The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 541 Digital Logic and Computer Design
Fall 2014

Lab #2: Hierarchical Design & Verilog Practice
Issued Wed. 8/27/14; Due Wed. 9/3/14 (11:59pm)

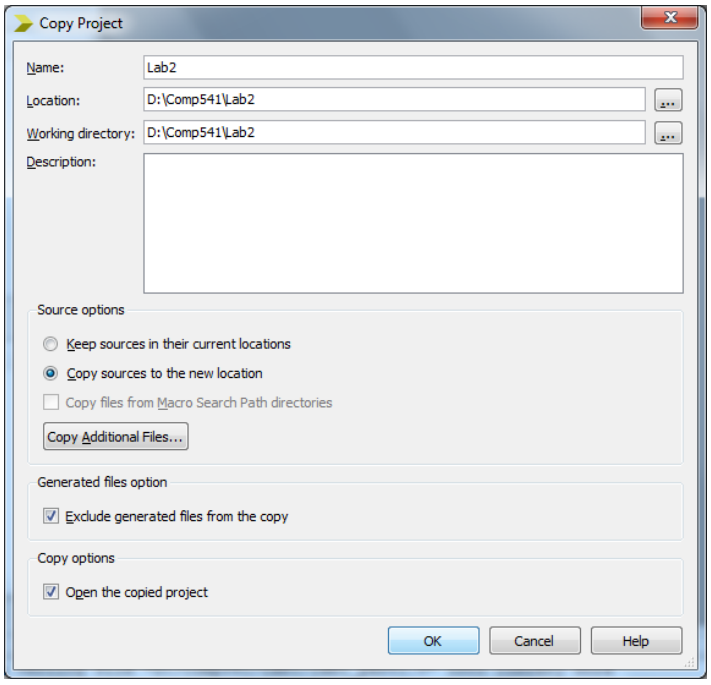
This lab assignment consists of several steps, each building upon the previous. Detailed instructions are provided, including screenshots of many of the steps. Verilog code is provided for almost all of the designs, but some portions of the code have been erased; in those cases, it is your task to complete and test your code carefully. Submission instructions are at the end.

You will learn the following:

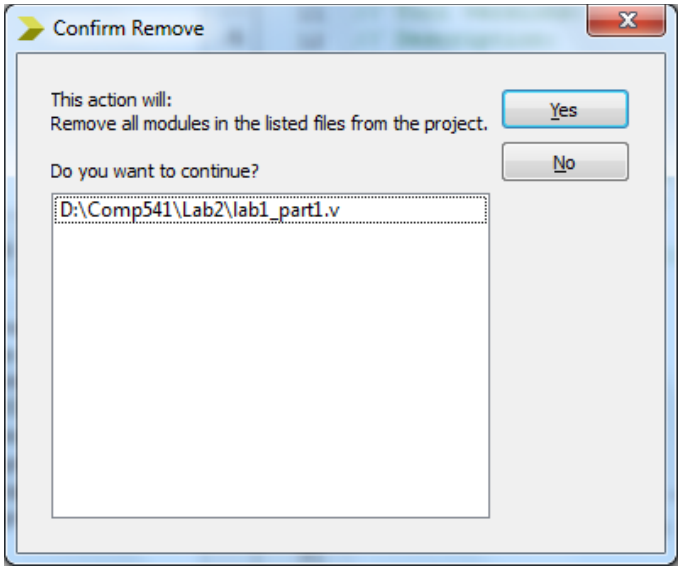
- Navigating the ISE development environment
- Designing a hierarchical system, with multiple module types
- Working with buses (multi-bit values)
- Verilog test fixtures and stimuli, including printing and monitoring
- Verilog simulation, including the graphical viewer

Make a New Project

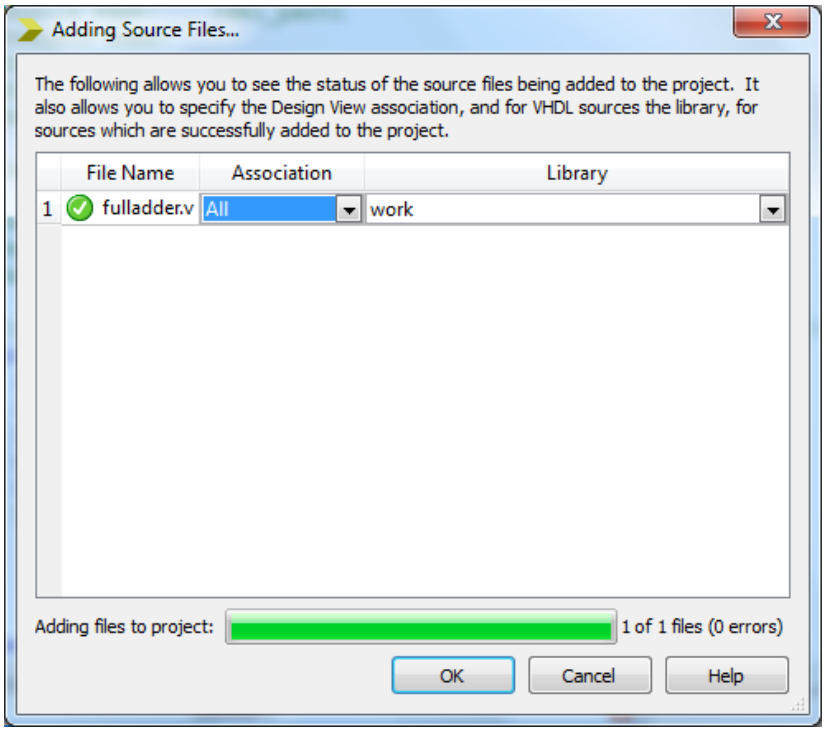
Start the Project Navigator, and open the Lab 1 project if not already open. Make a copy by clicking *File* → *Copy Project* (choose Lab2 as its name). Be sure the last two checkboxes are checked.



