Implementing Computational Logic:

A 4-Bit ALU Design Process

Authored by: **Youssef Mohamed Ahmed Awad** 23P0245

Submitted To:

Prof. **Hossam Hassan**

Dr. **Mohamed Omar**

Eng. **Osama Samy Tayel**

Date: **December 20, 2024**

Acknowledgements

I would like to express my deepest gratitude to everyone who contributed to the successful completion of this project. First and foremost, I extend my heartfelt thanks to my instructor, Dr. Mohamed Omar, for his invaluable guidance throughout this course. His expertise and encouragement have been instrumental in refining the design and methodology of the ALU.

Special thanks to my teaching assistant, Eng. Osama Samy Tayel, for his invaluable support and guidance throughout the project and who provided critical insights and practical advice during the hardware implementation phase.

Finally, I would like to acknowledge the support of Faculty of Engineering, Ain Shams University, for providing the necessary resources and tools, such as simulation software and laboratory facilities, which were integral to the success of this project. This work would not have been possible without the collective efforts and contributions of all those involved.

Table of Contents

I. Understanding the Core of Digital Arithmetic: Introduction to the ALU

From the earliest conceptions of computing systems, the Arithmetic Logic Unit (ALU) has been one of the major components making up the Central Processing Unit (CPU). In von Neumann's architecture, the ALU, then called the Central Arithmetic unit, was built from vacuum tubes and would take up whole rooms [1]. Nowadays these chips are made from billions of tiny transistors and can fit in the palm of your hand. In both cases, these circuits are responsible for all the processing power of the computing system. For modern computer systems these circuits have become immensely complex and are able to carry out billions of operations per second. That's how they can carry out the computations necessary for each screen pixel, each mouse click, each keyboard press and all the other functions and decision making necessary for the running of the computer, all "at the same time" — with almost no noticeable latency.

At its core, any ALU is just a circuit that manipulates 1s and 0s in a way that can be interpreted in a meaningful way. The complexity and speed of these manipulations determines the output that can be produced by the unit. The goal of this project is to design a much simpler ALU to aid in understanding the fundamental concepts behind its design and operation. The ALU to be designed is required to perform three main operations: addition, subtraction and comparison. It takes two 4-bit signed magnitude numbers as input from the user and outputs the result on 7-segment displays.

This report outlines the design process, beginning with the selection of the number representation system and the rationale behind it. It further provides an in-depth discussion on the design and implementation of each individual circuit. It elaborates on the simulation tools which were used to test and debug the circuits during development. The hardware implementation is described, covering the selection of ICs, wiring strategies, and techniques for minimizing signal interference. Finally, the testing methodology is explained in detail, including specific corner cases and strategies for ensuring system reliability.

II. Breaking Down Binary: How Numbers Can Be Represented in the ALU

To get started with the logic behind the circuit, we first need to understand some concepts regarding binary representations and how signed numbers can interact. Starting with the input format: signed magnitude representation.

i. The Trouble with Signed Magnitude: Complexity from Simplicity

In signed magnitude, for adding two numbers having the same sign, addition is as simple as adding up the magnitudes and the output will have the same common sign. In the case of having inputs with different signs, normal addition rules don't apply. To carry out this simple operation will require subtracting the smaller magnitude from the larger magnitude then have the sign of the result be the same as the larger magnitude number. To break this down briefly, this operation will first need us to find the larger magnitude, then flipping both signs if the larger number has a negative sign, then carrying out the subtraction, then returning the sign to the result. This complex process can be avoided if we can find a more arithmetic friendly representation format.

ii. Why BCD Stumbles: The Cost of Mixing Binary and Decimal

Another type of representation is Binary Coded Decimal (also known as BCD). Since the input magnitude is 3 bits, the BCD representation of the magnitude is equivalent to the signed magnitude representation so there would be no need for type conversion. The main advantage of this method would be that it would eliminate the need to convert the result to BCD to display it. But as with signed magnitude, signed operations are complex. And added to that we will need to check each result digit's decimal value and add 6 to it if it exceeds 9.

iii. 2's Complement: The Key to Simplified and Robust Arithmetic

In 2's complement representation we change the weight of the most significant bit into its negative equivalent. Then by adding up the weight of "ON" bits we can get the value of the number being represented. The main advantage and reason for using 2s complement is its straightforward arithmetic manipulation[1]. To add two 2's complement numbers, no matter their sign, we just simply add them, and the result is in 2's complement form. Another advantage of this representation is that it generalizes comparison of numbers which will be made clear in the comparator circuit. An added benefit is that unlike signed magnitude, it has only one pattern reserved for zero. This allows for a slightly larger range being represented using the same number of bits. On the other hand, there are two drawbacks to using this representation. The first limitation is that the input uses signed magnitude format, necessitating an additional conversion sub-circuit. The second drawback is that 2's complement form can't be directly displayed on 7 segment displays.

[1] In von Neumann's report on the EDVAC, his design used 2's complement arithmetic to simplify subtraction.

Decimal	Signed magnitude	BCD	2's complement	Decimal	Signed magnitude	BCD	2's complement
+0	0000	+000	0000	-0	1000	-000	—
+1	0001	+001	0001	-1	1001	-001	1111
+2	0010	+010	0010	-2	1010	-010	1110
+3	0011	+011	0011	-3	1011	-011	1101
+4	0100	+100	0100	-4	1100	-100	1100
+5	0101	+101	0101	-5	1101	-101	1011
+6	0110	+110	0110	-6	1110	-110	1010
+7	0111	+111	0111	-7	1111	-111	1001

Table 1: Input range in the three representation systems

III. From One Form to Another: The Key to Digital Arithmetic

i. From Binary to 2's Complement: Understanding the Conversion Mechanism

Starting with the obvious starting point, how to convert into 2's complement format. The 2's complement of a binary number N with n bits is derived using the following formula:

$$N' = 2^n - N$$

This is simple enough to do by hand. But for a circuit carrying this out would include an exponent which is a very tedious and inefficient circuit to implement. Instead, another definition of 2's complement can be utilized:

$$2's\ complement = 1's\ complement + 1$$

$$2^n - N = ((2^n - 1) - N) + 1$$

Seeing this it may seem that the operation is still complicated since it still requires finding the 1's complement. But actually, this operation in binary is as simple as toggling each bit. So, the complex operation of the exponent has been simplified into an inverter then adding 1. The actual circuit implementation will be discussed later.

ii. Mathematical Intuition Behind Negative Numbers in 2's Complement

It may still be unclear how finding the 2's complement makes signed operations easier. For any decimal magnitude, it has separate representations for its positive and negative values. The positive representation is the same as the unsigned binary representation. The negative value is the 2's complement of the positive representation. Using this we can represent any decimal integer. But how does representing negative numbers in this way make arithmetic easier? Using the mathematical definition of 2's complement:

$$x' = 2^n - x$$

By adding a bracket:

$$x' = 2^n + (-x)$$

In this form it is easy to understand the intuition behind it. By adding x' to any number we are effectively adding $-x$ which can be framed as just subtracting x. This also adds 2^n which is handled in the overflow condition which is discussed next.

iii. The Elegance of 2's Complement Addition

The last main concept we need before starting to design the circuit is how addition works in 2's complement. As stated before it is as simple as adding both numbers. With one caveat which may go unnoticed: the possibility of an overflow occurring. In normal binary addition we "naturally" add a bit when overflow occurs as it always represents a larger number being required. But due to the nature of how negative numbers are represented this may not always be the case. In 2's complement addition, overflow only occurs when the last two carry bits are different. Otherwise, the result has the same number of bits as the input and any extra digits are discarded.

