Digital Logic Design 3441 – Spring 2021

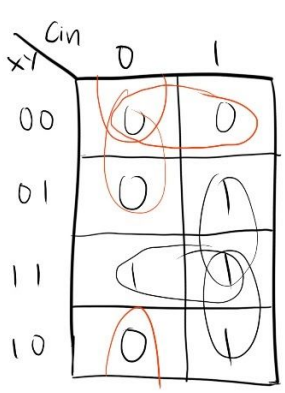
Lab #4: Full-Adder Subtractor Calculator

Cody Betzner, Thomas Nguyen

Overview: Goals of the Lab

The overall goal of this lab is to understand how an 8-bit circuit for arithmetic operations is created. It first introduces the bare minimum gate configurations for an adder, broken into two circuits. It then has us create a 8 bit full adder for subtraction. Following this, an adder that can compute both addition and subtraction is created in order for us to see both the difference and advantage of creating this new circuit. Both lab partners worked collectively on the first part of the lab to solve the K-map and equally divided the SimUAid simulation. Part 2 was done by Thomas and Part 3 was done by Cody.

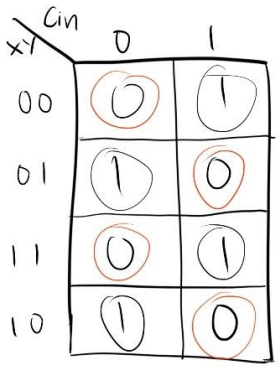
Part 1: Using K-Maps to Solve Full-Adder Truth Table



X	Y	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$C_{out} = C_{in}Y + C_{in}X + XY$$

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

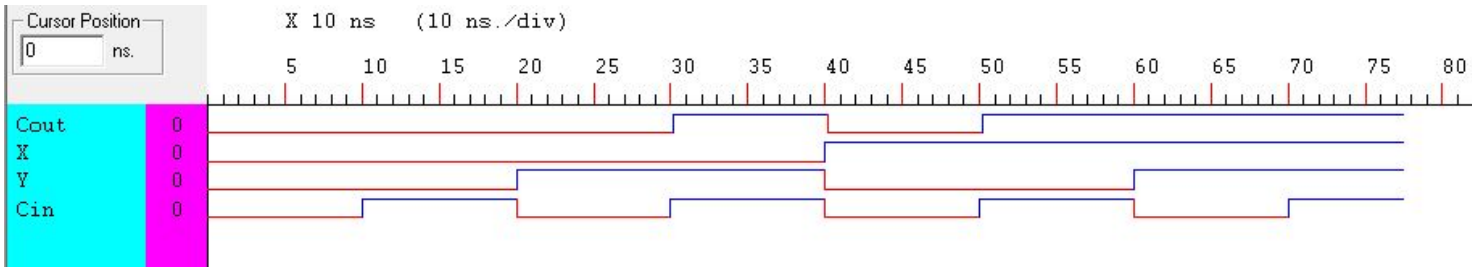


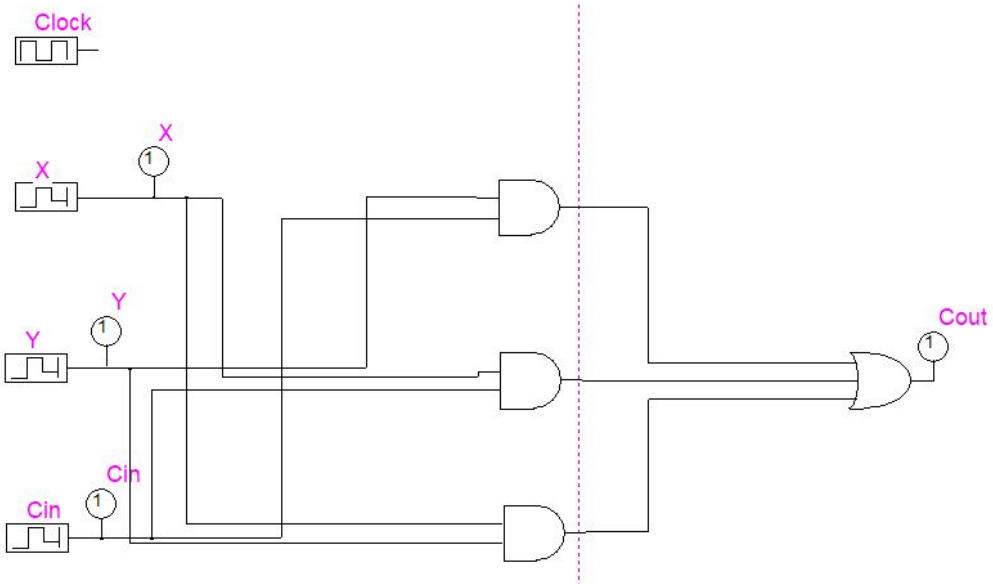
X	Y	Cin	Sum
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$Sum = x'y'Cin + x'yCin' + XYCin + Xy'Cin$$

