

VeriLogger Pro

User's Manual

www.syncad.com

VeriLogger Pro Manual (rev 6.5A) copyright 1999 SynaptiCAD

Trademarks

- Timing Diagrammer Pro, WaveFormer Pro, TestBench Pro, VeriLogger Pro, DataSheet Pro, and SynaptiCAD are trademarks of SynaptiCAD Inc.
- Verilog-XL is a trademark of Cadence Design Systems, Inc.
- Windows, Windows NT, and Windows 95 are registered trademarks of Microsoft.

All other brand and product names are the trademarks of their respective holders.

Information in this documentation is subject to change without notice and does not represent a commitment on the part of SynaptiCAD. The software and associated documentation is provided under a license agreement and is the property of SynaptiCAD. Copying the software in violation of Federal Copyright Law is a criminal offense. Violators will be prosecuted to the full extent of the law.

No part of this document may be reproduced or transmitted in any manner or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the written permission of SynaptiCAD.

For latest product information and updates contact SynaptiCAD at:

web site: <http://www.syncad.com>

email: sales@syncad.com

phone: (800)804-7073 or (540)953-3390

Chapter 1: A Quick Start to VeriLogger	5
Step 1: Add Files to the Project	5
Step 2: Build the Project	5
Step 3: Watch Signals and Components	5
Step 4: Simulate the Project	6
Step 5: Debug the Project	6
Step 6: Use Breakpoints and Single Step Debugging Techniques	7
Step 7: Save the Project, Code, and Waveform Files	8
Chapter 2: Project Functions	9
2.1 Opening, Saving, and Starting New Projects	9
2.2 Adding Files and Building the Project	9
2.3 Using the Project Tree Control	10
2.4 Watching Signals and Components	10
2.5 Using the Project Preferences Dialog	12
Chapter 3: Simulation and Debugging Functions	14
3.1 Simulation Button Bar	14
3.2 Displaying Errors, Breakpoints, and Results in the Report Window	15
3.3 Interactive Debugging in the Console Window	16
3.4 Simulation Control Commands	16
3.5 Interactive Verilog Commands	16
3.6 System Tasks	17
3.7 Compiler Directives	18
3.8 Waveform Comparisons	18
Chapter 4: Editor Functions	21
4.1 Opening, Saving, and Creating New Source Code	21
4.2 Displaying or Finding a Specific Line of Code	21
4.3 Using the Editor Preferences Dialog	22
4.4 Editor Cursor Commands	22
4.5 Using an External Editor	24
Chapter 5: Waveforms and Test Bench Generation	25
5.1 Grouping Waveforms for Different Simulation Runs	25
5.2 Automatic Test Bench Generation and Project Tree Options	25
5.3 Drawing Waveforms for Stimulus Generation	26
5.4 Generating VCD files	26
5.5 Reading VCD files	26
5.6 Working in the Diagram Window	27
Chapter 6: Faster Simulations with the Command Line Simulator	28
6.1 Preparing Verilog Source Files	28
6.2 Using the Command Line Simulator	28
6.3 Command Line Simulation Options	29
6.4 Predefined Plus Options	30
6.5 Simulator Control Commands	31
Chapter 7: Back-Annotated Simulation (Gate-Level Timing Simulation)	32
7.1 Using the Standard Delay File (SDF)	32
Chapter 8: VeriLogger Technology Background	33
8.1 Compilation Process	33
8.2 User-Defined Primitives and Memory Usage	33

8.3 Notes on Using Specify Blocks	33
Appendix A: VeriLogger Implementation Notes	34
A.1 Port Collapsing	34
A.2 Port Connections of Different Net Types	34
A.3 Working Around Pullup/Pulldown Gates	34
A.4 Using Trace Implementation	34
A.5 Predefined Macro __VERIWELL__	35
A.6 Simulation Statistics	35
A.7 Displaying the Location of the Last Value Edited	35
A.8 User Interrupt	35
Appendix B: Implementation Differences from Verilog-XLTM	36
B.1 Event Ordering	36
B.2 Module Ports and Port Collapsing	36
B.3 Control Expressions are Limited to 32 Bits	36
B.4 The \$Monitor System Task	36
Appendix C: License Agreement	37
VeriLogger Tutorial A: Basic Verilog Simulation	39
Part 1: Project Management and Simulation	39
1.1) Add Files to the Project	40
1.2) Build the Tree and use the Editor Windows	40
1.3) Simulate the Project	41
1.4) Watch and View Internal Signals	41
1.5) Save the Project, Waveforms and Source Code	41
1.6) Configure the Project for Part 2	42
Part 2: Graphical Test Bench Generation	42
2.1) Build the Project and Examine the Black Signals	42
2.2) Use the Debug Run Simulation Mode	43
2.3) How to Draw Waveforms	43
2.4) How to Edit Waveforms	43
2.5) Draw the Stimulus Waveforms	44
2.6) Simulate using the Auto Run Simulation Mode	44
2.7) Import and Generate Waveforms	45
Part 3: Breakpoints, Stepping, and Tracing	45
Index	47

Chapter 1: A Quick Start to VeriLogger

This chapter will cover all the basic steps involved in creating and simulating a Verilog project in VeriLogger. Please note that these are the steps to getting started. For more detailed information, refer to Chapters 2-7.

Step 1: Add Files to the Project

The VeriLogger simulator uses a project to control simulation options, set the files to be simulated, and set watches on signals (view signal waveforms). The first step in creating a project is to add Verilog files to the project's tree window.

To add a file to the project:

- Right click in the Project window to open the context menu, and choose the **Add HDL File(s)** menu option
- OR, choose the **Project > Add HDL File(s)** menu option.

Both of these functions open a file dialog. Determine the files that you would like to add to the project and click the **Open** button to close the dialog.

Notice that the filenames are listed in the Project window. The Verilog source code can be viewed by double clicking on the filename.

Step 2: Build the Project

When files are first added to the project, you can see the filename but you cannot see a hierarchical view of the modules inside the files. To view the internal modules on the project tree you must first **build** or **run** a simulation. The **build** command compiles the Verilog files and builds the Verilog tree. It does not run a simulation. For large projects **build** allows you to quickly construct the tree without having to wait for a simulation to run.

There are three ways to **build** a project:

- Press the yellow **Build** button on the simulation button bar,
- Choose the **Simulate > Build** menu option,
- OR, Press the <F7> key

Once the project is built you can view all the modules, signals, ports, and components in the Verilog files. One module name will be surrounded by brackets <<name>>. This is the top-level module for the project. It is the highest level-instantiated component. All sub-modules can be viewed by descending the top-level module's tree.

After you build the project, the signals or the ports in the top-level module are automatically added to the Diagram window. If the top-level module does not have port signals, the internal signals of the module are displayed. If the top-level module has port signals, the output ports are viewed as purple signals and input ports are viewed as black signals. The black input signals can be edited to provide stimulus to the top-level module. The waveform drawing functions are covered in Chapter 1 of the SynaptiCAD manual or on-line help.

The ability to draw waveform stimulus and immediately simulate is one of the unique features of VeriLogger. If the top-level module has ports, the project automatically wraps a test bench around the top-level module and creates signals in that test bench to drive and watch the user's top-level module. This makes it easy to quickly test small parts of a design before the design is complete.

Step 3: Watch Signals and Components

VeriLogger has two main output displays for the results of a simulation. The first is the **verilog.log** file that captures the simulator messages and any output from display statements embedded in the code. The second is the Diagram window. By default, only the top-level signals or ports are watched in the Diagram window.

To watch signals of components instantiated by the top-level component:

- Click on the plus sign to the left of the top-level module <<name>>. This will expand the top-level tree.
- Continue to open the top-level tree until you are able to see a signal or component that you would like to watch.
- Right click on the signal or component to open the context menu for that item.
- Select the **Watch Connection** or **Watch Component** menu option. This will add the signal with a full path reference to the Diagram window. If you are watching a component, then all the top-level signals of that component will be added to the Diagram window.

Step 4: Simulate the Project

To simulate the project:

- Press the **Run** button (large green triangle) on the simulation button bar,
- Choose the **Simulate > Run** menu option,
- OR, Press the <F5> key.

Once a project is simulated the waveforms will be displayed in the Diagram window. If you do not want to continue to watch a particular signal, left click on the signal name in the Diagram window and press the <Delete> key.

VeriLogger has two simulation modes, **Auto Run** and **Debug Run** that determine when a simulation is performed. The current simulation mode is displayed on the leftmost button on the simulation button bar. In the **Debug Run** simulation mode, simulations are started only when the user presses the **Run** or **Single Step** buttons (similar to a standard Verilog simulator). In the **Auto Run** simulation mode, the simulator will automatically run a simulation each time a waveform is added or modified in the Diagram window. This mode makes it easy to quickly test small modules and do bottom-up testing. Press the mode button to toggle between the two simulation modes.

VeriLogger also has two simulation states, **Sim Diagram & Project** and **Sim Project**, that determine what is simulated. The **Sim Diagram & Project** mode indicates that both the diagram waveforms and the Verilog source code should be simulated together. The **Sim Project** mode indicates that Verilog source code and the diagram should be simulated independently of each other. In this mode the Simulation button bar controls the simulation of the project, and the **Simulate Diagram** button in the Simulation button bar controls the simulation of the diagram. In the **Sim Project** mode waveforms can not be used as stimulus to drive signals contained in the project source code. This mode lets the user perform timing analysis on waveforms contained in the diagram without incurring the overhead of having to re-simulate the project. By default, VeriLogger should remain in the **Sim Diagram & Project** mode so that the waveforms are used to stimulate the project.

Step 5: Debug the Project

In VeriLogger there are two places to check the status of your simulation. The simulation status indicator on the button bar and the status bar in the lower right corner both display success/failure information about the last simulation. If **Simulation Error** or **Compile Error** are displayed on the button, then there is an error in the Verilog source code files in the project.

If your simulation fails, there are two different files in the *Report* window you can check to find out why the simulation failed. The first file is the **verilog.log** file. This file displays all the available information about the current simulation. The second place is the **Error** tab in the *Report* window. This tab displays the errors in a more concise manner.

To jump to a particular error:

- Press the **Error** tab in the *Report* window to display the *Error* window.
- Double click on an error. This will open the offending file in the *Editor* window and place the cursor at the error line.

If you have used the interactive simulation features, such as direct entry of Boolean equations in the *Signal Properties* dialog, then a third file, **waveperl.err**, will help you locate any errors. This file is covered in the WaveFormer Pro documentation.

Step 6: Use Breakpoints and Single Step Debugging Techniques

VeriLogger supports graphical breakpoints, single step debugging, and \$display statements.

To define breakpoints in the code:

- Open the Verilog file in an editor window, either by double clicking in the project window or by choosing the **Editor > Open HDL Source** menu option.
- In the *Editor* window, notice the black strip to the left of the source code. This is the breakpoint window.
- Left click on the breakpoint window at the line of code that you wish to stop the simulation. This will do two things: first, a red dot will be added to the breakpoint editor window; second, the breakpoint will be listed in the **breakpoint** tab in the *Report* window.

Note: Breakpoints can be removed by clicking on them in the editor window.

There are two single-step buttons:



The **Step over Calls** button has two medium size triangles on it to indicate that you are stepping to the next line of code at the current level.



The **Step into Calls** button has a small triangle between two medium-sized triangles on it to indicate that you are going to step into the next level of code.

It is best to use the single step buttons in conjunction with breakpoints, so that if you step into a loop you can use the **Run** button to simulate to the next breakpoint.

There are also three other buttons that will help you debug your simulation:



The **Stop** button stops the current simulation.



The **Restart** button stops the current simulation and clears the Diagram window. It sets the simulation to execute the first statement in the project when the **run** or **single step** buttons are pressed.



The **Goto** button opens an editor window and displays the currently executing line of HDL code.

VeriLogger also supports the Verilog **\$display** statements. Output from these statements goes directly to the **verilog.log** file displayed in the Report window.

Step 7: Save the Project, Code, and Waveform Files

In VeriLogger there are three types of files associated with a Verilog project.

- Project files have an extension of **HPJ** and are saved by using the **Project > Save HDL Project** menu option. This saves the file list and simulation options. It does not save the watch settings.
- HDL Source code files usually have an extension of **V** and are saved by selecting the editor window and choosing the **Editor > Save HDL Code** menu option.
- Waveform diagram files have an extension of **TIM** and are saved using the **File > Save Timing Diagram** menu option. This file saves any watched signals.