Double-click on the file lab1_part1 to open it. Save it under a new name: *File* → *Save as* → fulladder.v. Now right-click on lab1_part1.v and click *Remove*.



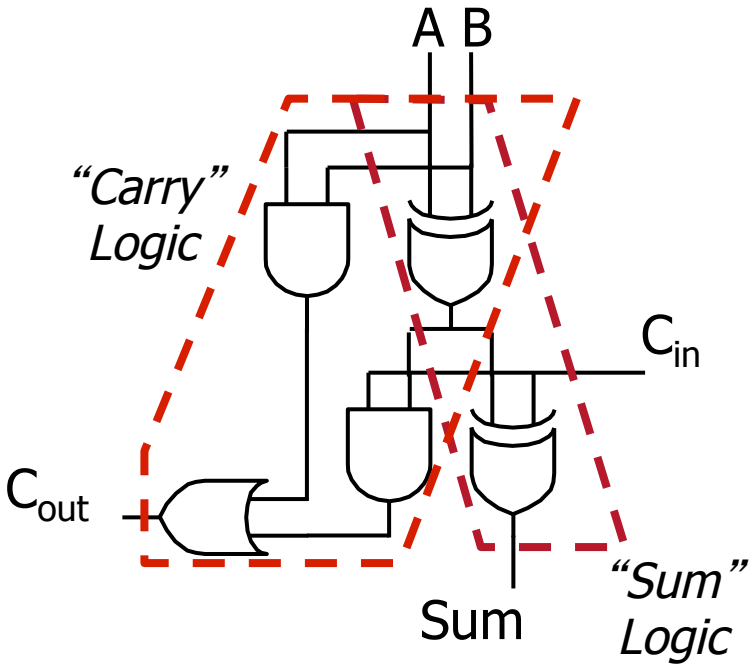
Click Add Source (below New Source), and add the file fulladder.v to the project. (This seems unnecessarily long, but is the only way to rename a file in the project!) Within this file, rename your module name to "fulladder."



Select Simulation view, and remove lab1_test (we will make a new test fixture in this lab). Now go back to Implementation view.

(Now from the Windows/Linux file explorer, you can delete the files lab_part1.v and lab1_test from the Lab2 folder.)

For your reference, here is once again the circuit and Boolean equations for a Full Adder (from Comp411).



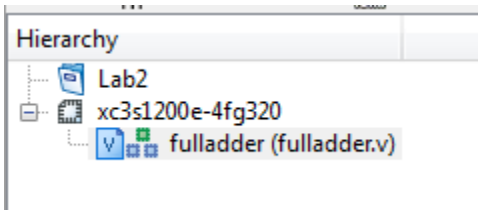
$$C_{out} = C_{in}\left(A \oplus B\right) + AB$$

$$Sum = C_{in} \oplus A \oplus B$$

Make sure the description of the module is *exactly* as follows. Note that we have added an intermediate signal *X* that computes *A xor B,* which is then *shared* by the sum logic and the carry logic (thereby saving a gate).

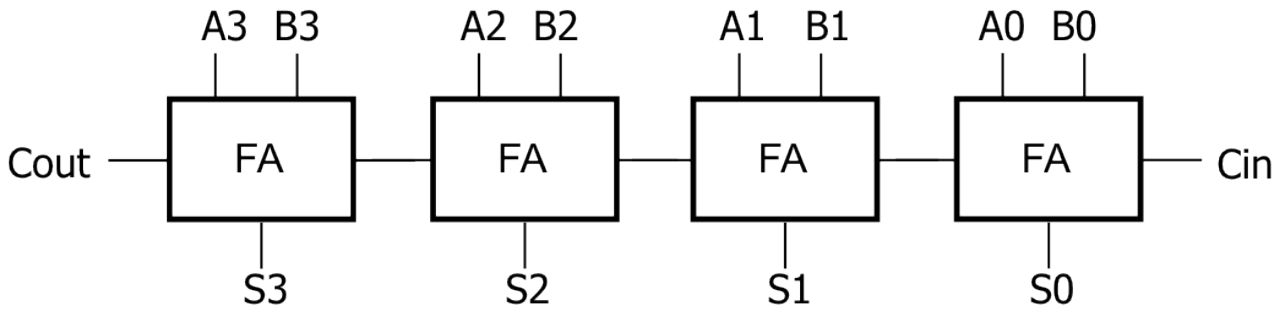
```
module fulladder(
    input A,
    input B,
    input Cin,
    output Sum,
    output Cout
    );

      wire X;
      assign X = A ^ B;
      assign Sum = Cin ^ X;
      assign Cout = (Cin & X) | (A & B);

endmodule
```

