

CS/ECE 552

Spring 2022

Homework 3

Due 11:59 PM Central Time on Friday, February 25th

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 applies: you may submit up to 2 days late with a 10% penalty for each late day. See *Hand in* directions for additional details.

Total Points: 20

As with previous homeworks, this homework requires using Verilog. A reminder of some important documents:

1. Follow the instructions on [ModelSim Setup Tutorial](#) to get your environment setup.
2. Read the [Command-line Verilog Simulation Tutorial](#). Additional references are on course website.
3. Read the Verilog [Cheat sheet](#) and Verilog [rules](#) pages. Everything you need to know about Verilog are in these documents.
4. Read the [Verilog file naming conventions](#) and [Verilog file naming convention checking](#) webpages and adhere to those conventions. We will be checking this in your submission to ensure you followed the rules.
5. Read the [Verilog rules checking](#) page on the course website and adhere to the conventions. This page also provides information on how you can check that your files conform to these rules.
6. Read the [Handin Verification](#) page on the course website. You will need to run this before submitting your answers.

You should simulate your solutions both to verify the correct function of your designs for yourself. You also have to hand in a copy of the Verilog files, as mentioned below.

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 (these will be handed in as scanned schematic.pdf file).
4. Then start writing Verilog.

Problem 1 [10 points]

Using Verilog, design an 8-by-16-bit register file (i.e., 8 registers, each of size 16 bits). See the Verilog interface below. 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 (`/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/verilog_code/hw3/hw3-templates.tgz`). 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 of the tests we run.

Problem 2 [10 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: `wsrn.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 Hand In

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, my Net ID is `msinclair`, so my submission would be called `msinclair-hw3.tgz` (or `msinclair-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](#). In addition, before submitting you should **run the Verilog check and the naming convention check on all the files** (just the new modules you are writing, you don't need to run it on your testbenches).

In addition to the Verilog, you should also turn in schematics for each of your components. The schematics may be handwritten or computer generated but must be legible if they are handwritten. The schematic files should be named **schematic.pdf** and placed in the corresponding problems subdirectory (e.g., `hw3_1/schematic.pdf`). **Any solution without a corresponding schematic drawing will not be graded.** Although the schematics may seem simple for some of these components, as your project gets bigger and bigger, you'll find that drawing schematics of each component and the bigger picture will make your task much, much easier.

Graphically your handin directory should look like:

- `hw3` (Top level folder) // tar/zip this as `<netID>-hw3.tgz` or `<netID>-hw3.zip`
 - `hw3_1` (folder containing everything for Problem 1, including `schematic.pdf`)
 - `hw3_2` (folder containing everything for Problem 2, including `schematic.pdf`)

Verifying Your Handin

We have also created a script to check that your submission correctly follows the format, located at:

```
/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/scripts/hw3/verify_submission_format.py
```

You should run this script before submitting in order to ensure that you don't lose points for incorrectly formatting your submission! Additional details about the handin script, including how to run and examples, are available [here](#). *Note: the submission script **will** check for .vcheck.out files for additional .v files you add.*