Saving watched signals in a separate file allows you to build several different test cases so you can compare and contrast future simulation results.

Chapter 2: Project Functions

VeriLogger Pro uses a project to list the files to be simulated, set watches on signals (view signal waveforms), and store simulation options.

2.1 Opening, Saving, and Starting New Projects

Projects are opened and saved using the **Project** menu options. By default, VeriLogger opens with a new untitled project.

To open an existing project:

- Select the **Project > Open HDL Project** menu option. This opens the *Open Project File* dialog where you can select a project file to open.

To save an open project:

- Select the **Project > Save HDL Project** menu option to open a *Save* dialog. By default, project filenames have an extension of **HPJ**.

To clear the current project and start a new project:

- Select the **Project > New HDL Project** menu option. You will be asked several questions about saving the files that are currently open and then a new project will be created.

2.2 Adding Files and Building the Project

The first step in creating a project is to add Verilog files to the project's tree window.

To add a file to the project:

- Right click in the *Project* window to open the context menu, and select the **Add HDL File(s)** menu option,
- OR, choose the **Project > Add HDL File(s)** menu option.
- Both of these functions open a *File* dialog. Select one or more files to add to the project and click the **Open** button to close the dialog.

When files are first added to the project, you can see the filename but you cannot see a hierarchical view of the modules inside the files. To view the internal modules on the project tree you must first **build** or **run** a simulation. The **build** command compiles the Verilog files and builds the Verilog tree. It does not run a simulation. For large projects, **build** allows you to quickly construct the tree without having to wait for a simulation to run.

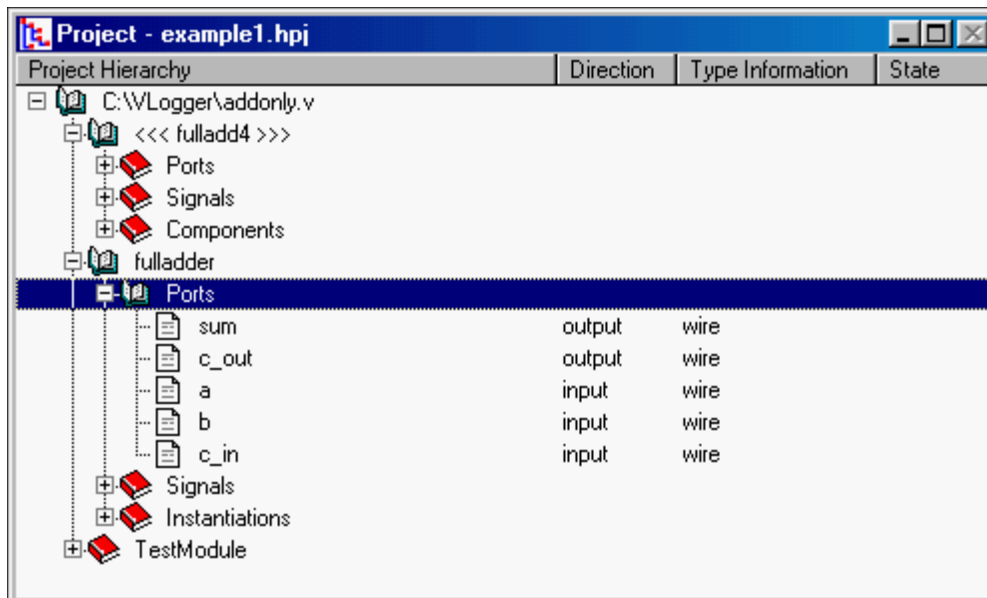
There are three ways to build a project:

- Press the yellow **Build** button on the simulation button bar,
- Select the **Simulate > Build** menu option,
- OR, press the <F7> key

After the project is built you can view all the modules, signals, ports, and components in the Verilog files. One module name will be surrounded by brackets <<name>>. This is the top-level module for the project. The top-level module is the highest level-instantiated component. All sub-modules can be viewed by descending the top-level module's tree.

2.3 Using the Project Tree Control



The Project Tree control is used to investigate the hierarchical structure of the Verilog components, view source code, and set watches on signals. Each node in the tree has a context sensitive pop-up menu that can be opened by right clicking on the node. The following functions describe the general viewing features of the tree control.



Viewing source code:

- **To open a Verilog file:** double click on a filename to open the file in a new editor window
- **To see the declaration of a signal or component:** double click on the signal or component to jump to the declaration in the HDL source code.

Expanding or hiding branches of a tree:

- Click  or  to expand or close a node. This will not open sub-nodes.
- To expand a node and all sub-nodes: right click on a node and choose **Expand Item** from the pop-up menu.
- To hide the entire tree except the selected node and sub-nodes: right click on a node and choose **Set Starting Point** from the pop-up menu.
- To return the starting point to the file level: choose **Reset Starting Point** from the right click pop-up menu.

2.4 Watching Signals and Components

The *Project* window is used to mark signals to be watched in the Diagram window. To maximize simulation speed, Verilog simulators do not automatically store signal transition times unless a signal is specifically tagged as one to watch.

To set a signal, component, or port to be watched:

- Expand the tree until you locate the signal or component that you would like to watch.
- Right click on the component to open the context menu for that item.
- Choose one of the **Watch** menu options. The watch menu options vary according to what type of object is selected.

- Notice that one or more signal names have been added to the Diagram window. The full path name of the signal is used in the Diagram window. The waveform data will be displayed after the next simulation run.

Watching a signal in a specific instance of a module: Most of the time you will want to expand the tree starting from the top-level module <<name>>. By expanding from the top-level tree you will be able to watch signals in specific instances of a module.

Watching a signal in all instances of a module: Expand the tree, starting from the filename where the module is defined. You can choose to watch all signals in a component, or in both the component and sub-components.

Automatic Watches in the "Sim Diagram & Project" mode: After you build the project, the signals or the ports in the top-level module are automatically added to the Diagram window. If the top-level module does not have port signals, the internal signals of the module are viewed. If the top-level module has port signals, the output ports are viewed as purple signals and input ports are viewed as black signals. You can edit the black input signals to provide stimulus to the top-level module. The waveform drawing functions are covered in Chapter 1 of the SynaptiCAD manual or on-line help.

Note: In the **Sim Project** mode, the top-level module *signals* are watched instead of the top-level module *ports* regardless of whether or not ports exist in the top-level module, because the timing diagram is not being used to generate stimulus for the ports.

The ability to draw waveform stimulus and immediately simulate is one of the unique features of VeriLogger. If the top-level module has ports, the project wraps a test bench around the top-level module automatically and creates signals in that test bench to drive and watch the user's top-level module. This makes it easy to quickly test small parts of a design before the design is complete.

The automatic port watching feature can be turned off by unchecking the **Grab top level signals** checkbox located in the **Project > Project Settings** menu.

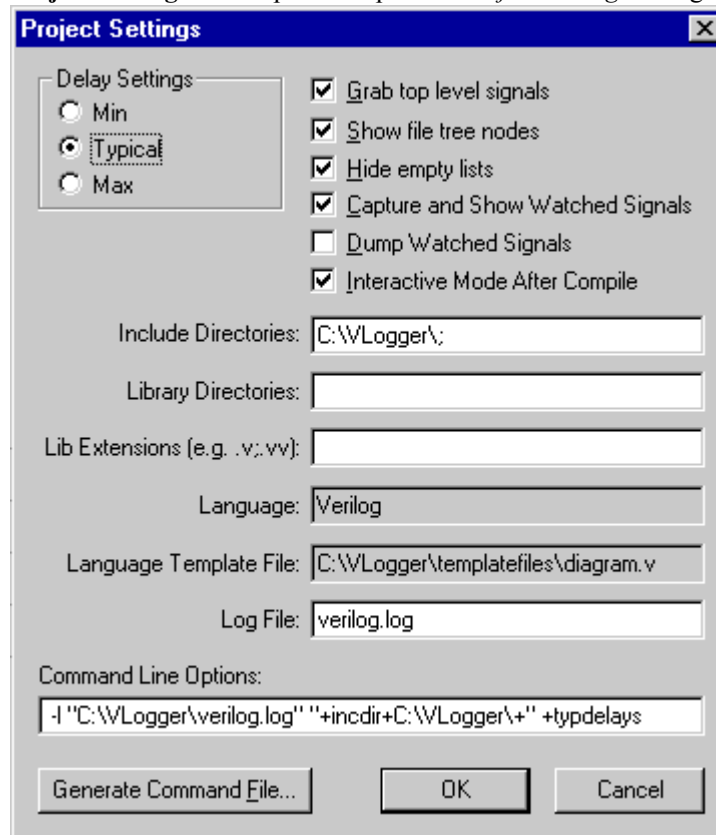
To stop watching a signal:

- In the Diagram window, left click on the signal name to select it.
- Press the <Delete> key on the keyboard. You may delete the entire signal including the automatically generated signal names.

2.5 Using the Project Preferences Dialog

The *Project Settings* dialog determines the simulator run time options. This information is stored inside the project **HPJ** file. To open the *Project Settings* dialog:

- Use the **Project > Project Settings** menu option to open the *Project Settings* dialog.



There are several sections in the *Project Settings* dialog.

- The **Delay Settings** radio buttons determines which delay value is used in **min:typ:max** expressions. This is similar to the **+maxdelays**, **+mindelays**, and **+typdelays** command line simulator options.
- The **Grab top level signals** check box turns on the automatic monitoring of ports or internal signals in the top-level module.
- The **Show file tree nodes** check box allows filenames to be shown in the project tree.
- The **Hide empty lists** check box hides nodes without any leaf nodes. Checking this makes the project tree more readable.
- The **Capture and Show Watched Signals** check box enables the display of waveform results from a simulation run.
- The **Dump Watched Signals** check box will generate a dump file for any watched signals in the diagram. The generated file will have the same name as the .tim file, only with an extension of **.VCD**.
- **Include Directories** edit box specifies the directories where VeriLogger searches for included files. The following is a Windows example (Unix users should use the / slashes):

```
C:\design\project;c:\design\library
```

- The **Library Directories** edit box lists the path and directories where VeriLogger searches for library files. VeriLogger will try to match any undefined modules with the names of the files that have one of the file extensions listed in the **Lib Extensions** edit box. The simulator does not look inside a file unless the undefined module name exactly matches a filename. The simulator does not look at any files unless there are file extensions listed in the **Lib Extensions** edit box. The following is a Windows example (Unix users should use the / slashes):

```
C:\design\project;c:\design\library
```

- The **Lib Extensions** edit box specifies the filename extension used when searching for library files in the library directory. Each library extension should begin with the period character followed by the extension name. Use a semicolon to separate multiple file extensions.

```
.v;.vv
```

- The **Logfile** specifies the name of the log file which receives all the simulation results and information. By default VeriLogger uses **verilog.log** file.
- The **Command Line Options** edit box contains the command line equivalents for all the options that are checked in this dialog. When the **Generate Command File** button is pushed, the text contained in the **Command Line Options** edit box along with the list of Verilog files specified in the project window are written to a command file. This file can then be used with the **Command line version** of the VeriLogger simulator.

Chapter 3: Simulation and Debugging Functions










One of the unique features of VeriLogger is that it has two simulation modes: **Auto Run** and **Debug Run**. The active simulation mode is displayed on the left most button on the simulation button bar. In **Debug Run** mode, simulations are started only when the user presses the **Run** or **Single Step** buttons (similar to a standard Verilog simulator). In **Auto Run** mode the simulator will automatically run a simulation each time a waveform is drawn or changed in the Diagram window. This mode makes it easy to quickly test small modules and do bottom-up testing. Press the mode button to toggle between the two simulation modes.

VeriLogger also has two simulation states, **Sim Diagram & Project** and **Sim Project**. The simulation states determine what is simulated. The **Sim Diagram & Project** mode indicates that both the diagram waveforms and the Verilog source code should be simulated together. The **Sim Project** mode indicates that Verilog source code and the diagram should be simulated independently of each other. In this mode the Simulation button bar controls the simulation of the project, and the **Simulate Diagram** button in the Simulation button bar controls the simulation of the diagram. In the **Sim Project** mode waveforms can not be used as stimulus to drive signals contained in the project source code. This mode lets the user perform timing analysis on waveforms contained in the diagram without incurring the overhead of having to re-simulate the project. By default, VeriLogger should be left in the **Sim Diagram & Project** mode so that the waveforms can be used to stimulate the project.

