

Using Modelsim

■ Introduction

To use Modelsim, you first need to create a testbench file. This is a special kind of Verilog file that doesn't synthesize to hardware – it is simply there to test existing modules. It has language features for initializing input pins for your test module (usually called the **UUT** – Unit Under Test).

For example, if I had a module called *CounterModule*, and I wanted to simulate its behavior to make sure it worked as I expected it to, I would create a Verilog testbench file, which looks similar to a module itself but acts differently. Then, I would test it in Modelsim.

■ The Testbench File

Creating the Project

First, create a new project in Xilinx ISE. Next, add a new source, consisting of the following:

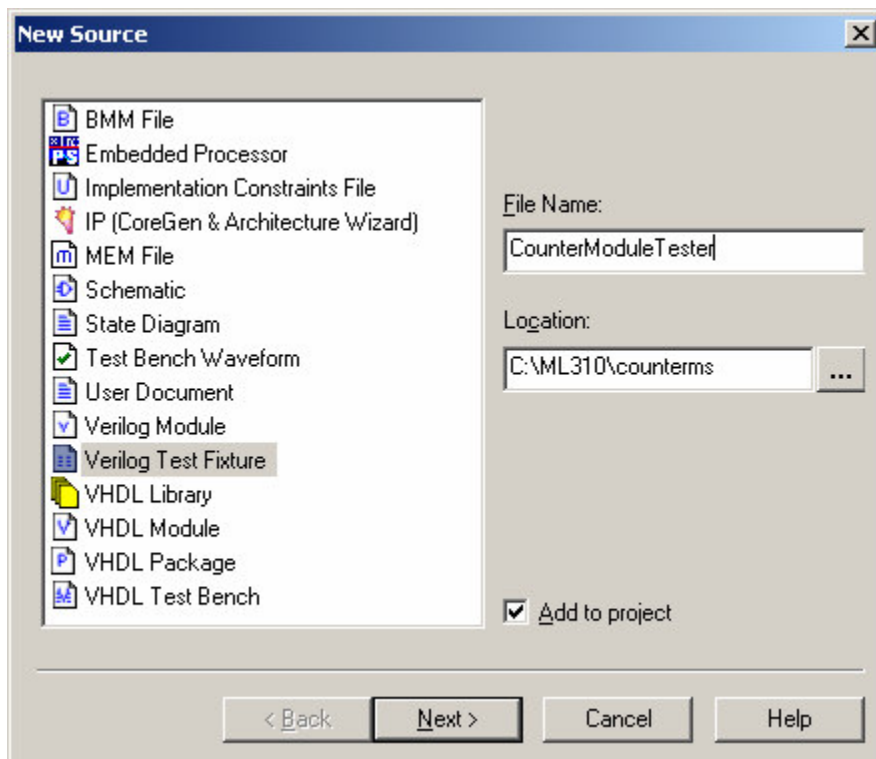
```
module CounterModule(clock, reset, direction, counter_value, counter_sum);  
  
    input        clock;  
    input        reset;  
    input        direction;  
    output [31:0] counter_value;  
    output [16:0] counter_sum;  
  
    reg [31:0]    counter_value;  
  
    wire [15:0] counter_x = counter_value[31:16];  
    wire [15:0] counter_y = counter_value[15: 0];  
  
    assign counter_sum = counter_x + counter_y;  
  
    always @(posedge clock) begin  
        if( reset ) begin  
            counter_value <= 0;  
        end else begin  
            if( direction == 0 ) begin  
                counter_value <= counter_value + 1;  
            end else begin  
                counter_value <= counter_value - 1;  
            end  
        end  
    end  
endmodule
```

As you can see, this module returns a 32-bit number that counts up if the *direction* pin is low, and counts down if *direction* is high. It stores this value in *counter_value*.

