EECE 444 Microprocessor System Design

LAB 4

Introduction:

In the first part of this lab you will use Verilog to design an ALU using a top-down approach. You will be building an 8-bit ALU using 2 4-bit ALU slices. The logic diagram for the 4-bit ALU is shown in Figure 1. In the second part of the lab you will design a 32-bit ALU succinctly in Verilog. The 32-bit ALU will be an important part of the MIPS microprocessor that you will build in later labs.

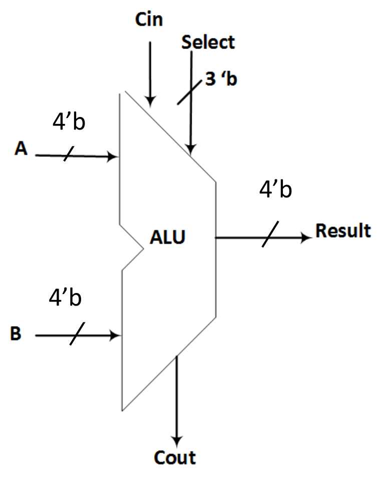

Figure 1

Design Details:

Part 1:

1. Start by designing **4-bit Carry Look-ahead Adder**, with ***A, B,*** and ***Cin*** as inputs and **Result** and ***Cout*** as outputs.
2. Design a 4-bit ALU using the adder you designed in step 1 to add and subtract.
3. The ALU will be performing the functions in Table 1:

Table 1

Select	Operation
000	Result =A AND B
001	Result = A OR B
010	Result = A + B
011	Not used
100	Result = A AND B’
101	Result = A OR B’
110	Result = A - B
111	Result = SLT

4. The SLT function performs a comparison between A and B by computing A-B. if the result is negative (A is less than B), Result will equal 1 otherwise it equals 0.
5. Build an 8-bit ALU ***(ALU_8Bit)*** using the previously designed ALU slice.
6. Inputs to the 8-bits ALU is two 8-bit numbers ***A*** and ***B***, and a 3-bits select ***ALUOp*** **ONLY.** The outputs are: ***ALUResult***, ***C*****, Z, N** and ***V***, where ***C*** is a carry out flag, **Z** is the zero flag, **N** is the negative flag, and ***V*** is an overflow flag.
7. Create a test bench to test the functionality of your ALU.
8. Create a module to implement 8-bit binary to 3-digit BCD decoder. You will need an additional output to indicate if the number is positive or negative. If the binary number is negative, the result should be the 2's complement of the number.
9. Create a module to implement a BCD-to-7segment display decoder. The inputs to this module should include the control signal indicating if the number is positive or negative. If the number is negative, use the left-most digit on your board to display the sign.
10. Create a top module to connect the ALU, the binary-to-BCD decoder, and the BCD to 7-segment display decoder.
11. Synthesize and implement your design, make sure to modify the UCF file to connect the inputs to the switches on the board and the output to the 7-segment display and the LEDs.
12. Test your design using a addition, subtraction, and all ALU operations.
13. Demonstrate your results to you TA on your next lab session.

Part 2: 32-bit ALU

1. Create a new module to implement a 32-bit ALU in Verilog. Name the file alu.v. it should have the following function declaration:

```
module alu(input  logic [31:0] a, b,
           input  logic [2:0]  f,
           output logic [31:0] y,
           output logic        zero);
```

   The output zero should be TRUE if y is equal to zero. Please note, an adder is a relatively expensive piece of hardware. Be sure that your design uses no more than one adder.
2. Test the 32-bit ALU in Xilinx only. It is prudent to think through a set of input vectors. Develop an appropriate set of test vectors to convince a reasonable person that your design is correct. Complete Table 2 below to verify that all 5 ALU operations work as they are supposed to. Note that the values are expressed in hexadecimal to reduce the amount of writing.

Table 2

Test	F[2:0]	A	B	Y	Zero
ADD 0+0	2	00000000	00000000	00000000	1
ADD 0+(-1)	2	00000000	FFFFFFFF	FFFFFFFF	0
ADD 1+(-1)	2	00000001	FFFFFFFF	00000000	1
ADD FF+1	2	000000FF	00000001		
SUB 0-0	6	00000000	00000000	00000000	1
SUB 0-(-1)		00000000	FFFFFFFF	00000001	0
SUB 1-1		00000001			
SUB 100-1		00000100			
SLT 0,0	7	00000000	00000000	00000000	1
SLT 0,1		00000000		00000001	0
SLT 0,-1		00000000			
SLT 1,0		00000001			
SLT -1,0		FFFFFFFF			
AND FFFFFFFF, FFFFFFFF		FFFFFFFF			
AND FFFFFFFF, 12345678		FFFFFFFF	12345678	12345678	0
AND 12345678, 87654321		12345678			
AND 00000000, FFFFFFFF		00000000			
OR FFFFFFFF, FFFFFFFF		FFFFFFFF			
OR 12345678, 87654321		12345678			
OR 00000000, FFFFFFFF		00000000			
OR 00000000, 00000000		00000000			

Build a self-checking testbench to test your 32-bit ALU. To do this, you'll need a file containing test vectors. Create a file called testalu.txt with all your vectors. For example, the file for describing the first three lines in Table 1 might look like this:

```
2_00000000_00000000_00000000_1
2_00000000_FFFFFFFF_FFFFFFFF_0
2_00000001_FFFFFFFF_00000000_1
```

Hint: Remember that each hexadecimal digit in the test vector file represents 4 bits. Be careful when pulling signals from the file that are not multiples of four bits.
You can create the test vector file in any text editor, but make sure you save it as text only, and be sure the program does not append any unexpected characters on the end of your file. For example, in WordPad select **File→Save As**. In the "Save as type" box choose "Text Document – MS-DOS Format" and type "testalu.txt" in the File name box. It will warn you that you are saving your document in Text Only format, click "Yes".
Now create a self-checking testbench for your ALU. Name it testbench.v.

Compile your alu and testbench in Xilinx and simulate the design. Run for a long enough time to check all of the vectors. If you encounter any errors, correct your design and rerun. It is a good idea to add a line with an incorrect vector to the end of the test vector file to verify that the testbench works!

What to turn in?

1. Please indicate how many hours you spent on this lab.
2. A complete diagram of your design, you can use any CAD tools to draw your logic.
3. Truth tables, K-maps, and equations, if used.
4. An image of your simulation waveforms for part 1, please test all possible functions.
5. Images of your test waveforms from part 2. Make sure these are readable and that they're printed in hexadecimal. Your test waveforms should include only the following signals in the following order, from top to bottom: f, a, b, y, zero.
6. In your conclusion, please include any comments or suggestions.
7. Feel free to expand the design for extra credit. Please note your design should satisfy the minimum requirements in the assignment.