실험 5. ALU랑 JK 플립플롭

20210774 김주은

1. 개요

이번 실습의 목표는 컴퓨터의 기초가 되는 산술 논리 장치(ALU)와 정보를 저장할 수 있는 JK 플립플롭(JK Flip-flop)을 구현한다. ALU를 구현할 때 이전에 배웠던 adder와 MUX 등을 활용하여 구현한다. 뿐만 아니라, JK flip-flop라는 새로운 sequential logic에 대해 정확히 이해한 후 이를 잘 반영하는 Master Slave flip-flop을 구현한다. 추가적으로 master slave flip flop 에서 reset input이 추가된 경우 어떻게 구현되는 지, 그리고 그 reset input이 negative reset일 때는 어떻게 구현되는지에 대해 이해하고 구현한다.

2. 이론적 배경

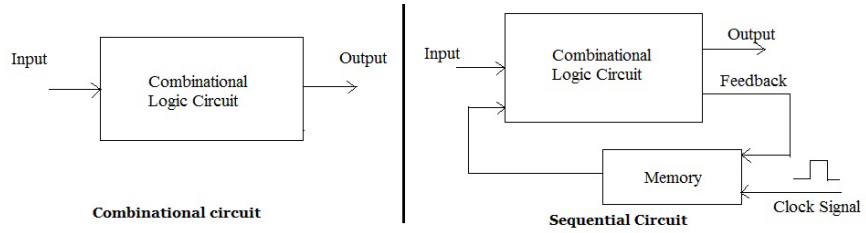
1) ALU

ALU는 선택 입력에 따라 데이터 입력된 것들을 사용하여 여러 산술과 논리 연산을 해주는 기능이다. 연산의 종류의 따라 arthimetic이나 logic 연산을 사용하는 것이기 때문에 ALU라는 이름을 붙인다. 산술연산과 논리연산에는 다음과 같은 종류가 있다.

S_3	S_2	S_1	S_0	Operation
0	0	0	0	G = X
0	0	0	1	G = X + 1
0	0	1	0	G = X + Y
0	0	1	1	G = X + Y + 1
0	1	0	0	G = X + Y'
0	1	0	1	G = X + Y' + 1
0	1	1	0	G = X - 1
0	1	1	1	G = X
1	x	0	0	G = X and Y
1	x	0	1	G = X or Y
1	x	1	0	G = X ⊕ Y
1	x	1	1	G = X'

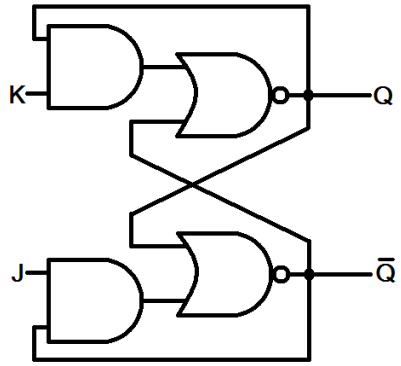
2) combinational logic, sequential logic

combinational logic은 현재 input에 대한 output이 무조건 하나여야 하고, 하나의 input에 대한 output의 종류는 하나다. 하지만, sequential logic은 현재의 input에 대한 output이 여러 개가 될 수 있다. Output이 다시 피드백하여 logic에 연결되어 영향을 준다.



3) JK latch

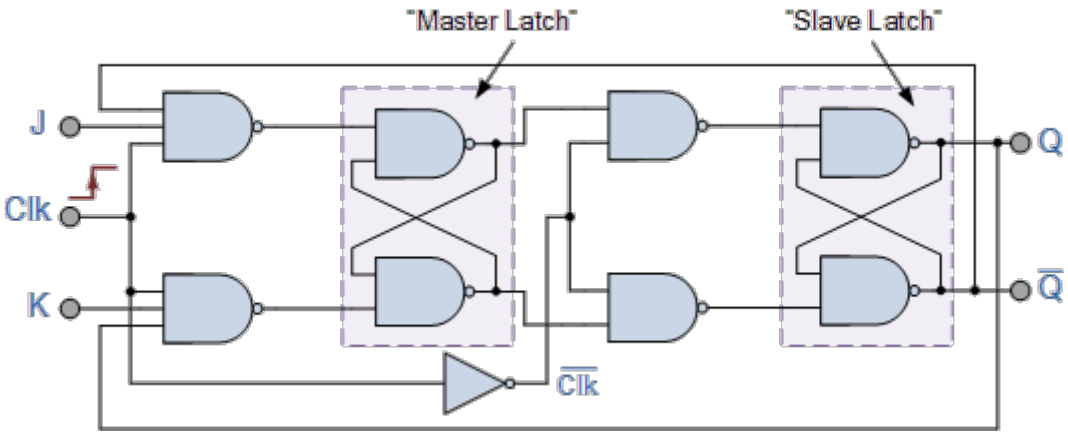
JK latch는 SR latch와 비슷하게 동작하나 forbidden한 부분이 존재하지 않는다. SR latch는 S와 R이 1인 경우는 Q와 Q'가 모두 0으로 출력되어 정상적으로 동작하지 않는다. 하지만 JK latch인 경우 J와 K가 S와 R에 대응되고, J와 K가 모두 1인 경우 toggle이 된다.