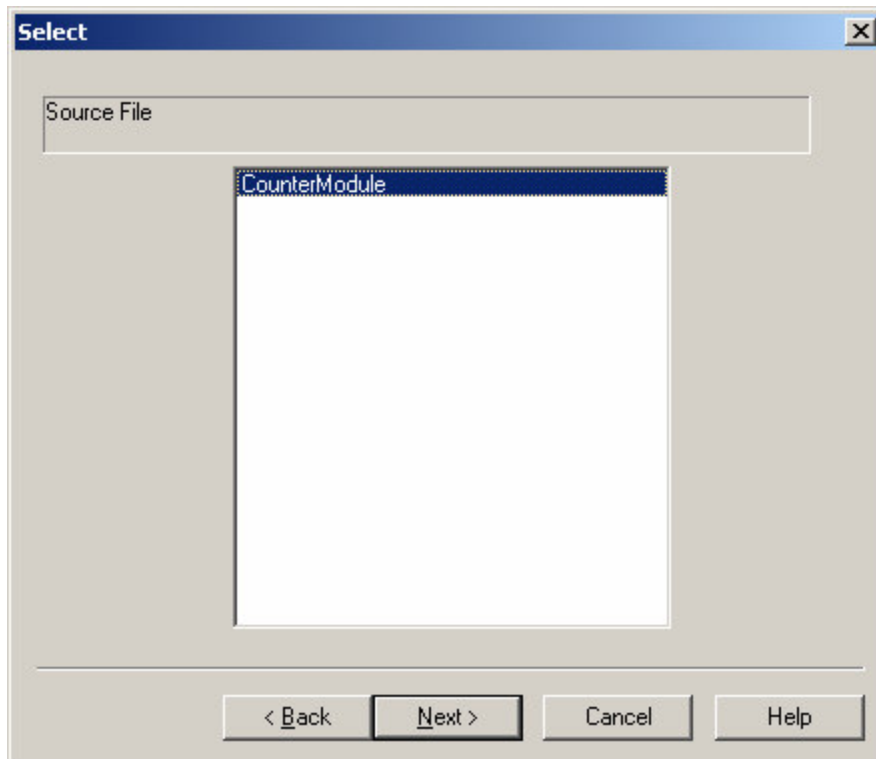
Additionally, the module will add the top 16-bit number with the bottom 16-bit number, and store this value in *counter_sum*. In the code, the upper 16 bits is represented by the wire bus *counter_x*, and the lower 16 bits are represented by the wire bus *counter_y*.

Creating The Testbench File

To create the testbench file, go into the *Project* menu, and select *New Source*. Select *Verilog Test Fixture* and give it a filename, in this case we'll use *CounterModuleTester*.



Click on Next, and the program will ask which source file we intend on testing. We'll select *CounterModule*.



Finally, click through *Next* and *Finish*.

It will generate the basic code that you need to start testing this. The first thing you'll notice is the *initial* block. This is where you setup your initial input values.

```
initial begin
    // Initialize Inputs
    clock = 0;
    reset = 0;
    direction = 0;

    // Wait 100 ns for global reset to finish
    #100;

    // Add stimulus here

end
```

You can see that it has already initialized all of the inputs to zero for you.

There is a new language structure here for delays. If you include a *#number* before a command, it will wait *number* ns before running that command. For example, if I typed in:

```
| #50 direction = 1;
```

It would wait 50 ns, and then set *direction* high.

Knowing this, there are two things you must take care of first: setting then unsetting the reset, meaning let it stay high for a little while, then push it low; and setting up the clock.

The reset is dealt with in the *initial* block. You simply initialize reset to 1, then insert a delay of about 100ns or so (depending on your clock speed) and set it to 0. The code would look like this:

```
initial begin
    // Initialize Inputs
    clock = 0;
    reset = 1;
    direction = 0;

    // Wait 100 ns for global reset to finish
    #100;
    reset = 0;

    // Add stimulus here

end
```

To add the clock, you'll have to insert an *always* block towards the end. For a 50 MHz clock, which has a period of 20ns, we'll want to flip the clock's value every 10ns:

```
always begin
    #10 clock = ~clock;
end
```

The reason we use 10ns is because half of the period the clock is high, and the other half it stays low.

Now we'll add some more stimulus to the direction, to ensure that it works well.

