CNG 331 COMPUTER ORGANIZATION TERM PROJECT PART I

BUILDING AN 8-BIT ALU

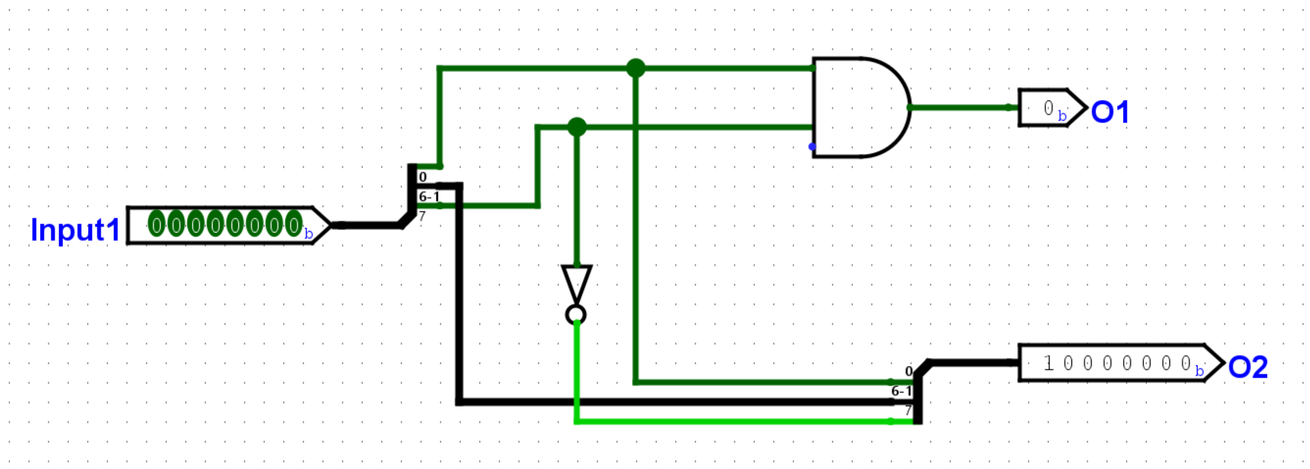
REPORT

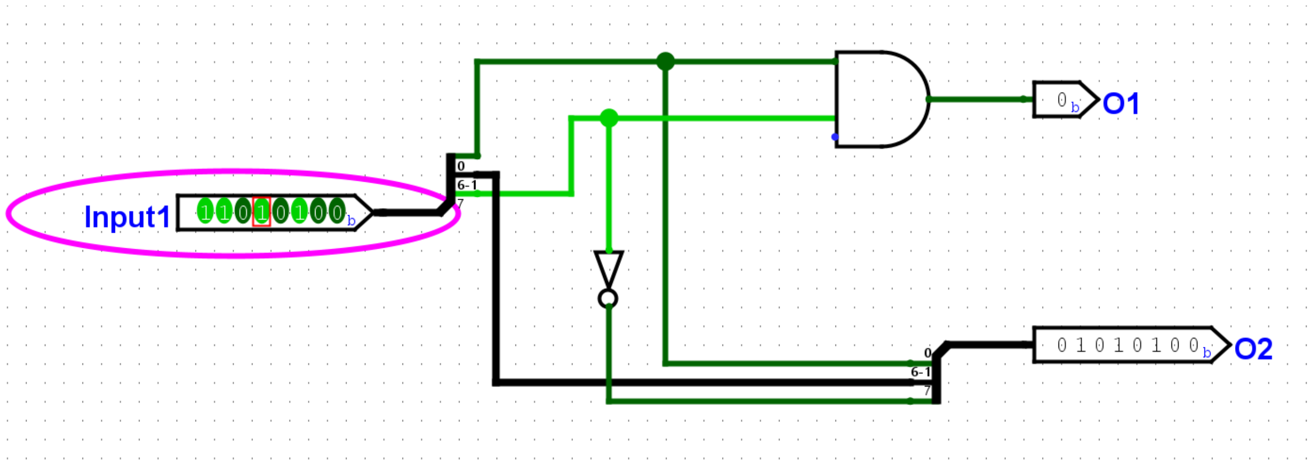
Fatma Erem Aksoy - 2315075

1.2 ADVANCED LOGISIM FEATURES

1.2.1 Splitters

11.





