실험 5 보고서. ALU와 JK 플립플롭

2022. 05. 09.

20210054 정하우

1. 개요

조합 논리가 기반인 대표적인 회로인 ALU(산술 논리 장치)와 순차논리가 기반인 대표적인 회로인 JK flip-flop를 구현한다.

2. 이론적 배경

1) ALU_Arithmetic Logic Unit

ALU 는 입력에 대해 여러 산술(Arithmetic) 및 논리(Logic) 연산을 수행한다. 연산의 종류에 따라 산술 장치와 논리 장치 두 부분으로 나눌 수 있는데, 산술 장치는 사칙 연산, 증감 등을, 그리고 논리 장치는 Bitwise 논리 연산 등을 맡는다.

2) 비동기 회로 / 동기 회로

비동기 회로는 모든 연산이 순서대로 진행된다. 조합회로와 클록이 없는 순차 회로는 비동기 회로라고 생각하면 된다. 동기회로는 연산이 클록 신호에 의존한다. 클록 신호에 따라서 연산이 진행되기도 하고 멈추기도 한다. 클록 신호 덕분에 전체 회로의 각각의 연산을 통제할 수 있다. 이를 통해 다른 회로와 같은 순간에 작동하도록 만들 수 있는 것이다.

모든 조합 회로와 클록을 따르지 않는 순차 회로는 비동기 회로다. 한편 동기 회 로는 동기화 되어, 즉 다른 회로와 같은 순간에 맞춰 작동하기 위해 클록 신호를 따른다.

3) JK 래치 / JK 플립플롭

JK 래치는 기존에 S, R 모두 1 을 입력하는 것이 안되는 SR 래치에 추가적인 회로를 더해 S 와 R 이 동시에 1 인 상황에서도 정상적으로 작동하도록 수정한 것이다. JK 래치에서 J 와 K 가 동시에 1 일 경우 토글, 즉 현재 상태에 상관없이 값을 반전시킨다. 래치가 입력이 바뀔 때 출력도 바로 바뀌는 비동기 회로라면 플립플롭은 입력이 바뀌더라도 출력이 클록에 맞추어 반영되는 동기 회로이다. 즉 JK 플립플롭은 클록 신호를 추가로 받아 이에 맞추어 작동한다.

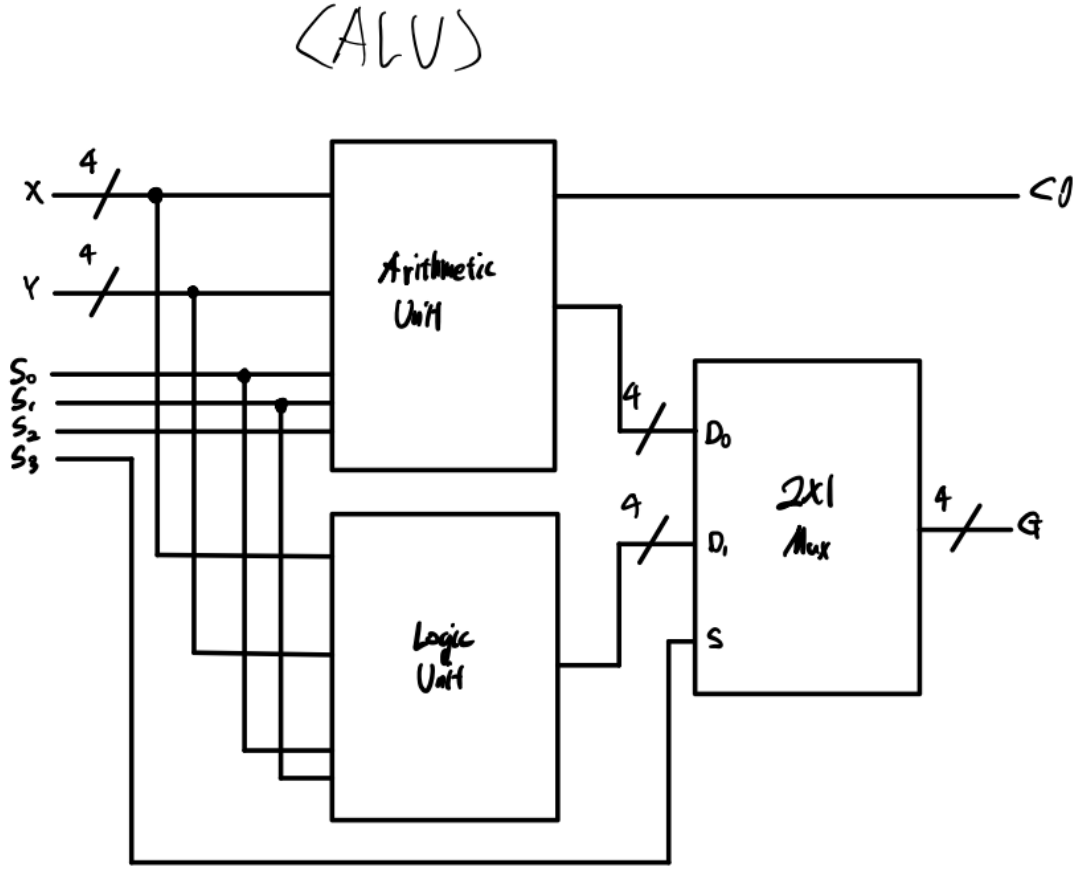
4) Master-slave JK 플립플롭

Master-slave JK 플립플롭은 SR 래치 두 개를 연결하여 만든 플립플롭이다. 이 플립플롭은 클록이 1 인 동안 Master 래치를 활성화해 입력을 임시로 저장한 뒤 클록이 0 이 되는 순간 Slave 래치로 전달한다. 따라서 Master 래치가 활성화되어있는 동안 글리치로 잠깐 입력값이 생기면 이 값이 Master 래치에 저장되어있다가 다음 클록이 0 이 되는 순간에 Slave 래치로 전파되는 문제가 있다. 이는 클록이 1 인 동안 계속 입력을 받기 때문에 생기는 문제로 클록이 0 에서 1 로, 혹은 1 에서 0 으로 바뀌는 순간에만 입력을 받는 Edge-trigger 회로를 사용하여 해결할 수 있다.

