디지털시스템설계 실습 보고서

5. ALU와 JK 플립플롭

20210716 최대현

1. 실습 개요

이번 실습에서는 컴퓨터의 기초가 되는 산술 논리 장치(ALU)와 정보를 저장하는 JK FlipFlop을 구현한다.

2. 이론적 배경

1) ALU

산술 논리 장치(arithmetic and logical unit, ALU)는 덧셈, 뺄셈 같은 두 숫자의 산술연산과 배타적 논리합, 논리곱, 논리합 같은 논리연산을 계산하는 디지털 회로이다. 산술 논리 장치는 컴퓨터 중앙처리장치(CPU)의 기본 설계 블록이기도 하다.

2) 비동기 회로, 동기 회로

Sequential Diagram 중 combinational Diagram과 clock을 따르지 않는 Sequential Diagram은 비동기 회로다. 한편 동기 회로는 말 그대로 '동기화된' 회로를 의미하며, 즉 다른 회로와 같은 순간에 맞춰 작동하기 위해 Clock 신호를 따르는 회로이다.

3) JK Latch / JK FlipFlop

기존의 SR Latch는 S와 R이 둘 다 1일 때 올바르게 작동하지 않는다는 문제점이 있다. 이를 보완하여 JK Latch는 SR Latch에 추가적인 회로를 더해 S와 R이 동시에 1인 상황에서도 정상적으로 작동하도록 수정한 형태이다. JK Latch에서 J와 K가 동시에 1일 경우 현재 상태에 상관없이 값을 반전시킨다.

한편, Latch가 입력이 바뀔 때 출력도 즉시 바뀌는 비동기 회로라면 플립플롭은 입력이 바뀌더라도 출력이 Clock에 맞추어 반영되는 동기 회로이다. 즉 JK 플립플롭은 Clock 신호를 받아 작동한다. Clock이 1일 때 작동하는 positive-clock flipflop과 clock이 0일 때 작동하는 negative-clock flipflop 두 종류가 존재한다.

4) Master-slave JK FlipFlop

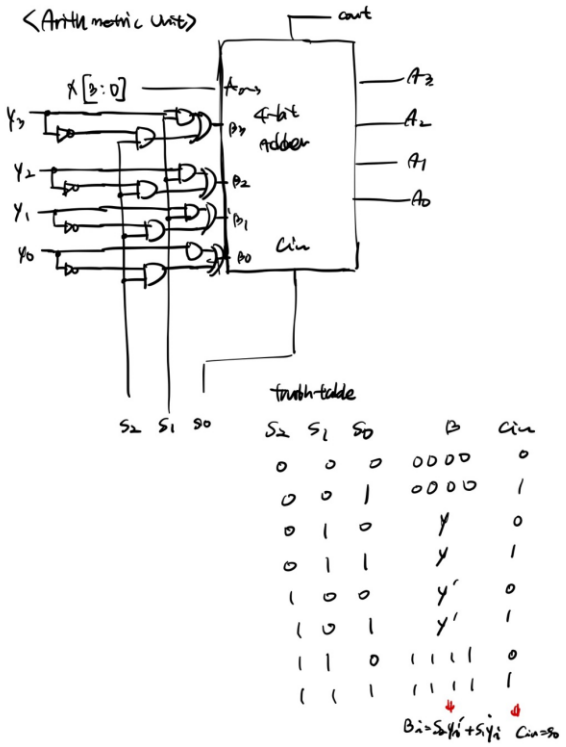
Master-slave JK 플립플롭은 SR Latch 두 개를 연결하여 만든 플립플롭이다. 이 플립플

롭은 Clcok이 1인 동안 Master Latch를 활성화해 입력을 임시로 저장한 뒤 Clock이 0이 되는 순간 Slave Latch로 전달한다. 따라서 Master Latch가 활성화되어있는 동안 glitch로 잠깐 입력값이 생기면 이 값이 Master Latch에 저장되어 있다가 다음 Clock이 0이 되는 순간에 Slave Latch로 전파되는 문제가 있다. 이는 Clock이 1인 동안 계속 입력을 받기 때문에 생기는 문제로 Clock이 0에서 1로, 혹은 1에서 0으로 바뀌는 순간에만 입력을 받는 Edge-trigger diagram을 사용하여 해결할 수 있다