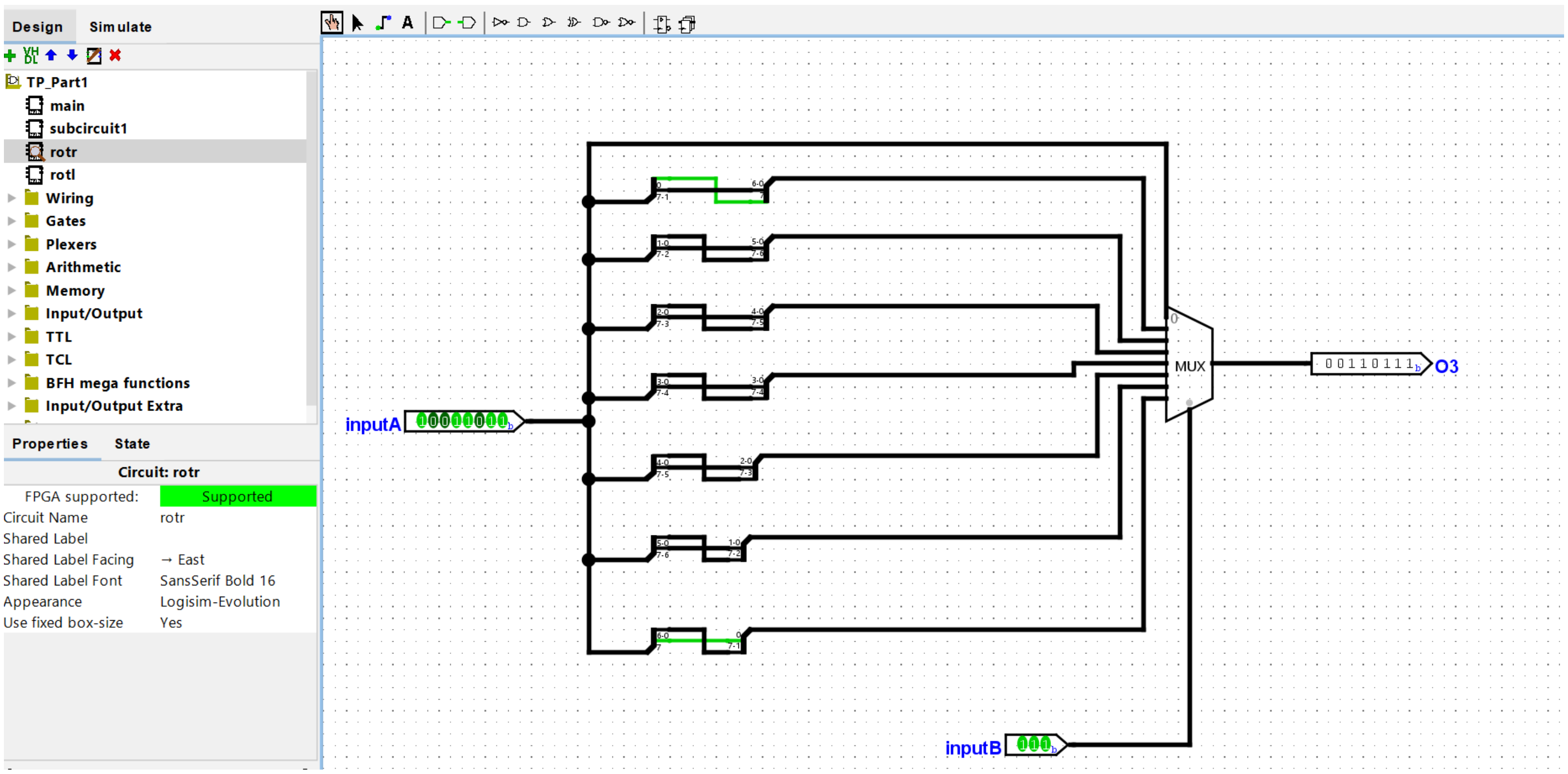
Two splitters are used to first split the 8-bit input and then recombine them for the O2.