하지만 latch의 경우 race condition이 발생하는 문제가 있고, 이를 해결하기 위해 Flip Flop이 존재한다.

4) master-slave JK flip flop

Master slave JK flip flop은 SR latch가 두 개가 들어가며 하나는 master, 그리고 다른 하나는 slave로 기능하여 master slave jk flip flop이라고 불린다. Flip flop의 경우 latch와 비슷하게 동작하지만 ck input이 추가적으로 존재한다. 하지만 master slave jk flip flop의 경우 latch에서 존재하던 race condition은 해결했지만 glitch가 발생했을 때 이를 무시하지 않고 영향을 받아 결과값이 바뀔 수 있는 문제가 있다.



3. 실험 준비

Lab 5-1)

$S_3=0$ (산술)

S_2	S_1	S_0	동작	A	B	C_{in}
0	0	0	x	0000	x	0
0	0	1	x+1	0000	x	1
0	1	0	x+y	y	x	0
0	1	1	x+y+1	y	x	1
1	0	0	x+$\bar{y}$	$\bar{y}$	x	0
1	0	1	x+$\bar{y}$+1	$\bar{y}$	x	1
1	1	0	x−1	1111	x	0
1	1	1	x	1111	x	1

S_2	S_1	A
0	0	0000
0	1	y
1	0	$\bar{y}$
1	1	1111

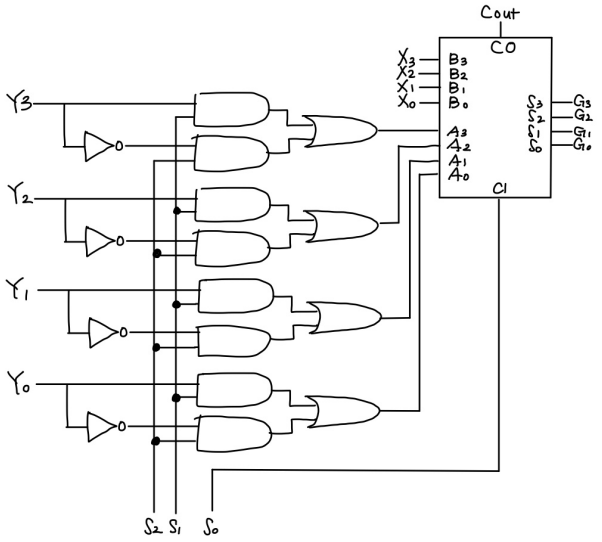
→

S_2	S_1	y_i	A_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

→

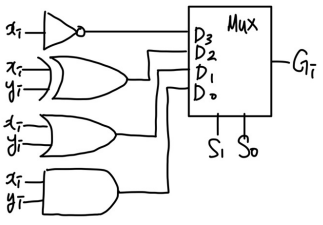
S_2 \ S_1y_i	00	01	11	10
0	0	0	1	0
1	1	0	1	1

