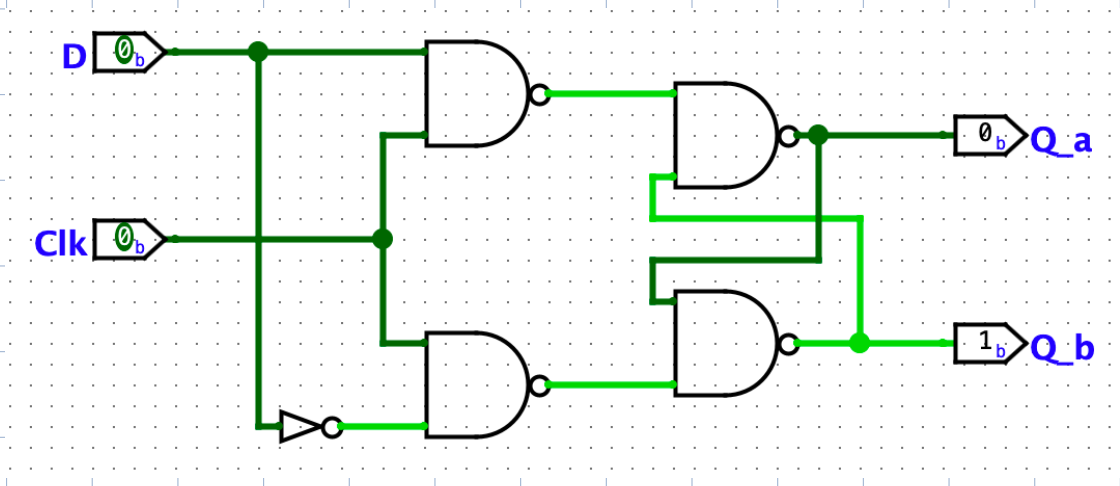
CSC258 Lab4 Pre-Lab Report

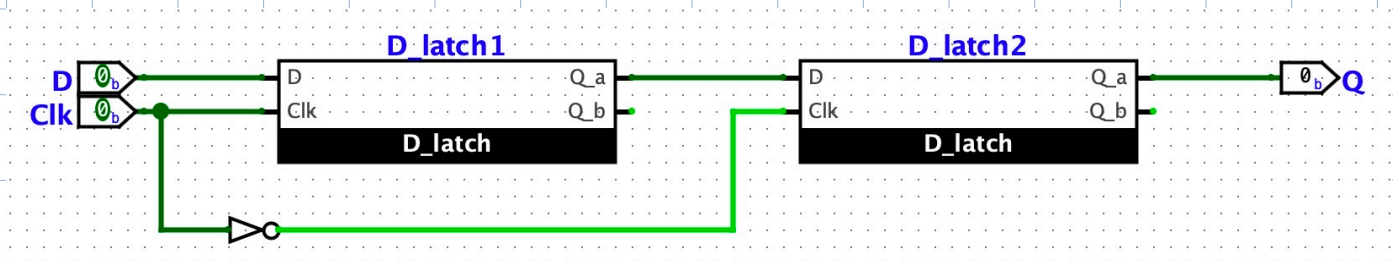
Name: Yulin Wang

Student #: 1003942326

Part I

1.



