EECS 470 Lab 2 Assignment

Note:

- The lab should be completed individually.
- The lab must checked off by a GSI before the end of next week Friday's lab.

1 Assignment

1.1 Writing a Robust Testbench

Some modules are simple enough that you can write a testbench with complete coverage. Often times, you will have to strike a balance between a thorough testbench and the amount of time required to generate the testbench, or for the testbench to run. We have supplied you with a slightly obfuscated full-adder (`fa1.v`) and a stub testbench (`fa1_test.v`). You need to write `fa1_test.v` such that it achieves complete coverage of the full-adder module. Once you have done this, it should be a quick fix to identify and fix the bug in `fa1.v`.
Note: to compile this lab, we need a Makefile, but we weren't provided with one. Think about what you learned today in lab. Write your own Makefile (or modify the one provided in Lab 1). We'll also need a TCL script since that wasnt provided either.

1.2 Using Arrays of Modules

Using the 1-bit adder from Part A, build a 64-bit adder as shown in the presentation. The included file named `fa64.v` contains a module stub `full_adder_64bit`. Make the module into a complete 64 bit adder. (This should take no more than 5 minutes, but if it does, please ask around for help and discuss with your neighbors.)

1.3 Writing a Harder Testbench

We have provided you with a slightly more complete `fa64_test.v`. As a first step, read through the code we provided and make sure you understand it. You will need a lot of this material for Lab 3.
Try running the testbench. What's happening? Why? Read the testbench and understand what it's doing.
Hint: Use `Ctrl-\` to kill an unresponsive simulation.
Since our testbench won't work as written, what kind of tests could you add to achieve good, but imperfect coverage? Modify `fa64_test.v` to be a useful testbench and make sure your `fa64.v` implementation passes. Specifically, you need to add some specialized test cases. What specific test cases are useful and/or a good idea to validate your adder? Be prepared to defend your choices. Our `compare_correct_sum` task is incomplete. You need to identify what it's missing and fix it.

1.4 Synthesizing Your Design

Synthesize your 64 bit adder. Ensure that your testbench from section 1.3 also passes successfully on the synthesized version. This is the first time we've synthesized a project with multiple source files so we'll need to make a few changes:

- In the TCL script: (1) `read_file -f sverilog [list ''fa1.v'' ''fa64.v'']`
  and (2) `set CLK_PERIOD 1`
- In the Makefile, the rule to make the `.vg` file needs to depend on all of the design files.
- The generated `.vg` file will match the design name set in the TCL script. This name must be the top level module in the design, in our case this is `full_adder_64bit`.

Once you have successfully run synthesis, look at the .rep file. Search for the word "slack". It's been violated! What does this mean? How do you fix this problem?

2 Submission

Once you are confident in your module and testbench, go to office hours, place yourself on the help queue and a GSI will check you off. To get a signoff, we will be looking for the following things (please have them ready). Also be prepared to answer some of the questions from the lab.

- fa1.v
- fa64.v
- fa1_test.v
- fa64_test.v
- Waveforms for fa64 testbench
- full_adder_64bit.rep