3.1 Simulation Button Bar

The simulation and debugging functions in VeriLogger are accessed from the simulation button bar located at the top of the main window.



-  **Build Test Bench** - This is a TestBencher button; it is not used in VeriLogger Pro.
-  **Build** - compiles the project files and builds the Verilog tree. It does not run a simulation.
-  **Run/Resume** - compiles the files and runs a simulation. This button also continues a simulation when it reaches a breakpoint.
-  **Single Step – Step Over calls** - steps to the next line of code. It does not step into function calls.
-  **Single Step – Step Into and trace calls** - steps to the next line of code and also sends a trace statement to the `verilog.log` file. This button will also step into function calls.
-  **Stop** - stops the simulation and places the simulator into interactive debugging mode. This button is only active during a simulation.
-  **Restart** - kills the current simulation, clears the Diagram window, and restarts the simulation at time zero.
-  **Goto** - opens an editor at the last line of code executed. Use this button when the simulation is stopped.
-  **Top Scope** – changes the interactive scope for console commands to the scope of the top-level module.

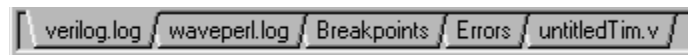
S **Local Scope** – changes the interactive scope for console commands to the scope of the last line of code to be executed.

S **Expand to Local Scope** – selects and expands the module node in the tree control that contains the last line executed.

Console Window (the drop-down edit box) accepts Verilog commands and special simulator commands. Most of this chapter is dedicated to describing the types of commands accepted by the console window.

3.2 Displaying Errors, Breakpoints, and Results in the Report Window

The *Report* window manages several tab windows, three of which are important to simulation and debugging.



The first important tab is the *verilog.log* tab. This tab contains the default log file for VeriLogger. All information generated by the simulator, such as compiler messages, and all user-generated messages from \$display tasks and traces are sent to this file. During a simulation run you should watch the *verilog.log* file for important messages. Click on the **verilog.log** tab to view this file.

The second important tab is the *Breakpoints* tab. This tab lists the breakpoints in the current project.

To add a breakpoint to a line in the source code:

- In an editor window, left click on the black line on the left side of the window. This adds a breakpoint, indicated by the red circle on the line. It also adds a breakpoint listing to the *Breakpoints* tab in the *Report* window.
- OR, right click anywhere in the *Breakpoints* tab window and select the **Add Breakpoint** option from the pop-up menu.
Note: In the *Add Breakpoint* dialog, enter the full path of the file in the **File:** edit box.

```
xor (s1, a, b);
and (c1, a, b);
xor (sum, s1, c_in);
and (c2, s1, c_in);
xor (c_out, c2, c1);
```

The breakpoint tab window also supports several mouse-oriented features:

- Double clicking on a breakpoint will open an editor starting at that line in the source code.
- Right clicking anywhere in the tab window will open a menu allowing you to add, edit, or delete breakpoints.
- Left clicking on a red breakpoint button will toggle it between active and inactive states. An inactive breakpoint is displayed as a small red circle and is ignored in a simulation.

Type	File Name	Line #	Time(ns)	Expression
●	Source Line C:\VLogger\addonly.v	10		
▪	Source Line C:\VLogger\addonly.v	20		
●	Source Line C:\VLogger\addonly.v	30		

The third important tab is the *Errors* tab. If the compiler discovers an error, a message is returned to the *verilog.log* tab and the *Errors* tab. The *verilog.log* tab displays all simulation information and finding a specific error can be difficult. Using the *Errors* tab enables you to quickly view all simulation errors. Double clicking on an error in the *Errors* tab will open an editor starting at the line of source code where the error was found.

3.3 Interactive Debugging in the Console Window

VeriLogger has an interactive command console that can be used to enter Verilog commands to observe, control, and debug the simulation. The interactive command console is the drop-down edit box located on the simulation button bar. During a simulation you can enter a Verilog command such as **\$finish;** (to control the simulation) or **\$display;** (to display the value of a variable).


To use the command console window:

- Stop the simulator during a simulation run. When a simulation is stopped, the simulation display on the status bar turns bright green and reports **Simulation Stopped**. There are four ways to stop the graphical simulator during a simulation run:
 - 1) Single-step into a simulation using the green single-step buttons.
 - 2) Insert a breakpoint.
 - 3) Press the **STOP** button (useful for long simulations).
 - 4) Embed a **\$stop** system task in one of your source code files.
- Type a command into the console window and press the **<Enter>** key.

The next two sections describe the simulation control commands and the types of Verilog commands that can be entered into the console window.

3.4 Simulation Control Commands

VeriLogger supports commands entered in the console window that perform the same functions as the simulation buttons. One of the advantages of console commands is that several commands can be entered at the same time.

For example:  would cause the simulator to execute five lines of code.

VeriLogger supports the following commands:

- To **continue** the simulation, type the period (.) character, or press the green **Run** button.
- To **step** to the next statement in the code, type the semicolon (;) character, or press the **Step Over** button.
- To **step-and-trace** (step to the next statement in the code and generate a trace message in the **verilog.log** file) type the comma (,) character, or press the **Step Into** button.
- To **display the current code-line** execution, (open an editor window and display the currently executing line of HDL code) type the colon (:) character, or press the **Goto** button (the magnifying glass).
- To **terminate** the simulation, type the **\$finish;** command or press the red **STOP** button.

3.5 Interactive Verilog Commands

Interactive Verilog commands are commands entered in the console window that act as lines of code added to the Verilog code in your source files. Generally, any behavioral statement used within an **initial** or **always** block can be entered into the console window. Statements that affect the project structurally, such as instantiating a model, are not allowed. All system tasks are accepted in the console window.

Because all Verilog commands require a terminating semicolon, the semicolon must be entered in the console window. Below are some examples of useful interactive commands:

- **Displaying Variables:** Use the **\$display(...);** system task to view a variable's current value. Make sure that the scope is correct. A common mistake is to view a trace, pause the simulation, and type **\$display;** without realizing that the variable may not be in the current scope. In interactive mode, the current scope is set using the

scope buttons or the **\$scope** system task. By default, the scope is set to the top-level module, not the scope at the current execution line. For example, the following statement could be used to view the variable **ireg**:

```
$scope (top.cpu1.iunit);
$display (ireg);
// OR, this can be expressed as a single statement
$display (top.cpu1.iunit.ireg);
```

All the variables in a given scope can be displayed using the **\$showvars** system task. **\$showvars** also displays the information about when the variable was last modified, specifically, the simulation time, the filename, and the line number of the reference.

- **Changing Variables:** Use an assignment statement to change a variable's value.

```
ireg = 4 * bar;
```

- **Variable Watches (breakpoints):** Interactive statements can be used to stop the simulation when a particular variable, or combination of variables, changes. For example:

```
@(top.cpu1.iunit.ireg) $stop;
```

This code will continue the simulation until the variable changes. However, this statement will not necessarily be the first statement executed after the variable changes. Due to the non-determinacy of Verilog code execution, other statements scheduled to execute at the same time unit may execute before the **\$stop** statement is performed.

- **Timed simulations:** A simulation can be set to run for a certain length of simulation time using a delay and the **\$stop** directive. The following statement suspends and waits for 1000 simulation time units to pass. After 1000 time units, the simulation is stopped.

```
#1000 $stop;
```

3.6 System Tasks

System tasks are special simulator commands that can be embedded into Verilog source code or passed to the simulator during a simulation run. During a simulation run, system tasks are entered through either the console window or the command line, depending on which simulator is being used. System tasks are standard Verilog functions found in a Verilog reference manual. The following is a list of useful system tasks:

Simulation Control Commands

\$finish ends the simulation.

\$stop stops the simulation and enters the interactive command mode. Usually this command is used in conjunction with a delay statement like: **#1000 \$stop;**

\$log and **\$nolog** enable and disable the log file for the current simulation. By default, the log file is enabled.

\$key and **\$nokey** enable and disable the key file that remembers the keystrokes entered at runtime. By default, the key file is created every time you enter the interactive simulation mode.

Debug Commands

\$display(formatString [,argument]...) displays the value of a variable. The syntax is similar to the **fprintf** statement in the C language.

\$scope(scope_path) sets the scope of the interactive simulator.

\$showvars displays all the variables at a given scope.

\$settrace and **\$cleartrace** enable and disable the trace mode that displays each Verilog statement as it is executed in the **verilog.log** file. **Related Function:** Entering the comma (,) character into the console or command line executes a single line of code and generates a trace message in the **verilog.log** file. This is called the **step-and-trace** mode.

VeriLogger VCD Commands

\$dumpfile("filename") specifies the waveform dump filename. If this system task is not used, the default filename is **verilog.vcd**.

\$dumpvars(level, module1_or_signal1, module2_or_signal2, ...) specifies the set of signals to probe for later viewing. When **\$dumpvars** is specified without any arguments, all signals are probed. When **\$dumpvars** is specified with arguments, the first argument indicates how many levels below each specified module instance to probe. Either modules or signals may be specified. When the level is specified to be 0, all levels below the listed modules will be probed. All signals will be saved in a file that can be opened after the simulation is complete.

\$dumpson and **\$dumpoff** enable and disable the VCD dump features.

\$dumpall shows the current value of all VCD variables.

3.7 Compiler Directives

VeriLogger supports all Verilog compiler directives. These directives control how a simulation is compiled. A complete list can be found in a Verilog reference manual.

The first thing you will notice when looking at the following compiler examples, is that all compiler directives begin with an accent grave (`) character. On standard keyboards there are two keys that look almost the same: the single quote (') and the accent grave (`). Be sure that you do not use the single quote by mistake.

The following is a short list of the most useful directives.

`timescale timeUnit / precisionUnit - specifies the time unit and precision of the simulation. The time unit is the unit of measurement for time values such as the delay values and simulation time. The time precision specifies how VeriLogger controls time values.

```
`timescale 1 ns /1 ps
```

`include filename – allows the insertion of the entire contents of a source file into another source file during compilation.

```
`include "count.v"
```

`define macroname textstring - creates macros for text substitution that can be used anywhere within the source text.

```
`define EXTEND 32
```

Conditional compilation directives allow you to select specific pieces of source code to compile based on macro definition.

<pre>`ifdef macroname verilog source `else verilog source `endif</pre>	<p>For example:</p> <pre>`ifdef EXTEND \$display("Using extended mode"); `else \$display("Using normal mode"); `endif</pre>
--	---

3.8 Waveform Comparisons

Waveform comparisons graphically display the differences between compared waveforms for two timing diagrams or individual signals. This feature is exceptionally useful comparing two different simulation runs, as well as for comparing logic analyzer data to a simulation run. The specific regions where waveforms differ turn red when the two waveforms are compared. By using the navigation buttons on the **compare** toolbar, you will be able to jump to the first

occurrence and browse from one difference to the next. A range of **tolerance** can be provided using the compare signal settings.

When comparing two waveforms, the signal names must match. This can be accomplished by changing the names of the compare signals in the *Signal Properties* dialog (see below) or by ensuring the that signal names in the two files match.

Comparing two timing diagrams

The timing diagrams that are being compared can be the result of two different simulation runs, or one or both could contain the data from a logic analyzer. To compare two timing diagrams:

- Load in the first timing diagram. Either use the **File > Open Timing Diagram** menu option to load a new file or use the current timing diagram.
- Choose the **File > Compare Timing Diagram** menu option to load in the signals to be compared. This opens a File dialog which lets you select the file to be compared. Closing this dialog loads the second set of signals and sets their signal type to compare. Any two signals that have matching names will automatically be compared. The compare signals will appear in red under the corresponding original signals.

Comparing two signals in the same timing diagram

A signal can be changed to a compare signal to be used for comparison. This method works for both unmatched signals in files that you are comparing, or for signals that you have created in this file. Any difference between signals that are being compared appears in red on the timing diagram. Multiple pairs of waveforms can be compared in this manner. To compare two signals in the signals must have the same name and one signal must have a signal type of compare.

- Double click on the signal name to open the *Signals Properties* dialog.
- Change the name of the signal to match the signal you want to compare with. This will not change the signal name in any other file, only in the file that you are working with.
- Select the **Compare** radio button located in the top part of the dialog. This makes the *Signals Properties* dialog display the tolerance controls used to define how the compare will be done.
- Next push the **Compare** button to force a comparison to be run.