Save the file **`fulladder.v`**. At this point, the hierarchy should look like this:



Designing a 4-bit ripple-carry adder

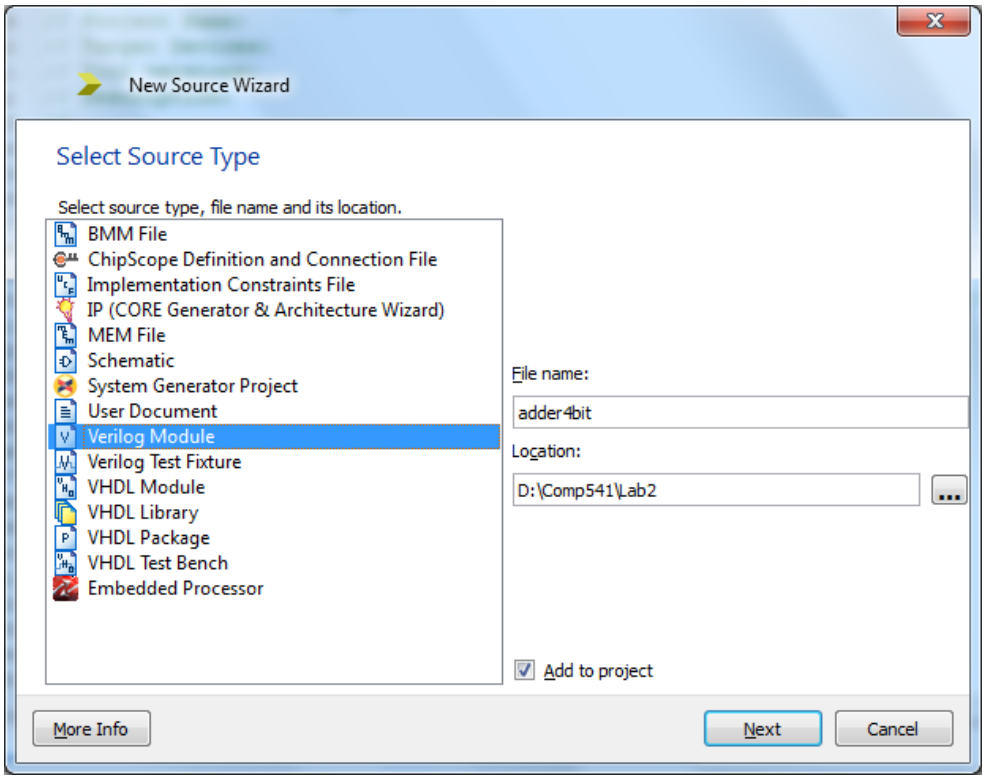
Let us now design a 4-bit ripple-carry adder by stringing together four full adders (FAs). The diagram of a 4-bit adder (again, from Comp411) is shown here for reference.

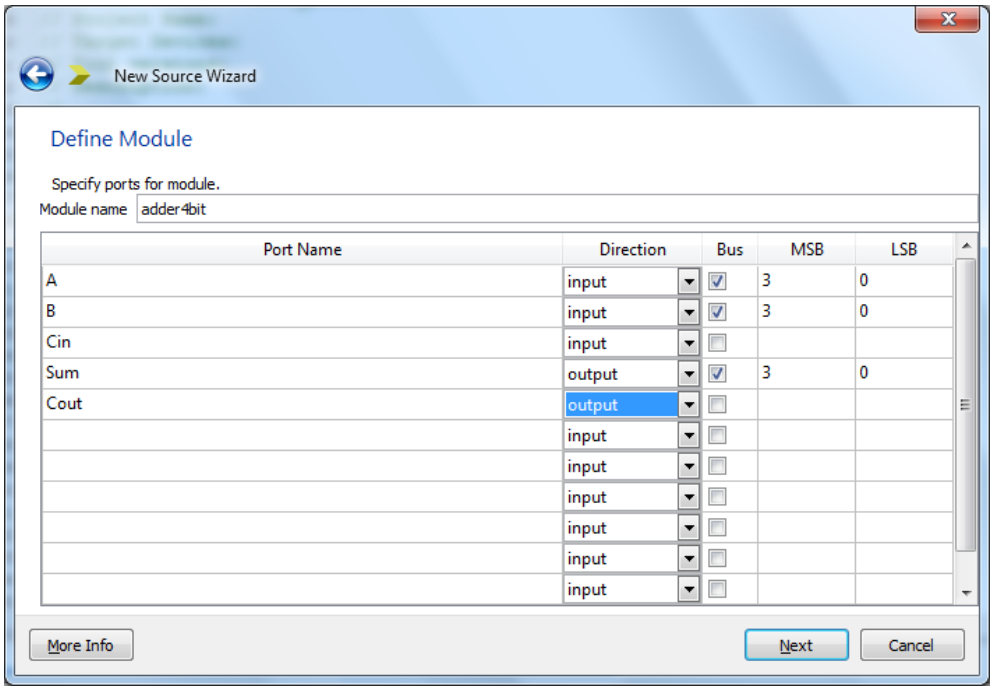


The corresponding Verilog code is shown here, but portions of it have been obscured. Please fill in appropriately. (You do not need to add any extra lines of code; just fill in the missing details into what is provided.)

```
module adder4bit(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] Sum,
    output Cout
    );

   wire C1, C2, C3;
   fulladder a0(A[0], B[0], Cin, Sum[0], C1);
   fulladder a1(A[1], B[1], [redacted], Sum[1], [redacted]);
   fulladder a2(A[2], B[2], [redacted], Sum[2], [redacted]);
   fulladder a3(A[3], B[3], [redacted], Sum[3], [redacted]);

endmodule
```