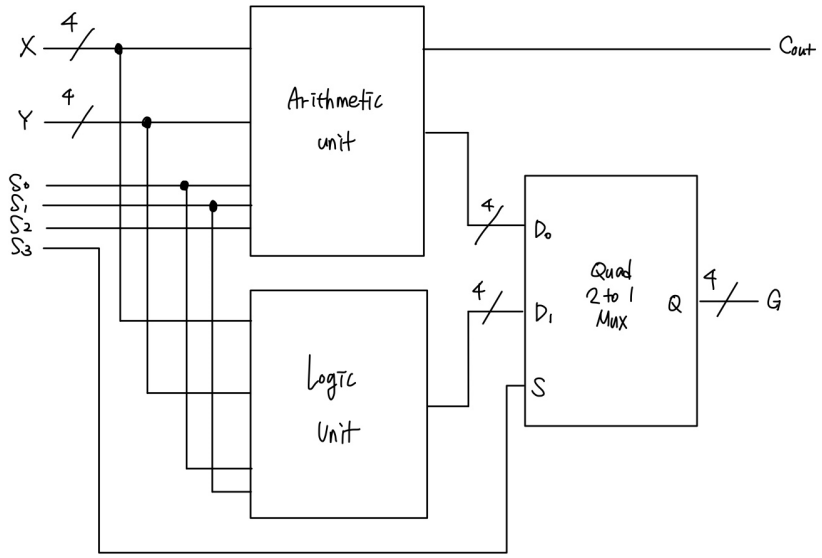
$\Rightarrow A_i = S_2\bar{y_i} + S_1 y_i$



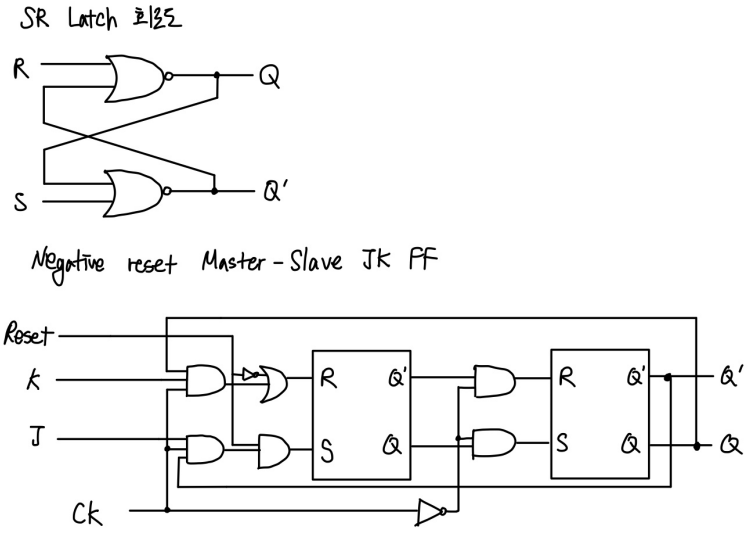
$S_3 = 1$

S_2	S_1	S_0	G_i
X	0	0	$x_i y_i$
X	0	1	$x_i + y_i$
X	1	0	$x_i \oplus y_i$
X	1	1	x_i'





Lab 5-2)

