Final Project

Design of a Simple 4-bit Processor

Julia Rasmussen (002119596)

rasmussen.j@northeastern.edu

Brian Jiang

rasmussen.j@northeastern.edu

Submit Date: 04/22/2022

Due Date: 04/22/2022

Abstract

For this final project, a simple 4-bit processor was built using Quartus Lite Prime. First the circuits for the adder/subtractor, multiplexer, AND operator and XOR operators were drawn and tested individually. Then, a decoder was used in conjunction with the block schematics for each of the parts in order to integrate all four circuits into the 4-bit processor. This final circuit was then compiled, and uploaded to the DE1-SoC board via a USB connector.

Introduction

The purpose of this project was to incorporate everything we had learned about in previous labs into a larger project, specifically using BCD diagrams in Quartus Lite Prime to make a functional 4-bit processor. This lab built upon several previous labs, such Lab 6 and Lab 7, in which the basics of Quartus Lite Prime and BCD diagrams were learned, as well as all the labs before that in which the DE1-SoC board was used. It was also assumed that we already knew how to make BCD diagrams, as well as how to export them as block schematics and then incorporate them into another project. Constraints of this project included the limitations of Quartus Lite Prime itself, as well as the limited number of pins, and thus limited number of inputs and outputs on the DE1-SoC board. This project assumed a familiarity with logic gates, BCD diagrams, multiplexors, and the DE1-SoC board.

Lab Discussion

The hardware used in this lab consisted of the DE1-SoC board, a power cable to power the board, a desktop computer to run Quartus Prime Lite on, and a USB cable to connect the two computers to each other. On the DE1-SoC board, the main components used throughout the lab were switches SW0 to SW9, the LEDs LED0 to LED9, and the push button KEY0. The software used for this lab consisted of Quartus Lite Prime, through which a BCD system for the 4-bit processor was designed.

Results and Analysis

1. The 4-bit adder-subtractor

In order to make an adder-subtractor, four full adders were connected in the configuration seen below in Figure 1. An enable bit was added so that the circuit would work as both an adder and a subtractor, depending on whether the enable bit is a 0 or a 1. Additionally, XOR gates were hooked up with the enable bit and the second bit of each pair, so that the circuit would perform the function correctly.

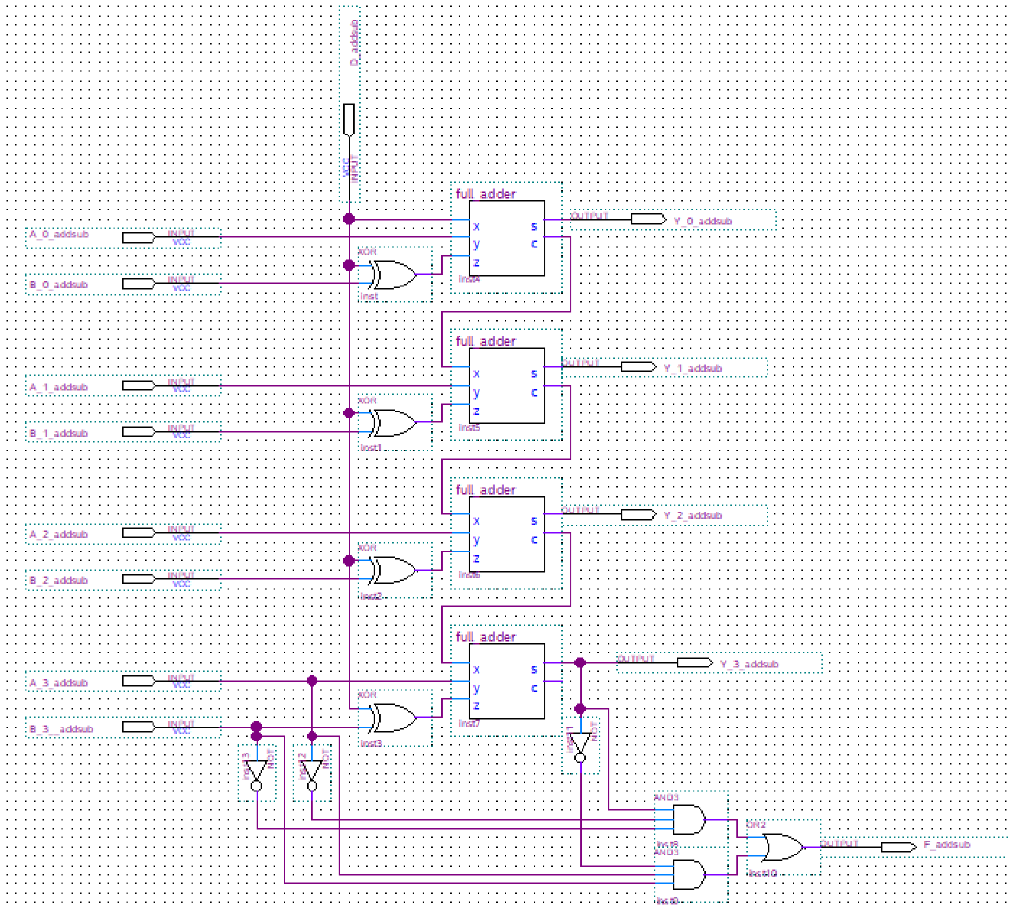

Figure 1: The adder-subtractor circuit.

One thing to note is that the final carry bit is not used at all. Instead, an overflow value of F is determined from two AND gates that are hooked up to the A_3_addsub, B_3_addsub, and Y_3_addsub values, and then connected to the final output of F_addsub with an OR gate. The first AND gate consists of a NOT for A_3 and B_3 off, and then the value for Y_3. For the second AND gate, A_3 and B_3 are attached as is, and Y_3 is connected to a NOT instead. With this configuration, an overflow value is achieved when needed as determined by the two's complement model.

2. The 4-bit multiplexor

The next function for the 4-bit processor was the multiplexor. In order to make a 4-bit multiplexer, a 2-bit multiplexer was made first, as can be seen below in Figure 2.