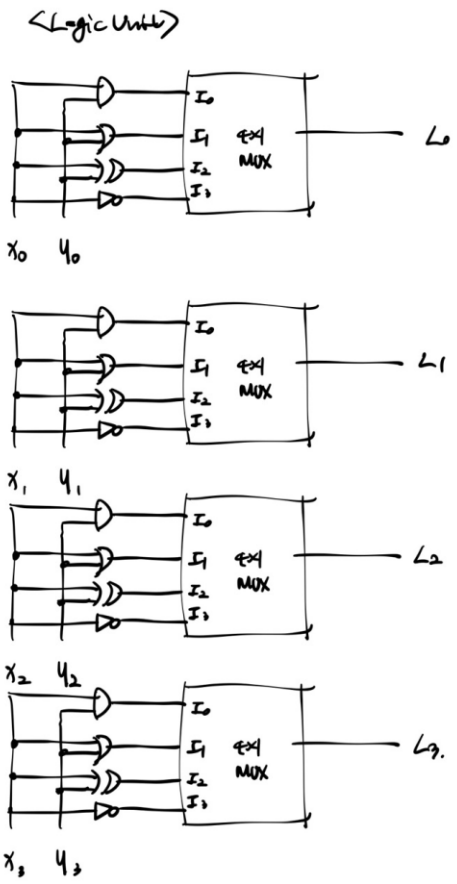
3. 실험 전 준비

1) ALU의 회로도, 식

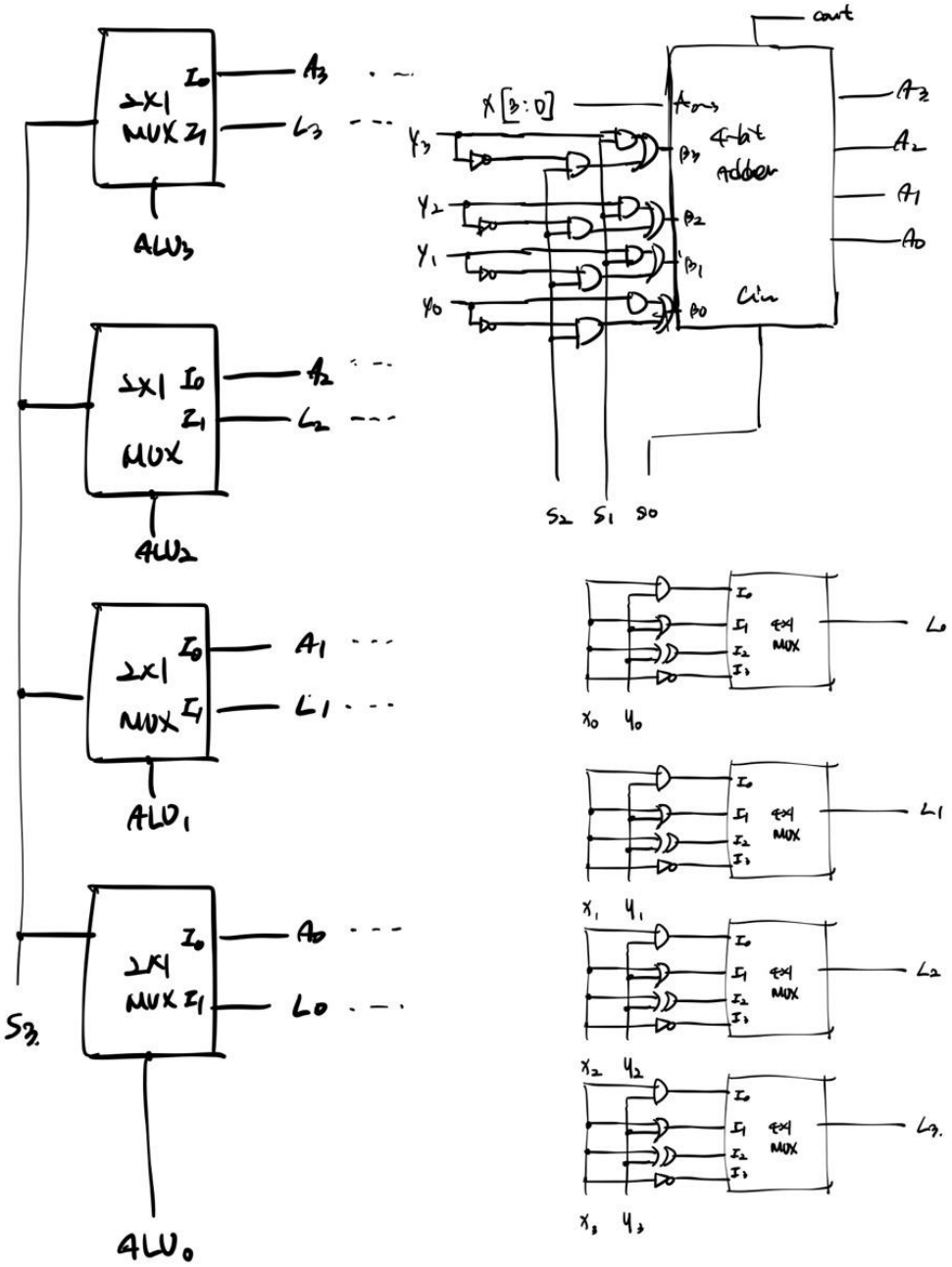
Arithmetic(산술 연산) part



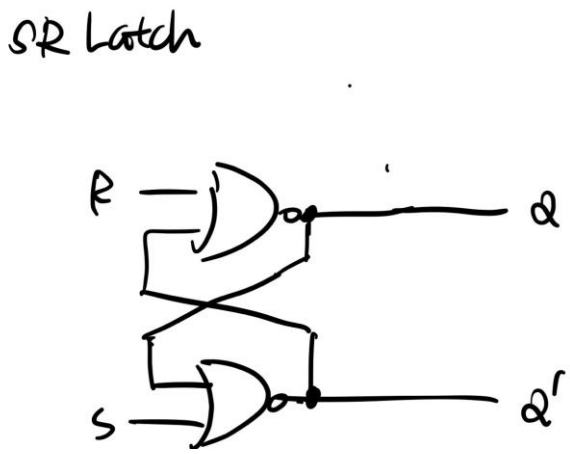
Logic(논리 연산) part



