

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.

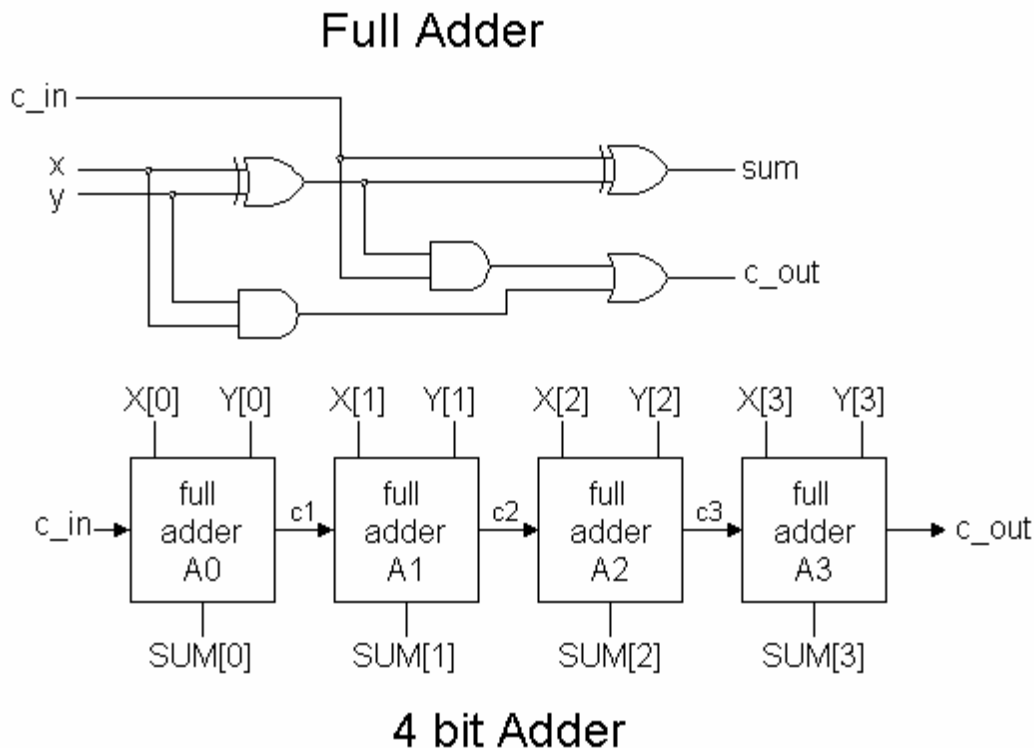


Figure 1: Schematic of the 4-bit adder simulated in this tutorial.

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.

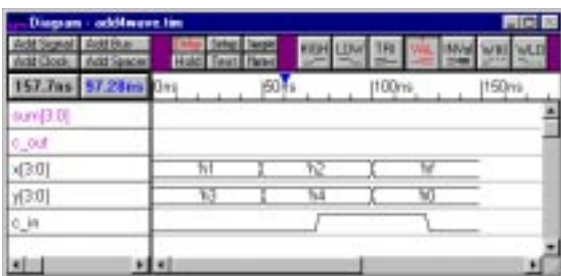


Figure 2: Stimulus vectors for the 4-bit adder circuit

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.