Before you can enter this code, you will need to create a new source file. See the following screenshots for what to enter into the dialog boxes:

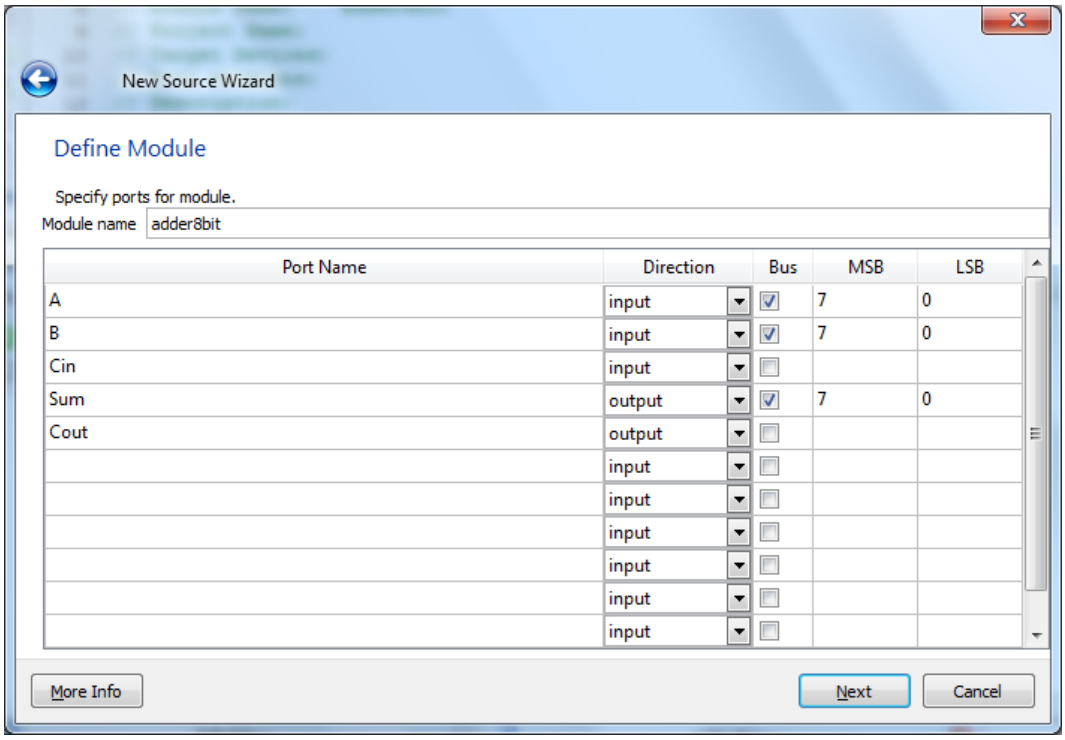




Create your own test fixture to test your 4-bit adder. *Really, do test your 4-bit adder before moving on!*

Designing an 8-bit ripple-carry adder

Now we will design an 8-bit adder using two 4-bit adders. The procedure is very similar: create a new source file, and this time use "adder8bit" as the name for the module, and specify the ports as follows:



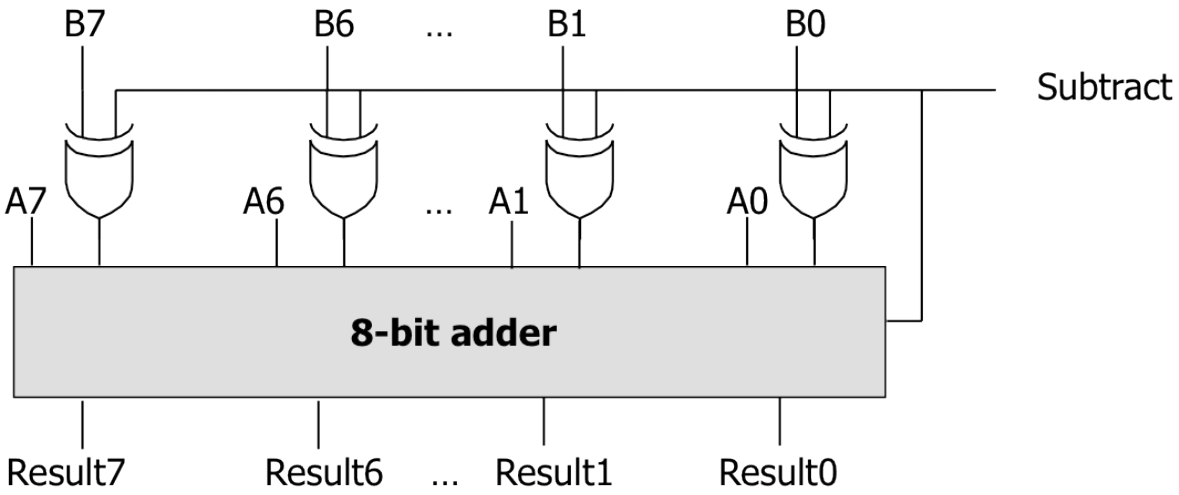
Use the following code, and fill in the missing details.

```
module adder8bit(
    input [7:0] A,
    input [7:0] B,
    input Cin,
    output [7:0] Sum,
    output Cout
    );

   wire C3;
   adder4bit A0(A[3:0], B[3:0], Cin, Sum[3:0], [redacted]);
   adder4bit A1([redacted], [redacted], [redacted], [redacted], [redacted]);

endmodule
```