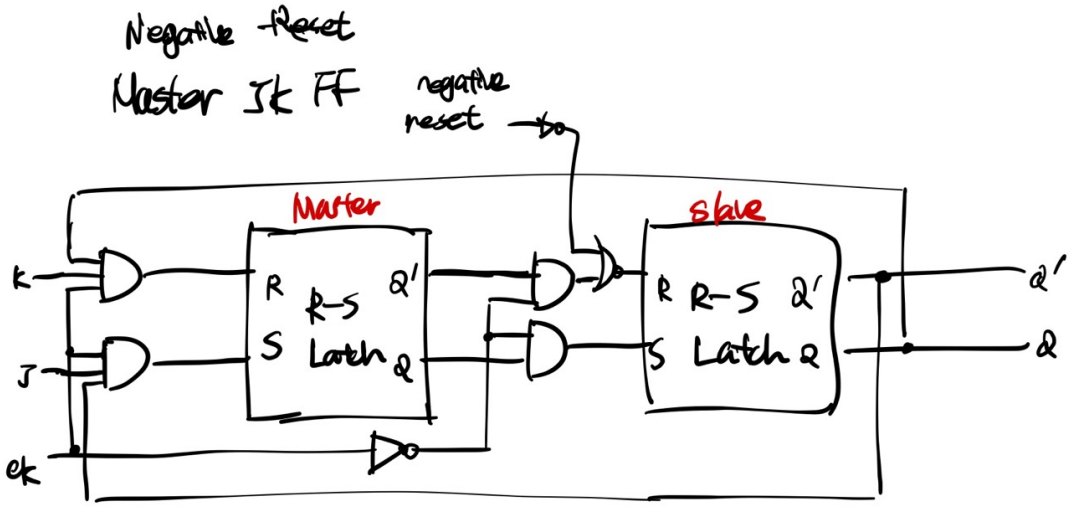
ALU 회로도


2x1 MUX
I0
I1
Z1
A3
L3
ALU3
A2
L2
ALU2
A1
L1
ALU1
A0
L0
ALU0
S3
cout
X[3:0]
Y3
Y2
Y1
Y0
4-bit Adder
B3
B2
B1
B0
Cin
A3
A2
A1
A0
S2 S1 S0
4x1 MUX
I2
I3
L0
X0 Y0
L1
X1 Y1
L2
X2 Y2
L3
X3 Y3

