MIDDLE EAST TECHNICAL UNIVERSITY

NORTHERN CYPRUS CAMPUS

Computer Engineering Program

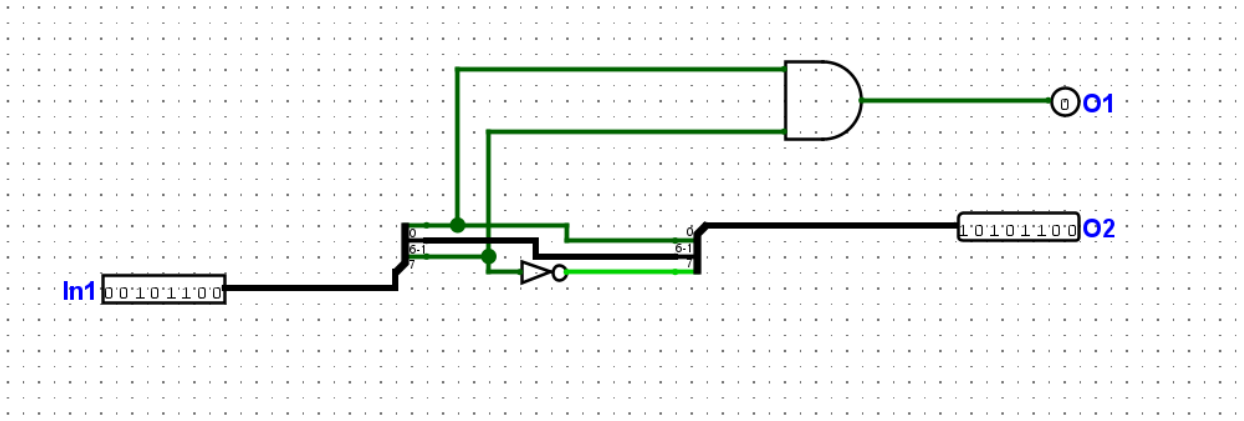
CNG 331

Term Project Part #1

Name: Oğuzhan Alpertürk

ID Number: 2315752

1.2.1



In the figure above, an 8-bit input pin comes to the splitter. The splitter splits the input value into 3 outputs. outputs:

1st output (1-bit) : 0th bit of the input

2nd output (6-bit) : 1-6th bit of the input

3rd output (1-bit) : 7th bit of the input

As requested in the manual, the 0th bit and 7th bit of the input are entered into an AND gate. The output of the AND gate connected to the 1-bit output pin, O1.

Then, I interpreted the input value as a "sign and magnitude" value. So, that means the 7th bit of the input is the sign bit and the 0-6th bits are magnitude bits. The 8-bit output pin O2 should be the negative "sign and magnitude" value of the input value. That means, I should invert the sign bit and the remaining bits (0-6th bit) should be the same. I use another splitter to bring the splitted bits together. I inverted and connected the 7th bit to the 7th bit of the other splitter. Then I connected the other bits directly to the other splitter. After that, I connected the new splitter to the 8-bit output pin O2. In the end, O2 is the negative "sign and magnitude" value of the Input.

After that, I use the poke tool to test the circuit.

I gave the "00101100" value as input.

O1 output is 0 because:

0th bit of the input : 0

7th bit of the input : 0

0 AND 0 = 0

O2 output is 10101100. The sign bit is inverted, and the magnitude part is the same as the input value.

1.2.1.1

The size of the input B should be 3-bit. The size of the input that will be rotated is 8-bits. The input places in the same position after the 7^{th} rotation. So, there is no point to rotate after 7 rotations. So, the maximum rotation number needed is 7. The bit size required for writing 7 in binary is 3. That's why the size of the input B should be 3 bits.

The logic behind the rotl and rotr circuits are nearly the same. I connected 8-bit input to the 8 splitters (8 Fan Out) to reach each bit. Then I connected each bit to another splitter. The connection place of each bit depends on the rotation amount. I used 8 splitters because using 3-bit input B, there will be 8 rotation conditions. I made all 8 rotation conditions and connected each them to a different input of the MUX. I connected input B to MUX as a select input. So, based on the rotation amount (input B), the multiplexer select the inputted (input B) rotation condition.

