



Simulating a Verilog HDL Design using the ModelSim® Compiler and Simulator in Stand-Alone Mode

B. Earl Wells and Aleksandar Milenkovic
Electrical and Computer Engineering

© 2008-2015 The University of Alabama in Huntsville


This tutorial provides a basic understanding of how to edit, compile, and simulate Verilog HDL models using ModeSim® from Mentor Graphics. This will allow for stand-alone simulation RTL and timing simulations that exploit the full set of modeling features present in the Verilog HDL. Methodologies discussed here are applicable to the pre-synthesis environments where timing values can be directly asserted in the model by the designer to better model its actual or intended behavior before the design goes through synthesis and post-synthesis simulation which better reflect its actual behavior of the final implementation.

The example that is being illustrated is a Verilog HDL representation of an 8-bit two's complement subtractor/adder design. The design flow should be generic enough to apply to most versions of ModelSim® such as that which is installed on the PCs in the Advanced Logic Design Laboratory in EB 226.

This version of the ModelSim ® software is optimized around Altera® libraries (even though it is being explained here in a general manner). A more generic free version of the ModelSim PE Student Edition ® software can be found at http://www.mentor.com/company/higher_ed/modelsim-student-edition.

To use the ModelSim VERILOG HDL compiler/simulator in the Advanced Logic Design Lab proceed as follows:

Before entering the ModelSim® design environment place your personal USB flash drive in the USB port. This is where you are to store your design files for this lab. It is suggested that you use a USB flash drive that has a capacity of 32 Mbytes or greater. Make sure that you are careful with this drive and that you exit the ModelSim® software and properly stop the drive before you remove it from the PC or you may have loss of data. It is the student's responsibility to properly backup his/her flash drive. Throughout this tutorial it is assumed that the Windows operating system maps your flash drive to the device labeled **e:**. If this is not the case you should substitute the actual drive letter for the **e:** labeling presented here.

After the USB flash drive has been placed in the USB port, double click on the ModelSim icon,  , that is on the desktop or under the programs *Start* menu.

A “Welcome to the ModelSim” popup window will appear which can now be closed. This will reveal the main ModelSim command window that is shown in Figure 1.