Finding Compare Errors

Once you have made modifications, you can compare the two sets of signals again by clicking the Compare button on the compare toolbar (far right toolbar; the Compare button has a lightning bolt icon). Other buttons on the compare toolbar allow you to quickly find and move to the differences that are located. The three buttons are:

- **First**, which moves to the first difference that has occurred,
- **Previous**, which moves back one, and
- **Next**, which moves forward one difference.

Adjusting Comparison Tolerances

Ranges may also be used for comparisons. Instead of looking for comparisons directly on an edge, you can allow a tolerance within a set range of the edge. To set the tolerance on a compare signal:

- Open the *Signal Properties* dialog by double clicking left on the signal name. Note: the tolerance can only be set on the compare signal not on the original signal.
- Specify the tolerance range previous to the edge by typing a value into the **-Tol:** textbox. (The value will be ns.)
- Specify the tolerance range after the edge by typing a value into the **+Tol:** textbox. (This value will also be in ns.)

- Click the **OK** button to close the dialog. Now when you run the comparison a tolerance will be provided as specified.

The tolerance for multiple signals can be set simultaneously by:

- If the *Signal Properties* dialog is open, close it.
- Left click the signal names in the signal window to select the signals to be edited. Notice that each signal can be selected individually.
- Right click one of the highlighted signal names and select **Edit Selected Signal(s)** from the pop up menu.
- Proceed with steps outlined above for editing signals.

Chapter 4: Editor Functions

VeriLogger's editor windows are an integrated part of the simulation environment. Double clicking in the *Project Tree*, *Errors*, or *Breakpoints* windows will open an editor and display the relevant source code. The editor windows are also used to display the current execution line for **single-step** debugging.

All editor windows provide color-syntax editing, search, single-click breakpoint placement, goto lines, and font control. The simulator automatically recognizes when a file is modified in an editor window, and will warn you when it needs to be saved.

4.1 Opening, Saving, and Creating New Source Code

Verilog source code files are opened and saved using the **Editor** menu options. When VeriLogger starts a new simulation, it checks for any unsaved files and automatically prompts you to save them.

To open an existing source code file use one of the following methods:

- Double click on the *filename* in the Project Tree control,
- OR, use the **Editor > Open HDL file** menu option.

To create a new source code file:

- Select the **Editor > New HDL file** menu option.

To save an open source code file:

- Select the **Editor > Save HDL file** menu option to open a *Save* dialog. By default, Verilog filenames have an extension of *v*.

To close the editor window and save the source code:

- Choose the **Editor > Close** menu option. If the file has been altered, you will be prompted to save the file.

4.2 Displaying or Finding a Specific Line of Code

Most VeriLogger display windows are linked directly to an editor window, making it easy to view relevant source code. Below is a list of windows and buttons that can be used to jump directly to a particular line of code.

- The *Errors* tab in the *Report* window displays compilation errors. Double click on an error to open an editor and display the line where the error was found.
- The *Breakpoints* tab in the *Report* window displays all the breakpoints in the current project. Double click on a breakpoint to open an editor and display the line where the breakpoint is located.
- The **Goto** button, the magnifying glass on the simulation bar, opens an editor starting at the last line executed. This button is only active during a simulation.
- The Project Tree control displays all signals, ports, and modules used in the project.

There are also several ways to search for line numbers or character strings in a file. Use the following keyboard combinations inside an active editor window to locate source code.

To move to a specific line in your code:

- Press <Ctrl> + **G** to open the *Jump To* dialog. Enter the line number to view.

To search for a character string:

- Press <Ctrl>+F to open the *Search* dialog. Enter the character string to locate.
- To perform another search, press the <F3> key.

4.3 Using the Editor Preferences Dialog

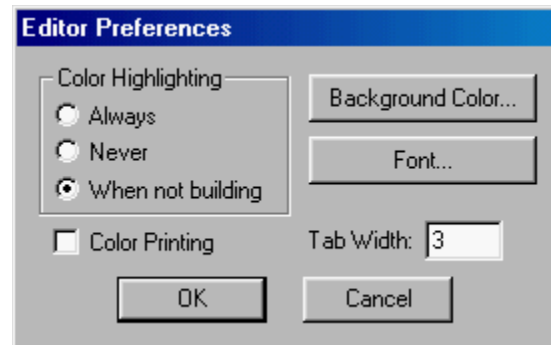
The *Editor Preferences* dialog controls the editor options. This information is stored inside the project **HPJ** file.

To open the *Editor Preferences* dialog:

- Select the **Editor > Editor Preferences** menu option.

There are several sections in the *Editor Preferences* dialog.

- The **Color Highlighting** radio buttons determine when color syntax editing is active. By default, the **When not building** option is selected so that the color syntax editing does not slow the build time of large projects.
- The **Color Printing** checkbox prints the source code in color. If unchecked, all code is printed in black.
- The **Background Color** button opens the Color dialog and lets you set the background color of the editor windows.
- The **Font** button opens the Font dialog. In the Font dialog you can set the font type, size, and color of the text in the edit windows.
- The **Tab Width** edit box sets the number of spaces that a tab character will generate. The default setting is 3 spaces, or it can be set to match the tab width of an external editor.



4.4 Editor Cursor Commands

The Report Window displays are full-featured editor windows. Listed below are the keyboard and mouse commands supported by the editor window.

<u>Key</u>	<u>Purpose</u>
Left/right arrow	Moves the cursor one space left or right
Up/down arrow	Moves the cursor one line up or down
Page Up	Moves the cursor one page up
Page Down	Moves the cursor one page down
Home	Move to the beginning of the current line
End	Move to the end of the current line
Ctrl+Up	Move to the beginning of the previous line

Ctrl+Down	Move to the beginning of the next line
Ctrl+Right	Move to the start of the next word
Ctrl+Left	Move to the start of the previous word
Ctrl+Page Up	Move to beginning of file
Ctrl+Page Down	Move to end of file
F1	Opens this help file
F4	Print from window
Shift+F4	Print options
F5	Search for a word or phrase
Ctrl+F5	Insert blank line before current line
F6	Replace
Shift+F9	Delete current line
Alt+J	Join lines (removes the next carriage return)
Alt+C	Block copy (duplicates selected text, and inserts it after selection)
Alt+Backspace	Undo
Alt+3	Protected text toggle (selected text cannot be modified while protected)
Alt+7	Change the color of the selected text
Ctrl+T	Tab
Ctrl+A	Select all
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Ctrl+Z	Undo
Ctrl+Y	Redo
Ctrl+F	Search
Ctrl+G	Jump to line#

4.5 Using an External Editor

External editors can be used with VeriLogger. If you use an external editor, make sure it is configured to detect when other programs externally modify a file. While simulating and debugging in VeriLogger, you will want to use the internal editors to make quick fixes to the code so you can continue simulating. If your editor does not detect that you have modified a file, it may overwrite your fixes.

Chapter 5: Waveforms and Test Bench Generation

VeriLogger's Diagram window is used to display simulation results and graphically generate stimulus vectors. This chapter describes the different techniques used to generate automatic test benches and how simulation data is displayed or stored.

5.1 Grouping Waveforms for Different Simulation Runs

Instead of watching your entire design each time you simulate, VeriLogger lets you create different sets of watched signals. Timing diagram files (files with a .tim extension) can be used to group waveforms together for different simulation runs. Watching small sections of your design makes it easier to detect bugs and speeds up the simulation time.

To save a group of signals in one file:

- In the Diagram window, arrange the signals that you want to group together.
- Select the **File > Save As** menu option to open a dialog of the same name.
- Enter a filename with a .tim extension and press **OK**.

To load a set of saved signals:

- Verify that the project is open and ready to simulate.
- Choose the **File > Open Timing Diagram** menu option to open a dialog of the same name.
- Select your file and press the **Open** button to load the waveforms.
- Your next simulation will display the values of the watched signals.

5.2 Automatic Test Bench Generation and Project Tree Options

After you build the project, the signals or the ports in the top-level module are automatically added to the Diagram window. If the top-level module does not have port signals, the internal signals are viewed. If the top-level module has port signals, the output ports are displayed as purple signals and input ports are displayed as black signals. If the top-level module has ports, the project automatically wraps a test bench around the top-level module and creates signals in that test bench to drive and watch the top-level module. This makes it easy to quickly test small parts of a design before the design is complete.

The black input signals provide stimulus to the top-level module. Input signals can be graphically drawn, generated by equations, or copied from existing signals. This ability to draw waveform stimulus and immediately perform a simulation is one of the unique features of VeriLogger. See the SynaptiCAD user's manual or the on-line help for complete information on the waveform drawing and editing functions.

If your simulation is very large, it can become constrained by the memory limitations of your computer. If this occurs, you can reduce VeriLogger's memory requirements by turning off the automatic test bench generation and project tree control options.

To turn off the automatic generation of the test bench:

- If user-generated signals provide stimulus for the simulation, you must first manually create a test bench model, then add the model to the project. Use the **Export > Export Signals As** menu option to manually create a stimulus model. Complete information on test bench generation is found in *Chapter 13.2: Export of VHDL and Verilog Stimulus* of the SynaptiCAD user's manual.

- Next, open the *Simulation Preferences* dialog by using the **Options > Simulation Preferences** menu option.
- Uncheck the **Auto-create test bench and tree** check box to disable automatic test bench and project tree generation.

5.3 Drawing Waveforms for Stimulus Generation

In the **Sim Diagram & Project** mode, VeriLogger can generate Verilog stimulus code for signals drawn in the diagram and use that code as inputs to the project. After you build the project, the signals or the ports in the top-level module are automatically added to the Diagram window. Any *input* ports are colored black and can be drawn on using the timing diagram editing features as described in the SynaptiCAD on-line help and user's manual: *Chapter 1 Signals and Waveforms*.

Internal Signals: VeriLogger can also generate stimulus for internal signal nodes. To do this just add the signal (using the full hierarchical name) to the Diagram window. A quick way to get hierarchical signal name is to use the Project window like this:

- In the Project window, right click on the signal name and choose Watch menu option to add the name of the signal to the diagram window. This adds the full signal name to the Diagram window but it sets this signal up as a purple watch signal.
- In the Diagram window, double click on the signal name to open the Signal Properties dialog and choose the Drive radio button to make the signal black.

Inout Signals: VeriLogger can graphically generate stimulus for inout ports and simultaneously watch the port's simulated output using another signal with the same name. To do this, add two signals with the same full hierarchical name. Make one signal a **watch** signal and the other a **drive** signal. Remember to draw tri-state values on the drive signal when that signal should not be driving the inout port.

5.4 Generating VCD files

The VCD format is a standard Verilog file format that can be used with external waveform viewers, static timing analyzers, or VeriLogger's graphical display. Watched signals in VeriLogger are displayed graphically, and by default are NOT dumped to a VCD file. Two check boxes in the *Project Preferences* dialog control the output of data gathered by watched signals.

To determine the output of watched signals:

- Select the **Project > Project Preferences** dialog to open a dialog of the same name.
- Check the **Capture and Show watched signals** checkbox to view watched signals in the Diagram window.
- Check the **Dump watched signals** checkbox to return data from watched signals to a VCD file.

It is less memory intensive to dump files than it is to actively view the data in the Diagram window. To speed up large simulations, turn off the waveform display and dump the watched signals. After the simulation, use the **Export > Import Signals As** menu option to view the resulting VCD file.

VCD files can also be generated by using the Verilog system tasks **\$dumpvars**, **\$dumpfile**, **\$dumpall**, **\$dumpon**, and **\$dumpoff** to save waveform data. See the **Verilog Language Overview** for more information on the syntax of these statements.

5.5 Reading VCD files

VeriLogger Pro can read VCD files created by other programs or simulation runs.

- Select the **Export > Import Signals As** menu option. This *Open* dialog is special in that it remembers the file type of the last file imported.

- Enter the VCD file you wish to open in the **File name:** edit box.
- Push the **Open** button to load the file. The waveforms are now visible in the Diagram window.

5.6 Working in the Diagram Window

The Diagram window in VeriLogger is the same timing diagram editor that is the basis of SynaptiCAD's WaveFormer Pro and Timing Diagrammer Pro products. Because of this, VeriLogger users have access to all timing diagram editor features. A comprehensive listing of the drawing and editing features can be found in the WaveFormer Pro on-line help.

There are several features that are particularly useful for viewing lengthy simulated signals. We have listed them here for your convenience.

Viewing all signal values

- Left click down in the time line in the Diagram window. This displays a marker line that shows the value of each signal at that particular time.