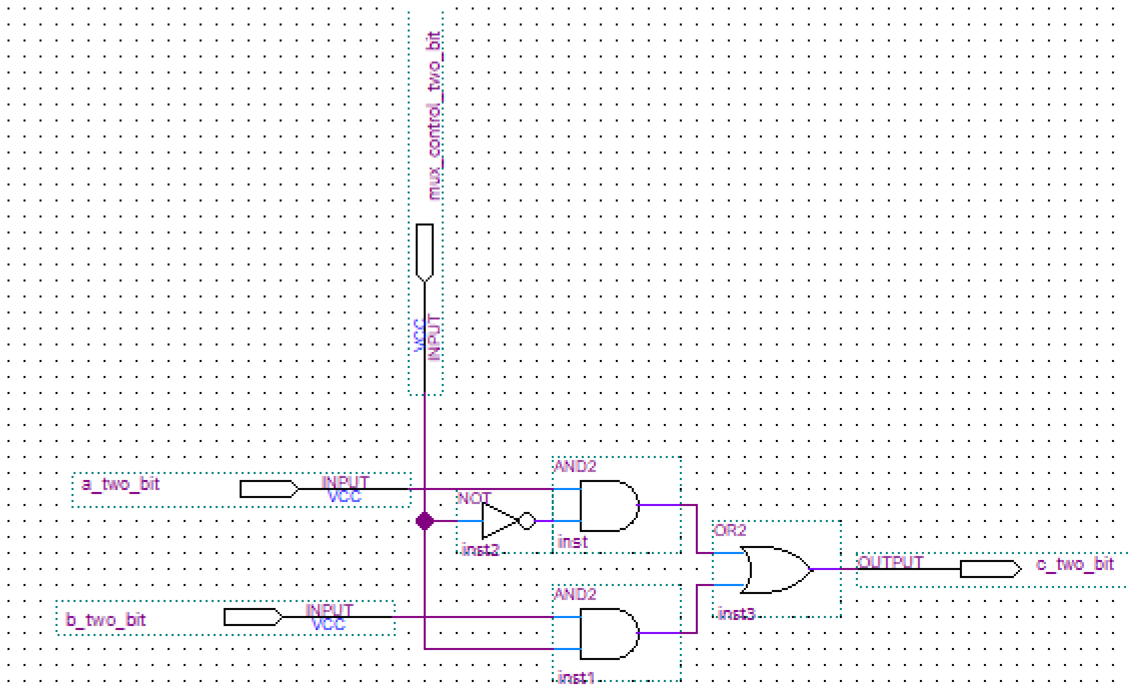

Figure 2: The circuit for the 2-bit multiplexer.

As the function of a multiplexor is simply to output the value of the input specified by the control bit, the two inputs were simply directed into separate AND gates, using a NOT with the control bit for the first input, and leaving the control bit as is for the second input. These AND gates were then routed into an OR gate, which allows both outputs through, but as the control bit will always make the result of the AND that is not needed 0, only the value of the chosen bit will pass through. The result of this can also be seen in the waveform diagram found in Figure 3.

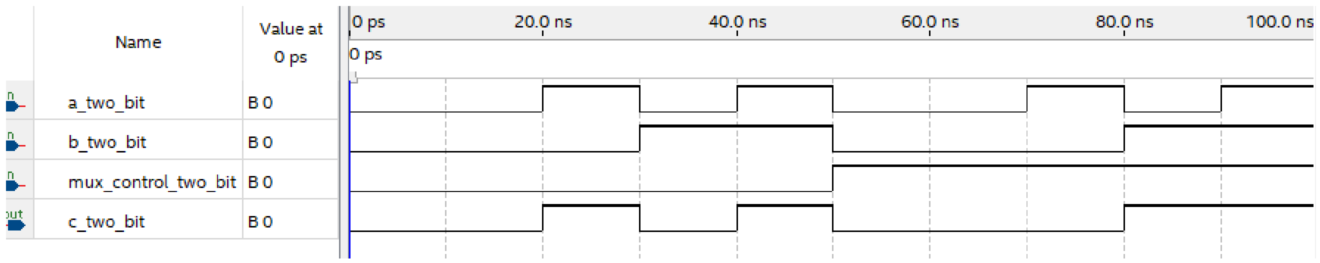

Figure 3: The waveform diagram of the 2-bit multiplexer.

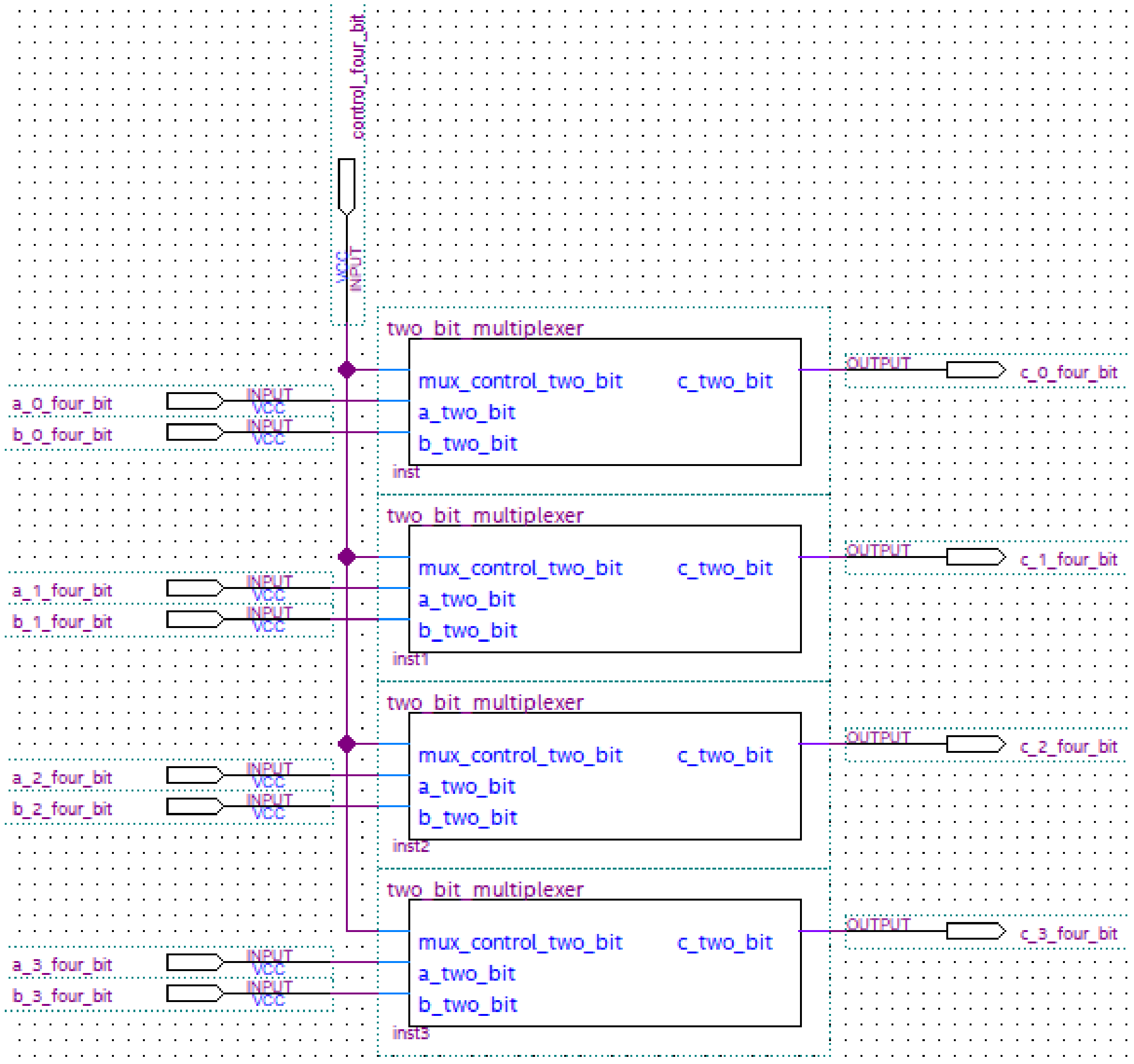

Figure 4: The 4-bit multiplexer.

This 2-bit processor was then exported as a block diagram, and used in the circuit for the 4-bit multiplexer seen in Figure 4. This allows the user to control which of the two numbers will be the output of the multiplexer; if the bit is set to 0, the output will be the first number, if the bit is 1, the output will be the second one.

3. The 4-bit binary multiplier

This next part of the project didn't end up being in the final 4-bit processor, but is nevertheless interesting. This circuit takes two 4-bit numbers and multiplies them together. It has eight inputs and eight outputs. By using a combination of half-adders, full-adders, and AND gates, as seen in the BCD diagram below in Figure 5, this multiplication can be achieved.