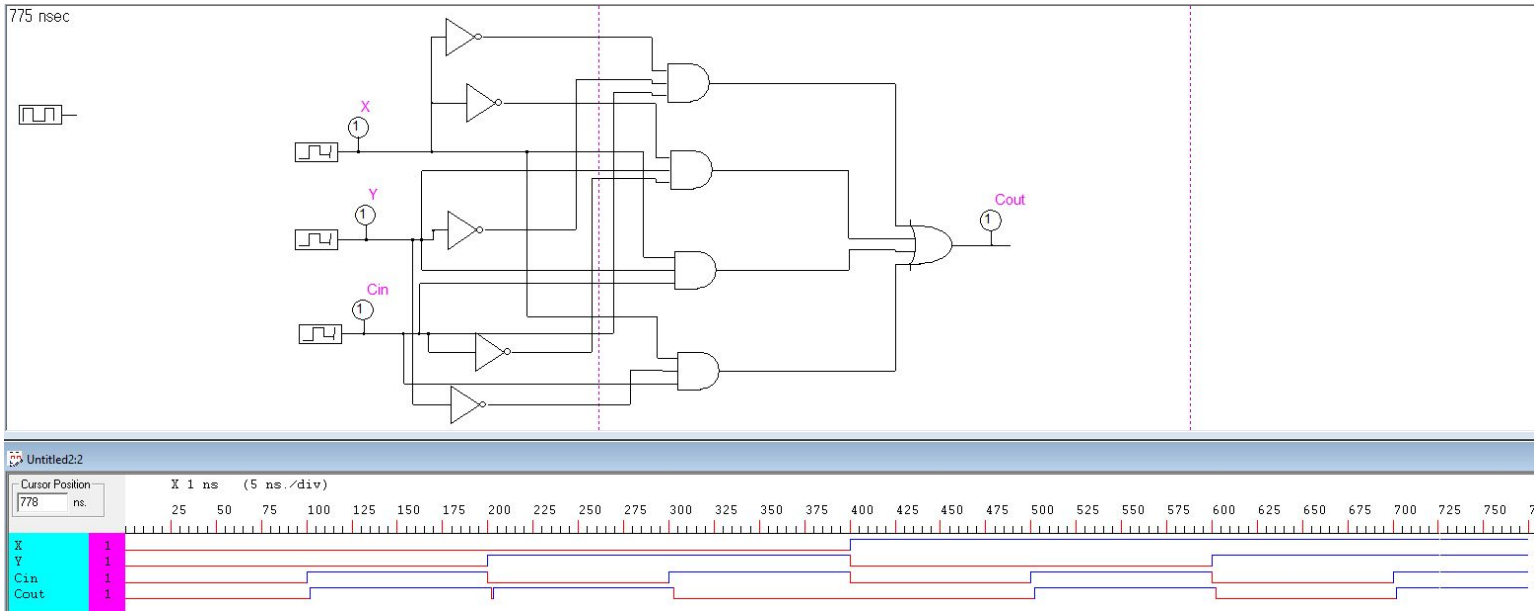
$$Sum = (X+Y+Cin)(X+Y'+Cin')(X'+Y'+Cin)(X'+Y+Cin')$$

Cout SimUAid Timing Diagram and Schematic





Sum SimUAid Timing Diagram and Schematic



1. Cout has 4 minterms. Sum has 4 minterms.

   Maxterm for Cout

   Cout = (Cin + Y)(X + Y)(Cin + X)