Zooming in the Drawing window

- **Click-and-drag** in the time line to zoom. Left click and hold inside the time line and drag the mouse to indicate the range in which to zoom. When you release the mouse, the diagram will zoom to show the range you selected. This provides a quick way to graphically specify the zoom level and range for a section in a large timing diagram.
- The **Zoom In** and **Zoom Out** buttons, located on the right of the button bar, change the zoom level in the timing diagram.
- The **Zoom Range** button opens a dialog that lets you specify the starting and ending times displayed in the Diagram window.
- The **Zoom Full** button displays the entire timing diagram on the screen.

Scrolling to a specific time or offset position:

- The two buttons directly above the signal label window provide an absolute time readout and a relative time readout. The **Time Button**, with the black writing, displays the current position of the mouse cursor in the drawing window. The **Delta Button**, with the blue writing, displays the difference between the mouse cursor and the delta mark (an upside-down, blue triangle) on the timeline above the drawing window. These buttons can also be used for quick scrolling in very long timing diagrams.
- Left clicking on either button opens an edit box that accepts time values.
- Entering a value in the **Time** button causes the drawing window to scroll to that exact time.
- Entering a value in the **Delta** button causes the drawing window to scroll that amount from its current position.

Chapter 6: Faster Simulations with the Command Line Simulator

Most of this manual is dedicated to the fast model development and testing features of VeriLogger's graphical user interface. As your project nears completion, you might want to trade ease-of-use for faster simulation times. For this reason VeriLogger ships with two simulators: the graphical simulator and the command line simulator. The command line simulator is streamlined for speed and simulates about twice as fast as the graphical simulator.

6.1 Preparing Verilog Source Files

Before using the command line simulator you may want to add statements to the Verilog source code to generate simulation display statements. Signals that were watched in the graphical simulator will not automatically generate output in the command line simulator.

There are several Verilog statements that will generate output:

- The **\$monitor** system task is used to continuously monitor a signal and produce an output message every time the signal changes.

```
$monitor("Counter = %d", count);
```

- The **\$display** system task is used to print text messages and look at values on signals. The \$display statements write the results to the **verilog.log** file. This statement is similar to a debug statement used to debug program flow in a standard programming language. See the **Verilog Language Overview** for more information on the syntax. An example of a display statement used inside a module is:

```
$display("Counter = %d", count);
```

- The **\$dumpvars**, **\$dumpfile**, **\$dumpall**, **\$dumpon**, and **\$dumpoff** system tasks are used to save waveform data in to a value change dump (VCD) file. The VCD format is a standard Verilog file format that can be used with external waveform viewers, static timing analyzers, or VeriLogger's graphical display. See the **Verilog Language Overview** for more information on the syntax of these statements.

6.2 Using the Command Line Simulator

To run the command line simulator:

- Open a command line window on your operating system. Windows users should open a DOS prompt.
- Navigate to the VeriLogger directory.
- Next, invoke the command line simulator with one or more source files and any desired simulation options. The following example starts the simulator, and executes the source file *model.v*:

```
vlogcmd model.v
```

If there is more than one file, then each file needs to be specified on the command line. The order that the files are entered in the command line is the order in which they are compiled. In most cases the order is irrelevant, but there are some cases where it is significant, particularly when using the same macros (**define**) across files.

```
vlogcmd cpu.v memory.v io.v
```

To avoid retyping the same source files and simulation options every time you perform a simulation, you can create a command file. A command file is a simple text file that contains a list of source files and simulation options used in

the simulation. To call a command file, use the **-f** simulation option (all simulation options are listed in section 6.3) followed by the name of the command file. The use of a command file is demonstrated below:

```
vlogcmd -f command.vc
```

Command files are user-created text files with a ***.vc** file extension. They consist of Verilog source files, simulator options, and other command files. When creating a command file, list only one file or simulation option per line. The following is an example of a command file with three Verilog source files and two simulation options:

```
cpu.v
memory.v
io.v
-s
-t
```

To automatically generate a command file that contains all project settings project files:

- Choose the **Project > Project Settings** menu option to open the *Project Settings* dialog.
- Press the **Generate Command File** button. This takes all the project commands and file names contained in the **Command Line Options** edit box and creates a command file.

6.3 Command Line Simulation Options

The VeriLogger command line simulator supports several simulation options that can be used to control and debug simulations. The simulation options may be displayed in any order and anywhere on the command line. To the simulator, the following statements are identical:

```
vlogcmd -t -s cpu.v memory.v io.v
vlogcmd cpu.v memory.v io.v -s -t
vlogcmd cpu.v -t memory.v -s io.v
```

Listed below are the simulation options supported by VeriLogger:

- Use a command file, **-f <command filename>** runs the simulator with the designated command file. All of the following simulation options can be used in a command file.

```
vlogcmd -f commfile.vc
```

- Stop, **-s** compiles the source code then enters the interactive mode before the execution begins.

```
vlogcmd -s cpu.v memory.v io.v
```

- Trace, **-t** enables a tracing mode that returns a trace history of each line executed into the log file.

```
vlogcmd -t cpu.v memory.v io.v
```

- Compile only, **-c** compiles the source code and exits without performing a simulation.

```
vlogcmd -c cpu.v memory.v io.v
```

- Key filename, **-k <key filename>** changes the name of the key file that contains a log of all keystrokes entered during the simulation run. By default, the key file is called **verilog.key**.

```
vlogcmd -k mykey.key -c cpu.v memory.v io.v
```

- No Key, **-k nokey** disables the key file.

```
vlogcmd -k nokey -c cpu.v memory.v io.v
```

- Log filename, **-l <log filename>** changes the name of the log file that contains all output generated during a simulation. By default, the log file is called **verilog.log**.

```
vlogcmd -l mylog.log -c cpu.v memory.v io.v
```

- **No log**, **-l nolog** disables the log file.

```
vlogcmd -l nolog -c cpu.v memory.v io.v
```

- **Library filename**, **-v <filename>** specifies the name of a library file. If this option is used, VeriLogger will try to match any undefined modules to modules inside the library files.
- **Library directory**, **-y <directory>** specifies the directory path where searches for library files are made. If this option is used, the simulator will attempt to match any undefined modules with files that have one of the file extensions set with the **+libext** option. The simulator does not look inside a file unless the undefined module name exactly matches the filename. The simulator will not look at any files unless file extensions have been set using the **+libext** option. The following examples show how to specify a directory path, a directory path with spaces, and how to use the **+libext** option (UNIX users should use a backslash):

```
vlogcmd model.v -y\mylibs +libext+.v
vlogcmd model.v -y"\My Libraries" +libext+.v
```

- **Interactive Command filename**, **-i <filename.vi>** allows the simulator to accept interactive commands from a file. Any legal interactive mode command can be included in the interactive command file. The file is submitted to the simulator before the simulation begins and starts to execute as soon as the simulator enters an interactive simulation mode. Therefore the **-i** command must be paired with a statement that stops the simulator and enters the interactive mode. There are two ways to do this:

- Use the **-s** option to stop VeriLogger and enter the interactive mode before execution begins,
- OR, embed the **\$stop** system task into a Verilog source code file and use it with a delay to stop the system at a later time. For example, assume the file **cpu.v** contains the following code fragment to stop the system 1000 time units after the simulation begins:

```
#1000 $stop;
```

This file is submitted with the following command:

```
vlogcmd -i interactive.vi cpu.v memory.v io.v
```

6.4 Predefined Plus Options

The VeriLogger simulator supports the following Verilog run time simulation options:

- +maxdelays | +mindelays | +typdelays** determines which delay used in the **min:typ:max** expressions. In the graphical simulator this command is set using the **Project > Project Preferences** menu option. In the command line simulator add the option to the command line:

```
vlogcmd cpu.v memory.v io.v +mindelays
```

- +define+<macro name>+<macro name> ...** defines macro names from the command line, generally for use with conditional compilation directives. Any number of macros can be defined by adding another **+<macro name>** to the list. For example, the *count.v* Verilog source code file had the following code fragment:

```
'ifdef EXTEND
    $display("Using extended mode");
'else
    $display("Using normal mode");
```

Then the following command will execute the first **display** statement:

```
vlogcmd count.v +define+EXTEND
```

- +synopsys** displays warnings for constructs that are either not supported or ignored by the Synopsys HDL Compiler.
- +noshow_var_change** disables the tracking of variable changes. By default, VeriLogger keeps track of the location and simulation time where variables are last written. This information can be displayed using the **\$show-vars** directive. This feature may cause a slight performance degradation, so it can be disabled with this option.

+libext+<ext>+<ext> ... specifies the filename extension used when searching for libraries in the library directory. This is most often used with the **-y** option. The following example will search the directory `\design\libs` for libraries whose filename ends with `.vl` and `.vv`:

```
vlogcmd cpu.v -y \design\libs +libext+.vl+.vv
```

+incdir+<directory1>+<directory2>+... specifies the directories that VeriLogger will search for included files. All the characters between the pluses are used in the directory name.

```
vlogcmd cpu.v +incdir+\design\project1+ -y \design\libs +libext+.vl+.vv
```

6.5 Simulator Control Commands

In addition to the Verilog commands, there are also several VeriLogger specific commands that can be used to control the simulation:

- To *continue* the simulation, type the period (.) character.
- To *step* to the next statement in the code, type the semicolon (;) character.
- To *step-and-trace* to step to the next statement in the code and generate a trace message in the **verilog.log** file type the comma (,) character.
- To *display the current code line* execution, type the colon (:) character.
- To *terminate* the simulation, type the **\$finish;** command, or press **<Ctrl>+C**.

Chapter 7: Back-Annotated Simulation (Gate-Level Timing Simulation)

In the initial stages of your design you will be performing "functional" analysis simulations to ensure that the logic in your circuit operates correctly. After your FPGA or ASIC tools generate a layout for your gate-level design, you may want to perform a final simulation with back-annotated timing information generated during the layout process to account for real world interconnect and gate delays. This "timing" simulation is often used as a final check to ensure that unexpected delays generated during the layout process don't create timing violations in your design. The layout tools will create a Standard Delay File (SDF) that includes this timing information. By including this timing information, the model can be tested based upon these propagation delays. This chapter will tell you how VeriLogger can be used to include the information in the SDF in your model. Note: This type of timing simulation is often unnecessary if you use a static timing analysis tool to verify the critical paths in your design meet the timing constraints of your design.

7.1 Using the Standard Delay File (SDF)

An SDF can be produced for any module in the hierarchy of your project. For example, if you are modeling a board-level design that contains an FPGA, your FPGA tools will probably produce an SDF file for the laid out gate level module of the FPGA. To include the timing information from this file in your design, add an `$sdf_annotate` command in the FPGA module whose timing is to be modified. Include the bold lines of code in the example FPGA module shown below to tell the simulator to read the SDF timing information:

```

module MyFPGA(ports...)
  // port declarations....
  initial
    begin
      $sdf_annotate("mydesign.sdf");
    end
  // ...other code...
endmodule

```

Note that if you have an `initial` block already in the module to be annotated, you can include the `$sdf_annotate` line in the existing block. Also note that "mydesign.sdf" shown above should be replaced with the filename your tool generated for the SDF. The file extension `.sdf` should be used.

Chapter 8: VeriLogger Technology Background

VeriLogger is an interpreted simulator. This means that when VeriLogger starts, it reads the source models, compiles them into internal data structures, and then executes them from these structures. The structures contain enough information that the source can be reproduced from the structures (minus formatting and comments.)

While the model is running, the simulation can be interrupted at any time by pressing <Ctrl>+C. This puts the simulator in an interactive command mode. From here VeriLogger commands and Verilog statements can be entered to debug, modify, or control the course of the simulation.

8.1 Compilation Process

VeriLogger uses a three-phase compilation process:

Phase 1: The files are read and converted into an internal data structure. Syntax errors and semantic errors regarding undeclared variables or illegal use of variables are reported in this phase.

Phase 2: In this phase the model hierarchy is built, module ports are connected, and storage for variables is allocated. If any module is instantiated more than once, its structure is copied as many times as needed. Also, module parameters are propagated. Errors reported in this phase deal with missing modules, irregularities of the parameters, and out-of-memory errors during the allocation. The project tree is built during this phase.

Phase 3: The entire structure is re-parsed during which time-forward references to tasks and functions are resolved, hierarchical names are resolved, and expression sizes are determined. Errors detected in this phase include semantic errors dealing with hierarchical references that could not be detected in Phase 1, illegal references to functions and tasks, port size discrepancies, and illegal expression sizes.

