CSE 352 Laboratory Assignment 2

Constructing Simple Logic Circuits - II

Objectives

In this laboratory assignment you will continue to learn how to use the Aldec Active-HDL tools. This will focus on how to create Verilog modules. You will also see how Verilog modules can be used as test fixtures to help test and verify your circuits. You should use the lib370 gates for all your circuits. These gates include delay, unlike the built-in gates, and this will allow us to look at the circuit delay in simulation.
Save all the files that you create in this lab. We will use them later.

Laboratory Procedure

Task 1

Complete Tutorial #2 to familiarize yourself with test fixtures and learn how to use them to test a circuit in Active-HDL. The second part of the tutorial describes how to use buses and bus input and output terminals. In this part, you will design four-bit adder using the one-bit full adders you created in Lab 1. Test your four-bit adder using this test fixture: addsub4_tf.v as described in the tutorial.

Note that in Tutorial #2 the four-bit adder is made with the pink lib370 gates. These gates model delay, unlike the built-in yellow gates. Make sure you use the lib370 gates or you will not be able to answer the check-off question.

Task 1: Check-off Requirements

1. Show your working full adder simulation and test fixture for the single full adder.
2. Show your working 4 bit adder/subtractor module block diagram, and simulation output verifying that it works.

Task 2

Complete Tutorial #3 , which how to write simple Verilog modules and use them in schematics. As part of the tutorial, you will write and test the Verilog module for a full-adder; a full adder is one that handles both carry-in and carry-out conditions in addition to the sums and two inputs. You should create a test schematic by using the provided test fixture: FA_tf.v (right click and "save as" if the download doesn't start automatically). Test fixtures will be covered in more detail later so don't worry too much about how it works. However the basic premise is the test

fixture generates input signals for the circuit and checks to see if the output signals have the correct values. An example of how to hook up your test fixture is shown below.

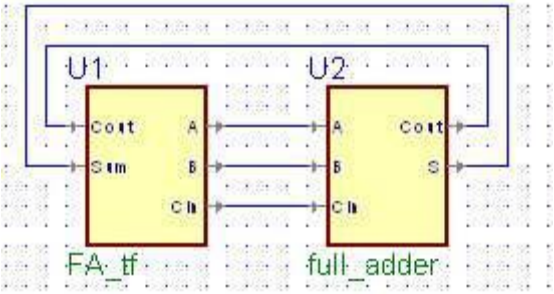

Figure 1 - Hooking up a test fixture

To test to see if your full adder works you do not have to manually create signals and simulate it in Active-HDL; the test fixture generates all the signals for you. All you need to do is run the simulation and the test fixture will pass a series of values in to your full adder and check the results. If all goes well the console at the bottom will inform you that your full adder passed. If there are errors it will let you know as well.

Warning: Do not put complete faith in test fixtures. You should always practice common sense and make sure that the values that come out make sense. Text fixtures cannot possibly test all cases for complex circuits, but they do increase you confidence that the circuit works correctly. Also, you may see 'X' values, which are a good indication that something is not working correctly.

Now replace the schematic-based full-adder in the 4-bit adder of Part 1 with the Verilog full-adder you made in Part 2. Test your circuit in simulation using the same addsub4_tf.v test fixture.

Check-off Requirements

1. Show your Verilog source code for the full adder and the test fixture results.

2. Show your test fixture simulation output of the 4 bit adder/subtractor using the Verilog full adder

Task 3

Construct an 8-1 multiplexer on your breadboard using two 4-1 multiplexers (the '153 chip) and one 2-1 multiplexer (the '157 chip). (You may use 4 2-1 and one 4-1 multiplexer if you like.) Wire the 8 inputs of your multiplexer to the first 8 switches, and wire the 3 select signals to first three keys. You should of course wire the 0 input to switch 0, etc., if you want to stay sane. Connect the output of your 8-1 multiplexer to one of the LEDs. Don't forget to connect VDD and GND to the chips.

Test your multiplexer by going through the inputs one at a time. Press the appropriate keys and make sure that the output follows the inputs. For example, with keys 0 and 1 ON, switching SW 3 should make the light blink, and switching any other switch should have no effect. (Recall from Lab 1 how the switches work – you may want to use inverters on the switch outputs.)

Check-off Requirements

1. Show your working 8:1 multiplexor on the board. Demonstrate that it works by switching through the inputs.

Task 4

You have now implemented a circuit that can implement any function of 3 inputs. The idea that we can build a circuit that can be "reconfigured" to implement any function is huge: it's the basic idea that FPGAs are built from. The 3 select inputs of the 8-1 multiplexer are the inputs to your reconfigurable function, and you "configure" the function that it implements by setting the 8 switches to on or off according to the function's truth table.

Start by implementing a 3-input AND gate: test by trying all combinations of the 3 inputs.

*Make sure you get the 3 input AND checked off before you disassemble your circuit to implement the next part.

Now reconfigure your circuit to implement the full-adder carry function.

*Make sure you get the full adder checked off before you disassemble your circuit to implement the next part.

Finally, reconfigure your circuit to implement a 2-1 multiplexer. Use input 0 and 1 as the 0 and 1 inputs of the 2-1 multiplexer and input 2 as the select input of the 2-1 multiplexer. (Using an 8-1 multiplexer to implement a 2-1 multiplexer may sound weird, but remember, it's not an 8-1 multiplexer any more, it's a reconfigurable 3-input function.)

Check-off Requirements

1. Show your working 3 input AND gate using an 8:1 multiplexor on the board. Demonstrate that it works by switching through the inputs.
2. Show your working full adder using an 8:1 multiplexor on the board. Demonstrate that it works by switching through the inputs.
3. Show your working 2:1 multiplexor using an 8:1 multiplexor on the board. Demonstrate that is works by switching through the inputs.

Comments to:moreau@cs.washington.edu