   Maxterm for Sum

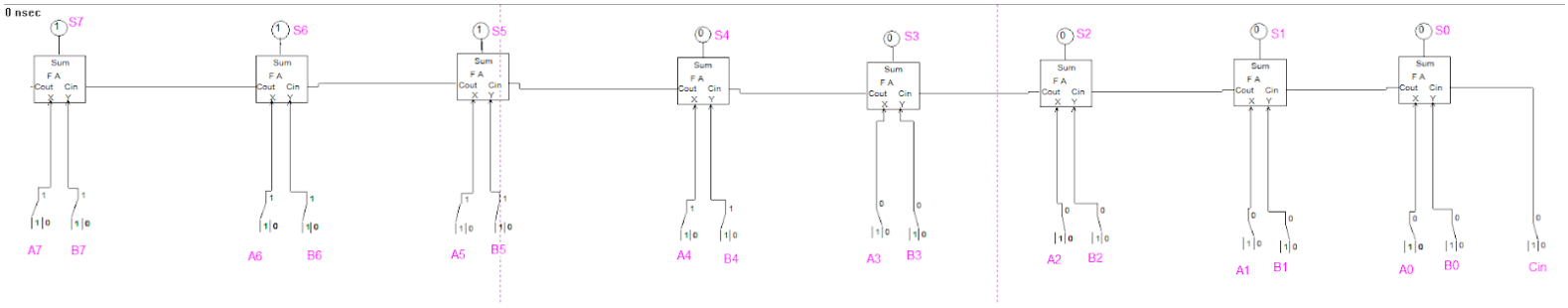
   Sum = (X + Y + Cin)(Cin' + X + Y')(Cin + X' + Y')(Cin' + X' + Y)

2. An ALU is an arithmetic logic unit and it is used to perform arithmetic and logic operations. The Full-Adder is an arithmetic operation, therefore it is an ALU.

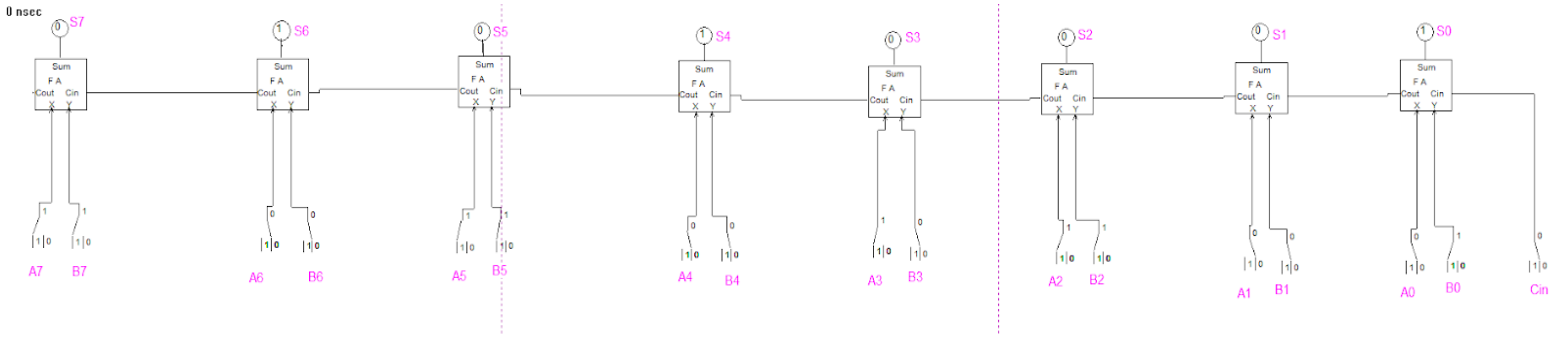
Part 2: Full-Adder Macro Chip Introduction

3. The Part 1 build is more fragmented while the Full-Adder Macro is inclusive of both components. The Full-Adder Macro is the better option because it is a simpler circuit and achieves the same task.
4. One limitation is that it is only limited to 8-bit operations and can't calculate arithmetic that would exceed 8-bits. If the sum requires a result larger than 8-bits, then that number is not fully shown. Another limitation is that it can only do addition.
5. See examples below:

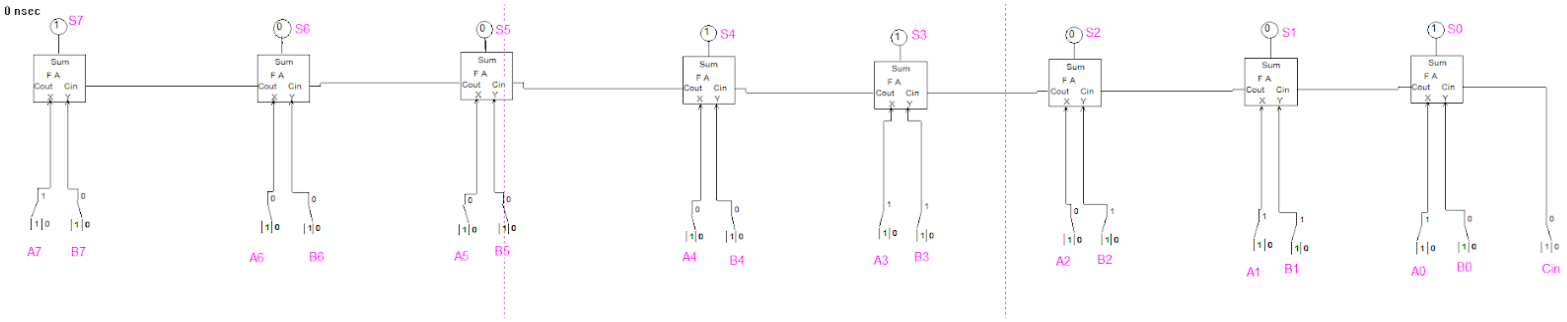
Example 1: 11110000 (-16) + 11110000 (-16) = 11100000 (-32)



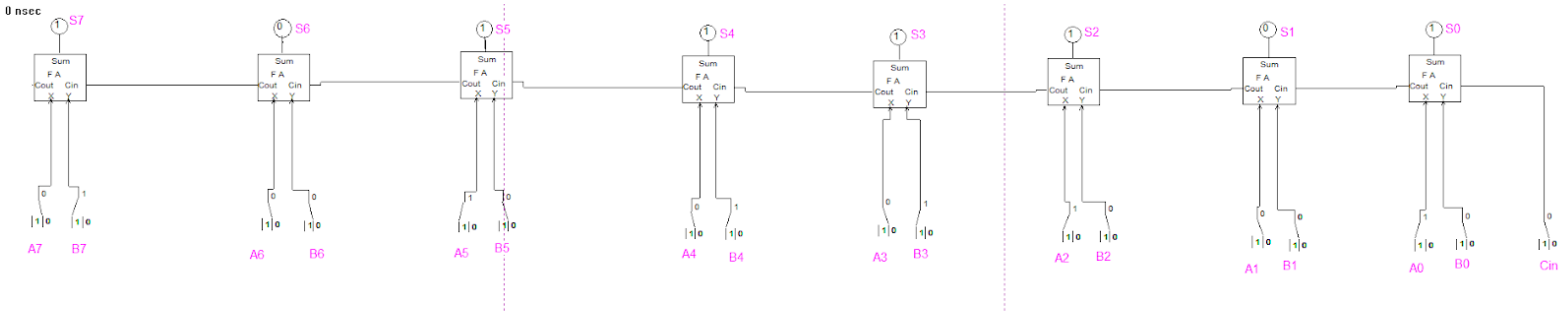
Example 2: 10101100 (-84) + 10100101 (-91) = 01010001 (-175)



Example 3: 10001011 (-117) + 00001110 (14) = 10011001 (-103)



Example 4: 00100101 (37) + 10011000 (-104) = 10111101 (-67)

