KUZEY KIBRIS KAMPÜSÜ
ODTÜ
METU
NORTHERN CYPRUS CAMPUS

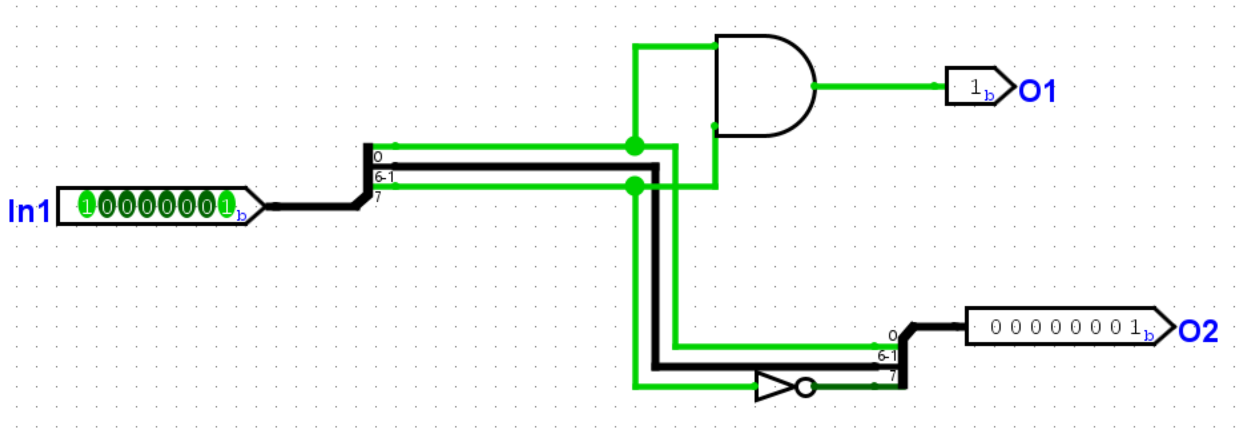
CNG 331 Project 1 Report

Name: Muhammad Somaan
ID: 2528404

1.2.1.0)

Sign and Magnitude Inverter:

Schematic:

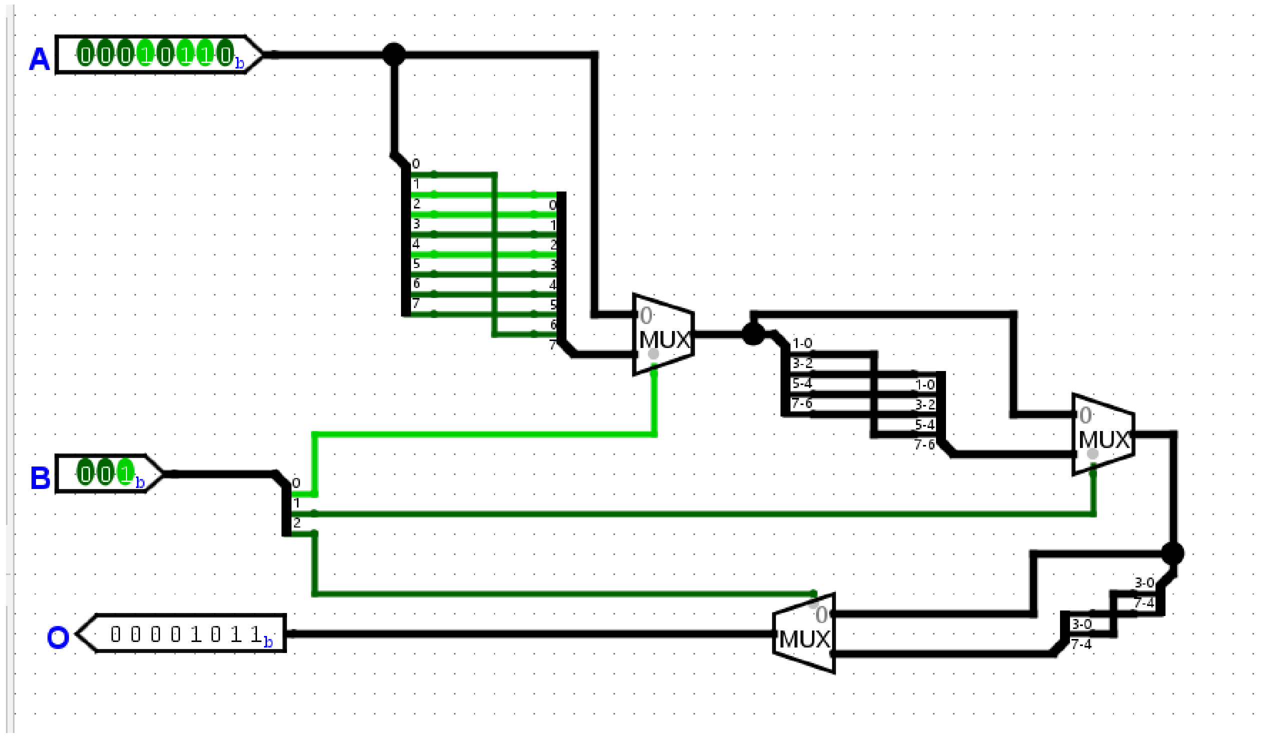


Comments: In1 is an eight-bit input, whereas O1 is one-bit output and O2 is eight-bit output. O2 is the resulting output after inverting the sign for a sign-magnitude input in In1. O1 is basically checking if both the most significant bit and the least significant bit are 1, resulting in a check for -1 in denary. Between In1 and O2 splitters are used so we can invert the MSB for O2 and use LSB for O1.
Test Value: For testing as can be seen in the schematic, I chose 10000001 as In1 which results in O1 being 1 because both MSB and LSB are 1, and in O2 the output is 00000001 so we can see that the MSB is inverted
.

1.2.1.1)

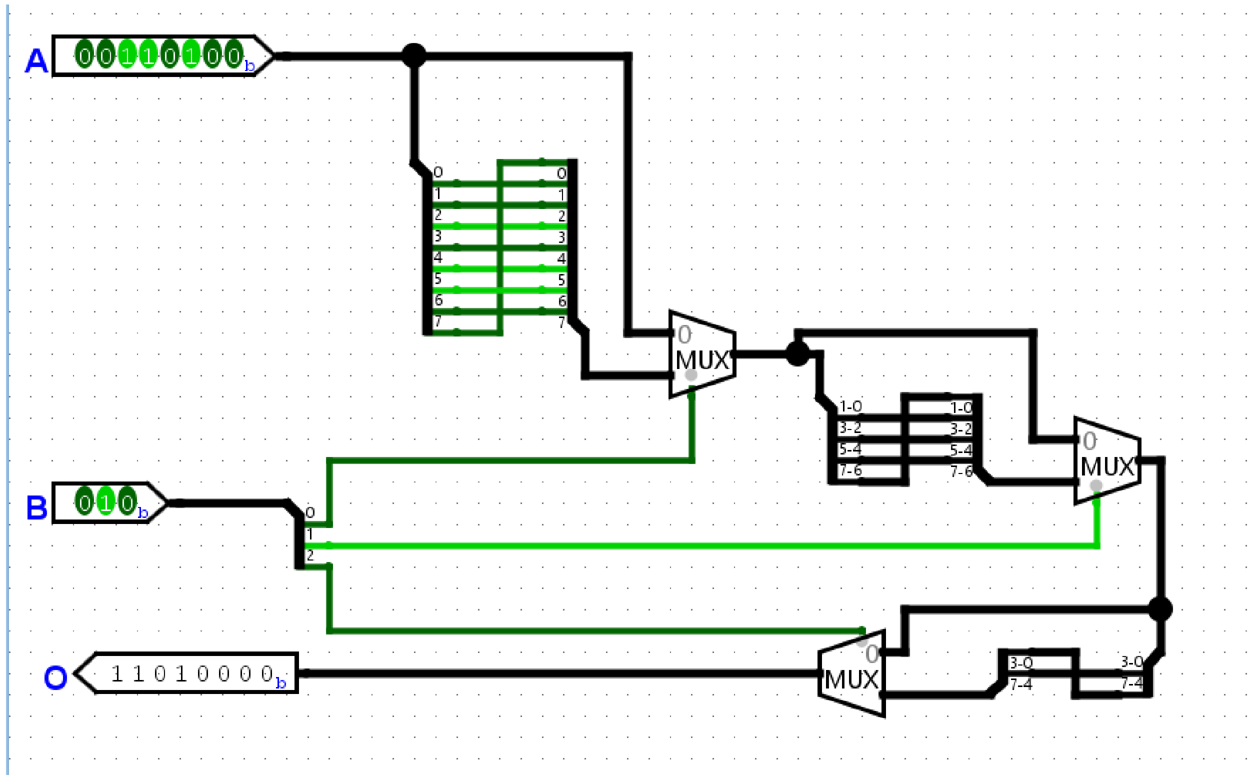
Rotate Right:

Schematic:



Rotate Left:

Schematic:



Comments: We have one 8-bit input A which is the main input. Then we have a 3-bit input B which stores the amount of rotation to apply on input A. The size of B is 3 bits because the max 3 bits can hold is 7 in denary, and in an 8-bit input, the max rotation needed will be 7 (if we do a rotation of 8, it returns to its original state).

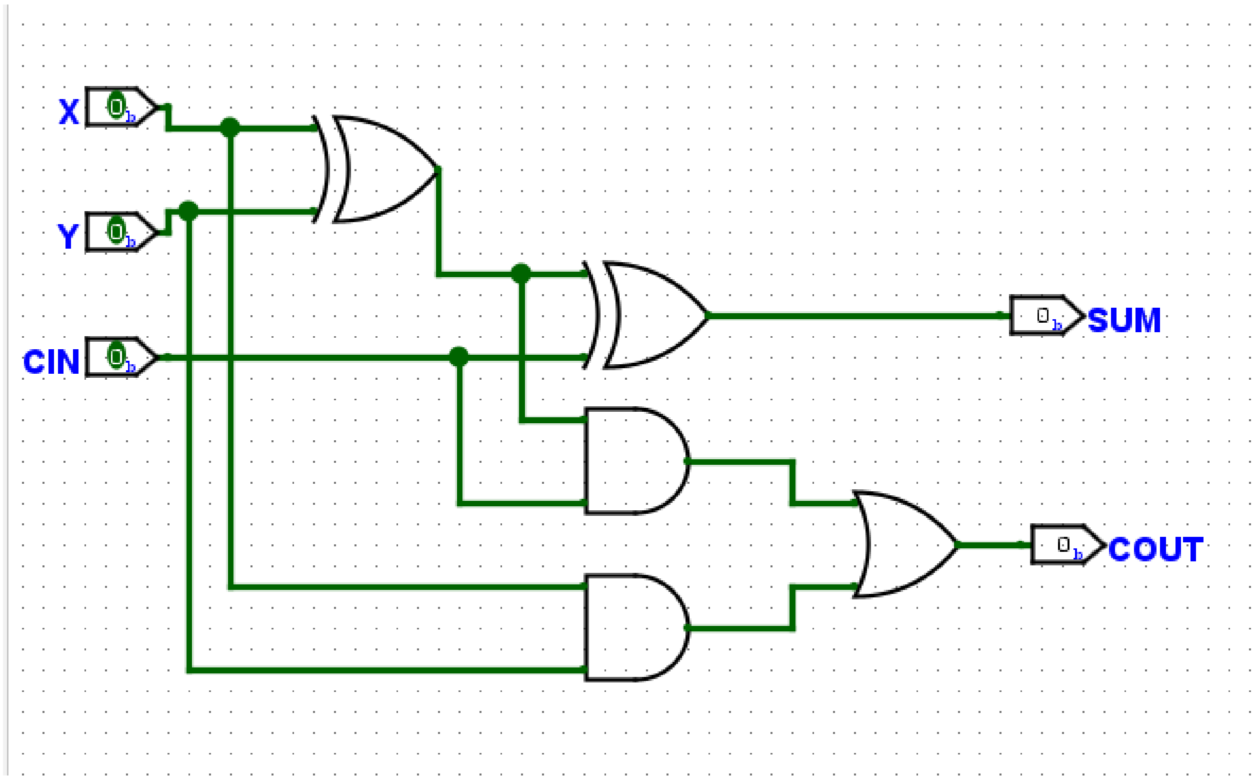
To implement the rotation I split the input A into using 1-bit fan out, and then connect each bit changing the order by 1 to another splitter(8-bit with 1-bit fan out), for example, the bit 0 of input connects to bit 1 and bit 1 of input connects to bit 2 and so on, in case of rotate left. The process is similar for rotate right, instead of connecting bit 0 to bit 1, it is connected to bit 7, and then bit 1 is connected to bit 0. The output of this operation goes into a mux, which uses the LSB of input B as select, if it is 0 A input is given, if it is 1 then the rotated output is given.

