Frank I Gomez
4/22/21
Lab 3, Section D.
ID: 1650550
figomez
CSE 100L

Lab 3 Write Up

1. Description

The purpose of this lab was to expand on our full-adder from Lab 2, this time adding two 8-bit two's complement (positive or negative) numbers together, in addition to subtracting them. We were then to represent this value on our seven segment displays, using 2 of the 4 displays this time. All of this was constructed with multiplexers, specifically 2-1, 4-1, and 8-1. Finally, we had to display them on the physical boards this time. Buses were also used for the first time in this lab.

2. Design

- Top Level

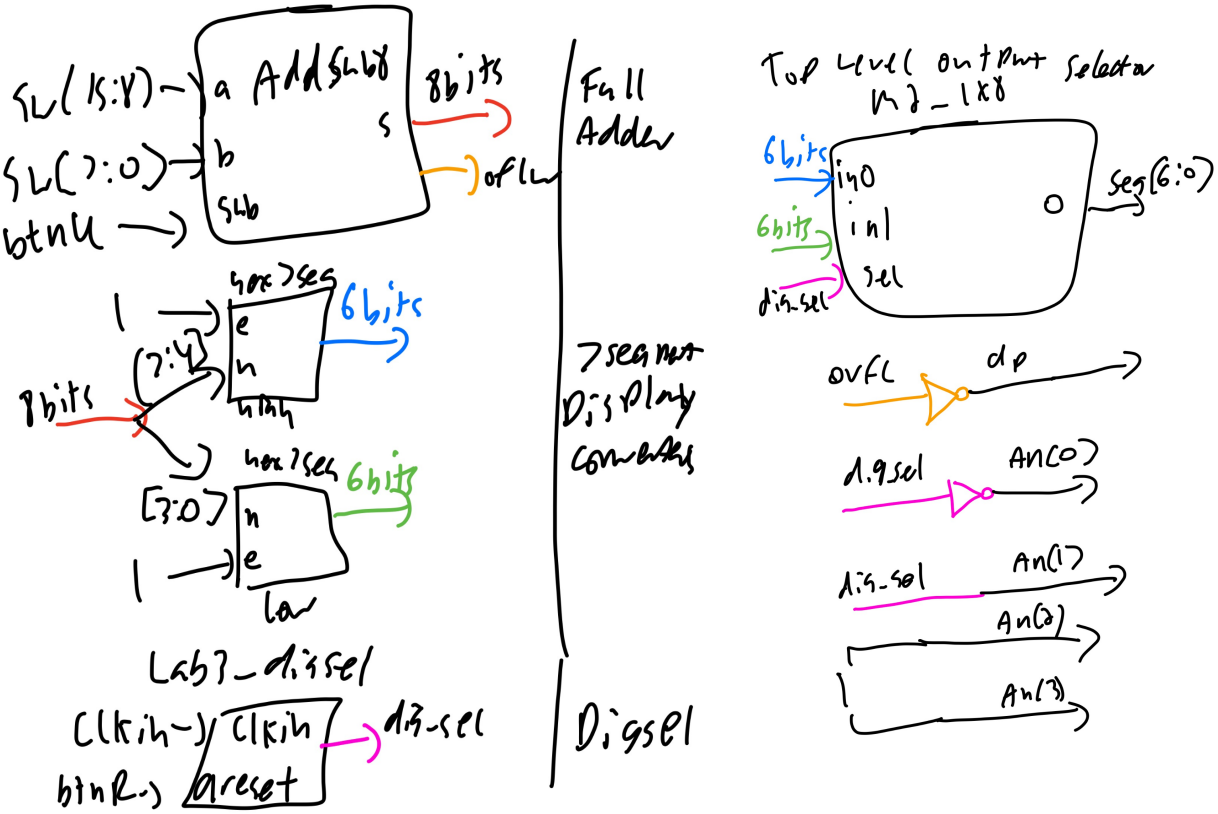

Diagram 1

There's a lot going on in the top level this time around. Starting from the top left, there's the full adder, with an extra input btnU, which allows subtraction. Following this, the 8 bits from the full adder are then split into 2 4 bit busses, which are then converted into their respective seven segment display values. The main point of the digsel module is to select between these two display values in the final, top level selector, where dig_sel will alternate between the high and low components of the 7 segment display, to display them on the board. Overflow is also inverted here, to fit with the board's inverted display. AN2 and AN3 are high, the same as before.