IV. Design Considerations

i. Determining the Bit Requirements: Establishing the Range for Signed Arithmetic

We can now analyze our input range to find out our possible output range. Using signed magnitude 4 bit input we get an input range of -7 to +7 for each number. This gives us an output range of -14 to +14. In 2's complement form, this range would require an extra bit to be represented: the sign bit. This will be represented in the carry bit resulting from the addition of the most significant bits.

ii. Checking for Overflow: Carry Bit Analysis

In by-hand addition, we can check for the significance of the last carry bit by checking if the last two carries are different. This check tells us whether the 5^{th} bit (last carry) or the 4^{th} bit are the most significant. Using this fact, we can do another check to test for the sign of the number. The most significant bit signifies the sign.

iii. Subtraction: Shifting Operations

By using the fact that the two numbers are in 2's complement form, we can modify the inputs to perform subtraction. Since subtraction is just adding the negative magnitude of the number to be subtracted, we find the 2's complement of the subtrahend and then carry out the addition.

iv. Universal Comparison

Following the same approach used for the other operations, we use 2's complement form. This makes the logic easier. Comparing two positive numbers in any representation system is almost equivalent. By comparing the most significant bits in order we can find the larger number. Comparing opposite sign numbers is even easier since the negative number is immediately known to be smaller. The deviation in comparison is when dealing with negative numbers. In most representations the comparison is done normally then the outputs are flipped since -7 would be smaller than -4 which wouldn't be considered when comparing the magnitudes. But in 2's complement representation, this fact, that larger magnitude negative numbers are less than smaller magnitude negative numbers, is taken into account. The 2's complement representation of -7 and -4 are (1001) and (1100) respectively. If we compare these two numbers as if they are unsigned numbers, we will get the correct result. This fact generalizes the logic and eliminates the need for exception handling for specific cases.

v. Mapping to Display

To display the magnitude from the addition, we need to find the Binary Coded Decimal value for each digit to be displayed. BCD values range from 0 to 9. So, any number outside this range (greater than 9) requires a dedicated converter circuit. Each BCD bit will be considered as a separate function, and a truth table will be constructed to find its representative Boolean expression. Two displays will be used to display the magnitude, and a third display will be used to display the sign. Then inputting these BCD signals into off-the-shelf ICs will convert them into signals that can correctly be represented on the displays.

V. Building the Foundations: The Core Units of the ALU

Before explaining the main logic behind the circuit, some basic circuit units will be defined to make further, more complicated circuits easier to understand.

i. Half Adder

This circuit is used when we want to add only 2 bits together. We will first build the truth table and derive the expression and then the circuit from it.

X	Y	Sum	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 2: Truth table for half-adder

It is very simple in this case to notice the pattern for both the sum function Sum and the carry function C_{out}.

$$Sum = X \oplus Y$$

$$C_{out} = X \cdot Y$$

These two expressions when implemented as logic gates, are known as a half-adder.

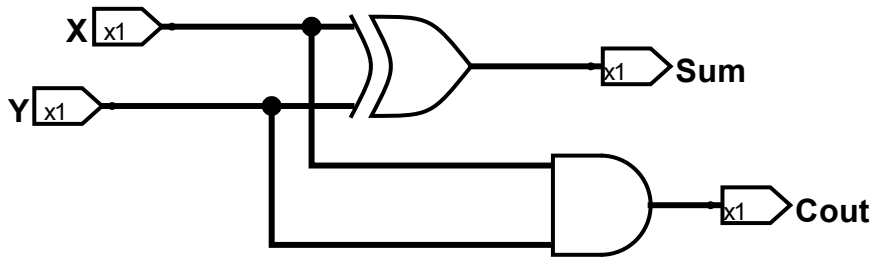


Figure 1: Half-adder circuit

ii. Full Adder

Taking it up a level, to add 3 separate bits, a half adder is not sufficient. But following the same method we can find an expression for a circuit that adds 3 bits:

X	Y	C_{in}	S	C_{out}
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

Table 3: Truth table for full-adder

For these 3 functions the expression cannot be as easily derived in its simplest form. Using a systematic approach, we find the canonical form for each function then using the Boolean algebra postulates we can simplify them.

$$S = X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in}$$

$$= X'(Y'C_{in} + YC_{in}') + X(Y'C_{in}' + YC_{in})$$

Both brackets can now be noticed to be XOR and XNOR respectively.

$$S = X'(Y \oplus C_{in}) + X(Y \oplus C_{in})'$$

By taking $(Y \oplus C_{in})$ as one term, we can compact the expression further with another XOR function:

$$S = X \oplus Y \oplus C_{in}$$

This is the simplest, most compact form of the sum function.

The same process will be repeated for the carry function.

$$C_{out} = X'YC_{in} + XY'C_{in} + XYC_{in}{}' + XYC_{in}$$

$$= C_{in}(X'Y + XY') + XY(C'_{in} + C_{in})$$

$$C_{out} = C_{in}(X \oplus Y) + XY$$

By carefully choosing the order of gates for the function Sum , we can reuse the same XOR gate output in C_{out}

The implementation of both these functions together is called a full adder.

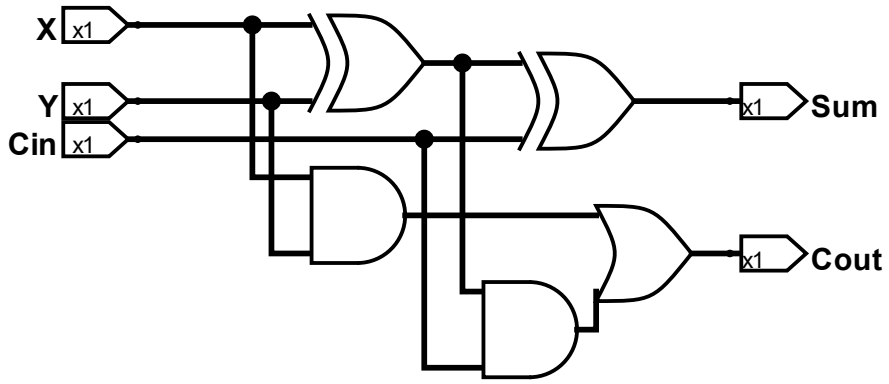


Figure 2: Full-adder circuit

iii. Comparator

For any comparison operation, there are three possibilities: the first number is greater, the first number is smaller or both numbers are equal. We will treat each case as a separate function.

A	B	A>B	A<B	A=B
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Table 4: Truth Table for comparator

$$(A > B) = A \cdot B'$$

$$(A < B) = A' \cdot B$$

$$(A = B) = (A \oplus B)'$$

Each of these functions outputs a signal signifying the validity of the relation they represent between the two bits.

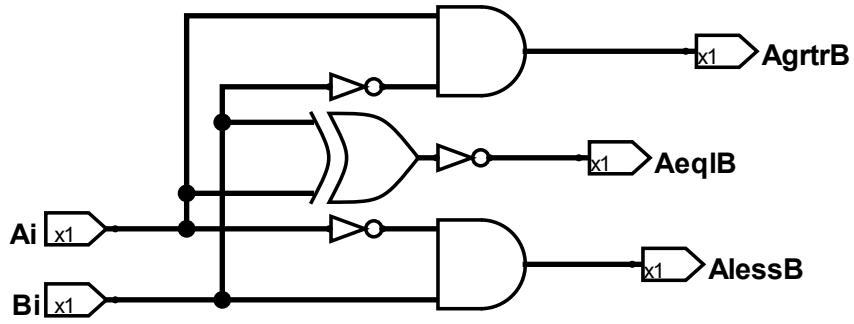


Figure 3: Comparator circuit

VI. Core Logic: Managing Input, Processing, and Displaying Results

The circuit logic can be summarized in the shown flowchart. The user inputs both numbers as signed magnitude along with the required operation (10 bits in total). The circuit then converts the input numbers into 2's complement representation. These converted signals are then routed to two parallel circuits: the adder/subtractor circuit and the comparator circuit. Due to the nature of dealing with electrical connections as logic signals, both circuits are active simultaneously to avoid unnecessary wiring. For the signal sent to the adder/subtractor circuit, depending on the operation, B may be complemented. Both numbers are then added, and the result is converted back to signed magnitude. This magnitude of this result is then converted to BCD format. At this point the signal is ready to be displayed on the 7 segment displays. Simultaneously, the comparator circuit compares A and B and outputs a signal depending on the relation between them. This signal can be visually represented after some manipulation. From the signals of the two circuits, we disable one or the other depending on the selected operation and display the other.