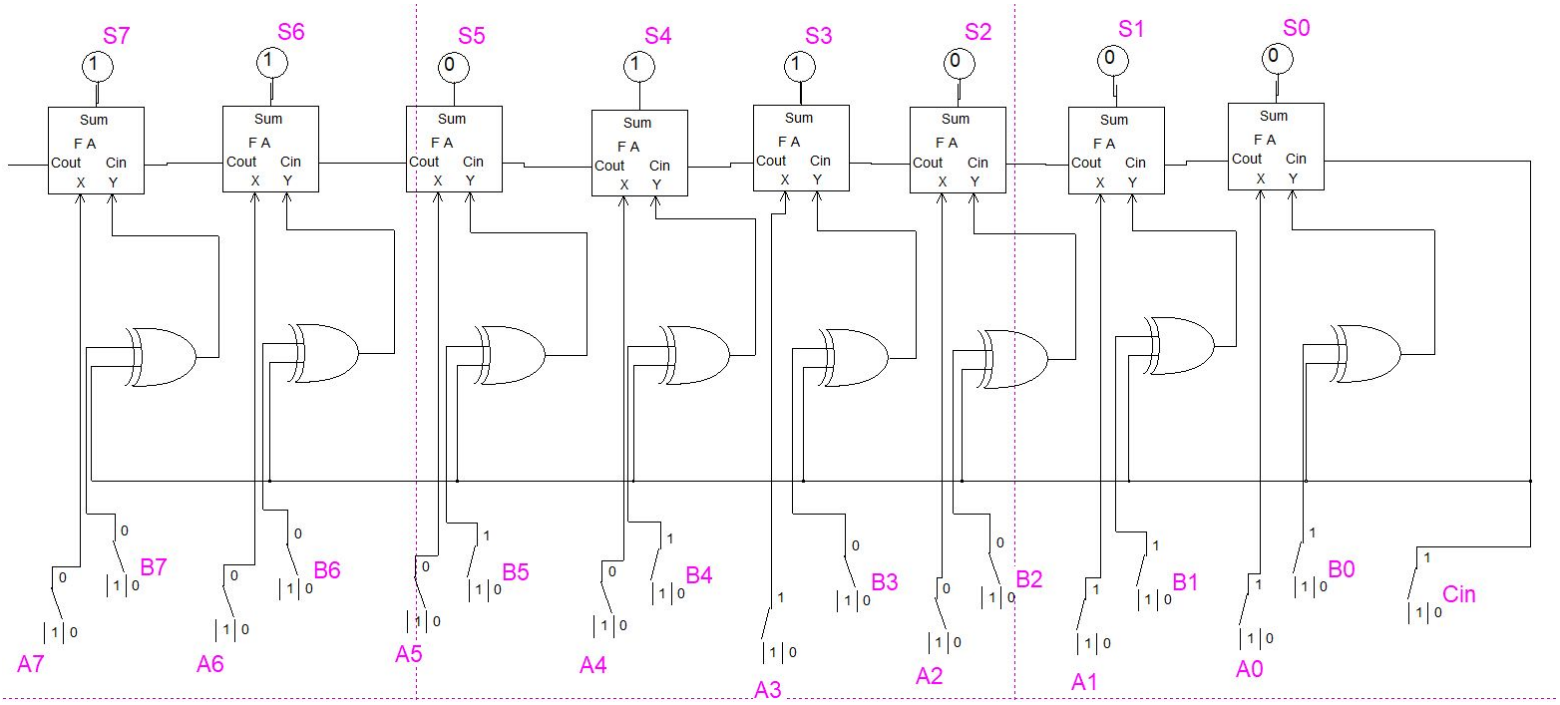


Part 3: Full-Adder Subtractor Build

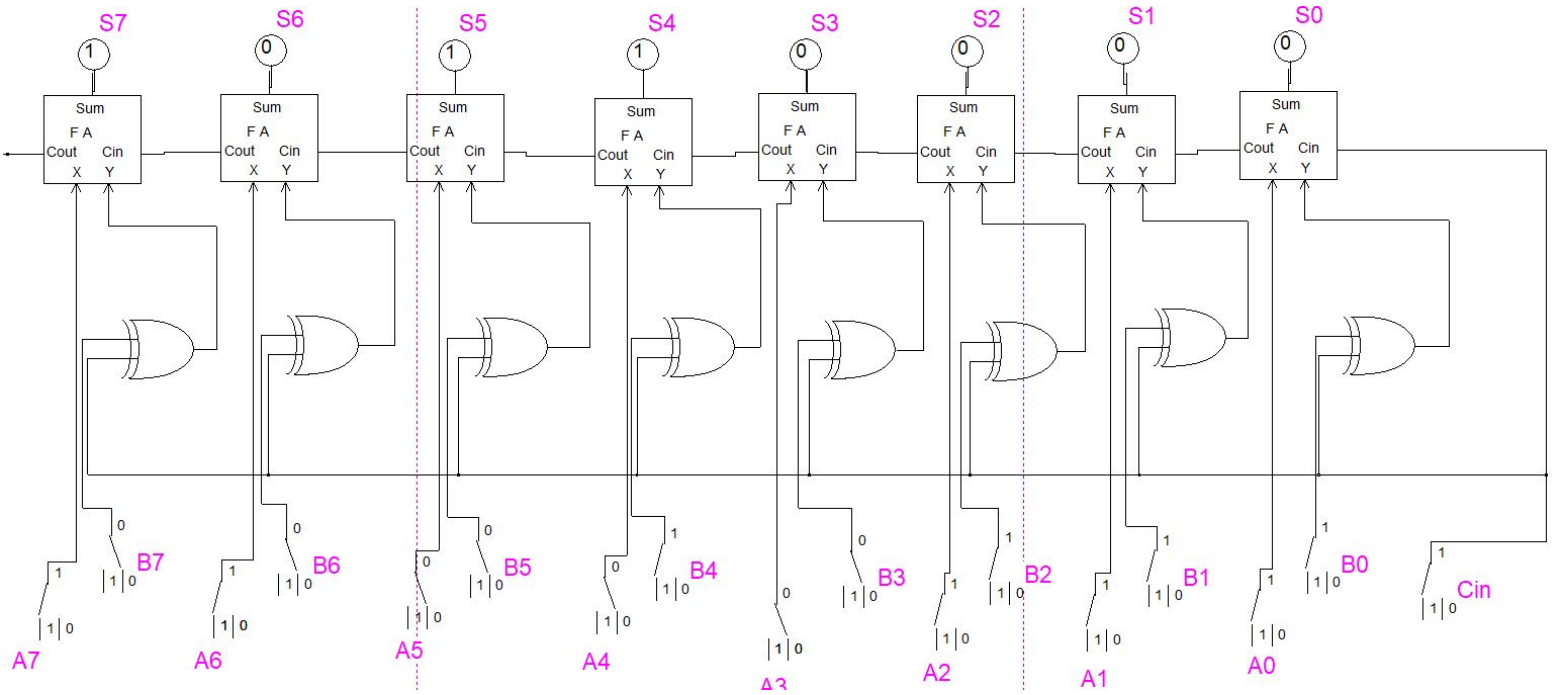
6. Adding the XOR gate allows the circuit a chance to make one of the collective inputs "negative". By pairing a constant value with an input into an XOR gate, we take the one's complement of that input and allow subtraction to occur between this input and the other input. This XOR gate also allows the possibility for standard addition as well.

