**EC311 – Introduction to Logic Design**

**Laboratory 2 (Lab 2)**

**Combinational Logic Design: 4-bit ALU, Decoder, and 7-Segment Display**

**Due 10/11**

**Introduction**

In this laboratory, you will build on your Lab 1’s ALU. In addition to modifying the ALU with new “modes” and a fundamentally new adder architecture, you will also add a “decoder” unit that interfaces with the 7-segment display. Decoders are a key part of digital logic design. Frequently they are used to translate "instructions" into the control signals for a datapath. You will also change the ripple carry adder to another adder design (of your choice).

**Recommendation: read this document entirely before implementing it. The order of implementation is up to you.**

**Here are the modifications for the new ALU:**

1. Two 4-bit inputs (A and B), one “Carry-In” input
2. 4-bits of output, one “Carry-Out” output
3. Two modes of operation, “Logical” and “Arithmetic”
4. Within these modes, there are various operations controlled by 3-bits (Op2,Op1, Op0)
5. The modes work as follows:

*M=0, Logical mode*

Op2 Op1 Op0 ALU Operation

0 0 0 F= A and B

0 0 1 F= A or B

0 1 0 F= A nand B

0 1 1 F= A nor B

1 0 0 F= A xor B

1 0 1 F= A xnor B

1 1 0 F= A'

1 1 1 F= B'

|  |  |  |  |
| --- | --- | --- | --- |
| *M=1, Arithmetic mode* | | | |
| Op2 | Op1 | Op0 | ALU Operation |
| 0 | 0 | 0 | F= negate A |
| 0 | 0 | 1 | F= negate B |
| 0 | 1 | 0 | F= A + B |
| 0 | 1 | 1 | F= A - B |
| 1 | 0 | 0 | F= A \* B |
| 1 | 0 | 1 | F= B - A |
| 1 | 1 | 0 | F= A + 1 |
| 1 | 1 | 1 | F= A - 1 |

1. Input should be controlled by the switches on the board. These inputs should go to the decoder as the “instruction” (explained below).
2. Output should be displayed on the **7 segment display** (using a single digit) on the board. Cout should go to a single LED.

**Decoder Unit**

In this lab, you must design a decoder unit. This decoder is hooked to the switches on the board for the inputs (i.e the instruction) and is hooked to the 7-segment display for outputs. The decoder takes in an instruction, decodes it, and sets the correct signals ALU.

The output of the decoder is driving the 7-segment display. It either shows the result of the ALU operation (aka the ALU outputs in **HEX**) or it shows the value of the OpCode from the input instruction in **HEX** . This is determined by a single control line driven by one of your switches. For example, if the output control is “1” (or any value you want to set it up to) the 7-segment display shows the HEX result of the ALU output, if instead, the output control is “0” then the 7-segment display will show the 4-bit OpCode from the instruction in HEX.

*Instruction format (12-bits)*

4-bit OpCode 4 bits A 4 bits b

Here are the OpCodes:

0000 – A AND B

0001 – A OR B

0010 – A NAND B

0011 – A NOR B

0100 – A XOR B

0101 – A XNOR B

0110 – A’

0111 – B’

1000 – Negate A

1001 – Negate B

1010 – A+B

1011 – A-B

1100 - A\*B

1101 – B-A

1110 – A+1

1111 – A-1

So if you wanted to add A and B together when A = 5 and B = 3 the instruction would be:

1010 0101 0011 in binary (what you would need to set the switches to)

Or

A53 in HEX (**the 7 segment display should only use a single digit and show A**).