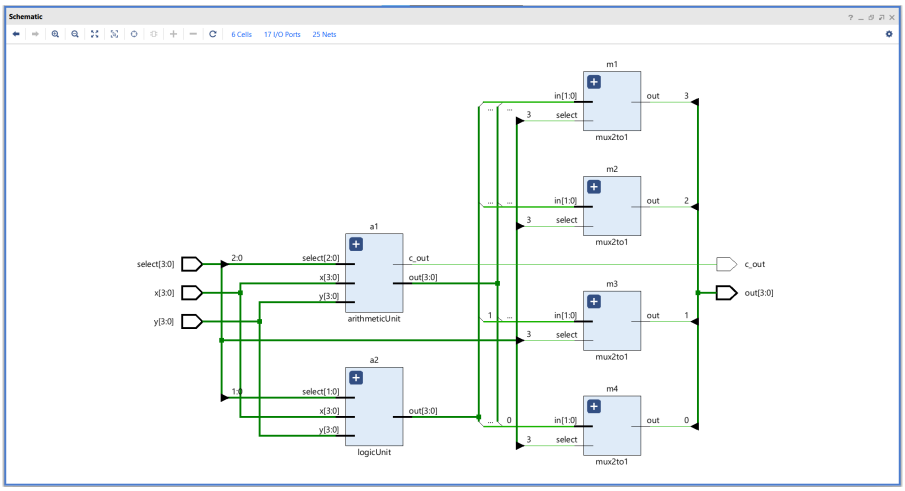


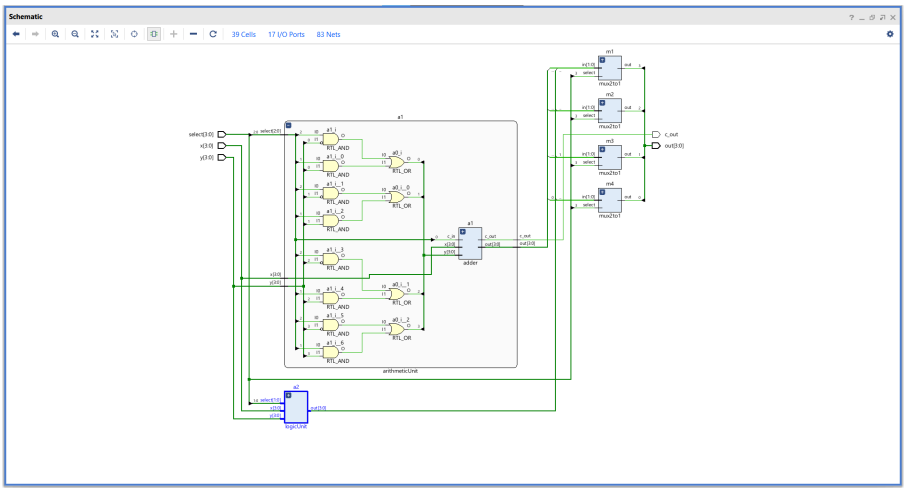
SR latch와 비교해 master slave JK flip flop이 해결 가능한 글리치와 해결 불가능한 글리치

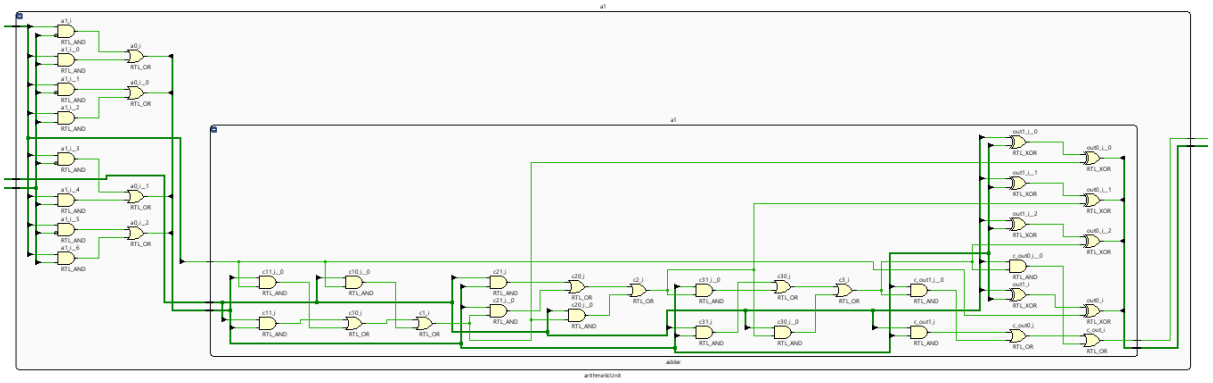
- 해결 가능한 glitch: 해결 가능한 glitch의 경우 ck input이 0인 상황에서 glitch가 발생했을 때는 이를 무시하고 입력값에 영향을 받지 않아 문제가 해결된다. (뿐만 아니라 glitch의 문제가 아니더라도 S와 R의 input이 1,1 이었다가 0,0으로 되었을 때 race condition이 생기는 것은 master slave jk flip flop이 해결 가능하다.)
- 해결 불가능한 glitch: flip flop에서 ck input이 1인 상황에서 glitch가 발생했을 때, 이를 무시하지 않고 영향을 받아 결과값을 출력하는 문제이다.

