CS/ECE 552
Fall 2022
Homework 3

You should do this assignment on your own, although you are encouraged to talk with classmates in person or on Piazza about any issues you may have encountered. If you take code from another student, you will be penalized according to the UW Student Code of Conduct, as laid out in the syllabus. The standard late assignment policy for the course applies.

Before starting to write any Verilog for each problem, you should do the following:

1. Break down your design into sub-modules.
2. Define interfaces between these modules.
3. Draw schematics (either by hand, or on a computer) for these modules.
4. Then start writing Verilog.

Total Points: 100

PROBLEM 1 [50 points]:

Using Verilog, design an 8-by-16-bit register file (i.e., 8 registers, each of size 16 bits). It has one write port, two read ports, three register select inputs (two for read and one for write,) a write enable, a reset and a clock input. All register state changes occur on the rising edge of the clock. Your basic building block **must be the D-flipflop given in the provided files**. The read ports should be all combinational logic. Do not use tri-state logic in your design.

Design this register file such that changing the width to 32-bit or 64-bit is straightforward (e.g., by using parameters).

The read and write data ports are 16 bits each. The select inputs (read and write) are 3 bits each. When the write enable is asserted (high) the selected register will be written with the data from the write port. The write occurs on the next rising clock edge; write data cannot flow through to a read port during the same cycle.

There is no read enable; data from the selected registers will always appear on to the corresponding read ports.

The reset signal is synchronous and when asserted (active high), resets all the register values to 0.

The err output should be set to 1 if any register (or other sub-module) in the register file has an error. For now, you may assume that an error only happens in a register if the input or enable is an unknown value, and in other sub-modules if the input is an unknown value. Otherwise, it should be set to 0.

You may find Figures B.8.7 and B.8.8 in your textbook good places to start, when thinking about how to design your register file.

You must use a hierarchical design. Design a 16-bit register first, and then put 8 of them together with additional logic to build the register file.

Do not make any changes to the provided regFile_hier.v file.

Testbench instructions:

You must verify your design using the testbench in the supplied tar file. Of course, you are welcome and encouraged to write additional tests on your own. Run the testbench in your hw3_1 directory using the command: wsrun.pl tb_regFile_hier *.v

The testbench for this problem (tb_regFile_hier.v) generates a random set of input signals to your module in each cycle, and compares outputs from your module with outputs that are expected from a perfect register file implementation.

If there are no errors in your design, you will see a "*TEST PASSED*" message. If the testbench failed with a "*TEST FAILED*" message, look for error messages like "*ERRORCHECK: Incorrect read data in cycle <cycle_number>*" in the testbench output. Above each of these error messages you will see the inputs to your module, your outputs and the expected outputs for that cycle which can help you debug. Note that when grading we may run additional tests, so if you pass the random tests, it does not guarantee you will pass all the tests we run.

What to submit:

1. Make directory hw3 and inside it makes another directory hw3_1.
2. Submit all the Verilog files in hw3_1.
3. Also run the Verilog rules checking script and submit the results in hw3_1.

hw3 ->
- *hw3_1 ->*
  - *All Verilog files*
  - *All Verilog rules checking output files*

PROBLEM 2 [50 Points]:

In Verilog, create a register file that includes internal bypassing so that results written in one cycle can be read during the same cycle. Do this by writing an outer "wrapper" module that instantiates your existing (unchanged) register file module; your new module will just add the bypass logic. The list of inputs and outputs of the outer module should be the same as that of the inner module.

Do not make any changes to the provided regFile_bypass_hier.v file.

Testbench instructions:

You must verify your design using the testbench in the supplied tar file (of course you are welcome and encouraged to write additional tests on your own). Run the testbench in your hw3_2 directory using the command: wsrun.pl tb_regFile_bypass_hier *.v

The testbench for this problem (tb_regFile_bypass_hier.v) generates a random set of input signals to your module in each cycle, and compares outputs from your module with outputs that are expected from a perfect register file bypass implementation.

If there are no errors in your design, you will see a "*TEST PASSED*" message. If the testbench failed with a "*TEST FAILED WITH xx ERRORS*" message, look for error messages like "*ERRORCHECK: Read data incorrect in cycle <cycle_number>*" in the testbench output. Above each of these error messages you will see the inputs to your module, your outputs and the expected outputs for that cycle which can help you debug. As in problem 1, we may run additional tests when grading.

What to submit:

1. Make directory hw3 and inside it makes another directory hw3_2.
2. Submit all the Verilog files in hw3_2.
3. Also run the Verilog rules checking script and submit the results in hw3_2.

hw3 ->

hw3_1 ->

...

hw3_2 ->

All Verilog files
All Verilog rules checking output files

What to Hand In (SUMMARY):

To submit this assignment, zip or tar your Verilog files together and submit them as a **single file named <netID>-hw3.tgz or <netID>-hw3.zip** on Canvas. Inside this tarball/zip, should be a top-level directory (e.g., hw3), which contains hw3_1/ and hw3_2/; inside hw3_1 should be all files for problem 1 and so on for other problems – you must keep this directory structure. For example, if my Net ID is tanand, so my submission would be called tanand-hw3.tgz (or tanand-hw3.zip) and in it would be hw3, which itself has 2 files/sub-folders: hw3_1/ and hw3_2/. If you don't have experience with tar, I recommend consulting tutorials such as this one: https://www.tecmint.com/18-tar-command-examples-in-linux/. In addition, before submitting you should **run the Verilog check and the naming convention check on all the files** (you don't need to run it on your testbenches).

Few Important points:

1. Please only include those Verilog files, which are part of your Verilog design. Do not submit any unfinished/partially implemented/unused Verilog files which are in no way being used by your main design.
2. Do not submit extra binaries, screenshots, transcripts etc or anything other than your Verilog design files & rule checking output files (as mentioned above) in your tar/zip submission.
3. Always try to compile & simulate your design on CSL Machines first using wsrun.pl as mentioned above before submitting your designs.
4. Please make sure that your module name & Verilog file name have one-to-one correlation (i.e both the names should be same).

5. Do not define multiple modules within the same Verilog design file. For Example, If your design file is named: mux1_4.v There should only be one module defined within this file with name mux1_4