

CS152 – Computer Architecture and Engineering
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

Compiled: 2004-09-01 for CS152 by Prof. Dave Patterson, John Lazzaro, Brandon Ooi
Sources: http://inst.eecs.berkeley.edu/~cs152/handouts/Toolflow_cs152spring2004_flow.doc

Mini Lab 1: Design Entry

[1 Introduction to the Design Tool Flow](#)

[2 Basic Project Tutorial](#)

1 Introduction to the Design Tool Flow

Refer to the illustration on the next page for the steps involved in the CAD tool flow we will use.

1.1 Design Entry

The first step in logic design is to conceptualize your design. Once you have a good idea about the function and structure of your circuit, you can start the implementation process by specifying your circuit. In this class we will use a Hardware Description Language (HDL) called Verilog. HDLs have several advantages over other methods of circuit specification: ease of editing (files can be written using any text editor), ease of management when dealing with large designs, and the ability to use a high-level behavioral description of a circuit. If you are familiar with emacs, you may find it convenient for writing and editing Verilog code. Alternatively, you may use the editor provided in Xilinx's Project Navigator or ModelSim.

1.2 Synthesis

Once your design is entered, the next step in the implementation path is synthesis. In our case, the function of the synthesis program is to translate the Verilog description of the circuit into an equivalent circuit comprising a set of primitive circuit components that can be directly implemented on an FPGA. In a way, the synthesis tool is almost like a compiler. Where a compiler translates to a sequence of primitive commands that can be directly executed on a processor, synthesis translates to primitive circuit components that can be directly implemented in FPGA. The final product of a synthesis tool is a netlist file, a text file that contains a list of all the instances of primitive components in the translated circuit and a description of how they are interconnected. The synthesis tool we will be using in this class is called Synplify. Note that Xilinx's software suite can also perform this synthesis procedure. The reason we use Synplify is because Synplify is an industrial strength CAD tool program. It's faster, produces better logic, and will give many more synthesis warnings—something very useful for students!