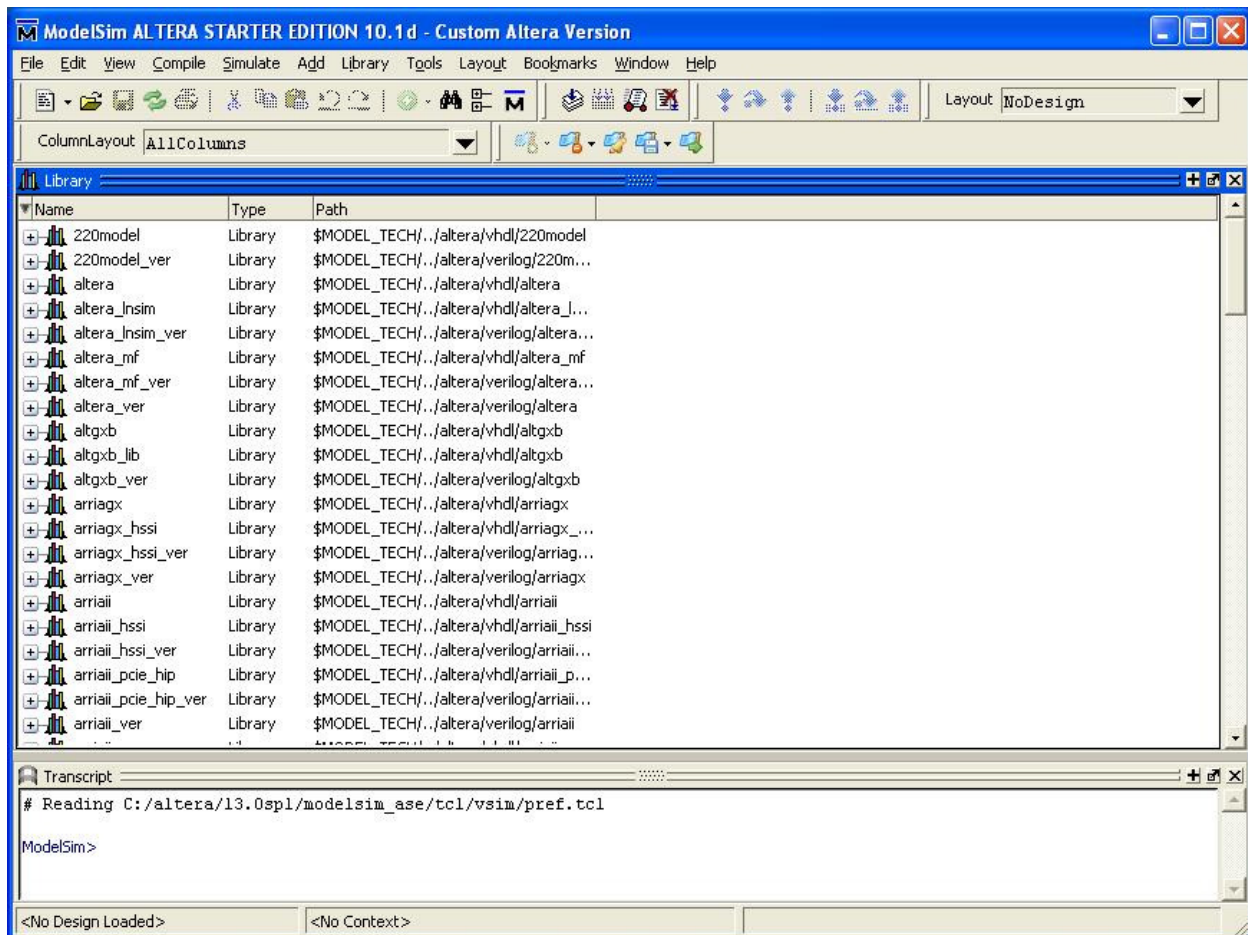


Figure 1. ModelSim® command window.

The first step is to set up the project for the current design. A project is the mechanism to keep track of all the design entities that are present in a design. To do this, click on the **File->New->Project** menu to bring up the *Create Project* dialog box as shown in Figure 2.

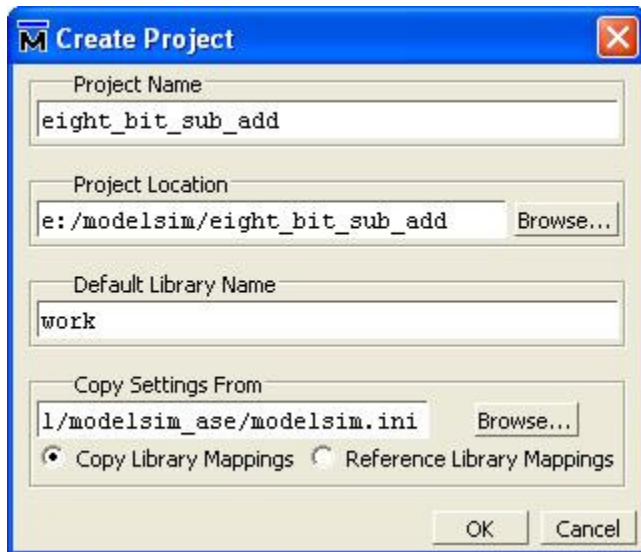


Figure 2. Create Project Dialog Box.

The project should be given a name (i.e. *eight_bit_sub_add* in this case) and the location of the project on your USB drive must be specified (i.e. *e:/modelsim/eight_bit_sub_add*). A dialog box will appear that warns you that these folders do not exist at which time a prompt will appear that will allow them to be created. Click on the **YES** button to continue. Next one should click on the **Create New File** icon which should launch the **Create Project File** popup dialog box as shown in below Figure 3.



Figure 3. Create Project File Popup.

Enter the appropriate VERILOG HDL source file name directly into the *File Name* field (for this example enter the file name *eight_bit_sub_add.v*). Check that the file type is *Verilog*, that this is a top-level entity file and click on the **OK** button. Then close the **Create Project File** popup. (You can always add more files to your project by using the **File -> Add to Project** menu options). The file that you specified should now be associated with the project and its name should appear in the *Workspace* section of the main ModelSim ® window (upper left-hand corner of the screen). At this point in time, the file is empty, to enter the Verilog model editor double click on the file name. Then enter the text for the Verilog HDL representation of an 8-bit two's complement subtractor/adder Verilog HDL model that is shown in Figure 4.