In the *initial* block, after the reset has been set low, we'll wait some arbitrary amount of time, then flip the direction. I've chosen 2000 ns, or 2 μ s:

```
initial begin
    // Initialize Inputs
    clock = 0;
    reset = 1;
    direction = 0;

    // Wait 100 ns for global reset to finish
    #100;
    reset = 0;

    // Wait 2 us, then flip the direction
    #2000 direction = 1;

end
```

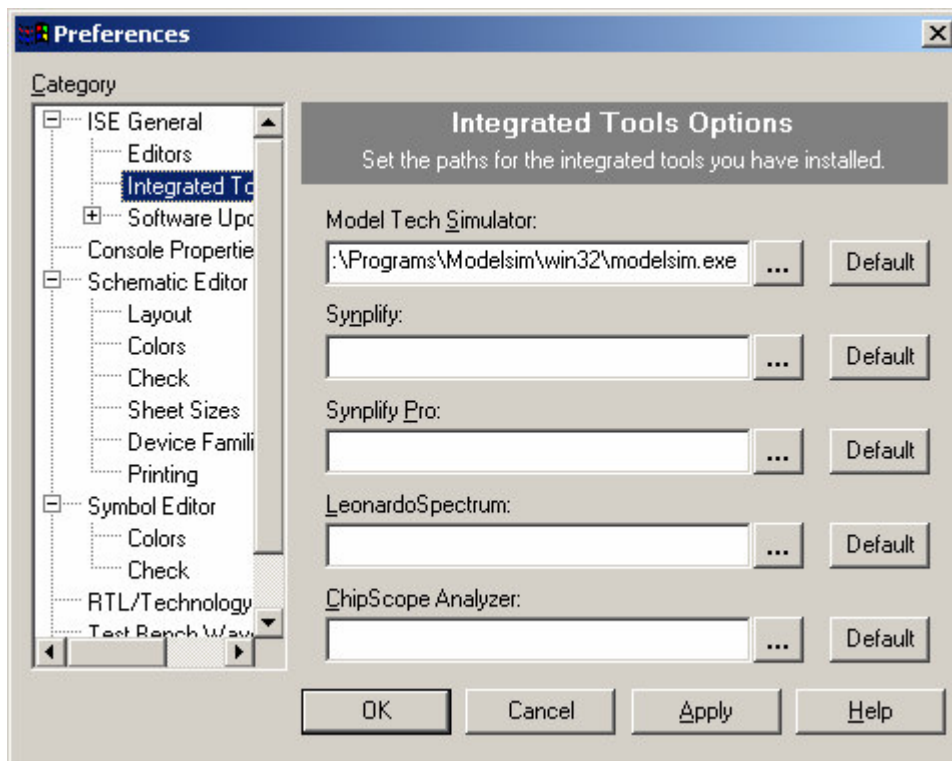
Now that the testbench file is completed, we are ready to test it in Modelsim.

▪ **Setting Up Modelsim**

Setting the Modelsim Path

Before you use Modelsim, make sure that it is properly setup to work with ISE.

In ISE, click on the *Edit* menu then select *Preferences*. On the left of this window, select *Integrated Tools* located under *ISE General*. Ensure that the correct Modelsim path is set under *Model Tech Simulator*. To browse for the file, click on the “...” button.