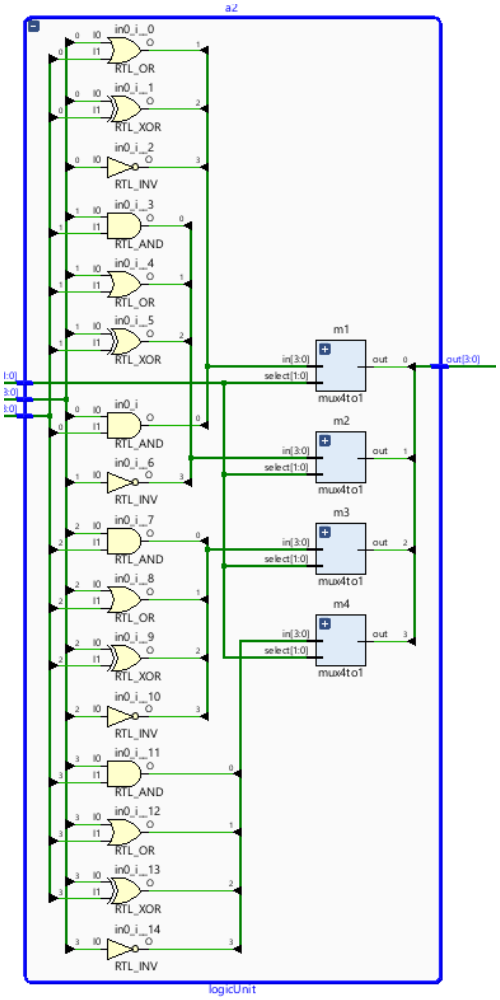
4. 결과

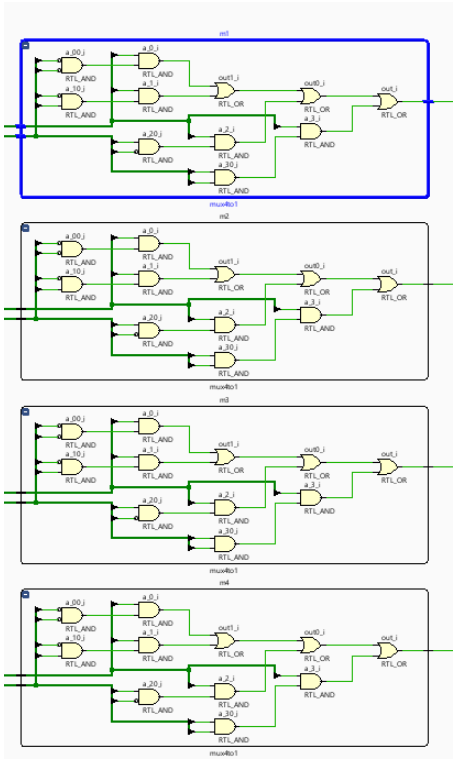
Lab 5-1)

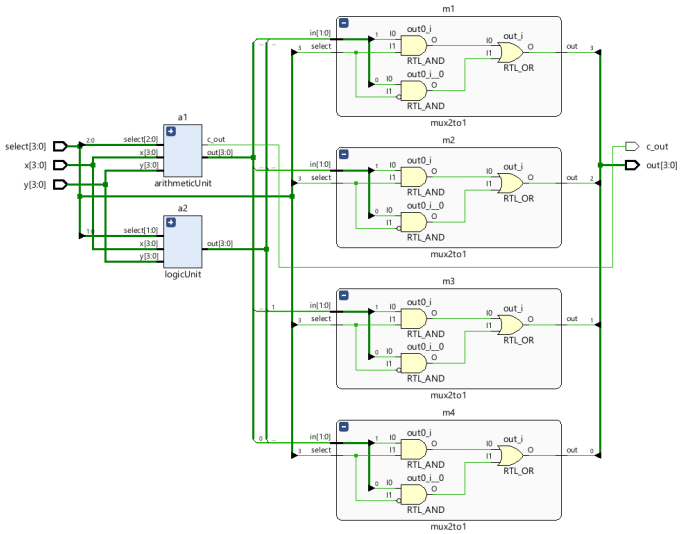

Schematic
6 Cells
17 I/O Ports
25 Nets
m1
m2
m3
m4
in[1:0]
select
out
mux2to1
a1
a2
select[3:0]
x[3:0]
y[3:0]
select[2:0]
select[1:0]
c_out
out[3:0]
arithmeticUnit
logicUnit