Note: Most memory is allocated in the first two phases of the compilation.

8.2 User-Defined Primitives and Memory Usage

User-Defined Primitives (UDPs) are used to define combinatorial primitives and two-state devices. In VeriLogger, UDPs are optimized for performance. This is accomplished by creating a table in memory for each UDP definition. Only one such table is used for each UDP definition; every instance of the definition uses the same table. When there are more than six inputs, the size of the table is very large. For this reason, the maximum number of inputs is ten (nine for state UDPs). The maximum table size is approximately 256K.

8.3 Notes on Using Specify Blocks

Specify blocks are used to define pin-to-pin timings and setup-and-hold checks. In VeriLogger, specify blocks function similarly to those described in the Verilog Language Reference Manual. However, there are some differences. In VeriLogger, there is no concept of expanded nets. Nets that are defined as vectors are not split into individual nets and cannot have their own timing information. Therefore, certain combinations of timing specifications will be ignored. Specifically, there are two ways to describe module paths. One is the parallel case ($=>$), and the other is the full case ($*>$). In VeriLogger, both cases are treated as if they were defined as the parallel case. This does not pertain to scalar nets. VeriLogger supports all of the defined setup and hold systems tasks.

Appendix A: VeriLogger Implementation Notes

Except for the following discrepancies, VeriLogger behaves exactly as specified by the IEEE-1364 LRM and Verilog-XL standards.

A.1 Port Collapsing

In certain versions of Verilog, if two nets are connected together via a port, the port is **collapsed** (combined to form one net). In VeriLogger, module ports are connected using transparent continuous assignments. If a register is connected to a net, then the port propagation does not occur immediately after the port changes. Instead, the port propagation is scheduled for later in the same simulation time. However, when a net is connected to a net, then a collapsed port is emulated by forcing the propagation to occur instantly. The effect of this implementation does not affect the functionality of the model being simulated, but becomes visible during trace.

A.2 Port Connections of Different Net Types

VeriLogger does not check the legality of the connections of different net types in the hierarchy. For example, if a parent module instantiates a child module, and the net on the parent's side of a port is a **tri1** while the net on the child's side of a port is a **tri0**, an oscillation will result. To use **tri1**, **tri0**, **triand**, and **trior** as ports effectively in VeriLogger, they should be declared only in the top-most level in the hierarchy. All lower-level connections should be declared as **wire** or **tri**.

A.3 Working Around Pullup/Pulldown Gates

When modeling an open-collector bus, a common technique is to have a **pullup** or **pulldown** gate drive a **wire** net and have drivers pull the bus in the opposite direction with a greater strength when asserting a signal. In VeriLogger, drive strengths are not implemented. Therefore, this technique will generate an unknown value (X) when a driver attempts to drive a signal in the opposite direction as the pull. The preferred method for modeling open-collector buses is to use the **triand** nets for pullup buses, and the **trior** nets for pulldown buses. This net type should only appear in the highest level of the hierarchy in which the bus exists.

A.4 Using Trace Implementation

Trace is an indispensable tool for debugging Verilog programs. It displays each statement as it is executed, as well as any results returned from the statement. There are three ways to enable a trace:

- 1) Specify the **-t** option at the command line.
- 2) Execute the system task **\$settrace** from either the program or the interactive command line.
- 3) Execute and trace a single statement by entering a comma (,) at the interactive command line. Enter multiple commas to execute the respective number of statements.

If a model uses continuous assignments or ports, VeriLogger displays the activation of these as part of the trace as soon as the activation occurs. For example, given the continuous assignment **assign test = bar;** when **bar** changes, the continuous assignment is executed immediately and displayed in the trace. The continuous assignment represents one of possibly many drivers to the net **test**; the net itself is scheduled for updating for sometime later in the current simulation time unit.

Because port connections are implemented as continuous assignments it may take several steps for a signal to propagate from an output port to an input port, especially in cases where there are several ports connected to a net. Trace shows part of this propagation. A signal emanating from an output port travels upward to its parent module; it then travels back down to other connected ports. Each time a signal reaches a new port, the net connected to that port is evaluated and the results are displayed in the trace.

A.5 Predefined Macro `__VERIWELL__`

The macro, `__VERIWELL__`, is predefined so that statements such as: `‘ifdef __VERIWELL__` are used for VeriLogger-specific code, such as for waveform display.

A.6 Simulation Statistics

The non-standard system task, `$showstats`, displays statistics about the current simulation, including the amount of memory used and available. Some of the information is provided for diagnostic purposes only.

A.7 Displaying the Location of the Last Value Edited

The `$showvars` system task optionally displays the current location in the module and the simulation time at which the module variables were last changed. This information is updated even if the value did not change (that is, the new data is the same as the old data). Tracking this update information may affect the performance of the simulation slightly. If this is a problem, this feature can be disabled with the `+noshow_var_change` command line option.

A.8 User Interrupt

Pressing `<Ctrl>+C`, `COMMAND+C`, or `<Ctrl>+BREAK` (in MS-DOS) during simulation will put the command line version of VeriLogger into interactive mode. Pressing any of these during compilation will halt the process and exit to the operating system.

Appendix B: Implementation Differences from Verilog-XL™

This appendix describes the differences between VeriLogger and Verilog-XL™. Note that these differences are subtle and will not affect the execution of well-written Verilog models.

B.1 Event Ordering

The order of event scheduling and execution is consistent with Verilog-XL™ in every extent possible. The reason for doing this is not so that models are guaranteed to work under both VeriLogger and Verilog-XL™ but rather because VeriLogger was designed so that users can trace models in it and in Verilog-XL™ with little noticeable difference. However, it should be noted that models that depend on the order of execution are considered to be unwisely written because they reflect race conditions and may perform unpredictably in other vendors' Verilog, or even in future releases of VeriLogger (or Verilog-XL™). In some cases, the order of net scheduling may be different. This is because Verilog-XL™ schedules nets differently depending on the type of net, whether it is sourced by a continuing assignment, a net assignment, or a port, and whether a port is collapsed. In most cases, net scheduling will track that of Verilog-XL™.

B.2 Module Ports and Port Collapsing

Port connections are implemented as continuous assignments in VeriLogger. Rules for port connections are similar to those of Verilog-XL, but there are some differences. In Verilog-XL, under certain circumstances, ports are **collapsed**, that is, if each side is a net, then one of the nets disappears and only one is used. This is a performance enhancement. VeriLogger emulates port collapsing by immediately propagating values across ports that have been **collapsed**. This is unlike Verilog-XL, which actually combines nets that have been collapsed. Verilog-XL will expand vector nets into arrays of scalar nets if a port connects two different sized nets, or if one or both sides are concatenations or part selects. VeriLogger does not implement expansion of nets, so it could not handle these cases by building continuous assignments. VeriLogger will **collapse** a port if both sides of a port are scalar nets or if both sides are vector nets. Therefore, there are some cases when VeriLogger will not collapse a port, but where Verilog-XL will. This may cause a disparity in the way nets are scheduled in the two simulators.

B.3 Control Expressions are Limited to 32 Bits

Expressions used by VeriLogger for control are limited to 32 bits. This includes repeat counts, delay values, part- and bit-select and array index expressions, and shift counts. A compile-time error will result if the expression attempts to evaluate a number greater than 32 bits.

B.4 The \$Monitor System Task

Unlike Verilog-XL, the \$monitor system task will be triggered if any variable in the argument list changes. In Verilog-XL, \$monitor changes only when an argument expression changes. For example, in Verilog-XL the following statement will not be triggered if both a and b change, and the sum stays the same. In VeriLogger, the statement will be triggered if both a and b change in this case.

```
$monitor (a + b);
```

- Verify that only one file, **add4.v**, is listed on the project tree, and that the Diagram window is empty.

Appendix C: License Agreement

SynaptiCAD

TestBench Pro - DataSheet Pro - WaveFormer Pro - Timing Diagrammer Pro - VeriLogger Pro
Software License Agreement

- Read Before Use -

Please read and understand this license.

Note: Throughout this agreement, the word Software refers to the software product that you have licensed from SynaptiCAD.

You have purchased a license to use one of the following products: **TestBench Pro, DataSheet Pro, WaveFormer Pro, VeriLogger Pro, or Timing Diagrammer Pro** software. The software is owned and remains the property of SynaptiCAD, is protected by international copyrights, and is transferred to the original purchaser and any subsequent owner of the Software media for their use only on the license terms set forth below. Opening the packaging for **TestBench Pro, DataSheet Pro, WaveFormer Pro, VeriLogger Pro, or Timing Diagrammer Pro** and/or using either **TestBench Pro, DataSheet Pro, WaveFormer Pro, VeriLogger Pro, or Timing Diagrammer Pro** indicates your acceptance of these terms. If you do not agree to all of the terms and conditions, return the unopened Software and manuals immediately for a full refund.

Use of the Software

- **SynaptiCAD** grants the original purchaser ("Licensee") the limited rights to possess and use the Software and User's Manual ("Software") for its intended purpose. Licensee agrees that the Software will be used solely for Licensee's internal purposes and that the Software will be installed on a single computer only. If the Software is installed on a networked system, or on a computer connected to a file server or other system that physically allows shared access to the Software, Licensee agrees to provide technical or procedural methods to prevent use of the Software by more than one user.
- One machine-readable copy of the Software may be made for BACKUP PURPOSES ONLY, and the copy shall display all proprietary notices, and be labeled externally to show that the backup copy is the property of SynaptiCAD, and that use is subject to this License.
- Use of the Software by any department, agency or other entity of the U.S. Federal Government is limited by the terms of the below "Rider for Governmental Entity Users."
- Licensee may transfer its rights under this License, provided that the party to whom such rights are transferred agrees to the terms and conditions of this License, and written notice is provided to SynaptiCAD. Upon such transfer, Licensee must transfer or destroy all copies of the Software.
- Except as expressly provided in this License, Licensee may not modify, reverse engineer, decompile, disassemble, distribute, sub-license, sell, rent, lease, give or in any other way transfer, by any means or in any medium, including telecommunications, the Software. Licensee will use its best efforts and take all reasonable steps to protect the Software from unauthorized use, copying or dissemination, and will maintain all proprietary notices intact.

LIMITED WARRANTY SynaptiCAD warrants the Software media to be free of defects in workmanship for a period of ninety days from the purchase. During this period, SynaptiCAD will replace at no cost any such media returned to SynaptiCAD, postage prepaid. This service is SynaptiCAD's sole liability under this warranty.

DISCLAIMER LICENSE FEES FOR THE SOFTWARE DO NOT INCLUDE ANY CONSIDERATION FOR ASSUMPTION OF RISK BY SYNAPTICAD, AND SYNAPTICAD DISCLAIMS ANY AND ALL LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR OPERATION OR INABILITY TO USE THE SOFTWARE, EVEN IF ANY OF THESE PARTIES HAVE BEEN ADVISED OF THE POSSIBILITIES OF SUCH DAMAGES. FURTHERMORE, LICENSEE IN-

DEMNIFIES AND AGREES TO HOLD SYNAPTICAD HARMLESS FROM SUCH CLAIMS. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY THE LICENSEE. THE WARRANTIES EXPRESSED IN THIS LICENSE ARE THE ONLY WARRANTIES MADE BY SYNAPTICAD AND ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE. THIS WARRANTY GIVES YOU SPECIFIED LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO JURISDICTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF WARRANTIES, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Term

- This license is effective as of the time Licensee receives the Software, and shall continue in effect until Licensee ceases all use of the Software and returns or destroys all copies thereof, or until automatically terminated upon failure of Licensee to comply with any of the terms of this License.

General

- This License is the complete and exclusive statement of the parties' agreement. Should any provision of this license be held to be invalid by any court of competent jurisdiction, that provision will be enforced to the extent permissible, and the remainder of the License shall nonetheless remain in full force and effect. This License shall be controlled by the laws of the State of Virginia, and the United States of America.

Rider For U.S. Governmental Entity Users

This is a rider to **TestBench Pro/ DataSheet Pro/ VeriLogger Pro/ WaveFormer Pro/ Timing Diagrammer Pro** SOFTWARE LICENSE AGREEMENT ("License") and shall take precedence over the License where a conflict occurs.