1.3 Placement, Routing

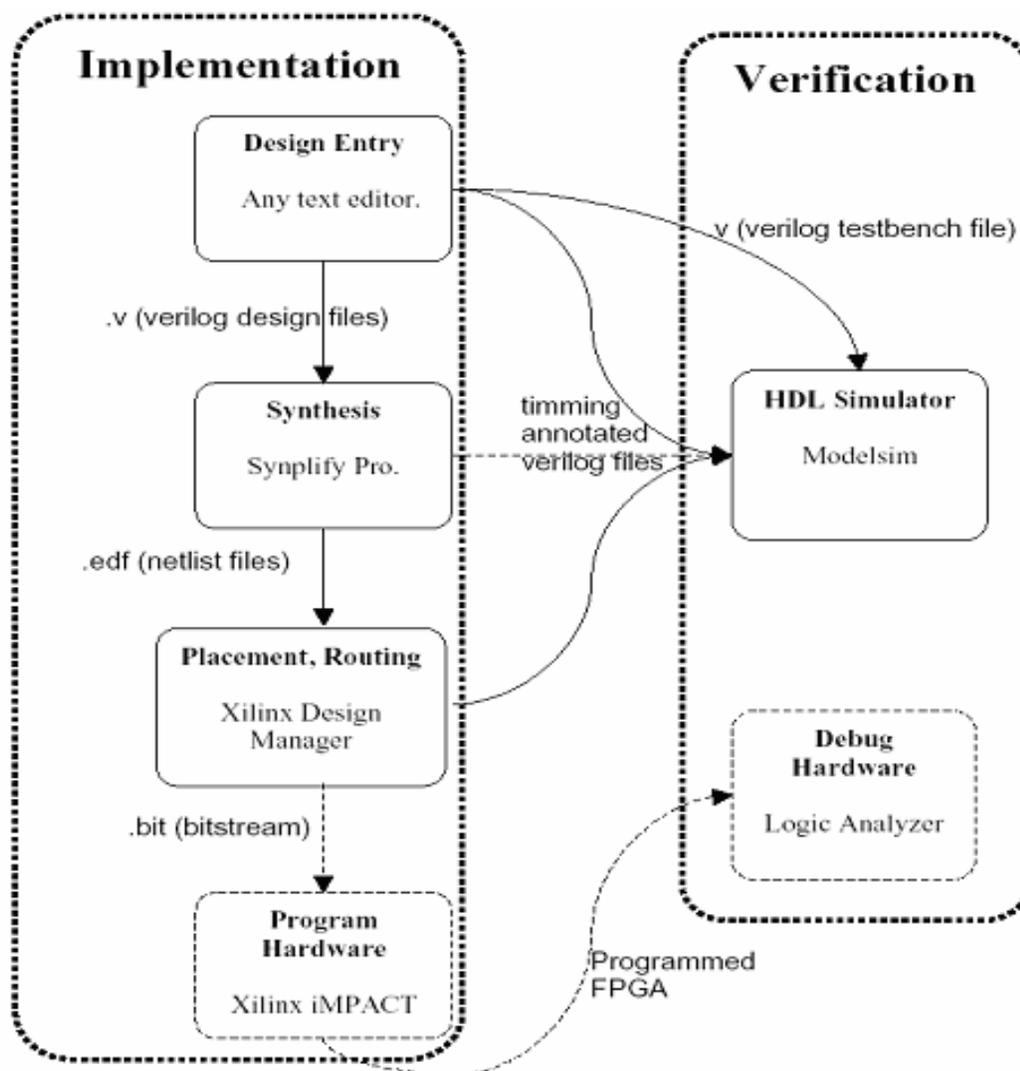
The next step in the implementation flow is to take the netlist of components generated by the synthesis tool and turn it into bits that are need to configure the LUTs, muxes, Flipflops, and other configurable resourses in the FPGA. To do that, first the primitive circuit components in the netlist need to be assigned to a specific *place* on the FPGA. For example, a 4LUT implementing the function of a 4 input NAND gate in a netlist could be implemented with any of the about 40,000 4LUTs in a Xilinx Virtex 2000E FPGA chip. Clever choice of placement will make the subsequent routing easier and result in a faster overall circuit. Once the components are placed, the proper connections must be made according to the netlist description obtained from the synthesis step. That step is called *routing*. Unlike synthesis, which only requires a set of primitive components to translate to; placement and routing are dependent upon the specific size and structure of the target FPGA chip. Due to this reliance, the FPGA vendor usually provides the placement and routing programs. Therefore, we will be using the Xilinx's Project Navigator to perform this step. The end product after placement and routing is a bit file containing the stream of bits used to configure the FPGA.

1.4 Program Hardware

The last step in the implementation flow is the simple act of transporting the configuration bits to the FPGA. There are also many ways of doing this. For this class we will be mostly using the Parallel Cable IV along with the iMPACT software to program the board.

1.5 Verification

As you should have learned from experience, a significant part of the time and effort spent on any sizable project will be spent on debugging, and logic design is no exception. There are two ways to verify the correctness of a design: to program the FPGA with the design and check if the circuit is behaving correctly, or to run simulations of the design in software. While programming the FPGA and physically checking the functionality sounds simple, the whole tool flow requires a significant amount of time to complete, especially as your designs get larger and more complex (20 minutes towards the end of the semester!). Repeatedly tweaking the input design to fix errors would require running the flow repeatedly, a huge waste of time. In addition it can be difficult to physically observe the causes for an error on a FPGA. For these reasons, software simulation is essential in the verification process. There are many places along the tool flow where you can use simulation to verify the correctness of your design. As you progress down the tool flow and more information about the physical implementation on the FPGA becomes available, more accurate timing simulations can be performed. **You should not attempt to verify that your design works on the board until it works in simulation!!!** To do so otherwise would just be a waste of time. In this class, we will be using ModelSim to do our verification.



CAD tool flow. Optional links are showed as dashed lines.

