# CS/ECE 552
# Fall 2022
# Homework 2

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.

## Total Points: 100

### PROBLEM 1 [20 points]:

1. Design a 1-bit full adder using only the provided NOT, NAND, NOR, and XOR gates (you do not have to use all four types of gates, you are just limited to using these gate types). Again, use the provided modules. Label the inputs as *a*, *b,* and *c_in* (carry-in). Label the outputs as *s* and *c_out*. **Your module should be named fullAdder_1b and your file should be named fullAdder_1b.v.**

2. Using the designed 1-bit full adder (in part 1 above), design a carry lookahead adder (CLA) that adds two 4-bit binary numbers. Make the inputs and outputs 4-bit buses (vectors) labeled *a*(3:0), *b*(3:0), and *sum*(3:0), respectively. Label the carry-in *c_in* and the carry-out *c_out*. **Your module should be named cla_4b and your filename should be cla_4b.v**

3. Using the 4-bit CLA you created above, design a CLA that adds two 16-bit binary numbers. Make the inputs and outputs 16-bit buses (vectors) labeled *a*(15:0), *b*(15:0), and *sum*(15:0), respectively. Label the carry-out from the adder *c_out* and the carry-in *c_in*. **Your module should be named cla_16b and your file should be named cla_16b.v**

4. Use the testbench provided for testing.

**What to submit:**

1. Make directory hw2 and inside it make another directory hw2_1.
2. Submit all the Verilog files in hw2_1.
3. Also run the Verilog rules checking script and submit the results in hw2_1.

*hw2 ->*

*hw2_1 ->*

*All Verilog files*

*All Verilog rules checking output files*

## **PROBLEM 2 [40 Points]:**

Design a 16-bit barrel shifter in Verilog with the following interface. 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.

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 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).
4. Then start writing Verilog.

Verify the design using the testbench in the supplied tar file (on Canvas post for HW 2 – hw2.tar)

**What to submit:**

1. Make directory hw2 and inside it make another directory hw2_2.

2. Submit all the Verilog files and schematic.pdf in hw2_2.

3. Also run the Verilog rules checking script and submit the results in hw2_2.

*hw2 ->*

*hw2_1 ->*

...

*hw2_2 ->*

*schematic.pdf*

*All Verilog files*

*All Verilog rules checking output files*

## PROBLEM 3 [40 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 2. 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 problem 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 (*note: InA/InB may be inverted before reaching the shifter, see below – if either/both is inverted, you should shift/rotate the inverted values instead*)*.

| 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).

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 2.

As in problem 2, 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 submit:**

1. Make directory hw2 and inside it make another directory hw2_3.

2. Submit all the Verilog files and schematic.pdf in hw2_3.

3. Also run the Verilog rules checking script and submit the results in hw2_3.

*hw2 ->*

*hw2_1 ->*

...

*hw2_2 ->*
   ...
*hw2_3 ->*

  *schematic.pdf*

  *All Verilog files*

  *All Verilog rules checking output files*