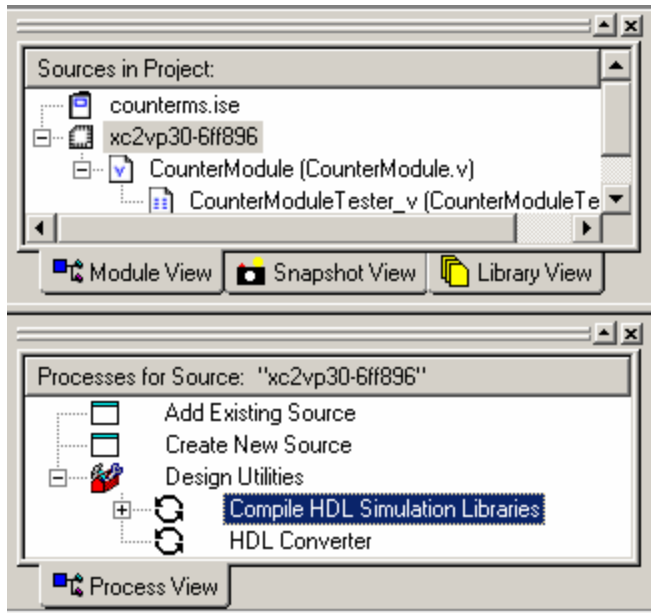


Compiling the Simulation Files

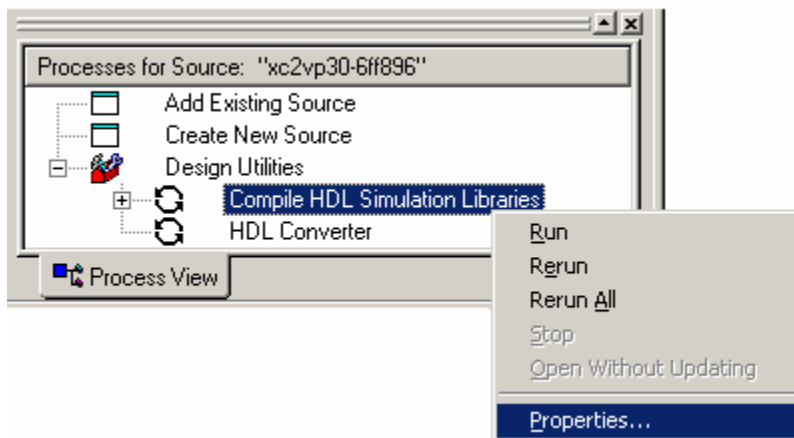
Next, we must make sure the simulation files are compiled correctly. **You should only have to do this once on a machine.** The only other time **you will have to do this is if you import a project from another machine.**

On the left-hand side of the ISE window, you should see two smaller windows. The top one contains a hierarchy of the modules and files in your project. The lower one contains the operations and commands that you can perform on the **currently selected** file.

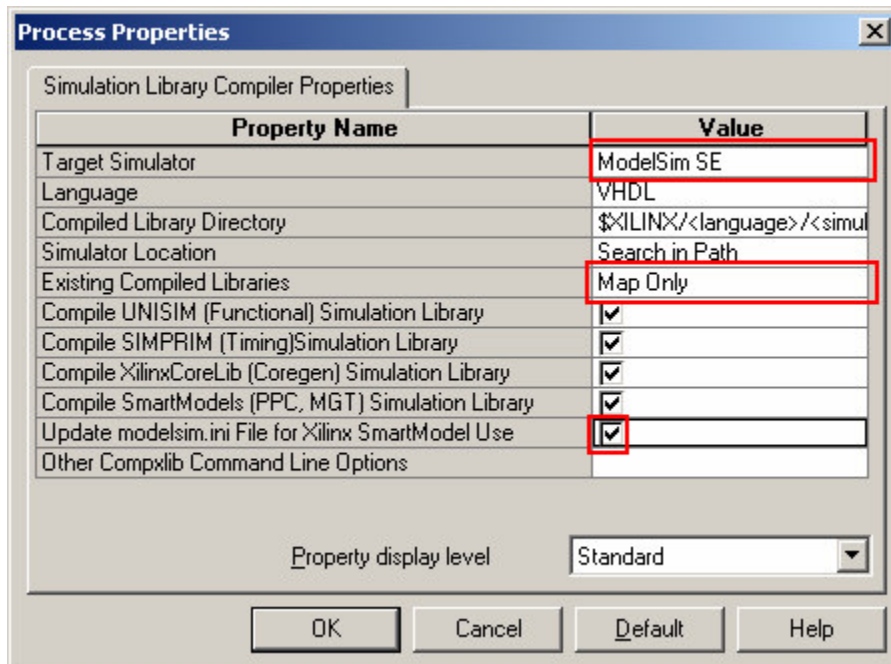
Click on the FPGA name in the hierarchy. This should have a name like xc2vp30, depending on the FPGA you're using. When you do this, the bottom window pane will change to display the functions you can perform for this chip.



Under *Design Utilities* in the lower window, right-click on *Compile HDL Simulation Libraries* and click on *Properties*:

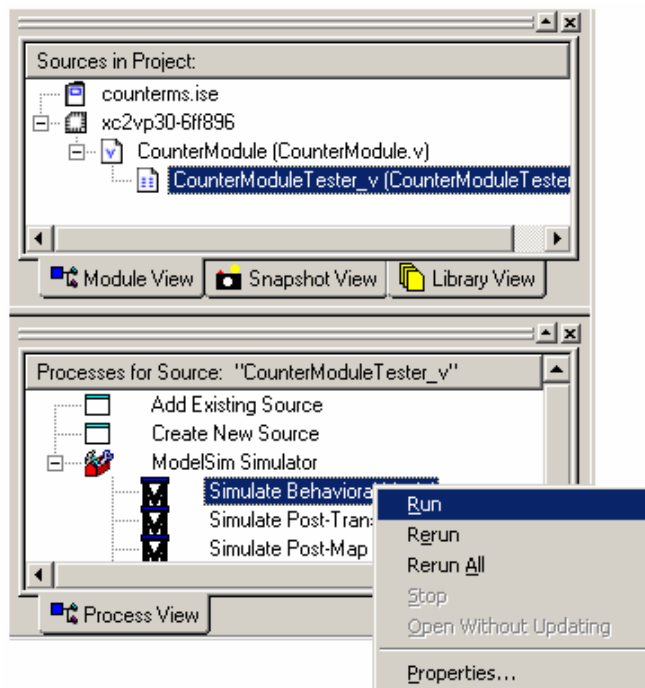


In the Properties window, select *Modelsim SE* as the Target Simulator, and change *Existing Compiled Libraries* to *Map Only*. This skips a lot of the compiling if it detects a previously compiled simulation set. Finally, select the checkbox for *Update modelsim.ini [...]*:



Now that the settings are all correct, **double click on the *Compile HDL Simulation Libraries*** option and wait for it to complete. This may take several minutes. You will know it has completed if there is a green checkmark icon or yellow exclamation icon (for warnings) in the *Process* window.

■ Simulating in Modelsim



From ISE to Modelsim

To start the simulation, select the testbench file in the *Module View* window on the top-left corner in ISE.

The *Process* window below it should update to show simulation-specific options.

Under the *Modelsim* option, double-click on *Simulate Behavioral Model*. This will launch Modelsim and test it with no timing propagations.

The Modelsim Environment

You should now be presented with two Modelsim windows: the main window, and the wave window (note that the wave window may be inside the main window, or it may be outside of it, showing up as a separate icon in your task bar).

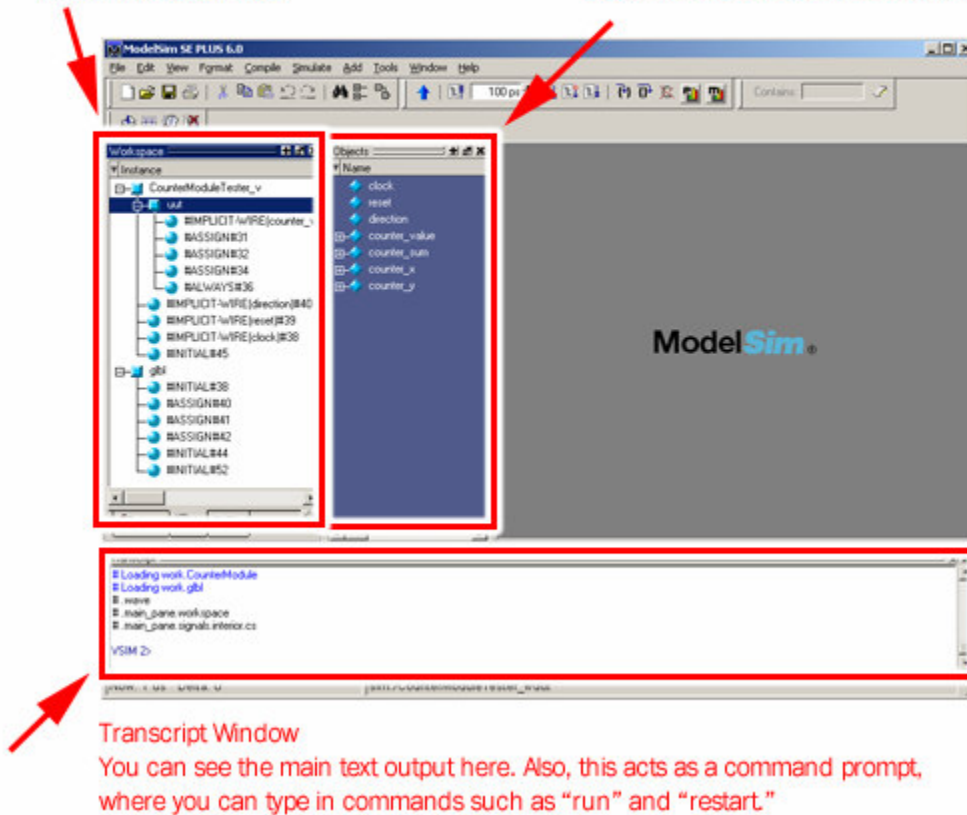
The Main Window:

Workspace Window:

This is the main listing for your project. You will see all of your modules listed in their hierarchy here.

