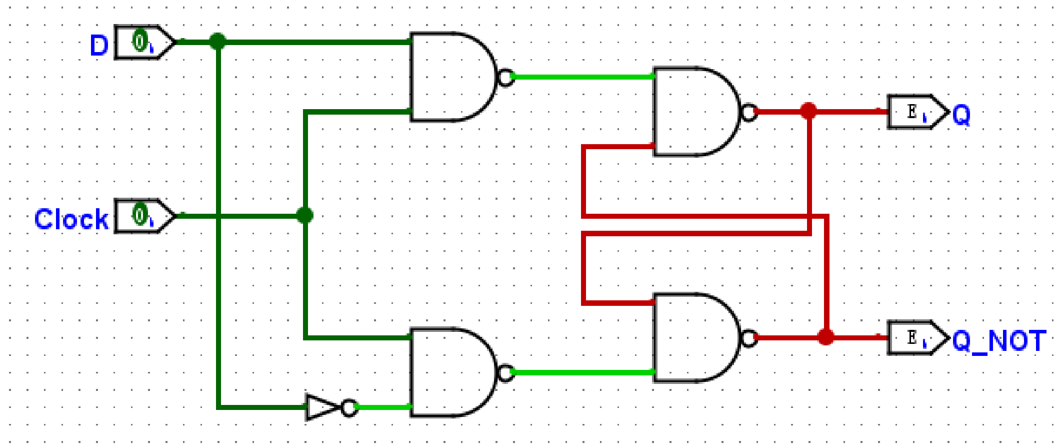
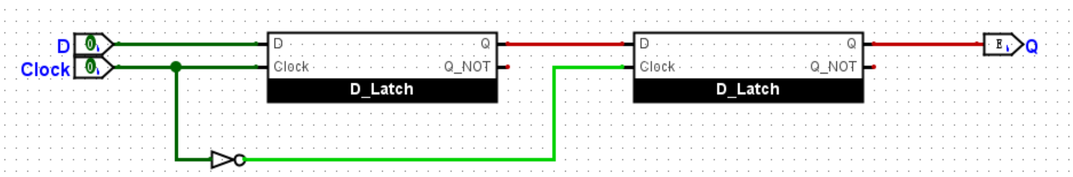
CSC258 Lab 4 Report

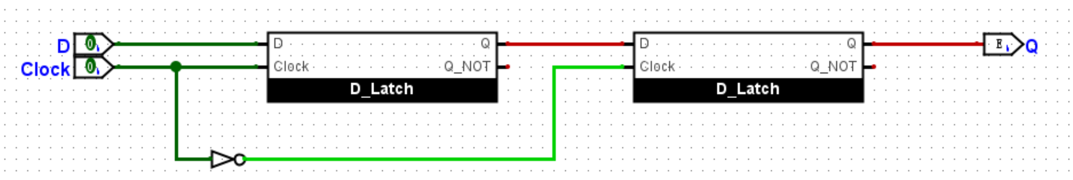
Tianle Wang 1006337028

PART I

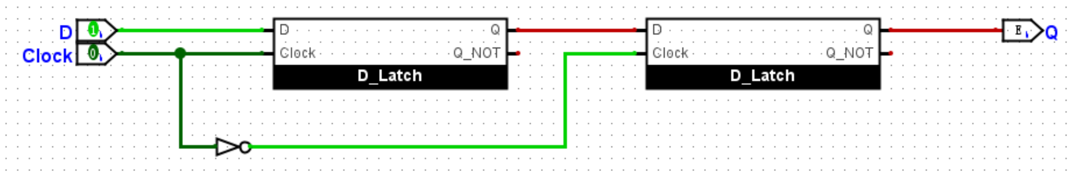
1. In Logisim, build this gated D latch from Fig. 1 and the master slave flip-flop from Fig. 2, each in its own module. The flip-flop should be implemented using the module you create for the D latch, and the latch should make use of the logic gates available in the Gates component set. Some of you have noticed that there is a special Clock signal in the Wiring set. You do not need to use that for this part (you will in future labs). For now, just use the default input pin type for the Clk input signal (the one that you've been using up to this point).
   - 

   - 

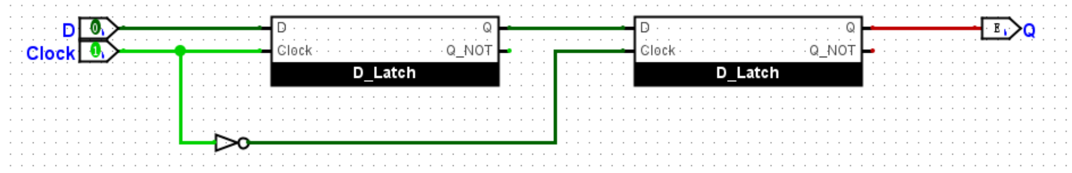
2. Study the behaviour of the latch for different D and clock (Clk) settings by using Poke.
   - If Clock is set to 0, then the whole circuit can't work and the initial status will be Error.
   - If Clock is set to 1, then we can change to D to control the status.
   - For a set status, we can set Clock to 0 so that D can't change the status anymore.
3. Try creating a test vector to test this circuit. In your prelab report, describe what happens when you test various input combinations with a test vector.
   - When I want to create test vector, I found that each of testing will reset the whole circuit so that each time the final result will be E stand error and latch can't work on this situation. As the tutorial said, the test vector can only test combination circuits
4. For the D latch and the flip flop, are there any input combinations of Clk and D that should NOT be the first you test with the Poke( ) tool? Explain this in your prelab and list them if applicable.
   - Any D input and Clock input as the first time input should not be the first test. Since we need Clock to give initial status for the first latch so that it won't result in error as input for the second latch. It means we need at least switch clock twice to setup initial status. For example `D=0` `clock =0`



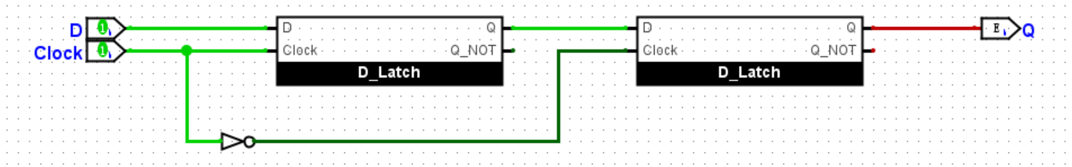
`D=1` `clock =0`



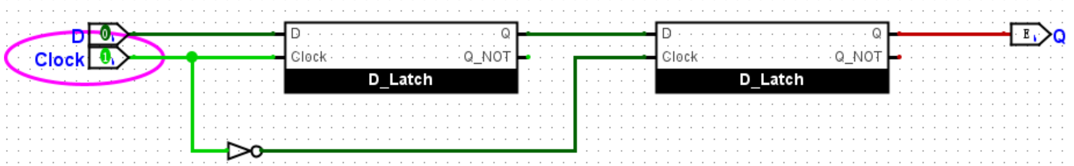
`D=0` `clock =1`



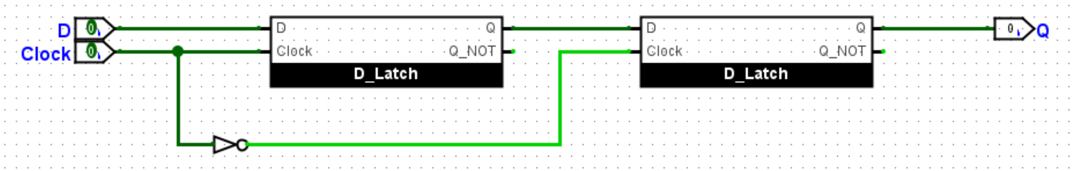
`D=1` `clock =1`



- As mentioned, we need to switch Clock to 1 so that the first D Latch can have some status as output instead error, then we need switch Clock so that the second latch can change the status.

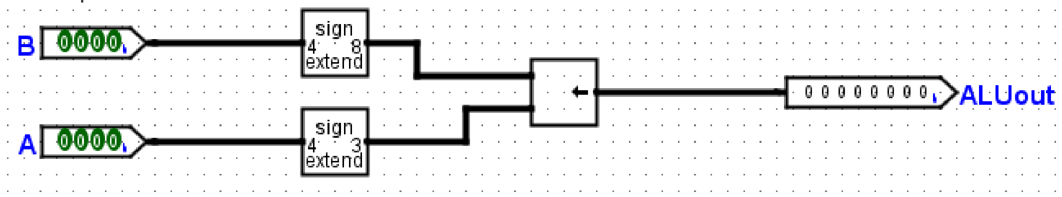


- Followed last step, we switch Clock to 0 so that the second Latch can have some status.

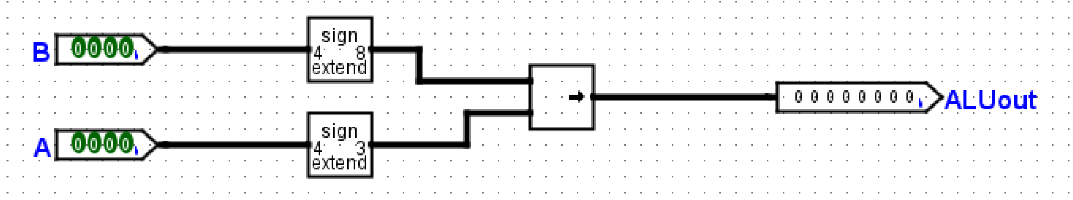


Part II

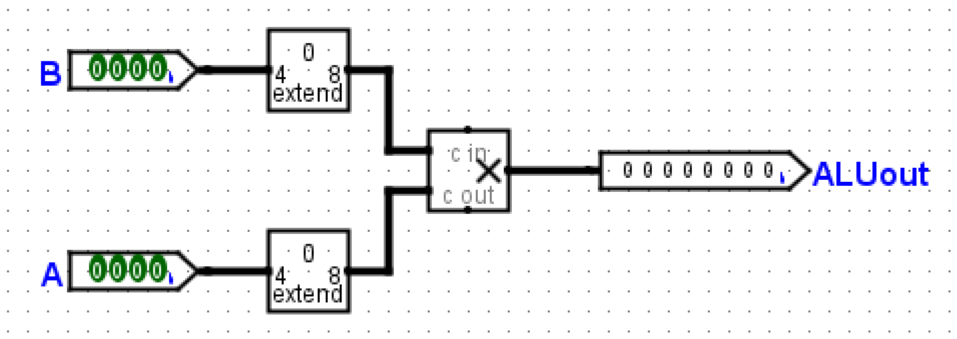
1. Build the Logisim module for the ALU described above (you are strongly encouraged to extend your design from Lab 3).
    - New operation 5



