

Two Bit Arithmetic Logic Unit

Course: Digital Electronics Lab

Fall 2025

Section 001

Date: 12/10/2025

Submitted By: Nathaniel Muhs

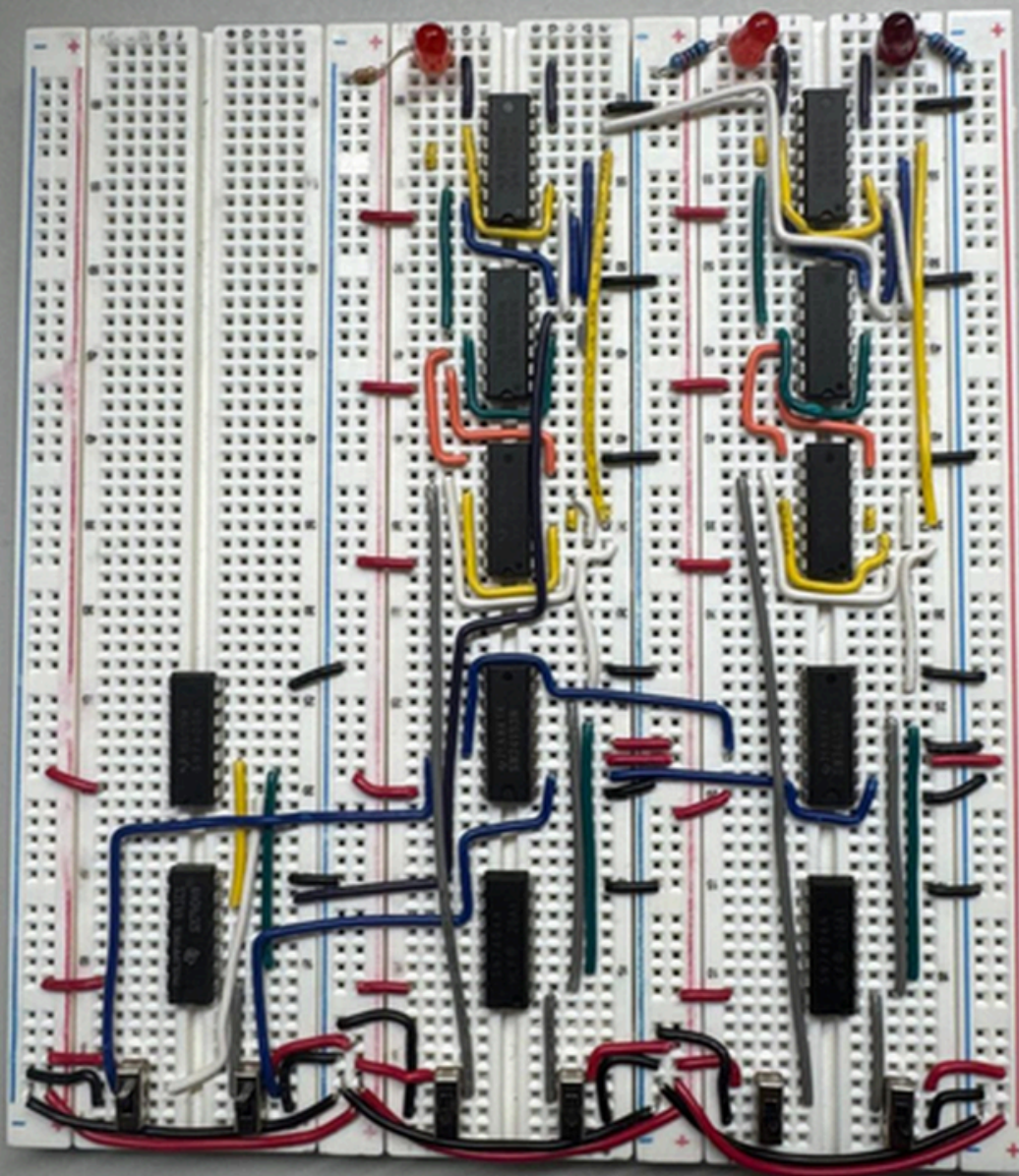


Table of Contents

[Objective](#)

[Procedure](#)

[Half Adder](#)

[Full Adder](#)

[Addition operation](#)

[Subtraction operation](#)

[Increment operation](#)

[Decrement operation](#)

[Selection between the 4 operations](#)

[Final Implementation](#)

[Simulation](#)

[Construction in Logisim](#)

[Testing Simulation](#)

[Breadboard Circuit](#)

[Construction](#)

[Testing](#)

[Future work and Conclusion](#)

Objective

The goal of this project is to design, simulate, and build an Arithmetic Logic Unit (ALU) for two, two bit inputs. The 4 operations that are to be implemented by the ALU will be controlled by a two input select. The select operation will implement the addition, subtraction, increment, and decrement operations on the A input. The Addition and Subtraction operations will use the B input as addend or Subtrahend respectively. Finally one of the most challenging additions to the objective is that the circuit must be built using only NAND, NOR, NOT, and XOR gates. No AND or OR gates can be used for the creation of the project.

Procedure

The design phase of this project included heavy abstraction compared to what was suggested in our lab handout. The following subsections include a breakdown of how the operations were designed and implemented for the simulation phase

Half Adder

After much research our team decided to step away from the truth table implementation and instead step toward a level of computer engineering abstraction for the design of the ALU. The following lays out the truth table and simplified circuit for a half bit adder.

Half Adder			
A	B	C Out	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

Figure #1: Half adder truth table.

After creating a k-map for both Sum and C Out we used the k-map method to find the following minterms

$$\text{Sum} = X \text{ OR } B$$

$$\text{C Out} = A \text{ AND } B$$

Using these minterms the following circuit was created for the half adder.

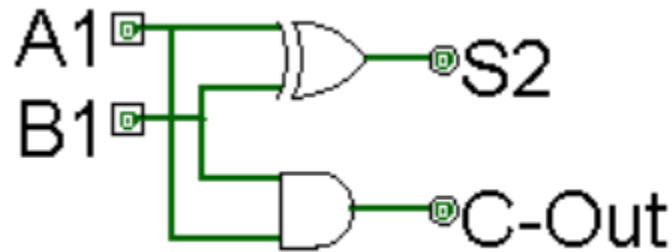


Figure #2: Logit circuit for the half adder

Of course the circuit shown in figure #2 would have to be altered to fit the objective constraints listed for this project. Altering the circuit shown in figure #2 gives us figure #3. This process of adding a NOT, in front of an AND gate and a NOT, in front of an OR gate, will be carried out throughout this entire project.

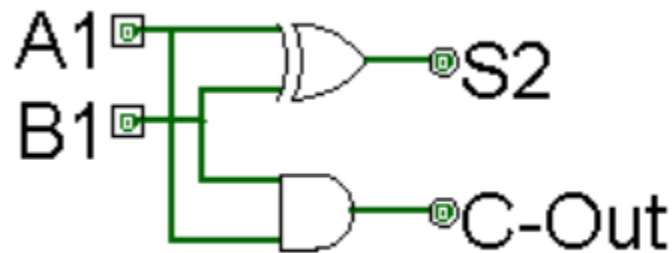


Figure #3: Half adder logic circuit that fits design constraints

Full Adder

The full adder carries on the implementation of the half adder. The full adder adds another input called carry in to the truth table. The implementation of the full adder is explained in the addition operation subsection. The truth table for the full adder is pictured in figure #4.