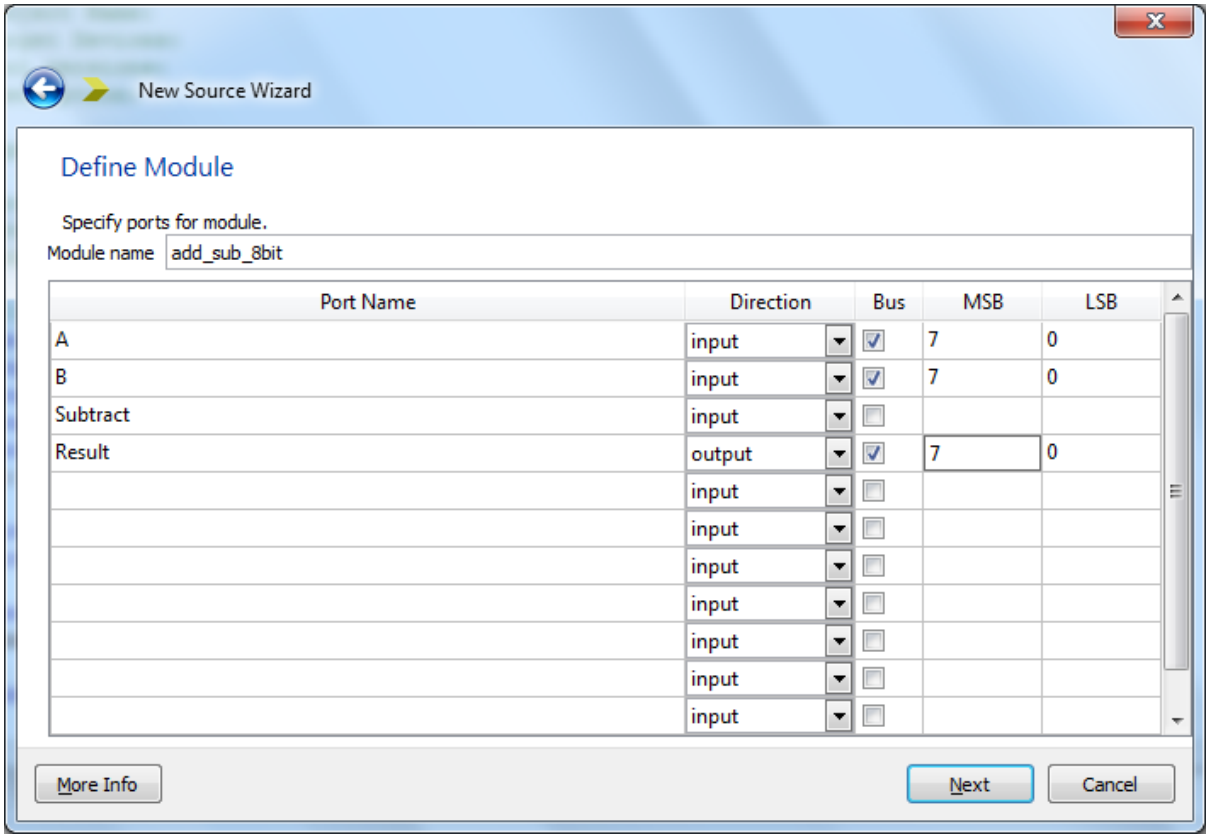
Once again, create your own test fixture to test your 8-bit adder. *Really, do test it before moving on!*

Designing an 8-bit Adder-Subtractor

Now you will design a circuit that can perform 8-bit additions as well as subtractions. That is, given *A* and *B,* the circuit will produce either the sum *A+B,* or the difference *A-B,* depending on whether the value of a Boolean input *Subtract* is 0 or 1, respectively. This circuit was also covered in Comp411, but is repeated here for reference.



Once again, you will create a new source file, with the name add_sub_8bit.v, and the following ports:



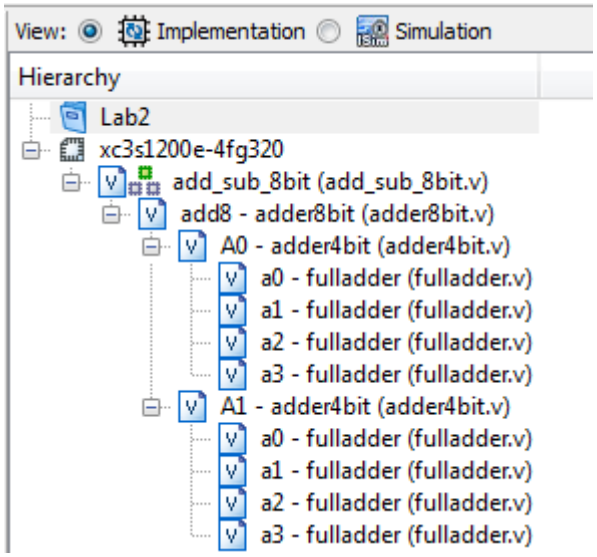
Note that there is no C_{in} and no C_{out}.