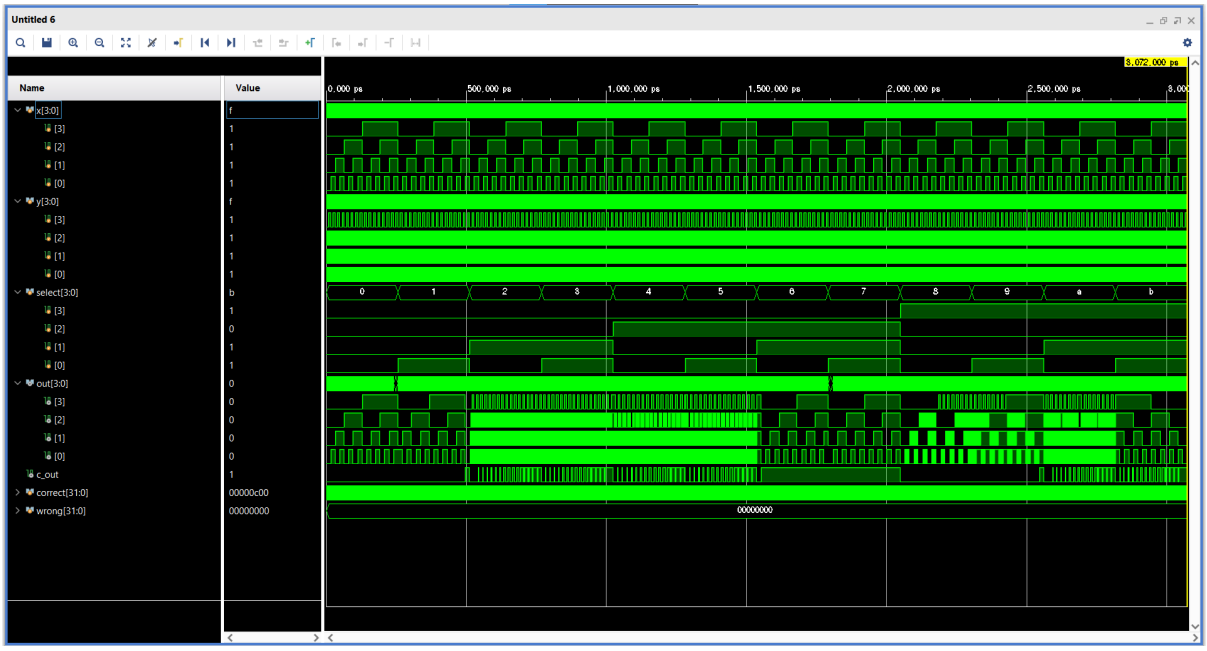

Schematic
39 Cells
17 I/O Ports
83 Nets

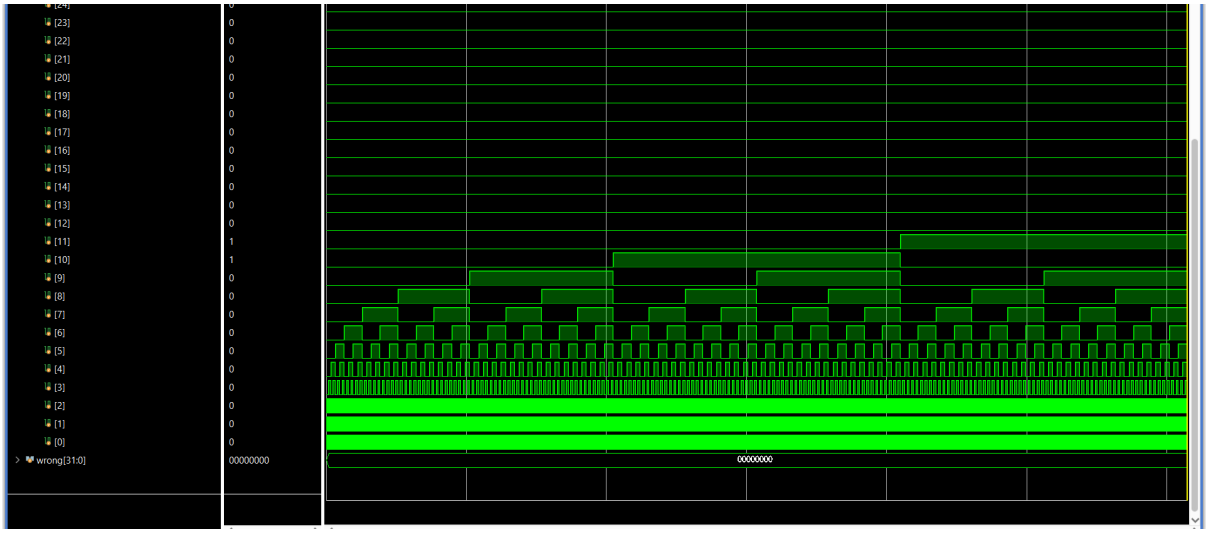



a2
in0_i__0
RTL_OR
in0_i__1
RTL_XOR
in0_i__2
RTL_INV
in0_i__3
RTL_AND
in0_i__4
RTL_OR
in0_i__5
RTL_XOR
m1
in[3:0]
select[1:0]
out
mux4to1
out[3:0]
in0_i
RTL_AND
in0_i__6
RTL_INV
m2
in0_i__7
RTL_AND
in0_i__8
RTL_OR
m3
in0_i__9
RTL_XOR
in0_i__10
RTL_INV
m4