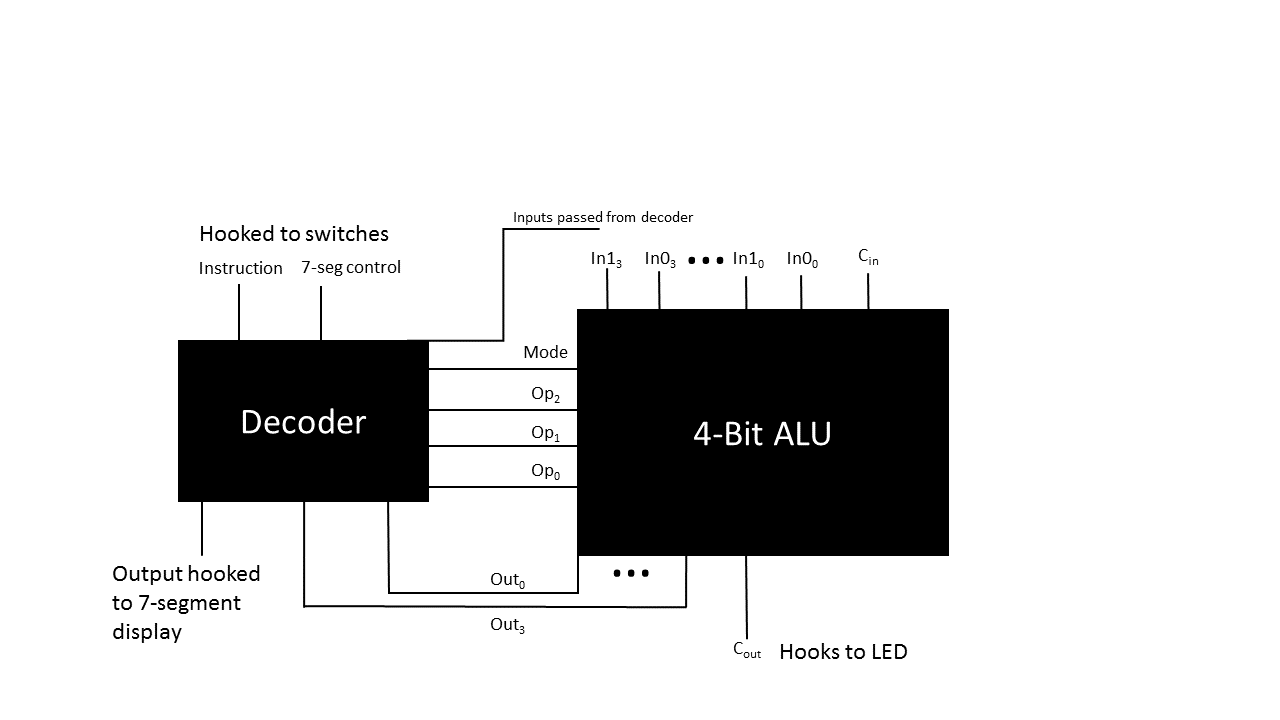
Else if you wanted to get A and B when A= 3 (0011) and B=9 (1001) the instruction would be:

0000 0011 1001 in binary (what you would need to set the switches to)

Or

039 in HEX (**the 7 segment display should only use a single digit and show 0**).

Here is a black-box view of the overall new design (Figure 1):



**Figure 1: Combined ALU and Decoder**

**Changing the Adder**

**(You can do this at any point in the lab exercise order)**

You already know what a ripple-carry adder is. There are many other types of adders. For example, Carry-select adders are another method to compute the sum. It is simple but rather fast. Schematic of 2 bit carry select adder:



Carry select adders generally consist of smaller addition blocks and multiplexers. The logic behind the method is simple: we don’t need to wait for the carry to propagate all the way to the nth bit. Calculations are performed simultaneously, and the results are chosen using multiplexers.

N-bit addition using carry-select adders could be done using two or more ripple carry adders. However, all the calculations are performed twice: once assuming the carry is one and the other assuming it is zero. The sum and carry-out are then fed into muxes which are controlled by the first addition step.

For this lab you need to create another Adder for your ALU. You can use any other architecture you like. The carry-select is just provided as an example. Others include Carry Lookahead and Cary Save. **Use the internet to do some research on adders.**

**What to turn in (ZIP 1-4 together):**

1. A README.txt file that explains your submission, its organization, and includes both team member’s names and UIDs. Feel free to include any other comments you want your TA to see that will be relevant while they are grading the assignment.
2. Verilog files (.v) for your new adder, your new ALU (adder + other modes), your decoder (by itself), and the full system (using the new ALU+ decoder+7-segment LED). Zip these together in an organized way and use good naming conventions.
3. Testbench files for each of the modules in #2. Feel free to have more if needed and explain in your README.
4. Write a short discussion about your new ALU (and save as a pdf). Which adder did you choose? Why? Which is faster, your ALU or decoder? Why do you say that? How big is your overall design (use FPGA terms like LUTs, CLBs, etc.)? Could it get smaller? What is the biggest part of your design?
5. Demo your design to your TA. Arrange a time. The lab would be ideal but that may not be possible. OH work as well. If necessary, you can record yourself using the FPGA and provide that video to your TA if needed. As with Lab 1, the demo is due 1 week after the Lab 2 submission. However, all Lab2 files must NOT have timestamps after the 10/11 due date.