1. The Software was: developed at private expense (no portion was developed with government funds) and is a trade secret of SynaptiCAD and its licensor for all purposes of the Freedom of Information Act; is "commercial computer software" subject to limited utilization as provided in any contract between the vendor and the government entity; and in all respects is proprietary data belonging solely to SynaptiCAD and its licensor.
2. For units of the DoD, the Software is sold only with "Restricted Rights" as that term is defined in the DoD Supplement to DFAR 252.227-7013 (b)(3)(ii), and use, duplication or disclosure is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Manufacturer: SynaptiCAD, PO Box 10608, Blacksburg, Va 24062-0608 USA.
3. If the Software was acquired under a GSA Schedule, the Government has agreed to refrain from changing or removing any insignia or lettering from the Software or Documentation or from producing copies of manuals or disks (except for backup purposes) and: (1) Title to and ownership of the Software and Documentation and any reproductions thereof shall remain with SynaptiCAD and its licensor; (2) use of the Software shall be limited to the facility for which it is acquired; and (3) if the use of the Software is discontinued at the original installation location and the Government wishes to use it at another location, it may do so by giving prior written notice to SynaptiCAD, specifying the new location site and class of computer.
4. Government personnel using the Software, other than under the DoD contract or GSA Schedule, are hereby on notice that use of the Software is subject to restrictions that are the same or similar to those specified above.

VeriLogger Tutorial A: Basic Verilog Simulation

This tutorial demonstrates the basic simulation features of VeriLogger Pro. It teaches you how to create and manage a project and how to build, simulate, and debug your design. It also demonstrates the graphical test bench generation features that are unique to VeriLogger Pro. This is a stand alone tutorial which you should be able to complete without reading any of the other tutorials. However, if you plan to make extensive use of the graphical stimulus generation features then you may also want to perform the *Basic Drawing and Timing Analysis* tutorial and Section 2 of the *Simulation, Waveform Generation, and Parameters* tutorial, which cover the time-saving features of the timing diagram editor.

In this tutorial, you will compile and simulate a 4-bit adder and a test bench module contained in files `add4.v` and `add4test.v`. Figure 1 shows a schematic of the circuit. Later in the tutorial you learn to graphically enter the stimulus vectors instead of using a test bench module. Also you will get to practice using the basic debugging features of breakpoints, single stepping, and viewing different signals in the file.

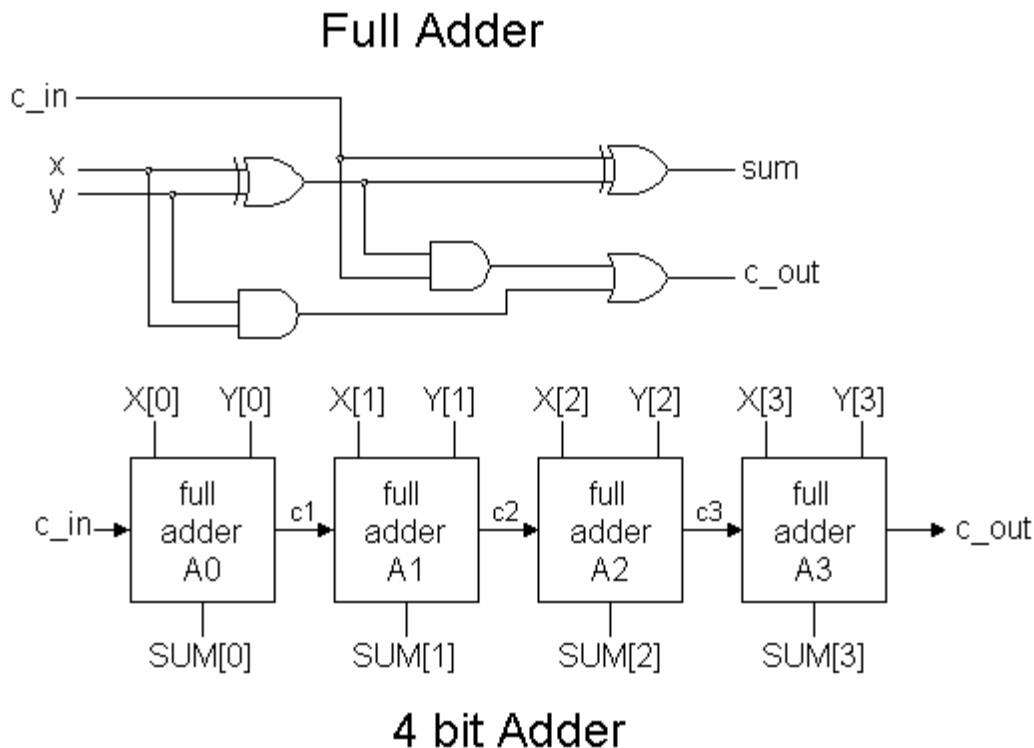


Figure 1: Schematic of the 4-bit adder simulated in this tutorial.

Part 1: Project Management and Simulation

In this section, you will create, build, and simulate a project. VeriLogger uses a project to control all aspects of simulation and design including specifying the files to be simulated, controlling simulation options, and setting watches on signals. The project also stores the hierarchical structure of the Verilog components contained in the design and displays this information on the tree control in the *Project* window.


1.1) Add Files to the Project

The first step in creating a project is to add Verilog files to the project tree window and save the project. To add a file to the project:

- Right click in the *Project* window to open the context menu and choose the **Add File(s)** menu option. This opens the *Add Files* dialog.
- Select the **add4.v** and **add4test.v** files located in the VeriLogger Pro install directory. To select multiple files at the same time, select the first file then hold down the <CTRL> key while using the mouse to select any additional files.
- Press the **Open** button to add the files to the project. Both filenames should be visible on the project tree. If you do not see both files then repeat the instructions and add the missing file to the project.

1.2) Build the Tree and use the Editor Windows

In this section we will build the project tree and use the Editor windows to view the source code. When files are first added to the project, you can see the filename but you cannot see a hierarchical view of the modules inside the files. To view the internal modules on the project tree you must first build or run a simulation. The build command compiles the Verilog files and builds the Verilog tree. It does not run a simulation. For large projects build lets you quickly construct the tree without having to wait for a simulation to run. To build a project:

- Press the yellow **Build** button  on the simulation button bar.
- Notice that the modules inside each source file are displayed on the project tree. One module, **testbed**, is surrounded by brackets to indicate that it is the top-level module (the highest-level instantiated component).
- Notice that the Diagram window now has several signals listed in the waveform display. By default, VeriLogger sets watches on all the top level modules internal signals. Later we will learn to set watches on the other signals.

All sub-modules can be viewed by descending the top-level module's tree. When the tree is expanded it can display the signals, ports, and components contained in each module. Expand the tree by using the + buttons or by using the node context menu **Expand Item**. Try both methods for expanding the tree:

- Press the + button to the left of <<<testbed>>>   <<< testbed >>> to expand the project tree. Notice that the sub-nodes of *Signals* and *Components* are not expanded.
- Right click on <<<testbed>>> and choose **Expand Item** from the context menu. This expands the node and all the sub-nodes.

Double clicking on a particular module or node in the Project Tree causes an editor to open and display the relevant source code. Let's view some source code:

- Double click on the **add4test.v** filename to open an editor loaded with that file. Notice that the editor is scrolled to the top of the file.
- Double click on the **fa0** component located down on the expanded branches of the <<<testbed>>> tree. You may have to scroll down to find this module. Notice that editor is now scrolled to the instantiation of **fa0** and there is a white arrow to the left of the editor screen indicating the correct line.


The editors can be used to edit and write Verilog source code. Most of the Editor window functions are accessed from the **Editor** menu tree.

- Left click on the **Editor** menu to open the menu. Notice the **Save HDL File**, **Open HDL File**, and **Editor Preference** menu options. These are the more commonly used menu options.

Chapter 4: Editor Window in the on-line VeriLogger help manual has more information on using the Editor windows.

1.3) Simulate the Project

When we built the project in the last section, the names of the internal signals in the top-level module were automatically added to the Diagram window. This feature allows you to quickly set up a project and start simulating and debugging without having to stop and specify a set of signals. For large projects you may want to turn off this feature by choosing the **Project > Project Settings** menu and un-checking the **Grab top level signals** check box. For small projects the automatic signal watches save a lot of time so we will leave it on for the tutorial. First, let's simulate with the default signals:

- Press the green **Run** button  on the simulation button bar. This causes a simulation to start and run until the end of the simulation time or until a breakpoint is reached. The Diagram window should contain purple waveforms.
- Verify that the **sum** and **c_out** are correctly being computed as $x + y + c_in$.

1.4) Watch and View Internal Signals

With VeriLogger you can watch any combination of signals listed under the top-level module tree. To demonstrate this we will set watches on the sum outputs for the *full adders* sub-modules that make up the 4-bit adder:

- In the Project window, expand the top-level module tree and find the **fa0** component.
- Right click on the **sum** port for **fa0** to open a context menu.
- Choose the **Watch Connection** menu option. This adds the **testbed.A1.fa0.sum** signal to the Diagram window.
- Press the green **Run** button to run another simulation. Verify that the **testbed.A1.fa0.sum** signal is the 0 bit of the **testbed.sum[3:0]** signal.

To remove signals from the watch list delete them from the Diagram window:

- In the Diagram window, left click on the **testbed.A1.fa0.sum** signal name to highlight it.
- Press the **Delete** key on the keyboard to remove the signal from the Diagram window.
- Press the green **Run** button to run another simulation. Verify that the **testbed.A1.fa0.sum** signal was not watched.

Next we will experiment with different ways to view waveforms in the Diagram window:

- In the time line above the signals in the Drawing window, left click down and hold to show a marker that displays the value of each signal. Release the mouse button without dragging.
- Left click and drag the marker about 50ns in the time line window. When you release the mouse button, the window will zoom to display the time range that the mouse was dragged over.
- Right click in the time line to zoom out on the waveforms.
- Press the **Zoom Full** button on the Diagram window to return the zoom level to the entire simulation range.

1.5) Save the Project, Waveforms and Source Code

Next we will learn to save the project, waveforms, and source code. The project saves the simulation options and the names of the files contained on the project tree. It does not save the source code or the watched signals. To save the project:

- Choose the **Project > Save HDL Project** menu option to open the *Save Project As* dialog.
- Type **add4test.hpj** into the *File name* edit box.

- Click the **Save** button to close the dialog. Notice that the name of the project is displayed in the titled bar of the Project window.

The watched signals are saved using timing diagram files. By making the watched signals separate from the project file, VeriLogger lets you set up different sets of watched signals so that you do not have to watch your entire design each time you simulate. Also watching small sections of your design makes it easier to detect bugs in a particular section and speeds up simulation execution. In the evaluation version of VeriLogger you cannot save the waveforms. However, in the full version you can save the watched signals using the following menu command:

- **File > Save Timing Diagram**

Each time you simulate, every open editor is queried to determine if the source code needs to be saved before the simulation starts. If you need to save the code before you are ready to perform a simulation, use one of the following menu options:

- The **Editor > Save HDL** source menu option to save the source code in the editor with the focus.
- The **Editor > Save All** menu option to save the source code in all opened editors.

To re-open a VeriLogger Project, first open the project and then load the timing diagram files.

1.6) Configure the Project for Part 2

Now we will set up the project for the next section by removing the test bench file and saving the project under a different name. To remove **add4test.v** from the project:

- Left click on the **add4test.v** filename to select it in the project tree.
- Press the **Delete** key on the keyboard to remove the file. Files can also be removed using the context menu or the **Project > Remove File** menu option.
- Choose the **Project > Save HDL Project As** menu option and save the project under the name of **add4wave.hpj**.
- Choose **File > New Timing Diagram** to clear the Diagram window.
- Verify that only one file, **add4.v**, is listed on the project tree, and that the Diagram window is empty.

Part 2: Graphical Test Bench Generation

In this section you will draw and simulate a test bench using the timing diagram editor.

2.1) Build the Project and Examine the Black Signals

In the previous section, all the signals were purple to indicate that they were simulated signals that were generated by the Verilog code. In this section we have deleted the testbed module and the new top level module has input port signals that are not being driven by any other module in the project. To verify this:

- Choose the **Sim Diagram & Project** option from the drop-down box on the left side of the simulation button bar. This simulation state option lets the simulator compile both the drawn waveforms and the Verilog source code files together.
- Press the yellow **Build** button on the simulation button bar.
- Notice that the Diagram window now has two purple signals and three black signals. The purple signals are "simulated" signals whose values will be determined during the next simulation. The black signals are input signals that need to be defined before a non-trivial simulation can take place.
- Use the Project tree to verify that the black signals are input ports of the <<<**FourBitAdder**>>> module.