Objects Window:

This is where the members (such as the internal wires and signals) of the selected module will appear.



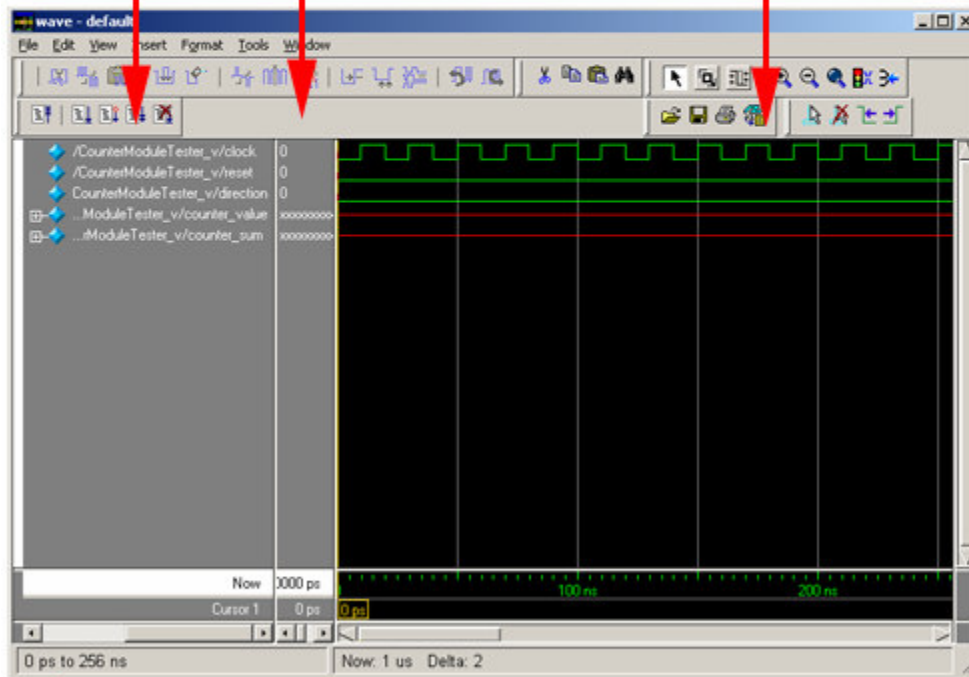
If your window does not look like this, click on *Window*, then select *Initial Layout*.

Now, the next main important window is the *Wave Window*. This should appear similarly to the following:

This will list all of the “nets” in the topmost module, which simply means all of the registers, wires, and parameters. You can add and remove to this list

Values in the radix of your choice of selected signals


The main output of the wave window – it shows the signals as they change through time.



The status bar will display how long the simulation has run, and the current timeframe you are observing.

I'll henceforth refer to the *Workspace* window, the *Objects* window, the *Transcript* window, and the *Wave* window. For the latter, I will refer to the *net list* and the *output*, which correspond to the column on the left and the black signal area on the right respectively.

Useful Features

Modelsim's windowing is very flexible. You can “break out” any of the windows by selecting the button to the left of the close button, that looks like this: 

Running the Simulation

The easiest method for running simulations is through the *Transcript* window. There are a few basic Modelsim commands that you will need to remember. The first one is the *run* command. It has the following format:

```
| VSIM 1> run [time value] [time units, such as ps, ns, us, ms, and sec]
```

For example, to run the simulation for 45 μ s, I would type in:

```
| VSIM 2> run 45us
```

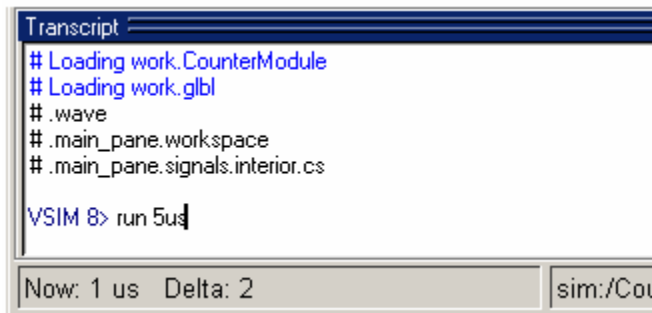
Oftentimes, however, you will need to restart the simulation. Later we will show you how to add more signals to the wave window, and when you do so, you will need to rerun the entire simulation to see how the newly added signal changes. To do this, you simply use the *restart* command:


```
| VSIM 3> restart
```

Note that if you prefer using toolbars, all of these features are available through the various toolbars in the main window. Once you get used to the commands, however, you will discover how fast and efficient they make your workflow.