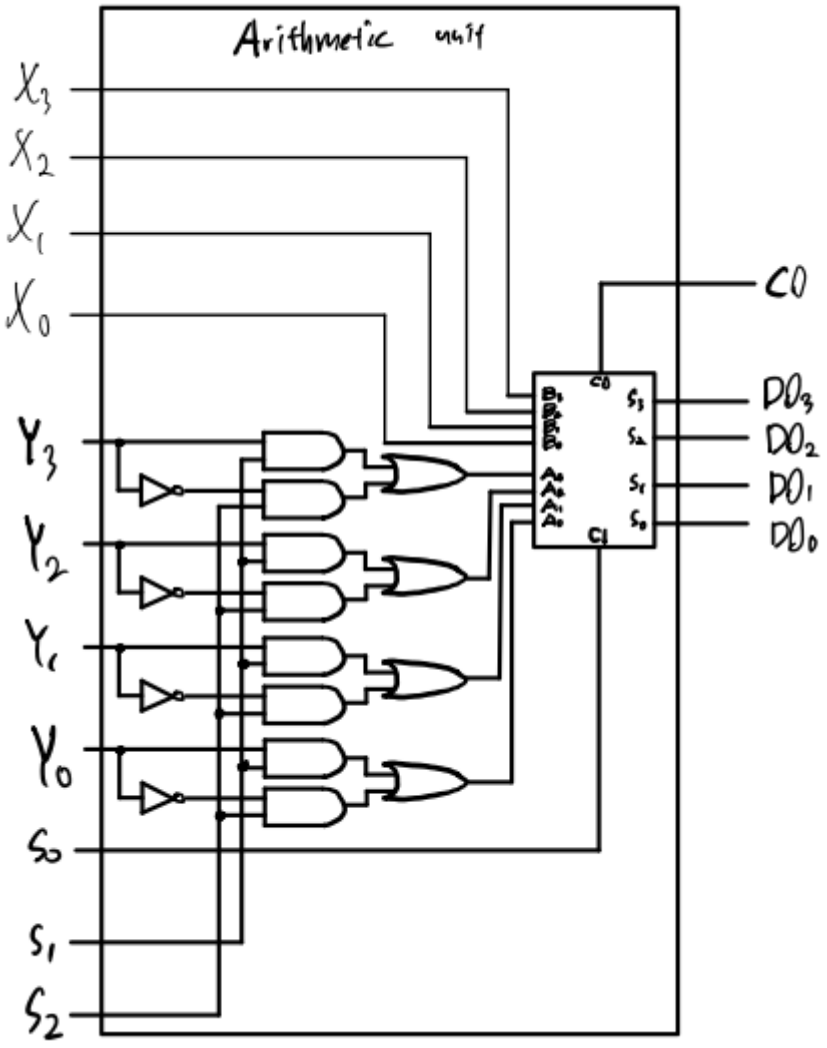
3. 실험 준비

1) ALU

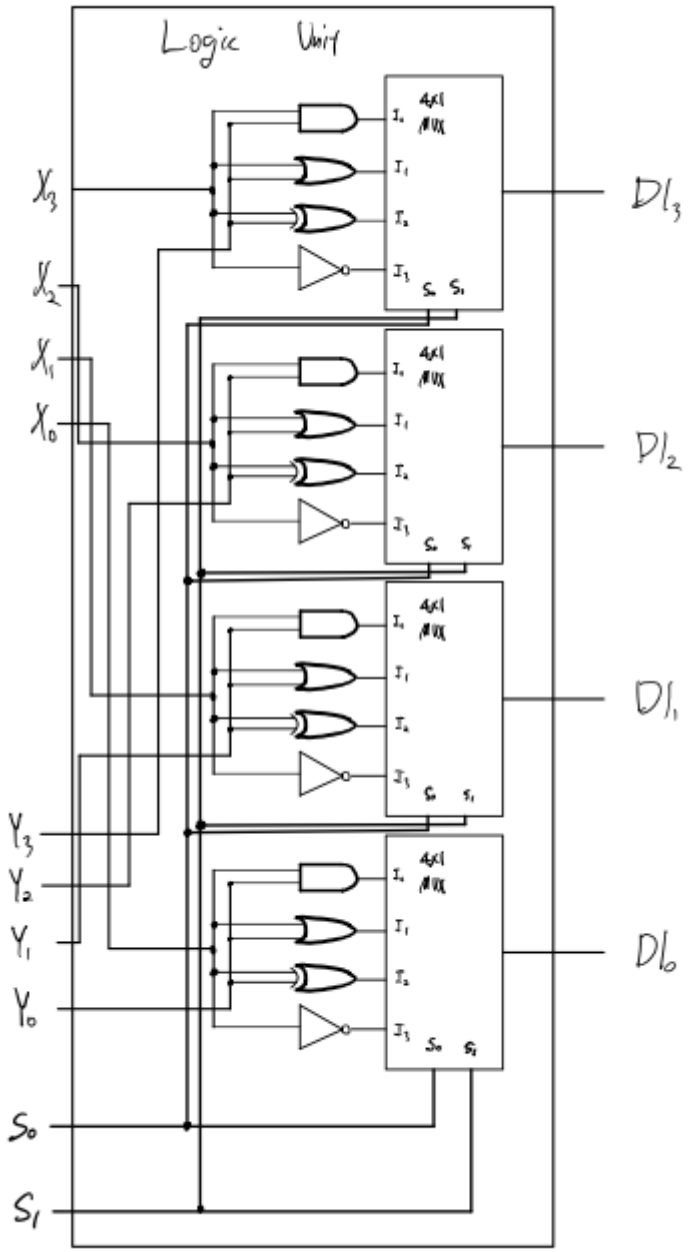
ALU circut



Arithmetic circut