Inputs			Outputs	
A	B	C _{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure #4: Full adder truth table

Once we have the truth table for the full adder we can follow the same procedure as the half adder to find the minterms. First we make a k-map and solve it for the following minterms

$$\text{Sum} = A \text{ XOR } B \text{ XOR } C\text{-in}$$

$$\text{Carry out} = A * B + C\text{-in} * (A \oplus B)$$

With these minterms we can create the logic circuit shown in figure #5. While figure #5 is an effective demonstration of a full adder, we can change to make it relevant for our design constraints. If we had to implement figure #5 we would need two XOR gates, 2 NAND gates, One OR gate, and three NOT gates. While this is not necessarily bad it would require 4 different logic IC chips and consume more power than the alternative. Instead we will rearrange the minterms to create Figure #6 which uses 9 NAND gates.

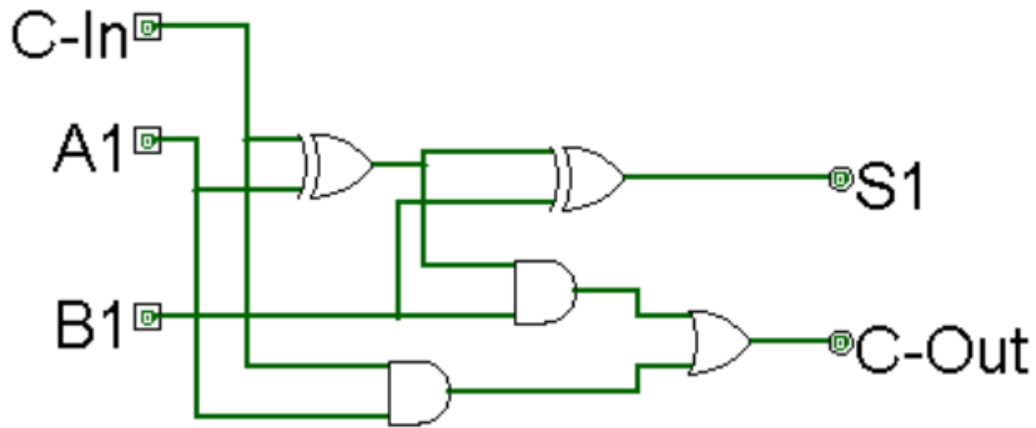


Figure #5: Full adder logic circuit

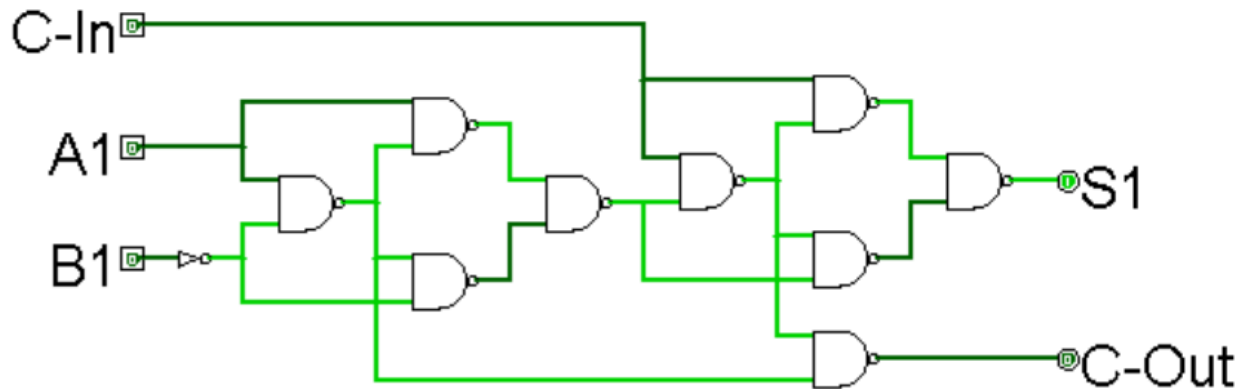


Figure #6: Full adder implemented with nine NAND gates

Addition operation

Usually when people cascade abstract logic blocks for addition using logic they first put a half adder then a sequence of full adders to meet their specifications. Our design cascades ONLY FULL ADDERS. This implementation makes our subtraction operation possible and lets us squeeze by with a few less logic gates.

So, to implement the addition operation we cascaded two full adders in sequence. With this cascade it would be easy to increase the number of inputs in future implementations. The cascaded full adders are shown in figure #7.

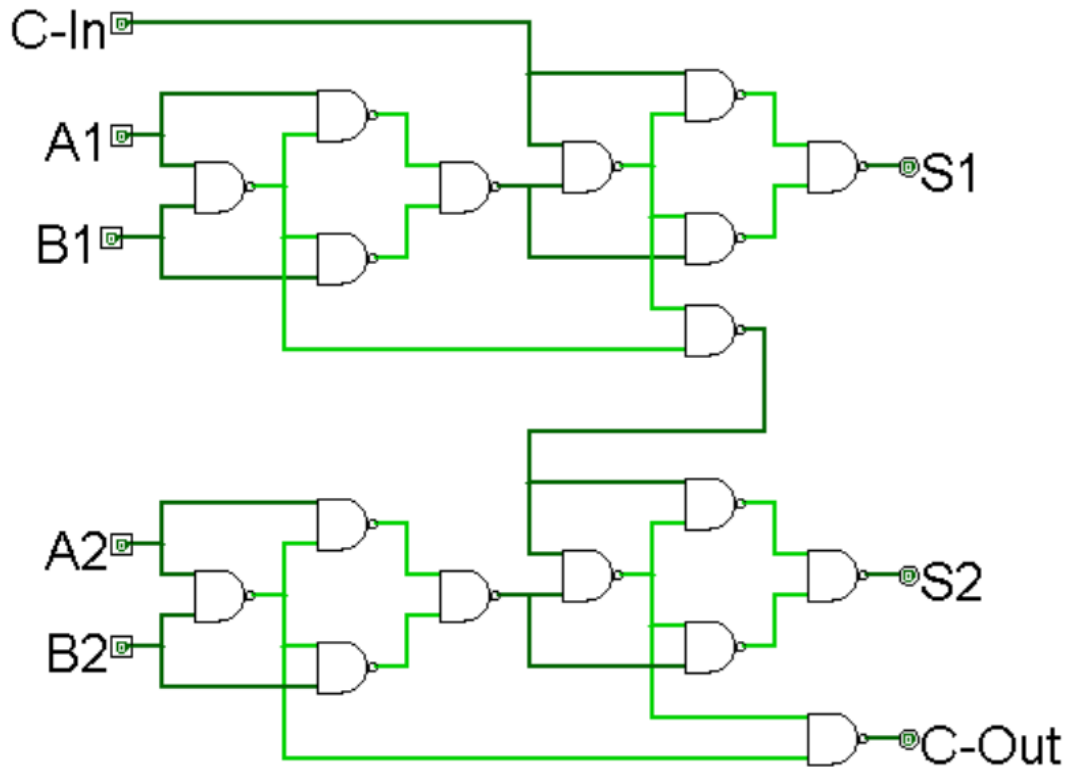


Figure #7: Two full adders cascaded together

Subtraction operation

This is when the benefits of abstraction start to show up. Once we have the implementation of the addition operation we can simplify the subtraction operation. For this operation we use the two's complement method for adding and subtracting boolean numbers. So literally we use two's complement on B (flip the sign on all bits and add one) then simply add that value to A. The name C1 means, control signal one and will come into play when we implement all of these operations. The subtraction circuit is shown in figure #8 and the circuit fitting the design constraints is shown in figure #9.