in0_i__11
RTL_AND
in0_i__12
RTL_OR
in0_i__13
RTL_XOR
in0_i__14
RTL_INV
logicUnit

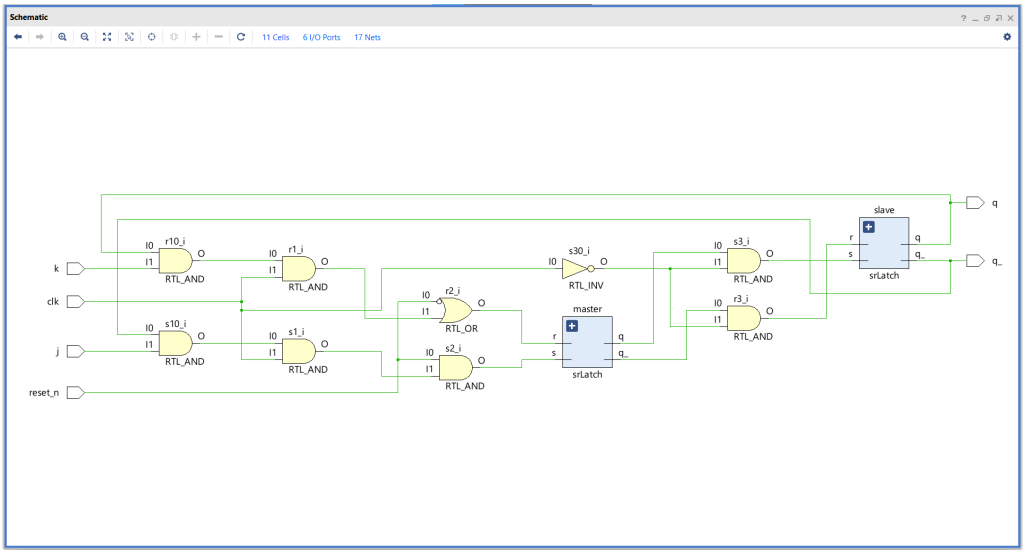

m1
m2
m3
m4
a_00_i
a_10_i
a_0_i
a_1_i
a_20_i
a_30_i
a_2_i
a_3_i
out1_i
out0_i
out_i
RTL_AND
RTL_OR
mux4to1

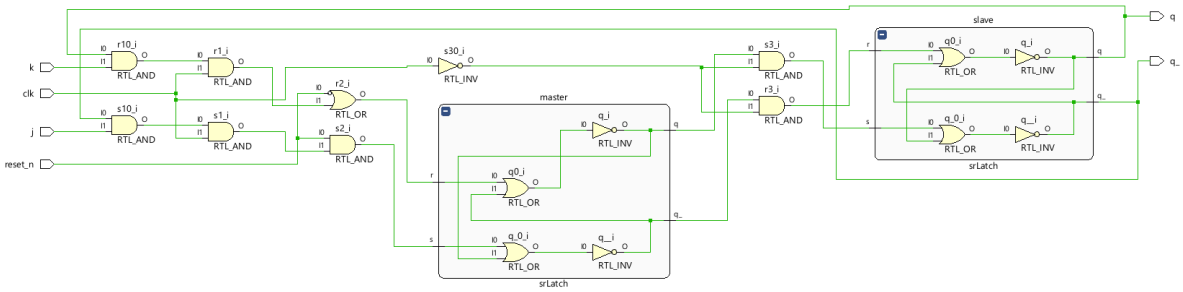

m1
m2
m3
m4
mux2to1
RTL_AND
RTL_OR
a1
a2
arithmeticUnit
logicUnit
select[3:0]
x[3:0]
y[3:0]
c_out
out[3:0]