- The least and most significant bits are combined with an AND gate for the O1.
- We use a NOT gate for the most significant bit and also take the rest of the input, then combine them with the second splitter to show the 2's complement (negative) version of the entered input. The test input and its output can be seen from the figure after the simulation starts (poke tool).

1.2.1.1

The size for the input B should be 3-bit because there are 8 possible options for a rotate operation. After every 8th rotate, the output will be the entered value again, so as we have 8 options to pick from, input B needs to be 3-bit so that it can satisfy the required condition. And we also need to use a MUX to pick the output. Based on the value entered for input B, the MUX will decide which rotation state to go and give the output.

Subcircuit "rotr":

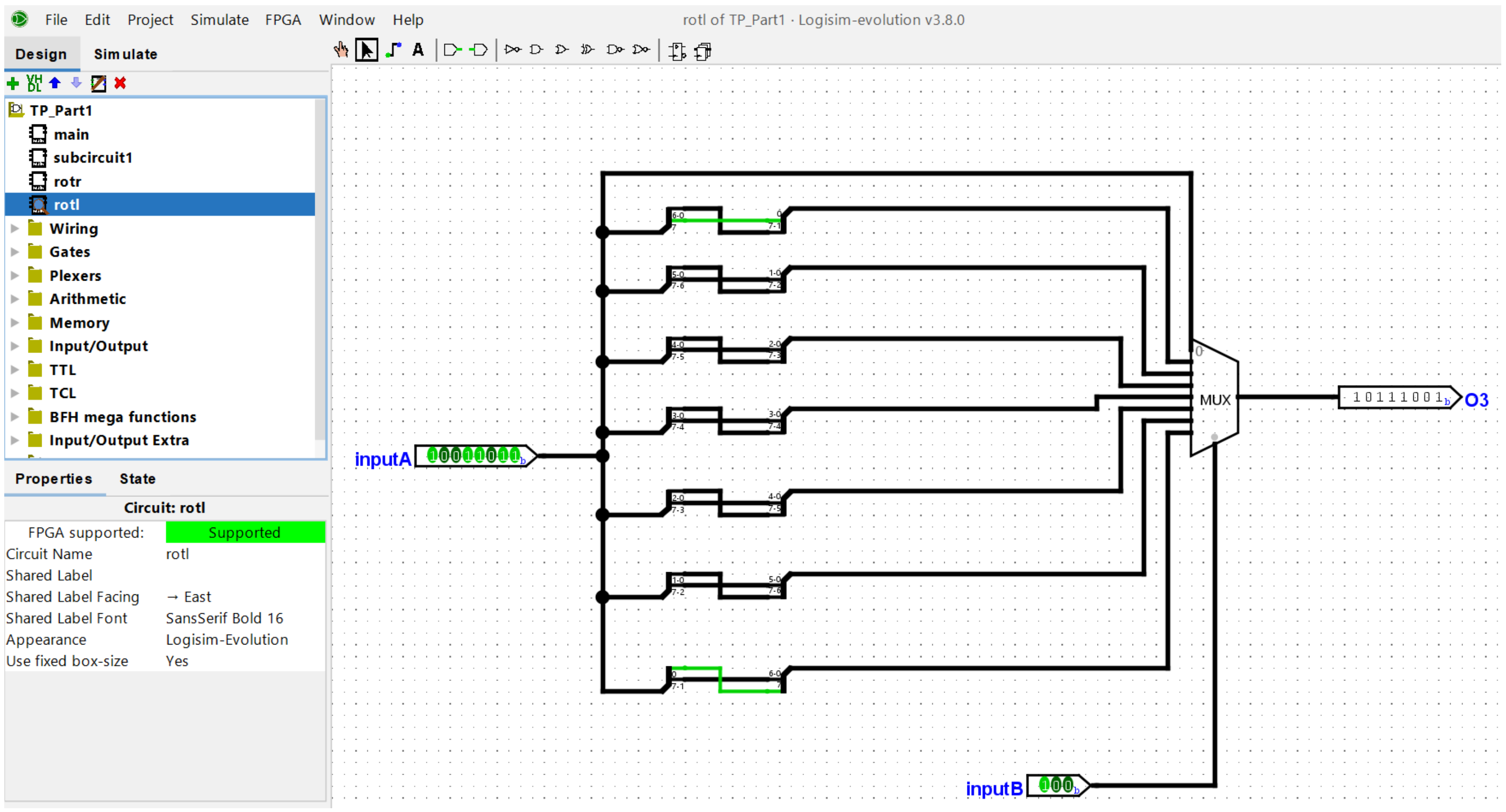


Circuit: rotr

FPGA supported:	Supported
Circuit Name	rotr
Shared Label	
Shared Label Facing	→ East
Shared Label Font	SansSerif Bold 16
Appearance	Logisim-Evolution
Use fixed box-size	Yes

The circuit above shows how to operate the rotation to right. For the first case, when input B is 000, the output will be the entered input A itself, which is 10011011 in this scenario. For the rest of the cases, the bits need to be shifted to right one by one in each iteration. Starting from 1 bit, the figure shows how to rotate larger number of bits every time and get the output based on the input B value. It can be seen from the figure that the largest value that input B can take is 111 and it will give the output 00110111. For the other values, the output will be generated accordingly.

Subcircuit “rotl”:



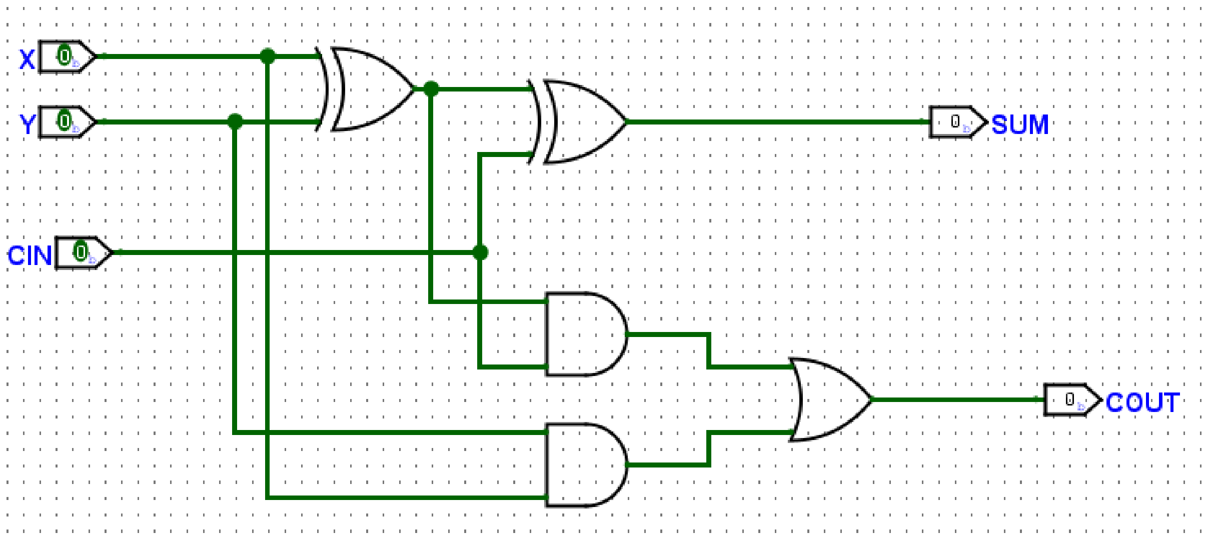
Circuit: rotl

FPGA supported:	Supported
Circuit Name	rotl
Shared Label	
Shared Label Facing	→ East
Shared Label Font	SansSerif Bold 16
Appearance	Logisim-Evolution
Use fixed box-size	Yes

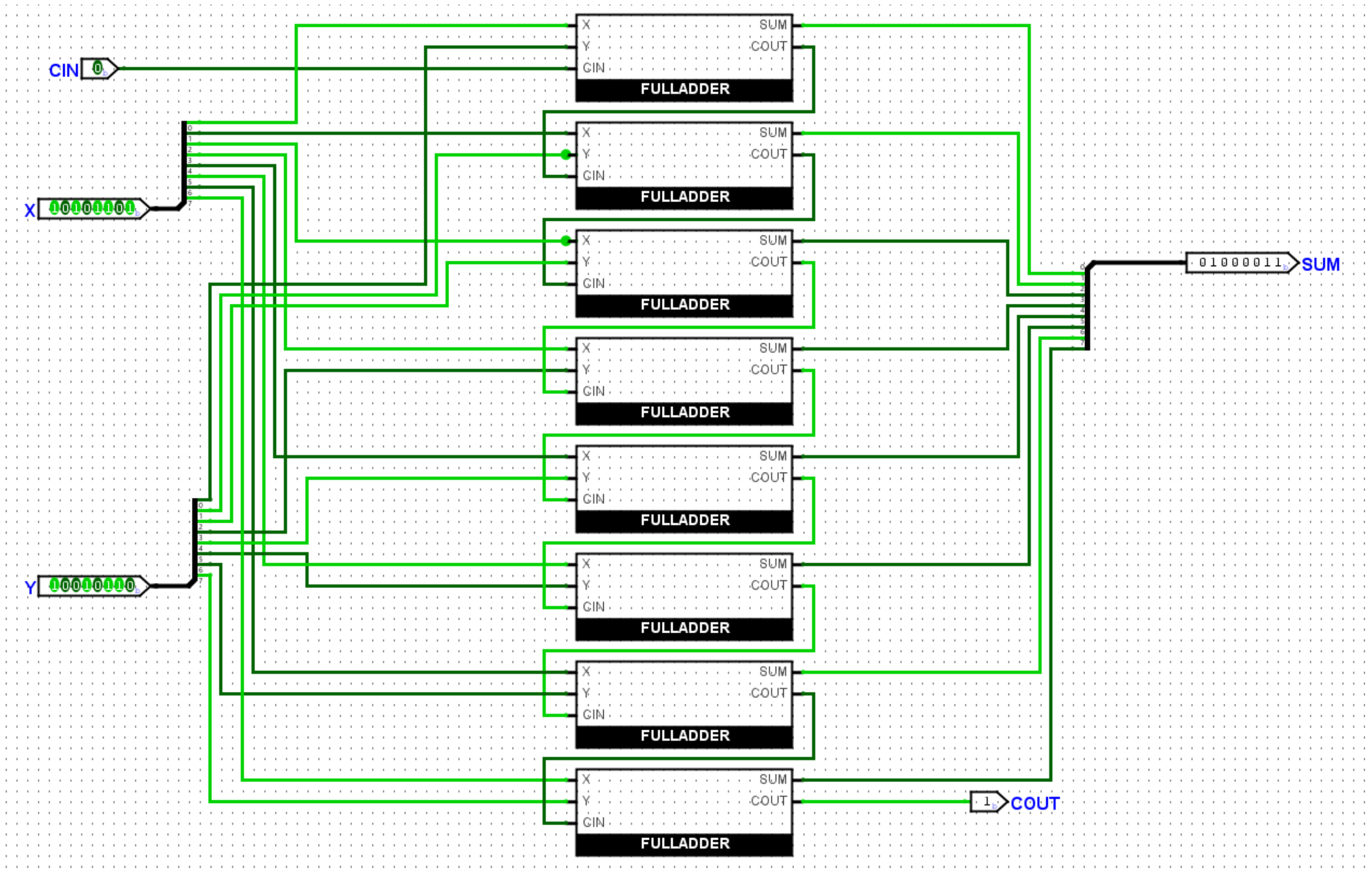
The circuit above shows how to operate the rotation to left. For the first case, when input B is 000, the output will be the entered input A itself, which is 10011011 in this scenario. For the rest of the cases, the bits need to be shifted to left one by one in each iteration. Starting from 1 bit, the figure shows how to rotate larger number of bits every time and get the output based on the input B value. It can be seen from the figure that when the input B is 100, MUX is supposed to pick the 5th option as the input is requested to rotate 4 bits, so the output is 10111001 in this case. For the other values, the output will be generated accordingly as well.