7. The circuit is now able to perform subtractions. However, a difference that requires 9 bits will still not be shown properly. A new limitation now, not entirely relevant to the simulation, is that the circuit is now dependent on the longest delay in the circuit. This is because the use of the C-in connection to the XOR gate has been implemented.
8. See examples below:

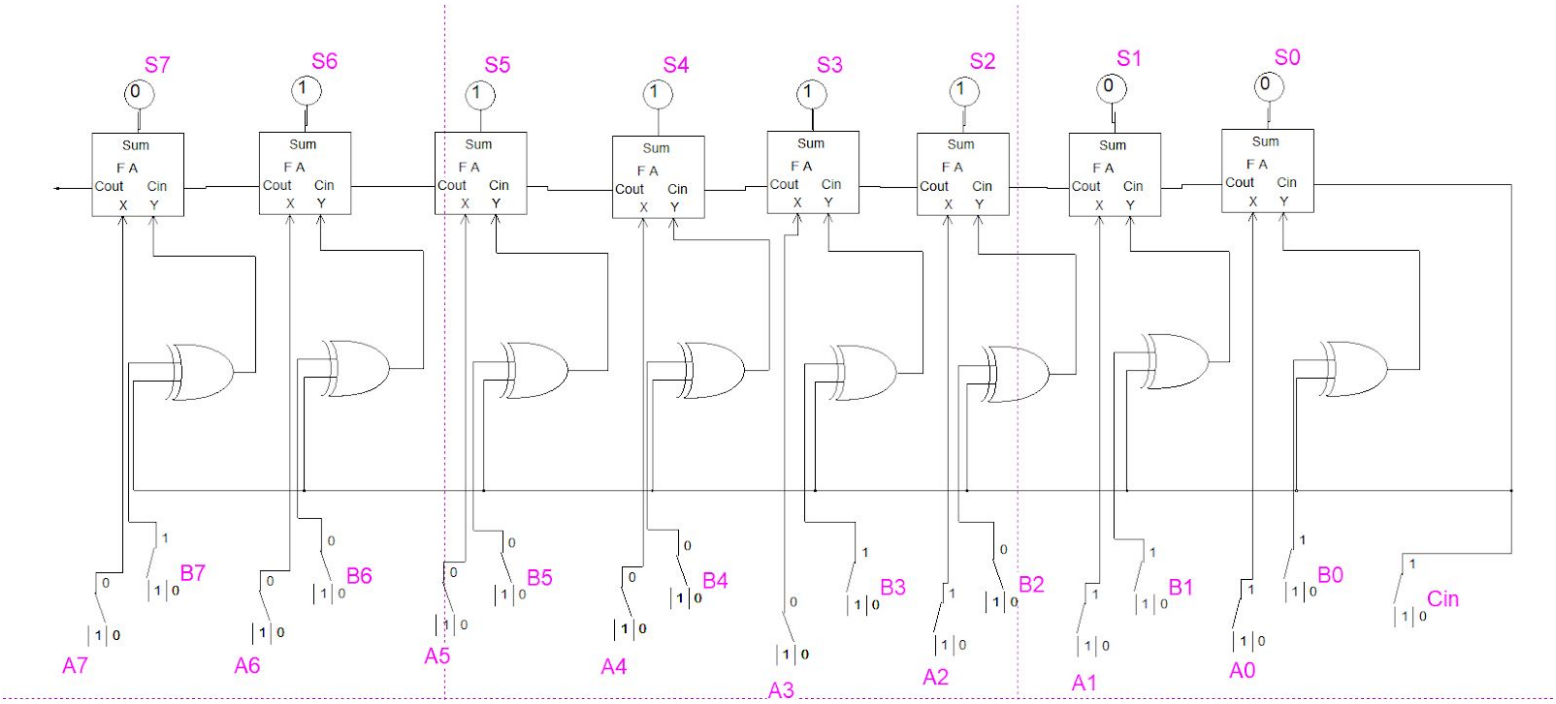
Example 1: 00001011 (11) - 00110011 (51) = 11011000 (-40)