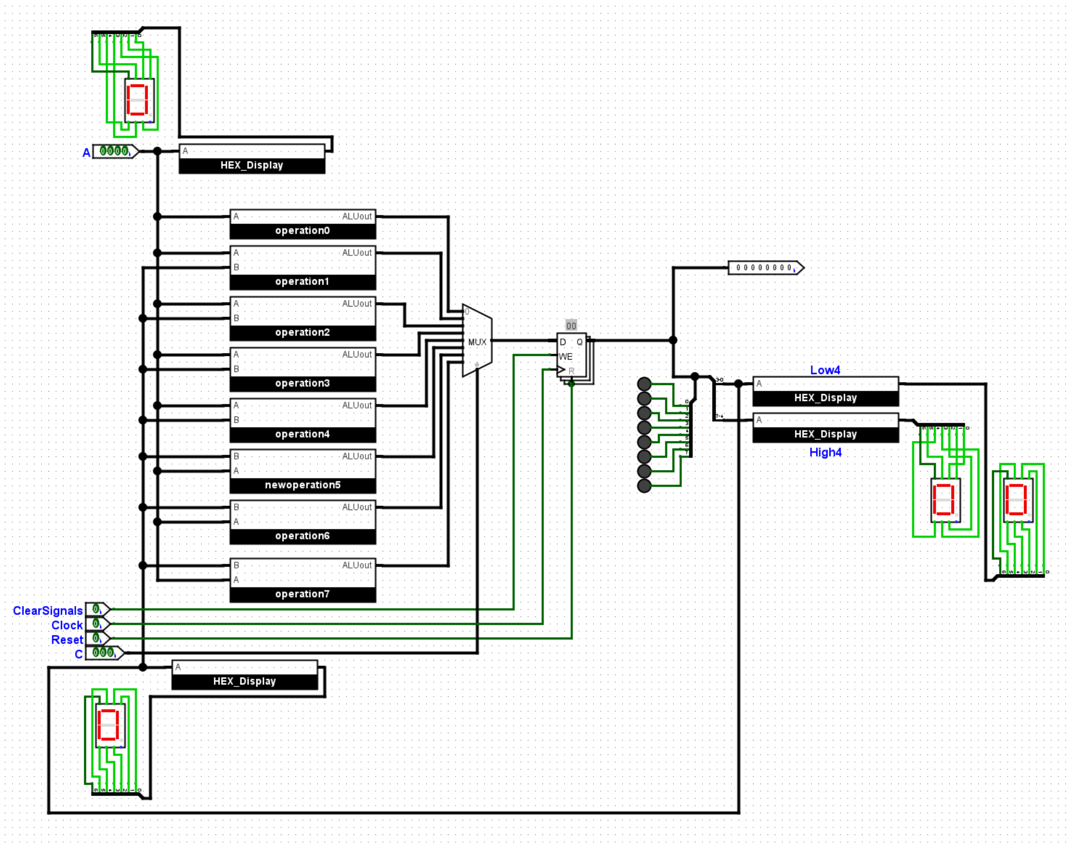
    - Operation 6



- Operation 7

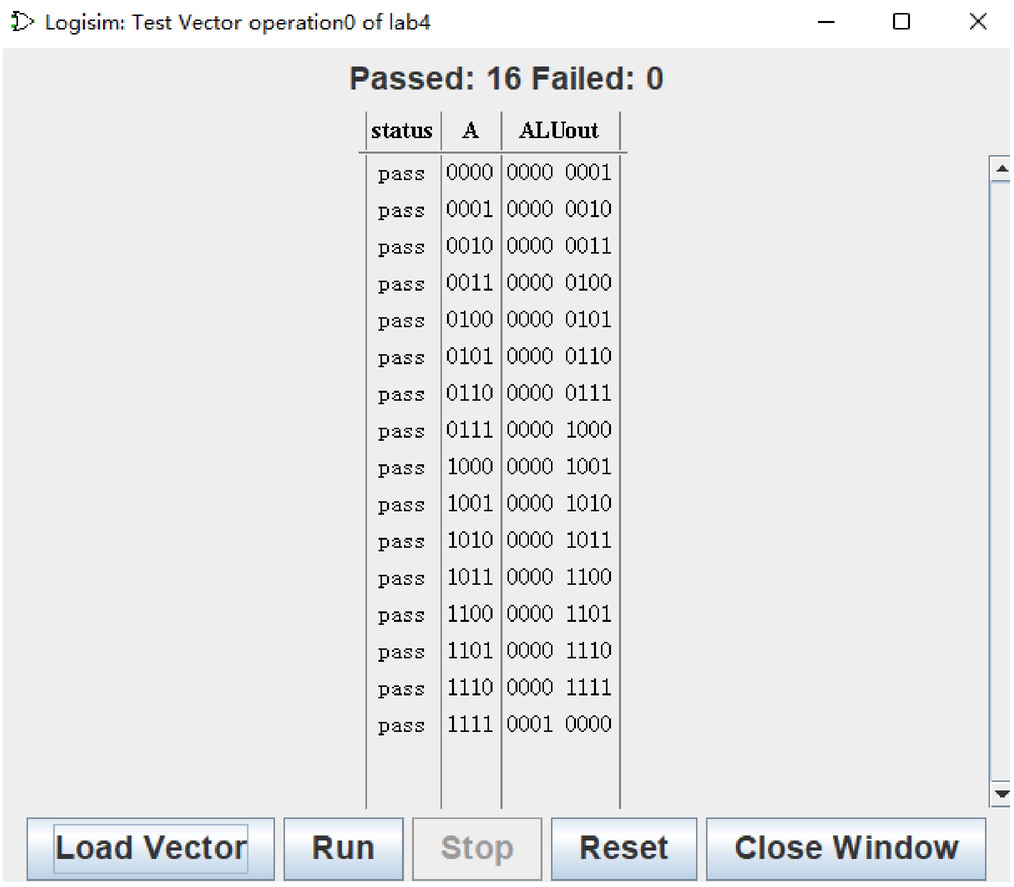


- Final Result



2. Include answers to the following questions in your prelab report:
   1. How would this ALU behave if you didn't include the register in your diagram (i.e. if the ALU output went straight into ALU input B)?
      - For operation 0, it will work since it only asks the value of A. But for the other operation, it will ask the value of B so that a infinite unstop loop will exists.
   2. How would this ALU behave if the register was implemented using clocked latches instead of edge-triggered flip-flops?
      - Similarly, a infinite loop will exists. Since we can only change the state when `clock = 1`. A simple clocked latches will continue change states while `clock = 1`.
   3. When multiplying two n-bit binary numbers, how many bits will you need to store the result?
      - $2 \times n$ bits
   4. (from the shift section above) What will the outputs be for the left and right shift operations when A is greater than 7?
      - For a 8-bits input, if we do 7 bits shift, there is only 1 bit from input left. If we shift bits greater than 7, then there is no value from original input. That means whatever the input is, it will result in the same value like 0000 0000 (logical shift) or 1111 1111 (negative arithmetic shift right)

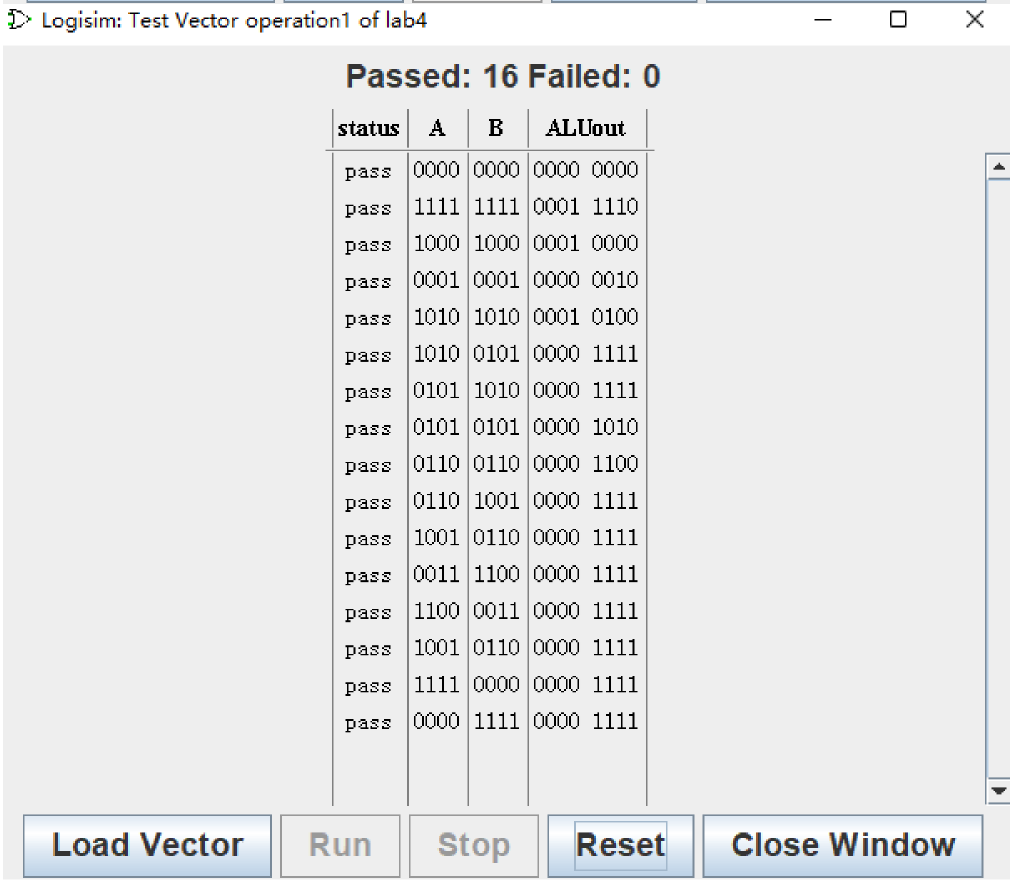
3. Test your modules with Poke. Choose test cases that make you feel confident about your ALU's correctness in preparation for you in-lab demo.
   1. Use test vectors to test the operation of your ALU in isolation (i.e. independent of the register and the feedback of the output into the ALU input B). Submit this test vector file.

- 

Logisim: Test Vector operation0 of lab4

Passed: 16 Failed: 0

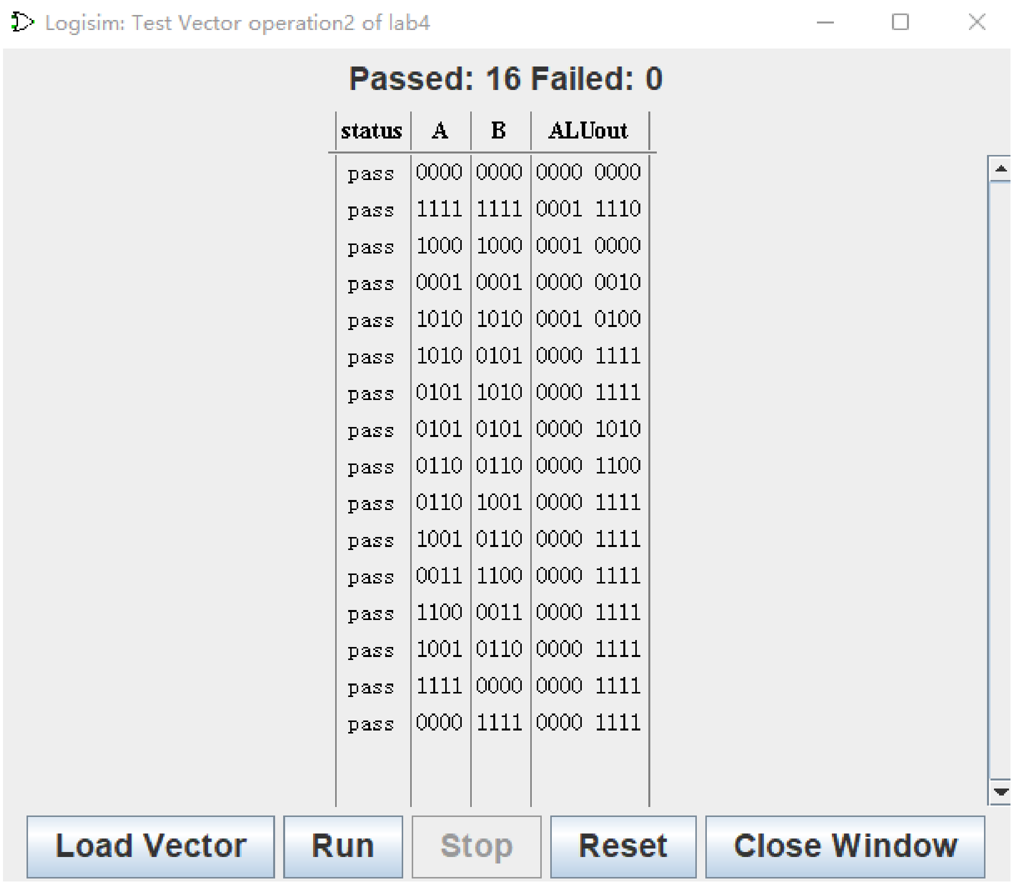
status	A	ALUout
pass	0000	0000 0001
pass	0001	0000 0010
pass	0010	0000 0011
pass	0011	0000 0100
pass	0100	0000 0101
pass	0101	0000 0110
pass	0110	0000 0111
pass	0111	0000 1000
pass	1000	0000 1001
pass	1001	0000 1010
pass	1010	0000 1011
pass	1011	0000 1100
pass	1100	0000 1101
pass	1101	0000 1110
pass	1110	0000 1111
pass	1111	0001 0000

- 

Logisim: Test Vector operation1 of lab4

Passed: 16 Failed: 0

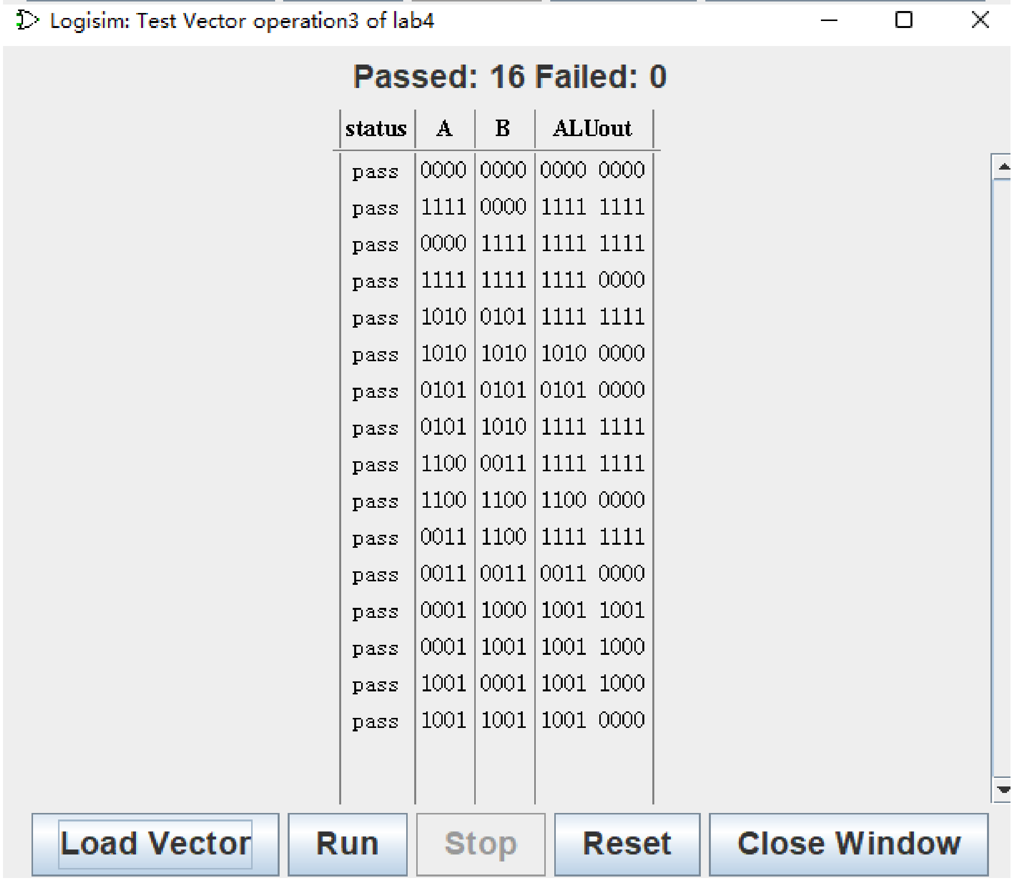
status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1111	1111	0001 1110
pass	1000	1000	0001 0000
pass	0001	0001	0000 0010
pass	1010	1010	0001 0100
pass	1010	0101	0000 1111
pass	0101	1010	0000 1111
pass	0101	0101	0000 1010
pass	0110	0110	0000 1100
pass	0110	1001	0000 1111
pass	1001	0110	0000 1111
pass	0011	1100	0000 1111
pass	1100	0011	0000 1111
pass	1001	0110	0000 1111
pass	1111	0000	0000 1111
pass	0000	1111	0000 1111