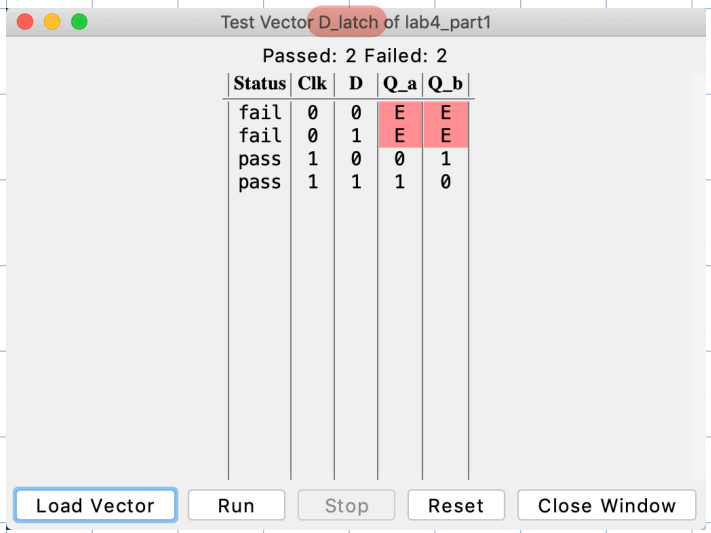


2. Use Poke to study the behaviour of the D latch.

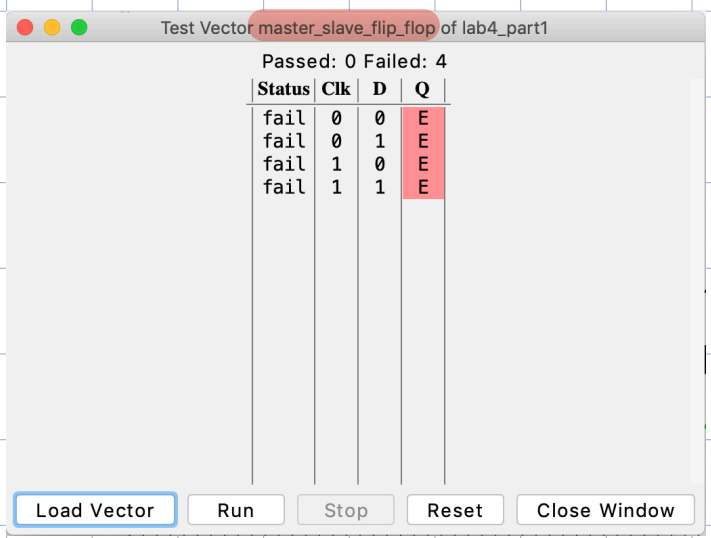
Time	Clk	D	Qa	Qb	
↓	1	0	0	1	
	1	1	1	0	
	0	0	1	0	} Qa, Qb would NOT change when Clk is 0.
	0	1	1	0	

⇒ Qa and Qb would change ONLY when setting Clk to 1.

3



For testing cases with Clk equals to 0, no matter D is 0 or 1, the values of Qa and Qb are unknown ("E").



For all test cases, the value of Q is unknown ("E"). Because for each case, there is one D latch instance with a Clk input with value of 0, but without previous saved Qa or Qb values.

Thus, we could not test them by test vectors.

4. Input combinations with Clk = 0 should NOT be the first test with Poke tool.

Specifically,

Clk	D
0	0
0	1

Since when starting with Clk = 0, the values of Qa and Qb are unknown, which could be either 0 or 1.

Part II

1. Built the logisim module.

2. (a)

The register is used for storing a temporary value, which stores B as an input for the next ALU operation. If we didn't include the register, the ALU output would immediately go straight into ALU input B each time, which would form an infinite loop.

2. (b)

The output would be transparent if use clocked latches, which means the output changes as long as the latch clock is high. However, if use edge-triggered flip-flops, the output changes only when the flip-flop clock rises from low to high.

2. (c)

Assume A, B are both n-bit binary numbers, then the maximum number of them would be $2^n - 1$.

Then the largest value of their multiplication result would be:

$$(2^n - 1)^2 = 2^{2n} - 2^{n+1} + 1$$

$$< 2^{2n}$$

Thus, 2n bits are needed to store the result.

2. (d)

The ALU output would be all 0's, that is, 0000 0000.