2.2) Use the Debug Run Simulation Mode

VeriLogger has two simulation modes: Auto Run and Debug Run. The simulation mode is displayed on the left most button on the simulation button bar. In the Debug Run mode, simulations are started only when the user presses the Run or Single Step buttons (similar to a standard Verilog simulator). In Auto Run mode the simulator will automatically run a simulation each time a waveform is edited in the Diagram window. This mode makes it easy to quickly test small modules and perform bottom-up testing. While drawing the original test bench we will set the simulator to Debug Run mode:

- Press the simulation mode button to toggle the display to **Debug Run**.



2.3) How to Draw Waveforms

If you are already familiar with SynaptiCAD's timing diagram editing environment, skip ahead to Section 2.5 where you will draw stimulus vectors and use the *Virtual State* edit box to define the values for the x and y busses.

If this is your first time using a SynaptiCAD timing diagram editor then we will first draw several random waveforms to familiarize you with the drawing environment.

1. Notice the buttons with the waveforms drawn on them. These are the state buttons. The active button is colored red and indicates the state of the next segment drawn. In this case, the **HIGH** state button is probably active.
2. Move the mouse cursor to inside the drawing window at the same level as the signal name **c_in**, and at about 40ns.
3. Left click to draw a waveform segment from 0ns to the cursor. Notice that a **HIGH** signal was created.
4. A different state button is now activated. The state buttons automatically toggle between the two most recently activated states. The small red **T** above the state name denotes the toggle state.
5. Move the cursor to about 80ns on the same signal and left click. Now a **LOW** segment is drawn from the end of the **HIGH** signal to the location of the cursor.
6. Left click on the **VAL** button to activate the valid state button and draw another waveform segment.
7. Draw more segments, using all the states except the HEX button. We will use this button later to define the state values for the multi-bit signals. For now, experiment with the graphical states on each of the black signals (the purple signals are outputs of the simulation and cannot be drawn on).

Your drawing should be a mess, or at least look nothing like Figure 2 located in Section 2.5.

2.4) How to Edit Waveforms

There are four main editing techniques used to modify existing signals (Note: these techniques will not work on clocks and simulated signals). The most commonly used technique is the dragging of signal transitions to adjust their location. The other three techniques all act on signal segments (the waveforms between two consecutive signal transitions). The segment waveform can be changed, deleted, or a new segment can be inserted within another segment. Use each of the following techniques:

1. **Move a signal transition:** Left click and hold on a signal transition. A green bar will appear that follows the mouse cursor. Release the mouse button when the green bar is at the desired location.
2. **Change the state of a segment:** A segment is the waveform between two consecutive signal transitions. Left click on the segment to select it (a selected segment has a highlighted box drawn around it). Then left click on the state button of the new state desired.

3. **Delete a segment:** Select a segment, then press the <delete> key.
4. **Insert a segment:** Inside a large segment, left click down and drag to the right, then release. A new segment will be added in the middle of the original segment. For this operation to work, the original segment must be wide enough to be selected.

For more detailed instructions, read the **TestBench and WaveFormer** on-line help *Chapter 1: Signals and Waveforms*.

2.5) Draw the Stimulus Waveforms

Now use the above techniques to edit the signals so they have roughly the same transitions and graphical states as the signals in the figure below. This is not the normal way to create a timing diagram, but it will teach you how to use the editing features of SynaptiCAD's timing diagram editor. Make sure you try all the editing techniques.

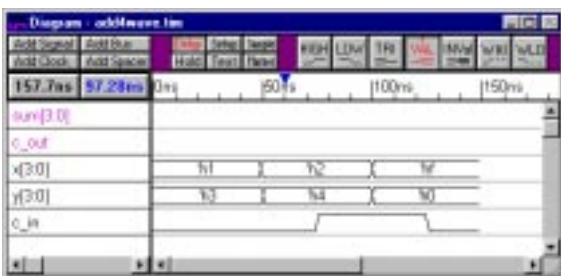


Figure 2: Stimulus vectors for the 4-bit adder circuit

Next, edit the virtual bus states of the valid segments on the x and y buses:

- Double click on the first segment on the **x** signal to open the *Edit Bus State* dialog.
- Enter the following hex value into the **virtual** edit box (make sure to type the single quote): **'h1**
- Press the **ALT-N** keys or **Next** button to move to the next segment on the signal.
- Continue to enter values into each segment so that it matches Figure 2 and press the **OK** button to accept the last value.
- Repeat the above instructions for the **y** signal.

At this point, the **c_in**, **x[3:0]**, and **y[3:0]** signals should look like Figure 2. Exact placement of edges is not required for this tutorial.

2.6) Simulate using the Auto Run Simulation Mode

Currently the simulator is in **Debug Run** mode, so simulations are started only when the Run button is pressed. Start a simulation:

- Press the green **Run** button on the simulation button bar.
- Verify that the **sum** and **c_out** are correctly being computed as **x + y + c_in**.
- Next, drag-and-drop an edge on the **x[3:0]** signal. Notice that the simulated signals do NOT change values because the simulator is in **Debug Run** mode.
- Press the green **Run** button to update the simulation values.

Next we will demonstrate the **Auto Run** mode which allows interactive debugging of modules. This mode is especially useful for debugging small modules.

- Press the simulation mode button to toggle the display to **Auto Run**.

- Drag-and-drop an edge on the **x[3:0]** signal. Notice that the simulated signals change values as soon as you drop the edge.
- Experiment with dragging edges and changing the values of the virtual states. If this was a low-level module that you just designed, you could quickly check the functionality of the module without having to design a formal test bench.

2.7) Import and Generate Waveforms

The most difficult and tedious part of designing test benches is accurately entering the waveform data. VeriLogger accelerates this process by accepting waveform data via six different methods: Verilog code, drawing, spreadsheets, simulator output, HP logic analyzer capture, and equation generation. So far we have demonstrated the drawing of waveforms and the use of standard Verilog code which are excellent choices for designing small test benches. However, for large test benches it is easier to use automated techniques to generate your data. The TestBencher and WaveFormer on-line help *Chapter 13: Stimulus Generation and Waveform Import* covers the spreadsheet, simulator import, and HP Logic analyzer features. The equation-based generation of waveforms is covered in *Chapter 11: Waveform Equation Generation*.

Now we will quickly demonstrate the waveform equation features using the following steps:

- Double click on the **x[3:0]** signal name to open the *Signal Properties* dialog box.
- Notice the drop-down edit box to the right of the **Wfm Eqn** button. This box is where temporal equations are entered. The default equation contains the syntax for all the possible states. If you start by editing this equation you will not have to look up the syntax for writing the temporal equation.

```
8ns=Z (5=1 5=0)*5 9=H 9=L 5=V 5=X
```

- Press the **Wfm Eqn** button to apply the above equation to the signal.
- Look at the generated waveform and compare it to the equation. Notice that the equation is a list of the form *time_duration=state_of_segment* elements. To repeat parts of the list use the syntax *(list)*repeat_number*.

You can also automatically label waveforms by using the **Label Waveform Equation** functions. These are more complex than the waveform equations, so you will have to read *Chapter 11* in the TestBencher and WaveFormer manual to get the full benefit of these features.

- Double click on the **x[3:0]** signal name to open the *Signal Properties* dialog box.
- Notice the drop-down edit box to the right of the **Label Eqn** button. This box is where label waveform equations are entered.
- Enter the following equation into the drop-down edit box: **Hex (Inc (0 , 1 , 16))**
- Press the **Label Eqn** button to label the signal for **x[3:0]**. This equation generates increments from 0 in steps of 1 for 16 times and outputs a hexadecimal value.
- Notice how the labels have changed on the signal (you may need to zoom in to clearly see all the segments). Also notice how the simulation output changed for the valid segments but it did not change for the non-valid segments. This is because the **virtual state** values are only used to define the state of the valid segments

Part 3: Breakpoints, Stepping, and Tracing

This section of the tutorial is still under construction. If you would like to practice debugging, first read the *Getting Started* and *Chapter 3: Simulation and Debug Functions* chapters in the on-line VeriLogger Help. Next, introduce a syntax error into the add4.v file and attempt to find it using the *Errors* tag in the Report window. Fix the syntax error, then introduce a semantic error in the full adder code so that it does not handle the carry correctly. Use breakpoints and single-step debugging to locate the error.

Index

\$cleartrace 17
 \$display 16, 17, 28
 \$dumpall 18, 26, 28
 \$dumpfile 18, 26, 28
 \$dumpoff 18, 26, 28
 \$dumpon 18, 26, 28
 \$dumpvars 18, 26, 28
 \$finish 16, 17
 \$key 17
 \$log 17
 \$monitor 28, 36
 \$nolog 17
 \$nokey 17
 \$scope 16, 17
 \$settrace 17
 \$showstats 35
 \$showvars 17, 35
 \$stop 17

A

Add

- breakpoint 15
- files to a project 5, 9
- signal to be watched 10

Auto Run simulation mode 14

Automatic test bench generation 25

Automatic Watches 11

B

back-annotated simulation 32

Breakpoints

- adding 15
- mouse shortcuts 15
- tab in Report window 15

Build button 14

Building a project 5, 9

C

Changing variables 17

Command file 28

Command line simulator

- creating a command file 29
- options 29

- predefined options 30

- preparing the source files 28

- running 28

- simulator control commands 31

comparing waveforms 18

Compiler directives 18

Console window 15

- entering commands 16

- interactive debugging in 16

- using 16

Continuing a simulation 16

D

Debug commands 17

Debug Run simulation mode 14

Debugging

- using the Console window 16

define (simulation option) 30

Diagram window usage 27

Differences from Verilog-XL 36

Displaying

- breakpoints 21

- errors 21

- the current code-line 16

- variables 16

Drawing stimulus 26

E

Editor keyboard commands 22

Editor Preferences dialog 22

Editors - external 24

Errors tab in the Report window 15

Expanding/hiding tree branches 10

Expand to Local Scope button 15

F

Finding a line of code 21

G

Goto button 14

Grouping waveforms 25

I

Importing VCD files 26

incdir (simulation option) 31

Inout stimulus 26

Interactive Verilog commands 16

Internal signal stimulus 26

J

Jump To dialog 21

K

Keyboard shortcuts in editor windows 22

L

libext (simulation option) 31

Local Scope button 15

M

maxdelays (simulation option) 30

mindelays (simulation option) 30

N

noshow_var_change (simulation option) 30

O

Open

- existing source code 21

- grouped signals 25

- project 9

- VCD files 26

- Verilog file 10

P

Port collapsing 34

Port connections of different net types 34

Project

- adding files to 5, 9

- building 5, 9

- opening 9

- saving 9

- simulating 6

- starting a new project 9

- using the tree control 10

Project Preferences dialog 12

Project Settings dialog 12

Project window 10

Pullup/Pulldown gates 34

R

Report Window 15

Restart button 14

Run/Resume button 14

S

Save

- existing source code 21

- grouped signals 25

- project 9

Scrolling to a time or offset position 27

SDF 32

Searching for a character string 22

Sim Diagram & Project simulation state 6, 14

Sim Independently simulation state 6, 14

Simulating the project 6

Simulation

- command controls 16

- continuing 16

- displaying the current code-line 16

- stopping 16

- system tasks 17

simulation

- back-annotated 32

Simulation button bar 14

Simulation modes 14

simulation states 6, 14

Specify blocks 33

Standard Delay File (SDF) 32

Step and trace 16

Step Into and Trace Calls button 14

Step Over Calls button 14

Stop button 14

Stopping a simulation run 16

synopsys (simulation option) 30

System tasks 17

T

Test bench generation, disabling 25

Top Scope button 14

Trace implementation 34

Tree control

- expanding/hiding branches 10

typdelays (simulation option) 30

U

User interrupt 35

User-defined Primitives and memory usage 33

User-defined primitives and memory usage 33

V

Variables

- changing 17

- displaying 16
- VCD commands 18
- VCD files
 - generating 26
 - reading 26
 - saving waveform data 28
- Verilog statements that generate output 28
- verilog.log file 15
- VeriLogger compilation process 33
- VERIWELL - predefined macro 35
- View
 - signal declarations 10
 - source code 10
- Viewing all signal values 27
- W**
- Watching a signal
 - determining output 26
 - in a specific instance of a module 11
 - in all instances of a module 11
 - selecting the signal 6, 10
 - stop watching a signal 11
- Waveform 18
- waveform comparison 18
- Z**
- Zooming techniques 27