Logisim: Test Vector operation2 of lab4

Passed: 16 Failed: 0

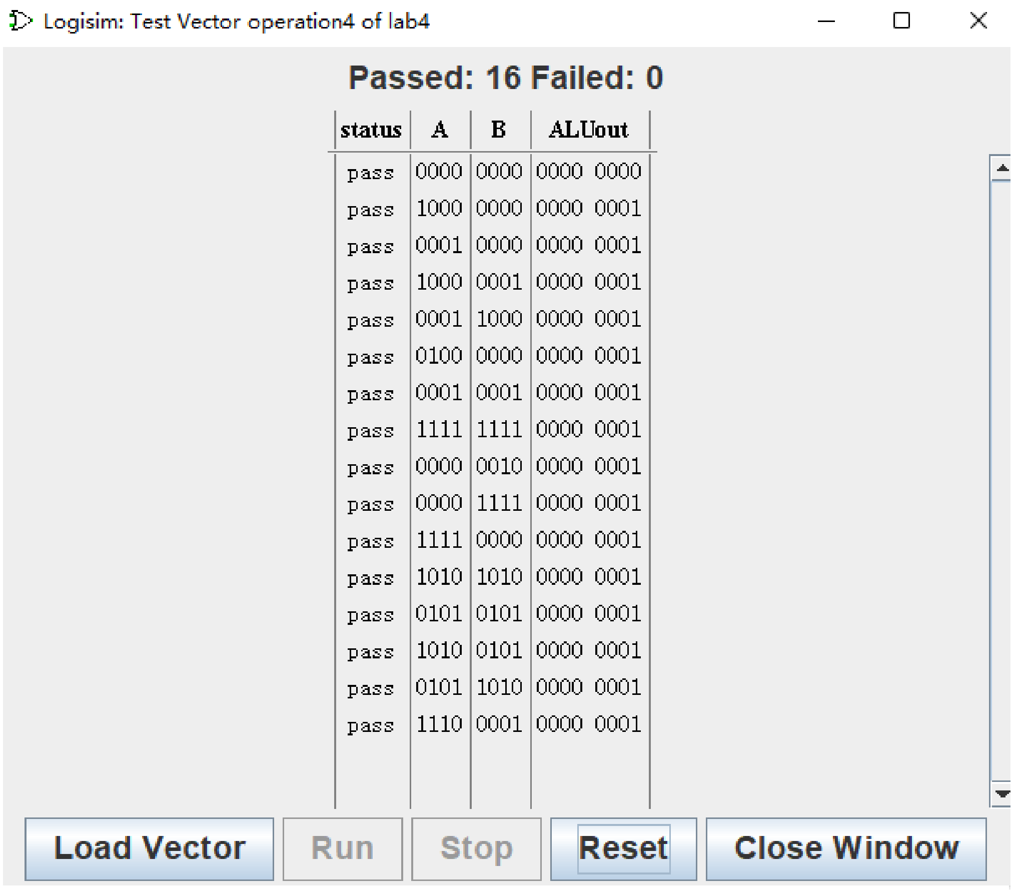
status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1111	1111	0001 1110
pass	1000	1000	0001 0000
pass	0001	0001	0000 0010
pass	1010	1010	0001 0100
pass	1010	0101	0000 1111
pass	0101	1010	0000 1111
pass	0101	0101	0000 1010
pass	0110	0110	0000 1100
pass	0110	1001	0000 1111
pass	1001	0110	0000 1111
pass	0011	1100	0000 1111
pass	1100	0011	0000 1111
pass	1001	0110	0000 1111
pass	1111	0000	0000 1111
pass	0000	1111	0000 1111



Logisim: Test Vector operation3 of lab4

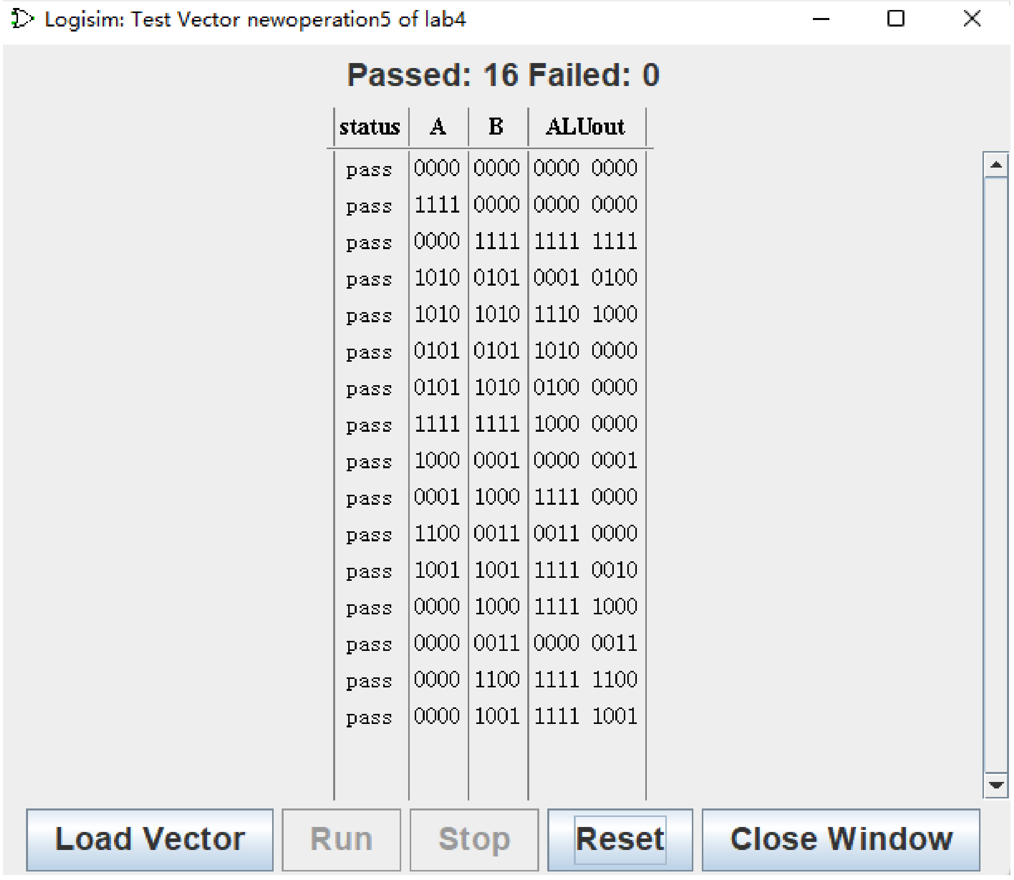
Passed: 16 Failed: 0

status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1111	0000	1111 1111
pass	0000	1111	1111 1111
pass	1111	1111	1111 0000
pass	1010	0101	1111 1111
pass	1010	1010	1010 0000
pass	0101	0101	0101 0000
pass	0101	1010	1111 1111
pass	1100	0011	1111 1111
pass	1100	1100	1100 0000
pass	0011	1100	1111 1111
pass	0011	0011	0011 0000
pass	0001	1000	1001 1001
pass	0001	1001	1001 1000
pass	1001	0001	1001 1000
pass	1001	1001	1001 0000



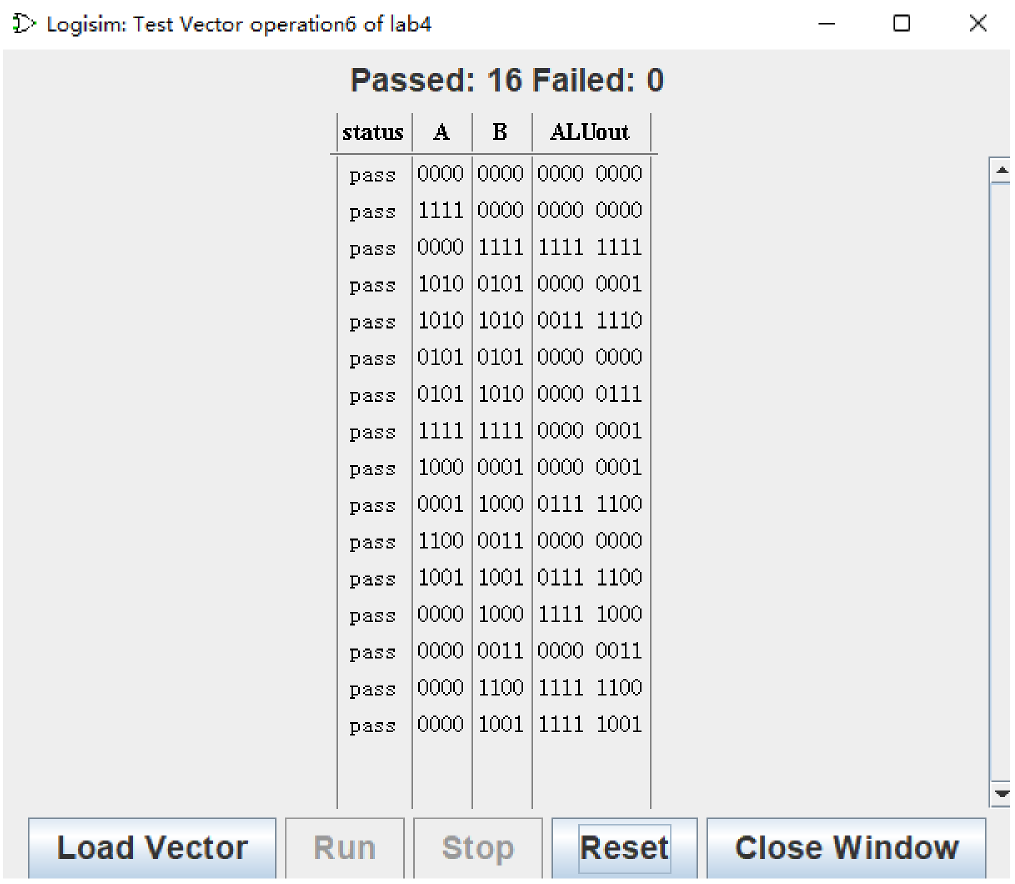
Passed: 16 Failed: 0

status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1000	0000	0000 0001
pass	0001	0000	0000 0001
pass	1000	0001	0000 0001
pass	0001	1000	0000 0001
pass	0100	0000	0000 0001
pass	0001	0001	0000 0001
pass	1111	1111	0000 0001
pass	0000	0010	0000 0001
pass	0000	1111	0000 0001
pass	1111	0000	0000 0001
pass	1010	1010	0000 0001
pass	0101	0101	0000 0001
pass	1010	0101	0000 0001
pass	0101	1010	0000 0001
pass	1110	0001	0000 0001