3 (a)

Test Vector ALU of lab4_part2

Passed: 26 Failed: 0

Status	A	B	function_value	ALU_output
pass	0000	0000	000	0000 0001
pass	0010	0101	000	0000 0011
pass	1111	1111	000	0001 0000
pass	0111	0000	001	0000 0111
pass	0101	0101	001	0000 1010
pass	1111	1111	001	0001 1110
pass	0111	0000	010	0000 0111
pass	0101	0101	010	0000 1010
pass	1111	1111	010	0001 1110
pass	0000	1111	011	1111 1111
pass	1111	1111	011	1111 0000
pass	0101	1111	011	1111 1010
pass	0000	0000	100	0000 0000
pass	1000	0000	100	0000 0001
pass	1111	1111	100	0000 0001
pass	0001	1111	101	0001 1110
pass	0010	1111	101	0011 1100
pass	1000	1111	101	0000 0000
pass	1001	1111	101	0000 0000
pass	0001	1111	110	0000 0111
pass	0010	1111	110	0000 0011
pass	1000	1111	110	0000 0000
pass	1001	1111	110	0000 0000
pass	0000	1111	111	0000 0000
pass	0001	0001	111	0000 0001
pass	0101	0010	111	0000 1010

Load Vector | Run | Stop | Reset | Close Window

3. (b)

	A	ALU_output
Function 0:	0000	0000 0001
	0010	0000 0011
	0000	0000 0001
Function 1: OR Function 2	0001	0000 0001
	0010	0000 0011
	0001	0000 0010
	0011	0000 0111
Function 3:	0001	0001 0001
	0001	0001 0000
	1111	1111 1111
	0101	1111 1010

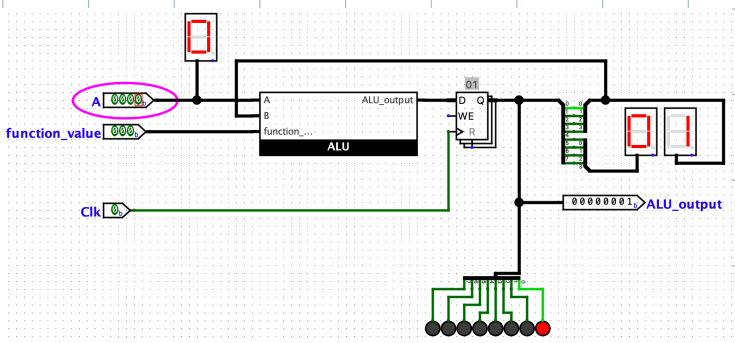
	A	ALU_output
Function 4:	0000	0000 0000
	0001	0000 0001
	0000	0000 0001
	1111	0000 0001
Function 5:	0000	0000 0010
	0001	0000 0010
	0010	0001 0000
	0000	0000 0000
	0001	0000 0000

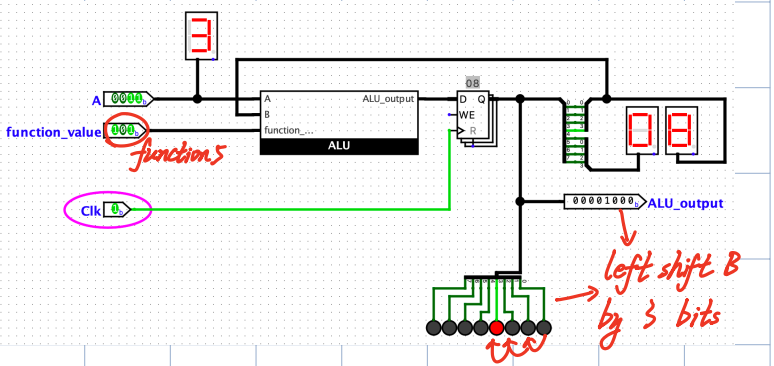
	A	ALU_output
Function 6:	0000	0000 1000
	0001	0000 0100
	0000	0000 0100
	0001	000 0010

	A	ALU_output
Function 7:	0001	0000 0010
	0010	0000 0100
	0010	0000 1000
	0010	000 1 0000

