CS/ECE 552
Spring 2019
Homework 4
Due 11:59 PM Central Time on Friday, February 22nd, 2019

See below for policy on working with others. The standard late assignment policy applies: you may submit up to 2 days late with a 20% penalty for each late day.

What to Hand In

To submit this assignment, zip or tar your Verilog files together and submit them as a **single file named <netID>-hw4.tgz or <netID>-hw4.zip** on Canvas. Inside this tarball/zip, all files for problem 1 should be a folder called hw4_1 and all files for problem 2 should be in a folder called hw4_2 – you must keep this directory structure. For example, my Net ID is msinclair, so my submission would be called msinclair-hw4.tgz (or msinclair-hw4.zip). 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). *You may work both problems with your groupmates for the project and submit a single solution per group. If you choose to work separately, you should submit your own solution. Names must be included in the partners.txt file included in the supplied tar file (on Canvas) and Github Classroom.*

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., hw4_1/schematic.pdf and hw4_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 much, much easier.

Total Points: 15

As with homework 3, this homework requires using Verilog. A reminder of some important documents (all available on Canvas):

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 Canvas under "Verilog Resources" on the main page.
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 page and adhere to those conventions. Additional information including a link to the script you can use to make sure your files adhere to these rules are on Canvas, directly under the file naming conventions page.
5. Read the Verilog rules checking page on Canvas and adhere to the conventions. This page also provides information on how you can check that your files conform to these rules.

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 (or submit one per group and update partners.txt accordingly), as mentioned above.

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.

Inputs:

- [15:0] *In* - 16-bit input operand value to be shifted
- [3:0] *Cnt* - 4-bit amount to shift (number of bit positions to shift)
- [1:0] *Op* - 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.*)

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).
4. Then start writing Verilog.

Verify the design using the testbench in the supplied tar file and on Github Classroom. For a simple walk-through of how to run the testbench and example outputs see the Homework 4 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 a ripple carry adder (e.g., from homework 3) in your design. For all the shift and rotate operations, assume the number to shift is input *A* to ALU and the shift/rotate amount is bits [3:0] of input *B*.

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

- *A*[15:0], *B*[15:0] - Data input lines *A* and *B* (16 bits each).
- *Cin* - A carry-in for the LSB of the adder.
- *Op*(2:0) - The OP code (3 bits). The OP code determines the operation to be performed. 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 *B* input to be inverted before the operation is performed.
- *sign* - A signed-or-unsigned input (active high for signed) that indicates whether signed or unsigned arithmetic to be performed for ADD function on the data lines (this affects the *Ofl* output).

Outputs

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

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 is included in the supplied tar file (on Canvas) and on Github Classroom. **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.