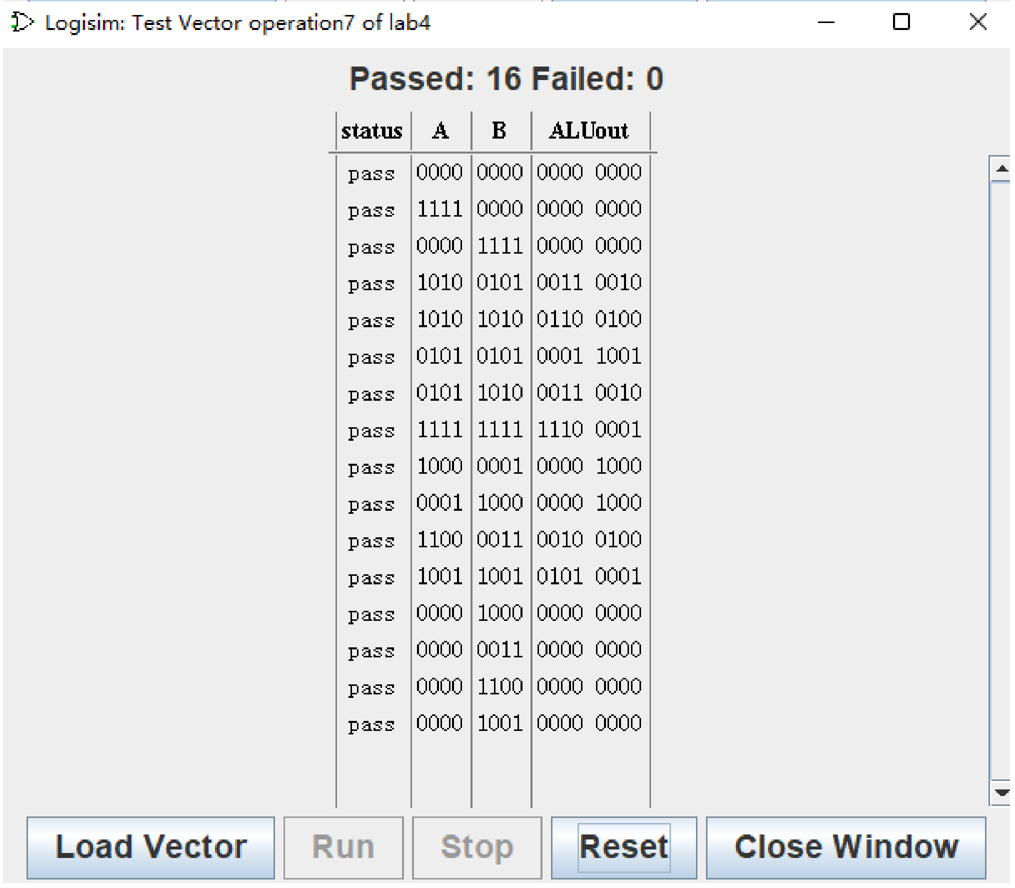
Passed: 16 Failed: 0

status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1111	0000	0000 0000
pass	0000	1111	1111 1111
pass	1010	0101	0001 0100
pass	1010	1010	1110 1000
pass	0101	0101	1010 0000
pass	0101	1010	0100 0000
pass	1111	1111	1000 0000
pass	1000	0001	0000 0001
pass	0001	1000	1111 0000
pass	1100	0011	0011 0000
pass	1001	1001	1111 0010
pass	0000	1000	1111 1000
pass	0000	0011	0000 0011
pass	0000	1100	1111 1100
pass	0000	1001	1111 1001

- 
Logisim: Test Vector operation6 of lab4

Passed: 16 Failed: 0

status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1111	0000	0000 0000
pass	0000	1111	1111 1111
pass	1010	0101	0000 0001
pass	1010	1010	0011 1110
pass	0101	0101	0000 0000
pass	0101	1010	0000 0111
pass	1111	1111	0000 0001
pass	1000	0001	0000 0001
pass	0001	1000	0111 1100
pass	1100	0011	0000 0000
pass	1001	1001	0111 1100
pass	0000	1000	1111 1000
pass	0000	0011	0000 0011
pass	0000	1100	1111 1100
pass	0000	1001	1111 1001

- 
Logisim: Test Vector operation7 of lab4

Passed: 16 Failed: 0

status	A	B	ALUout
pass	0000	0000	0000 0000
pass	1111	0000	0000 0000
pass	0000	1111	0000 0000
pass	1010	0101	0011 0010
pass	1010	1010	0110 0100
pass	0101	0101	0001 1001
pass	0101	1010	0011 0010
pass	1111	1111	1110 0001
pass	1000	0001	0000 1000
pass	0001	1000	0000 1000
pass	1100	0011	0010 0100
pass	1001	1001	0101 0001
pass	0000	1000	0000 0000
pass	0000	0011	0000 0000
pass	0000	1100	0000 0000
pass	0000	1001	0000 0000

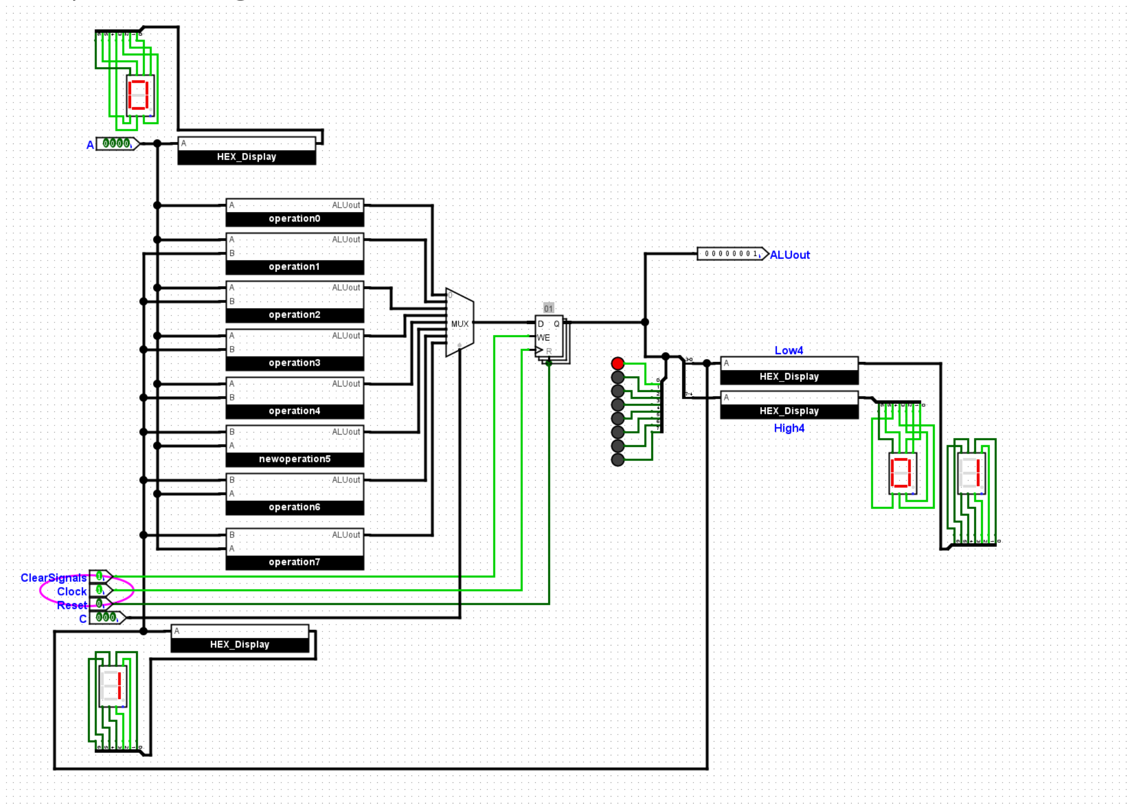
2. When testing the circuit in its entirety, you can't use test vectors. Instead, document your testing of this circuit in your prelab report by providing a list of the test sequences you used for each ALU operation to verify its correctness.

   Notice: each test base a previous 0 value

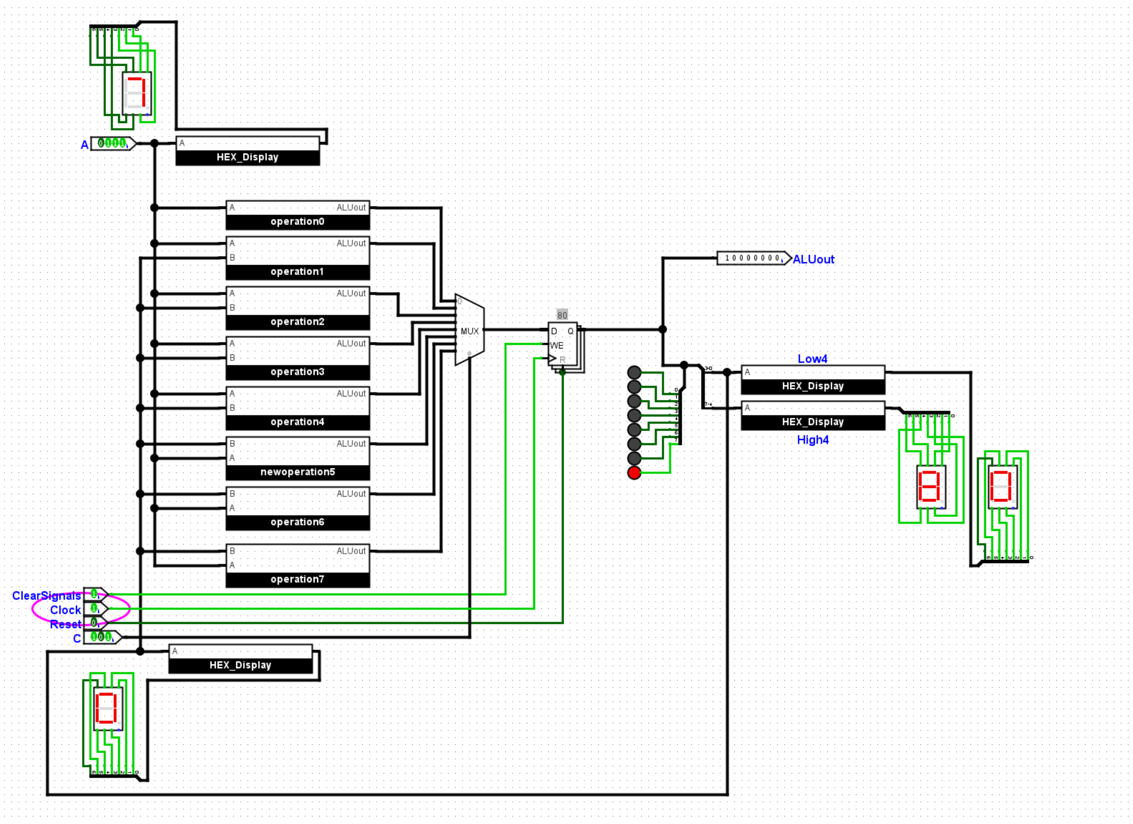
A	C	ClearSignals	Clock	Reset	ALUout
0000	000	1	1	0	00000001
1111	000	1	1	0	00010000
0000	001	1	1	0	00000000
1111	001	1	1	0	00001111
0000	010	1	1	0	00000000
1111	010	1	1	0	00001111
0000	011	1	1	0	00000000
1111	011	1	1	0	11111111
0000	100	1	1	0	00000000
1111	100	1	1	0	00000001
0000	101	1	1	0	00000000
1111	101	1	1	0	00000000
0000	110	1	1	0	00000000
1111	110	1	1	0	00000000
0000	111	1	1	0	00000000
1111	111	1	1	0	00000000

3. For the new operations that weren't implemented in Lab 3, include a few select screenshots with your prelab report of test cases that effectively demonstrate that these functions are operating correctly.

