MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

DEPARTMENT OF COMPUTER ENGINEERING

CNG331
TERM PROJECT: PART 1

NAME: MERT CAN
SURNAME: BİLGİN
STUDENT ID: 2453025

1.2.1

In the first question, I created an 8-bit input and labeled it as “In1”.

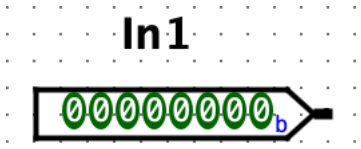


Figure 1

After that, I added the outputs and labeled them as “O1” and “O2”.

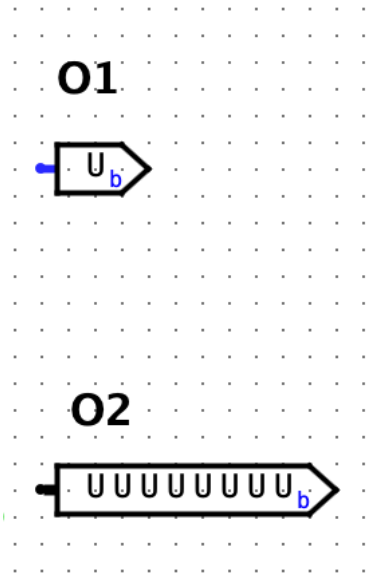


Figure 2

Then, I set the splitter as in Figure 3.

Properties	State
Splitter (180,250)	
FPGA supported:	Supported
Facing	→ East
Fan Out	3
Bit Width In	8
Appearance	Left-handed
Spacing	1
Bit 0	0 (↑ Top)
Bit 1	1
Bit 2	1
Bit 3	1
Bit 4	1
Bit 5	1
Bit 6	1
Bit 7	2 (↓ Bottom)

Figure 3

After setting the splitter, I placed it to the circuit, and arranged the input and outputs according to the instructions in the Term Project.

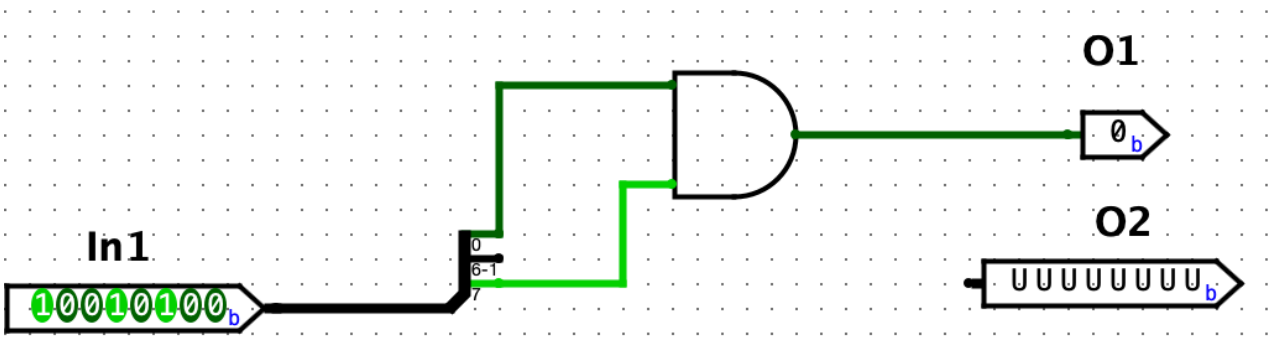


Figure 4

By the help of splitter, I used the most significant bit as a signed value, so if I change the most significant bit from '1' to '0', I can make the value positive. Otherwise, it will be negative. To do that, I used a NOT gate, and connected them to output by using a splitter.

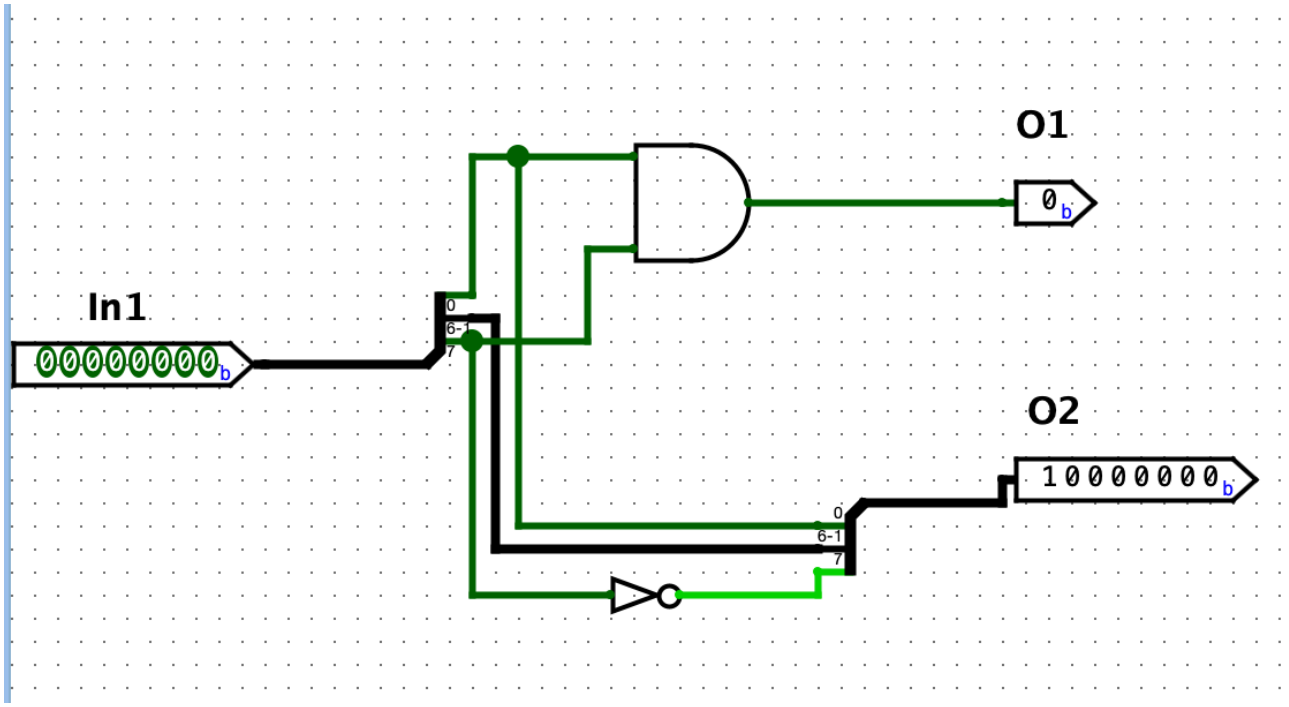


Figure 5

Then, I tested my circuit using poke tool as in Figure 6 and Figure 7. In the Figure 6, I tested the value "00010100", and the output is resulted as "10010100" by making the most significant bit as '1'. For the Figure 7, the number turned to positive from negative by using the same method. Therefore, the circuit works.

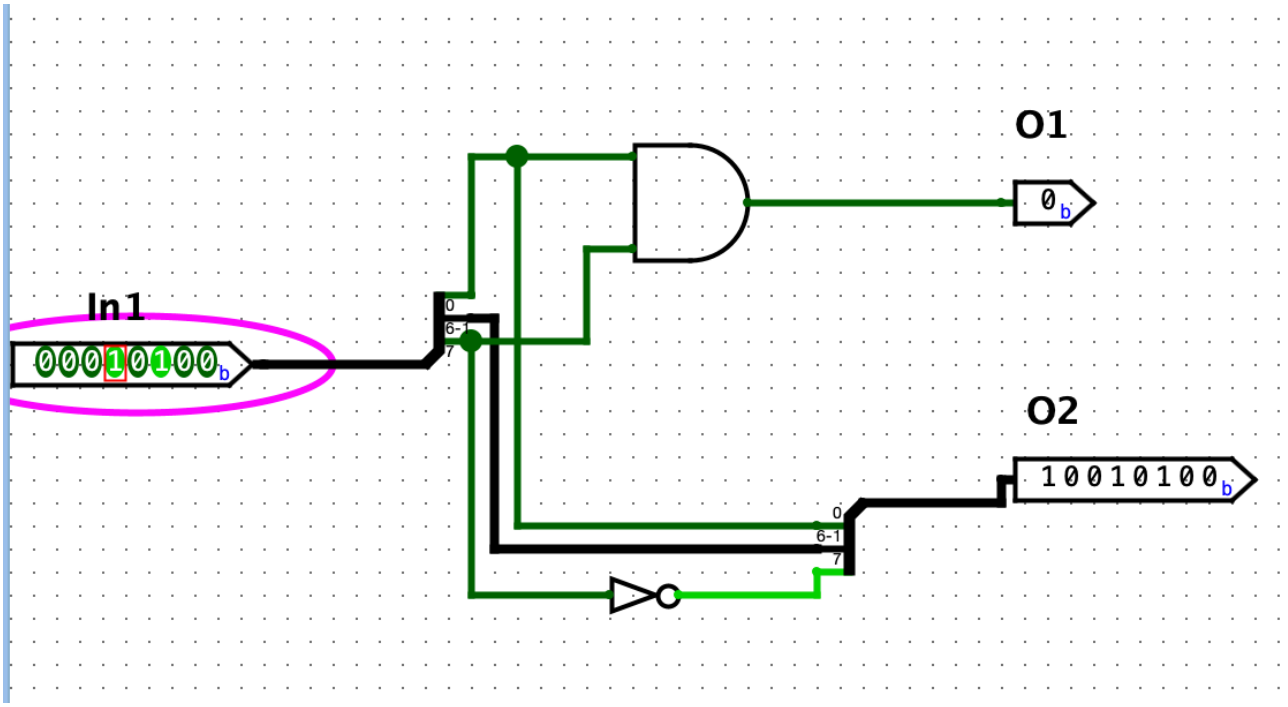


Figure 6

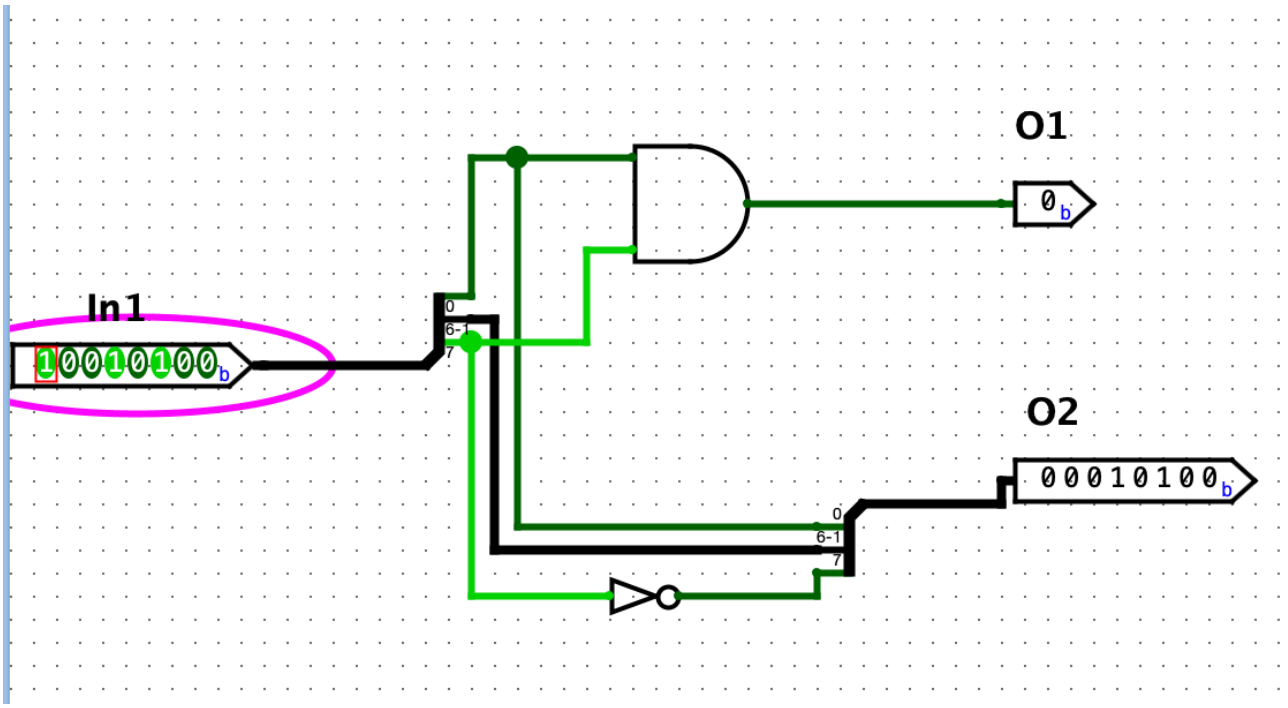


Figure 7

1.2.1.1

In this question, I designed two sub-circuits one to rotate-right and one to rotate-left. There are who inputs A and B. A is the value we want to rotate, and the B is the rotation amount. A is 8-bit, and B is 3-bit because we can rotate 7-bit at maximum, and 7 can be represented as “111” and it is 3-bit. The answer can also be formulised as (Size of A) = $2^{(\text{Size of B})}$. I used multiplexers to design rotl. In Figure 8, D labels shows the amount of rotation; for example, D7 means the number is rotated 7-bit to the left. Also, I connected B input to selection input of the multiplexers. I tested it bu using poke tool as in Figure 8. The value “11100101” is rotated by “011”, and gave the result “00101111” as expected.

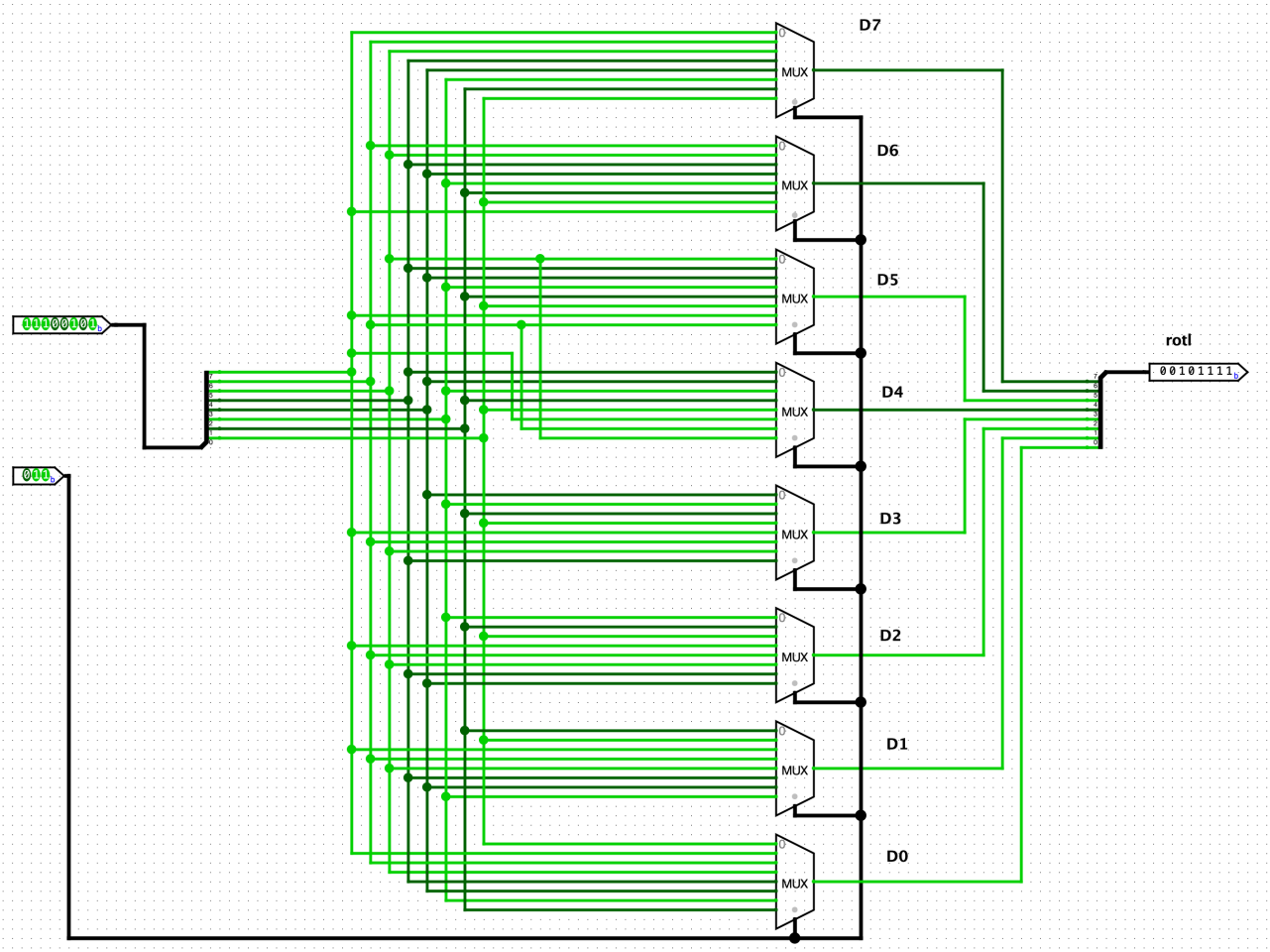


Figure 8

I designed the circuit to rotate right with the same technique, but changing the multiplexer inputs. After that, I also tested it by using poke tool as in Figure 9. I set the A as “00011000” and B as “011”, and the result was given as “00000011” as I wanted. Therefore, rotation circuits works correctly.

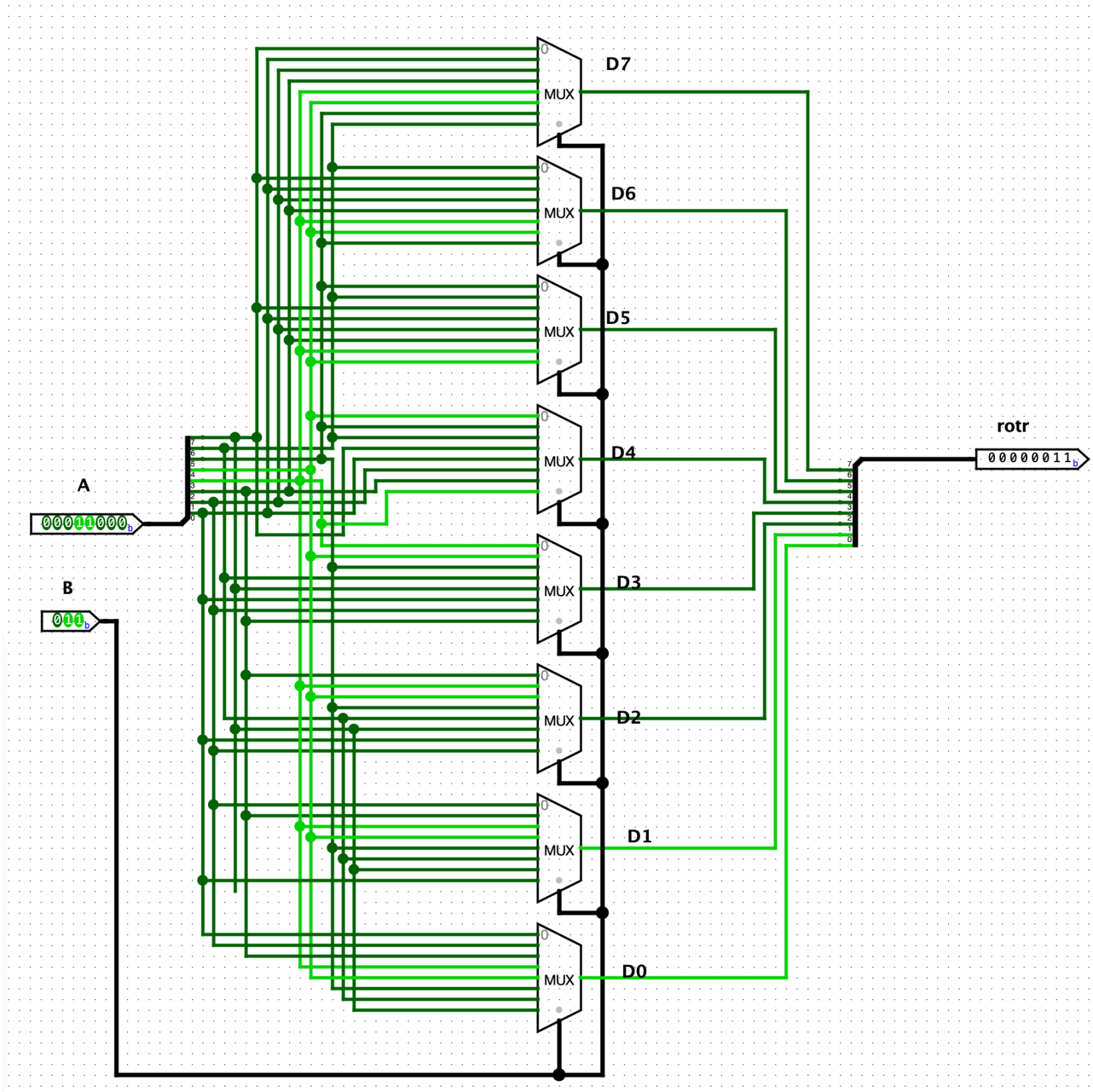


Figure 9

1.3.1

In this question I built an 8-bit ALU. Before starting, I designed it by cutting into small pieces. First of all, I designed a 1-bit full adder by using 2 XOR gate, 2 AND gate and a OR gate as in Figure 10.

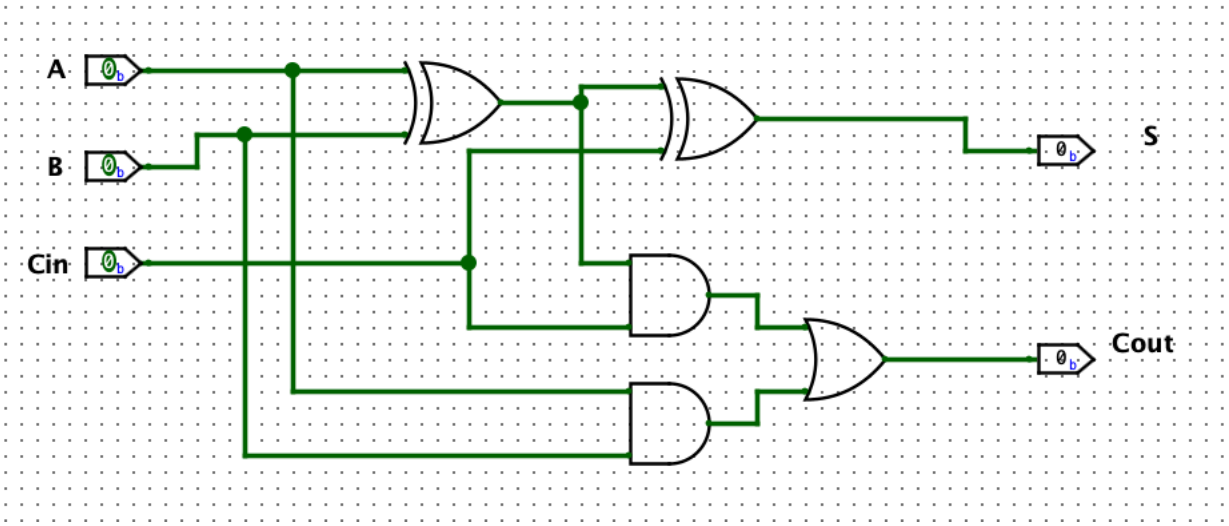


Figure 10

By using the 1-bit full adder, I constructed an 8-bit full adder by using Ripple Carry method as in Figure 11.

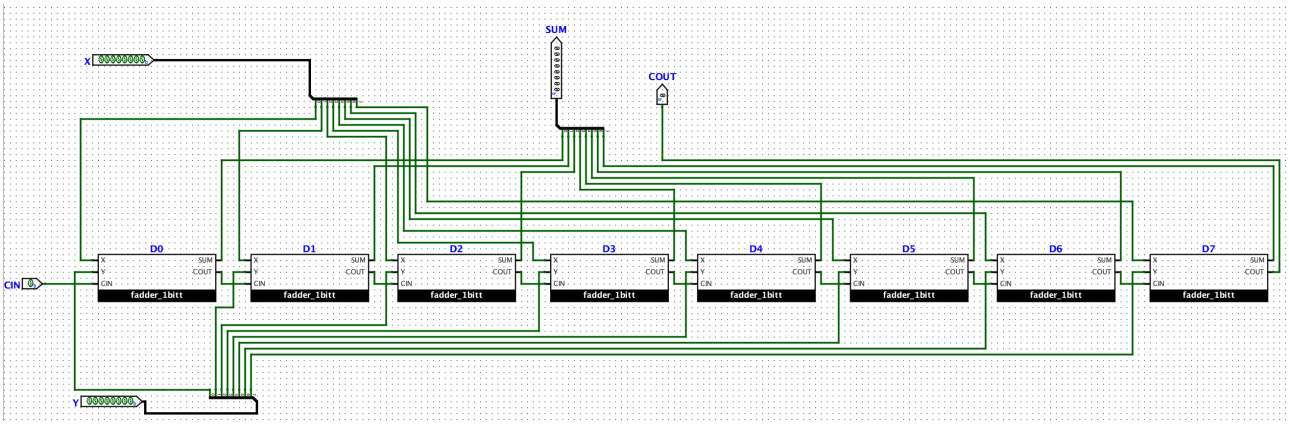
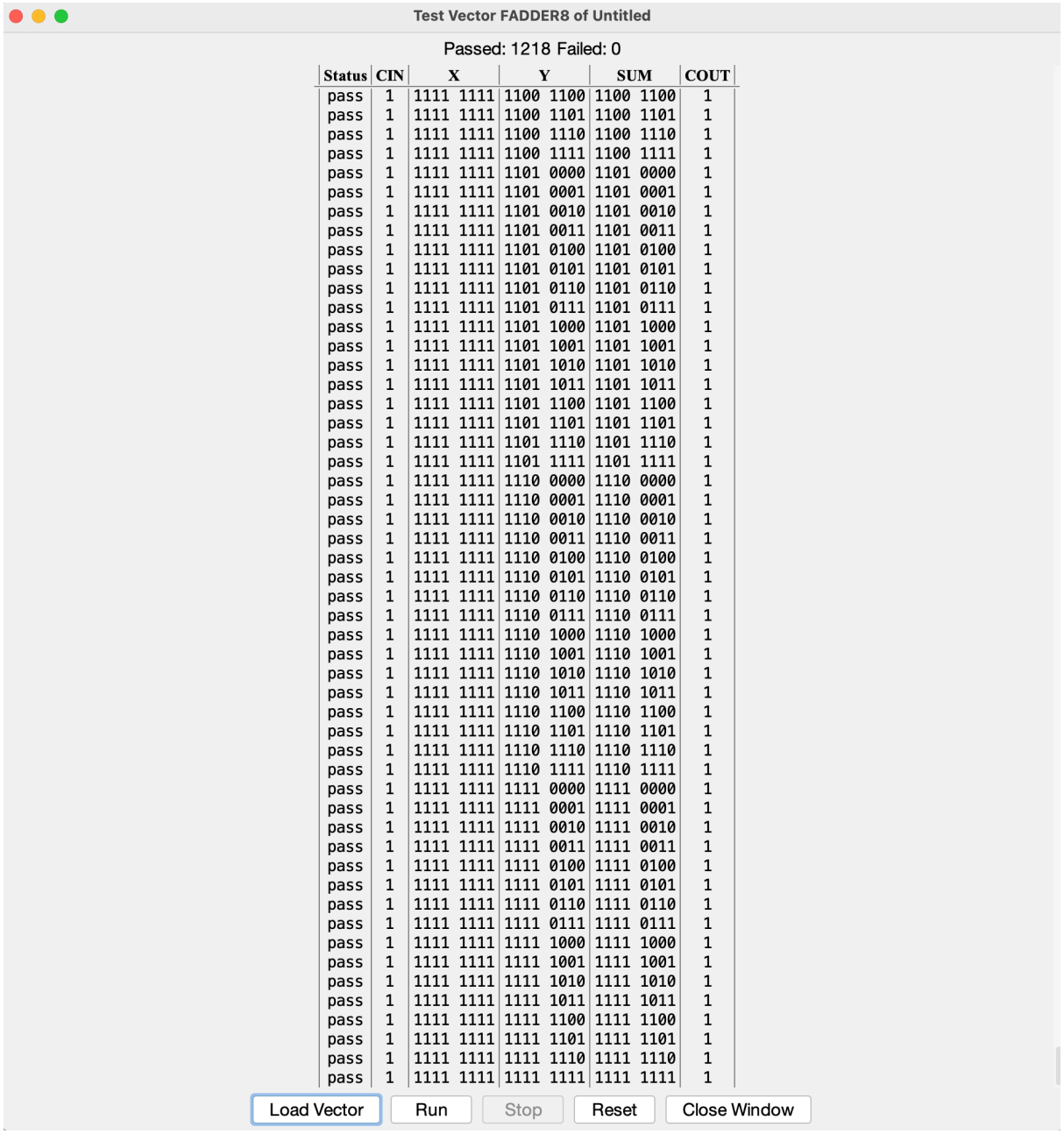


Figure 11

Then I tested my design by using the test vector uploaded on ODTUClass, and verified that the circuit works correctly.



Status	CIN	X	Y	SUM	COUT
pass	1	1111 1111	1100 1100	1100 1100	1
pass	1	1111 1111	1100 1101	1100 1101	1
pass	1	1111 1111	1100 1110	1100 1110	1
pass	1	1111 1111	1100 1111	1100 1111	1
pass	1	1111 1111	1101 0000	1101 0000	1
pass	1	1111 1111	1101 0001	1101 0001	1
pass	1	1111 1111	1101 0010	1101 0010	1
pass	1	1111 1111	1101 0011	1101 0011	1
pass	1	1111 1111	1101 0100	1101 0100	1
pass	1	1111 1111	1101 0101	1101 0101	1
pass	1	1111 1111	1101 0110	1101 0110	1
pass	1	1111 1111	1101 0111	1101 0111	1
pass	1	1111 1111	1101 1000	1101 1000	1
pass	1	1111 1111	1101 1001	1101 1001	1
pass	1	1111 1111	1101 1010	1101 1010	1
pass	1	1111 1111	1101 1011	1101 1011	1
pass	1	1111 1111	1101 1100	1101 1100	1
pass	1	1111 1111	1101 1101	1101 1101	1
pass	1	1111 1111	1101 1110	1101 1110	1
pass	1	1111 1111	1101 1111	1101 1111	1
pass	1	1111 1111	1110 0000	1110 0000	1
pass	1	1111 1111	1110 0001	1110 0001	1
pass	1	1111 1111	1110 0010	1110 0010	1
pass	1	1111 1111	1110 0011	1110 0011	1
pass	1	1111 1111	1110 0100	1110 0100	1
pass	1	1111 1111	1110 0101	1110 0101	1
pass	1	1111 1111	1110 0110	1110 0110	1
pass	1	1111 1111	1110 0111	1110 0111	1
pass	1	1111 1111	1110 1000	1110 1000	1
pass	1	1111 1111	1110 1001	1110 1001	1
pass	1	1111 1111	1110 1010	1110 1010	1
pass	1	1111 1111	1110 1011	1110 1011	1
pass	1	1111 1111	1110 1100	1110 1100	1
pass	1	1111 1111	1110 1101	1110 1101	1
pass	1	1111 1111	1110 1110	1110 1110	1
pass	1	1111 1111	1110 1111	1110 1111	1
pass	1	1111 1111	1111 0000	1111 0000	1
pass	1	1111 1111	1111 0001	1111 0001	1
pass	1	1111 1111	1111 0010	1111 0010	1
pass	1	1111 1111	1111 0011	1111 0011	1
pass	1	1111 1111	1111 0100	1111 0100	1
pass	1	1111 1111	1111 0101	1111 0101	1
pass	1	1111 1111	1111 0110	1111 0110	1
pass	1	1111 1111	1111 0111	1111 0111	1
pass	1	1111 1111	1111 1000	1111 1000	1
pass	1	1111 1111	1111 1001	1111 1001	1
pass	1	1111 1111	1111 1010	1111 1010	1
pass	1	1111 1111	1111 1011	1111 1011	1
pass	1	1111 1111	1111 1100	1111 1100	1
pass	1	1111 1111	1111 1101	1111 1101	1
pass	1	1111 1111	1111 1110	1111 1110	1
pass	1	1111 1111	1111 1111	1111 1111	1

Figure 12

After verifying that the full adder works, I started to design the ALU. By using the multiplexer, I connected the output of full adder to the first input of MUX. Also, I need a Carry-in input, but I decided that it is the least significant bit of opcode in my design. For the second operation, I used an 8-bit AND gate to get the result, and for the third operation I placed an 8-bit OR gate. In addition to that, I used the least 3-bit of Y input for the rotation amount in the last operation. For instance, if the input is “00000011”, the rotation amount will be “011”. To do that I used splitter to get the necessary bits, I selected ‘None’ option for the others.

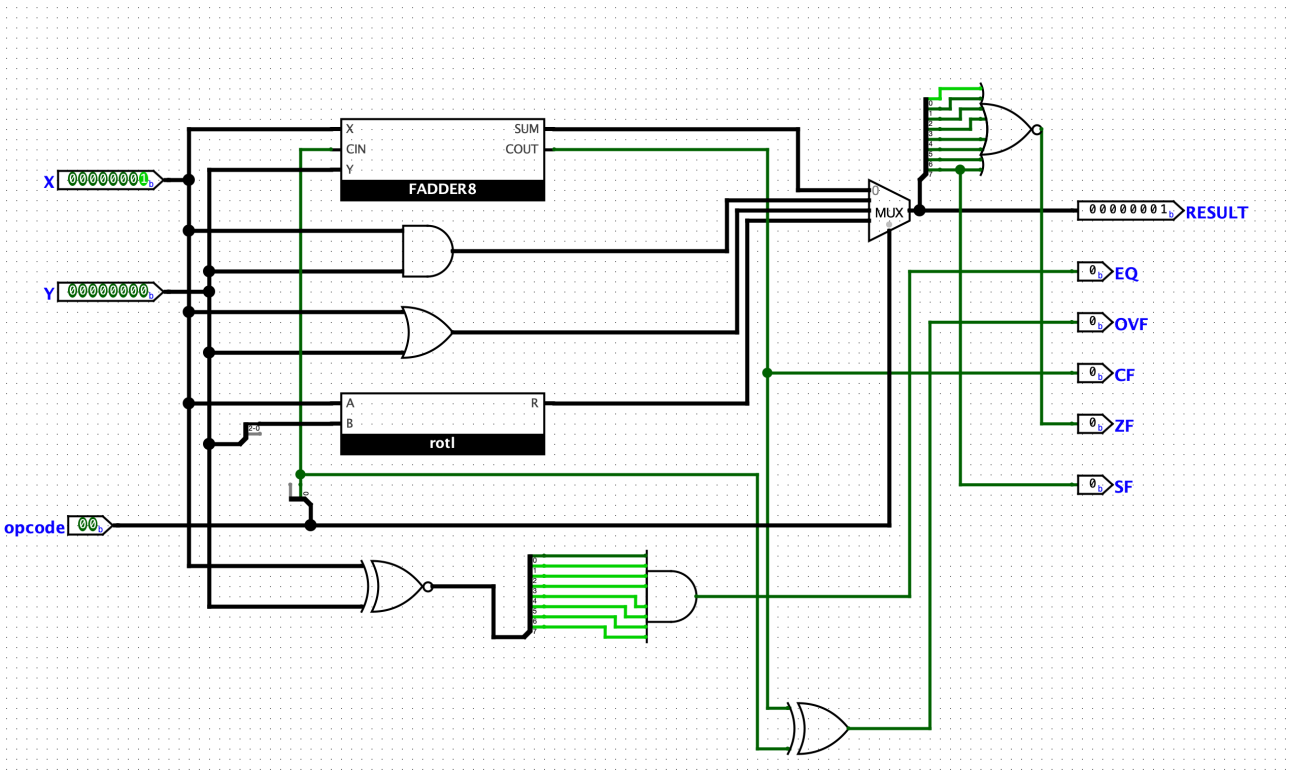


Figure 13

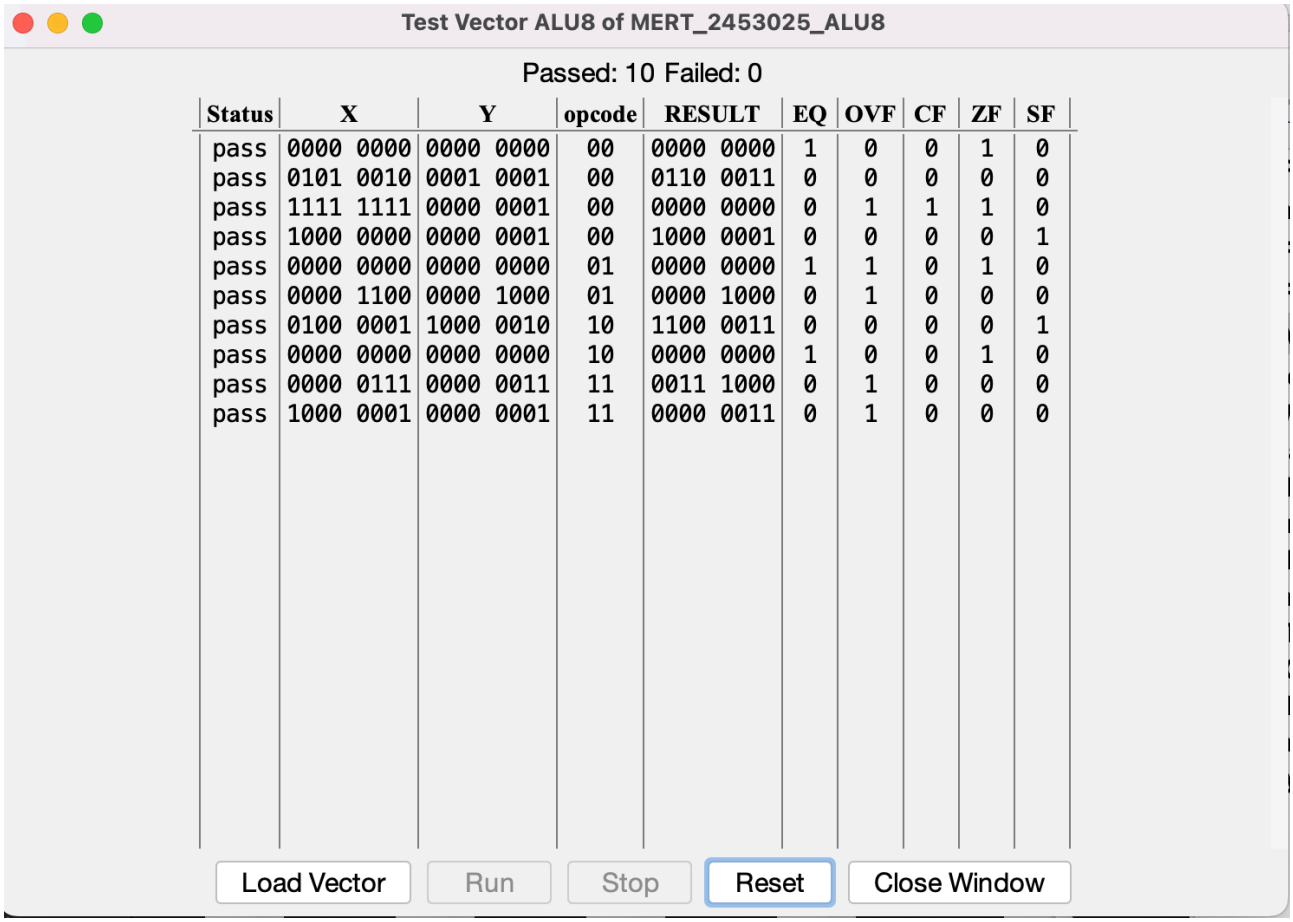
Figure 13 represents the ALU design. There are five flag bits as outputs. First, it check whether the numbers are equal or not. To decide the equality, I used an 8-bit XNOR gate, so if the numbers are different from each other the result will be “11111111”, and I connected them to an AND gate by using splitter in order to decide each digit is equal. Secondly, OVF flag decides signed overflow, and it only happens when CIN XOR COUT is ‘1’. I got this result by the help of truth table in Figure 14.

X	Y	CIN	COUT	SUM	OVF
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

Figure 14

Thirdly, CF flag decides if there is an overflow for unsigned numbers. It is more simple in unsigned numbers, if COUT is '1', it means there is an overflow. Therefore, I directly connected COUT output to CF flag. Fourthly, I designed ZF flag, decides if the result equals to zero. To do that, I used an 8-bit NAND gate with the help of splitter, so if the result is "00000000", NAND gate will be '1'. This technique helped me to check if the result is zero. Finally, I constructed SF flag which checks if the result is negative. The result is negative only when the most significant bit is '1', so I connected the most significant bit to SF flag by using splitter.

After completing all the steps, I wrote a test vector as in Figure 15.



Status	X	Y	opcode	RESULT	EQ	OVF	CF	ZF	SF
pass	0000 0000	0000 0000	00	0000 0000	1	0	0	1	0
pass	0101 0010	0001 0001	00	0110 0011	0	0	0	0	0
pass	1111 1111	0000 0001	00	0000 0000	0	1	1	1	0
pass	1000 0000	0000 0001	00	1000 0001	0	0	0	0	1
pass	0000 0000	0000 0000	01	0000 0000	1	1	0	1	0
pass	0000 1100	0000 1000	01	0000 1000	0	1	0	0	0
pass	0100 0001	1000 0010	10	1100 0011	0	0	0	0	1
pass	0000 0000	0000 0000	10	0000 0000	1	0	0	1	0
pass	0000 0111	0000 0011	11	0011 1000	0	1	0	0	0
pass	1000 0001	0000 0001	11	0000 0011	0	1	0	0	0

Figure 15

In the first line, I tested to add "00000000" and "00000000" in order to check if EQ and ZF flags and adder work correctly. The result is "00000000" as expected, and we can see that EQ and ZF is '1', so they work. In the second line, I tested to add "01010010" and "00010001" and the result is "01100011", and also they are not equal and the result is not zero, so all the flags work. In the third line, I tested overflow, I added "11111111" and "00000001", the result was given me as "00000000" with a OVF and CF flags, so we can see that they work. In the fifth line, it uses AND gate, and gives correct result, but with an OVF because I connected the least significant bit of opcode to CIN in my design. In the seventh line, I checked SF flag with OR gate for "01000001" and "10000010", it gave the correct result because the design checks the most significant bit, so SF is '1' at that time. In the ninth line, I checked rotl funtion with "00000111" and "00000011", it means the number should be rotates by 3-bit. It gave the correct result, but OVF is '1' because of my design decision as I explained above.

CONCLUSION:

With the help of this assignment, I warmed up to designing circuits by using Logisim. I also remembered what I learned from CNG242 (Logic Design). With the help of splitters, I realized that it eases the process of design. The techniques that I learned from this assignment opened new doors for me in my career and academic life.