To further expand on this the result of the mux is then rotated again but now 2 bits at a time, and this time the mux's select is bit-0 of input B, continuing forward, the output is rotated 4 bits at a time, and now the mux's select is the MSB of input B, then the final output of this mux provides the Output O, which correctly rotates the input A according to the value provided in input B.
A test value can be seen in the schematic, where Input A was rotated right by 1-bit according to rotate right's input B. For the rotate left, we can see a rotation of 2, represented by input B.

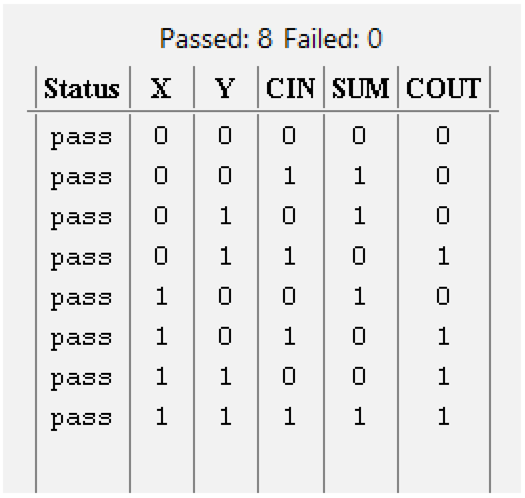
1.3.1)

1-bit Full Adder:

Schematic:



Test:

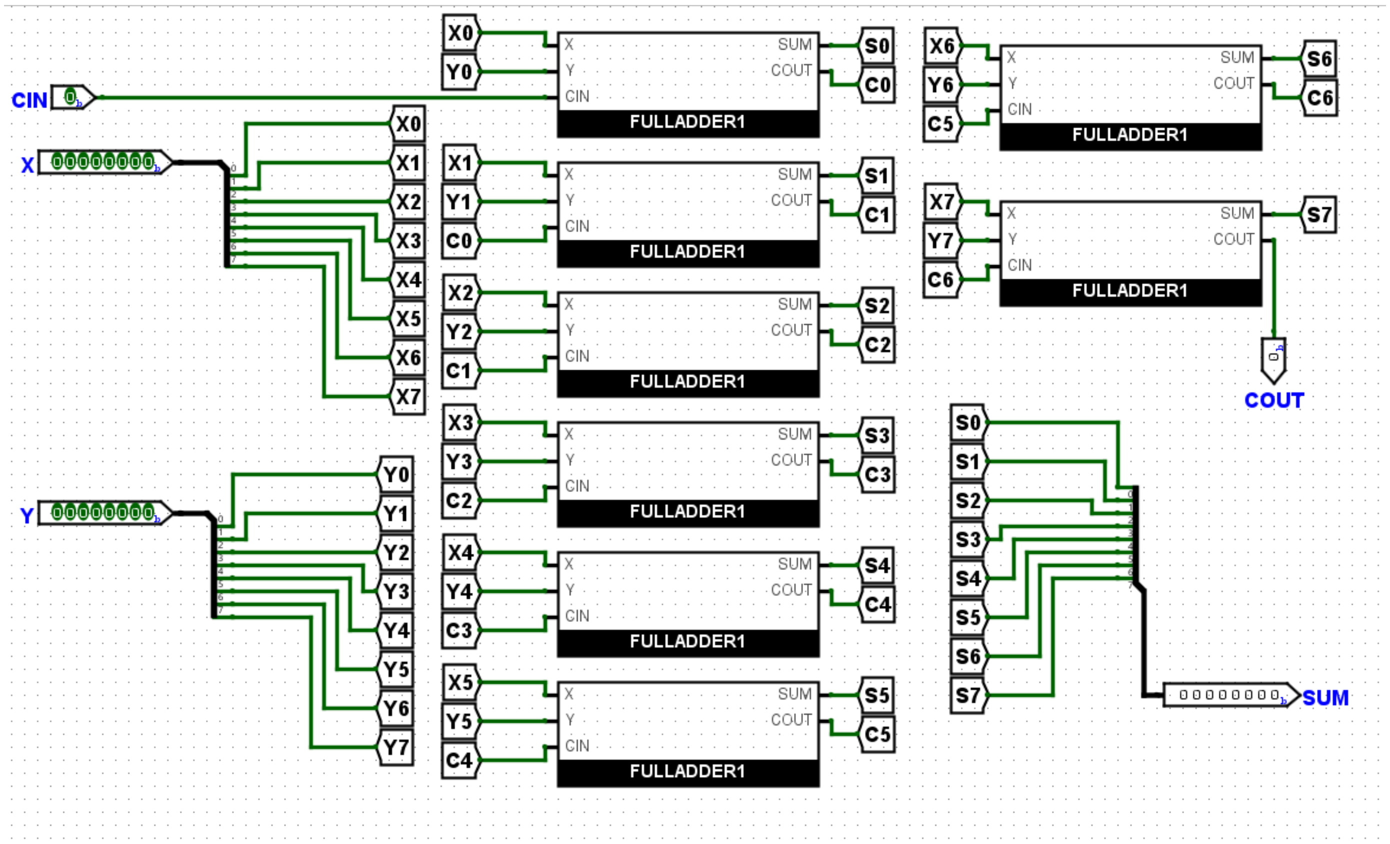


Passed: 8 Failed: 0

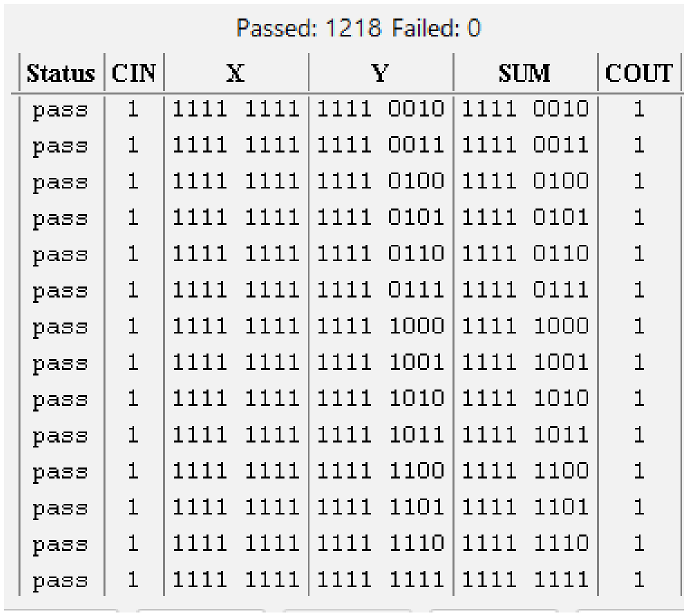
Status	X	Y	CIN	SUM	COUT
pass	0	0	0	0	0
pass	0	0	1	1	0
pass	0	1	0	1	0
pass	0	1	1	0	1
pass	1	0	0	1	0
pass	1	0	1	0	1
pass	1	1	0	0	1
pass	1	1	1	1	1

