EECS 470 Lab 2 Assignment

Note:

- The lab should be completed individually
- The lab check off is due **Friday, January 26th**

1 Introduction

In this lab assignment, you will practice writing testbenches for modules. We will be using 1-bit and 64-bit full-adder modules as examples, and will be writing comprehensive and non-comprehensive testbenches for each respectively.

Some modules are simple enough that you can write comprehensive testbenches that can test every possible input and output. But often you will have to strike balance between a thorough testbench and the amount of time required either to generate the testbench or for the testbench to run.

2 1-Bit Adder: A Comprehensive Testbench

For this section, we'll explore writing a comprehensive testbench for a 1-bit adder module. Full-adders have 3 inputs: two inputs to add and the carry input; for a 1-bit adder, there a $2^3 = 8$ possible input combinations to test.

We've supplied you with a slightly obfuscated full-adder (`full_adder_1bit.sv`) and an incomplete stub testbench (`full_adder_1bit_test.sv`). The full-adder has a bug in it, which you will find by first fixing the testbench. The testbench already computes the correct sum and fails the module if an incorrect value is output. To finish the testbench you need to do two things:

1. Wire up the module – Open the testbench and finish wiring the signals into the module ports.
2. Write the comprehensive tests – Add assignments to the `initial` block and add eight tests to handle every combination of inputs for the module.

Once you've done this, run the testbench with `make sim` and identify and fix the bug in `full_adder_1bit.sv`.

3 64-Bit Adder: A Harder Testbench

Using the 1 bit adder, you will now build a 64-bit adder. The file `full_adder_64bit.sv` contains a module stub: `full_adder_64bit`.

Use an array of 1-bit adders to complete the module as described in the lab presentation. (This should take no more than 5 minutes; feel free to get help from a neighbor!)

Then, update the Makefile to compile this new module (change "`1bit`" to "`64bit`" in the `TESTBENCH`, `SOURCES`, and `SYNTH_FILES` Makefile variables).

3.1 Where Comprehensive Tests Fail

We have provided you with a slightly more complete testbench in `full_adder_64bit_test.sv`. It initially implements a comprehensive testbench similar to what you wrote for the 1-bit adder, but you're going to need to change it. To understand why, first just run the testbench:

Run the testbench with `make sim`.

Hint: Use `Ctrl-\` to force-quit an unresponsive simulation.

Now read through the testbench and try to understand what went wrong. It has a lot of comments, but you don't need to read them all. You will find a pair of for-loops that implement the comprehensive testing.

Read through them and understand why they won't work for a testbench written by humans with finite lifetimes. Feel free to discuss with a neighbor.

Comment out the for-loops in the testbench.

Read the "`Test Specific Edge Cases`" section, and add tests there. Try to test edge cases or interesting values. The most interesting values in any test are 0, 1, -1, and infinity (aka MAX). Add a few tests for combinations of those and for some other simple numbers.

You also need to fix the `compare_correct_sum` task in the testbench. Look at the 1 bit testbench and try to see what's missing in the 64-bit test.

Run the testbench and make sure your module passes.

3.2 Synthesizing Your Design

Synthesize your 64-bit adder and compile it with your testbench (`make syn`).

Once you've run synthesis, run `make slack` to grep the generated .rep file. This will find the reported "slack" for the module. It's been violated! For the check-off you should be able to answer what this means and how you can fix it. Fix it, and be able to show that slack is no longer violated.

Ensure that your 64-bit testbench also gives a pass to the synthesized module.

4 Submission

For lab 2, be ready to show your 1-bit testbench, your fix to the 1-bit module, your 64-bit module, your 64-bit testbench, and the output of `make slack` on the 64-bit module.

Place yourself on the help queue during lab or office hours once you're confident you've completed the lab satisfactorily. Turn in your check-off to Gradescope by the end of the day of next week's lab.