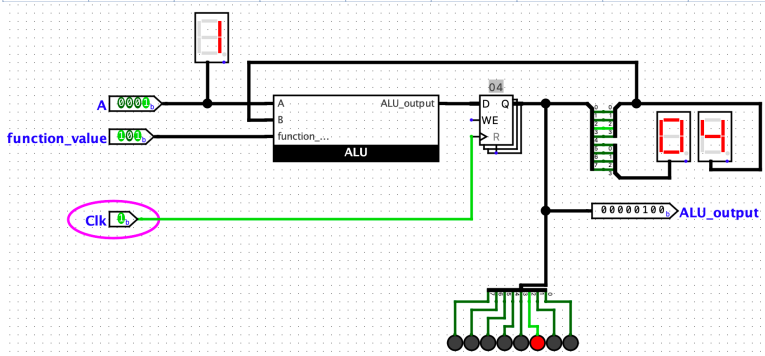
3.(c)

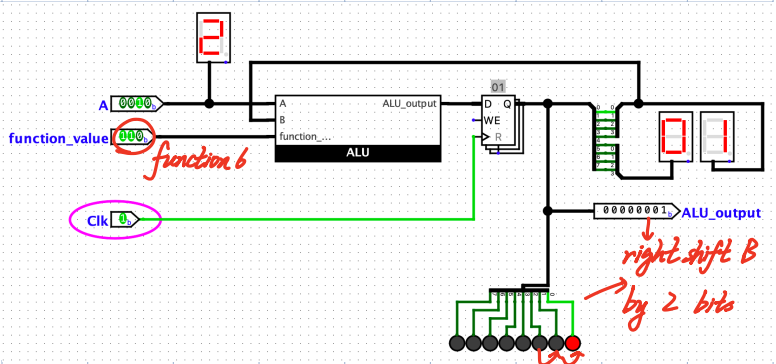
function 5: Set the ALU_output to 0000 0001 first. Then perform function 5 with A = 0011. (3_{10})



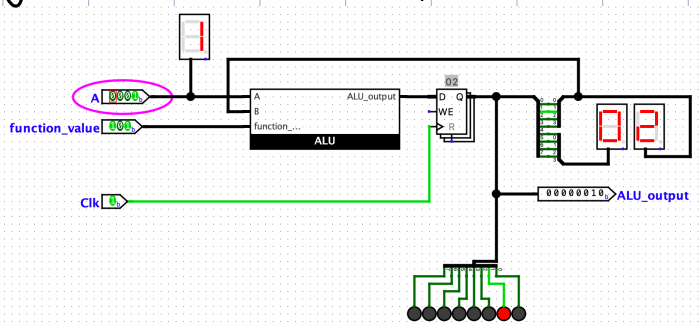


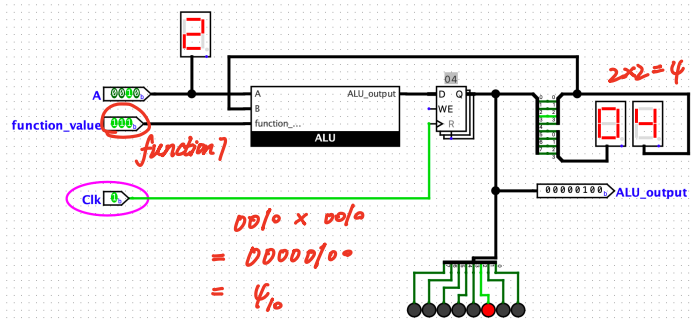
function 6: Set the ALU_output to 0000 0100 first. Then perform function 6 with A = 0010. (2_{10})





function 7: Set the ALU_output to 0000 0010 first. Then perform function 7 with A = 0010. (2_{10})