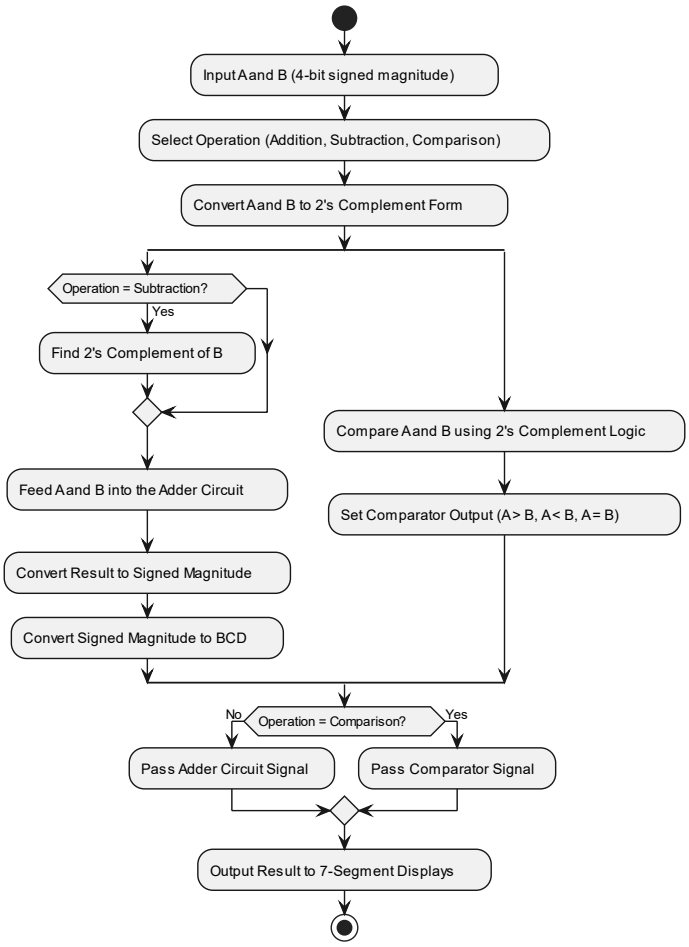


Figure 4: ALU flowchart (Created using PlantUML)

i. User Input Handling

For the input of the 10 bits, two 4x dip switches are used; one for each number and a 2x dip switch for operation selection. For the physical implementation, each input bit must be pulled down so that it always has a defined state. This is achieved by connecting the "negative" side of each individual switch in series with a 10kΩ resistor and then to the ground. This resistance is specifically used to make sure any noise is sufficiently pulled down while not drawing too much current.

For the rest of this document the bits of each number and/or function result X will be referred to as:

$$X = X_4X_3X_2X_1$$

With each bit's index indicating its significance (higher index means higher significance).

ii. Converting to 2's Complement

Following the usual methodology, a truth table can be constructed for the 4 bits and using k-maps, a simplified function can be found for each output bit. This would require an 8x32 truth table which produces very complex expressions to implement. Instead, we can make use of the mathematical definition of 2's complement that was discussed earlier. To convert the input from signed magnitude to 2's complement representation, we first check the sign (the most significant bit). For positive numbers, both representations are equivalent, so no change occurs. For negative numbers, each bit is toggled and then we add one to the new inverted binary value.

To conditionally flip each bit depending on the sign, we can construct a truth table and derive the expression.

X_4	X_i	$X_{inverted}$
0	0	0
0	1	1
1	0	1
1	1	0

Table 5: Truth table for inverting X_i

We notice that we can do the inversion by XORing the sign bit with each of the other 3 bits.

$$X_{inverted} = X_4 \oplus X_i \ (i = 1,2,3)$$

To complete the 2's complement we add 1 which will make use of the half-adder circuit shown before. Using three cascaded half adders we can add 1 to the inverted binary value. The AND gate used for carry out of the last bit can be removed since it won't be needed as the most significant bit, as a general rule, is unchanged in both representations. One exception in this conversion is the case of negative zero which was briefly mentioned earlier. In signed magnitude its possible to input a binary value for negative zero which has no corresponding value in 2's complement. This input pattern needs to be handled separately. Logically, negative zero is equivalent to zero since zero as a number has no sign. All three magnitude bits are not affected by this exception, only the most significant bit. Therefore, a k-map is constructed for the function of this bit to find its expression.

X_1X_2 \ X_3X_4	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	1	1	1

Table 6: Truth table for sign bit conversion

$$X_{4_{(2's)}} = X_4X_1 + X_4X_2 + X_4X_3$$

$$X_{4_{(2's)}} = X_4(X_1 + X_2 + X_3)$$

This expression simply makes the value of the most significant bit 1 only when the sign bit is 1 and **any** of the other bits are 1.

This circuit is the implementation of the signed magnitude-to-2's complement converter. This circuit is repeated for both A and B. The input of the circuit is connected to the corresponding switches and the outputs are used as the inputs for any other logical operations in the circuit.

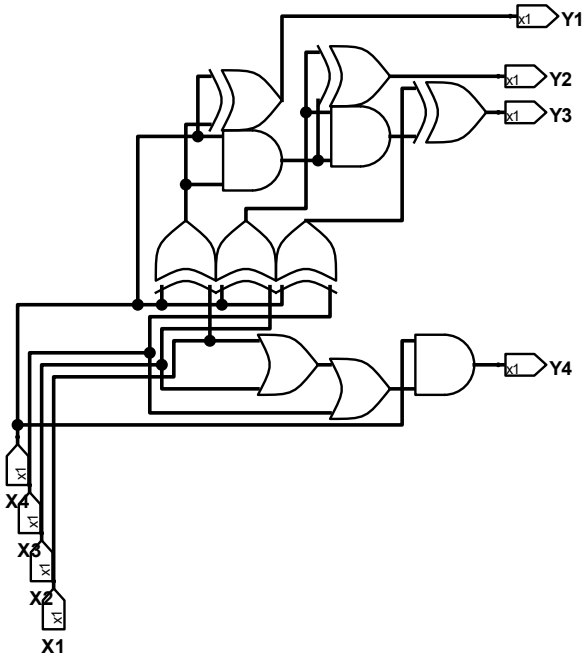


Figure 5: Signed magnitude-to-2's complement converter circuit

iii. Operation Configuration and Control

In this ALU we are implementing 3 different operations. This requires 2 bits for selection. These 2 bits produce 4 patterns which means there is one extra unused pattern. This case needs to be handled.

O_2	O_1	Operation	Expression	Signal
0	0	Addition	-	-
0	1	Comparator	$O_2'O_1$	Comp
1	0	Subtraction	O_2O_1'	Subt
1	1	Don't care	-	-

Table 7: Truth table for operation selection

$$Comp = O_2'O_1$$

$$Subt = O_2O_1'$$

The subtraction operation signal will be referred to as Subt and the comparator signal referred to as Comp. Addition is treated as the default operation with a requirement of each operation's bit to be HIGH ***on its own*** to switch to different operations.