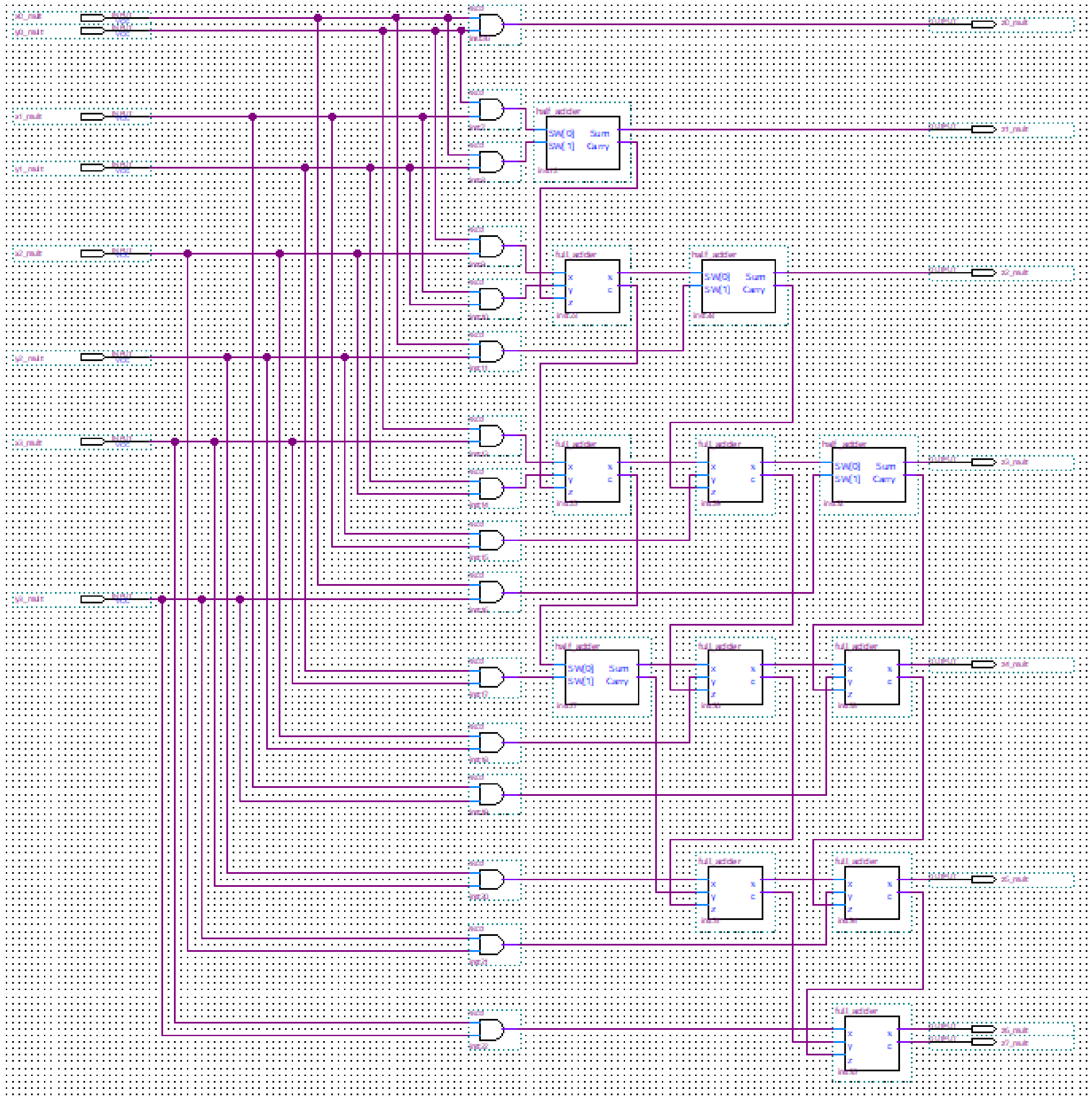

Figure 5: The circuit for the 4-bit multiplier.

4. The bitwise “AND” operator

For the four bit bitwise “AND” circuit, all that needed to be done was connect each digit from the two four bit integer inputs to an AND gate, and output the resulting bit to the same digit for the output. An implementation of this can be seen in the figure below, Figure 6.