Use the following code, and fill in the missing pieces:

```
module add_sub_8bit(
    input [7:0] A,
    input [7:0] B,
    input Subtract,
    output [7:0] Result
    );

   wire [7:0] ToBornottoB;
   wire Cout;

   assign ToBornottoB[7:0] = (Subtract) ? [redacted][7:0] : [redacted][7:0];
   adder8bit add8(A[7:0], ToBornottoB[7:0], Subtract, Result[7:0], Cout);

   // It is also okay to write it as follows, but there is less chance
   // of error while coding if you use the above version

   // adder8bit add8(A, ToBornottoB, Subtract, Result, Cout);

endmodule
```

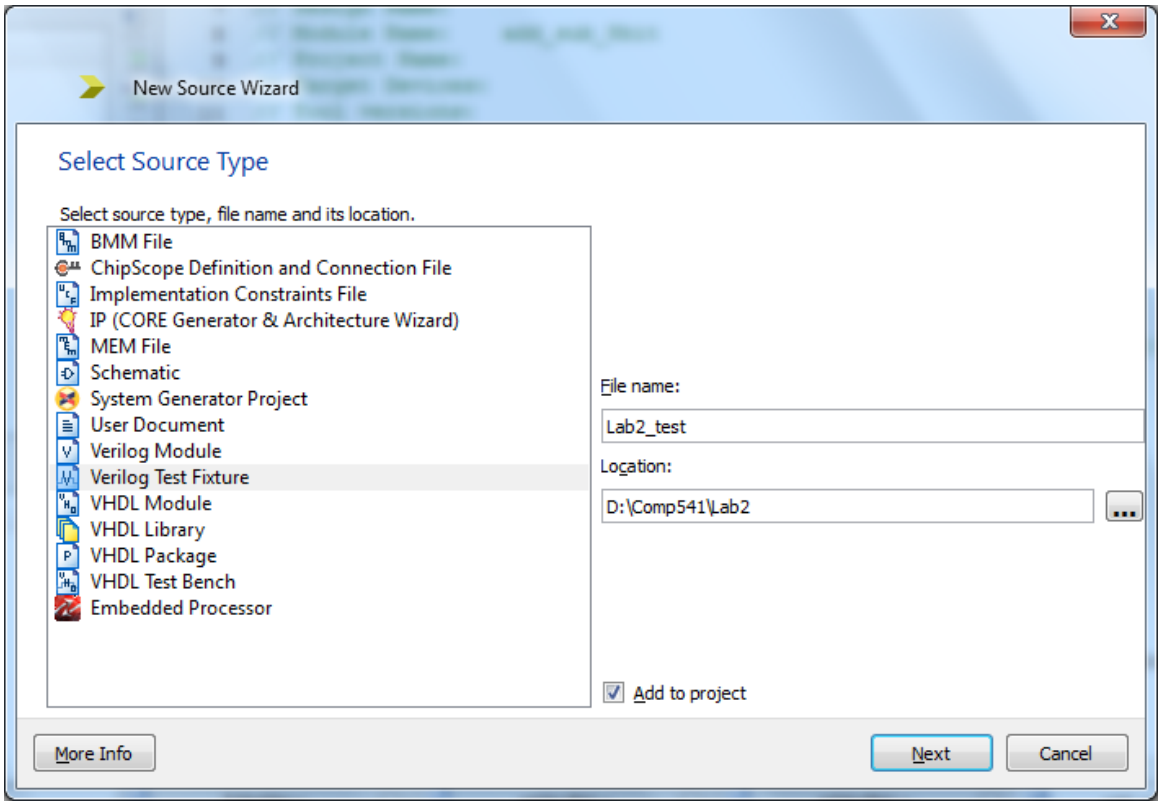
Note that while the 8-bit adder has a carry out, the add_sub_8bit module does not send it out! Also, observe carefully what the carry in of the adder is connected to.