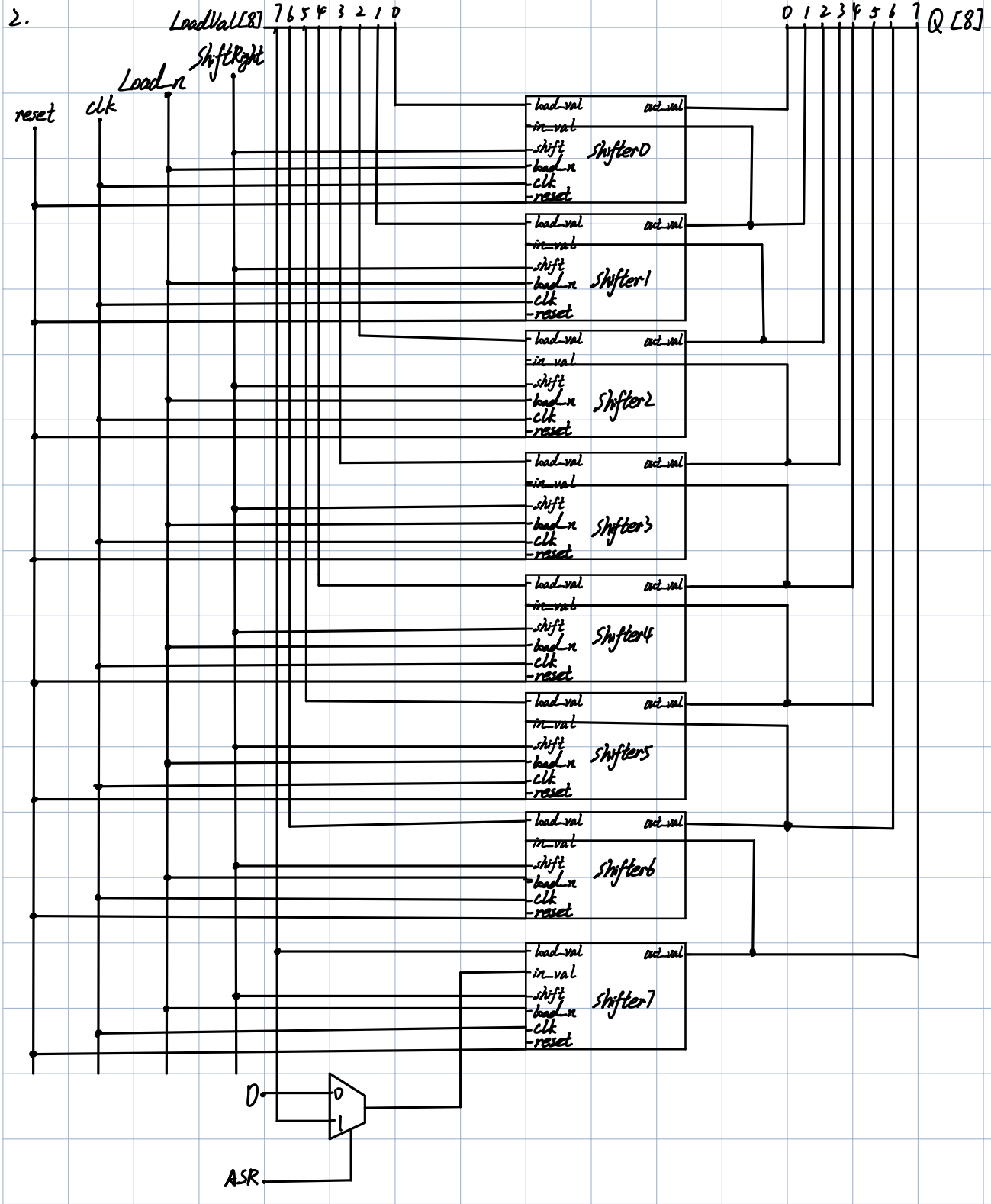


Part III

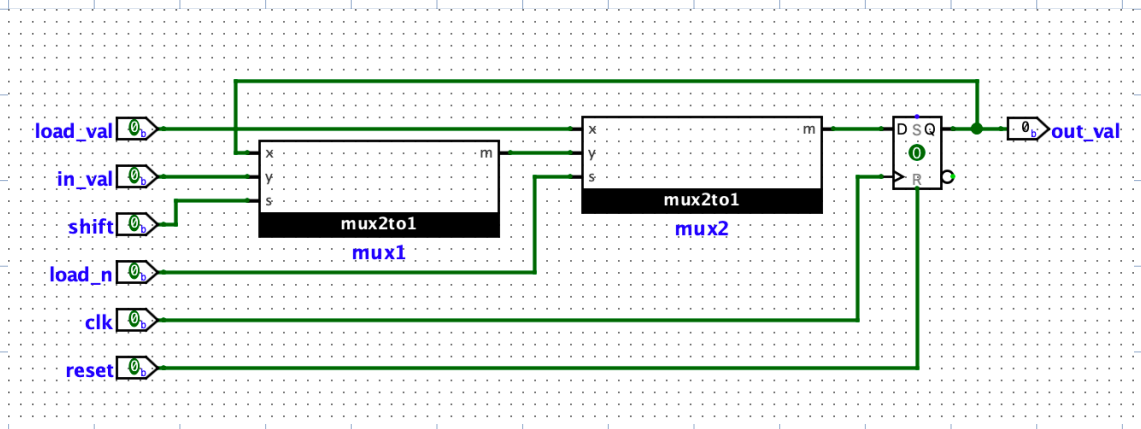
1.