2 Basic Project Tutorial

2.1 [Creating a Project](#)

2.2 [Adding New Modules](#)

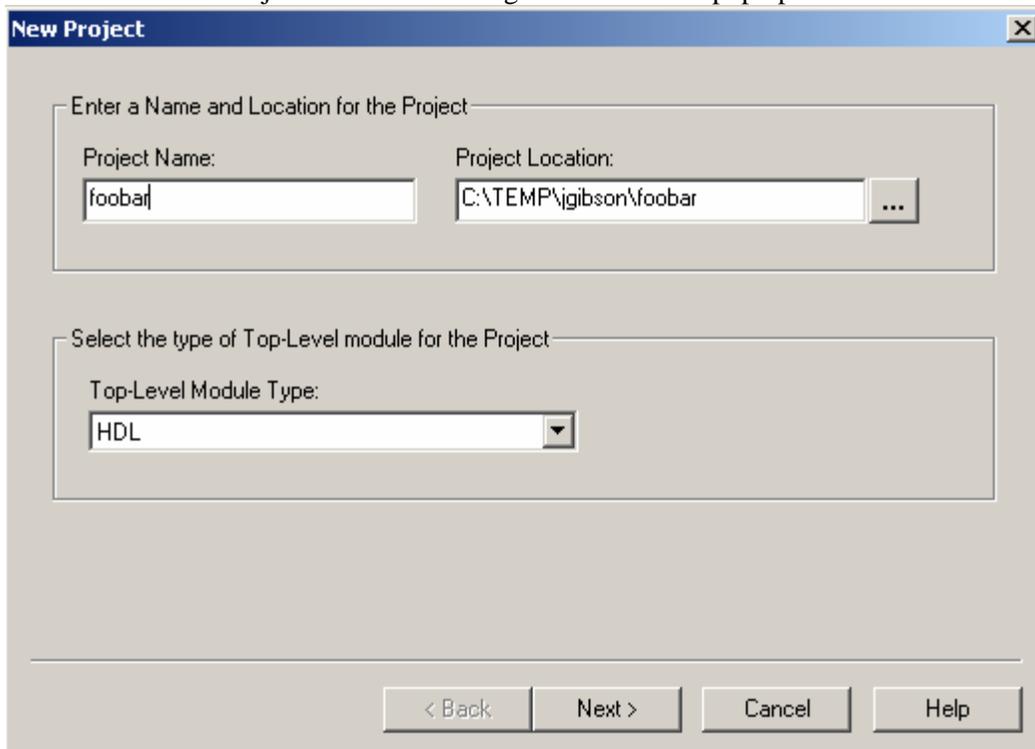
- i. [Adding a Verilog Source](#)
- ii. [Adding a Schematic Source](#)
- iii. [To view your schematic in Verilog HDL](#)
- iv. [Creating a Schematic Symbol from Verilog or Schematic Sources](#)
- v. [Combining Verilog and Schematic Sources](#)

2.3 [Simulating your design](#)

- i. [Adding Waves](#)

2.1 CREATING A PROJECT

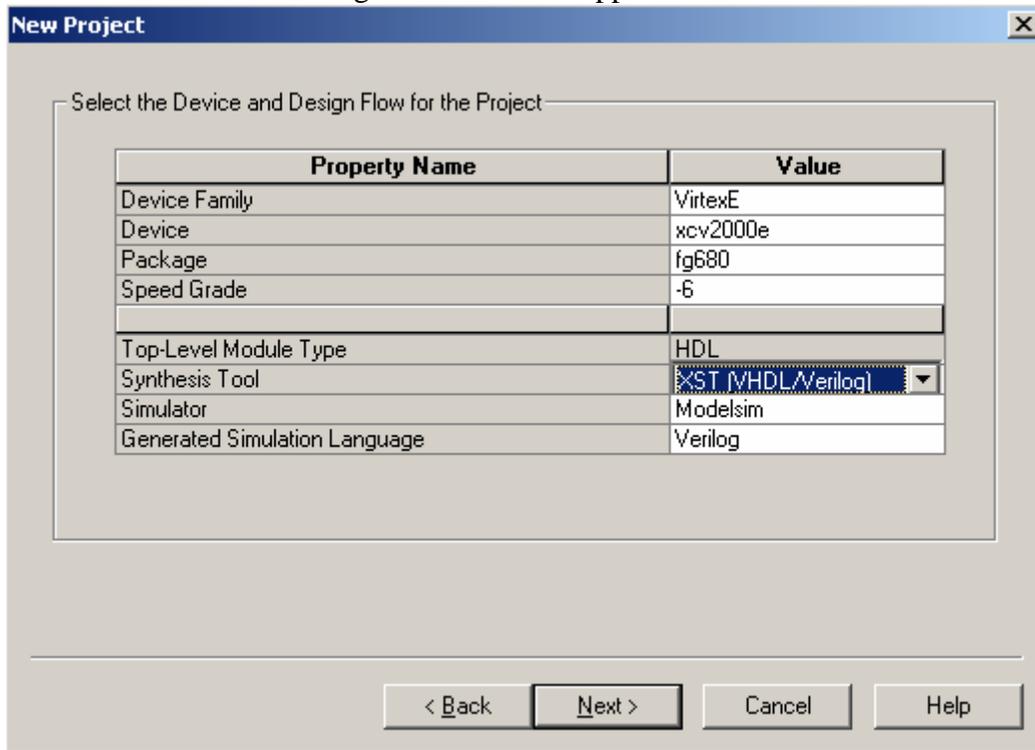
- 1) Load up Xilinx Project Navigator.
- 2) Go to File→ New Project and the following window should pop up:



- 3) Make the project settings the same as the ones seen above.

Project Name	What you wish to call the project
Project Location	Place in c:\Temp and later move the project directory into your U:\ for storage
Top-Level Module Type	HDL

- 4) Click next and the following window should appear



- 5) Set the project settings as seen above and click next when you're finished:

Device Family	VirtexE
Device	xcv2000e
Package	fg680
Speed Grade	-6
Synthesis Tool	XST (Verilog/VHDL)
Simulator	Modelsim
Generated Simulation Language	Verilog

- 6) Now you'll have the opportunity to add a new source to the project, you can do this now or later. Hit next when you're finished.
 7) Now you can add existing sources to the project. Like adding a new source this can be done after the project is created as well. Hit next when you're finished.

You should be on the final screen of the New Project wizard. It will list the settings that you have chosen, verify that they are correct and then click Finish to create the project.

2.2 ADDING NEW MODULES

There are 3 types of files you will add: Verilog Modules, Schematics and Verilog Test Fixtures.

Adding Verilog Modules:

1) Go to Project → New Source and then Select Verilog Module. Under *File Name*, give the Verilog module you want to create an appropriate name. * **Note:** names should start with an alphabetic letter (meaning do NOT begin the name with a digit or special character). For this tutorial, call the module "bit1adderv". Click Next.

2) The next window that pops up allows you to specify the inputs and outputs of your Verilog module. You don't have to specify them all at the beginning; you can add more inputs and outputs directly in the Verilog file. You

can add inputs a, b, cin and outputs sum and cout now, but it's usually easier to add them directly in the Verilog file later. Click next. And then click finish.

3) The Verilog file should pop up and edit it from there. You can find details on how to program in Verilog elsewhere. Don't forget to save when you're done.

Your code for bitladderv.v should look somewhat like this:

```
module bitladderv(a,b,cin,sum,cout);  
    input a;  
    input b;  
    input cin;  
    output sum;  
    output cout;  
    wire sum, cout;  
  
    assign {cout, sum} = a + b + cin;  
  
endmodule
```

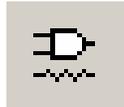
Adding Schematic Modules:

Now you will learn how to make schematics, which basically means you will be creating logic designs down to each single gate and wire.

1) Go to Project→New Source and then Select Schematic Module. Under *File Name*, give the Schematic module you want to create an appropriate name. * **Note:** names should start with an alphabetic letter (meaning do NOT begin the name with a digit or special character). For this tutorial, call the module "bit1adder". Click Next. And then click finish.

2) ECS, Xilinx's schematic program should load. A blank sheet should appear.

To add basic **logic gates** click on the button that looks like the following:



Or you can use the hot-key Ctrl-F

Two windows on the right hand side should appear including all the gates that you can add to the schematic. It should be fairly self explanatory. Press the ESC key when you are done adding symbols/gates.

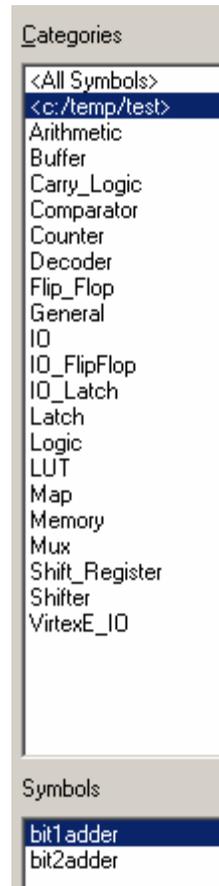
*Note:

Schematic symbols you create within your project can be accessed by selecting the directory in which you placed your project.

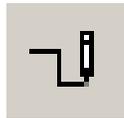
Go back to project navigator. Select the bit1adder.v file, and then double click on the *Create Schematic Symbol* under *Processes for Current Source* inside of *Design Entry Utilities*.

This will allow you to use the block you created in the source file in the ECS schematic editor program. This allows you to reuse a block you created in ECS without having to redraw it over and over again.

In the picture on the right, the project was placed in c:/temp/test and two symbols created for that project from Verilog or schematic sources are named "bit1adder" and "bit2adder".



To add **wires** click on the button that looks like the following:



Or you can use the hot-key Ctrl-W

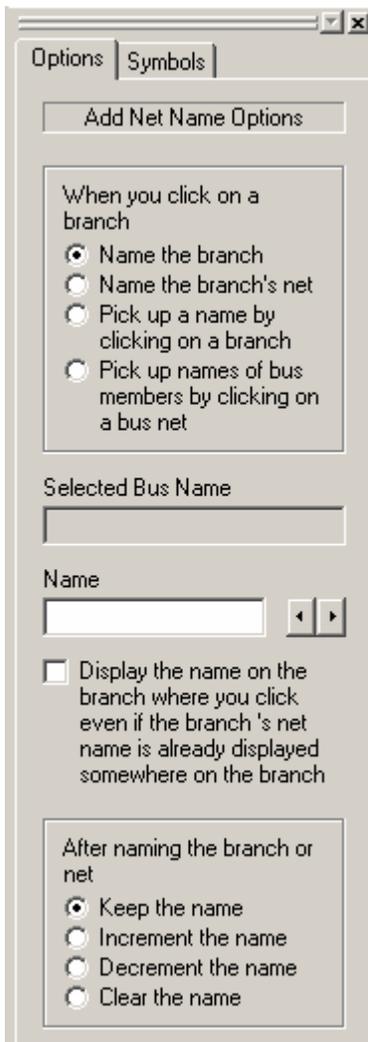
Put your mouse head over the wire you want to begin the wire until you see a red outline and left click. Then drag your mouse and the wire to the net you want to connect it too until you see a red outline and left click. The wire then should be connected. Press the ESC key when you are done. Use can use the same method to add busses.

To **name nets/wires** click on the button that looks like the following:



Or you can use the hot-key Ctrl-D

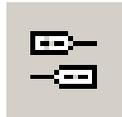
The following window shows up on the left hand side of the screen:



Type the net name in the middle text box:

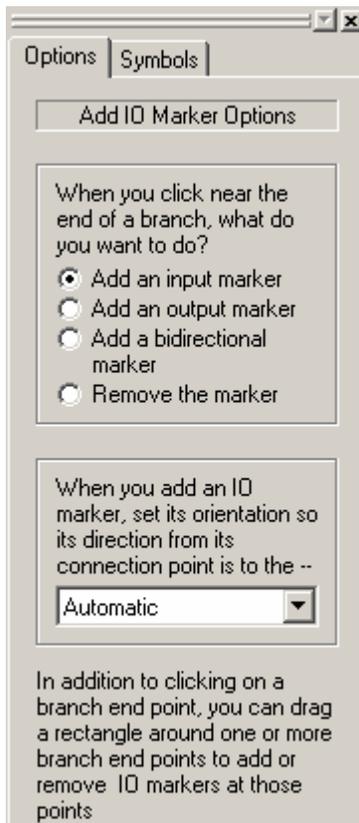
Finally you should see the net name at the end of your cursor when you place the cursor over the schematic window. Place the cursor over the wire/net you wish to name and left click. Press the ESC key when you are done. Note that you can connect pins by giving them the same name, or by physically connecting a wire.

To add **I/O pins** click on the button that looks like the following:



Or you can use the hot-key Ctrl-G

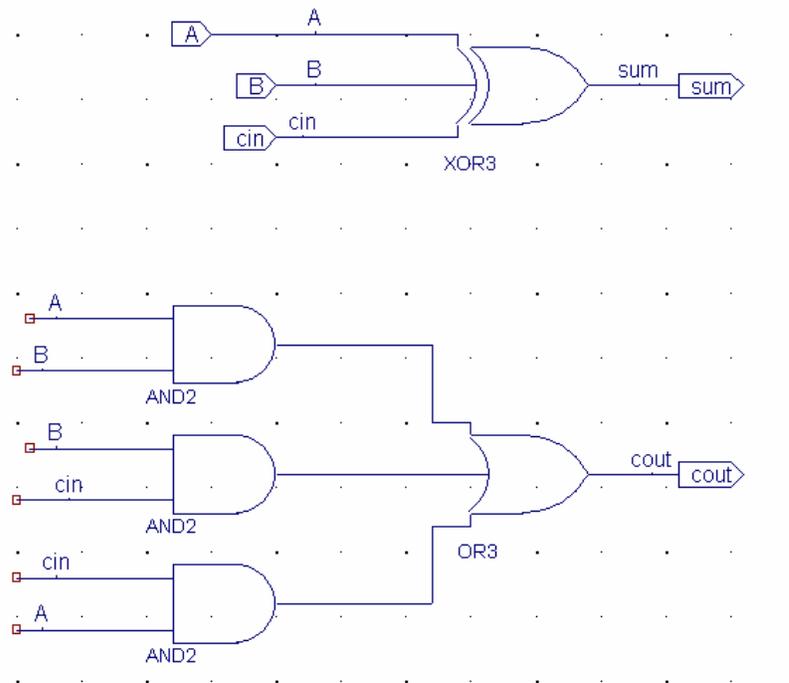
Select the type of pin you wish to add in the box as seen below:



Finally place the cursor over the wire/net you wish to add the pin to and left click. Press the ESC key when you are done.

This should be enough for you to complete a basic design. If you need more help consult the Xilinx ECS manual.

Now after playing around with the tools, create a 1 bit adder that looks like the following:

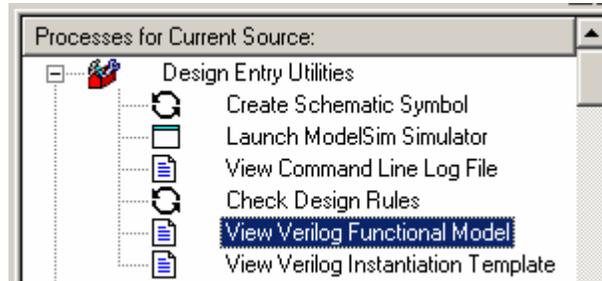


Don't forget to save when you're done.

The schematic file "bitladder.sch" should then appear in the *Sources in Project* window in Xilinx Project Navigator.

To view your schematic in Verilog HDL

Select your schematic file in the *Sources in Project* window. Then in the *Processes for Current Source* window, expand the Design Entry Utilities box. Double click View Verilog Functional Model and the Verilog file should open. The file is read-only though. You can go to *File -> Save As*, and save it as a .v file so that you can add it into Synplify later.



Combining Verilog and Schematics

You can easily combine schematic and Verilog modules in your project. For this tutorial, we will use both 1 bit adders you created in Verilog and in schematic by “wrapping” both adders in a Verilog file to create a 2 bit adder. Basically in Verilog you can reference each schematic module by just referencing the name of the schematic block.

1. Create a new Verilog module as detailed in the previous sections named “bit2adder”.
2. For the appropriate inputs and outputs, don’t forget some are 2 bit busses.
3. Your code should look somewhat like the following:

```
module bit2adder(a,b,cin,sum,cout);
    input [1:0] a;
    input [1:0] b;
    input cin;
    output [1:0] sum;
    output cout;
    wire ctemp;
    // note that the input output position of both adders are different
    // cout precedes sum out in bitladder though the opposite occurs for bitladderv
    bitladder ba (a[0], b[0], cin, ctemp, sum[0]);
    bitladderv bav (a[1], b[1], ctemp, sum[1], cout);
endmodule
```

4. Don’t forget to save the file.

2.3 SIMULATING YOUR DESIGN

In this course, you can also use ModelSim to simulate your designs. It allows you to run testbenches and it can generate waveforms for you to view your design’s outputs. We will only be covering one of many ways to use ModelSim below:

Simulating your design using a testbench module.

First, we need to create a testbench. In *Project Navigator*, add a new Verilog file called bit2adder_tb.v. It should look something like this:

```

`timescale 1ns/1ps

//note the backwards apostrophe before the t in
//timescale--it's the button to the left of the number 1 on the keyboard
//1ns refers to how fast time goes in this file. i.e. #2 means 2 nanoseconds
//1ps refers to the resolution at which things are calculated

module bit2adder_tb();

    reg [1:0] a, b;
    reg cin;
    //inputs to the module you're testing should be regs
    //this way, you can actually assign them a value

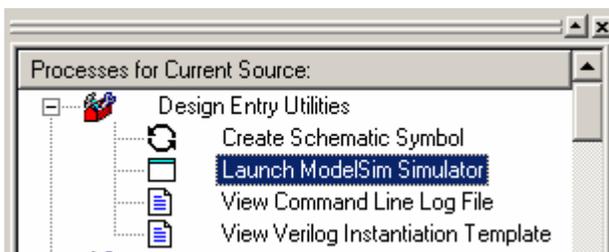
    wire [1:0] sum;
    wire cout;
    //outputs to the module you're testing should be wires

    bit2adder myAdder(.a(a),
                     .b(b),
                     .cin(cin),
                     .sum(sum),
                     .cout(cout));

    initial begin
        a = 2'b00;
        b = 2'b00;
        cin = 1'b0;
        #10
        a = 2'b01;
        #10
        b = 2'b10;
        #10
        cin = 1'b1;
    end
endmodule

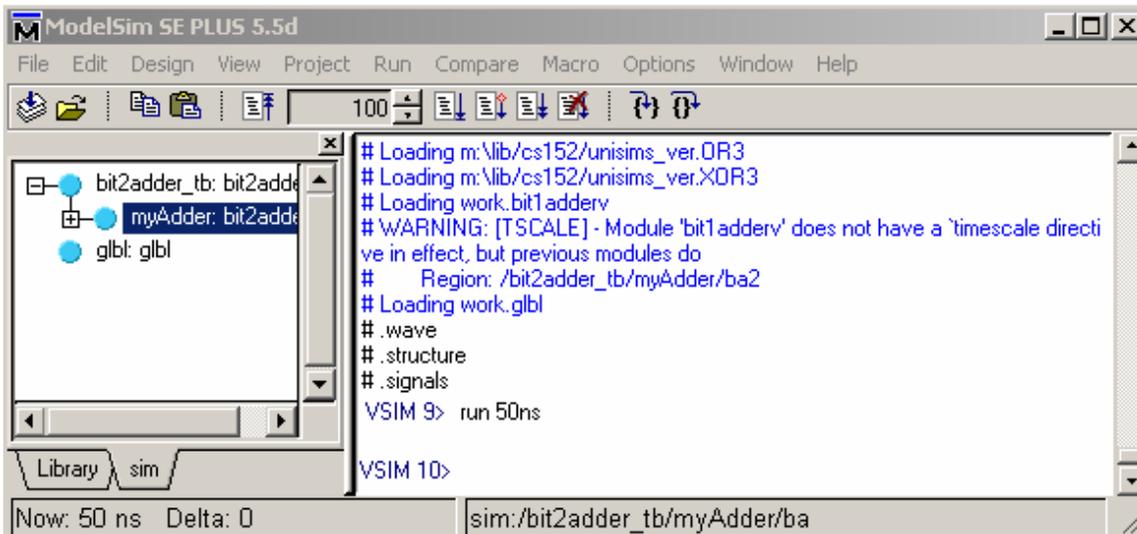
```

Now, we can begin testing. First select your previously created testbench source in the *Sources in Project* window.



In the *Processes for Current Source* window, expand the *Design Entry Utilities*, and double click *Launch ModelSim Simulator*.

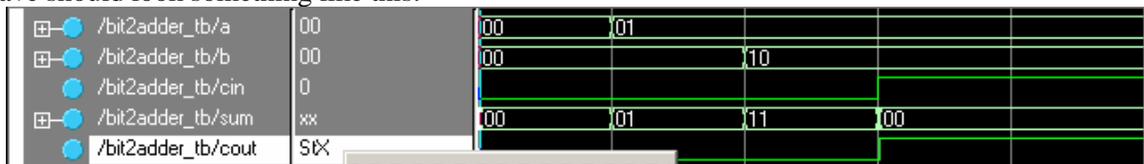
This will launch ModelSim which will load your testbench (simulate the input values you set in the testbench). You then need to tell ModelSim to actually run your code, by typing in *run 40ns*. You can specify any amount of time (use ps, ns, or us).



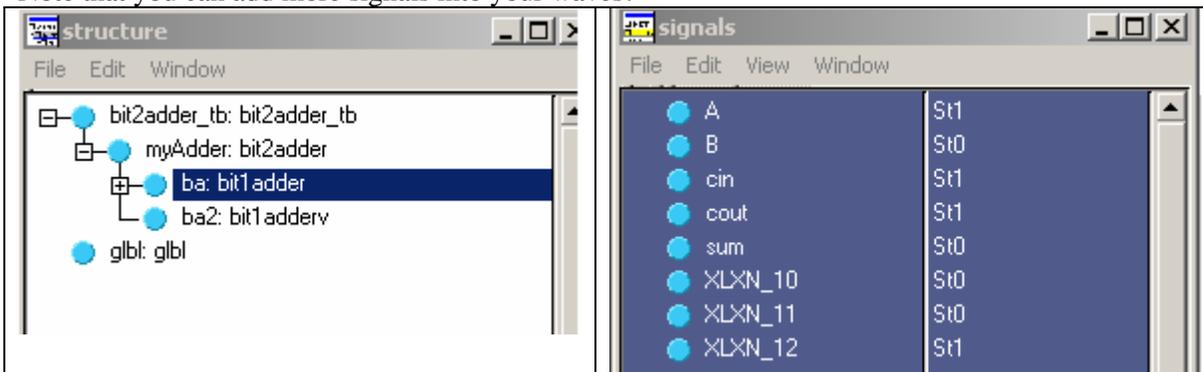
The resulting waveforms in the wave window may not be easily seen b/c the time interval between clock cycles may be too long or too short. To remedy this you can either use the zoom out tool (the magnifying glass with the plus sign) to shorten the length between clock cycles and view more of the wave at once, or the zoom in tool to do the opposite. You can also click on the very right magnifying glass to zoom full.



Your wave should look something like this:



Note that you can add more signals into your waves!



Under the structure box, you need to select where you want your signals to be coming from. In the above example, we want the signals internal to the ba instantiation of the bit1 adder. The signals box on the right contain the actual signals themselves. Click and drag the ones you want into the wave diagram.

Now you have the ability to write a testbench which will allow you to run your design, and see what each and every signal is at any moment in time. This is a very powerful debugging tool, and is one that you will put to heavy use in the weeks ahead.

A final note: If you right click on a signal inside the wave diagram, a box with plenty of options opens up. For example, you can choose to display the signal's value as binary, hex, or unsigned numbers, among others. Use these options to aid you in your debugging. We encourage you to play around with Modelsim, as there are many, many, more features that are not discussed here.