Simulating the Counter

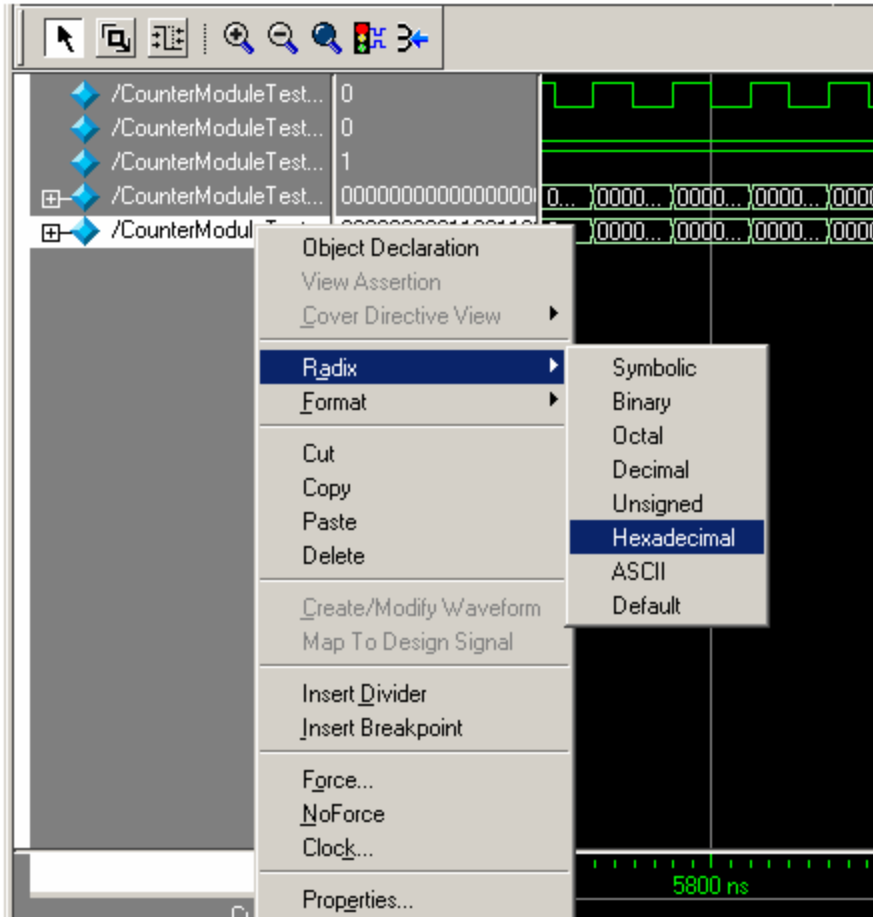
Now, in the counter module testbench, let's run the simulation. Type in *run 5us* in the *Transcript* window. Every time you run the simulation, it will fill in only the registers and wires that you have listed in the open wave window. This means that if you run the simulation, and decide later that you wanted to watch another signal, you must restart the simulation.



Now, if you look in the *Wave* window, you should see the signals changing. If you do not see much, try zooming out using the  zoom out toolbar button.

This should give you a good understanding of what the module is doing. Before 2 μ s, we should see the counter counting up. After that point, it should be counting down.

At this point, I recommend that you explore the different options in the wave window. For example, if you right-click on a net, you can change the format that it displays in by selecting the Radix submenu and choosing a numerical base such as hexadecimal. This is useful for long registers and large buses.



As you can see from the menu displayed, you can also insert a divider to organize your nets from different modules. If you cannot see the net name, you can scroll each one of these columns until you are able to see everything.