2) SR Latch의 회로도



3) Negative reset Master -slave JK 플립플롭의 회로도

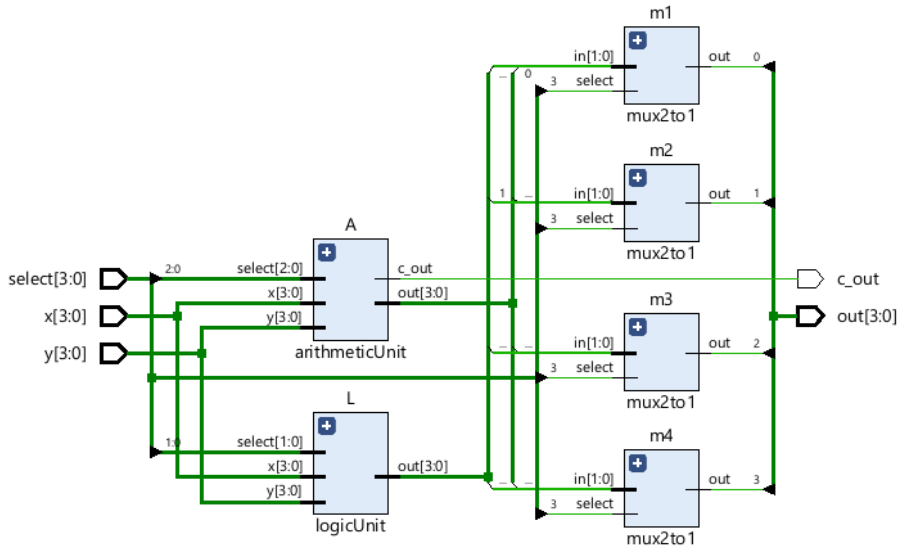


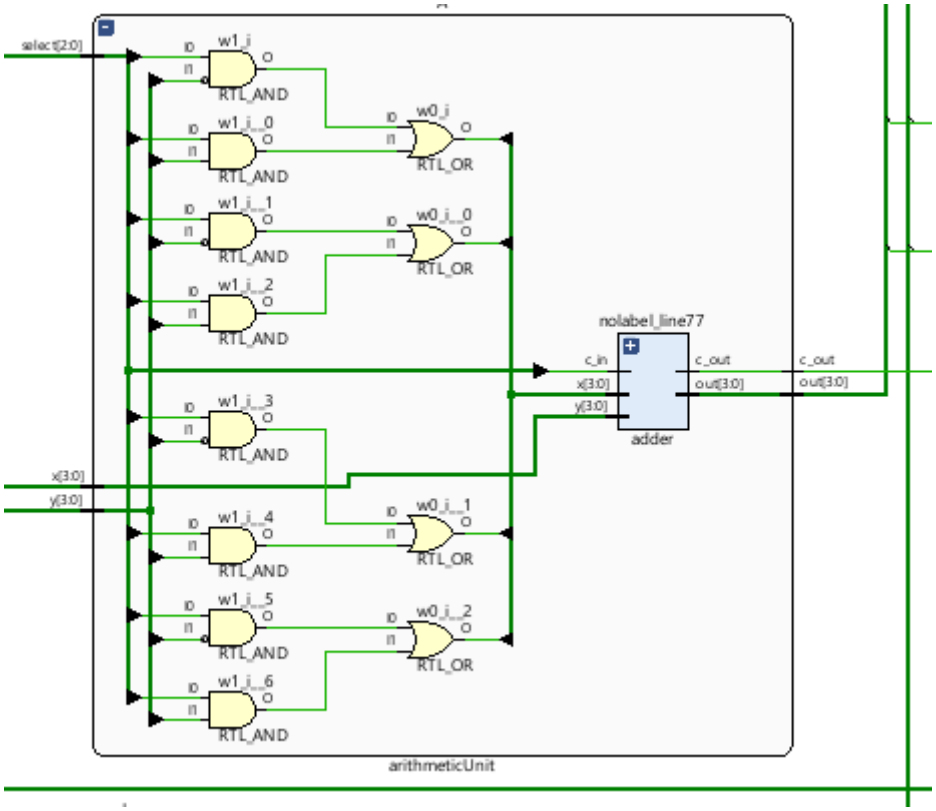
4. 실험