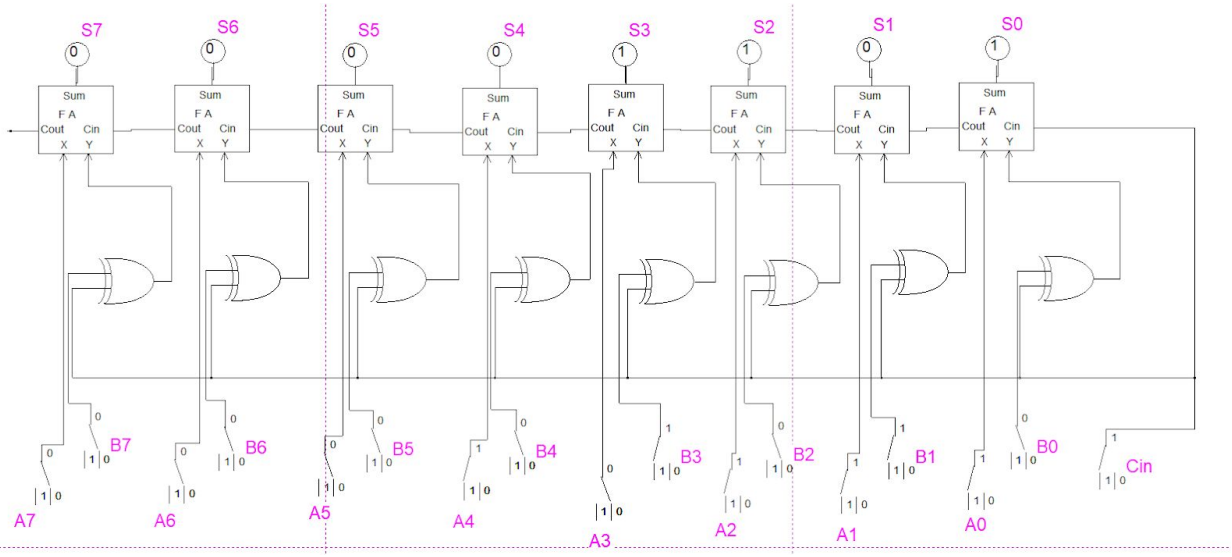
Example 2: 11000111 (-57) - 00010111 (23) = 10110000 (-80)



Example 3: 00000111 (7) - 10001011 (-117) = 01111100(124)



Example 4: 00010111 (23) - 00001010 (10) = 00001101 (13)



Conclusion

In this lab, we learned how to make an 8-bit adder from scratch using logic gates. Starting at the very base level, we built it using K-Maps and the logic table. Increasing in height of simulation, we also built an Adder circuit using logic gates. Finally we learned how to add a subtraction feature to our 8-bit adder which allowed us to have both arithmetic operations available. In summary, we learned the basis of a simple ALU and the logic that goes behind making this component work.