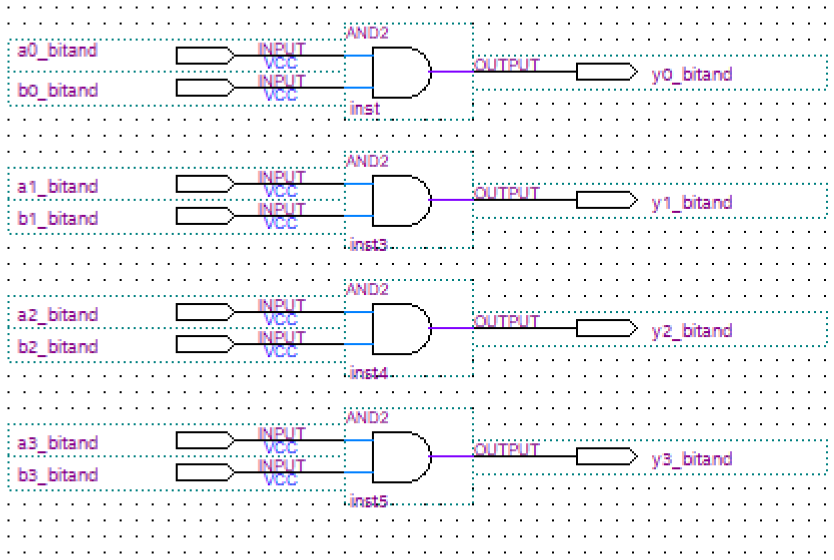

Figure 6: The circuit for the bitwise AND operator on the two 4 bit numbers.

5. The bitwise “XOR” operator

This part of the project was similar to part 4 in its implementation; all that needed to be done was connect the two inputs for each of the digits in a XOR gate, and then output the resulting bit into the same digit for the output. This can be seen in Figure 7 below.

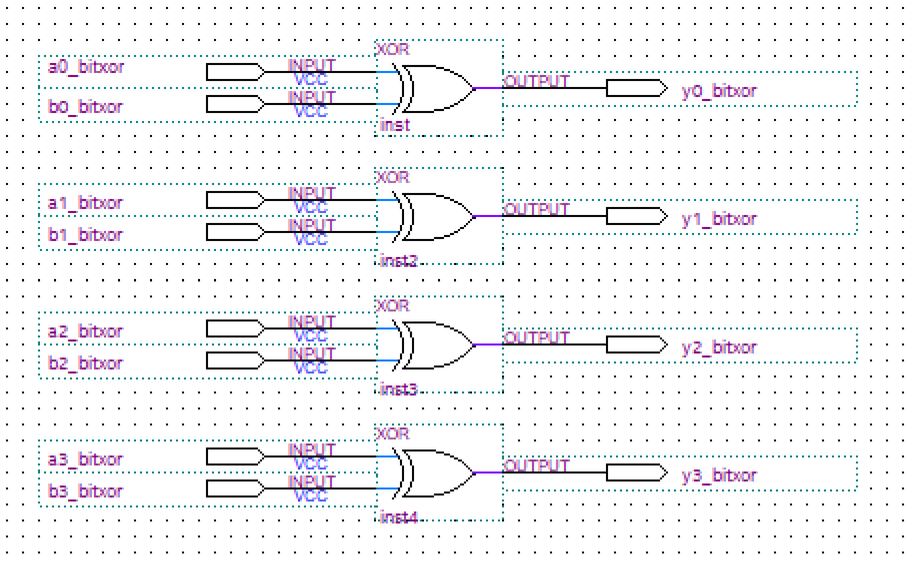

Figure 7: The circuit for the bitwise XOR operator on the two 4 bit numbers.

6. Putting it all together: The 4-bit Processor

This last part of the project consisted of incorporating the circuits designed in parts 1, 2, 4, and 5 into a functional 4-bit processor. This turned out to be relatively straightforward to implement; all eight inputs were connected to each of the circuits, and would be routed into an OR gate for the specifically outputted bit. However, in order to determine which of the four different outputs should be used for the final output, a decoder was needed.