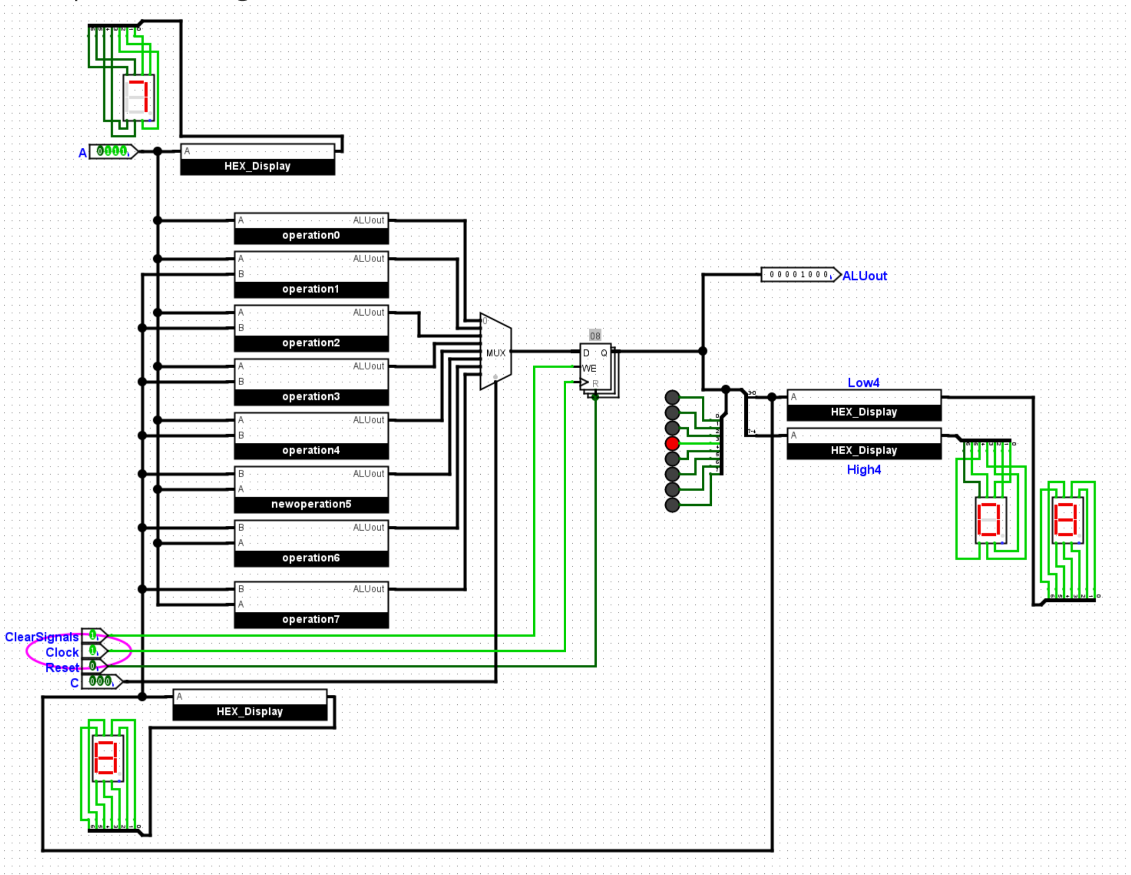
Use operation 0 to give B value of 1



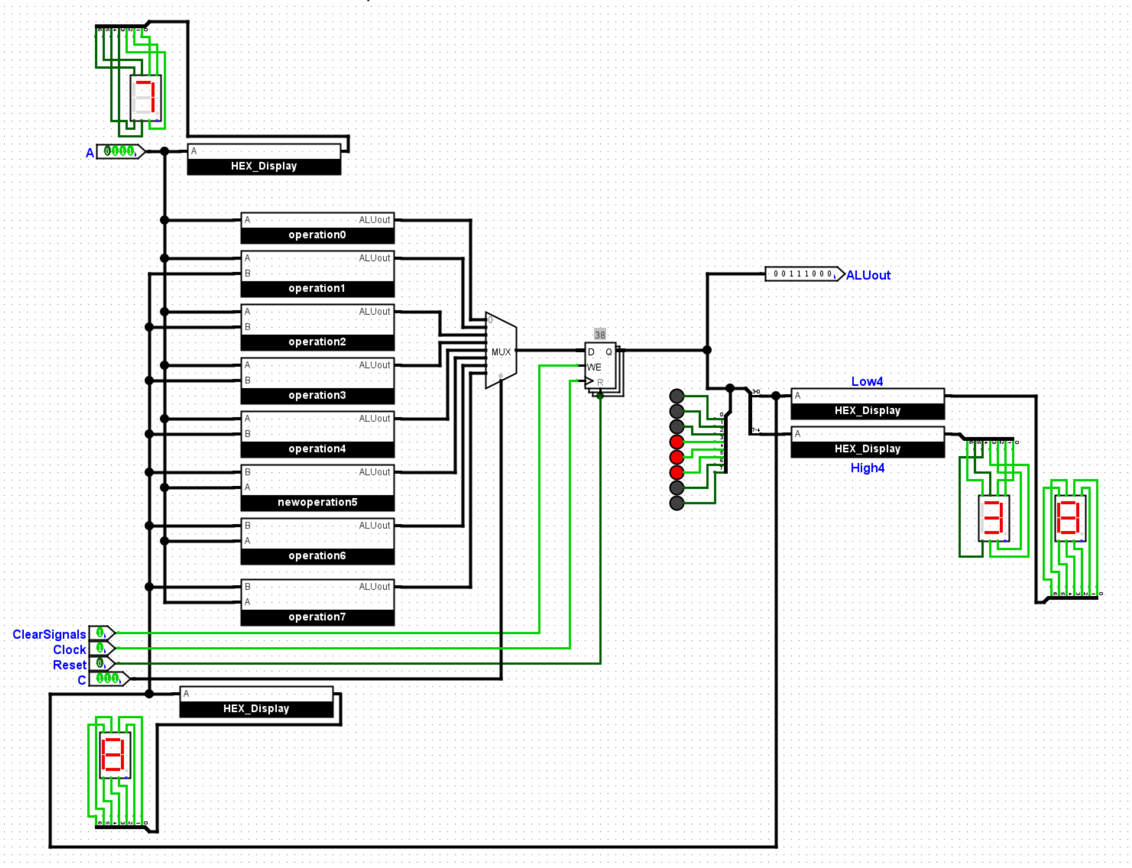
then use operation 5 to left shift 1 by 7 bit



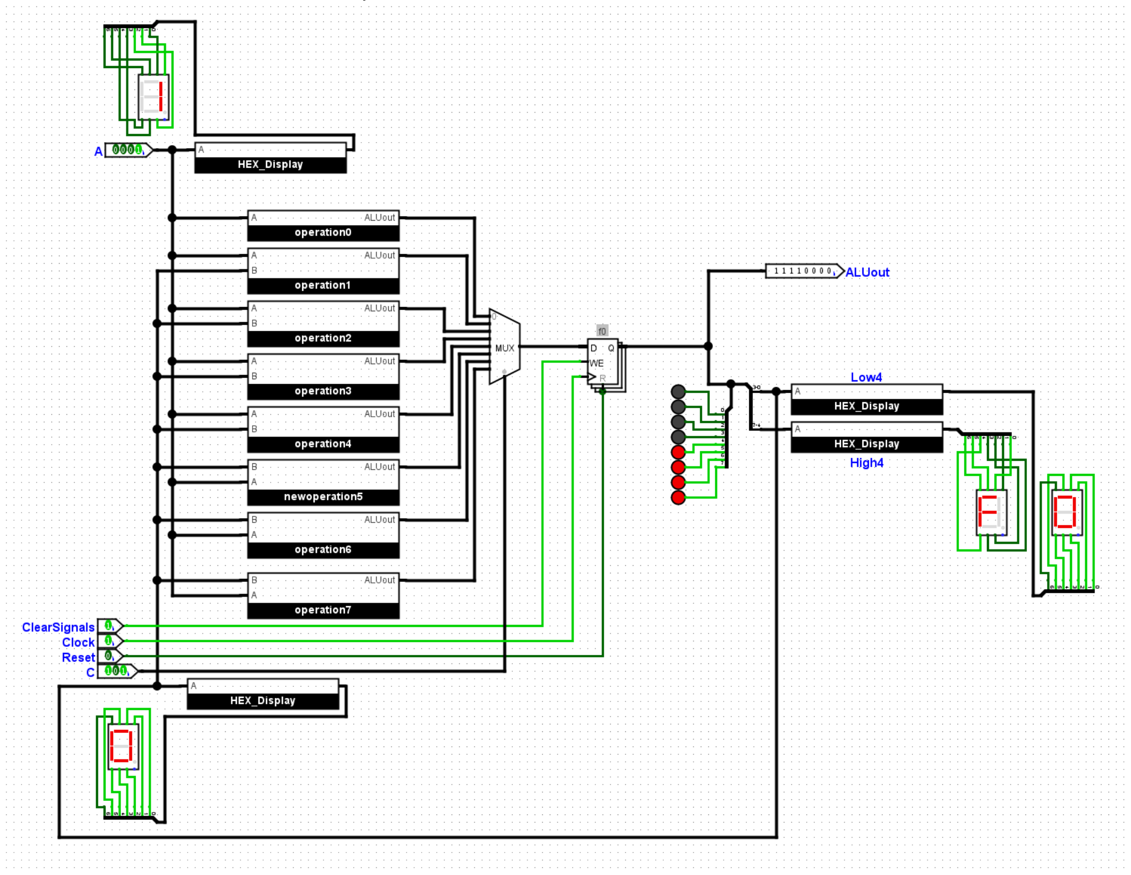
Use operation 0 to give B value of 1000



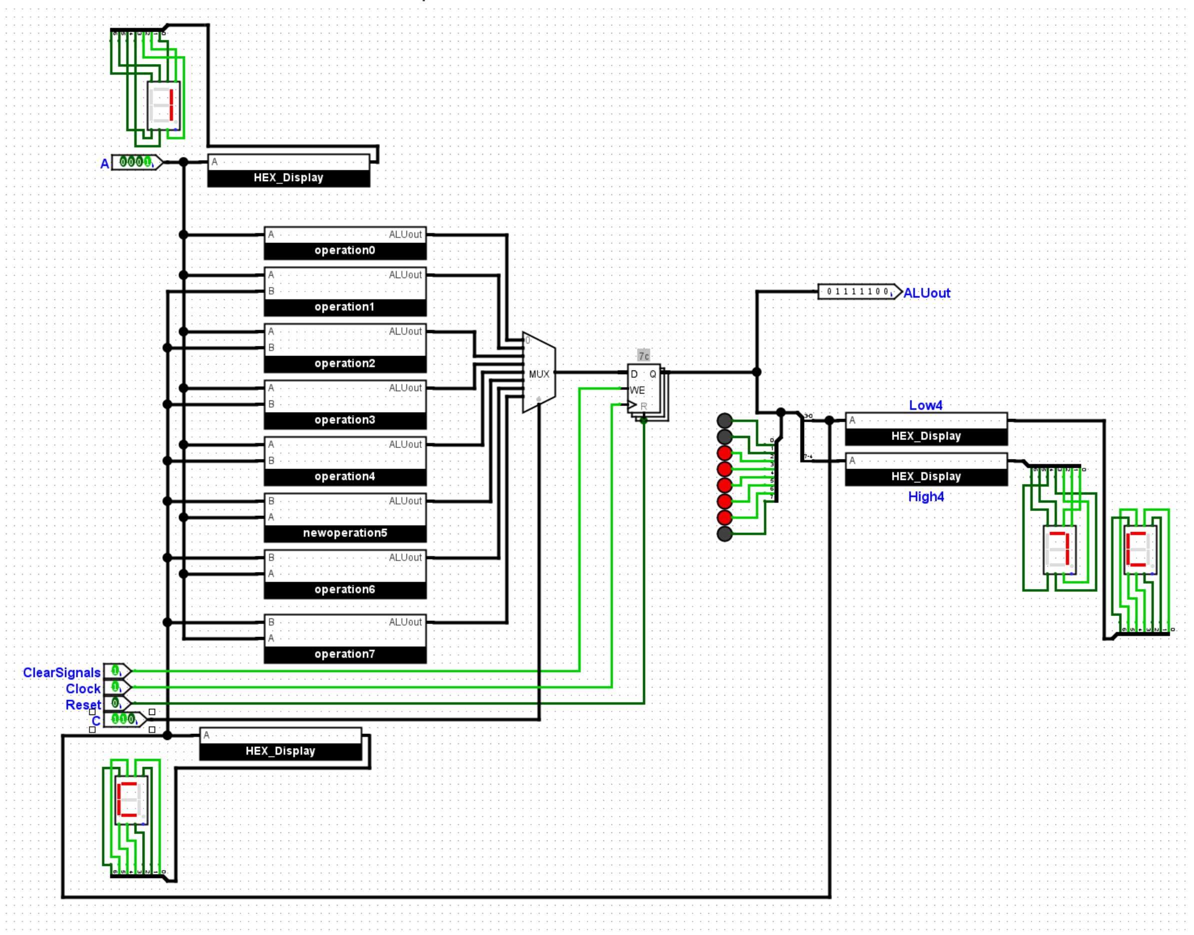
Continue with 1000 and use operation 7 to calculate 7×8 or 0111×1000