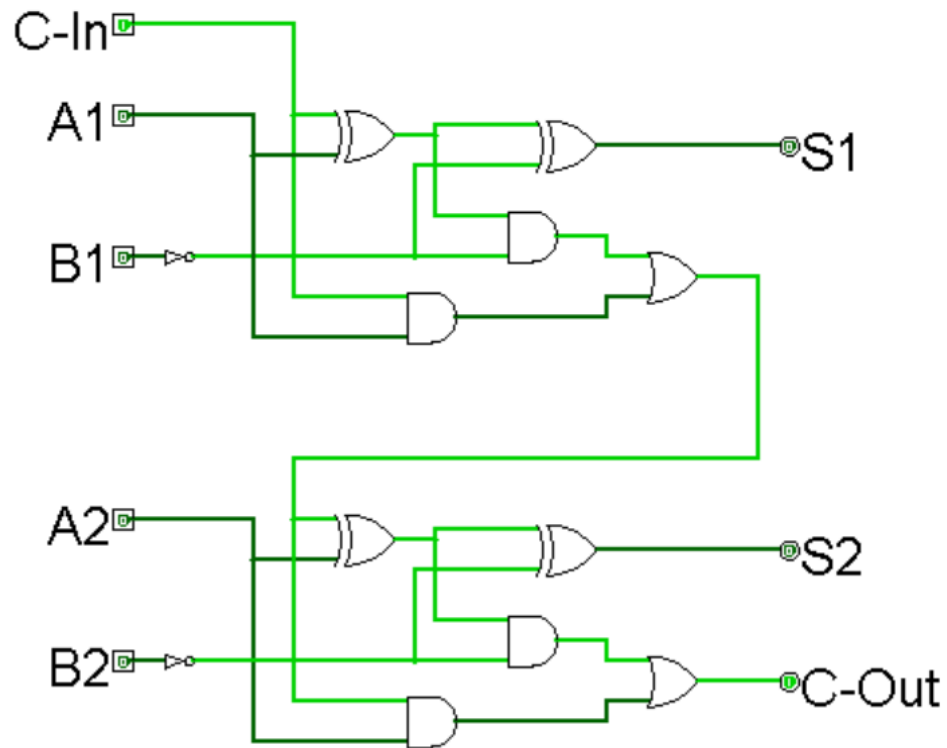


Figure #8: Subtraction circuit implementation

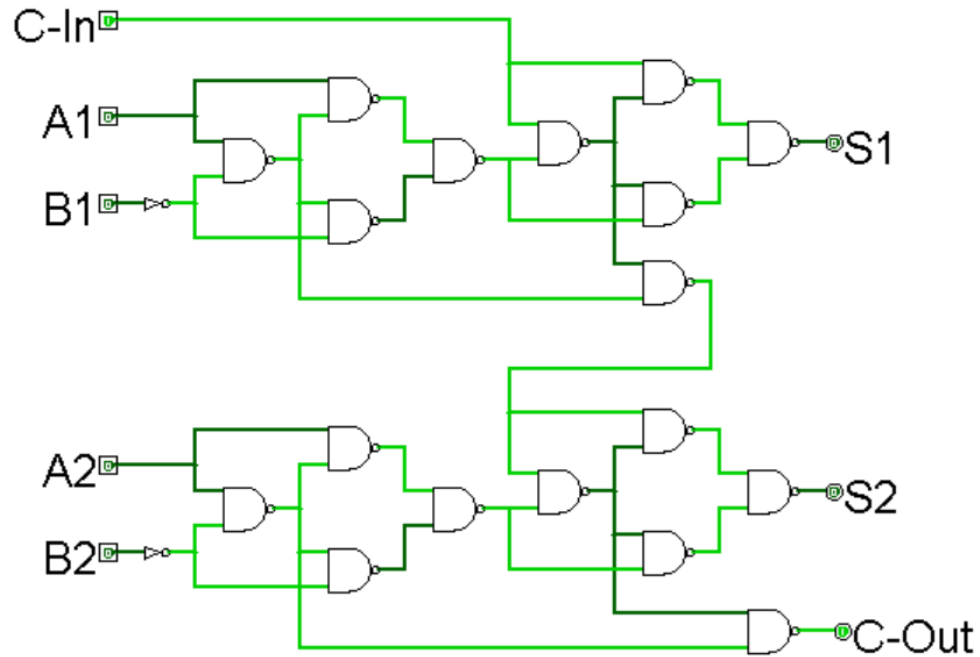


Figure #9: Subtraction circuit implemented with design constraints

Increment operation

Out of all the operations that our team had to implement this was the easiest. There are two ways that A could be incremented, both of which disregard the B input. Firstly you could set the carry in signal to high and disregard all the B lines. The second way to implement an increment operation would be to set the B1 line to high. We decided to set the carry in signal to high which would decrease the number of logic gates we would use in the long run. The circuit implementation of the increment operation is shown in figure #10 with the circuit that fits our design constraints pictured in figure #11.