8-bit Full Adder:

Schematic:



Test:



Passed: 1218 Failed: 0

Status	CIN	X	Y	SUM	COUT
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

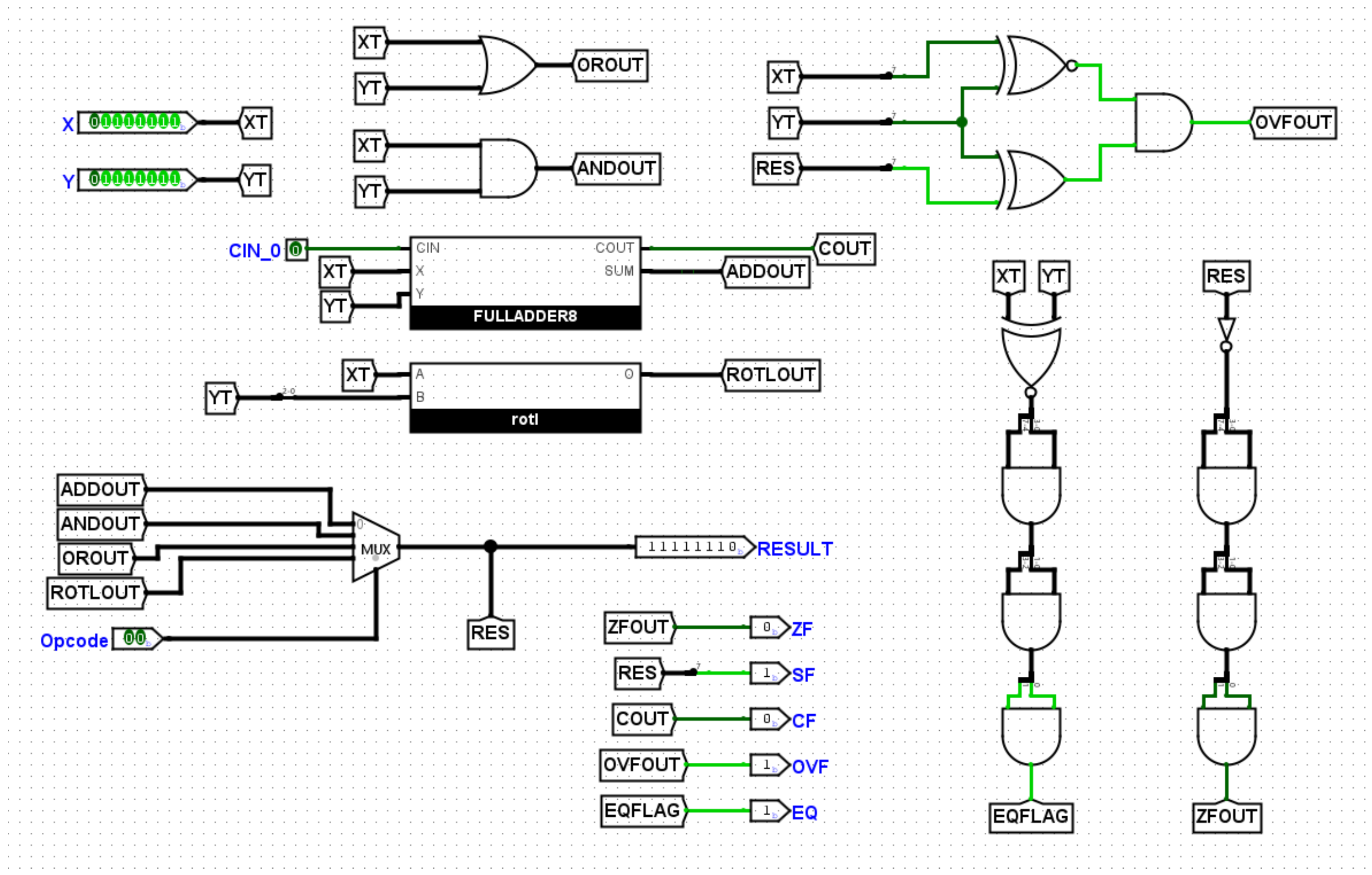
Comments: For 1-bit full adder, it is designed according to the method explained in the tutorial examples. All the inputs and outputs are 1-bit. CIN is the carry-in input, and COUT is the carry-out output, in case of overflow.