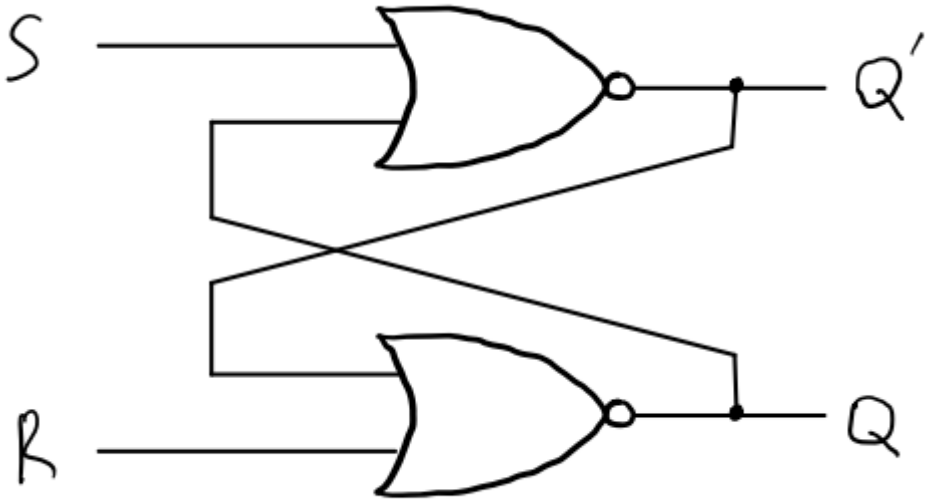


Logic circuit

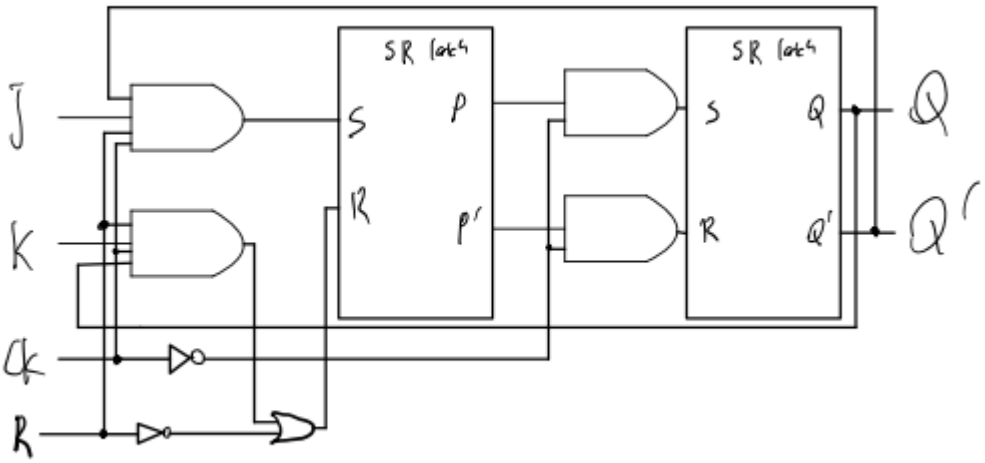


2) Master-slave JK 플립플롭

SR latch circuit



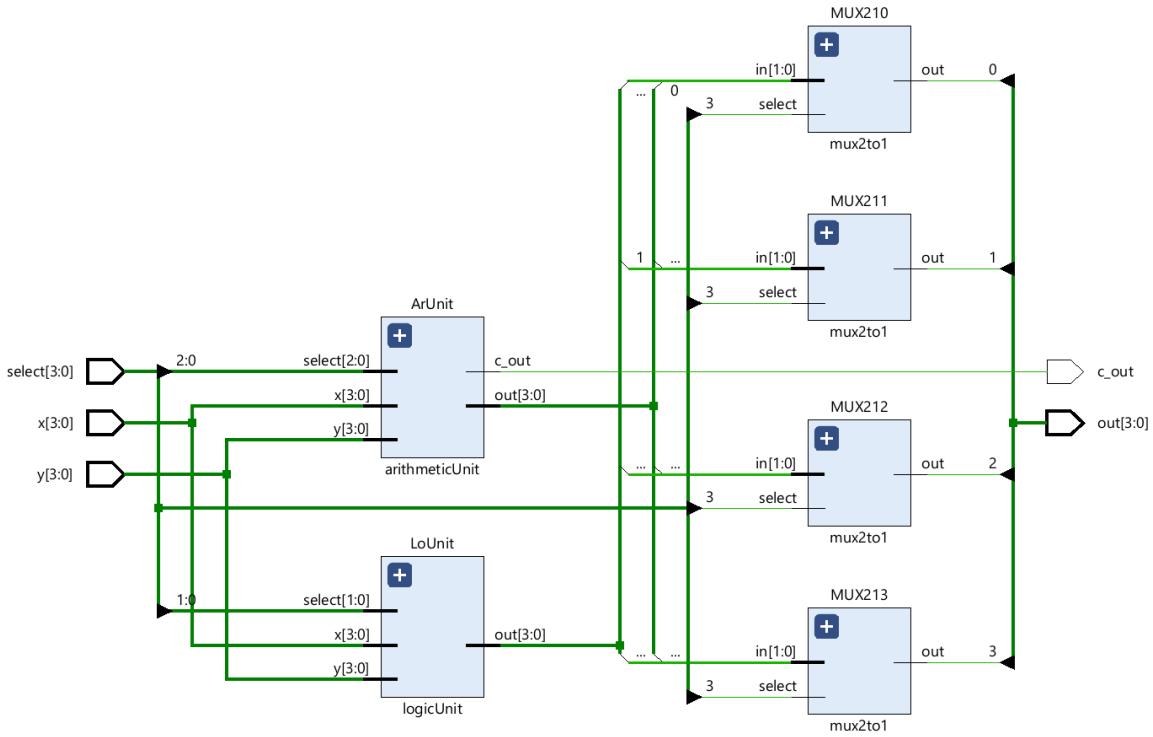
Negative reset JK 플립플롭 회로도



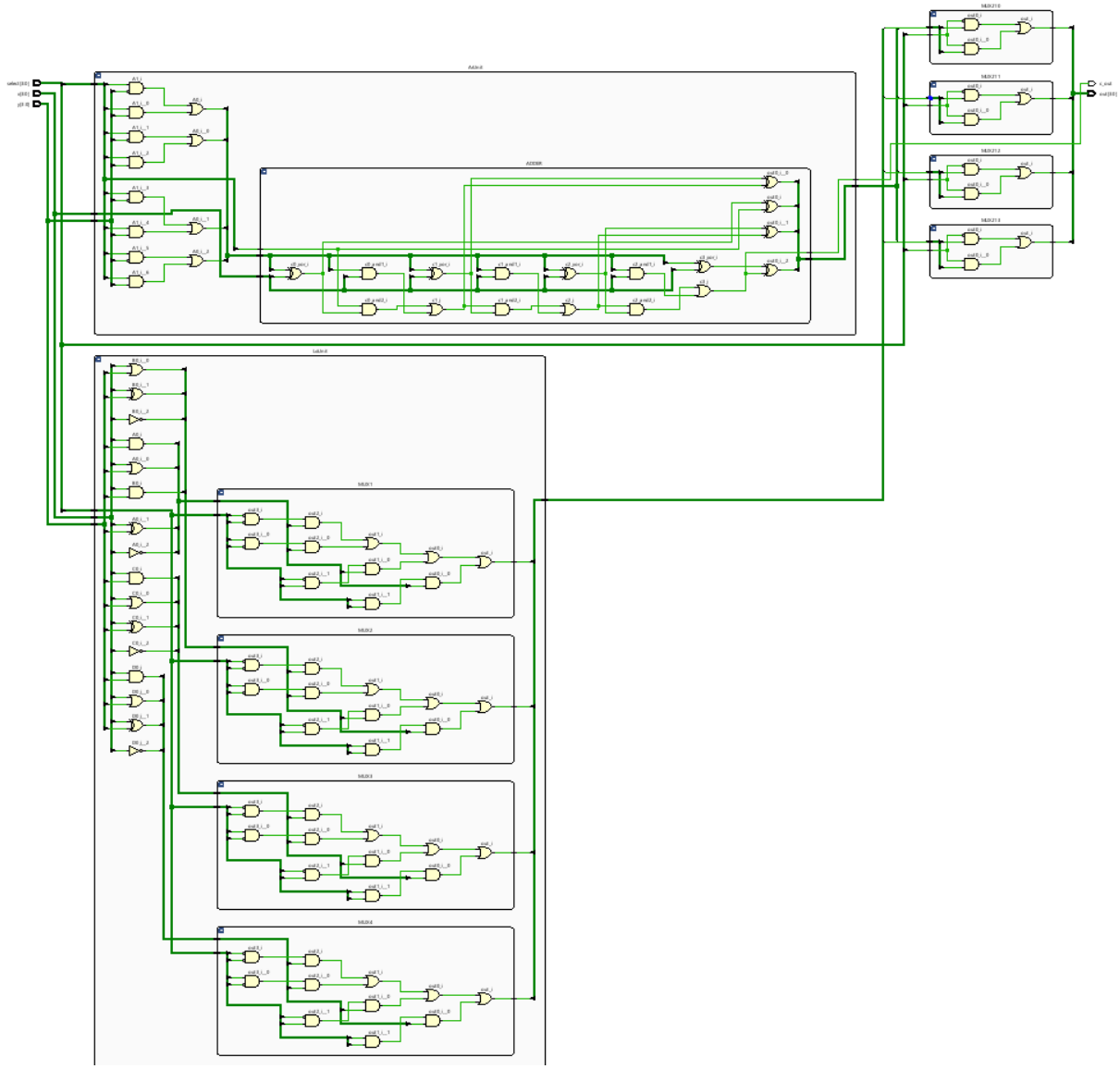
4. 결과

1) ALU lab5_1.v, lab5_1_tb.v

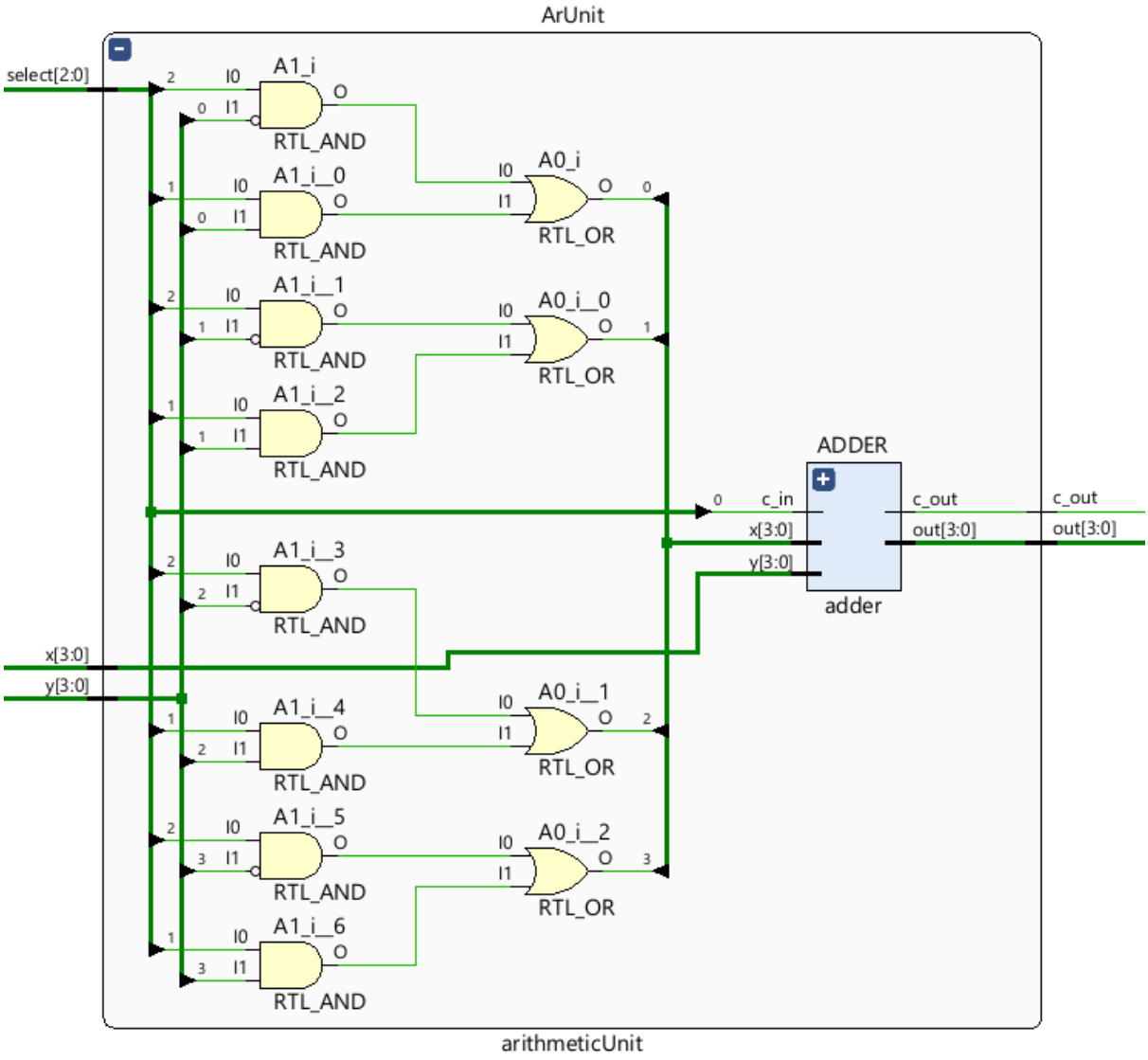
구현한 회로는 다음과 같다.



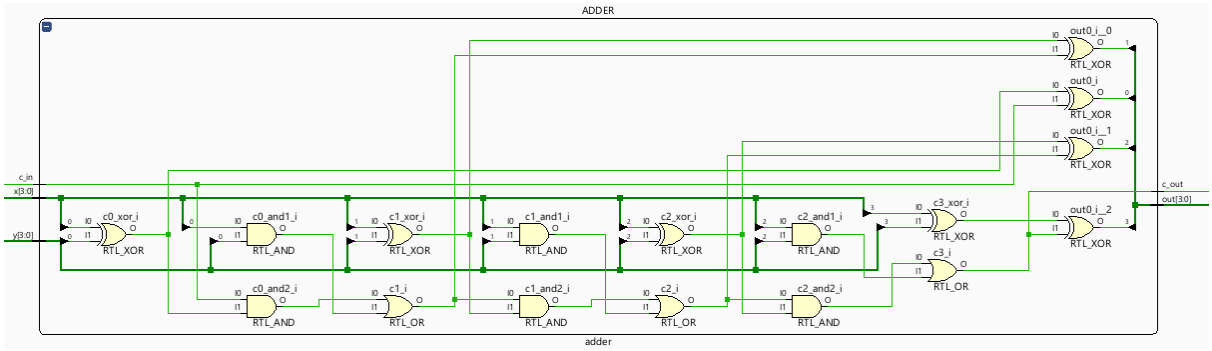
모든 회로를 펼치면 다음과 같다.



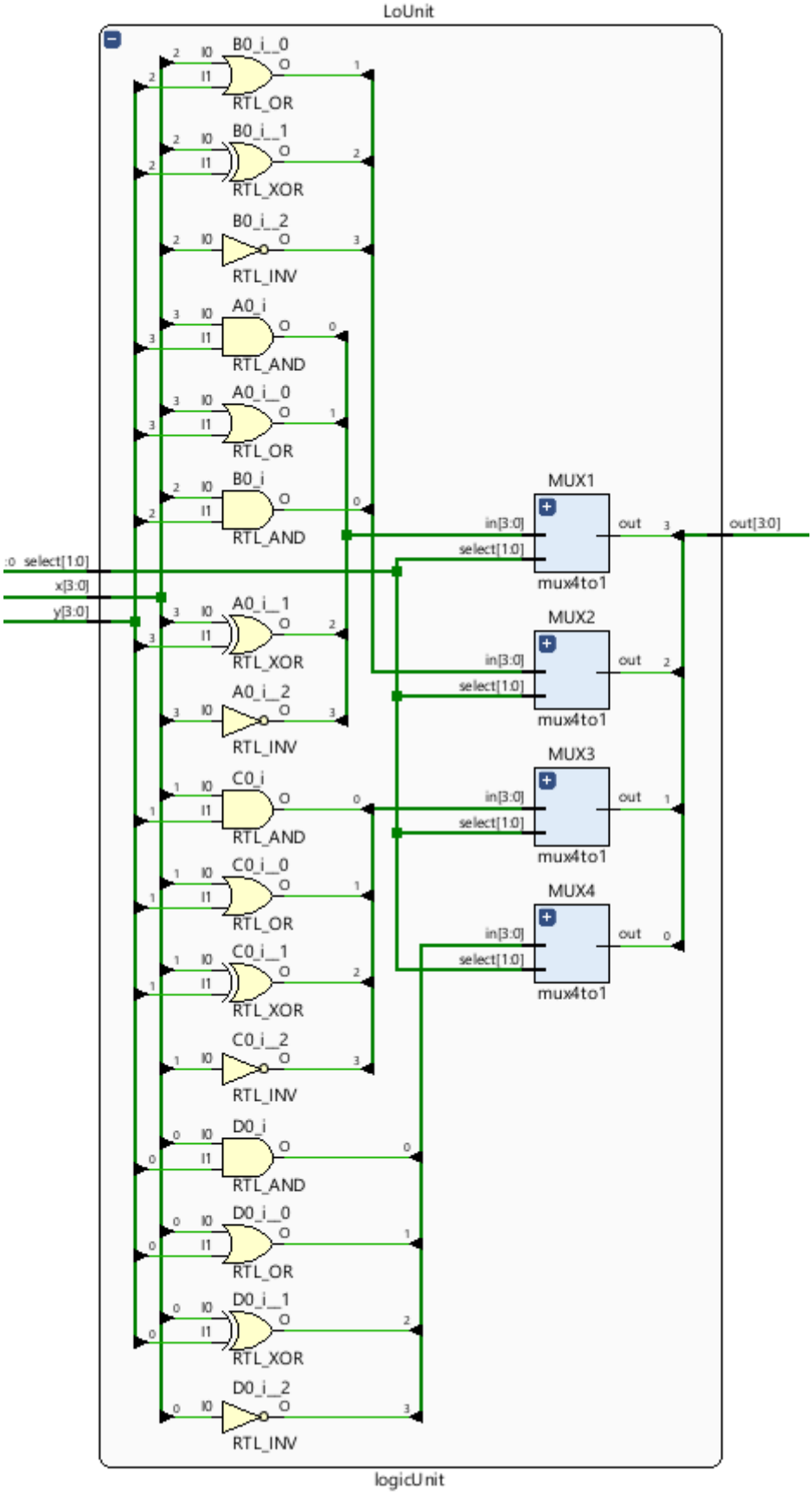
Arithmetic Unit



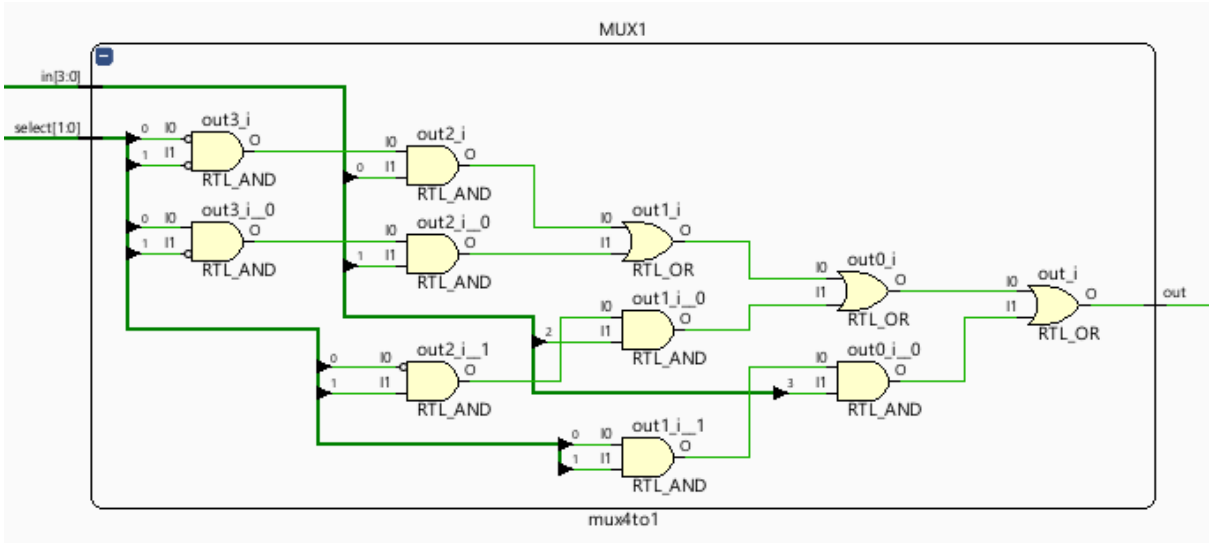
adder



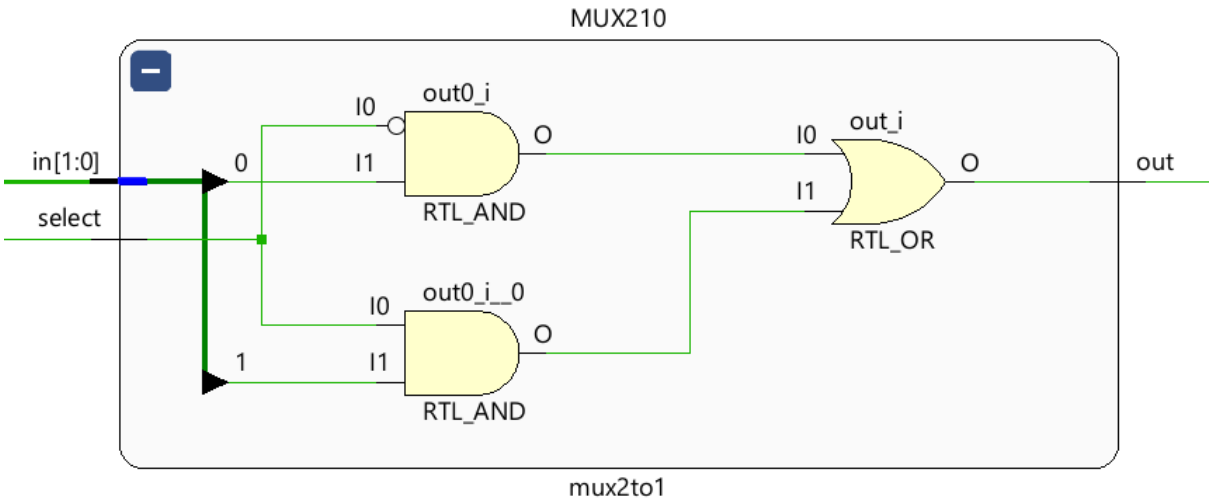
Logic Unit



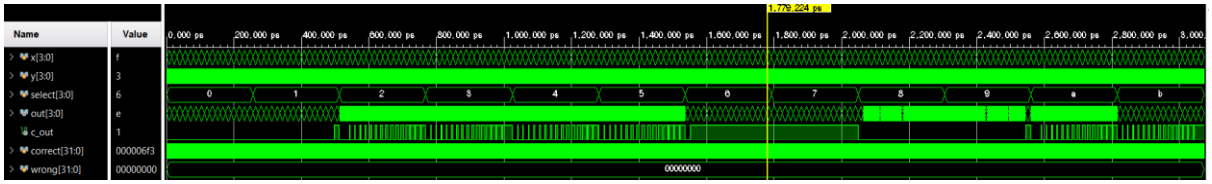
Multiplexer(4-1)



Multiplexer(2-1)

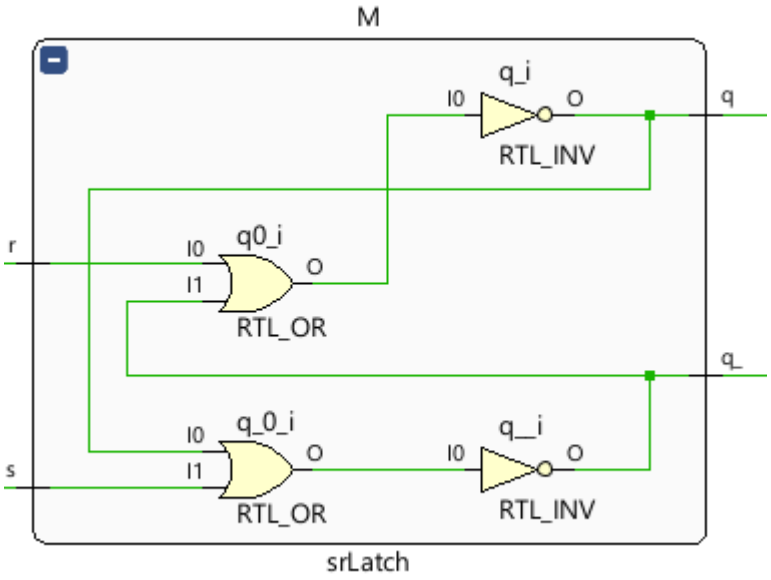


Wrong 값이 수행 종료까지 0 인걸 보아 제대로 작동했다고 판단할 수 있다..