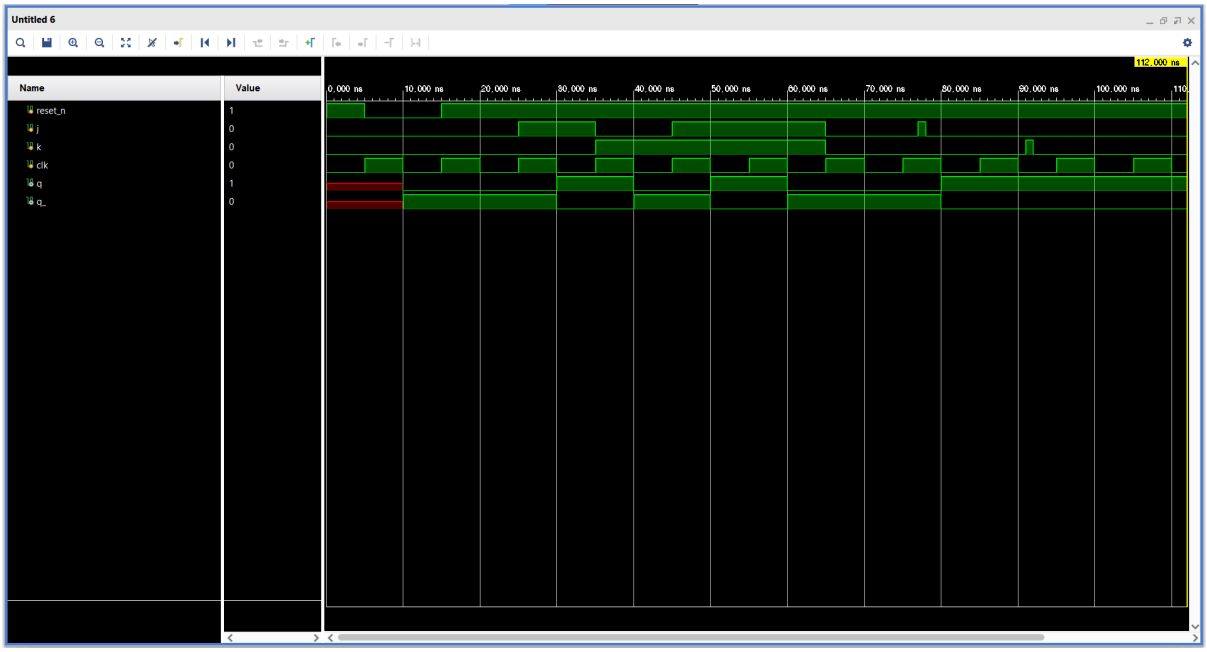

Untitled 6
Name
Value
3,072.000 ps
0.000 ps
500.000 ps
1,000.000 ps
1,500.000 ps
2,000.000 ps
2,500.000 ps
x[3:0]
y[3:0]
select[3:0]
out[3:0]
c_out
correct[31:0]
wrong[31:0]
00000c00
00000000


wrong[31:0]
00000000

Lab 5-2)







위의 결과값과 같이, ck 이 0일 때는 glitch가 발생되어도 무시되지만 ck이 1일때는 glitch가 발생되면 무시되지 않고 출력값이 영향을 받는다.

5. 논의

이번 랩은 처음으로 내가 직접 테스트벤치를 구현하는 과제였다. 아직 베릴로그 문법에 익숙치 않아 테스트벤치만 구현하는데 꽤나 며칠이 걸리고 어렵고 힘들었다. 특히나 alu의 테스트벤치의 경우 단순 연산을 적는 것이라 문법을 공부해보고 구현하는 것이 그나마 쉬웠지만 flip flop의 테스트 벤치의 경우 glitch를 어떻게 생성해야 하는지 막막하여 그 부분에서 꽤나 막혔던 것 같다. 그래도 #10ns 와 같은 문법이 시간을 지연시킨다는 뜻임을 깨닫고 ck의 지속 시간인 5ns보다 작은 1ns의 정도의 지연시간인 glitch를 생성시키는 데 성공했다.

비록 이번 테스트벤치가 상당히 어렵고 힘든 부분이었지만 flip flop과 alu의 시뮬레이션을 직접 구성해보면서 더욱 더 구동방식에 대한 이해를 깊이 있게 할 수 있었다.