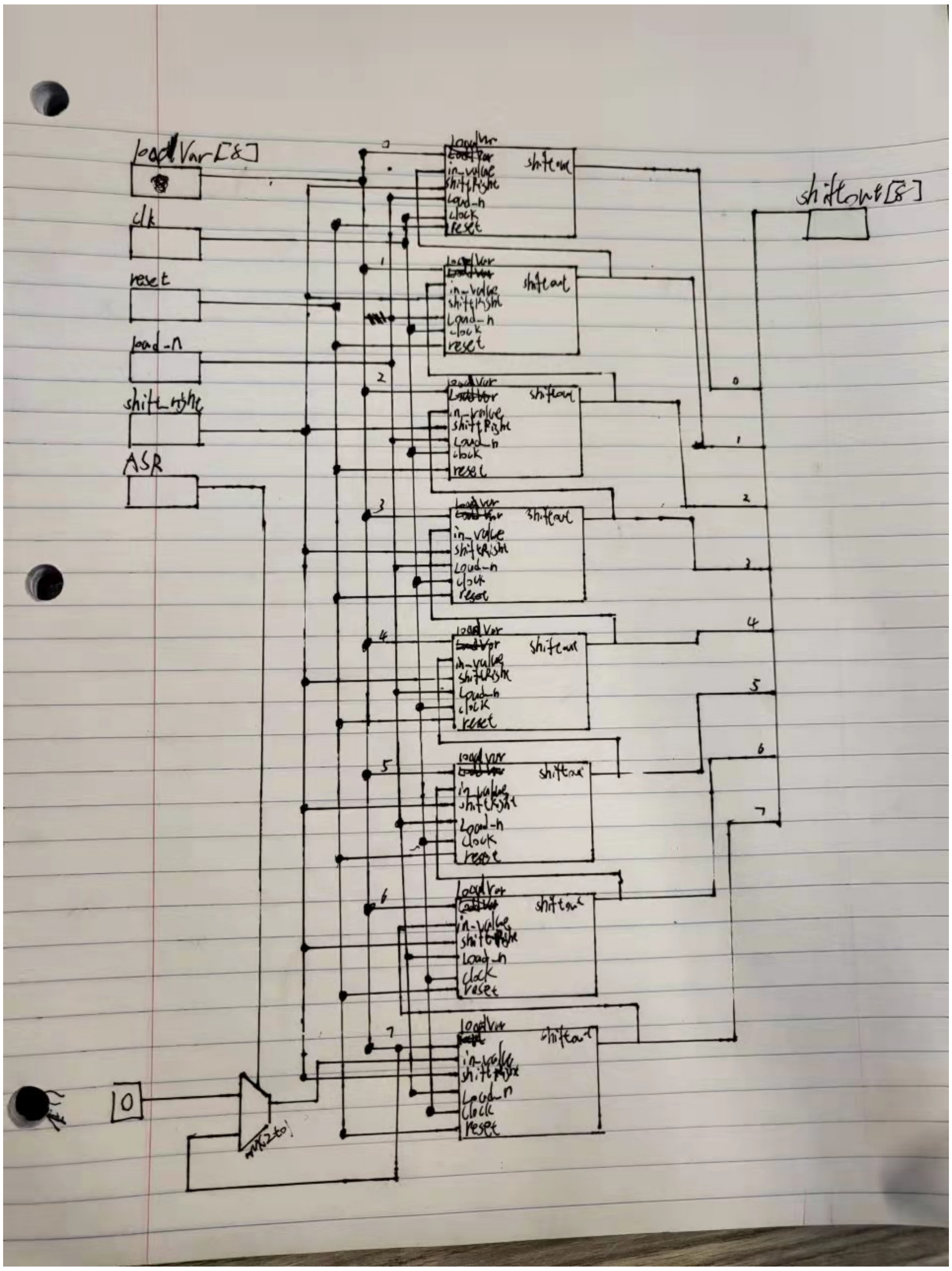
Continue with 1000 and use operation 5



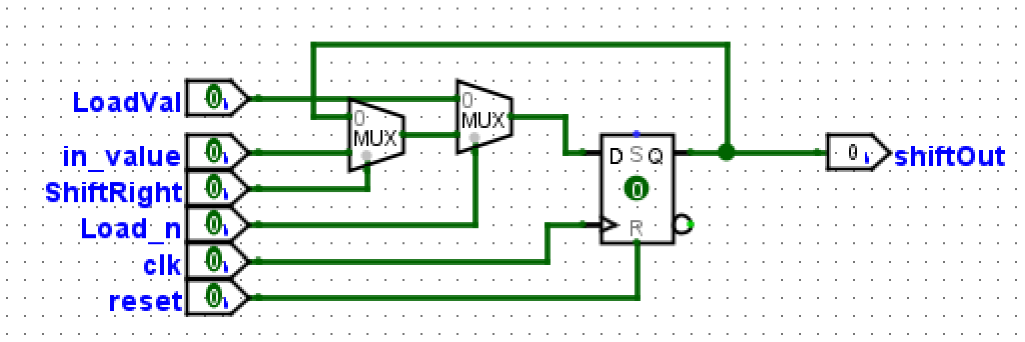
set value 1000 to B and then use operation 6



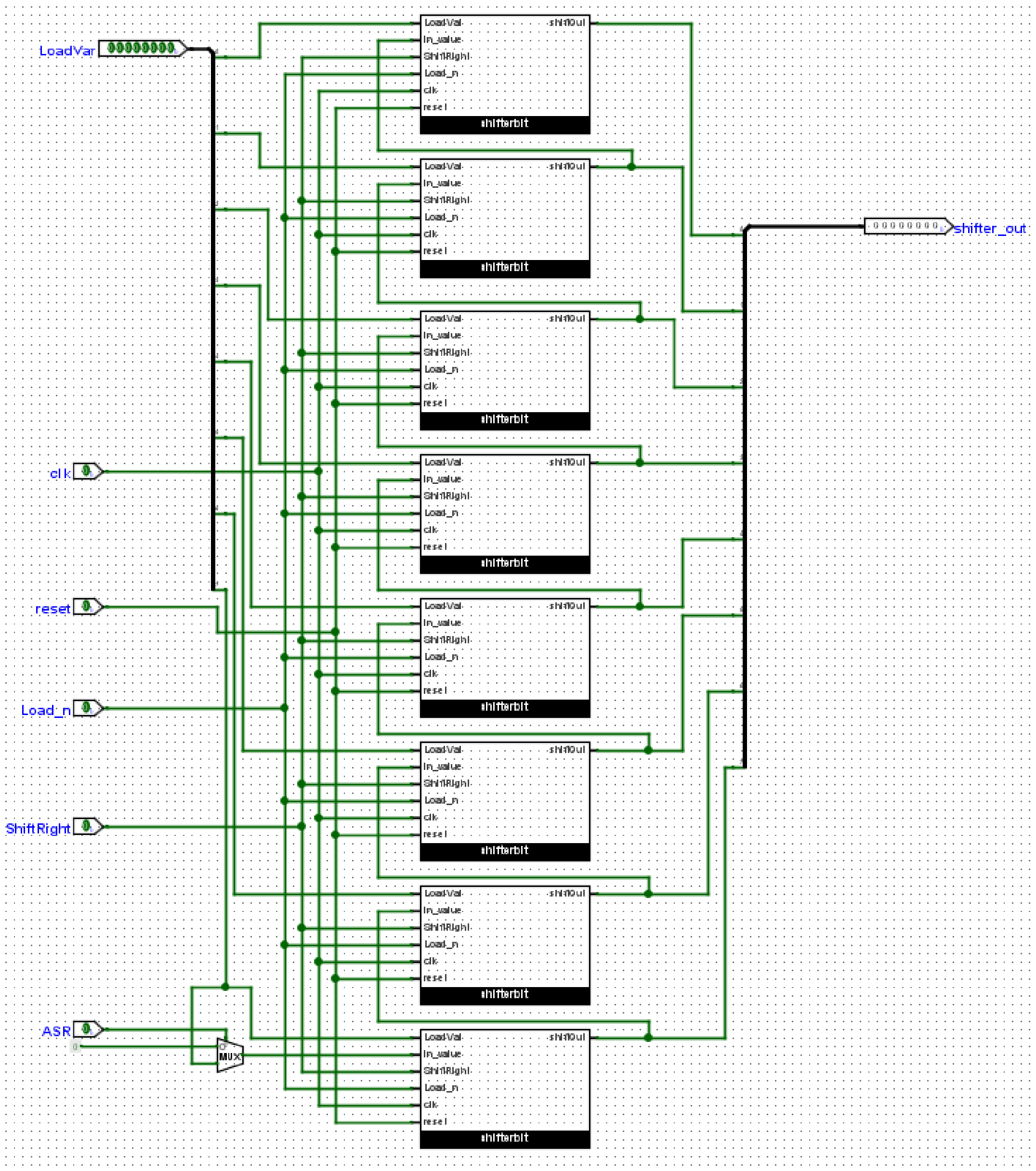
Part III

1. What is the behaviour of the 8-bit shift register shown in Figure 5 when $Load\_n = 1$ and $ShiftRight = 0$? Briefly explain in your prelab.
    - Since the $Load\_n$ controls to load initial value, if it sets to 1, then it does not do the load work. And $ShiftRight = 0$ decides if to do right shift, if it sets to 0, then it does not work to right shift. Thus the result won't be change when when $Load\_n = 1$ and $ShiftRight = 0$. Specially in circuit, the $Load\_n$ and $ShiftRight$ control 2 muxs output. $ShiftRight = 0, Load\_n = 1$ will depend the output as input and the output 0 is from the initial D flip-flop
2. Draw a schematic for the 8-bit shift register shown in Figure 5 including the necessary connections. Your schematic should contain eight instances of the one-bit shifter (i.e. the ShifterBit) shown in Figure 4 and all the wiring required to implement the desired behaviour. Label the signals on your schematic with the same names you will use in your Logisim circuit.
    - 

3. Starting with the built-in positive edge-triggered D flip-flop found at Memory > D Flip Flop, use this D flip-flop with instances of the mux2to1 module from Lab 2 to build the one-bit shifter shown in Figure.

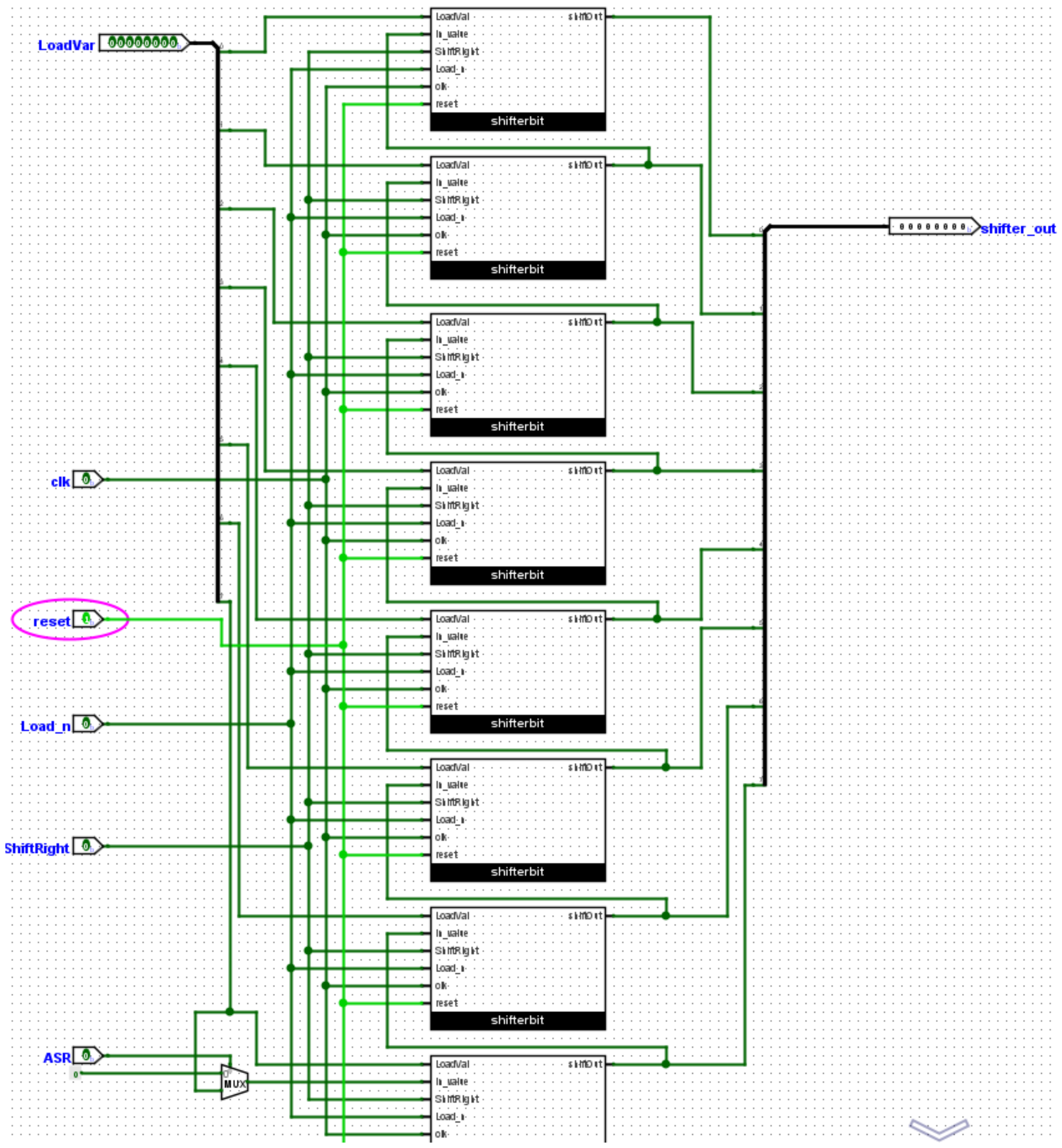


4. Build your Logisim module for the shift register that instantiates and connects eight instances of the ShifterBit. This module should match with the schematic in your prelab.