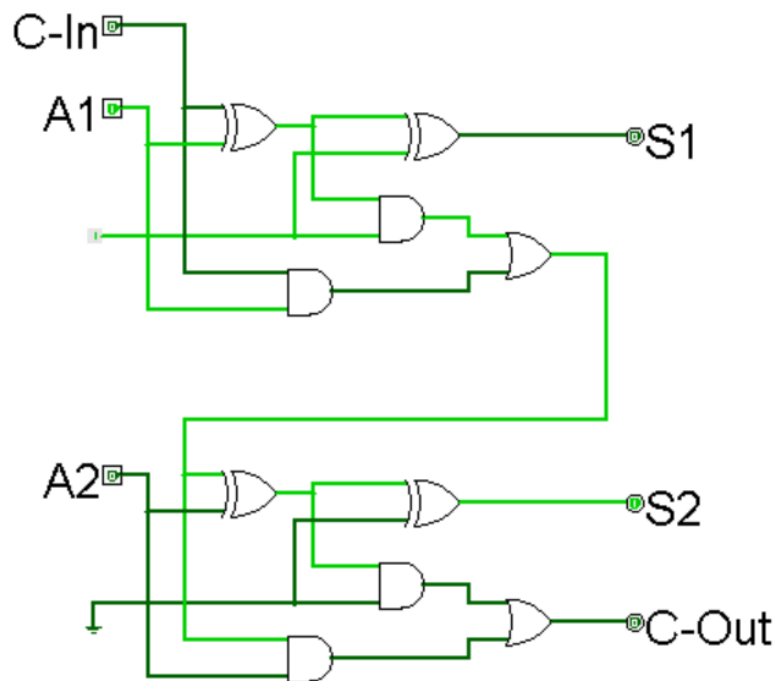


Figure #10: Increment design

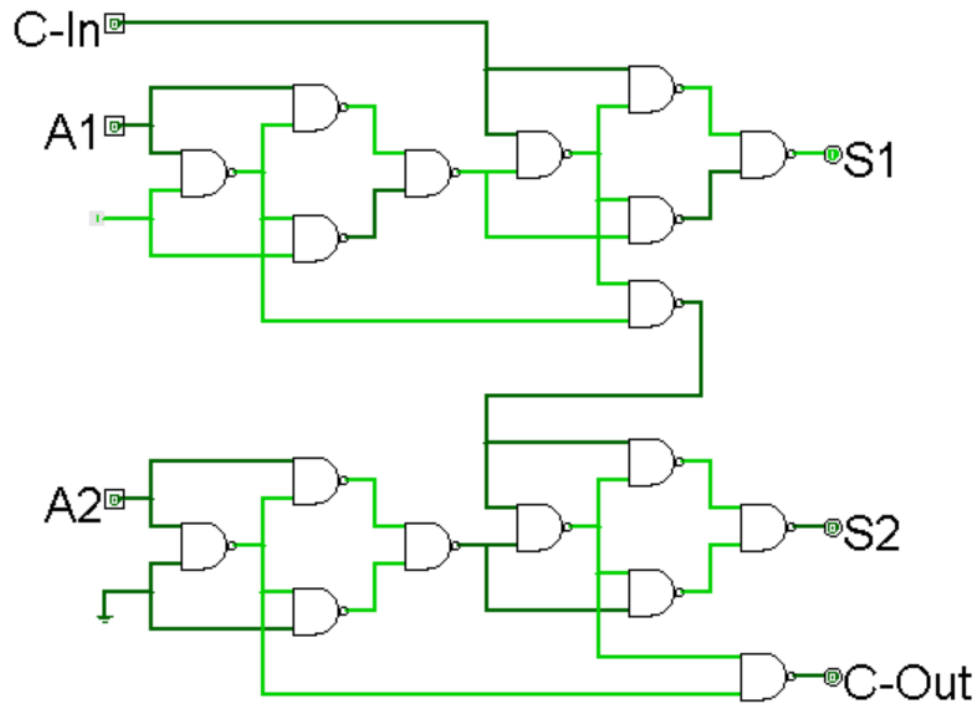


Figure #11: Increment meeting design constraints

Decrement operation

The decrement operation acts like a much more controlled version of the subtraction operation. For the design of this operation we act like we are subtracting a binary one which in twos complement is 11. So we set the B1, B2, and Control inputs to high. The circuit implementation of the decrement operation is shown in figure #12 with the circuit that fits our design constraints pictured in figure #13.

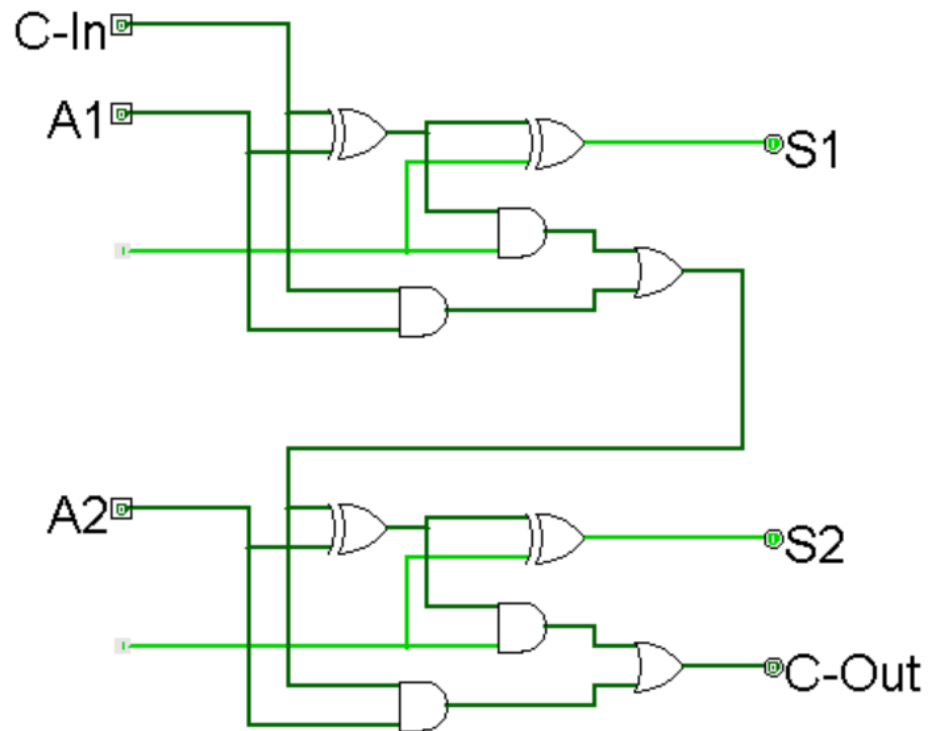


Figure #12: decrement operation circuit

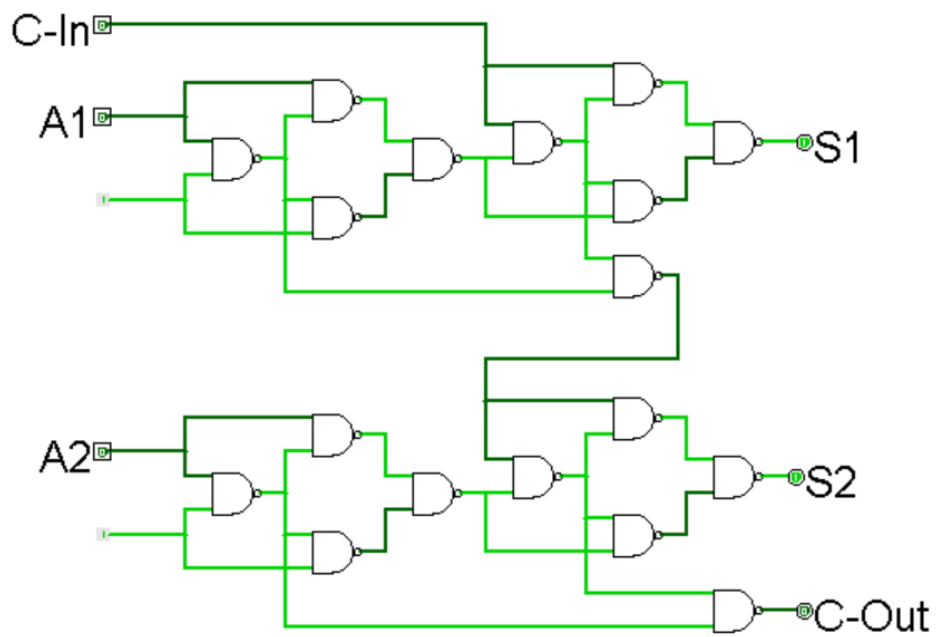


Figure #13: Decrement operation circuit that fits design constraints

Selection between the 4 operations

Selection between the four operations is where things got a little tricky for our team. We needed to use 2 control signals to differentiate between the 4 different operations that could be performed. The simplest way to implement these operations would be to use 3, 1 to 4 multiplexers. After our team implemented this solution we found that this over complicates the amount of logic gates used for the implementation. Our solution uses a simplified form of 1 to 4 multiplexers. We started by implementing three 1 to 4 mux then deleted the redundant logic gates to construct the logic expression of the three muxs. The simplified construction of the 1 to 4 mux that meets our design requirements is shown in figure #14.