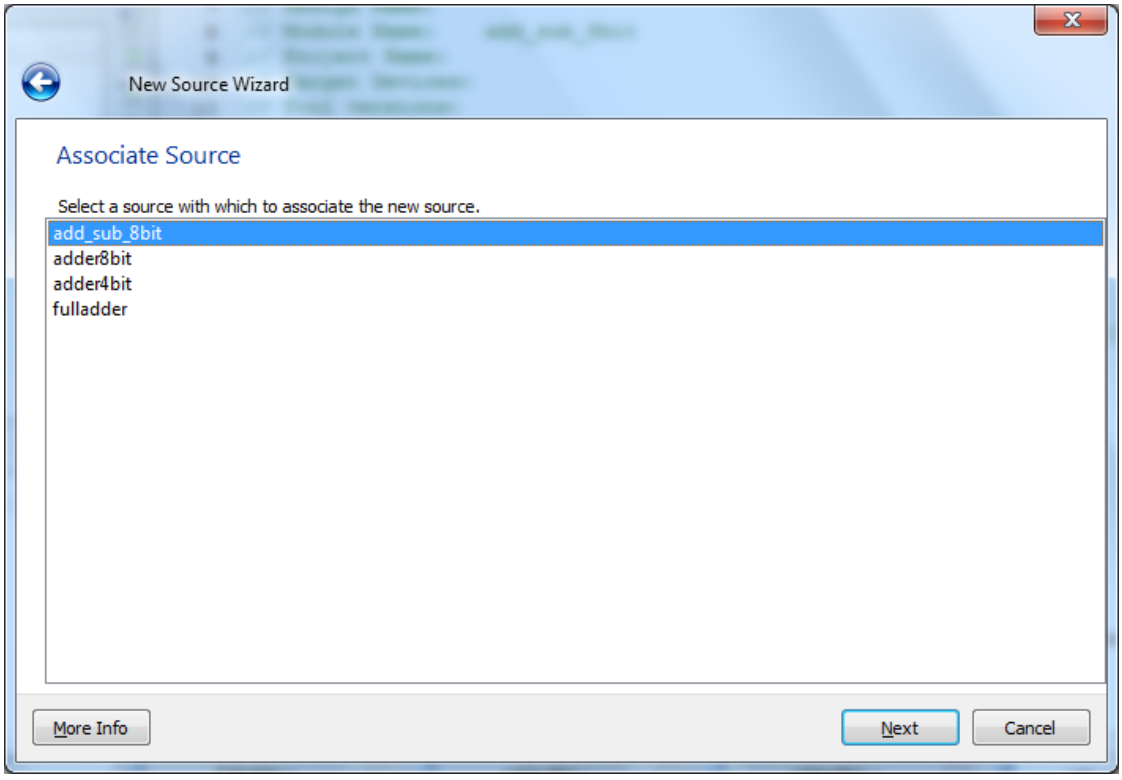
Save the file, and take a look at the hierarchy; it should look exactly like this when you expand all the nodes:



Verilog Test Bench

Create a new source file, and select *Verilog Test Fixture* as its type, and name it Lab2_test, as shown.





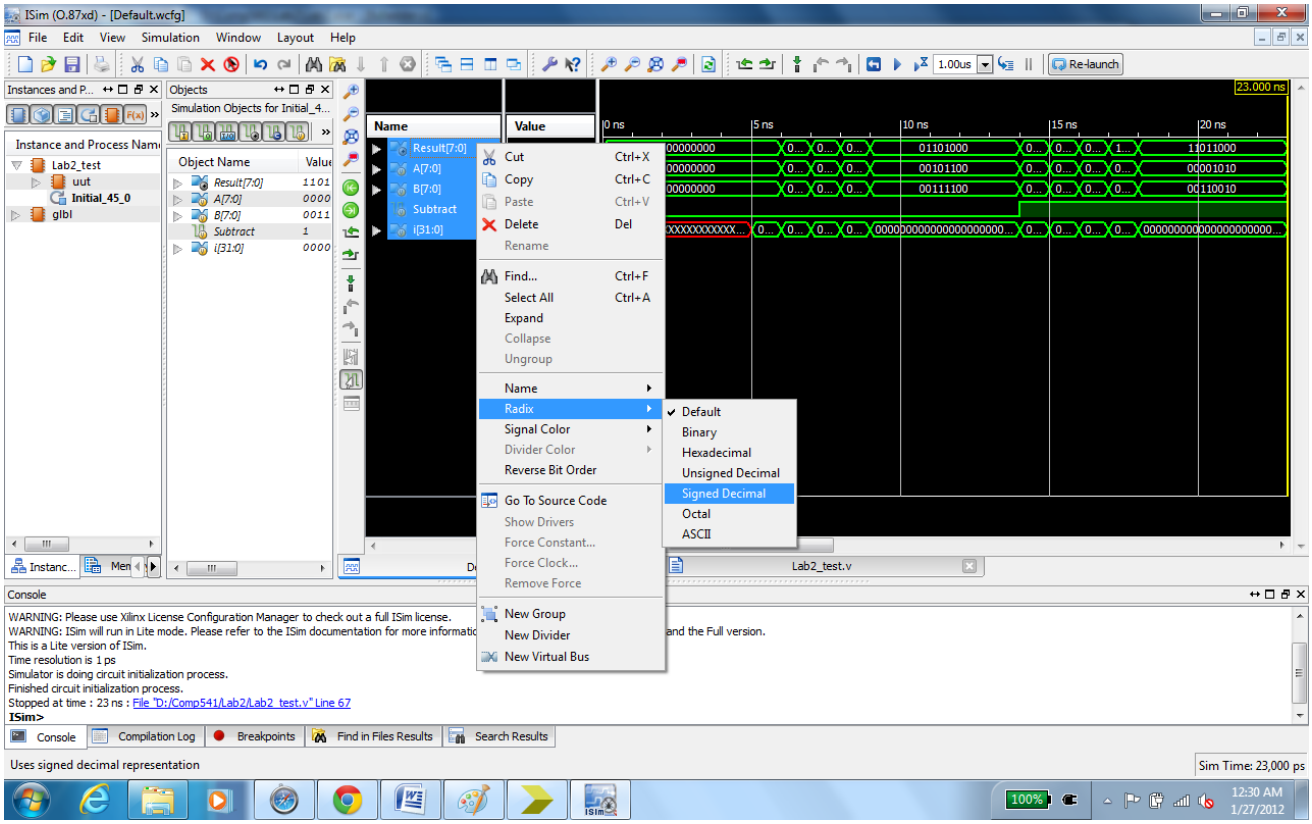
Download the code for the test fixture from the website (you can copy-and-paste it into the stub that the tool automatically creates for you).