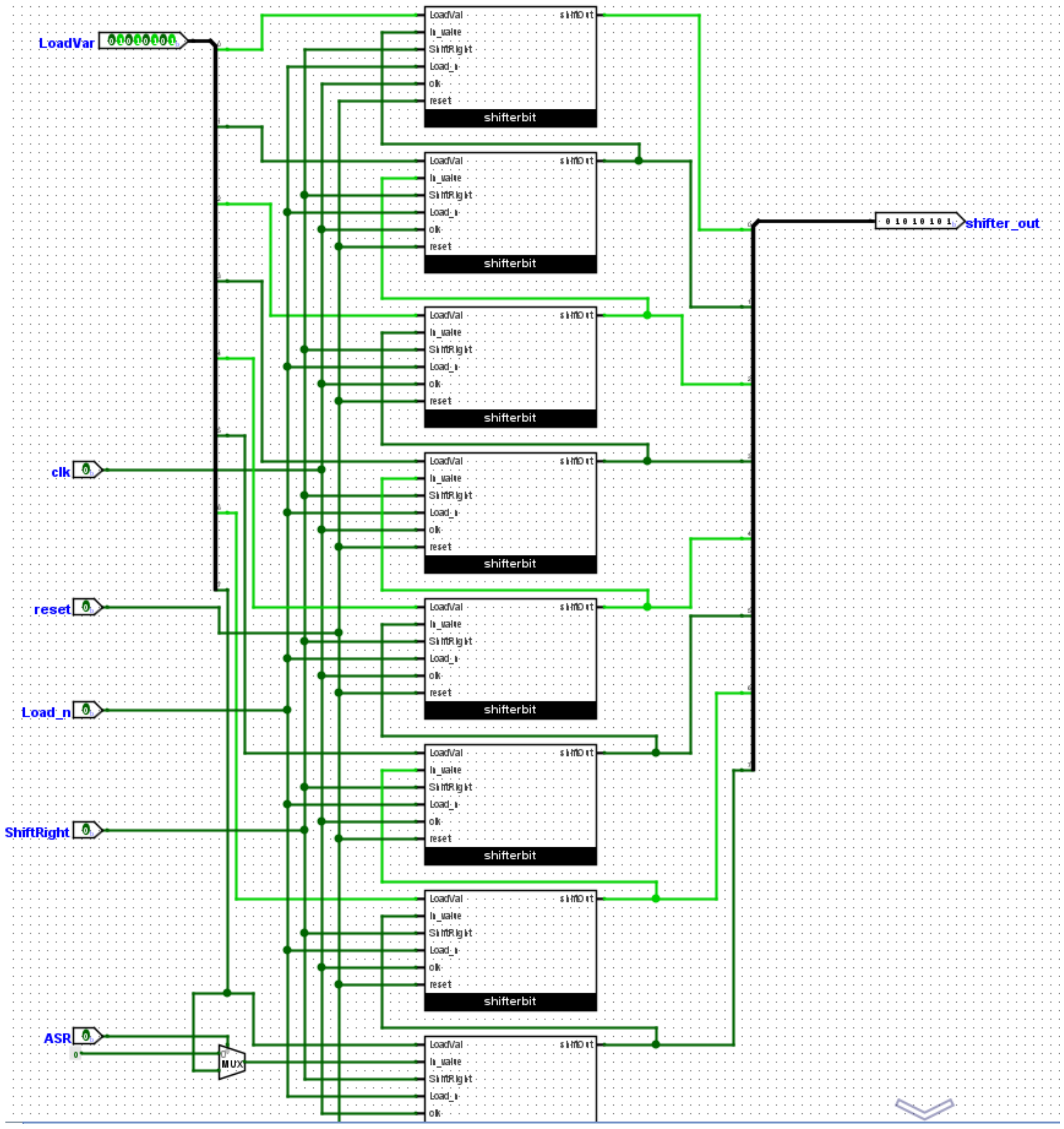


5. Simulate your modules with Poke. Choose test cases that make you feel confident about your shifter’s correctness, in preparation for your in-lab demo. Make sure to include a few selected screenshots of these cases when you hand in your prelab. In your simulation, you should perform the reset operation on the first clock cycle, then do a parallel load of your register on the next cycle. Finally, clock the register for several cycles to demonstrate both types of shifts. (NOTE: If you do not perform a reset first, your simulation will not work! Try simulating without doing reset first and see what happens. Can you explain the results?) Include one (or a few) screenshot of simulation output in your prelab.

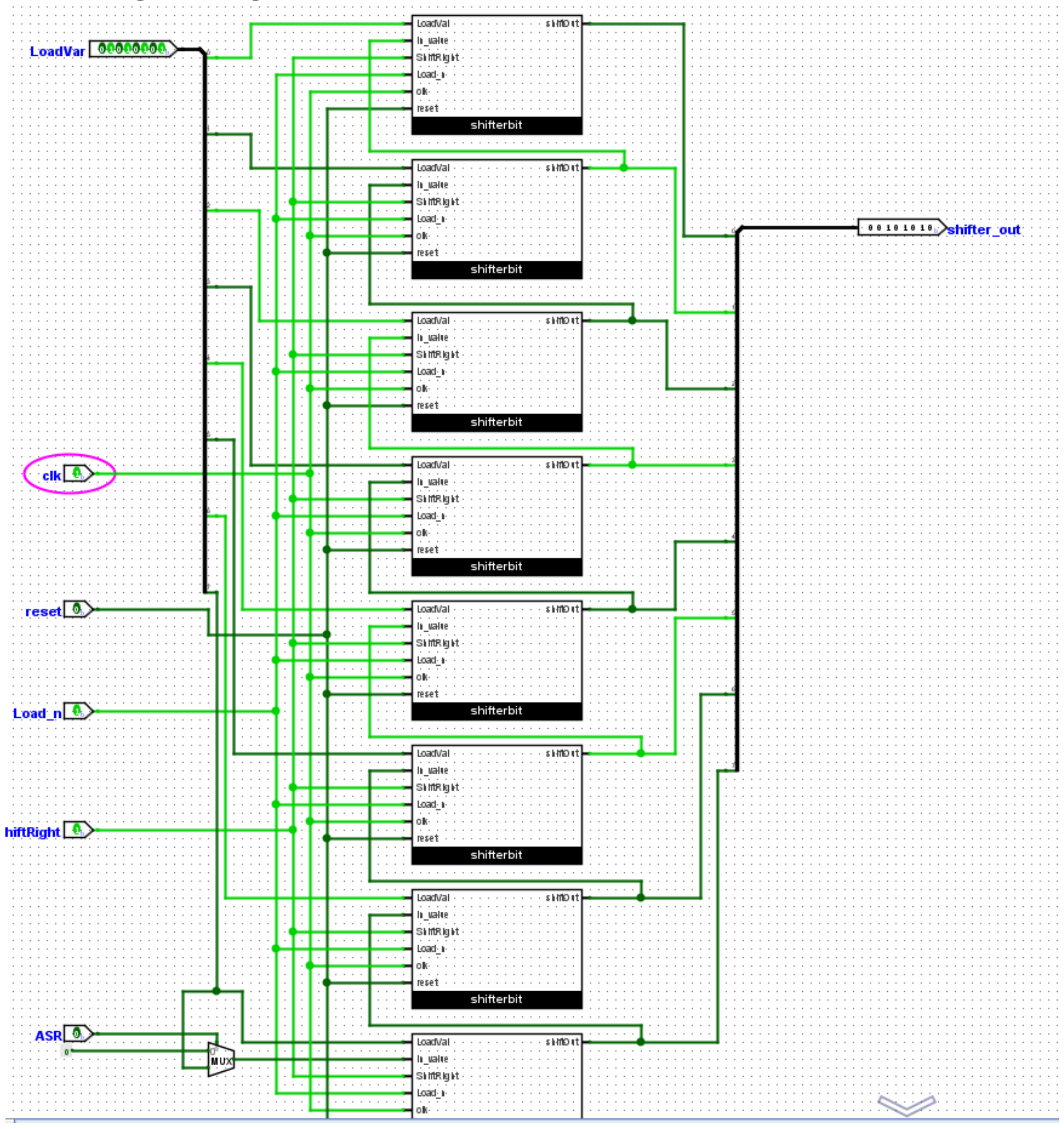
1. Rest


LoadVar
00000000
clk
reset
Load_n
ShiftRight
ASR
MUX
LoadVal
In_value
ShiftRight
Load_n
clk
reset
shiftOut
shifterbit
00000000
shifter_out

