CS/ECE 552
Spring 2022
Homework 2
Due 11:59 PM Central Time on Friday, February 18th, 2022

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: 15

As with homework 1, 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 to verify the correct function of your designs for yourself. You also have to hand in your copy of the Verilog files, as mentioned below.

Problem 1 [5 points]

Design a 16-bit barrel shifter in Verilog with the following interface. If you need additional information on barrel shifter design, you can consult the ALU lecture notes. Note that we call this shifter a barrel shifter because it is capable of shifting and rotating bits all the way around in any direction (like a barrel). You should **not** just write a case statement with each constant shift amount – instead think about how the hardware would be designed for a shifter, and what underlying modules (e.g., muxes) you might be able to utilize to do shifting. You may consider starting with slide 9 of Unit 3 for inspiration.

Inputs:

- [15:0] *In* - 16-bit input operand value to be shifted
- [3:0] *ShAmt* - 4-bit amount to shift (number of bit positions to shift)
- [1:0] *Oper* - shift type, see encoding in table below

Output:

- [15:0] *Out* - 16-bit output operand

Opcode	**Operation**
00	Rotate left
01	Shift left
10	Shift right arithmetic
11	Shift right logical

(*Aside: you should think about if the above 4 opcodes are sufficient to represent all of the shift operations you need to implement for your project once we release it.*)

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

1. Break down your design into sub-modules.
2. Define interfaces between these modules.
3. Draw paper and pencil schematics for these modules (these will be handed in as scanned schematic.pdf file(s)).
4. Then start writing Verilog.

Verify the design using the testbenches in the supplied tar file (`/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/verilog_code/hw2/hw2-templates.tgz`). You are also encouraged to write additional tests, as we may use more tests when grading your assignment. For a simple walk-through of how to run the testbench and example outputs see the Homework 2 Demo page.

Problem 2 [10 points]

This problem should also be done in Verilog. Design a simple 16-bit ALU. Operations to be performed are 2's Complement ADD, bitwise-OR, bitwise-XOR, bitwise-AND, and the barrel shifter unit from problem 1. Additionally, it must have the ability to invert either of its data inputs before performing the operation and have a *Cin* input (to enable subtraction). Another input line also determines whether the arithmetic to be performed is signed or unsigned. Use your CLA (from homework 1) in your design, extended as needed to take things like overflow and sign into account. For all the shift and rotate operations, assume the number to shift is input *InA* to your ALU and the shift/rotate amount is bits [3:0] of input *InB*.

Opcode	**Function**	**Result**
000	rll	Rotate left
001	sll	Shift left logical
010	sra	Shift right arithmetic
011	srl	Shift right logical
100	ADD	A+B
101	AND	A AND B
110	OR	A OR B
111	XOR	A XOR B

The external interface of the ALU should be:

Inputs

- *InA*[15:0], *InB*[15:0] - Data input lines *InA* and *InB* (16 bits each).
- *Cin* - A carry-in for the LSB of the adder.
- *Oper*(2:0) – A 3-bit opcode that determines which operation should be performed and outputted. The opcodes are shown in the table above.
- *invA* - An invert-A input that causes the A input to be inverted before the operation is performed. *invA* is active high, which means it inverts *A* when *invA* is 1.
- *invB* - An invert-B input (also active high) that causes the *InB* input to be inverted before the operation is performed.
- *sign* – A flag to indicate whether signed-or-unsigned arithmetic should be performed in the ADD function on the data lines (this also affects the *Ofl* output). The sign input is active high for signed arithmetic and active low for unsigned arithmetic.

Outputs

- *Out*(15:0) – Output data from your ALU (16 bits).
- *Ofl* - This should be high (1) if an overflow occurred (1 bit).
- *Zero* - This should be high if the result is exactly zero (1 bit).

Other assumptions:

- You can assume 2's complement numbers.
- In case of logic functions, Ofl is not asserted (i.e., kept logic low).

The top-level module definitions and a testbench are included in the supplied tar file (`/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/verilog_code/hw2/hw2-templates.tgz`). **You must not change these top-level module names**.

Simulate and verify your design using the supplied testbench or create one yourself to test any of your submodules. You must reuse the barrel shifter unit designed in Problem 1.

As in problem 1, before starting to write any Verilog, you should do the following:

1. Break down your design into sub-modules.
2. Define interfaces between these modules.
3. Draw paper and pencil schematics for these modules (these will be handed in as schematic.pdf file).
4. Then start writing Verilog.

What to Hand In

To submit this assignment, zip or tar your Verilog files together and submit them as a **single file named <netID>-hw2.tgz or <netID>-hw2.zip** on Canvas. Inside this tarball/zip, should be a top-level directory (e.g., hw2), which contains hw2_1/ and hw2_2/; inside hw2_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-hw2.tgz (or msinclair-hw2.zip) and in it would be hw2, which itself has 2 files/sub-folders: hw2_1/ and hw2_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 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., hw2_1/schematic.pdf and hw2_2/schematic.pdf). 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 significantly easier.

Graphically your handin directory should look like:

- hw2 (Top level folder) // tar/zip this as <netID>-hw2.tgz or <netID>-hw2.zip
    - hw2_1 (folder containing everything for Problem 1, including schematic.pdf)
    - hw2_2 (folder containing everything for Problem 2, including schematic.pdf)

We will be grading whatever you submit in this zip/tar, so please make sure to include everything needed to verify your design works.

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/h
w2/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 (unlike HW1's script).*