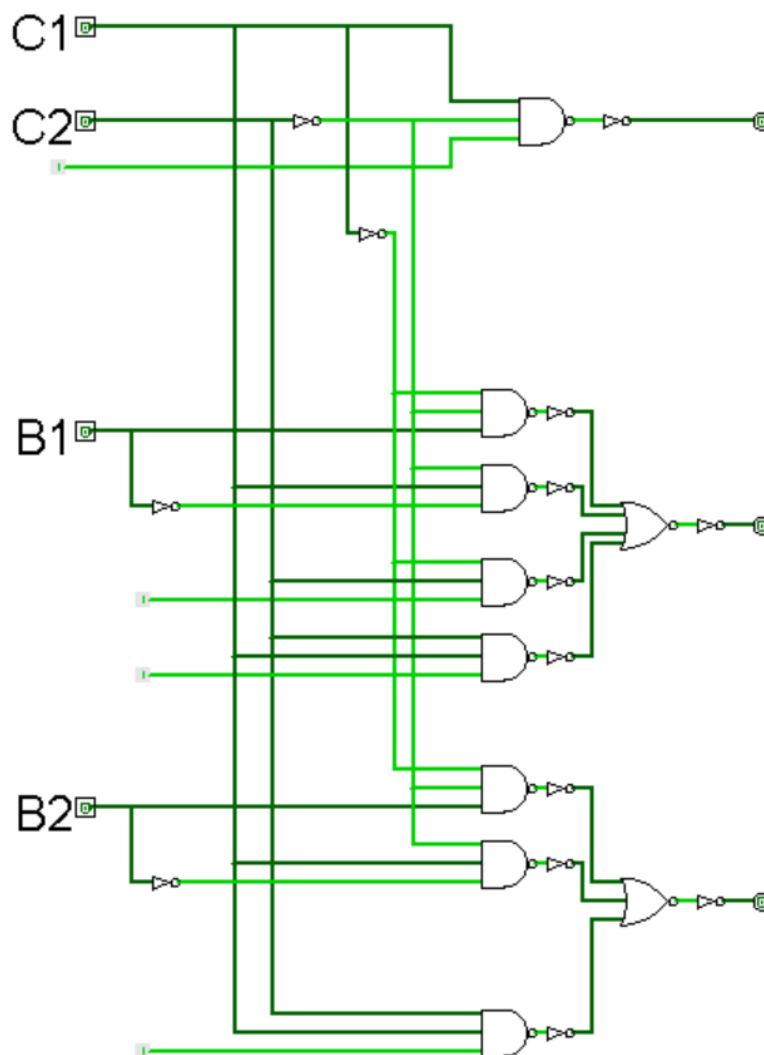


Figure #14: Logic gate expression for the selection and implementation for the 4 operations to be performed on the A and B inputs.

Final Implementation

Once our team had the circuit logic and the ability to differentiate between the different operations that only things left was to put it all together. The final circuit implementation is shown in figure #15 with the final circuit that fits our design constraints pictured in figure #16.

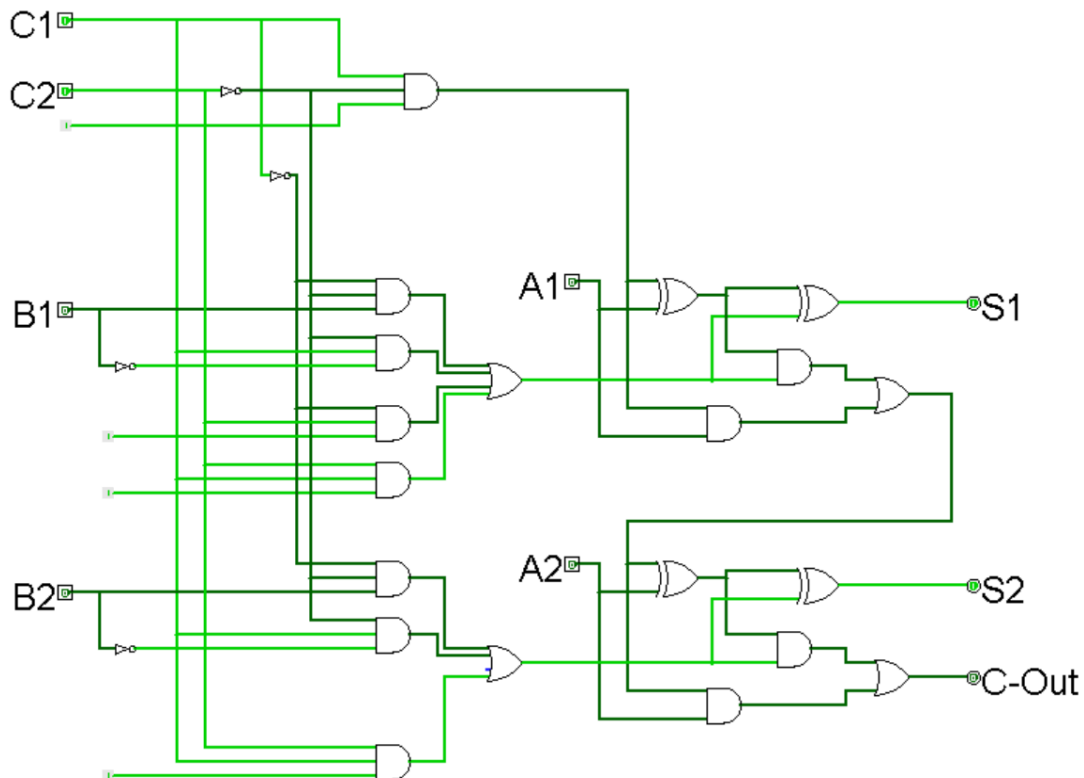
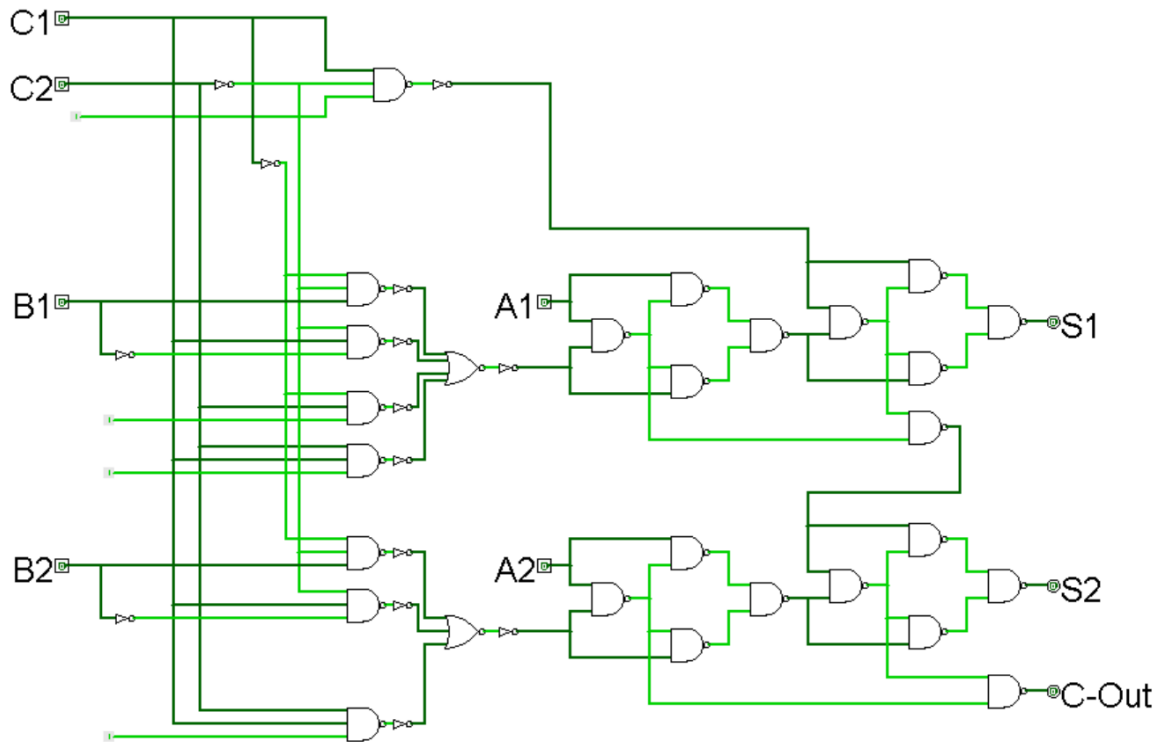


Figure 15: Final circuit implementation



Simulation

Construction in Logisim

With the abstraction of the final circuit complete we built the circuit in a software called Logisim. Figure #16 demonstrates the final implementation of the circuit in the Logisim software. Our team chose to test the circuit in Logisim first before moving on to building the physical circuit.

