Department of Electrical and Computer Engineering

*University of Wisconsin – Madison*

ECE 552 Introductions to Computer Architecture

Homework #3 (Due 2/13)

For all the questions below, paste your entire verilog code in a single text/doc file and submit the text/doc file on canvas.

1. ***Adder/Subtractor***

Use four 1-bit full adder slices and design a 4-bit Ripple Cary adder/subtractor with the overflow flag to indicate whether overflow is occurred (hint: [detect overflow](https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Comb/overflow.html)). The circuit has a single bit input to indicate whether it is going to calculate the add (0) or sub (1). To implement subtraction, use the “A-B = A+(-B)” and “-B = ~B + 1” properties of binary 2’th complement numbers.

Implement your design in Verilog and provide a testbench for your design to generate random input patterns (hint: use “$random” system task). The testbench should be checking the addsub\_4bit design’s outputs, i.e. you can use ‘+’, ‘-’ in the testbench to check if the outputs of the DUT (design under test) are correct.

The module interface you can use is:

module addsub\_4bit (Sum, Ovfl, A, B, sub);

input [3:0] A, B; //Input values

input sub; // add-sub indicator

output [3:0] Sum; //sum output

output Ovfl; //To indicate overflow

……….

full\_adder\_1bit FA1 (…); //Example of using the one bit full adder (which you must also design)

…...

endmodule

**2. ALU**

Design a 4-bit ALU module to implement the following instructions - ADD, SUB, NAND, XOR. Use a 2-bit opcode to identify these instructions.  The opcodes should be used to control the output of the ALU. The add, sub calculations should interface with the module design for Q.1.

You are required to submit the design and an exhaustive testbench. The design interface to use is:

module ALU (ALU\_Out, Error, ALU\_In1, ALU\_In2, Opcode);

input [3:0] ALU\_In1, ALU\_In2;

input [1:0] Opcode;

output [3:0] ALU\_Out;

output Error; // Just to show overflow

…….

endmodule