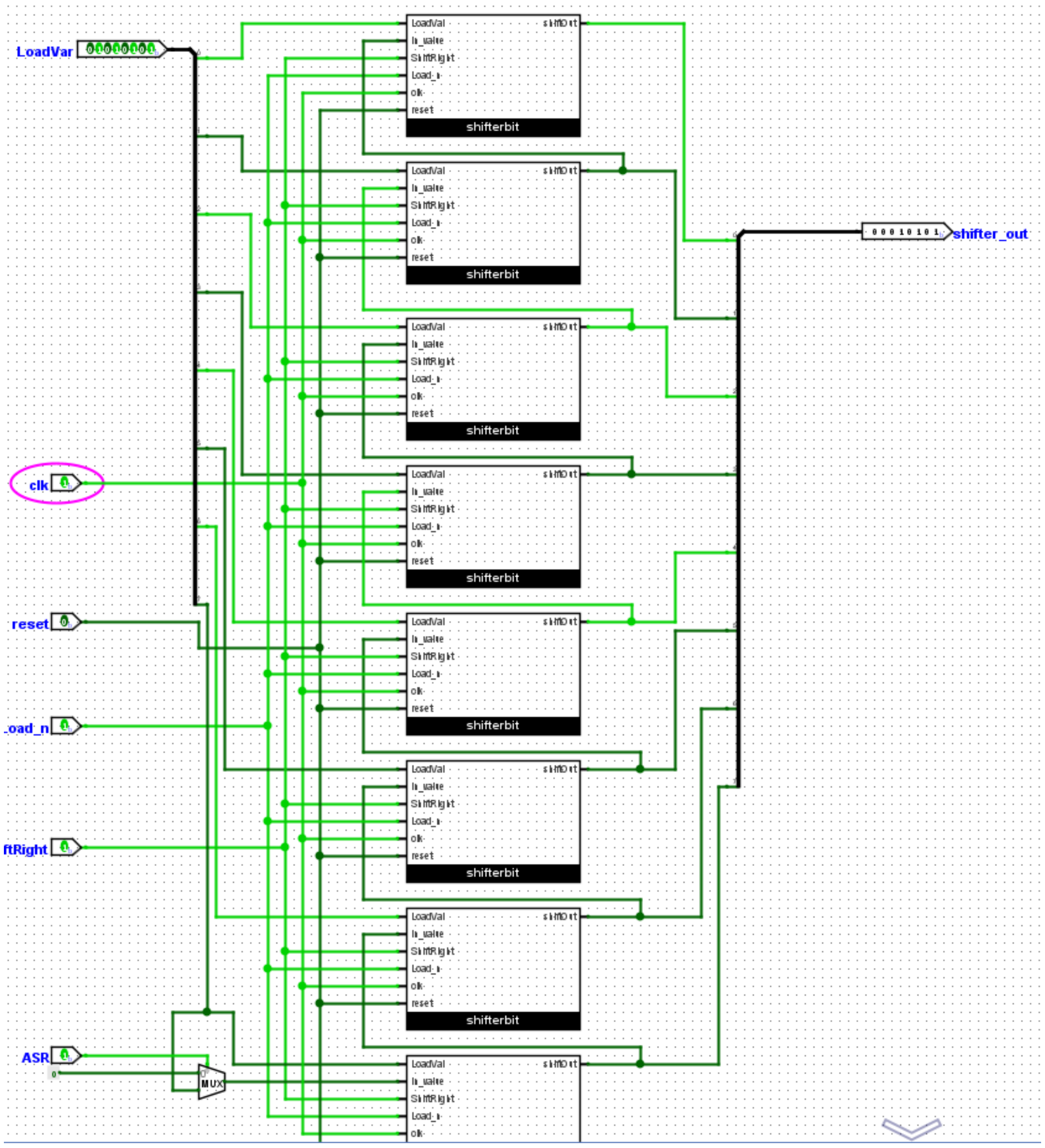
2. then set and load LoadVar



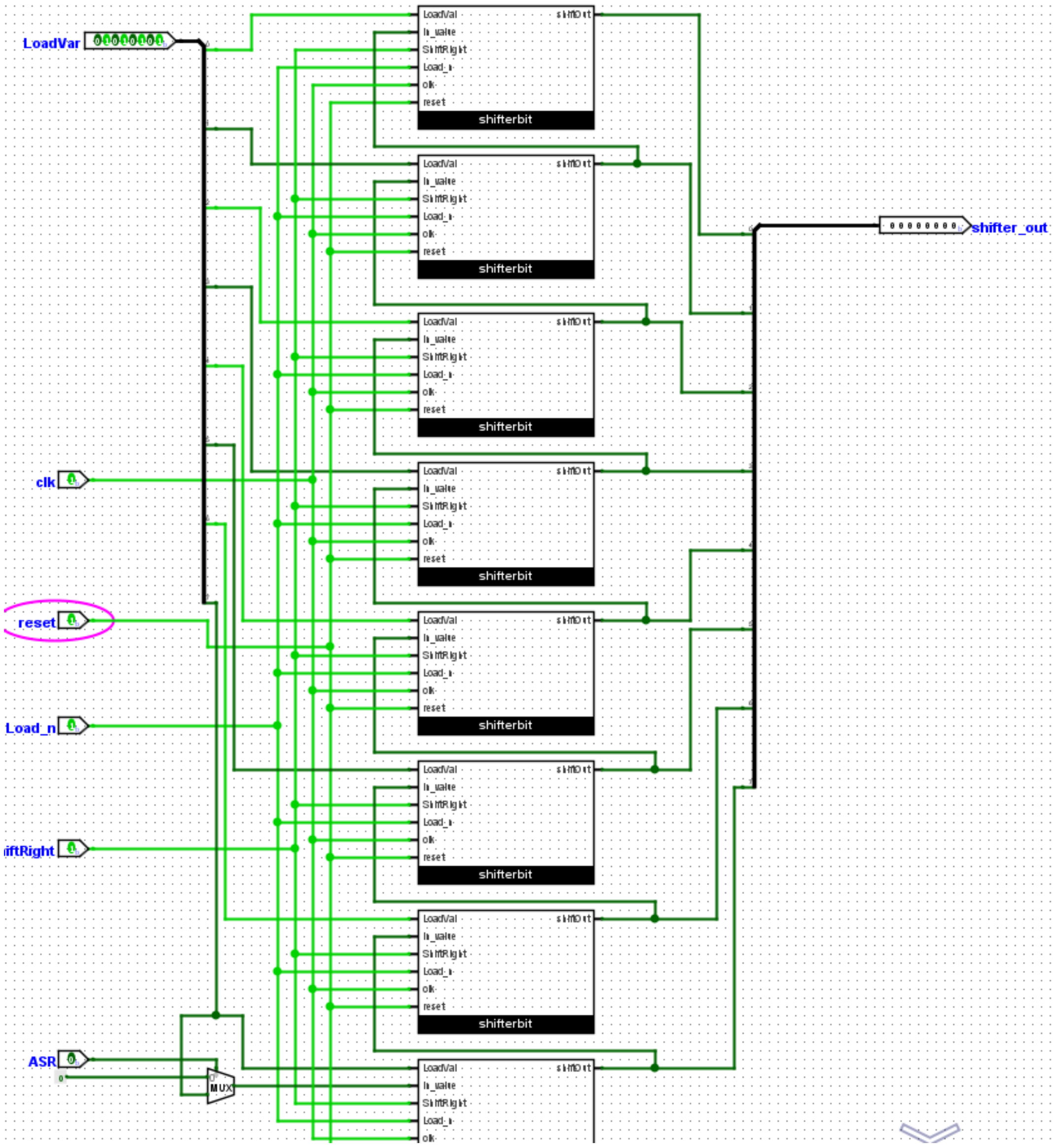
3. then do Logic shift right once



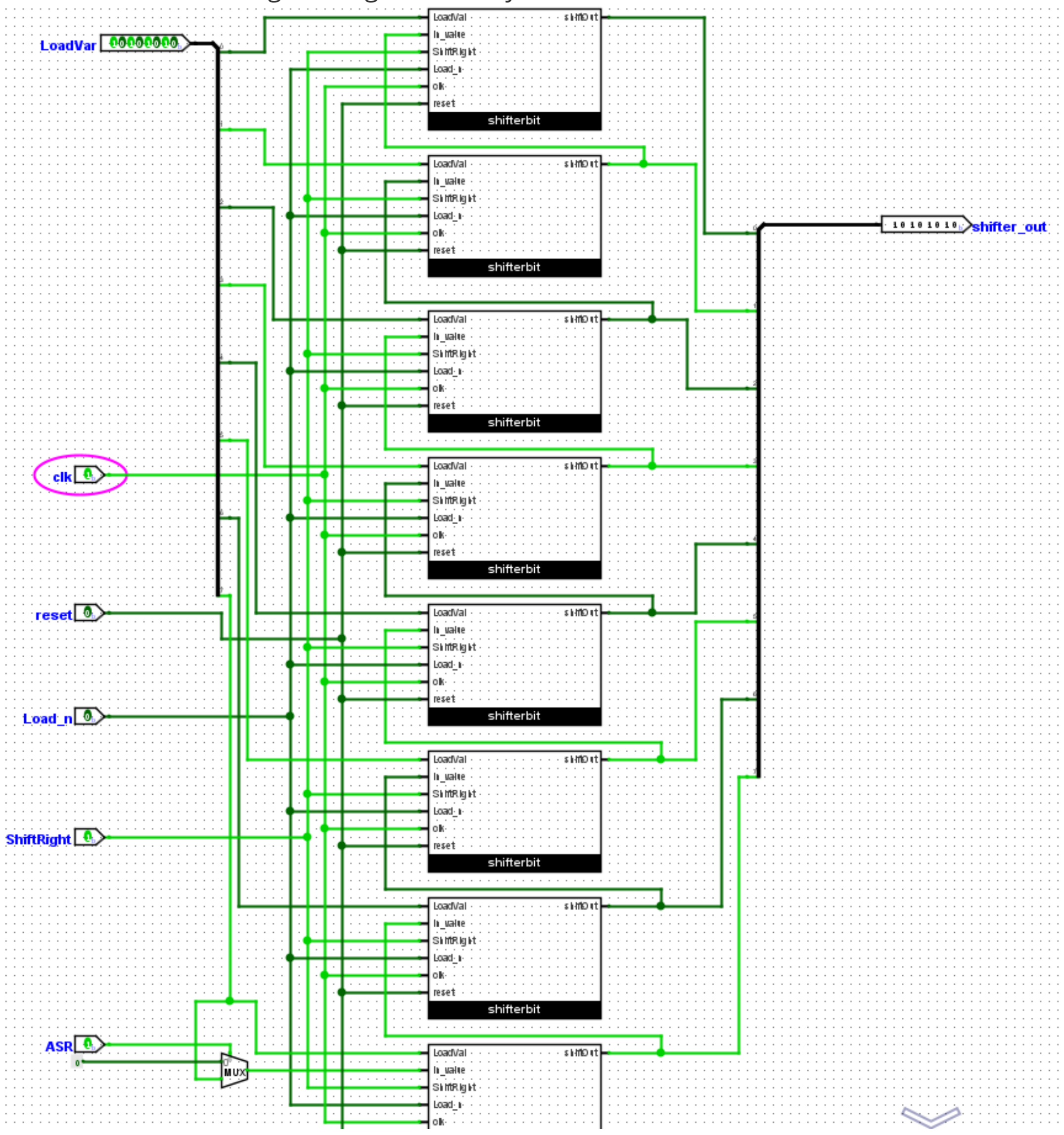
4. then do Arithmetic shift right once



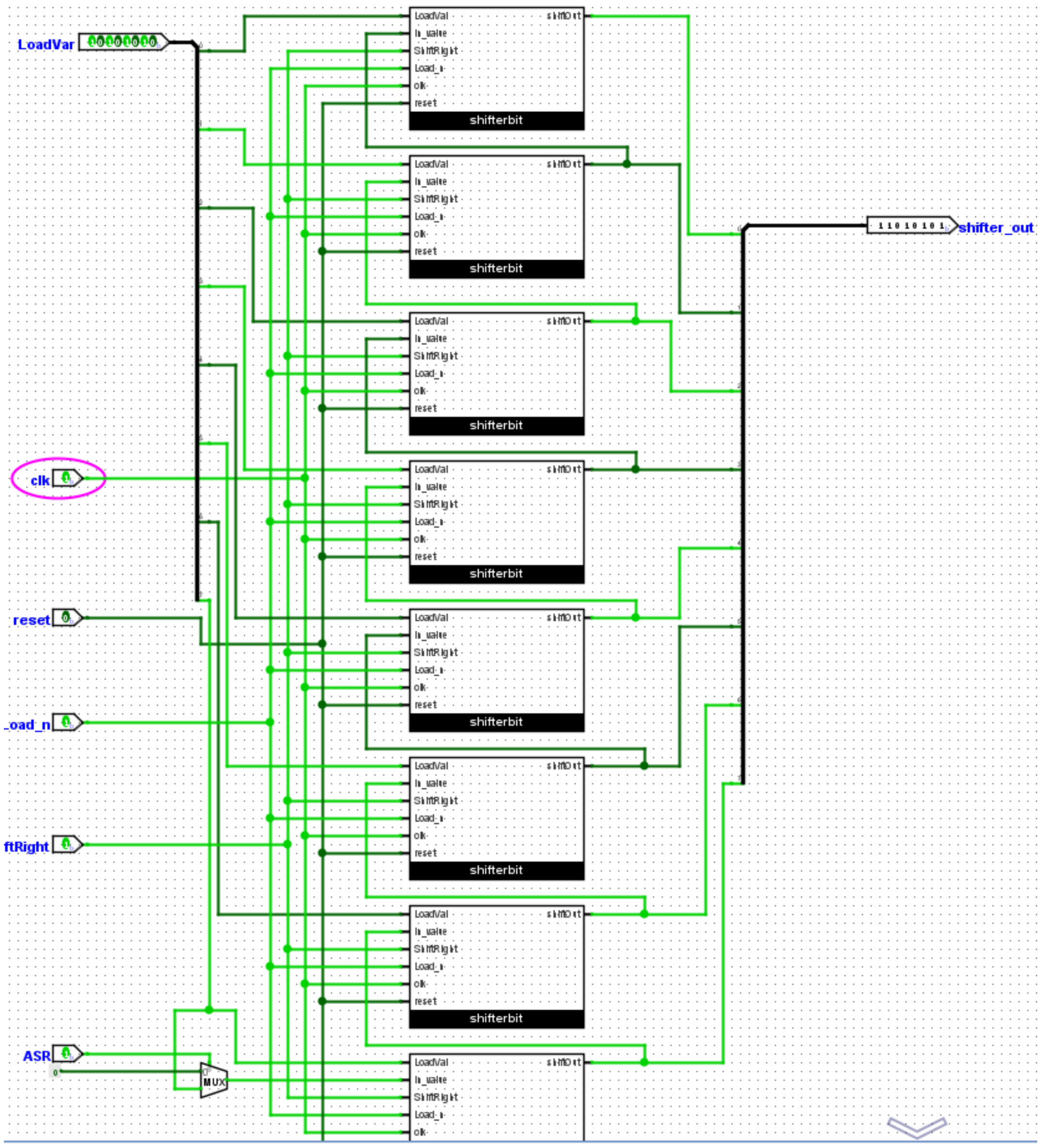
5. then Reset the register



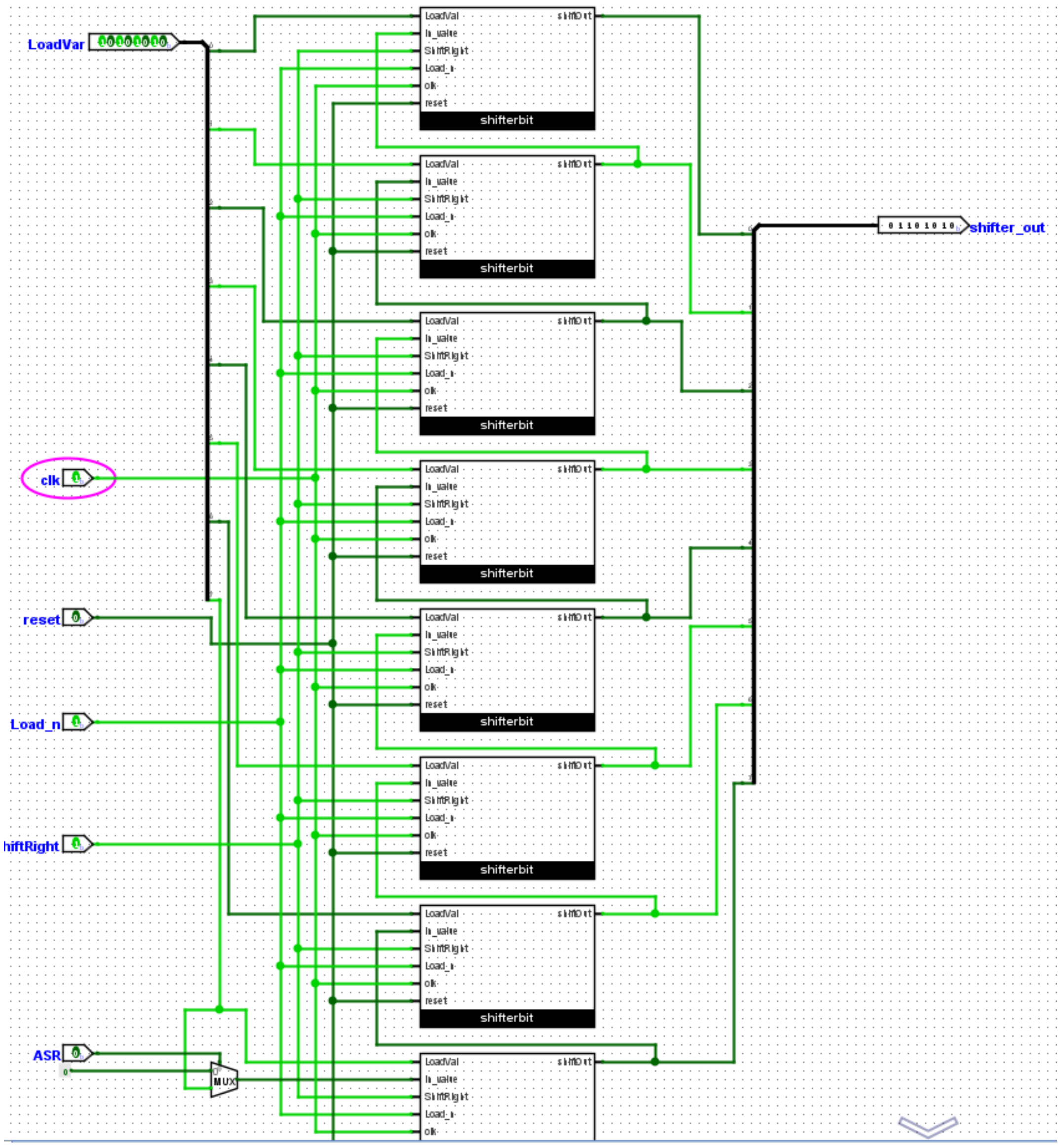
6. then set and Load a signed negative binary value



7. then Arithmetic shift right once



8. then Logic shift right once



LoadVar
10101010
clk
reset
Load_n
hiftRight
ASR
MUX
LoadVal
In_value
ShiftRight
Load_n
clk
reset
shiftOut
shifterbit
01101010
shifter_out