Testing Simulation

We chose to include one test case for each operation. The Logisim file can be downloaded to test other possible input combinations. Table #1 shows the test cases that we used to test the circuit. Figures #17, #18, #19, and #20 shows screenshots of each of those test cases respectively.

Table #1: Simulation ALU Test Cases

Operation	C_2C_1	A_2A_1	B_2B_1	$C_{out}S_2S_1$
Addition	00	01	10	011
Subtraction	01	01	11	d10
Increment	10	01	dd	011
Decrement	11	11	dd	d01

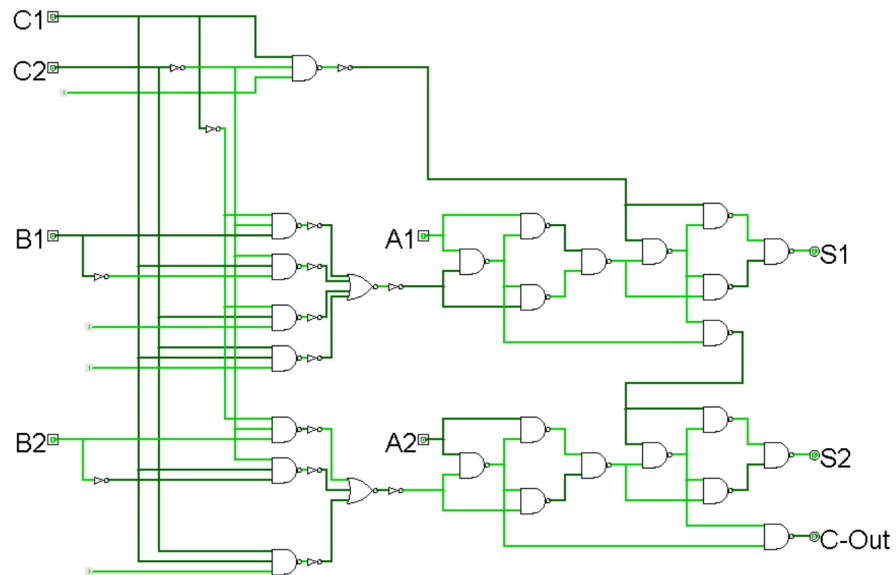


Figure #17: Addition operation performed on the simulation

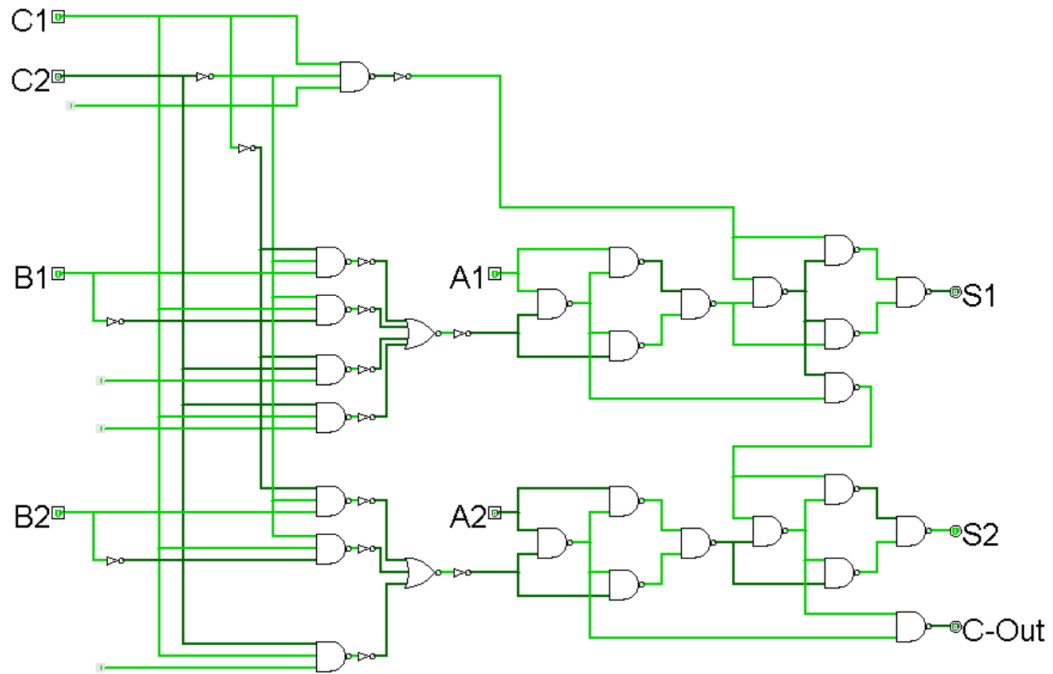


Figure #18: Subtraction operation performed on the simulation

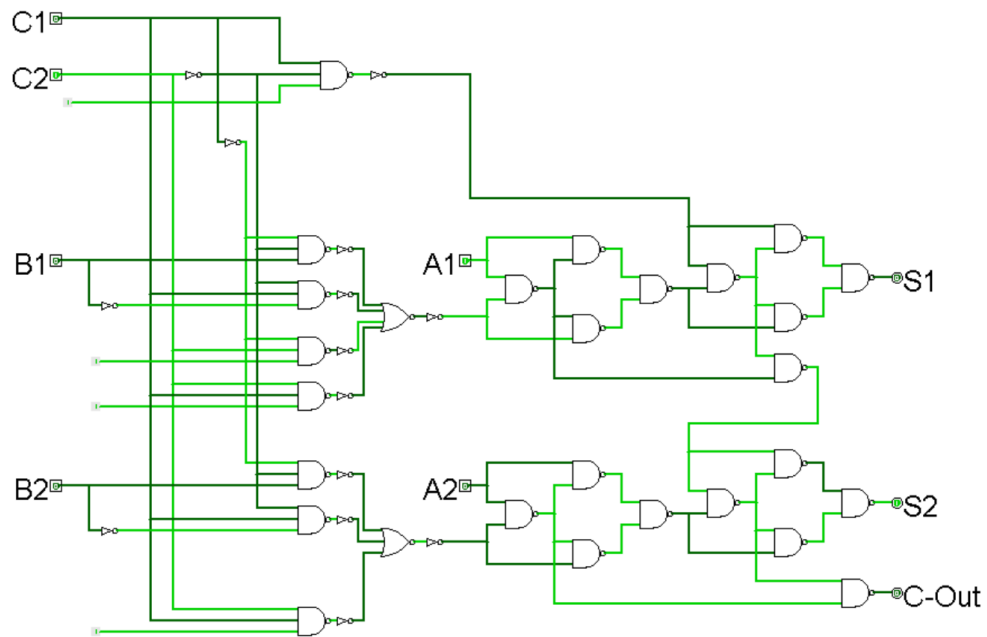


Figure #19: Increment operation performed on the simulation

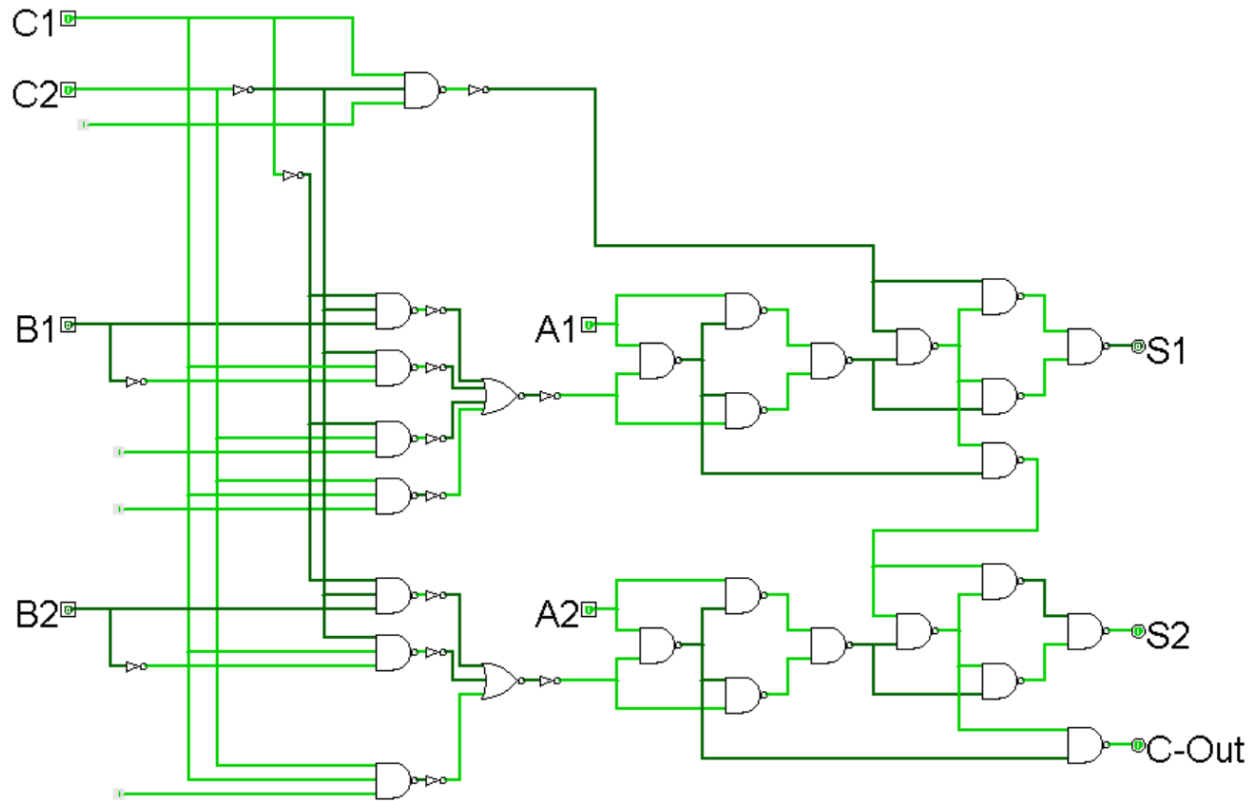


Figure #20: Decrement operation performed on the simulation

Breadboard Circuit

Construction