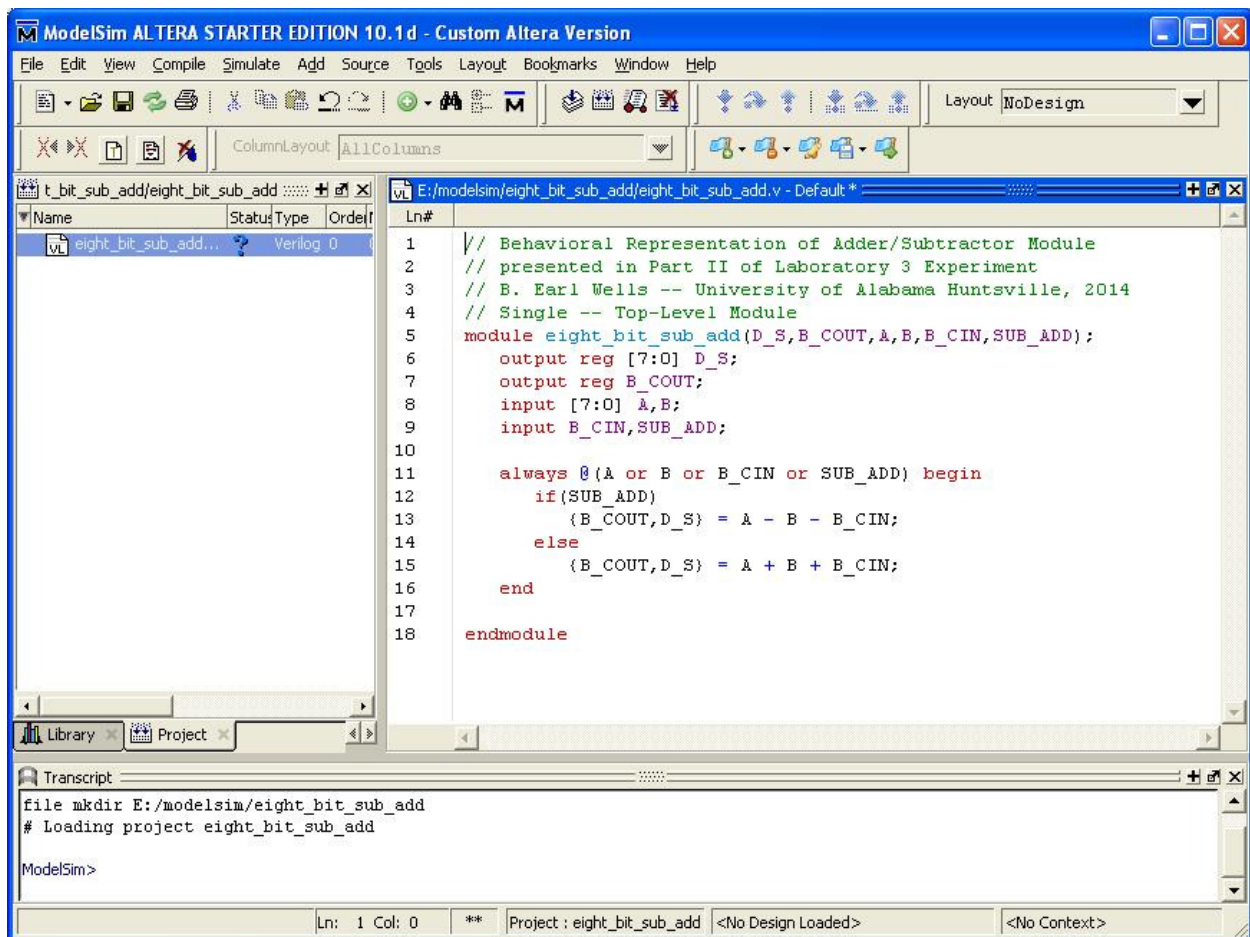


Figure 4. Behavioral Verilog File for 8-bit Two's Complement Subtractor/Adder Module

This source file represents a behavioral description of an eight bit two's complement subtractor/adder module that has a borrow/carry in and a borrow/carry out I/O that allows it to be cascaded with other such modules to extend the precision of the arithmetic. After the file (both has been entered it should be saved using the **File->Save** option. At this point, in the workspace area of your ModelSim® Window there should be an associated Verilog HDL reference (i.e. eight_bit_sub_add.v) that has a status that is tagged as unknown (i.e. ?). The next step is the compilation of the Verilog HDL module.

To compile the Verilog HDL module

After saving the source file to compile the design, click on the **Compile->Compile All** option in the top of the window menu. A series of messages that are similar to the ones shown below should appear in the *transcript* window that is present at the bottom of the ModelSim ® window.

```
# Reading C:/altera/13.0sp1/modelsim_ase/tcl/vsim/pref.tcl
file mkdir E:/modelsim/eight_bit_sub_add
# Loading project eight_bit_sub_add
# Compile of eight_bit_sub_add.v was successful.
```

The *transcript* window is where pertinent information is displayed and where you can enter command-line instructions to both the simulator and the compiler. If there are errors this process will have to be repeated as many times as necessary by editing and altering the Verilog HDL source file to remove the errors and then save it and recompile it.