1.3 BUILDING AN ALU

1.3.1

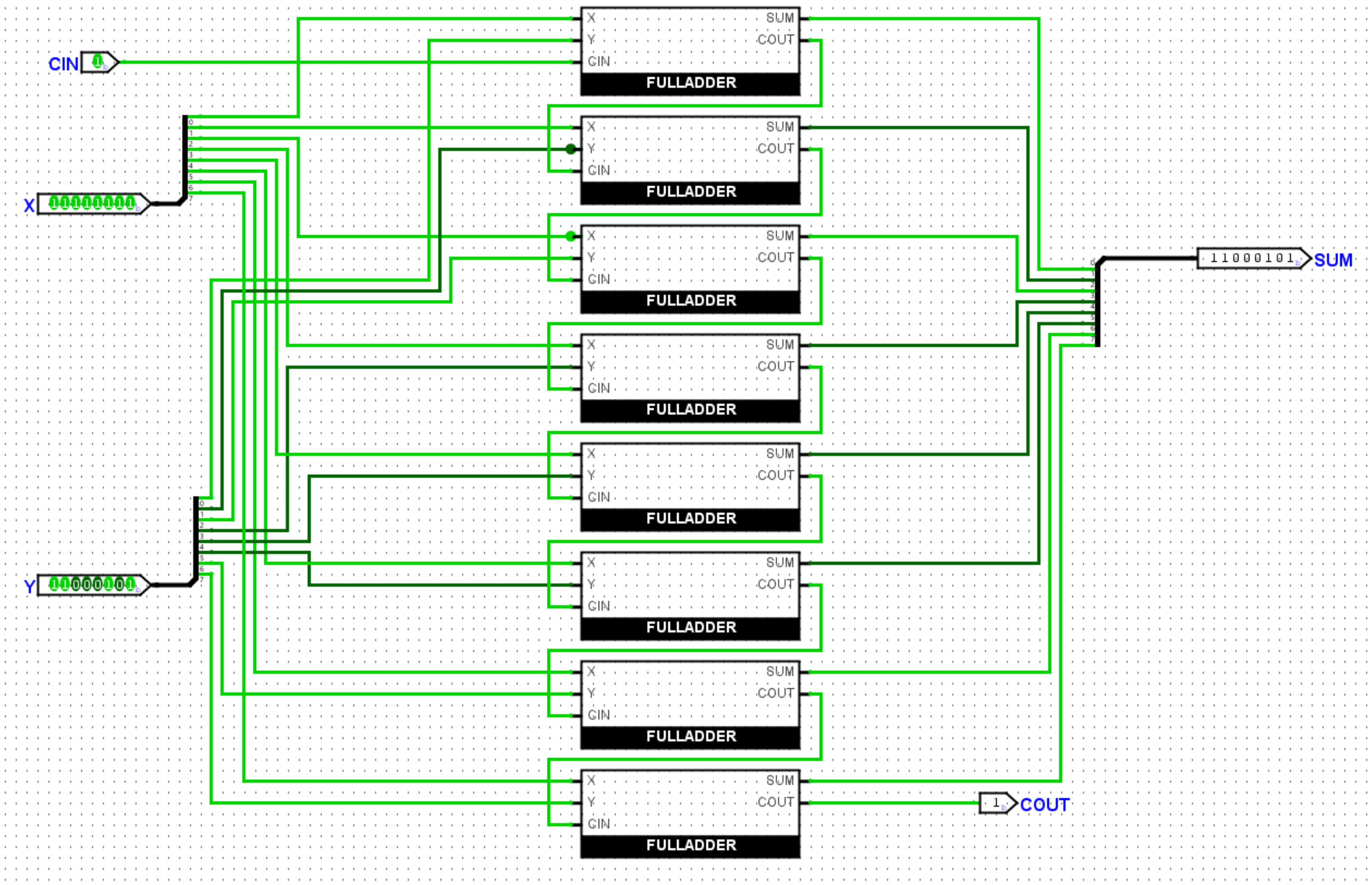


The figure above shows the implementation of 1-bit full adder and it will be used to implement 8-bit full adder.

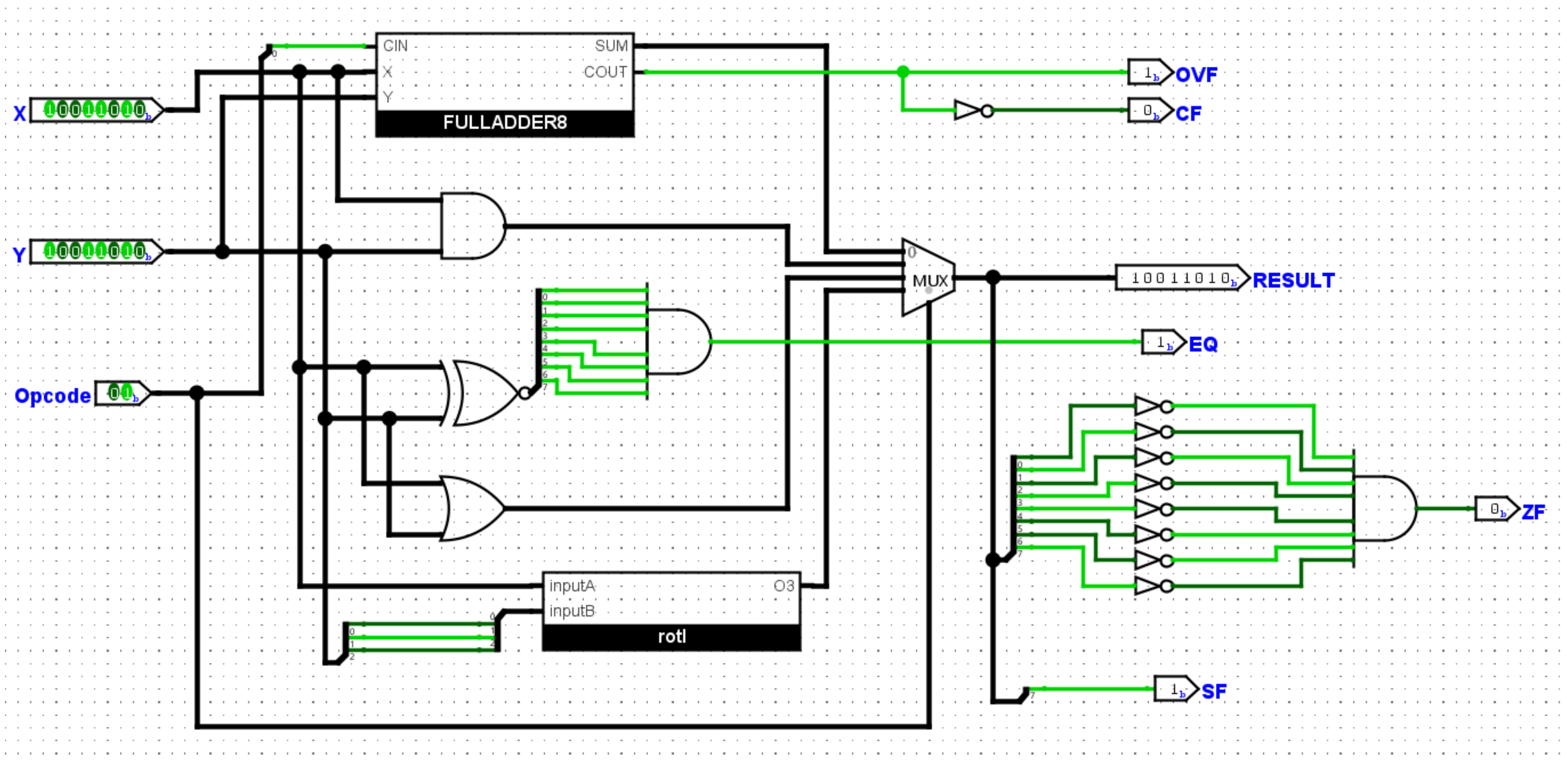


The 8-bit full adder is created by using the 1-bit full adders. By using the test vector, the outputs can be tested and seen that the implementation is correct. For example, in the figure above, when CIN=0 and the 8-bit inputs are X=10101101 and Y=10010110, the expected result is SUM=01000011 and COUT=1 as seen in the figure as well.

As another example of the case when CIN=1 and the 8-bit inputs are X= 11111111 and Y=11000101, the expected result is SUM=11000101 and COUT=1 as also seen in the figure below. The outputs values for both and the rest of the cases can be checked from the test vector file as well.



1.3.2



The figure above shows the implementation of an 8-bit ALU design. In this design, 8-bit full adder and rotate left circuits that implemented in the previous sections are used as sub-circuits.

A MUX is used to perform one of the four functions based on the opcode entered. When opcode=00, an addition operation will be performed, so 8-bit full adder is used for this function. And as OVF and CF outputs also depend on this operation, they'll be generated accordingly, OVF=1 when there is a signed overflow and CF=1 when there is an unsigned overflow.

When opcode=01, an AND operation will be performed between two 8-bit inputs entered.

When opcode=10, an OR operation will be performed between two 8-bit inputs.

When opcode=11, input X will be rotated based on the least 3 significant bits of input Y. The circuit for left rotation that implemented in the previous session is used here, and the input B for that circuit will be generated by taking the 3 least significant digits of input Y. So the input X will be rotated to left by the amount of those 3-bits of value.

Output EQ will be handled by using XNOR and AND gates with a splitter. Two 8-bit inputs, X and Y, will be compared by using the XNOR gate, so the output will be 1 whenever X and Y are equal to each other. After comparing them, a splitter is be used to split the bits of the output of the

XNOR gate and sent those bits to an AND gate separately. So if any of the bits is not equal and gives 0, AND gate will detect it and the result will be 0, meaning that the inputs are not equal.

Output ZF will be calculated by using NOT and AND gates, or NAND preferably. The logic here is if all the bits of the result is 0, then we toggle them and make them 1, and send all the bits to an AND gate to make the output of this operation 1, and we assign this to the output ZF to show that the result is 0.

Output SF will be generated by checking the most significant bit of the result. So if the most significant bit is 1, then SF will be 1 as well as it shows that the number is negative. SF will be 0 when the most significant bit of the result is 0, meaning that the result is a positive number.