Our team initially made the two full adders with jumper cables to ensure that the proper connections were made. Once we had the correct functionality we used solid core wire to construct reliable connections between the integrated circuit. Next we cascaded the two full adders, shown in figure #21. Our team initially planned to implement the multiplexers only using NAND, NOR, XOR, and NOT to fit the given design constraints. We ended up using two SN74153S multiplexer ICs due to the limited breadboard space. Using the two SN74153S we were able to implement the entire circuit on just three 830 tie point breadboards. The final circuit implementation on the breadboard is shown below in figure #22.

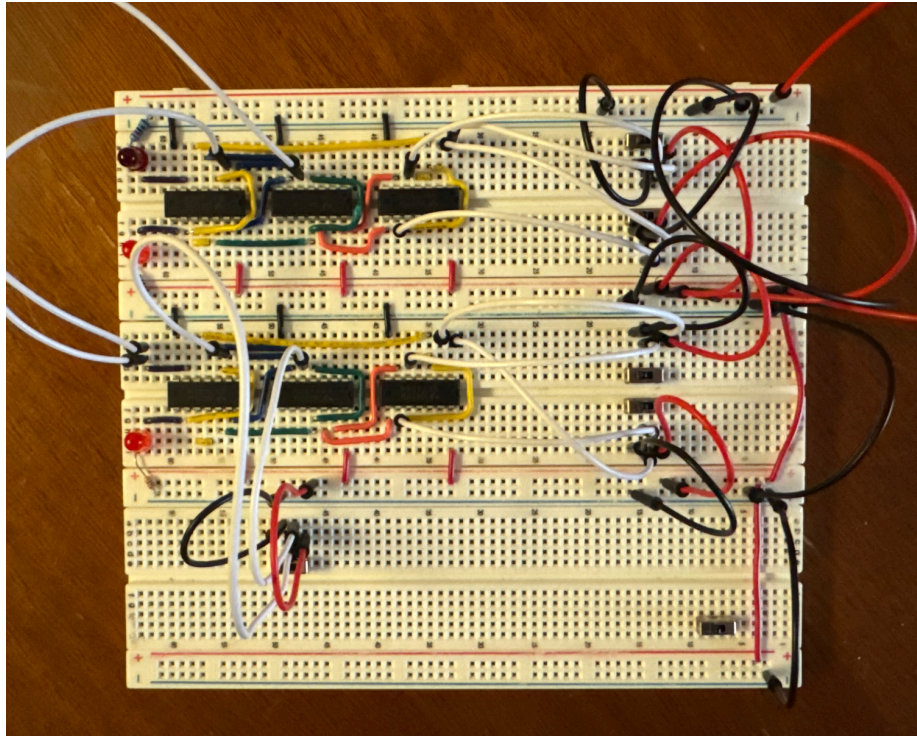


Figure #21: Cascaded full adders on breadboard

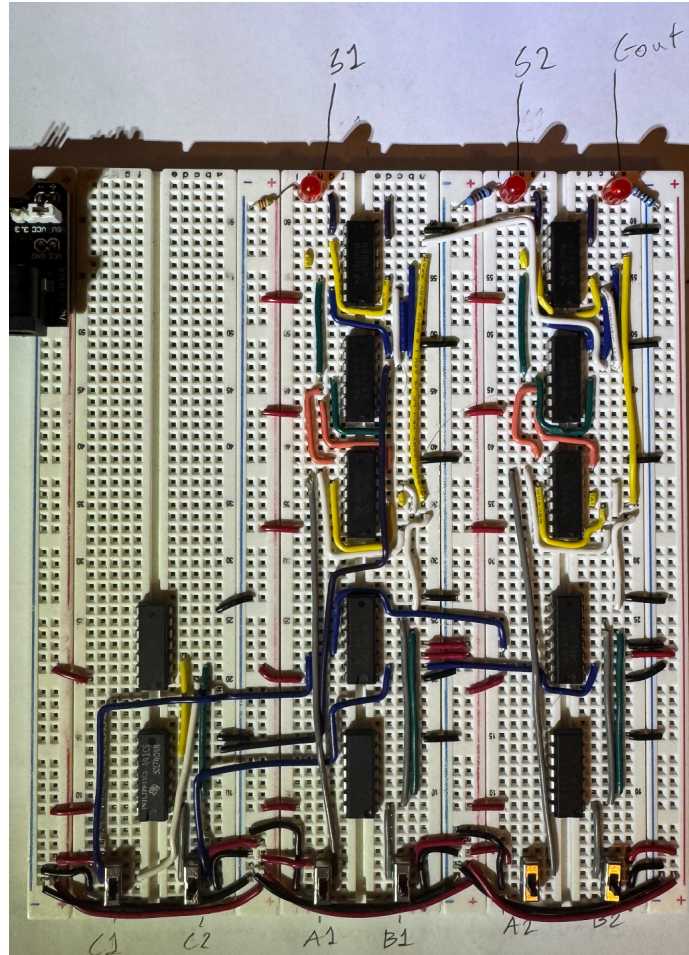


Figure #22: Breadboard implemented Arithmetic Logic Unit

Testing

The Final step of this project was to test the breadboard ALU. We used the switch inputs corresponding to the simulation labels and test case table. Table #2 demonstrates the test cases that were used to test the functionality of the circuit. Figures #21, #22, #23, and #24 demonstrate the inputs and outputs corresponding with the test cases.