As stated before, the adder and comparator circuits are active simultaneously but for the sake of clarity each will be covered individually

iv. Adder/Subtractor Design

Unlike adding 1, adding multiple consecutive bit pairs, produces a carry out that needs to be considered in the next addition. This leads us to the fact that for consecutive bit addition we need a circuit that adds 3 bits. We can now make use of the full adder unit discussed before. By connecting multiple full adders back-to-back, we can add any two binary numbers of any bit-width. For this application we need to add four bit pairs so we will use four full adders. This adder circuit outputs five bits:

$$F = C_4F_4F_3F_2F_1$$

$$\begin{array}{ccccc} C_4 & C_3 & C_2 & C_1 & \\ & A_4 & A_3 & A_2 & A_1 \\ + & B_4 & B_3 & B_2 & B_1 \\ \hline C_4 & F_4 & F_3 & F_2 & F_1 \end{array}$$

Figure 6: Addition of two 4-bit numbers

For subtraction a separate circuit could be constructed but this would be incredibly inefficient. By analyzing the operation of subtracting, we can make use of the existing circuit to implement both operations. In its most fundamental form, subtraction is the operation of adding a negative number. Using this reasoning, we can modify the adder circuit to perform subtraction. Or more accurately we will modify the input to the circuit to take the negative sign into account. As explained, any number's negative equivalent is just its 2's complement. In short, we need a circuit to complement B in the case of subtraction. Using almost the exact logic behind the converter we can find the 2's complement. We toggle each bit using XOR gates. And instead of using half adders to add 1, we can make use of the unused carry-in input of the first full adder. The last remaining piece to complete the adder/subtractor is to make the logic toggleable. Simply connect the subtraction signal with each XOR gate and the carry in. This way, the circuit will work as a subtractor only when $Subt$ is 1 [Appendix v].

v. Converting Results for Display

A feature of 2's complement addition is that its result is also in 2's complement form. For displaying the results, negative numbers would be misinterpreted in this format. To remedy this, the result will be converted back to signed magnitude. This will be identical to the input converter and will only change the result if it's negative. Which takes us to the next component of the circuit: how do we check if a number is negative? The word definition is that a 2's complement number is negative if its most significant bit is 1. The only trick here is that the most significant bit can change depending on the overflow condition.

Overflow check	4^{th} bit (F_4)	5^{th} bit (C_4)	Sign bit
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 8: Truth table for sign determination

Essentially, it's a signal selector between the 4^{th} and 5^{th} bits using the overflow check as the control signal.

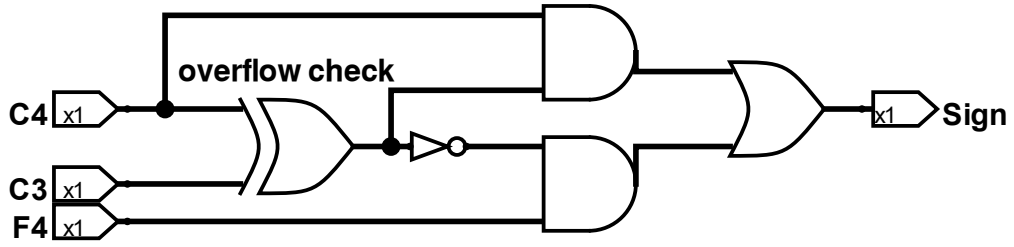


Figure 7: Sign determination circuit

Using the newfound sign bit, we can complement the number in the case of 1. This gives the positive form of the number ($H_5H_4H_3H_2H_1$) with an additional separate sign bit.

The last step to be able to display the result on a 7-segment display is to convert this number into BCD (since different displays are not internally connected and each one can only handle values from 0-9). Following the methodology, we treat each BCD as a separate function and create a truth table for each one.

H_5	H_4	H_3	H_2	H_1	J_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	1	0	0	1	0
0	1	1	0	1	1	0	0	1	1
0	1	1	1	0	1	0	1	0	0

Table 9: Truth table for conversion from 5 bit binary to BCD (with don't care patterns ignored)

We can first observe that H_5 doesn't affect the output. It's included in the circuit to account for the case of using the full range of 2's complement. The second observation is that $J_4J_3J_2$ are not included. This is due to the fact that the tens digit will never exceed 1 so the value of these bits would always be zero.

We can create a k-map for each bit $J_1 G_4 G_3 G_2 G_1$.

G1 / H2, H3	000	001	011	010	110	111	101	100
00	0	0	0	0	1	1	1	1
01	0	0	0	0	1	1	1	1
11	0	0	0	0	1	1	1	1
10	0	0	0	0	1	1	1	1

Figure 8: K-map for G_1

$$G_1 = H_1$$

G2 / H2, H3	000	001	011	010	110	111	101	100
00	0	1	1	0	0	1	1	0
01	0	1	1	1	1	1	1	0
11	1	1	1	0	0	1	1	1
10	1	1	1	0	0	1	1	1

Figure 9: K-map for G_2

$$G_2 = H_2' H_3 H_4 + H_2 H_4' + H_5$$

G3 / H2, H3	000	001	011	010	110	111	101	100
00	0	1	1	0	0	1	1	0
01	1	1	1	0	0	1	1	1
11	1	1	1	1	1	1	1	1
10	0	1	1	0	0	1	1	0

Figure 11: K-map for G_3

$$G_3 = H_2 H_3 + H_3 H_4' + H_5$$

G4 / H2, H3	000	001	011	010	110	111	101	100
00	0	0	1	1	1	1	0	0
01	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0

Figure 10: K-map for G_4

$$G_4 = H_2' H_3' H_4$$

J1 / H2, H3	000	001	011	010	110	111	101	100
00	0	1	1	0	0	1	1	0
01	0	1	1	1	1	1	1	0
11	0	1	1	1	1	1	1	0
10	0	1	1	1	1	1	1	0

Figure 12: K-map for J_1

$$J_1 = H_2 H_4 + H_3 H_4 + H_5$$

$$J_2 = J_3 = J_4 = 0$$

Now we can output these signals into a BCD-to-7-segment decoder. The pins for the 3 most significant bits in the tens digit decoder IC are pulled down to avoid floating values since their logic state will affect the output [Appendix vii].

vi. Comparator Design

Back to the comparator circuit. The circuit inputs are the converted 2's complement numbers. As explained, this means that all numbers are compared in the same way regardless of their sign. We will use the comparator module multiple times cascaded to construct a 4-bit comparator. The only exception is that for the most significant bit the values are reversed. Meaning that for the first bit to be compared, we consider 0 to be the larger number. This stems from the fact that this bit's weight is negative so if it's HIGH the value would be negative, which is less than zero.

The way the comparator modules are connected is as such: each module compares its 2 respective bits and outputs a result for that bit position. For single bit comparison that would be sufficient. But in this case, we need to take bit significance into account. Meaning that having a bit of higher significance be smaller for example over-rides another lower significance bit being greater. In logic circuits, this can be translated as a condition:

$$For any\ bit's\ comparison\ output\ signal\ to\ have\ importance,$$

$$all\ more\ significant\ bits\ must\ be\ equal.$$

This statement can then be converted into gates. For each result bit we AND it with the result of the previous bits' equality signal. The outputs of all these ANDS are ORed to get a final signal for each relation [Appendix vi].

The fact that this circuit has only three possibilities can be used to simplify the gates. By logical reasoning, we can declare that for any two bits, if the relation is one of the three, then it can't be either of the remaining two. And the reverse is true. If two of the relations are false, then the last one must be true. This can be used to derive any one of the three relations from the other two. For educational purposes, the implementation of this fact was not carried out in the optimal way.

vii. Output Signal Selection and Routing

Now we have two input signals: one from the adder/subtractor and the other from the comparator. We need to select which one to display. This selection will depend on the Comp signal. If it's on, the comparator signal will be passed and vice versa. This part is just a fundamental gate implementation of a multiplexer. We AND each signal to be selected to its ON condition. This gives us two signals for each LED segment, one from each circuit. These signals are ORed and connected to the LED (some outputs can be simplified using the absorption theorem).

For the adder circuit the magnitude is displayed on the two rightmost displays while the sign bit is displayed on the third display by connecting the signal directly to the G segment.