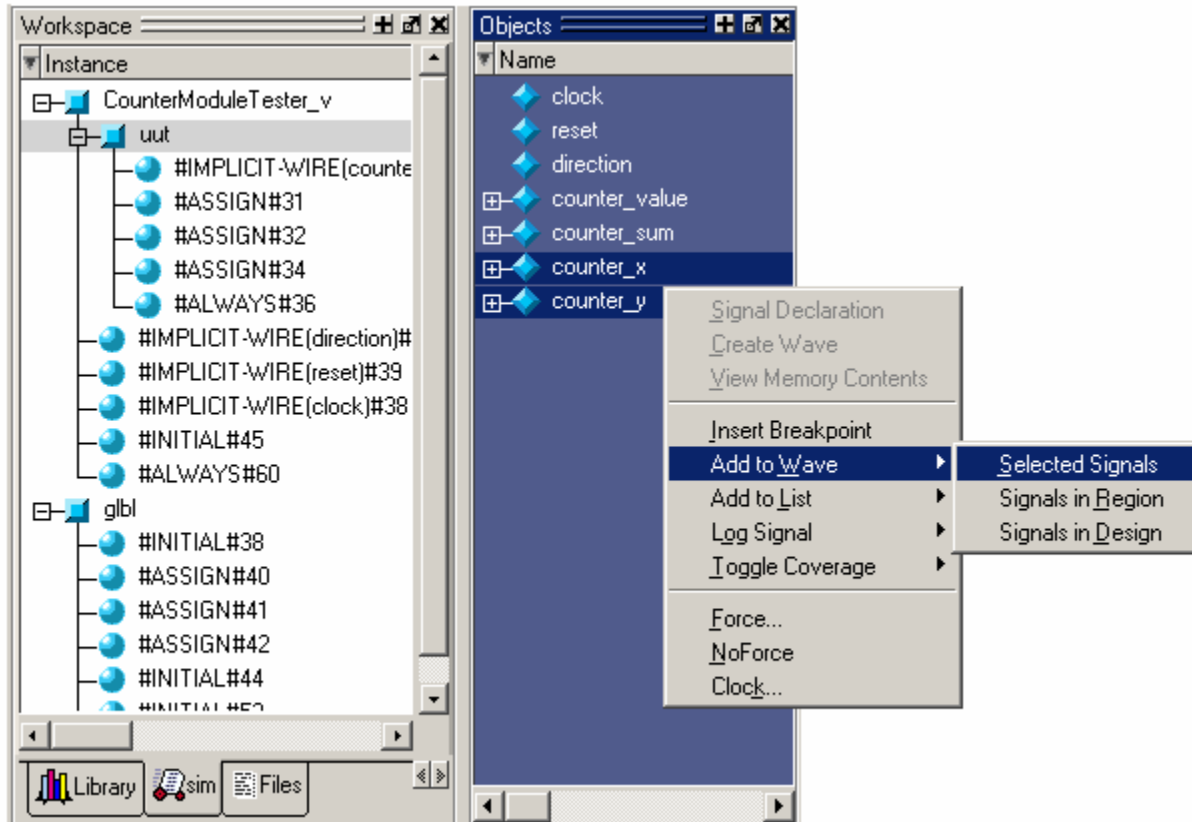
Adding More Signals

The final step in this tutorial is to add signals that are not added by default. This is especially useful for spying on signals that are many levels deep into your top module. To do this, we'll be using the *Workspace* and *Objects* windows.

For our example, we will add the upper and lower 16 bits of the counter to ensure that they are correctly summing to *counter_sum*. If you look in the *Workspace* window, you should see at the very top, *CounterModuleTester_v*. This is our testbench, and when you select it, the *Objects* window should display all of the signals used in that file.

Below that is *uut*, which is our Unit Under Test. This is simply the main *CounterModule*. If you select it, you should now be able to see *counter_x* and *counter_y* in the *Objects* window. These were the wires we used to represent the upper and lower bits of the counter, respectively.

To add them to the Wave window, right-click on them and from the submenu, select *Add To Wave*, then click on *Selected Signals*. You can select multiple nets at once by holding down the CTRL key and clicking on each signal you want to observe. Then add them all to the wave window at once.



This should be enough to get you through starting Modelsim. Note that Modelsim is a very powerful platform whose capabilities are plentiful, so feel free to explore the different options and features.