To simulate a Verilog HDL module

Once the Verilog HDL model is successfully compiled into the default *work* library, the simulator can be invoked. To do this first type in the following command directly into the transcript window (Note: there are also equivalent items on the menu windows which will perform the same function for each of these commands which you can use instead.)

vsim eight_bit_sub_add

Messages similar to the following should then appear:

```
# vsim eight_bit_sub_add
# Loading work.eight_bit_sub_add
```

At this point the design is loaded and is ready to be simulated. To simulate a design one needs to *force* the inputs of the design to specific logic values to see if it behaves in the expected manner. We often use the Verilog HDL language itself to provide this design input stimulus by creating a separate top-level Verilog HDL module that is called a *testbench*. In such a scenario, our entire design would be instantiated as a single Unit Under Test component that is driven by top-level Verilog HDL statements that are present in the *testbench*. For now, though, we will use the built-in stimulus generating commands in the simulator. ModelSim® command files are an easy way to implement a series of commands. The ModelSim® command interpreter is actually a Tcl interpreter with ModelSim® specific functions added. There are dozens of ModelSim® commands but the ones we are interested in are **force**, **run**, **add list** and **add wave**. Note that any of the commands in a command file can be interactively entered at the VSIM command prompt in the ModelSim® window. First we should select the signals we which to monitor in the simulation (in our example here the A, B, B_CIN, and SUB_ADD input signals and the D_S and B_COUT output signals). To monitor these signals in a textual manner we must first enter the following command in the transcript area

add list A B B_CIN SUB_ADD D_S B_COUT

(Note: To monitor the same set of signals as a graphical waveform use the command:

add wave A B B_CIN SUB_ADD D_S B_COUT

This will be discussed more fully later in this tutorial)

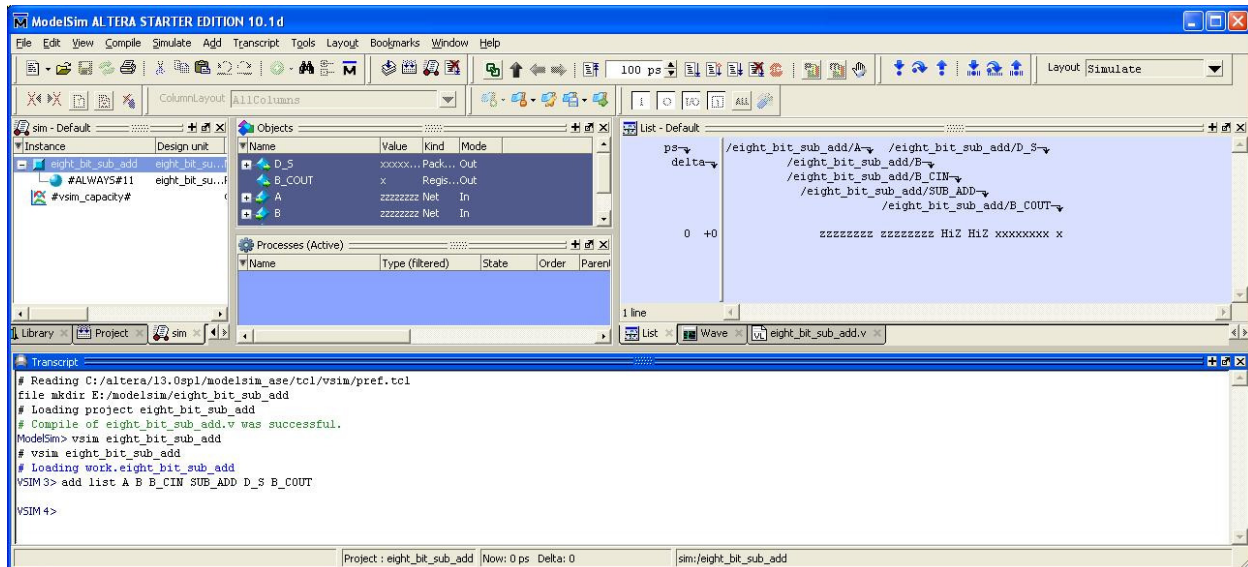


Figure 5. Initial Textual Simulation Output Window

To stimulate the design to add and then subtract two numbers a set of force commands can be entered into the transcript area of the Model Sim® window. These are shown below.

```
force A "11111111" 0 ns, "01010101" 500 ns
force B "00000001" 0 ns, "11111110" 500 ns
force B_CIN 1 0 ns, 0 500 ns
force SUB_ADD 0 0 ns, 1 250 ns, 0 500 ns, 1 750 ns
```