For the comparator, a HIGH signal is passed to the leftmost and rightmost display segments to show the letters A and B respectively by connecting directly to the segments [2]. And the relation is shown on the center display.

Since common cathode displays are used, HIGH signals are 1 [3].

VII. Testing the Limits: A Comprehensive Evaluation of ALU Performance

Each module was tested on its own to guarantee functionality then bottom-up testing was carried out as each module was integrated.

The complete ALU circuit has 10 inputs. Testing each input combination would mean having 1024 test cases to check through. Instead, some specific corner cases were tested. This utilizes the knowledge of knowing the system's dependencies to design specific test cases. Then, the fully integrated circuit was tested using black box random testing to avoid bias and ensure system reliability.

Some example corner cases would include all four possible input combinations of signed zeroes, the maximum possible output values and cases that would produce conflicting interpretations due to the representation conversion [Appendix viii].

VIII. From Virtual Simulations to Physical Implementation: Tools Overview

i. Virtual Simulation

Each module was initially designed and then implemented on Logisim for logic testing. The modules were then integrated one by one and tested thoroughly before further additional functionality was added. After the debugging process was completed, the circuit was implemented on TinkerCAD. The main advantage provided by TinkerCAD was simulating close-to-life behavior for the circuit allowing testing of other elements of the circuit such as power sources and resistance values. It also gives a baseline for the required components needed for hardware implementation.

ii. Physical Circuit Implementation

The circuit was implemented physically using breadboards and jumper wires for their ease of use and modifiability. The 74 series ICs were used for this implementation. All the circuit was implemented using fundamental gate ICs with the exception of the 7-segment driver. Using the simulations as a reference, the logic circuit was implemented while considering physical factors such as limitation of physical space and the internal resistance of the various components. Outputs were tested using a multimeter to make sure they provided an in-range value for a logical HIGH according to the datasheets.

Part Number	Description
7404 [6]	Hex inverter
7408 [7]	2-input AND
7432 [8]	2-input OR
7448 [10]	Common cathode 7-segment driver
7486 [9]	2-input XOR

Table 10: ICs used

To power the circuit a 9V battery is used. To step down the DC voltage to the recommended 5V, a voltage regulator (7805) was used. This brings down the voltage within the acceptable range during steady state. For transient states (for example during connecting the supply to the board), due to the sudden change in current demand, this will produce a voltage excursion which may exceed the recommended level for the ICs causing

electrical breakdown of the internal transistors making the component unusable [4]. This is avoided by using capacitors to "absorb" this voltage spike. By referencing the regulator's datasheet [5], the recommended regulator circuit is connected as in figure 13. The components are soldered on a printed circuit board for easy disconnection from the main logic circuit [Appendix iv].

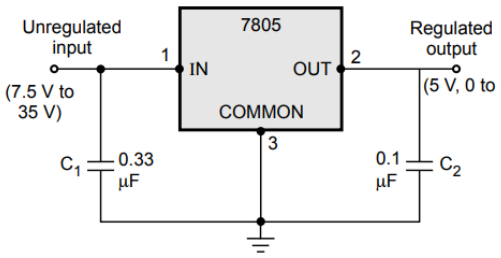


Figure 13: Fixed-output regulator

The most significant challenge faced during hardware implementation was wire complexity and keeping track of all the needed signals. The main strategy used to fix this was to keep a rough color code for certain important signals. This way during debugging it would be easier to track error-causing signals. Another helpful tactic was to label each module's wire outputs so if needed later they can be found easily.

Color	Signal	Color	Signal
Purple	X_1 (Least significant bit)	Brown	A>B
Yellow	X_2	Black	A<B
Red	X_3	Green	Control signals
Blue	X_4	White	Power
Orange	Carry	Gray	Power

Table 11: Wiring color code

IX. Reaching the Finish Line: Final Insights into the ALU

The design and implementation of the ALU provided an opportunity for comprehensive understanding of the logic driving modern systems. It offered insight into the design process and its intricacies, bridging the gap between theoretical concepts and their actual applications in real life.

Ultimately, the project not only achieved its primary objectives but also demonstrated the importance of the systematic design approach, testing, and optimization in digital circuit design. This ALU serves as a foundational example of how computational logic can be translated into hardware, offering insights into the operation of more advanced processors.

Future improvements could include expanding the ALU's functionality to handle additional operations such as multiplication, and/or scaling the design to work with larger inputs. These enhancements would further solidify the connection between theoretical learning and practical engineering.

X. References

[1] J. v. Neumann, "First Draft of a Report on the EDVAC," 1945. [Online]. Available: https://web.archive.org/web/20070623082626/http://qss.stanford.edu/%7Egodfrey/vonNeumann/vnedvac.pdf.

[2] "7 Segment Display Pinout," [Online]. Available: https://www.electronicsforu.com/resources/7-segment-display-pinout-understanding.

[3] Avago, "Common Cathode 7-Segment Datasheet," [Online]. Available: https://www.farnell.com/datasheets/2095789.pdf.

[4] C. Simpson, "Engineers note: Capacitors are key to voltage regulator design," [Online]. Available: https://www.ti.com/lit/wp/snoa842/snoa842.pdf.

[5] Texas Instruments, "7805 Datasheet," [Online]. Available: https://www.sparkfun.com/datasheets/Components/LM7805.pdf.

[6] Texas Instruments, "7404 Datasheet," [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74ls04.pdf.

[7] Texas Instruments, "7408 Datasheet," [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74ls08.pdf.

[8] Texas Instruments, "7432 Datasheet," [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74ls32.pdf.

[9] Texas Instruments, "7486 Datasheet," [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74ls86a.pdf.

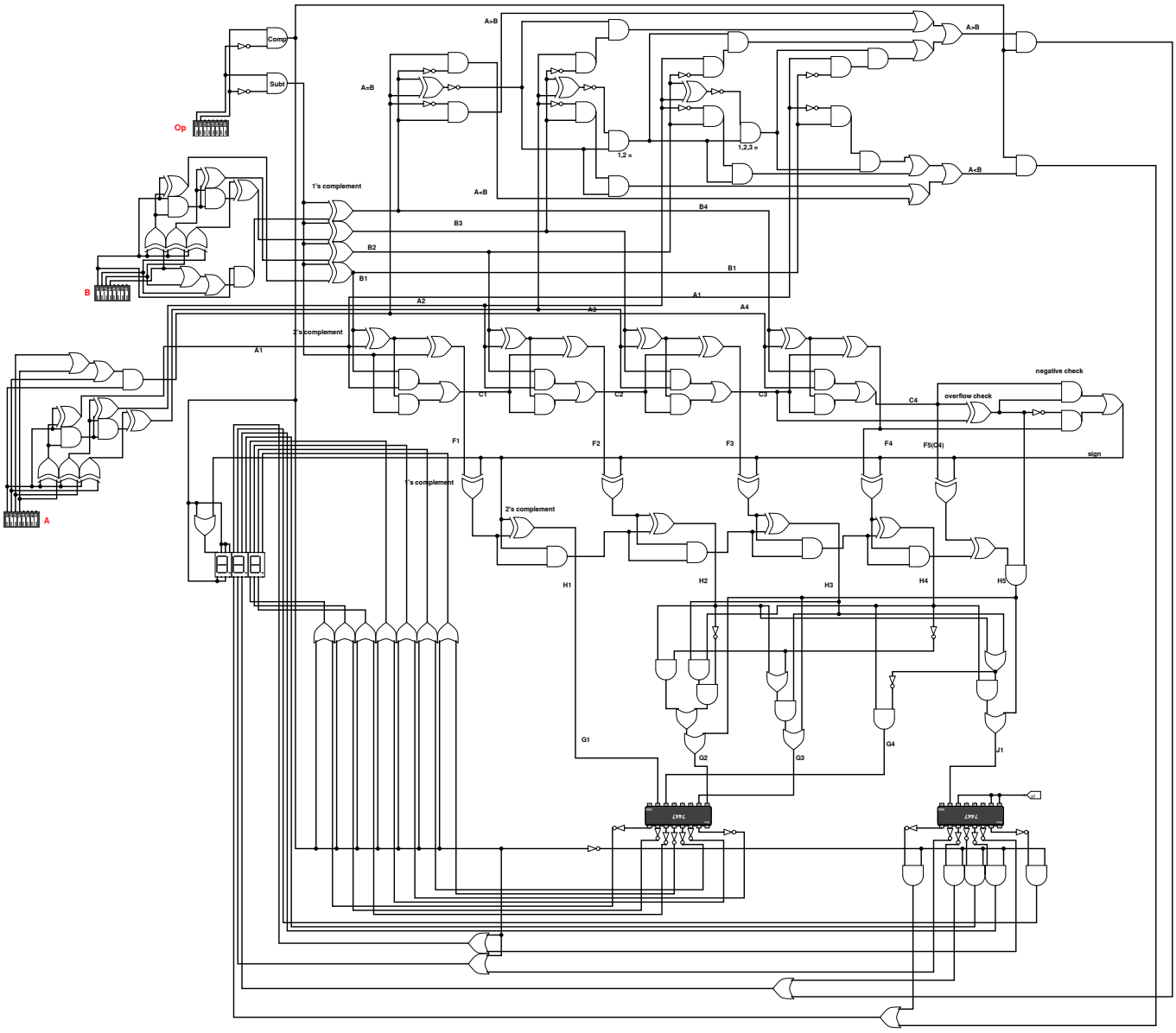
[10] Texas Instruments, "7448 Datasheet," [Online]. Available: https://www.ti.com/lit/ds/symlink/sn54ls47.pdf.