When ShiftRight = 0, the output would NOT change on every positive clock edge.

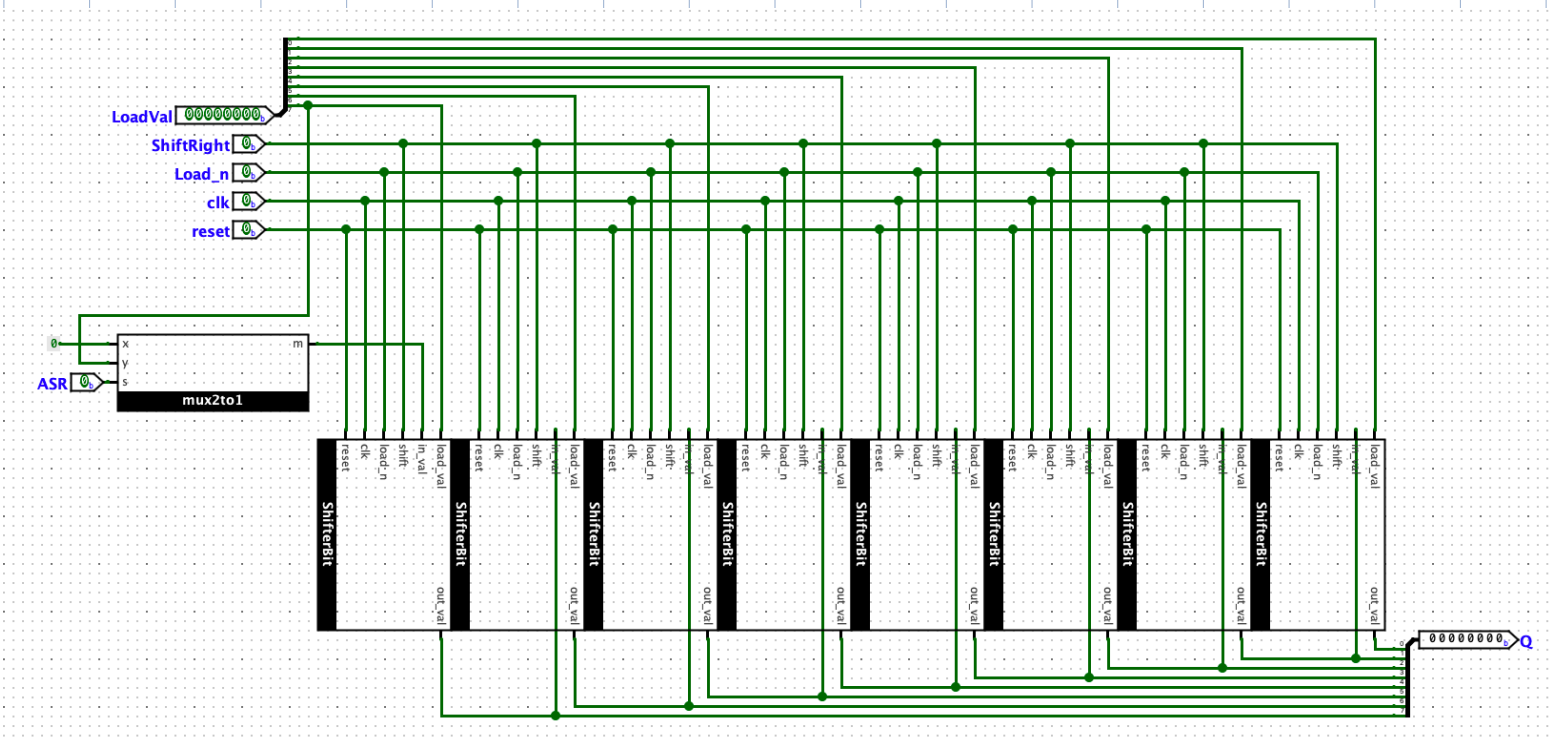
2.