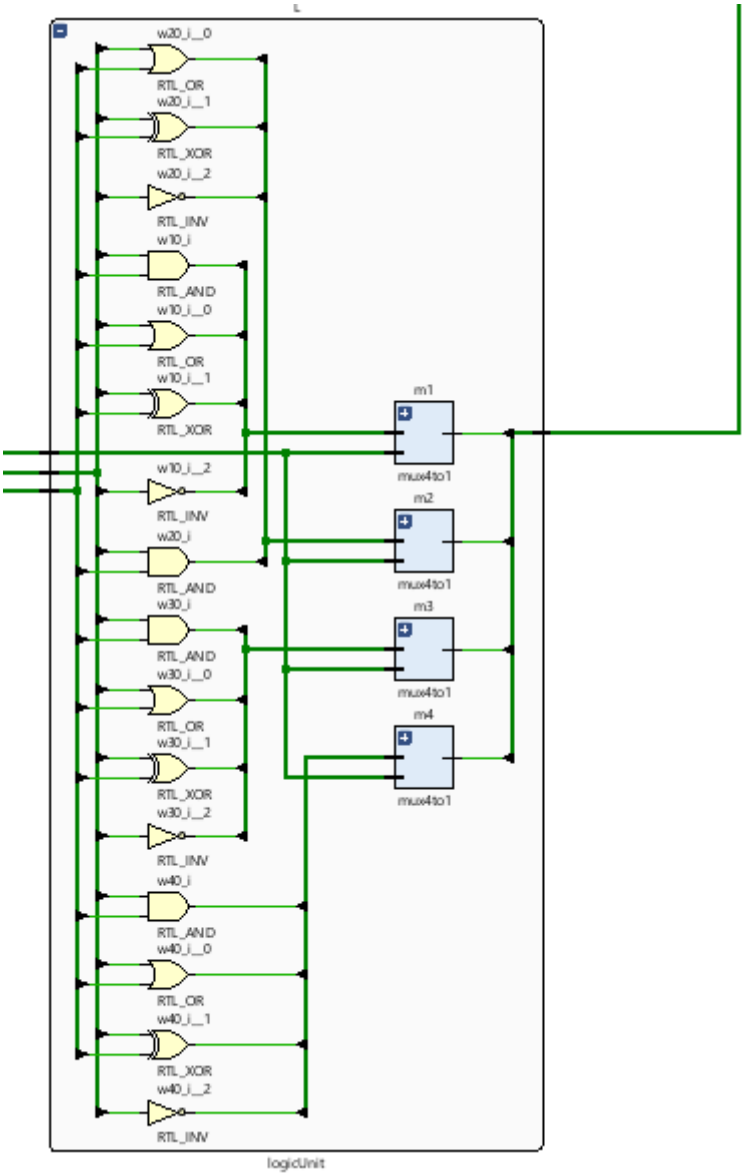
본 실험은 Verilog 프로그래밍으로 회로를 구현하며 진행되었다.

4-1. lab5_1.v

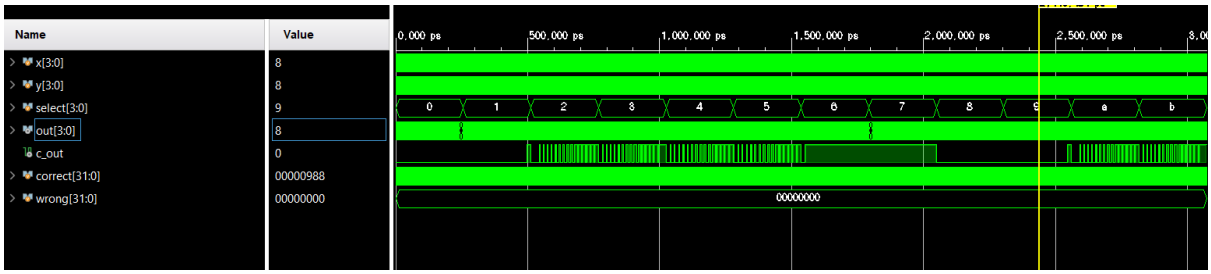
과제 설명에 수록된 표의 기능을 수행하는 ALU를 구현한 코드이다.


m1
in[1:0]
out
select
mux2to1
m2
m3
m4
A
select[3:0]
x[3:0]
y[3:0]
select[2:0]
c_out
out[3:0]
arithmeticUnit
L
select[1:0]
logicUnit
c_out
out[3:0]


select[2:0]
w1_i
w1_i_0
w1_i_1
w1_i_2
w1_i_3
w1_i_4
w1_i_5
w1_i_6
RTL_AND
w0_i
w0_i_0
w0_i_1
w0_i_2
RTL_OR
x[3:0]
y[3:0]
nolabel_line77
c_in
c_out
out[3:0]
adder
arithmeticUnit



<Schematic 기능으로 출력한 회로도>



< simulation 결과 >

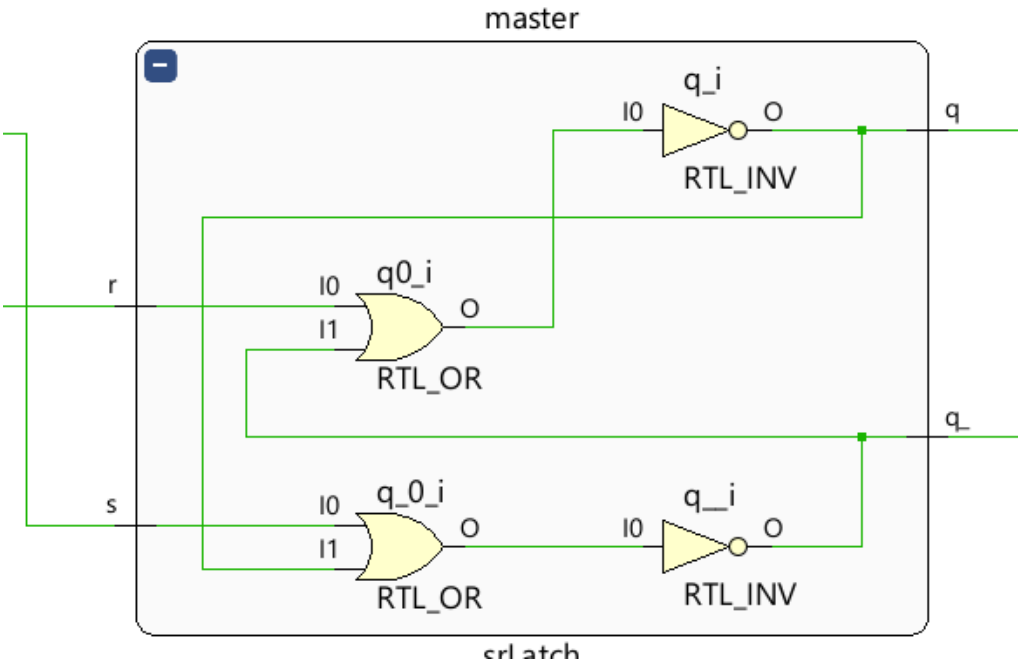
4-2. lab5_2.v

1) S-R Latch를 구현한 코드이다.

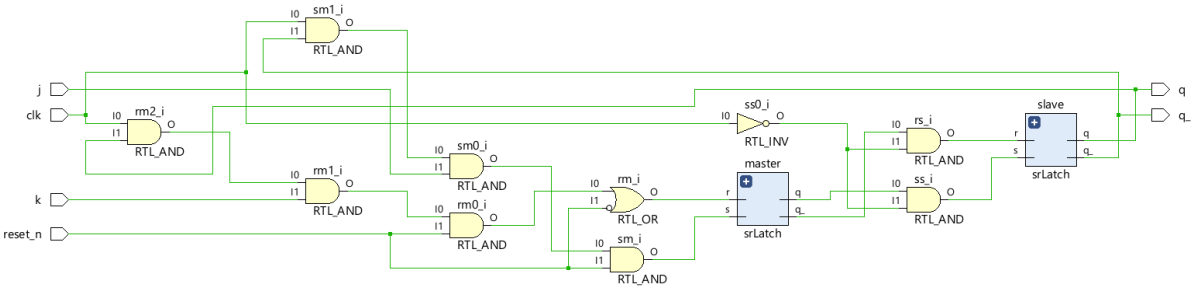
```
module srLatch(
    input s, r,
    output q, q_
    );

    //////////////////////////
     nor(q, r, q_);
     nor(q_, s, q);
    /* Add your code here */
    //////////////////////////
```

