the signals to show their values at that time.

7. Click the Previous Edge button to move the simulation time back to the previous
   edge of the selected signal.

8. Start at simulation time 16,700 ns, select the current_state signal, and click the Next
   Edge button to see the same sequence of events in the Register window that was
   shown in the waveform window. That is:
      - The user adds a nickel, and the machine transitions to the next state, five.
      - The user adds a dime, and the machine transitions to the state fifteen.
      - The user adds a quarter, and the machine transitions to the state forty.
      - The user adds a dime, the machine transitions to the state fifty.

---

Fixing an Error in the Source Code

You can use SimVision to locate the line in the source file where an error occurs, as follows:

1. In the Waveform window, select the current_state variable and choose Explore –
   Go To – Cause.
The Signal Flow Browser shows the signal you have selected and a list of the signal’s drivers.

2. Click on the Source Browser button to open the Source Browser.

3. From the Signal Flow Browser, select the first driver: assign current_state = 0, as shown in Figure 5-5 on page 45.

**Figure 5-5 Displaying Drivers in the Signal Flow Browser**

The Source Browser now points to the line in the source file where current_state is set to ‘idle, as shown in Figure 5-6 on page 46.
4. Scroll the list of drivers in the Signal Flow Browser, and you can see that there are other places where `current_state` is set to 0. Click on these drivers to see the lines in the source code where these state transitions occur.

Notice that the code for ‘fifty sets the `current_state` to `idle`, but does not check to see if additional coins are added.

5. To fix the error in the drink machine, choose `Edit – Edit File` from the Source Browser and add the necessary logic to the `case` statement for ‘fifty. For example:
'fifty :                             //4'd10
begin
  dispense <= 1;
  if (nickel_in == 1)
    begin
      current_state <= 'five;
    end
  else if (dime_in == 1)
    begin
      current_state <= 'ten;
    end
  else if (quarter_in == 1)
    begin
      current_state <= 'twenty_five;
    end
  else
    begin
      current_state <= 'idle;
    end
end

Important

You can copy the above code and paste the solution into the source file. However, make sure that the tick marks face the correct direction after being pasted, otherwise the simulation will fail.

6. Save these changes to the file.


   Note: If the Reinvoke dialog box appears, click Yes.

   SimVision compiles and elaborates the design, and restarts the simulator. All of the SimVision windows that were opened in the previous session are opened again, and the mnemonic map and condition that you created are still defined.

8. In the Console window, choose Simulation – Run to generate new simulation data.

9. In the Waveform window, display the view that you previously saved; you can see that the machine transitions correctly to state twenty_five.

Ending a SimVision Session

To exit from SimVision:


2. If the Waveform window remains open, choose File – Exit SimVision from the Waveform window.
SimVision displays a confirmation message.

3. Click Yes to exit and close all SimVision windows.

**For More Information**

This chapter describes some basic steps for debugging a design. You may also find these other debugging tools and features helpful, as follows:

<table>
<thead>
<tr>
<th>Tool/Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement window</td>
<td>You create a table of measurements for selected variables during a specified period of simulation time. For example, you can look at the value, slope, and minimum and maximum values of the variables, which is useful for measuring the properties of analog signal waveforms. See Chapter 20, “Measuring Signal Values,” in the <em>SimVision User Guide.</em></td>
</tr>
</tbody>
</table>