Comments: For 8-bit full adder, we have 2 8-bit inputs X and Y, and 1 1-bit input CIN. We have 8-bit output SUM and 1-bit output COUT. The circuit is implemented using the 1-bit full adder designed. The 1-bit full adder is used to build a ripple carry adder. We have 8 1-bit full adders, that perform 1 bit add operation for each bits of inputs X and Y and then stored in the corresponding bit of output SUM, for example, bit 0 of X is added to bit 0 of Y and the result is stored in bit 0 of SUM out. When adding bit-0, the input CIN is provided to the CIN of 1-bit full adder, but after that the COUT of that adder is input to the next adder's CIN. Continuing this behaviour the last 1-bit full adder's COUT is directly output as the output COUT for the 8-bit full adder.
In my design I have used splitters to split the bit's of input X and Y and used tunnels to avoid messy wires and similarly used a splitter to combine all the bits for SUM output.

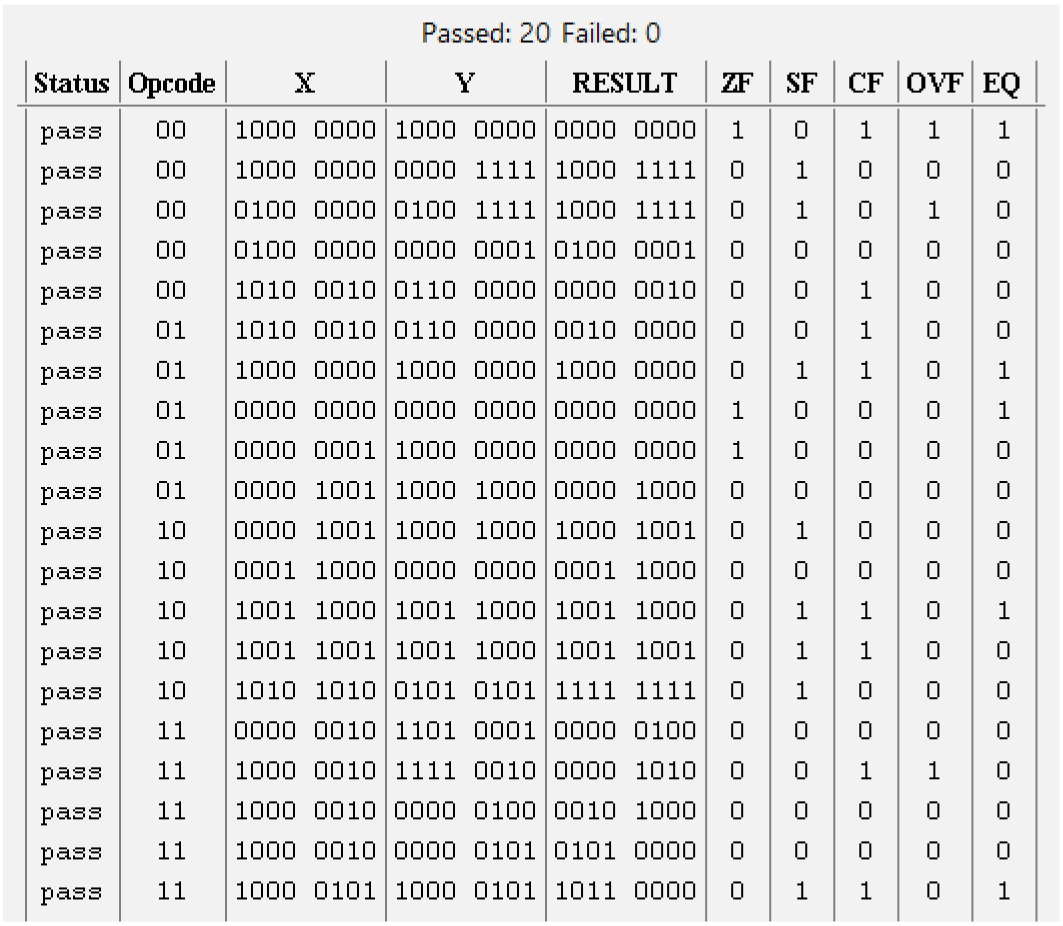
1.3.2)

8-bit ALU:

Schematic:



Test:



Passed: 20 Failed: 0

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

Comments:
In this circuit, I'm using Tunnels to avoid messy wires. The tunnels are as follows: XT is X input. YT is Y Input. OROUT is the output of the Or function. ANDOUT is the output of And function. ADDOUT is the output of Adder output. ROTLOUT is the output of the rotate left function. COUT is the COUT output of the adder. OVFOUT is output for calculation of signed bit overflow output OVF. ZFOUT is output for the calculation of zero flag output ZF. EQFLAG is output for calculation if input X is equal to Y input function output EQ. RES is the output RESULT.
CIN_0 is a constant 0 for input to CIN for the adder.
For And function calculation I just used an 8-bit and gate between X and Y.
For Or function calculation I just used an 8-bit or gate between X and Y.
For Add function, I am using the 8-bit adder designed before.

For Rotate left function I am using the rotl circuit designed before, for B input to rotl, I use a splitter on Y input to make sure to get the least 3 significant bits.
For zero flag calculation, I take the result and invert it, making all the 0s 1s. Then I split the 8-bit into 2 4-bits and use an and gate on them, then again repeating the process by converting the 4-bit output to two 2-bits and then using an and gate on them until I have a single-bit output, which will be the zero-flag. The reasoning behind this is that after inverting the result if the result is 0, it will become all 1s, and then we just need to use and gate between the 8 bits to know if the result is zero or not.
For signed bit calculation, output SF is just the MSB of the result output.
For unsigned addition overflow output CF, it is just the COUT, which comes from the adder.
For signed addition overflow output OVF, we need to only consider the MSB of X, Y, and Result. Then by checking if they are either 0 0 1 or 1 1 0, I created a logic for it, which is (Y XOR Result) AND (X XNOR Y).
For checking if X and Y are equal for output EQ, I used an XNOR between X and Y, to get a 1 if both X Y were either 0 0 or 1 1. Then as I did for zero flag calculation, I just used and gate between all the bits, to confirm if they were equal.

Test Cases for Opcode 00: I made sure to include all tests that check ZF, SF, CF, and OVF for the addition function. I also include a case in which none of the flags are 1.
Test Cases for Opcode 01: I just used different values of X and Y to obtain the result of and gate between them.
Test Cases for Opcode 10: I just used different values of X and Y to obtain the result of or gate between them.
Test Cases for Opcode 11: I just used different values of Y mainly to show how much it rotates in each case, and also included cases where the first 5 bits also included 1s to show that those values don't matter and that only the least 3 significant bits are used for the rotation amount.