[11] M. M. Mano and M. D. Ciletti, Digital Design With an Introduction to the Verilog HDL.

XI. Appendix

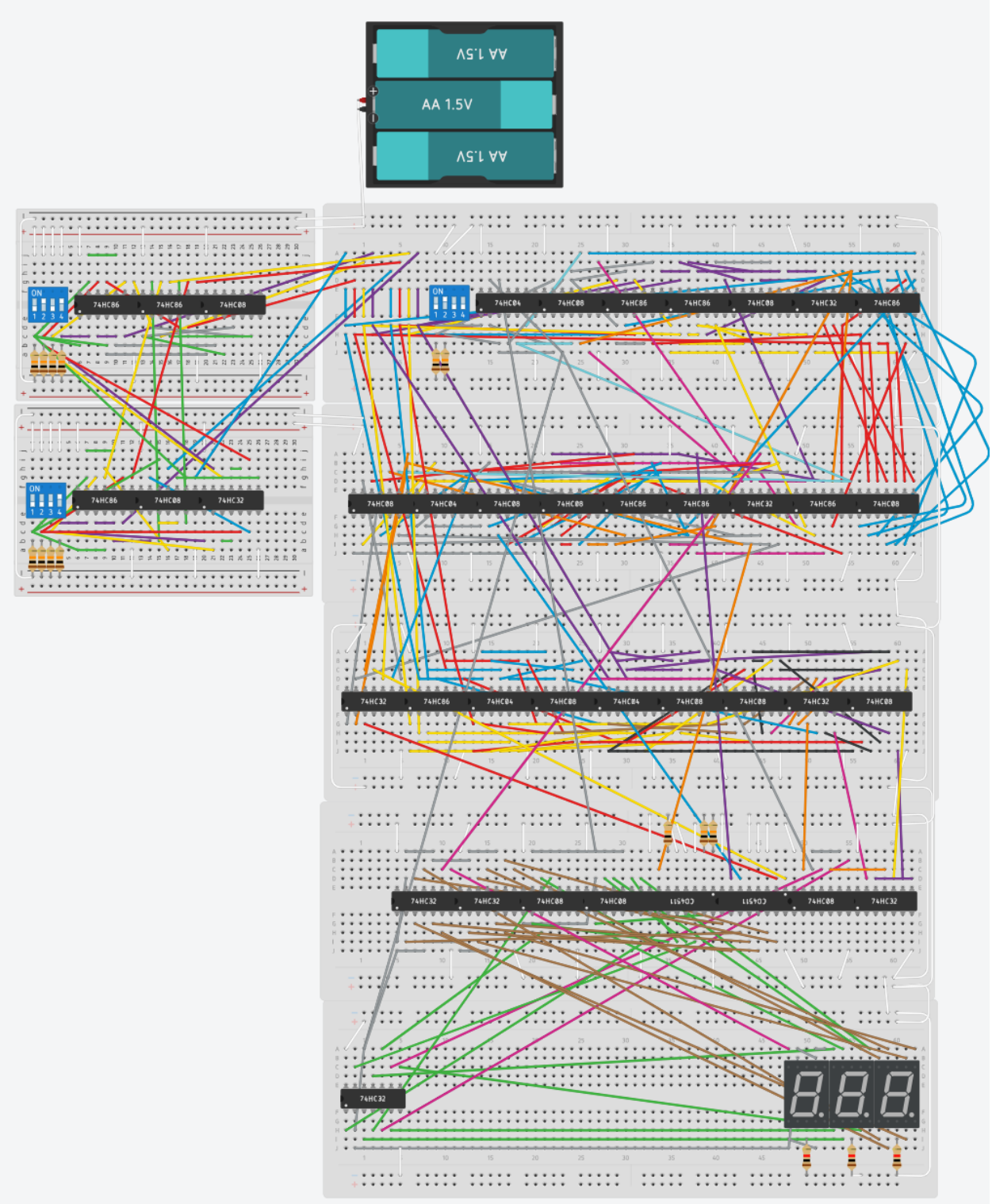
i. Logisim Circuit

Link to Logisim circuit

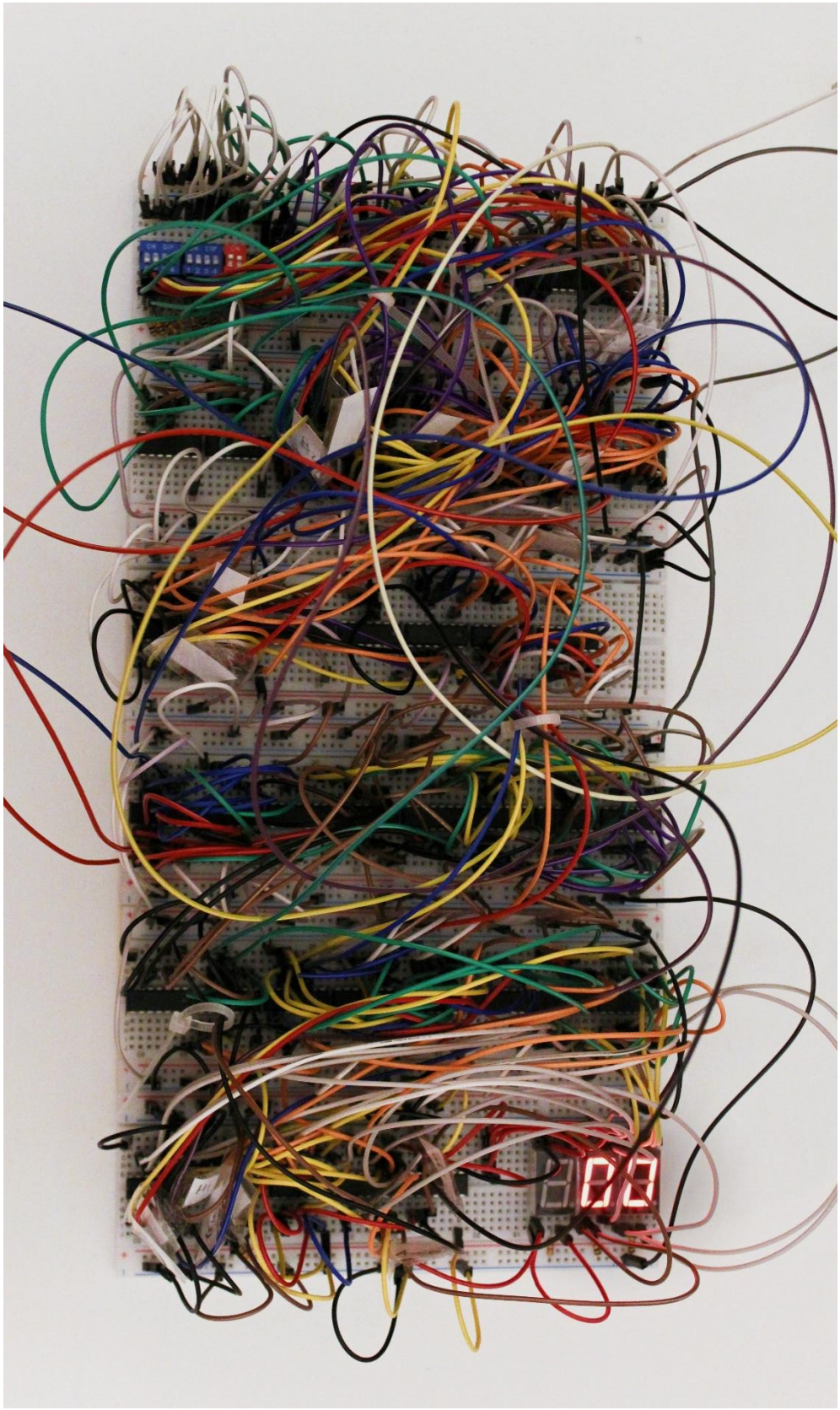


ii. TinkerCAD Circuit

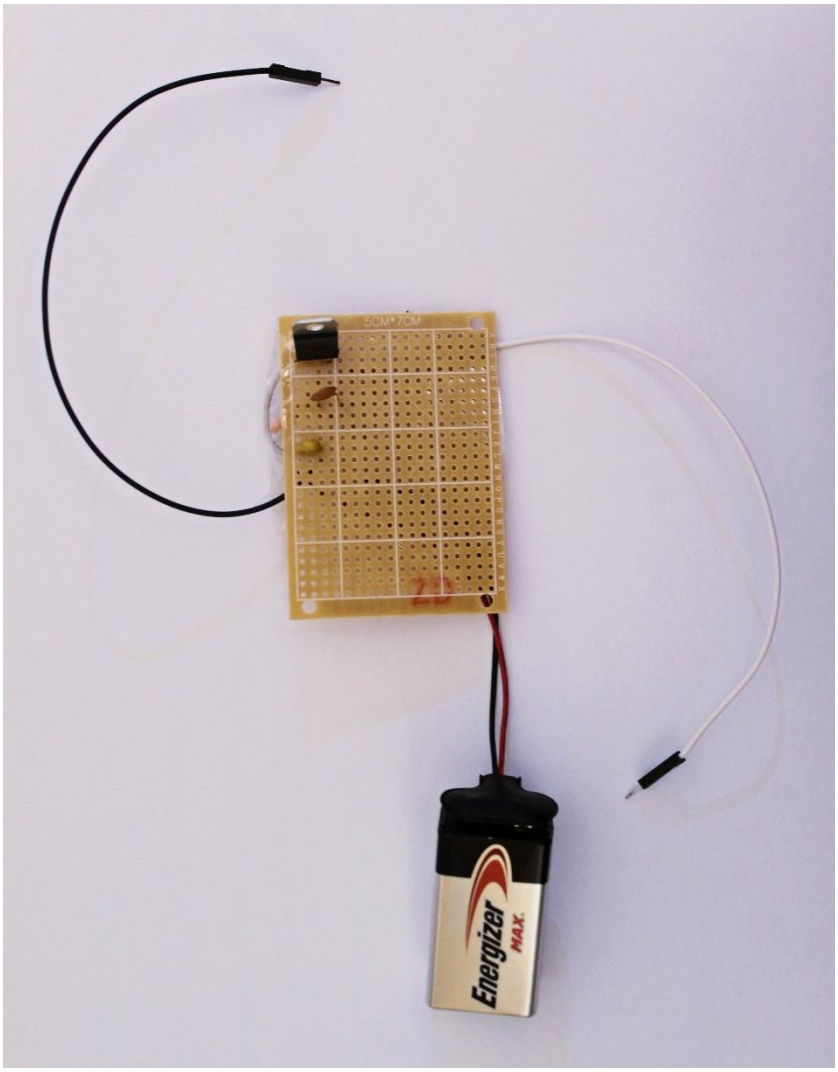
Link to TinkerCAD circuit



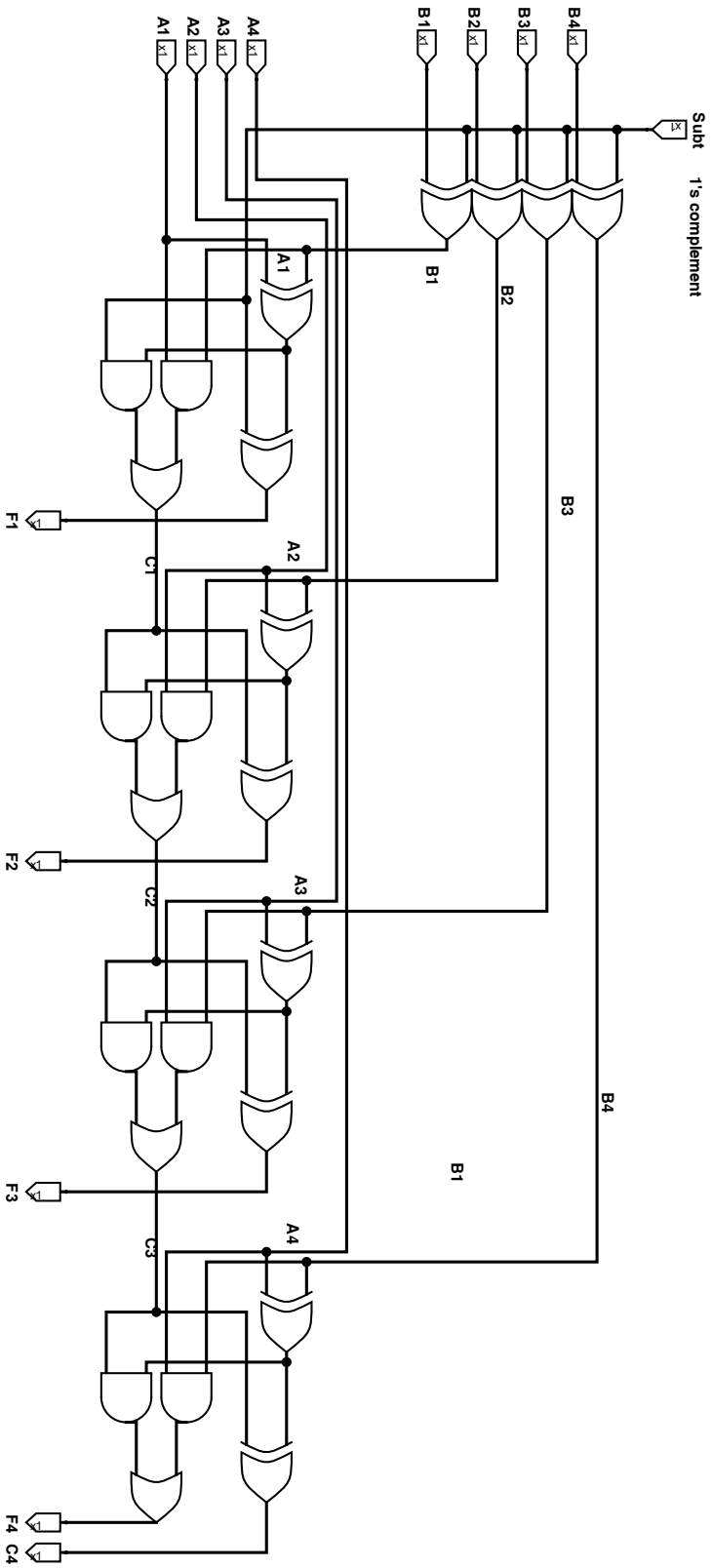
iii. Hardware Circuit



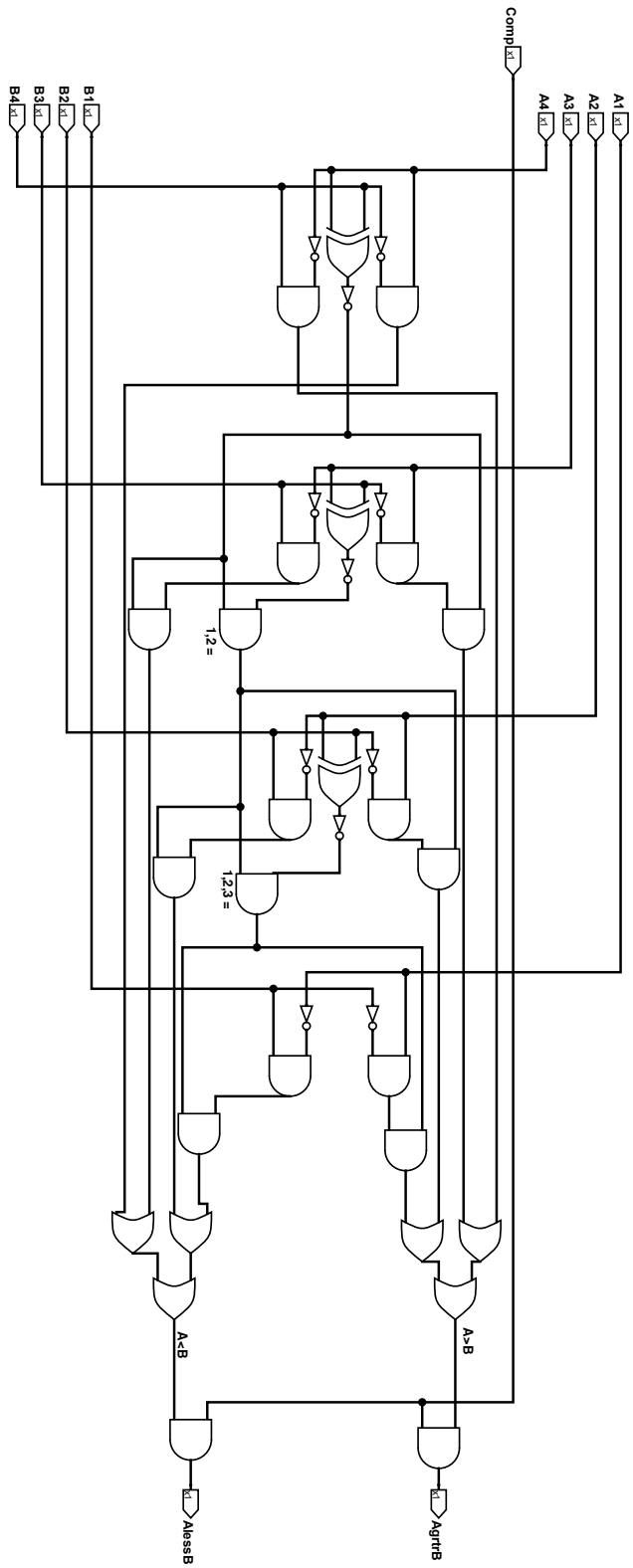
iv. Fixed-Output Regulator



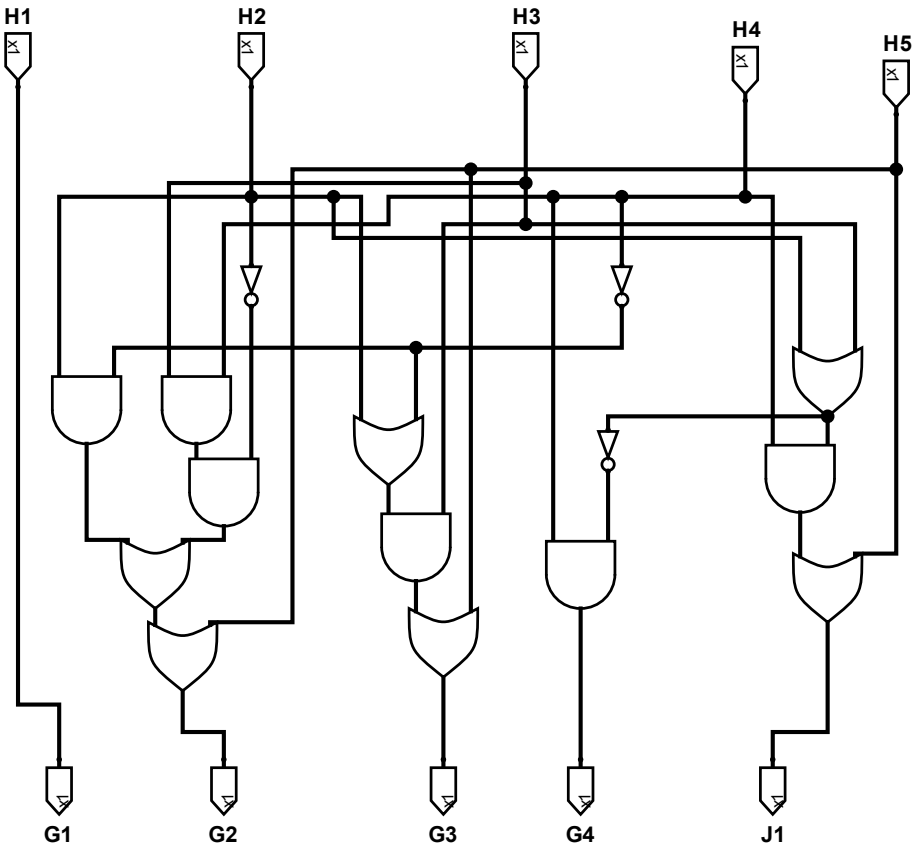
v. Adder/Subtractor Circuit



vi. Comparator Circuit



vii. 5-bit Binary to BCD Converter Circuit (within possible output range)



viii. Some Test Cases

A and B are tested as being entered by the user while the output is the 2's complement form and what would be displayed on the 7-segment displays

For adder:

A	B	Expected Output	Description
1000	0111	0111 (07)	Verifies that negative zero doesn't affect result
0111	1111	0000 (00)	Correct addition for positive and negative numbers
1111	1111	10010 (-14)	Maximum possible negative output
1011	0111	0100 (04)	Correct general addition (random case)
0110	0100	1010 (10)	Correct general addition (random case)

For subtractor:

A	B	Expected Output	Description
1000	0111	1111 (-07)	Verifies that negative zero doesn't affect result
0110	0110	0000 (00)	Correct subtraction of equal numbers
0111	1111	1110 (14)	Maximum possibly positive output
1011	0111	10110 (-10)	Correct general subtraction (random case)
0110	0100	0010 (02)	Correct general subtraction (random case)

For comparator:

A	B	Expected Output	Description
0000	1000	A=B	Verifies both representations of zero are handled correctly
0110	0110	A=B	Correct equality detection
1001	1011	A>B	Correct relation detection for negative numbers
1010	0001	A<B	Correct relation for different-sign numbers
1000	1111	A>B	Correct relation with negative zero

ix. List of Tables

x. List of Figures