- m4_1e

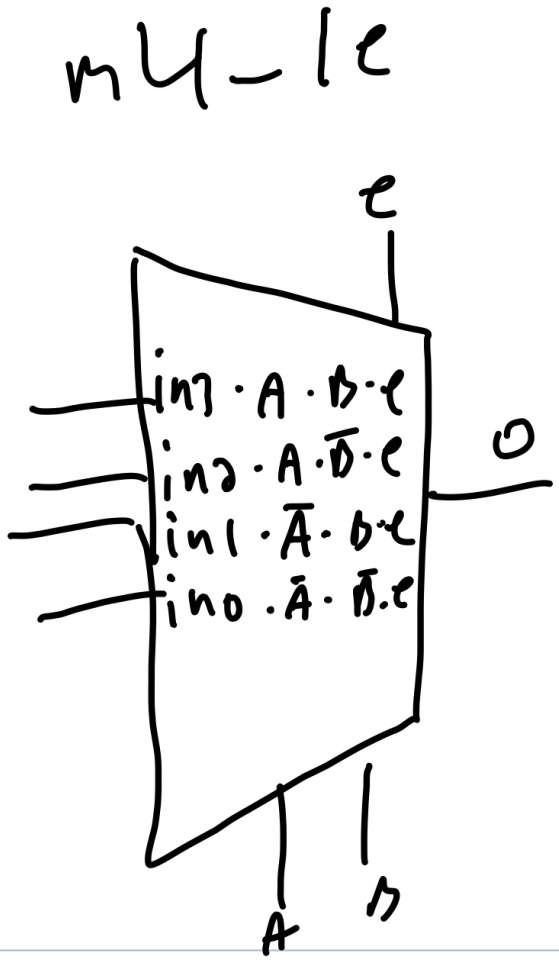

Diagram 2

The 4-1 mux selects between 4 possible outputs, based on the values of the two selectors, A and B, and the input bits in the bus in[3:0]. It's just a sum of products at the end of it, so only one case needs to pass for the entire thing to pass. The enabler bit, e, forces the result to always be 0, when it is low.

- m8_1e

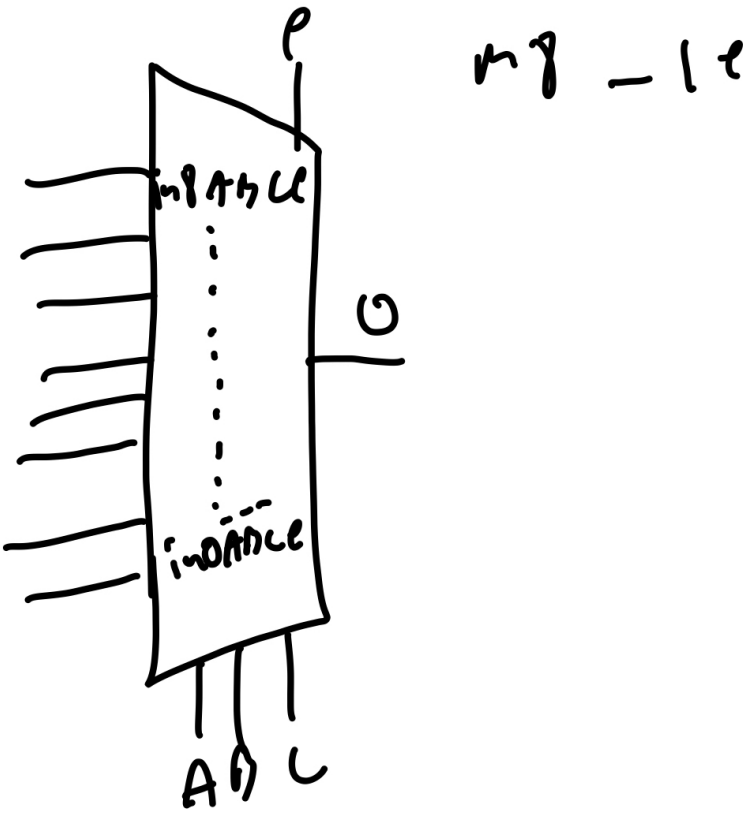

Diagram 3

The 8-1 mux follows the same process as the 4-1 mux, just with an 8 bit input, alongside 3 selector bits.

- m2_1x8

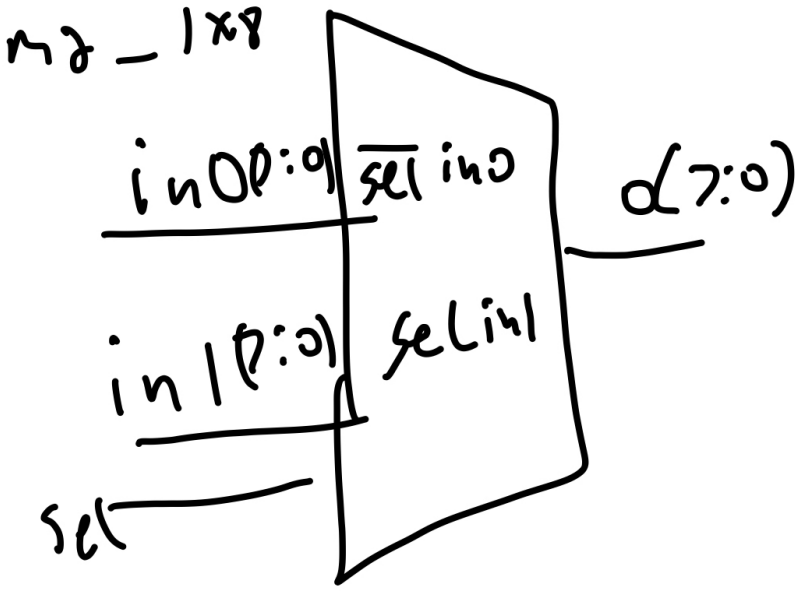

Diagram 4

The 2-1 mux, on the other hand, follows a different process. It has 3 inputs, 2 8-bit buses, and a selector bit. This time around, the output is also an 8-bit bus, and will be equal to either in0 or in1, entirely dependent on the value of sel. If sel is low, the output will be in0, and likewise for high and in1.

- Full_Adder

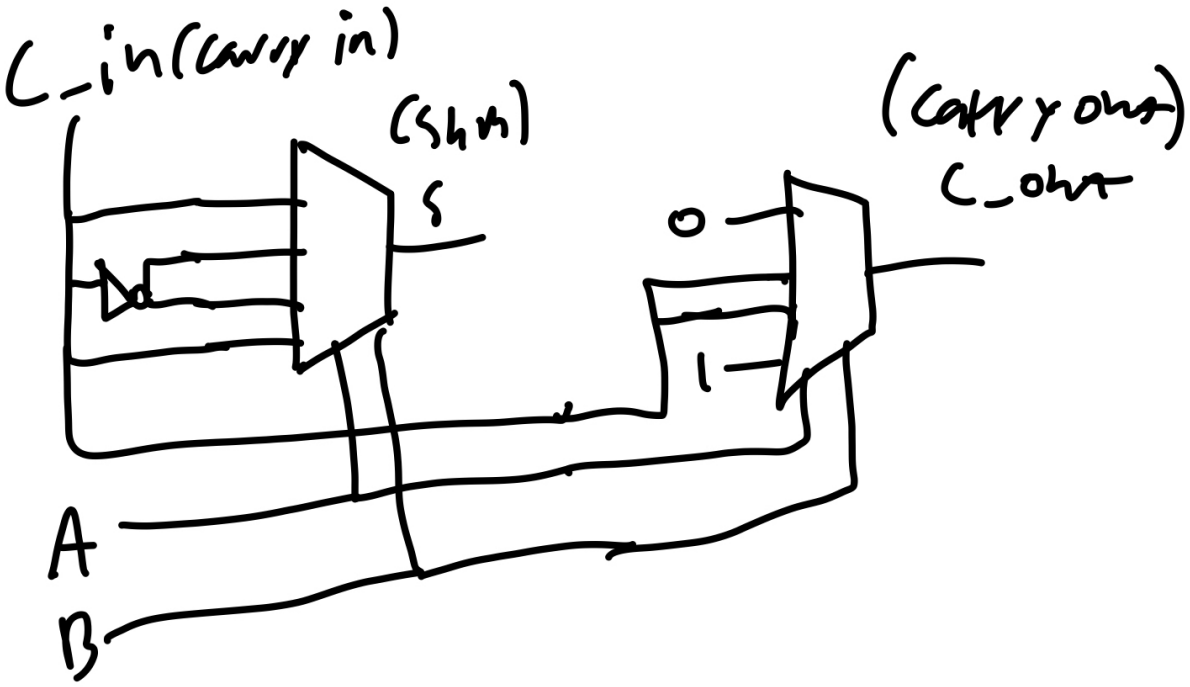

Diagram 5

In all regards, the full adder for lab 3 was the same as the full adder for lab 2, with relatively the same underlying logic, only using 2 4-1 muxes instead of XOR gates. A 4-1 mux with the inverse of the input at the 1st and 2nd positions is essentially the same as a 3 input XOR gate. Carry out is similar, receiving a 0 and 1 at the 0th and 3rd positions, and the input at the 1st and 2nd.

- AddSub8

AddSub8 is just a series of 8 full adders, adding the two full, 8-bit inputs. If subtraction is enabled, the second variable, B, will be converted to two's complement, by inverting its values and adding a 1 at the first carry in. Overflow is also handled here, a simple logic equation based around the values and carry_outs of the 2 numbers at the 6th and 7th bit positions.

- hex7seg

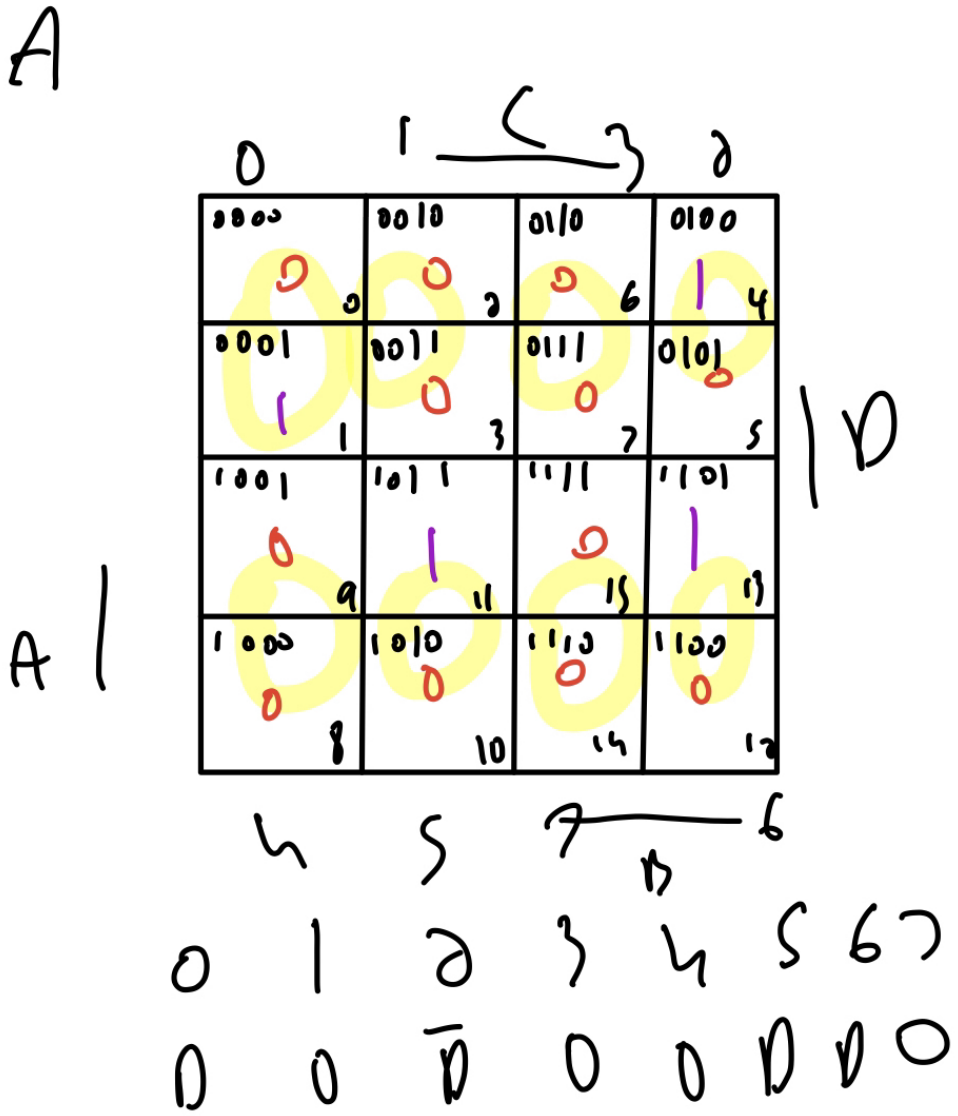

Diagram 6

The last coded section, hex7seg, takes in a 4-bit input, and obtains the respective 7-segment hex display. The logic for this section was obtained by creating 7 K-maps, each representing one of the 7 segments, which is then used as the 8-bit input for the respective 8-1 mux seen early. All 16 hex values, 0-F, can be obtained through this.