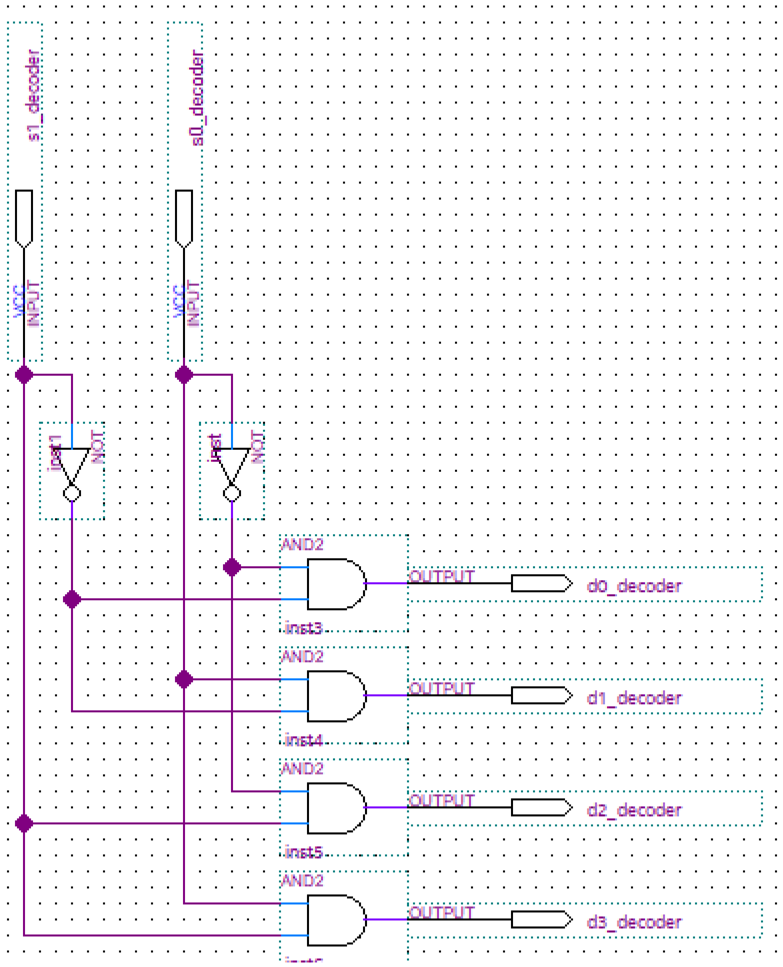

Figure 8: The circuit for the 2-bit decoder.

This decoder outputs a 1 for the bit in the position of the binary number represented by the two inputs. That is, if the input is 00, the zeroth bit will output a 1, and all the other bits will output a 0. If instead 10 is inputted, the second bit will output a 1, and the other bits will output a 0 instead. This decoder was then exported as a block diagram, and hooked up to the final circuit of the 4-bit processor, seen in Figure 9.