I used poke tool to try rotl and rotl. Both works properly. As you see in the figures below:

Rotr:

Input A: 11011010

Input B: 100

Output O1: 10101101

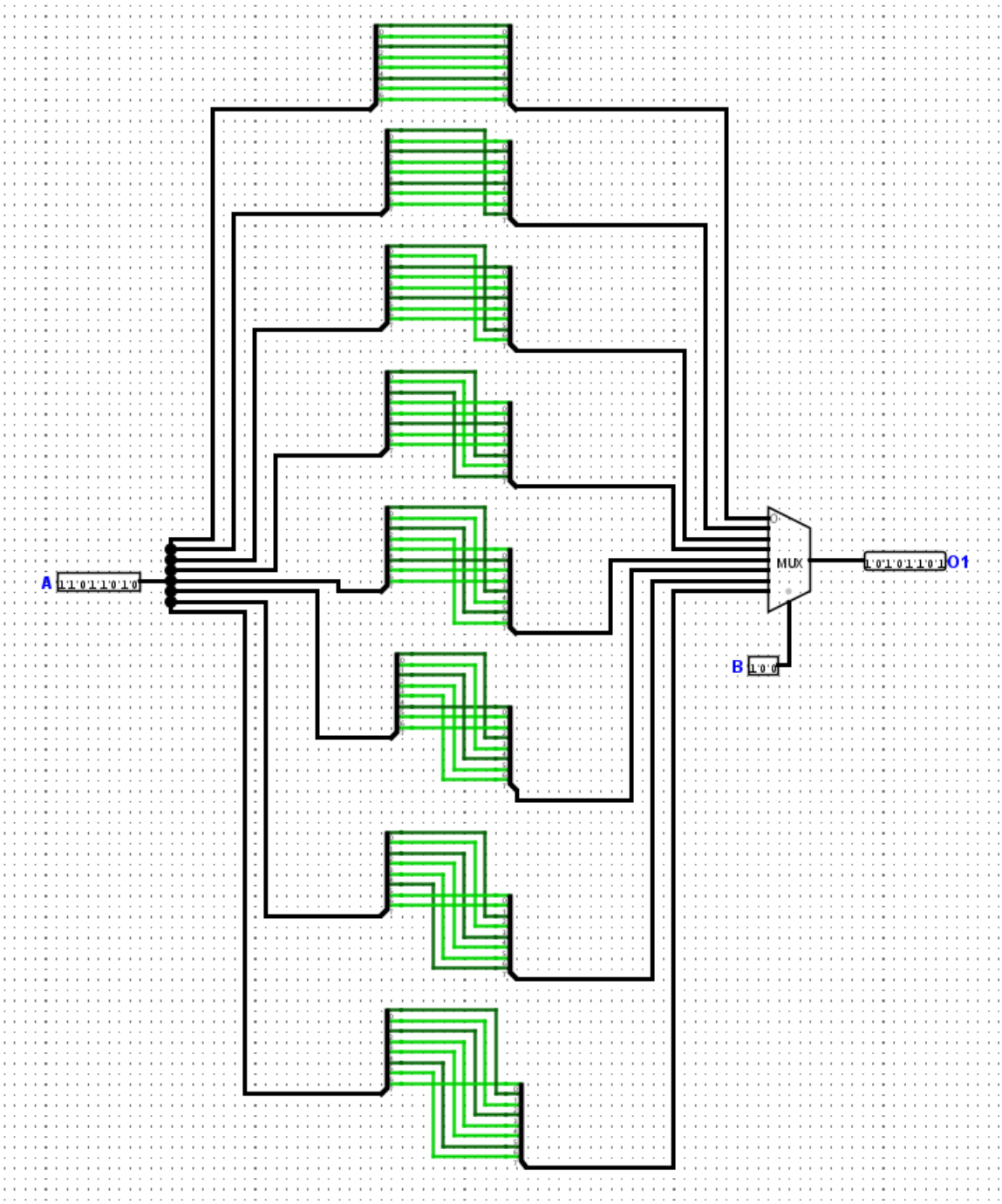
Rotl:

Input A: 00111001

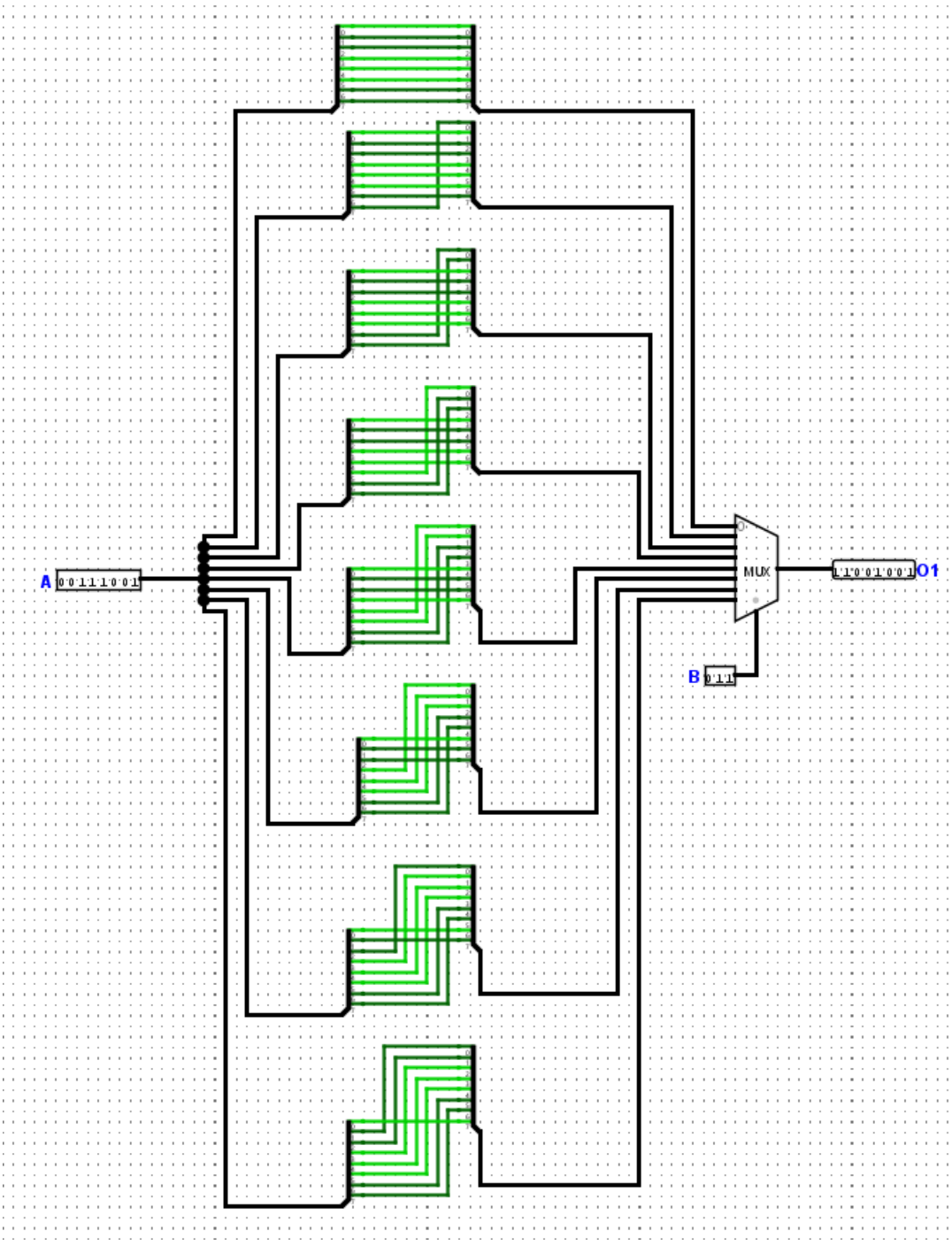
Input B: 011

Output O1: 11001001

Rotr:



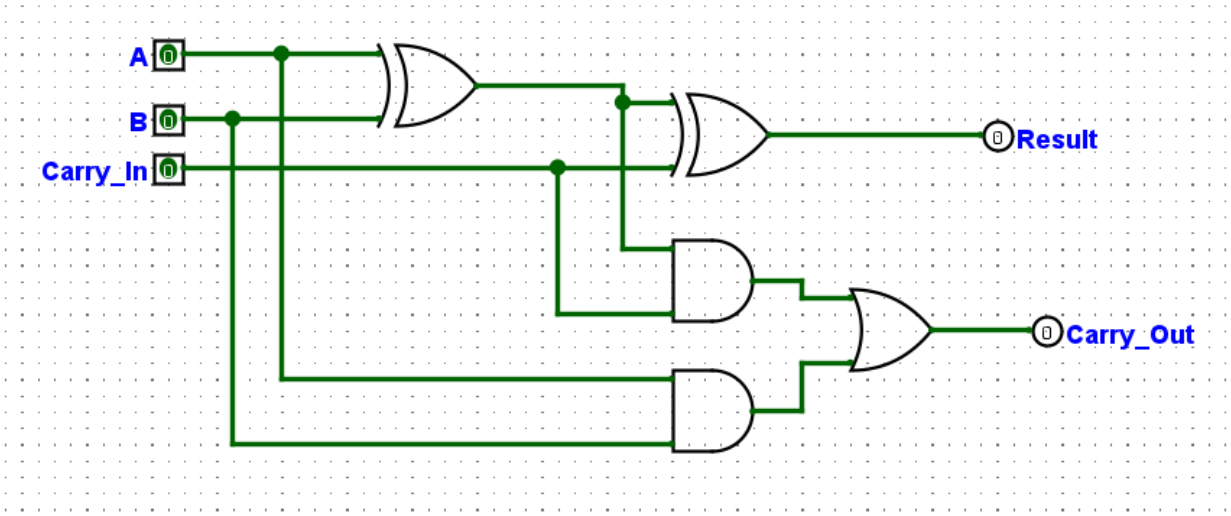
Rotl



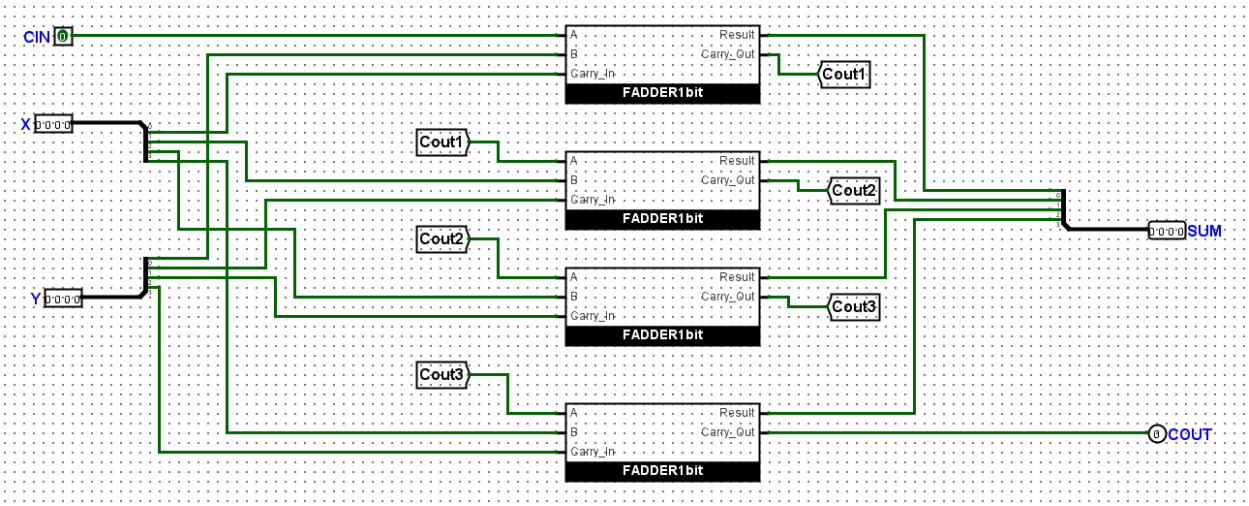
1.3.1

For building an 8-bit full adder, I built a 1-bit full adder first, and then I used 4 of them to built a 4-bit full adder. After that, I used two 4-bit full adders to build an 8-bit full adder.

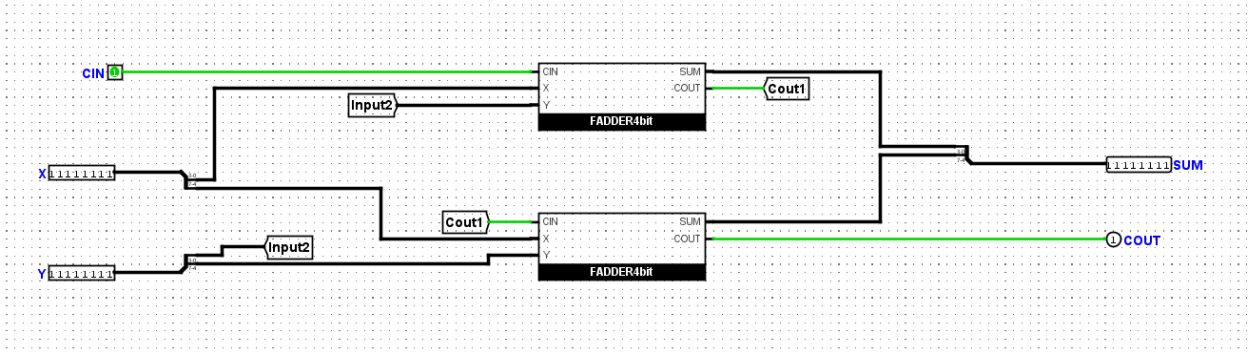
1-bit Full Adder:



4-bit Full Adder:

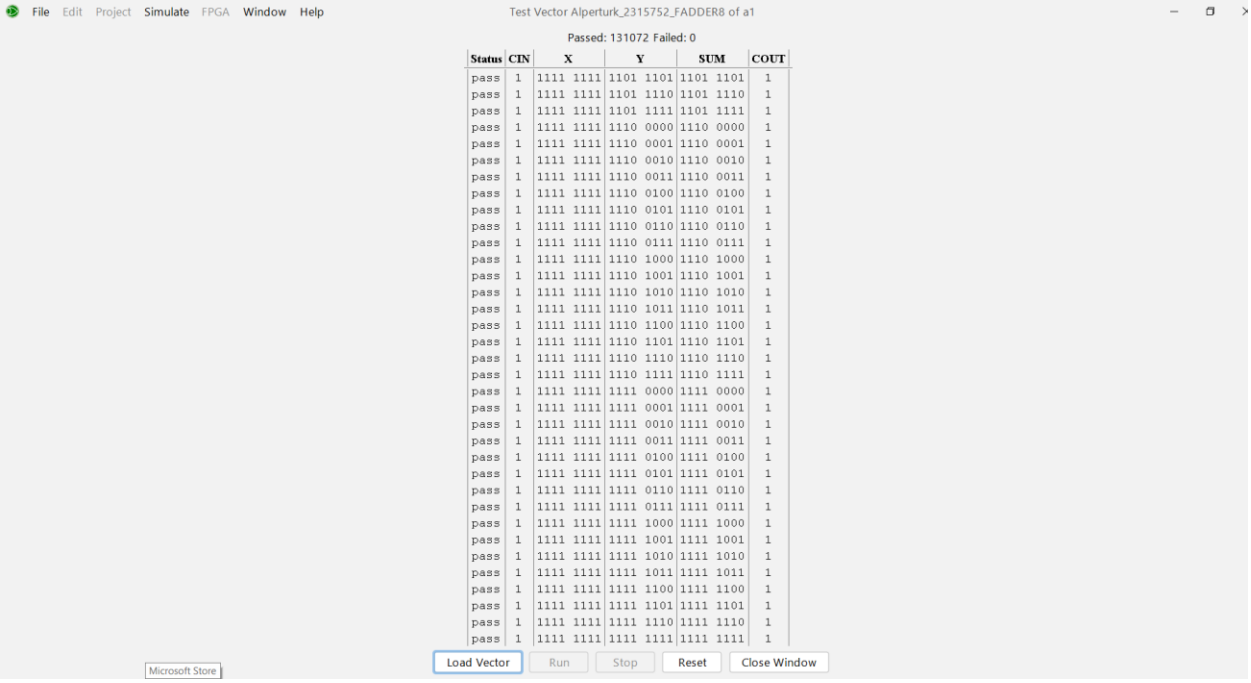


8-bit Full Adder:



Test Results of 8-bit Full Adder:

In this section, I built the same circuit with Logisim's 8-bit adder again. Then, I create a vector file with it. After that, I export the vector file of this circuit. Then, I simulate my original circuit with this vector file.

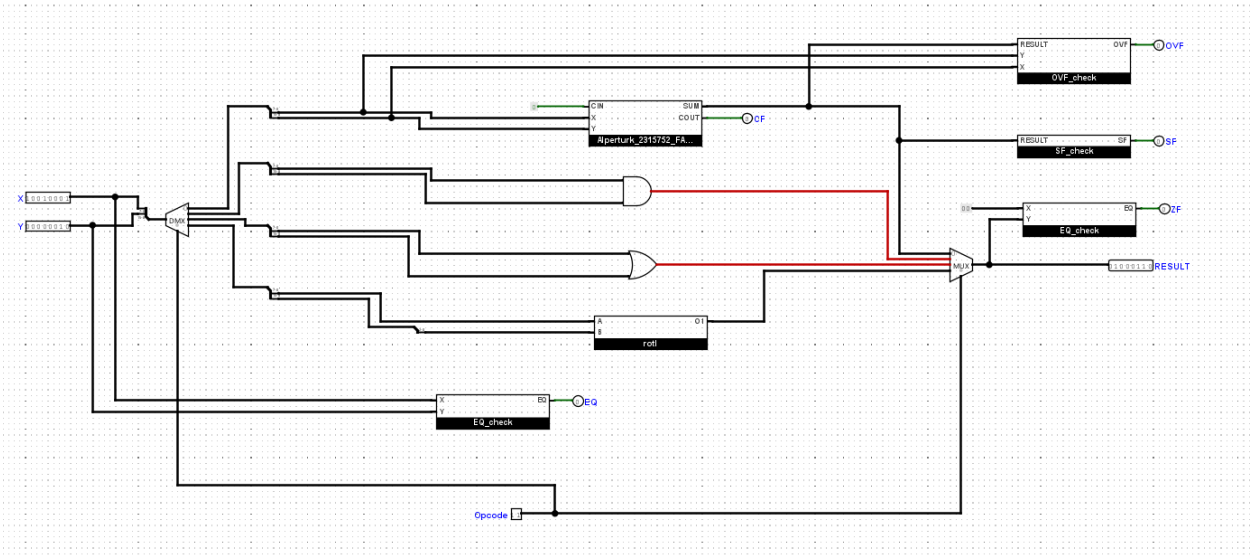


Status	CIN	X	Y	SUM	COUT
pass	1	1111 1111	1101 1101	1101 1101	1
pass	1	1111 1111	1101 1110	1101 1110	1
pass	1	1111 1111	1101 1111	1101 1111	1
pass	1	1111 1111	1110 0000	1110 0000	1
pass	1	1111 1111	1110 0001	1110 0001	1
pass	1	1111 1111	1110 0010	1110 0010	1
pass	1	1111 1111	1110 0011	1110 0011	1
pass	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