3


load_val
in_val
shift
load_n
clk
reset
x
y
s
m
mux2to1
mux1
mux2to1
mux2
D S Q
out_val

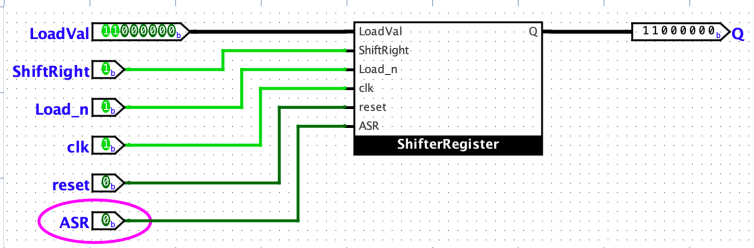
4.


LoadVal
ShiftRight
Load_n
clk
reset
ASR
mux2to1
ShifterBit
reset
clk
load_n
shift
in_val
load_val
out_val
00000000
Q

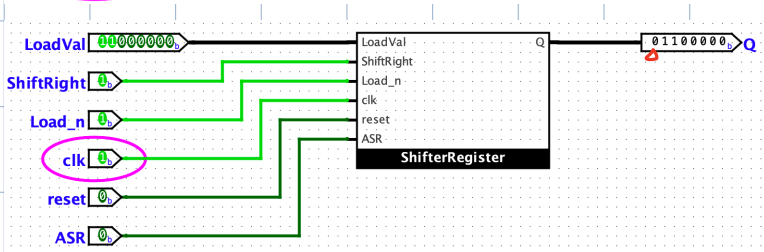
5.

Case 1: Load_n = 1, shiftRight = 1, ASR = 0; it has logic right shift.

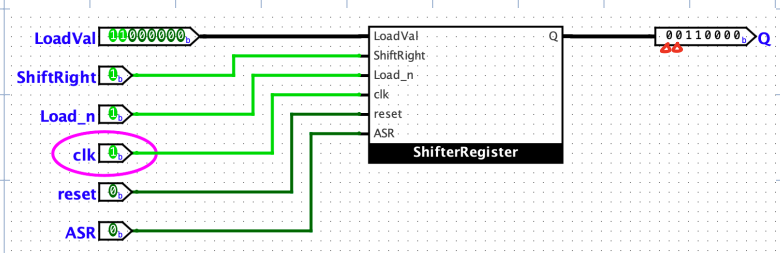
Cycle 0:



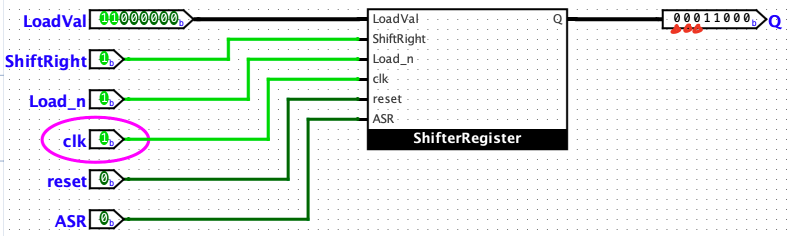
Cycle 1:



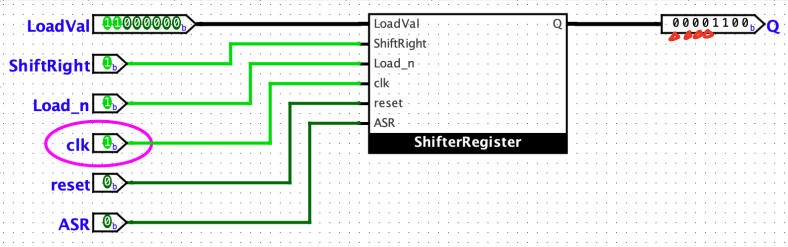
Cycle 2:



Cycle 3:

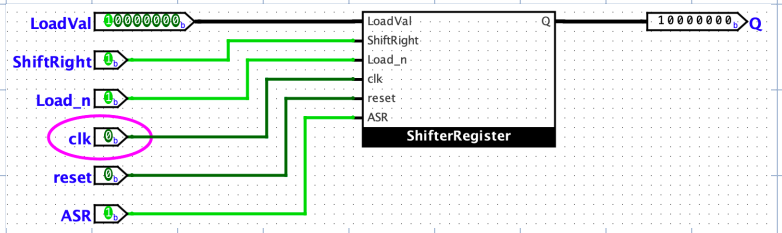


Cycle 4:

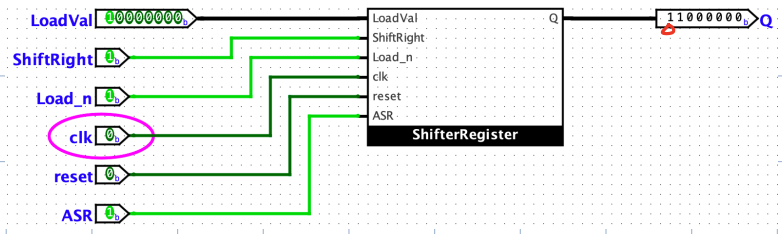


Case2: Load_n = 1, ShiftRight = 1, ASR = 1; it has arithmetic right shift.

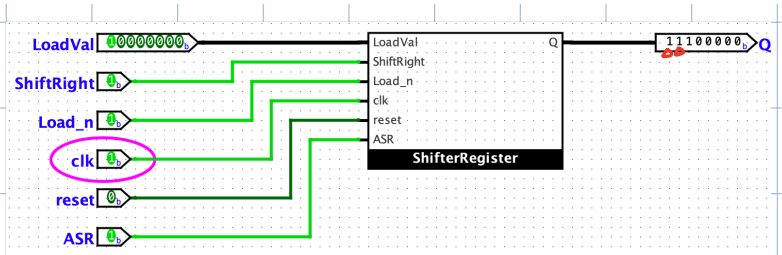
Cycle 0:



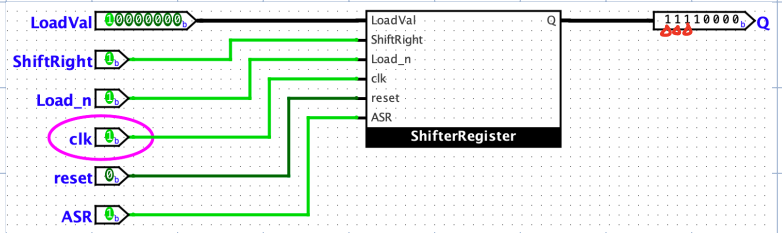
Cycle 1:



Cycle 2:



Cycle 3:



Cycle 4: