Gabriel Dimas

April 22, 2022

Section C

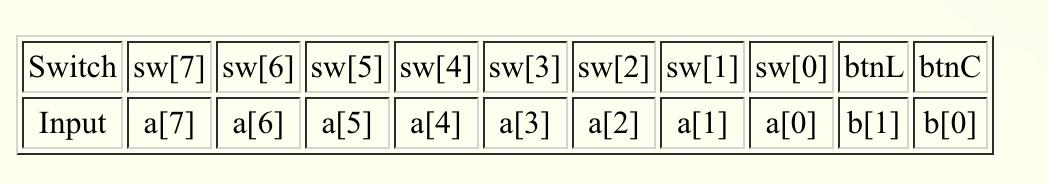
**Description**

The purpose of this lab is to build and use multiplexers, full adders, and a seven-segment display converter to display clock-led signals from eight switches and buttons. This lab is also meant to teach how to use buses, vectors, and multiplexor designs. The project implements an 8-bit binary number along with a 2-bit (buttons) binary number and takes their sum.

**Design**

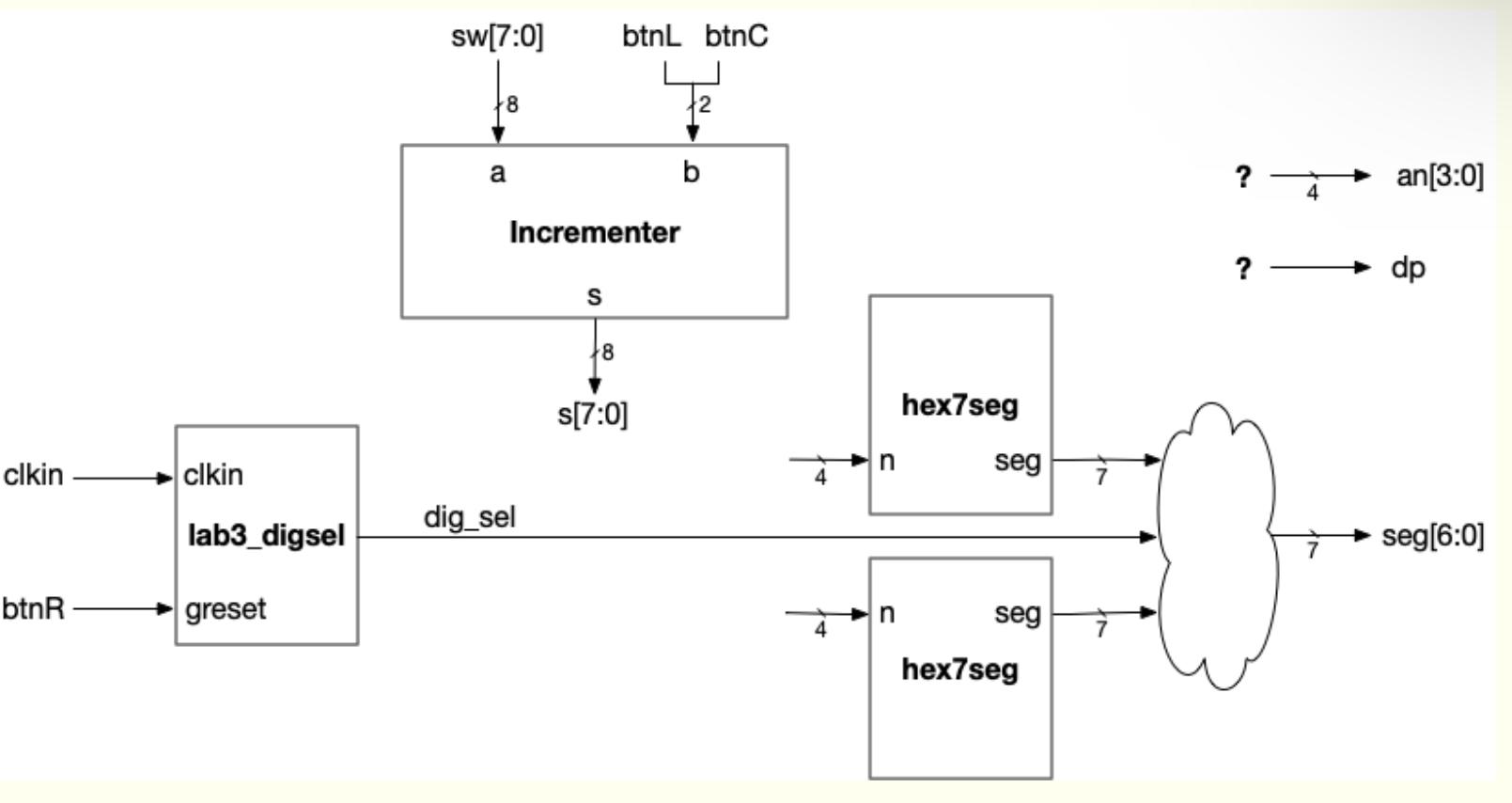
Top Level

* Inputs
  + **sw[7:0]**: Basys3 Board switches are used to simulate HIGH (1) and LOW (0) binary numbers.
  + **btnL**: Basys3 Board button used to simulate HIGH (pressed) and LOW (depressed). Used with btnC.
  + **btnC**: Basys3 Board button used to simulate HIGH (pressed) and LOW (depressed). Used with btnL.
  + **btnR**: Basys3 Board ground reset. HIGH (reset) LOW (do nothing)
  + **clkin**: input waveform to update the status of all pressed buttons
* Outputs
  + **seg[6:0]**: maps to LEDs on the seven-segment display. Mapped ‘0’ for ‘a’ through ‘6’ for ‘g’[[1]](#footnote-0).
  + **an[3:0]**: maps which of the displays to light. ‘0’ is right-most, while ‘3’ is left-most. This lab only uses displays ‘0’ and ‘1’.
* Implementation
  + The purpose of this module is to bring all of the other modules together and is used as our top-level design module. To begin, create an instance of the Incrementer module and connect the input switched to the input of that instance. Create a bus called “sum\_output” to hold the value that the Incrementer outputs to hold the sum of the bit addition.
  + Here is the mapping of switches to inputs for the **Incrementer[[2]](#footnote-1)**:

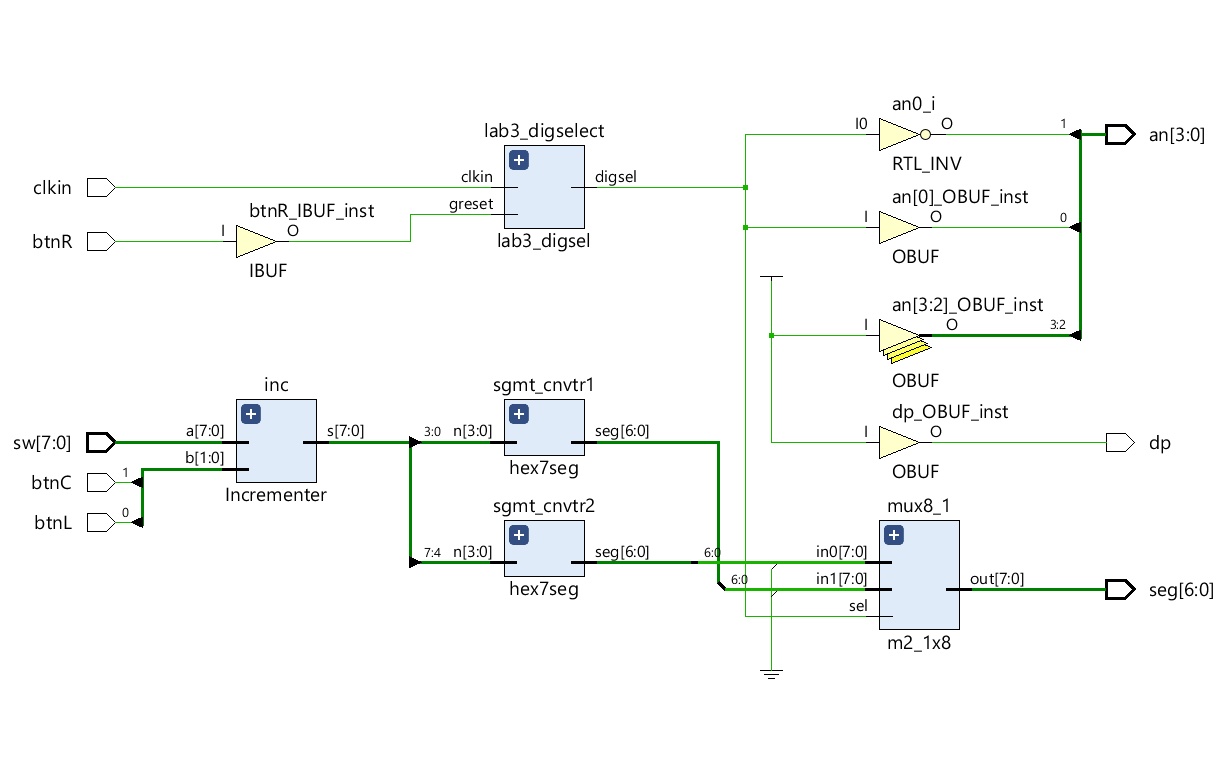


**Figure 1**

* + The output of the Incrementer is an 8-bit output. The first four bits are connected as inputs to an instance of **hex7seg** and the other four bits are connected to another instance of **hex7seg**. The outputs of each module output 6-bit values. The outputs of each module call should be connected to each of the two seven-segment displays for this project (AN0 and AN1). Create an instance of Lab3\_digsel[[3]](#footnote-2) and connect the respective inputs and outputs. This module uses the clock signal to refresh the displays and the switch values. Then, create an instance of m2\_1x8. The inputs of this are two buses of wire, which the mux will select (based on the selector value) to assign or refresh a screen. Here is an implementation of the top module[[4]](#footnote-3):

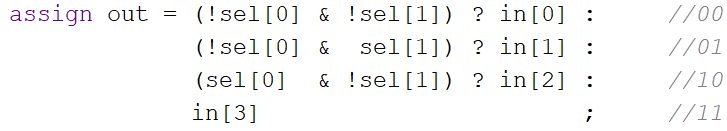
**Figure 2**

Here is how the top level module was actually implemented:

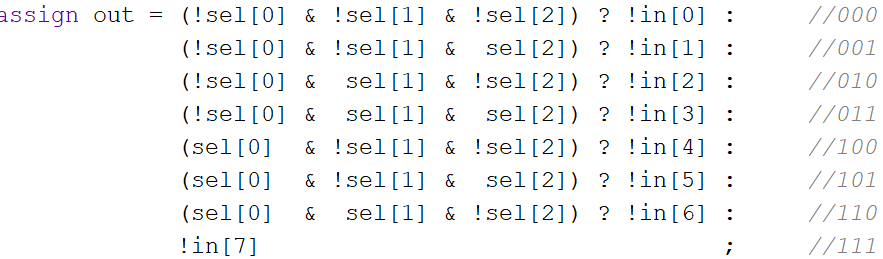
**Figure 3**

Multiplexers

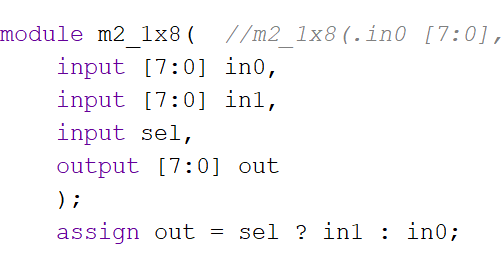
* **4 to 1**: in[3:0] bus and sel[1:0] bus inputs and ‘out’ wire as the output. This uses conditional statements depending on the selector value to choose what the output would be. Here is a snippet of what that looks like:

**Figure 4**

* **8 to 1**: in[7:0] bus and sel[2:0] bus inputs and ‘out’ wire as the output. This uses conditional statements depending on the selector value to choose what the output would be. Here is a snippet of what that looks like:

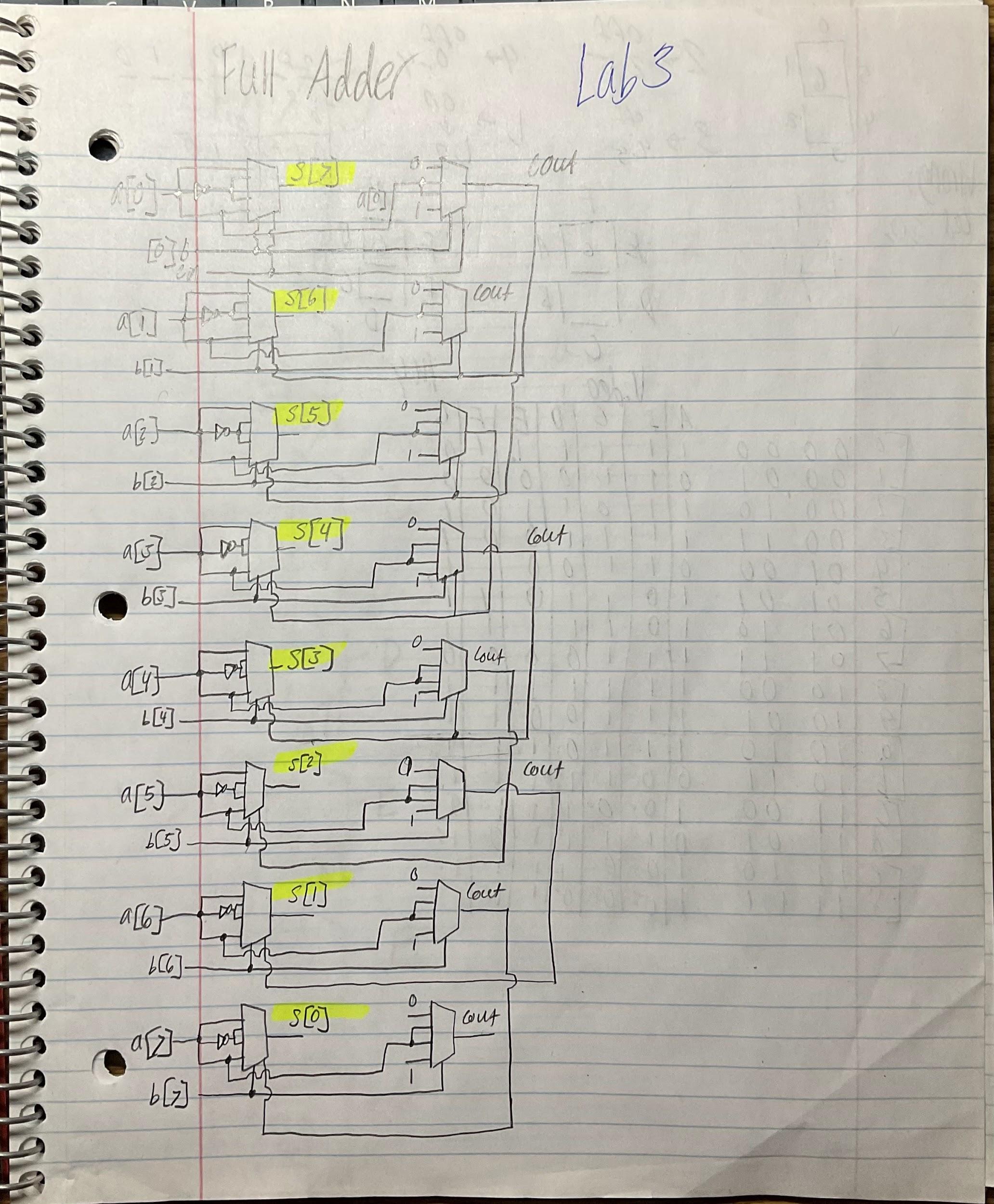
**Figure 5**

* **2 to 1 (busing 8)**:in0[7:0] bus, in1[7:0] bus, and ‘sel’ inputs, and out[7:0] bus wires as the output. This multiplexer takes in two buses and outputs one of the two, depending on the selector. This module also uses conditional programming to choose what the output should be. This selector wire is based on what the anode wire is. If the selector pin is ‘0’, then this mux would output the bus of in0 wires. Else, the mux would output the bus of in1 wires. Here is what that looks like:

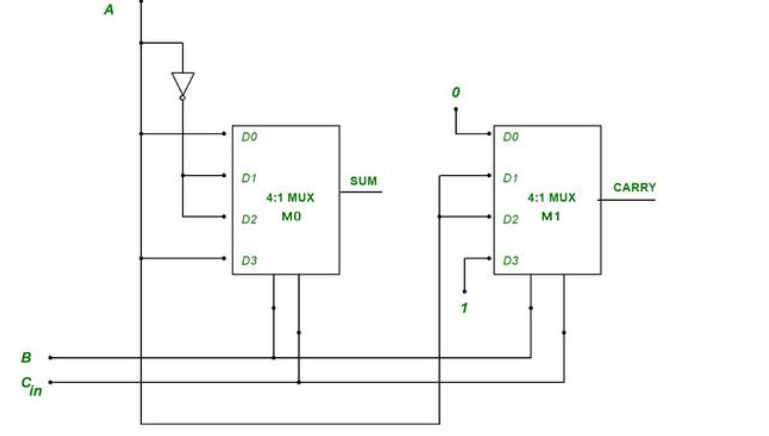
**Figure 6**

Full Adder

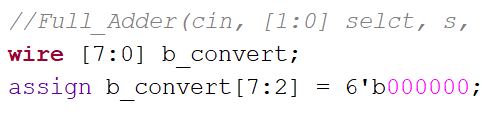
* Inputs
  + **cin**: the carry-in bit used for the addition
  + **select**: a bus of two wires used for the multiplexers. Holds the two values we are adding together
* Outputs
  + **s**: the most significant bit of the addition
  + **c-out**: the carry-out bit used for the next full adder carry-in
* Implementation
  + This module uses two 4 to 1 multiplexers in series with each other. The first mux uses **sel[0]** value for selections 0 and 3, and the inverse of sel[0] for selections 1 and 2. The second mux uses a constant ground for selection 0, **sel[0]** for selections 1 and 2, and HIGH for selection 3. Both multiplexers use **sel[1]** and **cin** as selector wires. The first mux outputs the value for **s** and the second mux outputs the **cout** bit. Here is a drawing of what this looks like:

**Figure 7**

* + Below is the source used to create the Full Adder and was the layout of the entire module[[5]](#footnote-4).

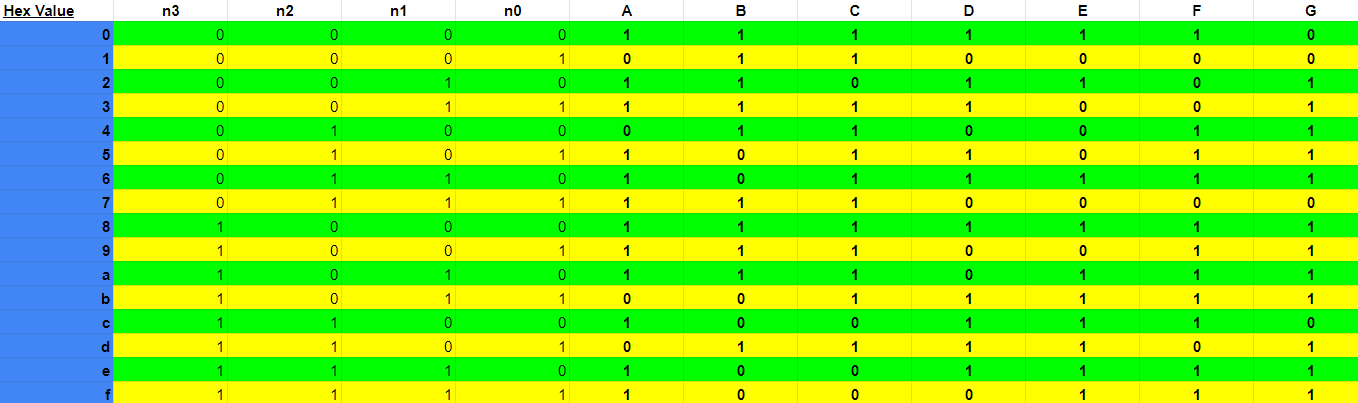
**Figure 8**

Incrementer

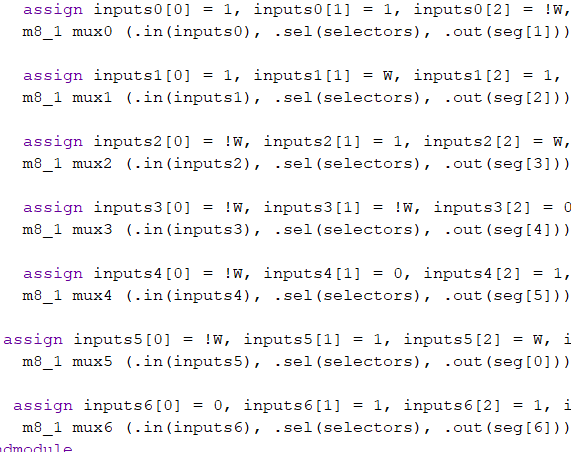
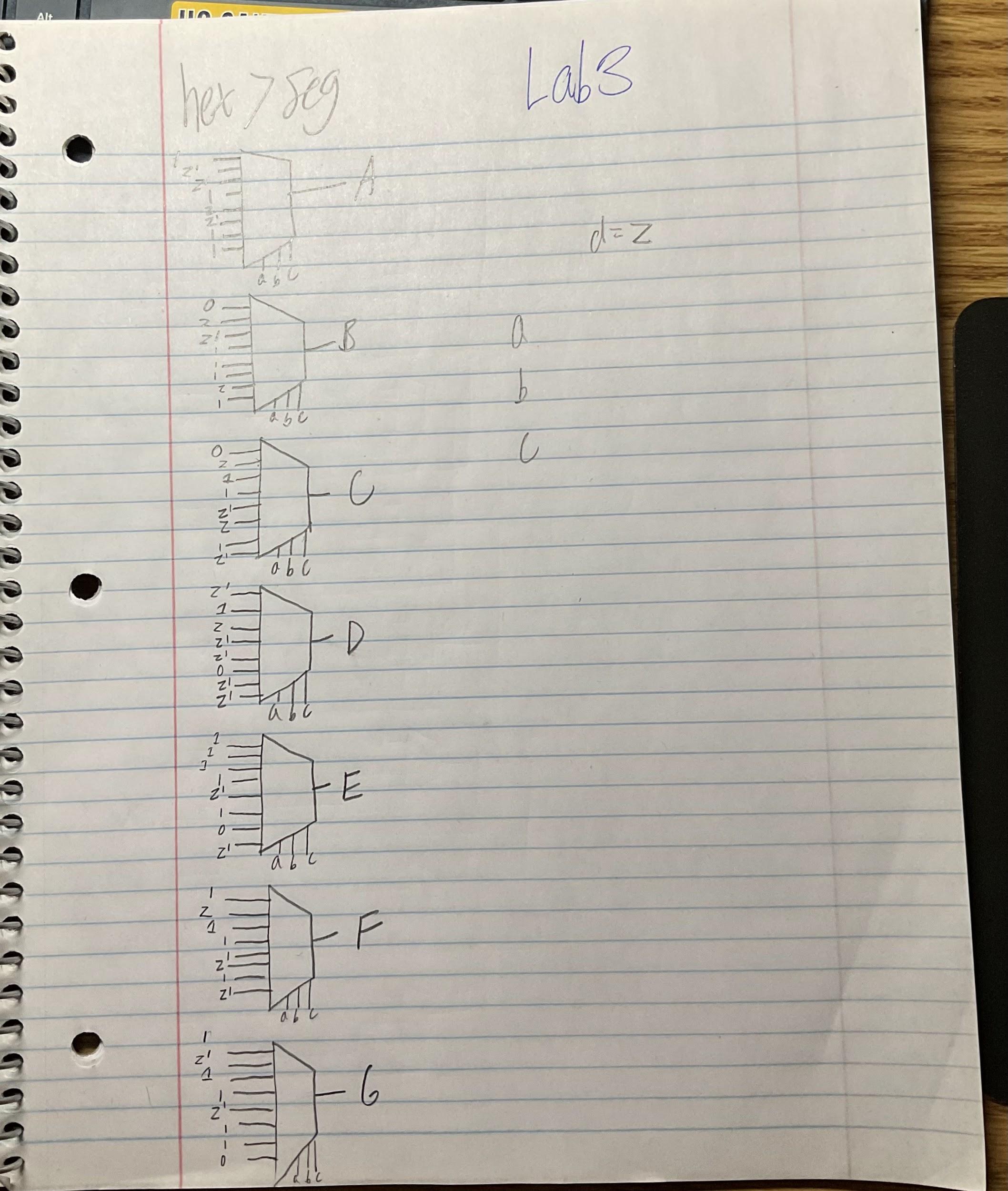
* Inputs
  + **a[7:0]**: this is the inputs of the switch state which are represented as 8-bit binary input.
  + **b[1:0]**: this is the inputs of the btnL and btnC, which need to be extended with leading zeros. Because this counts as a 2-bit input, there need to be six leading zeros. This can be accomplished by busing 6 wires all connected to the ground along with these two inputs as the two least significant bits. Here is a snippet of how that is accomplished:
  + **Figure 9**
* Outputs
  + **s[7:0]**: this output holds the sum of both inputs, an 8-bit number.
* Design of Implementation
  + Take eight full adders. For each adder, one input should be one bit from ‘a’ input and one from the ‘b’. Note that the index of each full adder should match the index from the bus of ‘a’ and ‘b’. Create a bus of 7 wires in which the carry-out of one full adder is the carry-in for the next. The output for each full adder should be assigned to the respective ‘s’ output of the Incrementer. This builds what the output should look like.

Hex7seg

* Inputs
* n: bus of 4 wires
* Output
* seg: bus of 7 wires as inputs to the seven-segment display
* Implementation
* The purpose of this module is to convert 4 bits into hexadecimal values and display that value on the seven-segment display. To do this, use seven 8 to 1 multiplexers in series with each other according to the truth table:

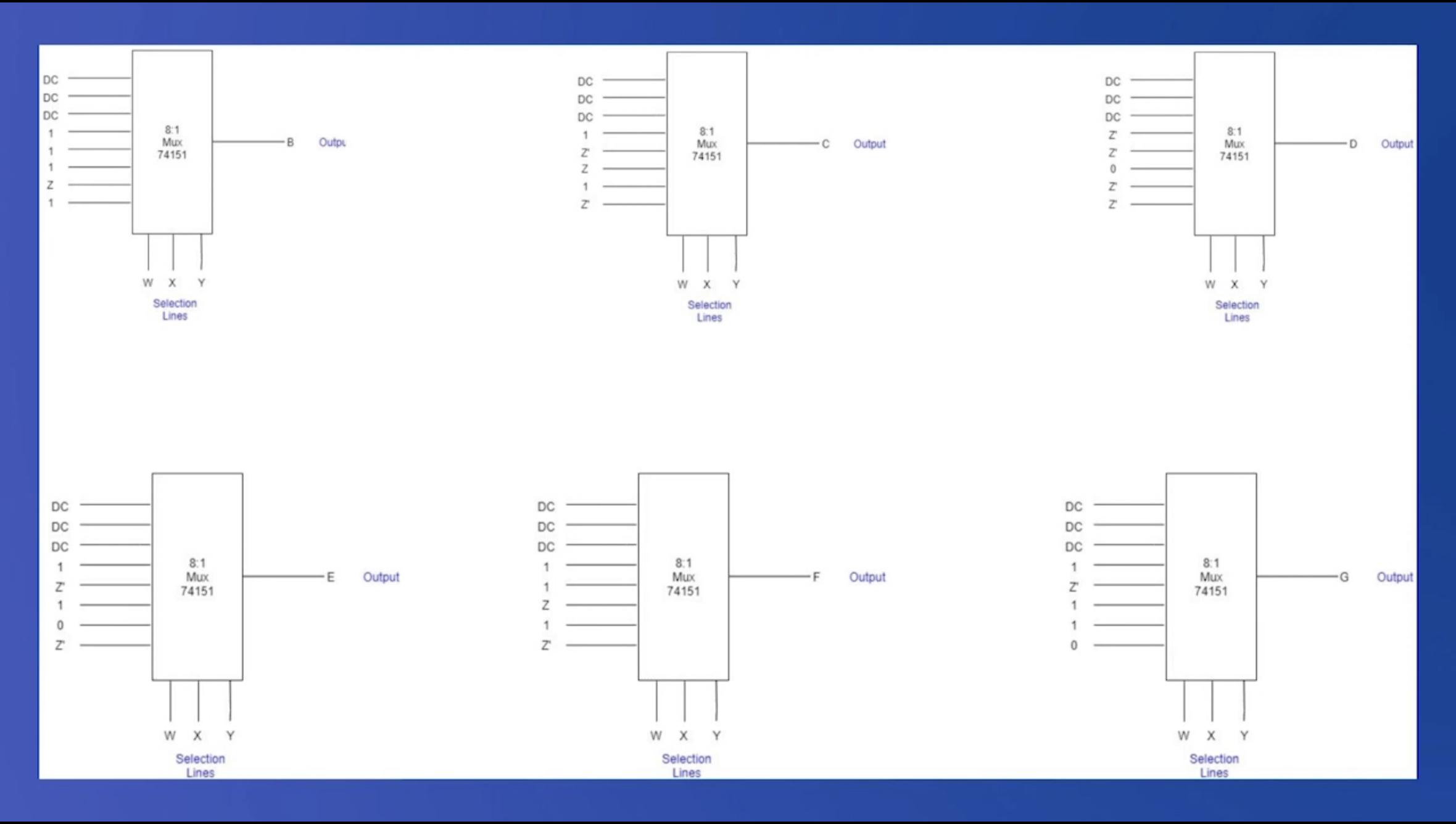
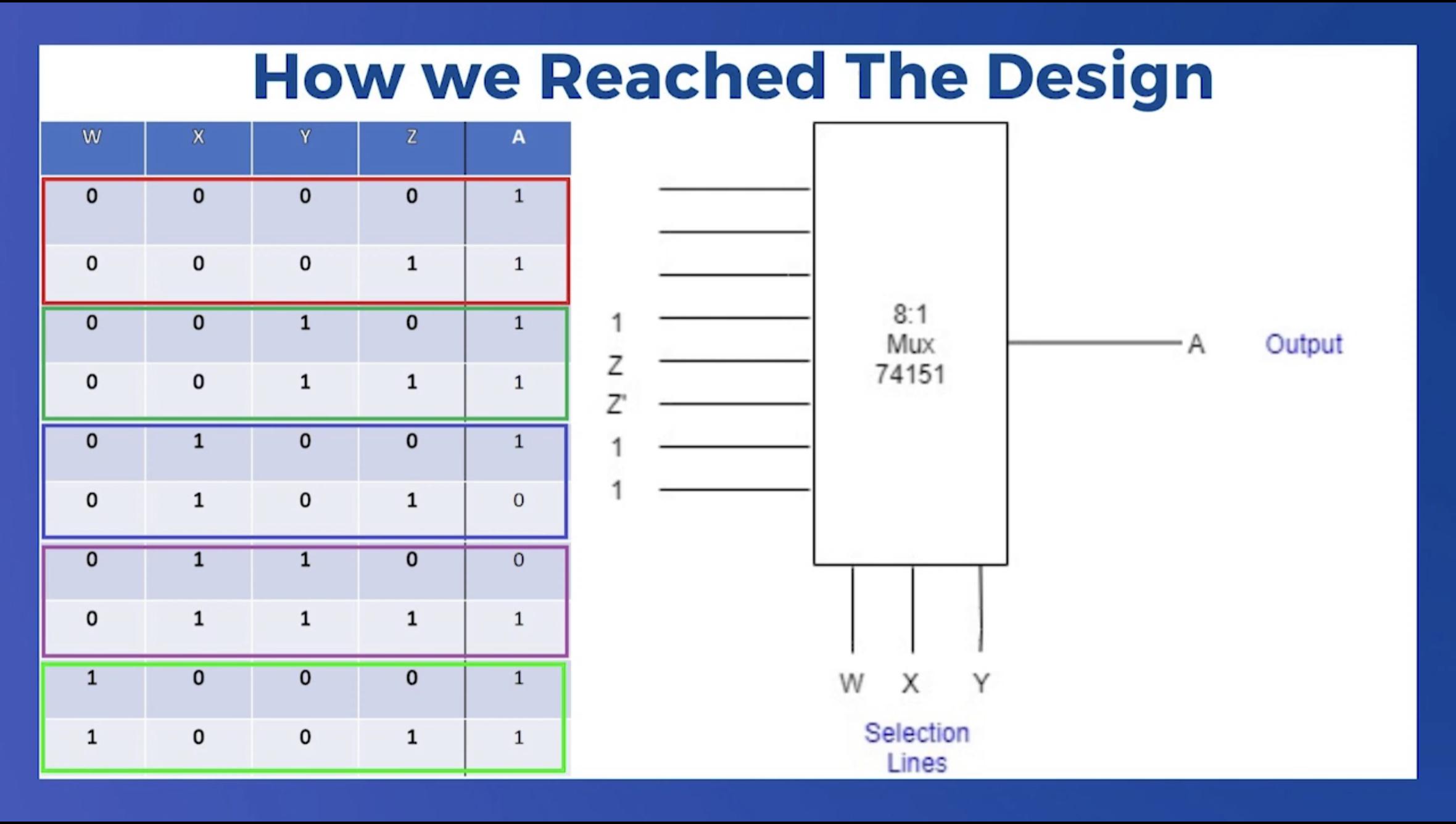


**Figure 10**



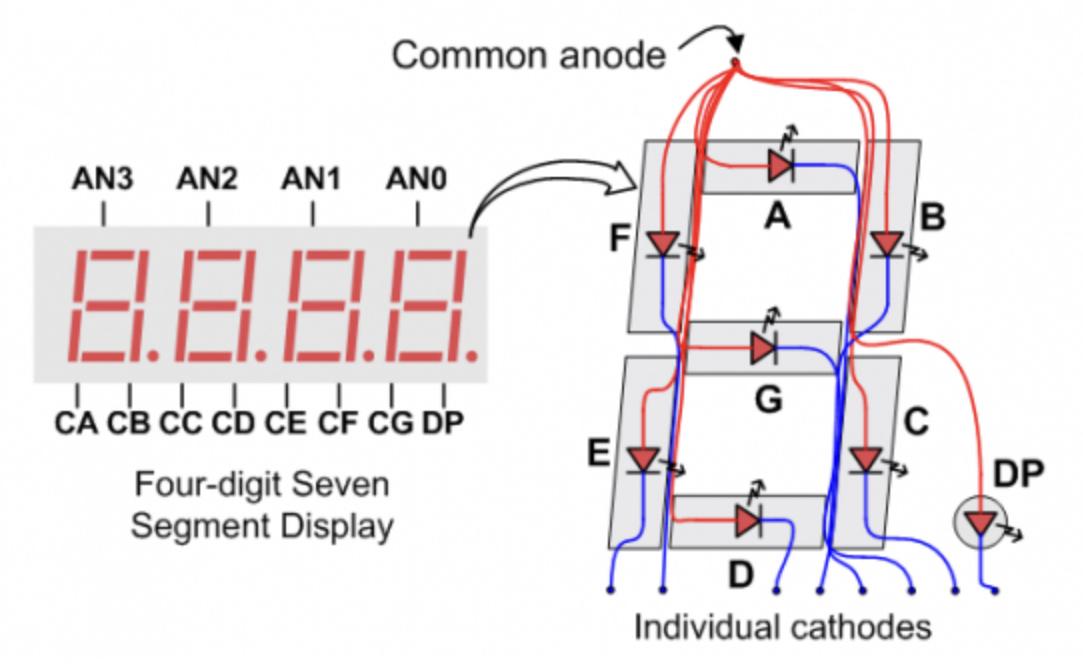
**Figure 11 Figure 12**

* This portion of the lab was very similar to *Lab 2*[[6]](#footnote-5) because of the same truth table. Much of the logic from this module matches what is in Lab 2. Notice the inverter in the figure. This is used because of how the truth table was created. For certain inputs, there need to be inversions to stat consistent with the created truth table. Fortunately, online sources proved helpful for this module and were able to provide tips on how these multiplexers should be mapped. Here are their diagrams[[7]](#footnote-6):



**Figure 13 Figure 14**

And how the seven-segment display is meant to be mapped for this lab:

**Figure 15**

**Testing & Simulation**

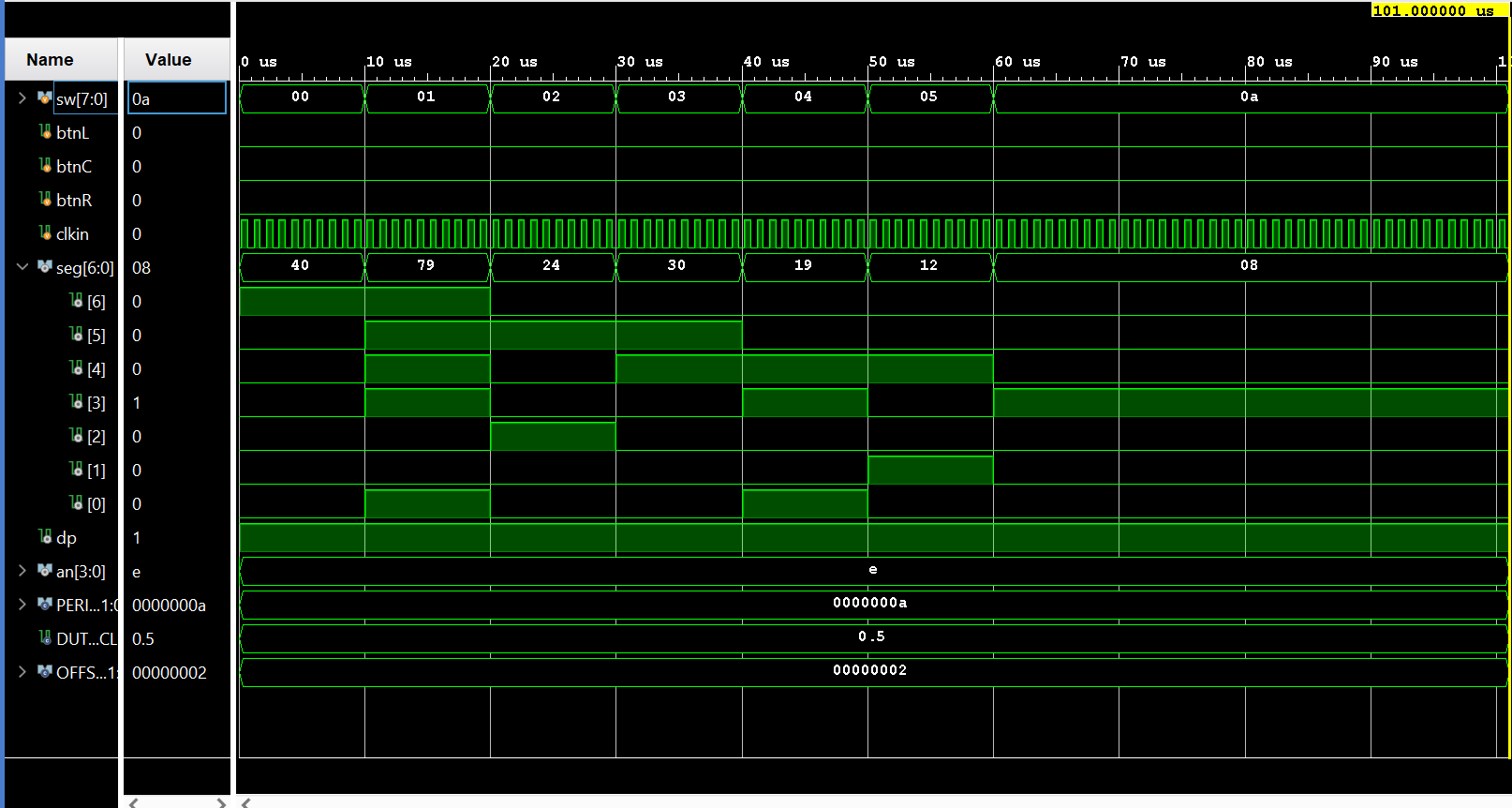
Testing Lab 3 became quite easy when one knows how to use a testbench. When creating test benches for each module, be aware of what the inputs and outputs are. After many trials and errors, the testbench was found to not have the correct inputs and outputs, which prolonged the module-testing period. After this problem was fixed, testing the lab as a whole became quite exciting. This is very helpful especially for the **hex7seg** module because of the meticulous placement of LEDs. For example, if one value of the truth table is incorrect, then the incorrect lights would either be on or off when they should be in their correct place. For the purposes of this lab, only values ‘2’, ‘6’, ‘a’, ‘b’, and ‘f’ were tested because these are the values with the most variation of HIGH and LOW segments. If these values were working correctly, then it would be safe to assume that the others would work just as well. For this reason, the logic in Lab 2 became essential because of the similarities between the lab. The difference is that Lab 3 uses multiplexer logic, while Labe 2 uses Full Adder logic.

Some corner cases that became evident as one of the sources for this lab were not consistent with the naming conventions that were used. For example, our naming conventions for segments ‘0’ through ‘6’ were ‘a’ - ‘f’, which the source was mapped as ‘f’ rounded through ‘e’ (See **Figure 15**). This became unnoticed and created many problems throughout the lab. When this inconsistency was caught, the appropriate measures were taken and the corrective mapping was created. Another problem with this lab was the logic of using 8 to 1 multiplexers was not too apparent or straightforward. Although with some practice and learning the sequence of mux with truth tables, the module was able to be fitted appropriately.

In addition to correcting this source in our lab, there was confusion about which bit was the most significant: position 0 or position n-1. This became apparent when loading the hex7seg with the 4-bit inputs. For example, there was the use of 0, 1, 2, 3 instead of 3, 2, 1, 0. This became problematic and ultimately needed to be mapped so that the correct significance would be used in every instance of the module.

**Results**

Overall, the lab was very successful. The modules created worked as intended, and the test benches outputted promising results. After the many trials of trying to figure out the issue with the **hex7seg** module, the overall performance of the lab went very smoothly. Here is some information about the outputs from the lab:



**Figure 16**

**Questions**

1. Dig\_sel is oscillating at about 100MHz. This is the default value loaded into the constraint file for the Basys3 Board.
2. I did not observe any flickering with the seven-segment display. For some, the reason there may be flickering is that their oscillations were too low and they were noticeable by the human eye. Normally a higher clock cycle would limit the amount of flickering going on with the display.
3. See the [**Appendix**](#kix.ppszpbrm8dav) for the schematics of the top-level, Incrementer, hex7seq, FullAdder, and all three of the multiplexers
4. See the [**Appendix**](#kix.ppszpbrm8dav) for the Waveform PDF of our output.

**Conclusion**

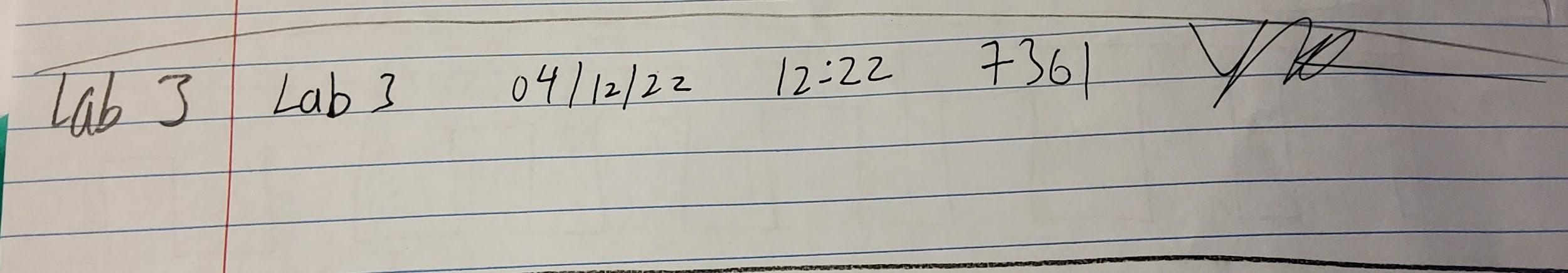
This lab pertains to many real-world applications. In the case of computers, they need to be able to perform thousands of calculations in order to be efficient at what they do. In addition to calculations, their hardware needs to be reliable for many years of its life. This lab taught the importance of a clock and how they are used in computers, along with the importance of how different components work together. This lab also taught the importance of testing using test benches. Throughout this lab and previous labs, test benches were not used all that much. But it was brought to light the importance of using them in order to ensure the module’s functionality.

This lab also brought to light the efficiency of clock cycles. It is important to know how the clock works and its function throughout the program. For instance, the clock in this lab was used to refresh the seven-segment displays. Without the refresh, the displays would constantly show its previous value, and not update to the most current one. This is also very prominent in current displays on all screens today. It also shows the high speeds of hardware and the clock cycle: producing ultra-high frequencies to refresh the displays.

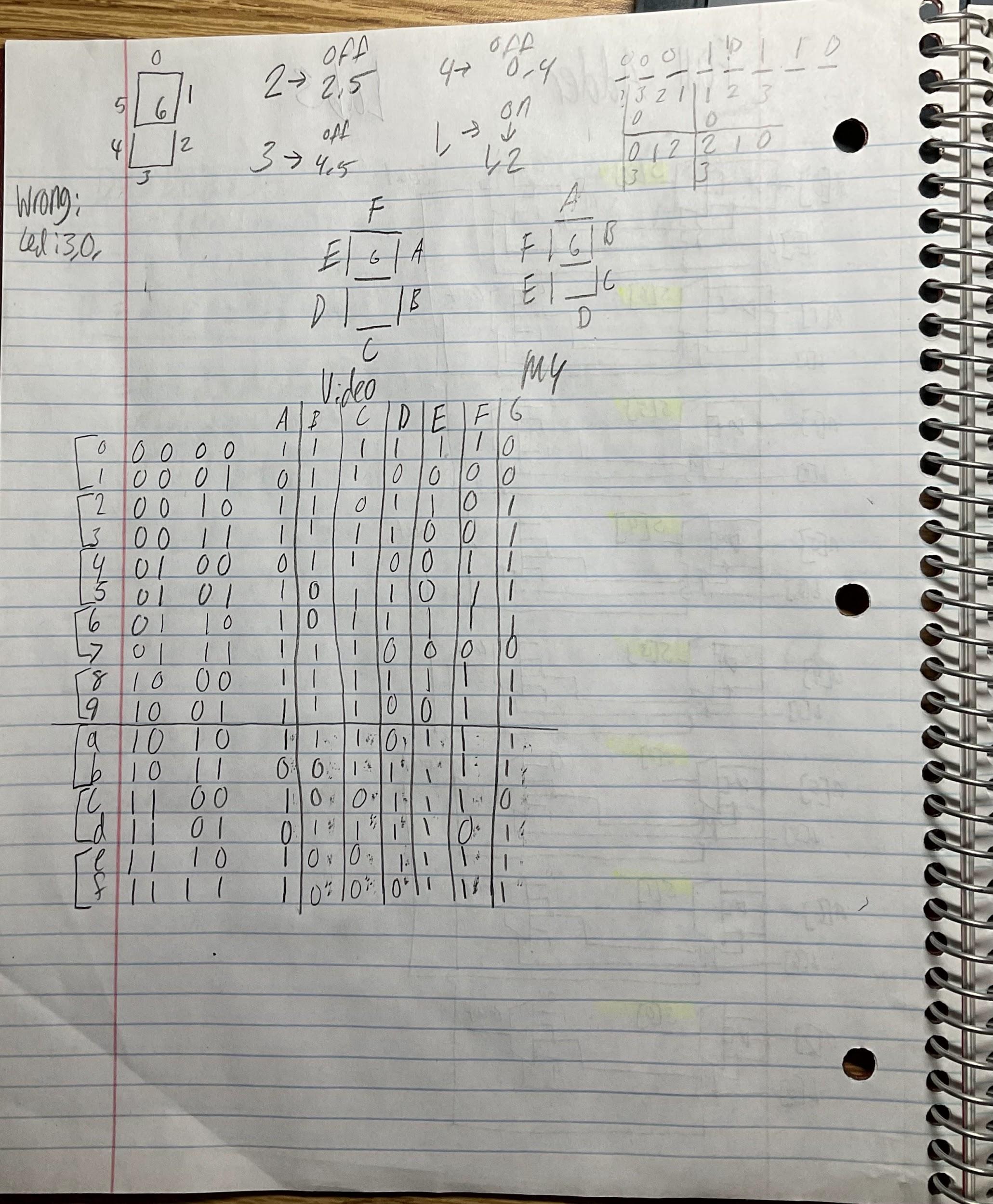
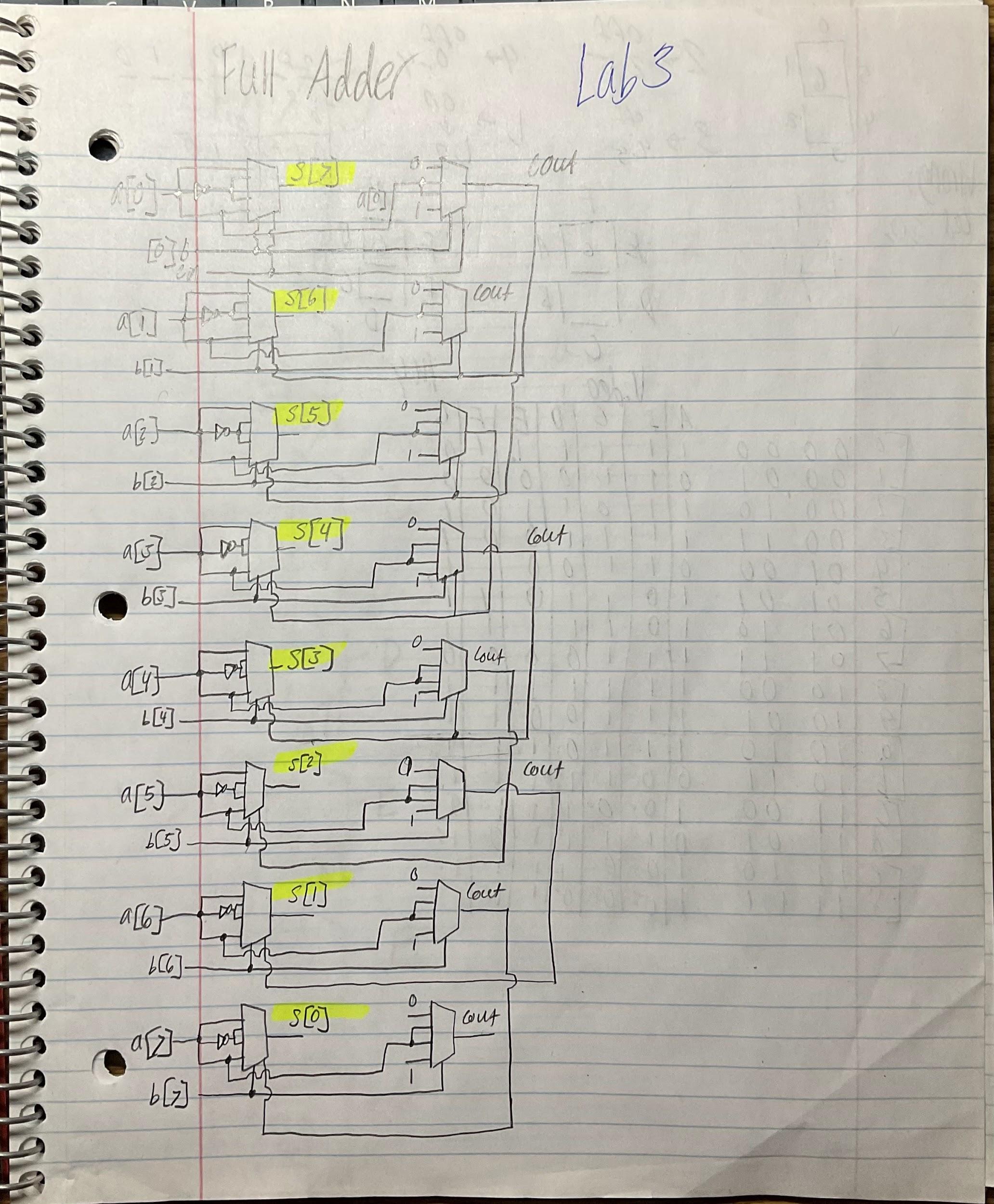
If this lab were to be done again, there should be one type of recommended measure. Ensure that every module works as intended before applying it to the final design and ensure that the sources used are mapped correctly to the sources of the final design. This lab taught the importance of these two rules and how they should be applied in all future labs. Optimization can be used in the Full Adder module. It would have been easier (for simplicity) to only use the logic gates that were learned instead of using multiplexers. But it is understood the efficiency of how a mux can behave.

**Appendix**

* Lab Check-Off



Checked 4/12/2022 12:22PM **CODE: 7361**



* The next pages are the supplementary documents and the Verilog code shown in **Figures 4, 5, 6, and 9.**

1. See [**Appendix**](#kix.ppszpbrm8dav) for details on mapping. [↑](#footnote-ref-0)
2. https://classes.soe.ucsc.edu/cse100/Spring22/lab/lab3/lab3.html [↑](#footnote-ref-1)
3. https://classes.soe.ucsc.edu/cse100/Spring22/lab/lab3/lab3.html [↑](#footnote-ref-2)
4. https://classes.soe.ucsc.edu/cse100/Spring22/lab/lab4/lab4.html [↑](#footnote-ref-3)
5. <https://www.geeksforgeeks.org/1-bit-full-adder-using-multiplexer/#:~:text=Multiplexer%20and%20Full%20adder%20are,and%20Carry%20as%20an%20output>. [↑](#footnote-ref-4)
6. Dimas, Gabriel *Lab 2: Seven\_Segment\_Display* [↑](#footnote-ref-5)
7. https://youtu.be/BkyaHwt6l84 [↑](#footnote-ref-6)