Table #2: Breadboard ALU Test Cases

Operation	C_2C_1	A_2A_1	B_2B_1	$C_{out}S_2S_1$
Addition	00	00	10	010

Subtraction	01	10	10	d00
Increment	10	01	dd	010
Decrement	11	10	dd	d01

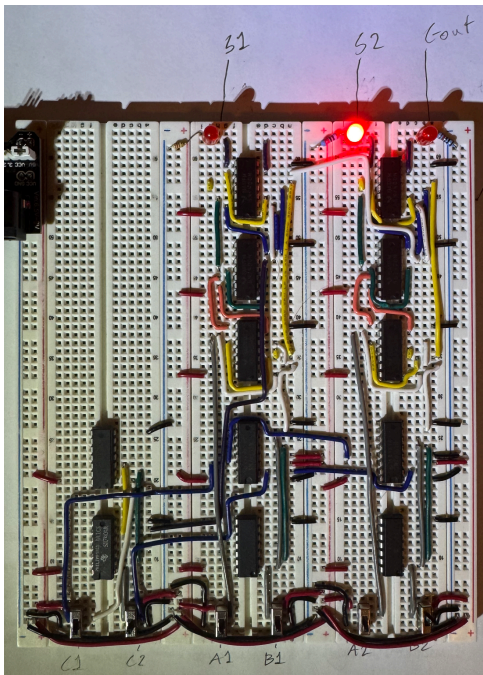


Figure #21: Addition operation performed on the breadboard ALU

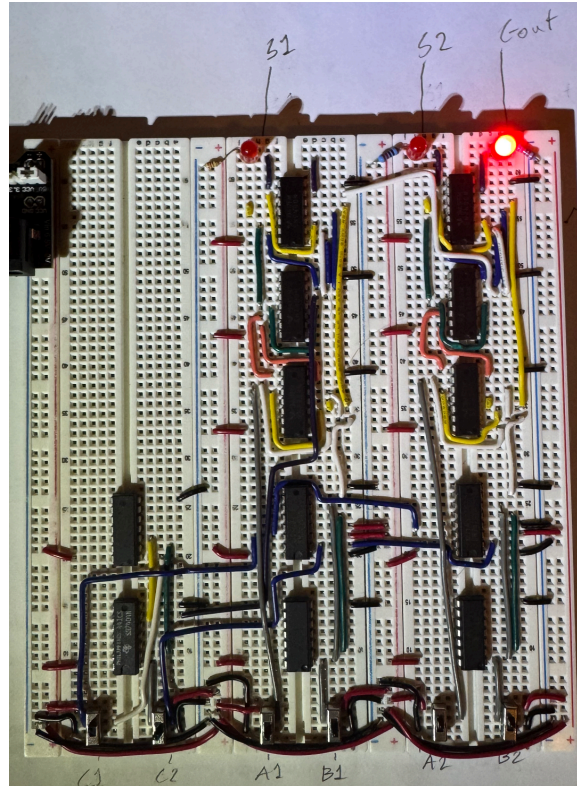


Figure #22: Subtraction operation performed on the breadboard ALU

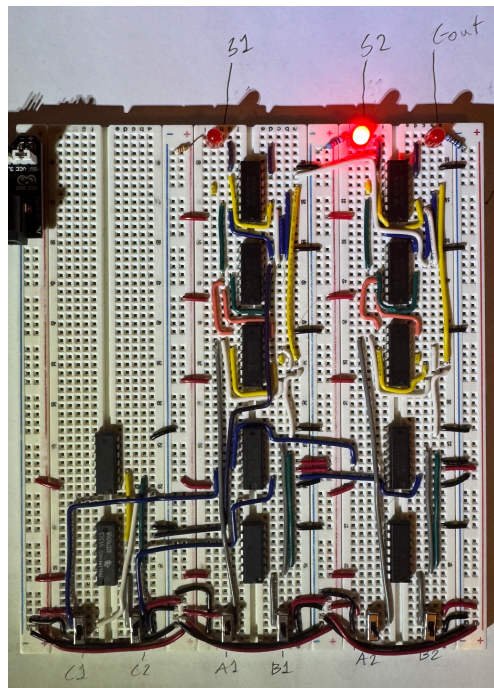


Figure #23: Increment operation performed on the breadboard ALU

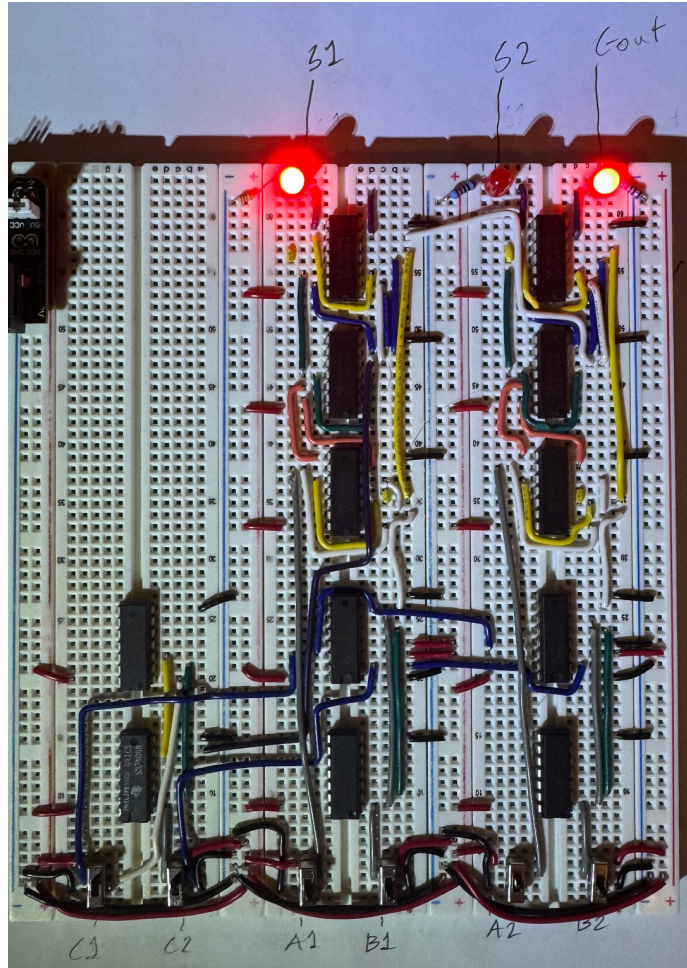


Figure #24: Decrement operation performed on the breadboard ALU

Future work and Conclusion

Through the process of this project our team was able to use a two input, two bit, Arithmetic Logic Unit. Capable of performing 4 operations, addition, subtraction, increment, and decrement. While we set out to only use NAND, NOR, XOR, and NOT gates we ended up also using two SN74153S multiplexer chips. The multiplexer chips were only used on the breadboard implementation. The multiplexer did not affect the operation of the breadboard circuit, just the size of the circuit.

This project was extremely valuable for our team to learn the large scale design and creation of complex processing unit components. While this was an effective design for an arithmetic logic unit we do have considerations to improve the design. Firstly, the design could

be improved by adding more operations to perform on the inputs. While this is most likely the most challenging improvement, our team believes that it would be the most beneficial. Second, you could cascade even more full adders and multiplexers to increase the number of inputs to the unit. This would exponentially increase the computational power of the logic unit. Overall, our team is extremely thankful for our teaching assistant and professor for their guidance throughout this project.