Carefully go through every line of the test bench, and make sure you understand it! Refer to the online Verilog reference linked from the class website.

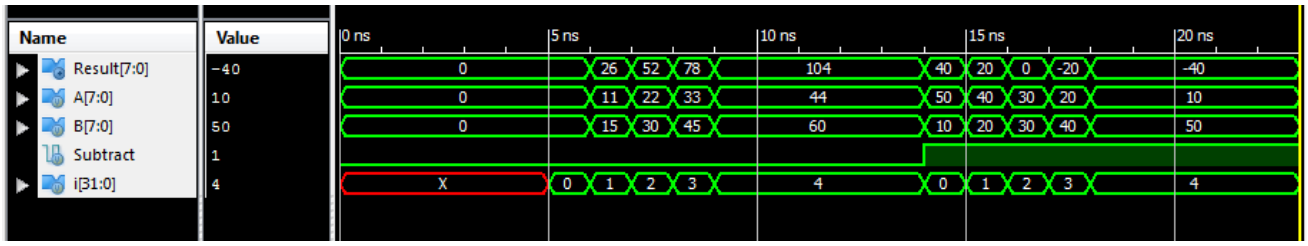
Verilog Simulation

Change to Simulation view (instead of Implementation view), and click to select Lab2_test underneath, and then double-click *Simulate Behavioral Model.* The simulator ISim will launch, and the results should look like the picture below (click *Zoom to Full View*).

Since it is hard to make sense of all the 0's and 1's, select all the signals under *Name* (use shift-select), right-click, choose *Radix,* and select *Signed Decimal.*

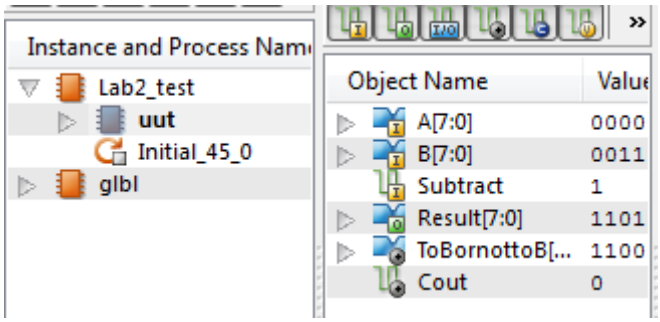


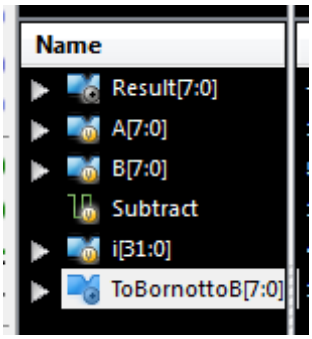
You should now see the simulation outputs in decimal.



Look through them carefully to make sure they are correct.

Now, let us display the bus *ToBornottoB* that is inside the add_sub_8bit module. Go to the left, select **uut,** and you will see the objects and wires inside it. Click and drag *ToBornottoB[7:0]* into the *Name* column in the waveform window:





Click *Simulation* → *Restart,* and then *Simulation* → *Run All.* This time the value of *ToBornottoB[7:0]* is also displayed in the waveform window. Right-click *ToBornottoB* in the name column, and change its radix back to *Binary.* Also, change the radix of *B* back to *Binary.* Observe that they are identical for the first half of the simulation, and bitwise complements during the second half.

This exercise showed you how to examine objects and wire that are not at the top level, but down the hierarchy.

Also, observe that you can click at a particular time instant in the waveform window. The *Value* column is updated to show the values of all the signals at that time instant. There are also other buttons available for zooming in/out, skipping to next transition, etc.

Why does the value of *i* appear as *X* for the first 5 ns?

Using Display and Monitor Commands

Look at the bottom of the test fixture. You will see commands using **$time, $timeformat,** and **$monitor.** The **$monitor** command tells the simulator to print a message whenever any of its arguments (except for **$time** itself) changes value. The output appears in the horizontal panel at the bottom of the simulator.

Please refer to the online Verilog reference website for details on these commands, and make sure you understand them well! You should also look into the **$display** and **$write** commands.

What to submit: **A screenshot of the *ISim* window clearly showing the final simulation result of the adder-subtractor, i.e., with *ToBornottoB[7:0].***

- **The values of *ToBornottoB[7:0]* and *B[7:0]* should be shown in binary.**
- **The values of *Result[7:0], A[7:0], Subtract,* and *i* should be shown as signed decimals.**

How to submit:

- **Send email to: `comp541submit-cs@cs.unc.edu`, with "Lab 2" in the subject line.**
- **Attach the simulator screenshot using the filename `waveforms.png` (or other appropriate extension).**
- **Submit your work by 11:59pm on Wednesday, September 3.**