- lab3_digsel

Finally, lab3_digsel. This file was given to us, and its function is providing a single bit output that rapidly switches between high and low, allowing the 7-segment display to display 2-1 mux to alternate which 4 bits are being displayed on the display.

3. Testing and Simulation

Similarly to lab 2, verilog's simulation feature was used to test every important value of this lab, confirming that each module was functioning properly, or displaying when they weren't. All the muxes functioned properly on their first try, and the 7 segment display only had a single incorrect value that was easily fixed. The source of the most problems, unfortunately, was the full adder, particularly, the overflow. Coming up with all possible cases of overflow was difficult at first, only getting easier when I realized I could obtain all of them with 3-bit addition, which could be done mentally. Finally, once this was complete, it was programmed onto the board, where the overflow bit was, again, wrong, as I had forgotten that high means off, and vice versa, for the boards. This was quickly fixed.

4. Results

- How fast the signal **dig_sel** is oscillating.

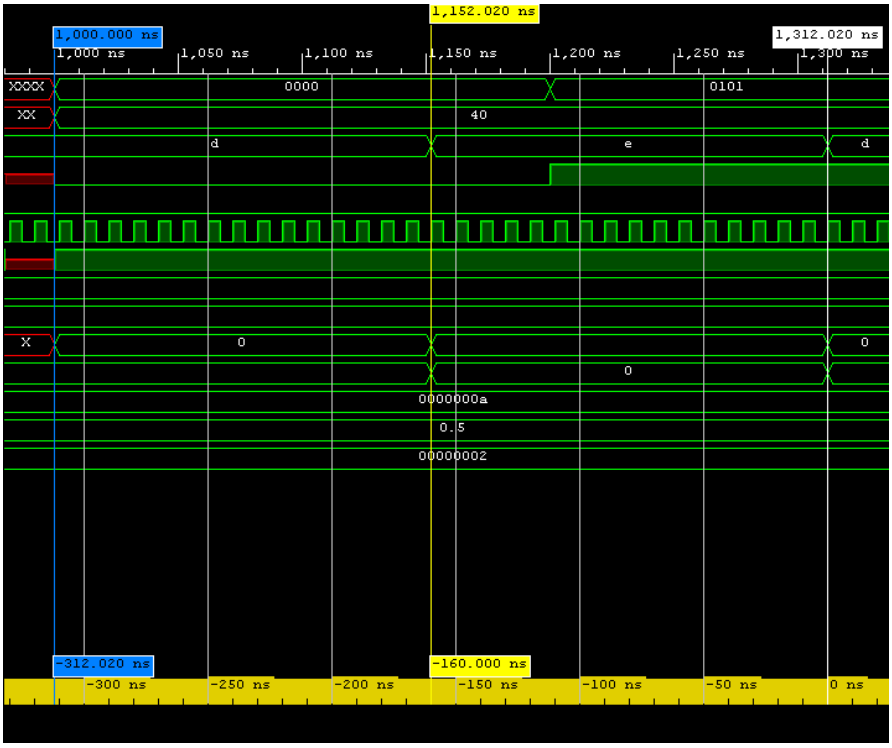

Diagram 7

Placing markers at the start and end of a full oscillation period, it can be estimated that the period of dig_sel is 312.2 nanoseconds, flipping every 156.1 nanoseconds. This means it oscillates at a frequency of 3.2 Megahertz.

5. Conclusion

There was one more error in the program, not stated earlier, as it was less of a code problem, and more of a conceptual one. I had forgotten that, in verilog, the value 1 is different from the value 1'b1, as "1" by itself is an integer. The actual result of this is that the first simulation run that involved these values resulted in every expected output being incorrect, and about an hour and a half of troubleshooting, until randomly swapping from 1 to 1'b1 while trying to figure out why it was not working. If there was anything I would change on a subsequent run of this lab, would be to remind students that the variables are in fact, different. Otherwise, I learned how to use busses, multiplexers, and how to run the simulator for specific increments of time.

6. Appendix

(Note: Waveform Viewer cannot be printed to PDF, and as such, it is added here as a screenshot.)