The first statement will cause the input A to take on the values of “11111111” at time 0 ns, and then forces it to “01010101” at time 500 ns. The second statement causes the input B to take on the values “00000001” at 0 ns, and then forces it to “11111110” at 500 ns. In a similar manner the B_CIN input is made to initially be a logic 1 at 0 ns and then is made a logic 0 at 500 ns, and the input SUB_ADD is toggled back and forth between a logic 0 to 1 every 250 ns (ending at a logic ‘1’ at 750 ns). This in effects Adds A to B for the first 250 ns with the carry in set, then Subtracts B from A during the next 250 ns with the borrow in bit set. At the 500 ns point, the value of A and B are changed and A is again added to B but with the carry in cleared and at the 750 ns point B is subtracted from A with the borrow in cleared. Once the input stimulus is specified the simulation can be executed. To run the simulation for a duration of 1 us, then one would simply enter the **run** command in the transcript area as shown below:

```
run 1 us
```

A *list* window will then be created that displays the logic values associated with the inputs and outputs in a textual form as shown in Figure 6.

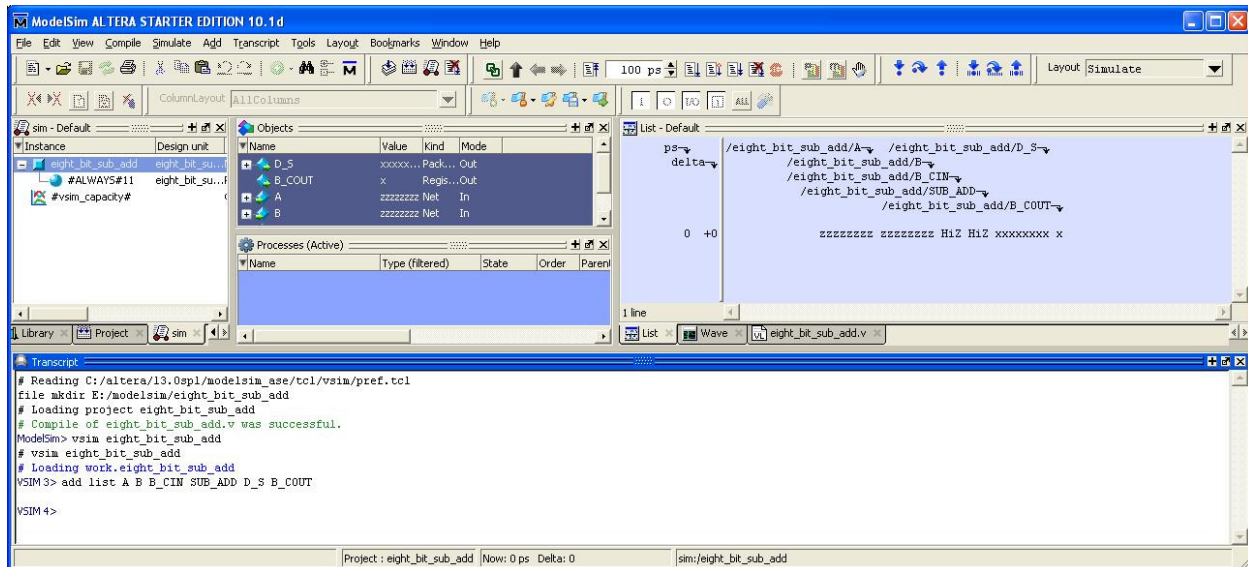


Figure 6. Final List Window of Eight-Bit Two's Complement Subtractor/Adder Design

The same design can then be run for a larger number of times steps by issuing another **run** command where the time value argument then representing the additional time from the current point in the simulation that the simulation is to be run. In between subsequent **run** commands the user can interactively change the input stimulus by issuing new **force** commands. (Of course, in this example there are no additional events that occur after about 750 ns no matter how long the simulation is run.)