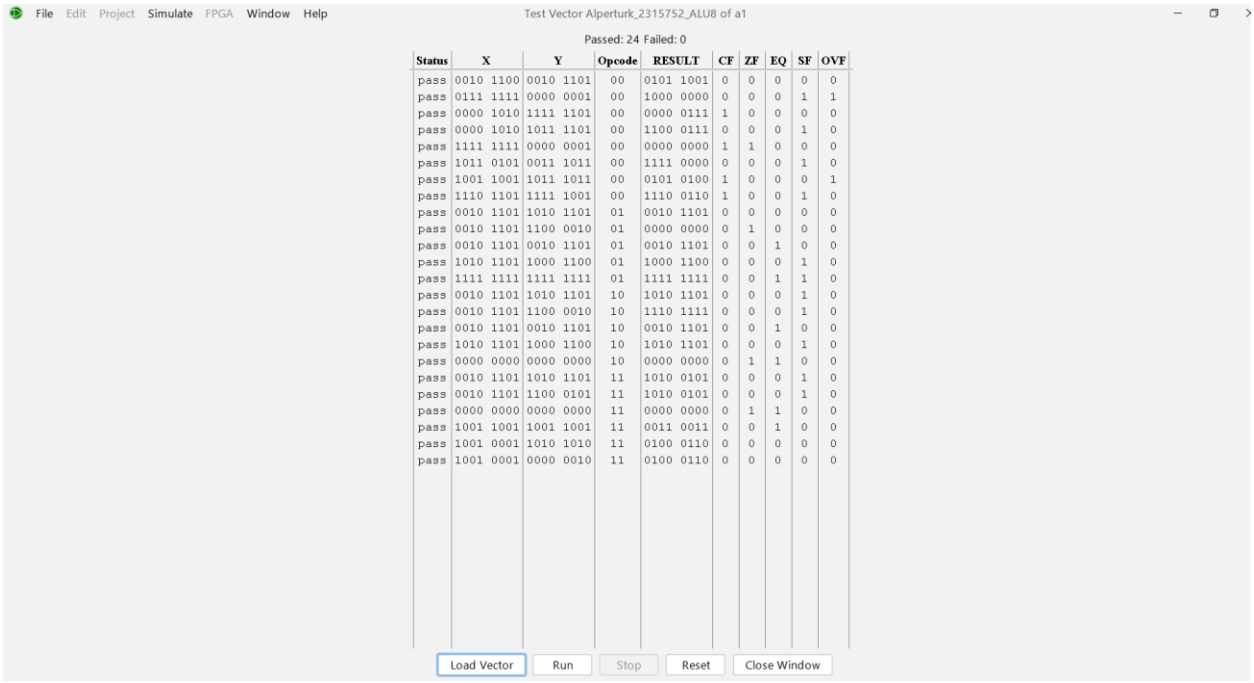
1.3.2

I used demultiplexer and multiplexer to execute operations separately. I connected Opcode to select parts of MUX and DMX. Then, I connected the EQ flag directly to the X and Y inputs to check their equality. For the addition part, I used 8bit full adder I designed. I connected CF to the COUT part of it. I connected OVF and SF output to the result of the 8bit full adder because they should only check the result after the addition operation. For AND and OR operation, I basically used the 8-bit input AND and OR gate. For the left rotation part, I used the rotl circuit I have already designed. I used X as an input and Y's rightmost 3 digits for the rotation amount. For ZF, I used the equality check circuit I have already designed. I connected the result and 0 to it for checking the equality of the result and 0.

ALU8:



ALU8 Test Results:



Status	X	Y	Opcode	RESULT	CF	ZF	EQ	SF	OVF
pass	0010 1100	0010 1101	00	0101 1001	0	0	0	0	0
pass	0111 1111	0000 0001	00	1000 0000	0	0	0	1	1
pass	0000 1010	1111 1101	00	0000 0111	1	0	0	0	0
pass	0000 1010	1011 1101	00	1100 0111	0	0	0	1	0
pass	1111 1111	0000 0001	00	0000 0000	1	1	0	0	0
pass	1011 0101	0011 1011	00	1111 0000	0	0	0	1	0
pass	1001 1001	1011 1011	00	0101 0100	1	0	0	0	1
pass	1110 1101	1111 1001	00	1110 0110	1	0	0	1	0
pass	0010 1101	1010 1101	01	0010 1101	0	0	0	0	0
pass	0010 1101	1100 0010	01	0000 0000	0	1	0	0	0
pass	0010 1101	0010 1101	01	0010 1101	0	0	1	0	0
pass	1010 1101	1000 1100	01	1000 1100	0	0	0	1	0
pass	1111 1111	1111 1111	01	1111 1111	0	0	1	1	0
pass	0010 1101	1010 1101	10	1010 1101	0	0	0	1	0
pass	0010 1101	1100 0010	10	1110 1111	0	0	0	1	0
pass	0010 1101	0010 1101	10	0010 1101	0	0	1	0	0
pass	1010 1101	1000 1100	10	1010 1101	0	0	0	1	0
pass	0000 0000	0000 0000	10	0000 0000	0	1	1	0	0
pass	0010 1101	1010 1101	11	1010 0101	0	0	0	1	0
pass	0010 1101	1100 0101	11	1010 0101	0	0	0	1	0
pass	0000 0000	0000 0000	11	0000 0000	0	1	1	0	0
pass	1001 1001	1001 1001	11	0011 0011	0	0	1	0	0
pass	1001 0001	1010 1010	11	0100 0110	0	0	0	0	0
pass	1001 0001	0000 0010	11	0100 0110	0	0	0	0	0