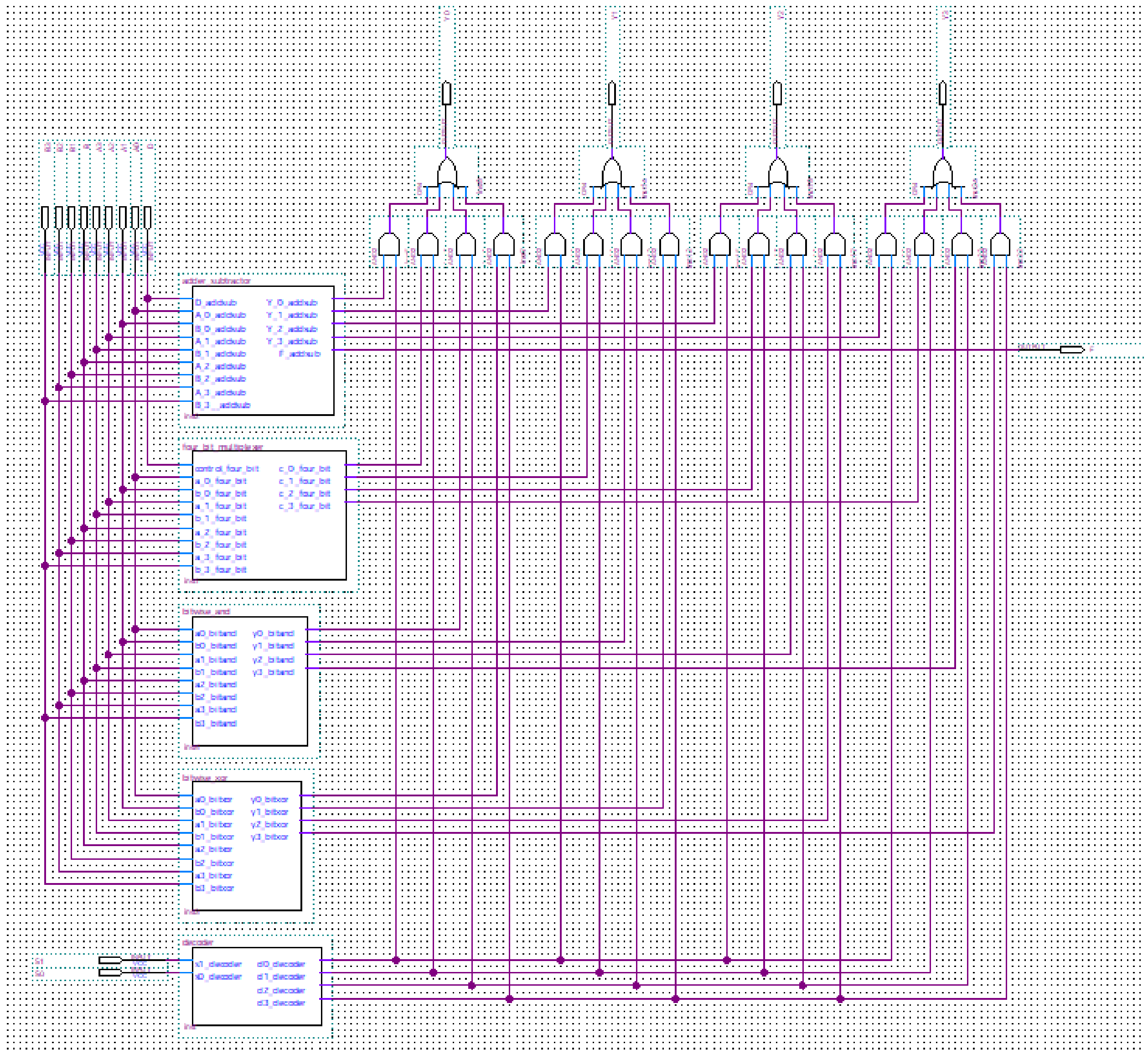

Figure 9: The circuit for the 4-bit processor.

As can be seen in Figure 9, the 4-bit processor consisted of eight inputs for two 4-bit numbers, as well as two inputs for the 2-bit control. There are four processes that can be done by the 4-bit processor: add or subtract, use a multiplexor, the bitwise AND operation, and the bitwise XOR operation. Both of the two 4-bit numbers are connected to all four of these processes, and then connected to an AND gate that either enables or disables the value, which is then routed to a four-input OR gate, so that the correct value will be outputted at each of the bits. The other end of the AND gate is connected to the relevant bit from the decoder. That is, each adder or subtractor output is in an AND gate with the zeroth output of the decoder, then the multiplexer outputs are in an AND gate with the first output of the decoder, and so on. This way, the unwanted outputs will always default to 0, and only the value of the wanted outputs will be routed to the final output.

Conclusion

This project went relatively smoothly; all of the components ended up working well, and the only thing that took slightly more time to implement was the overflow bit on the adder/subtractor circuit. I can definitely see how this lab could translate to a real-world setting; this could be scaled up to an 8-bit processor, a 16-bit processor, so on and so forth. Additionally, the knowledge of how to build BCD diagrams that are dependent on each other helps with the organization of the overall circuitry, just as how using classes helps with organization and readability when coding. Knowing how to make BCD diagrams from a given problem is just a really useful skill all around, and will no doubt be useful in future courses, research, and coops.