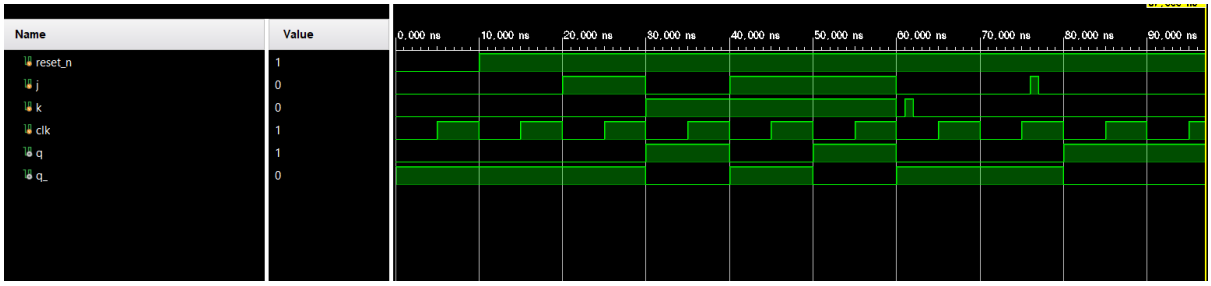
<SR Latch - source code>



<SR Latch- Schematic>



< JK flip flop with negative reset - Schematic >



<simulation 결과>

Simulation 분석

기존의 S-R latch에서는 모든 noise에 대해 (무시해야 하는 noise에 대해) 결과값인 q가 영향을 받는 문제가 있었다. 하지만 clock을 탑재하고 2개의 S-R latch을 이어 붙인 master-slave jk flipflop은 clock값이 0일 때 noise를 무시하는 것을 확인할 수 있다. (61ns 구간에서 k noise를 무시함)

그러나, 여전히 **1's catch**라는 현상이 발생한다. 즉, clock값이 high일 때 J 또는 K에서 glitch가 발생할 경우 문제가 생긴다. (76ns 구간에서 j noise를 무시함) Clock값이 0일 때는 문제가 되지 않지만, Clock값이 1인 상태에서 0->1->0으로 J 또는 K에 noise가 발생했을 때 그것을 정상적인 입력으로 받아들여 output에 영향을 준다. 이러한 문제점은 master-slaved D flipflop을 사용하거나

edge-triggered flipflop을 사용함으로서 해결할 수 있다.

5. 결론

Lab 5_1에서는 ALU의 기본이 되는 2:1, 4:1 MUX와 Adder를 만들고 각각의 input에 따라 다양한 연산을 수행하는 ALU를 구현했다. Lab5_2에서는 SR Latch를 구현해보고 이를 2개 연결하여 Master-slaved JK FlipFlop을 구현했다. 또한, 각각의 파일에 대해 testbench 파일을 직접 구현해보면서 내가 짠 모듈이 정상적으로 진행되고 있음을 확인할 수 있었다.

6. 고찰

이번 실습에서는 이론적으로 학습했던 ALU를 구현해볼 수 있었다. 또한, 이를 활용하여 직접 testbench 파일을 만들어보면서 내가 만든 모듈이 정상적으로 작동하는 지 확인해볼 수 있었던 시간이었다. 베릴로그 문법을 완전히 숙지하지 못한 상태에서 이를 만드는 것은 다소 어려운 난이도였던 것 같다. 하지만 완성 이후 많은 것을 얻을 수 있었고 내가 만든 module을 가지고 새로운 module을 만들어본다는 점에서 유익한 시간이었다.