[Notes: In this design example, the **force** command was used to generate an aperiodic set of transitions for each input where the last specified value is held for the duration of the simulation. This allows one to enter the entire input stimulus in single set of statements at the beginning of the simulation with the run command being entered just one time. An alternative method of generating input stimulus in an interactive manner by issuing and reissuing as necessary the **force** command without a time value for the signals that change value during the current portion of the simulation (i.e. **force B_CIN 1**) and then issuing a new **run** command to advance the simulation through the next specified time period.

The **force** command can also be used to generate periodic waveforms that never terminate. For example to drive the SUB_ADD signal indefinitely one could enter

```
force SUB_ADD 1 0 ns, 0 250 ns -r 500ns
```

This will set up a periodic 500ns 50% duty cycle clock type signal that has the same affect as the previous statement but does not terminate after 1 us.]

Another useful view of the simulation can be obtained by using the **add wave** command in the same manner as the **add list** command. For this example the syntax for this command would be

```
add wave A B B_CIN SUB_ADD D_S B_COUT
```

The resulting simulation view will be a graphical one as shown in Figure 7.

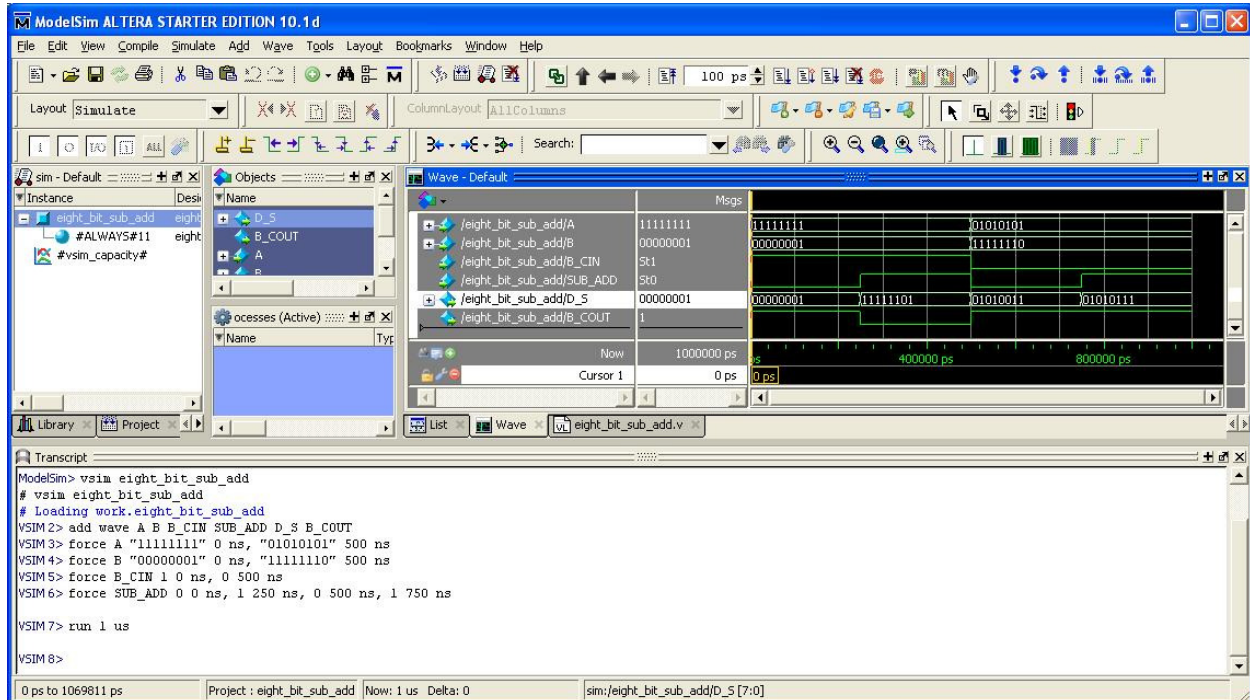


Figure 7. Wave Window of Eight-Bit Two's Complement Subtractor/Adder

In this view the zoom out icon (negative magnifying glass) was used to view the entire waveform in a single window. It is important to note that if both the **add list** and **add wave** options are entered at the beginning of the simulation then both the *list* and *wave* windows will be accessible and viewable by selecting the appropriate tab at the top of the window. There are many options for viewing these waveforms graphically, which include, zooming in and out, setting the radix of the signal buses and expanding the signals into their binary equivalents. Another command that is useful is the

restart –f

command that resets the simulation to the beginning. This is useful when experimenting with the effects of different input stimuli. There are many options for viewing the simulation state so the student is encouraged to read additional documentation and experiment on their own!