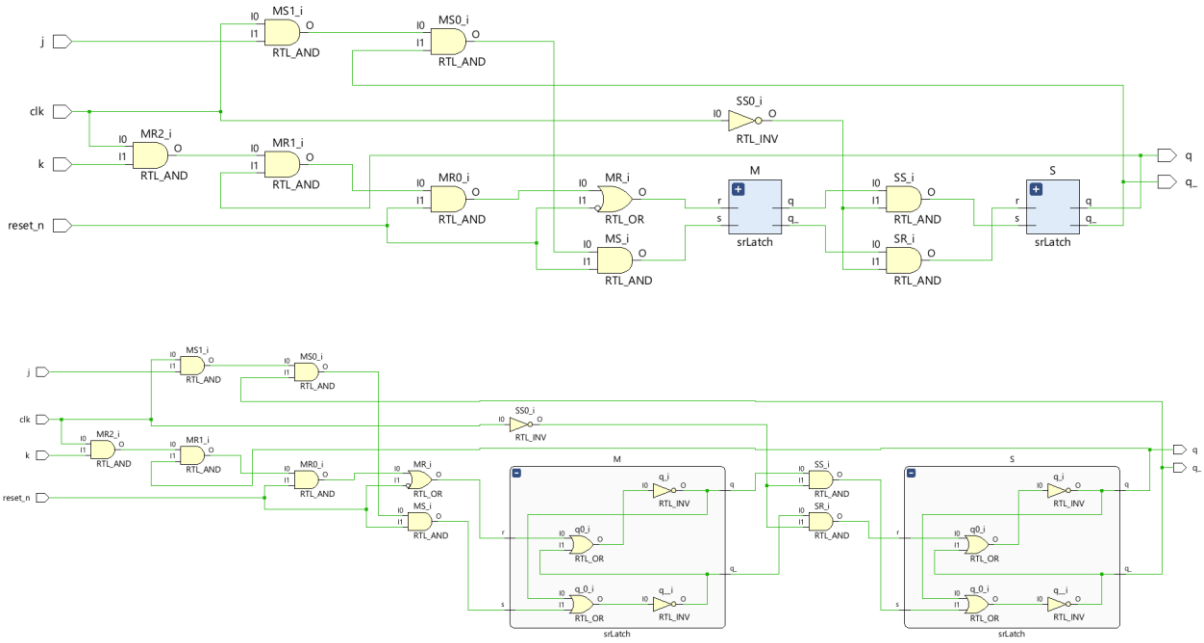


2) Master-slave JK flip flop

SR letch

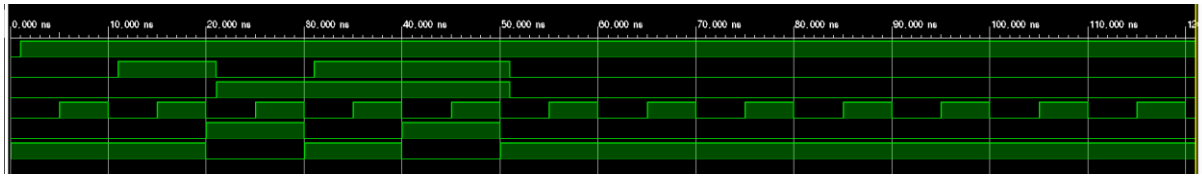


Negative reset JK flip flop



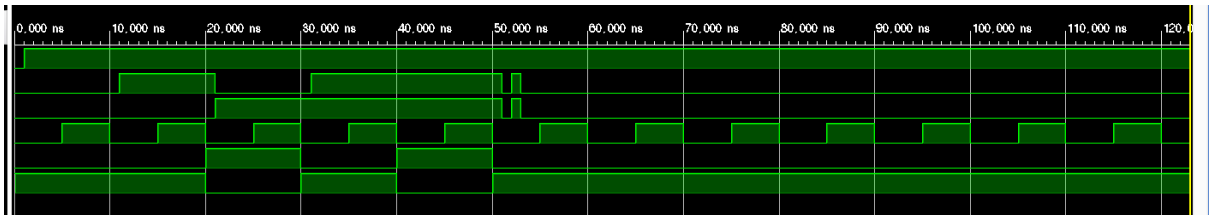
S-R latch 에서는 S 와 R 이 동시에 0 인상태에서 1 로 바뀌는 glitch 가 발생하면 race 가 발생하는 문제가 발생한다. 하지만 본 실습에서 구현한 flip flop 에서는 clock=0 일 때 j 와 k 가 0 에서 1 로 가는 glitch 가 발생해도 문제가 없다.

<주석 내부 작성을 하지 않은 상태(#20 finish->#70 finish)>>



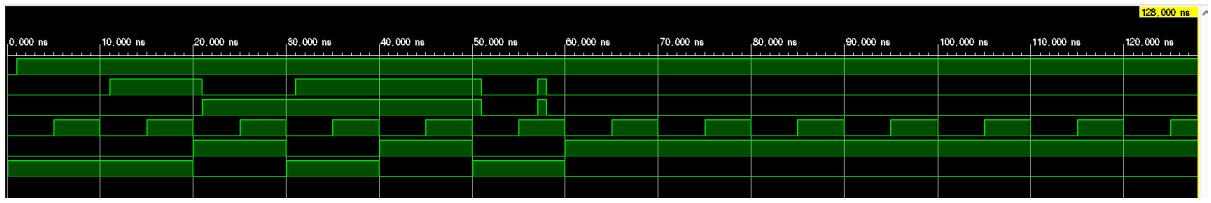
```
//////////////////////
#1  j=1;k=1;
#1  j=0;k=0;
//////////////////////
```

<EX1 Clock=0일 때 J 와 k에 glitch(0->1)이 동시에 발생한 경우(#20 finish->#70 finish)>>



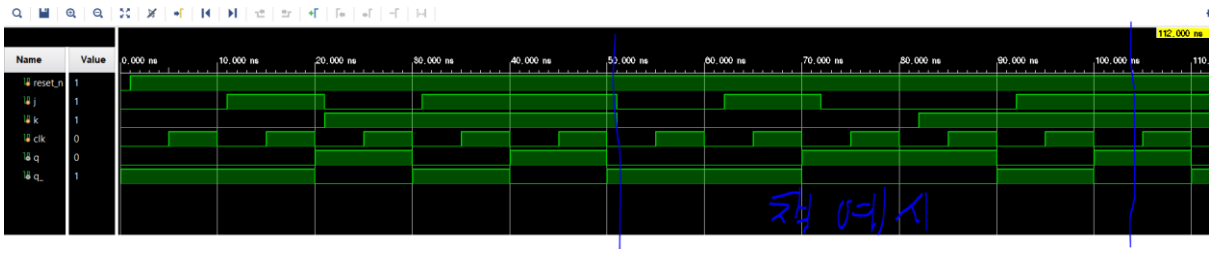
하지만 clock=1인 경우에 glitch가 발생한 경우에는, 원하지 않은 값이 나온다.

<EX2 Clock=1일 때 J 와 k에 glitch(0->1)이 동시에 발생한 경우(#20 finish->#70 finish)>>



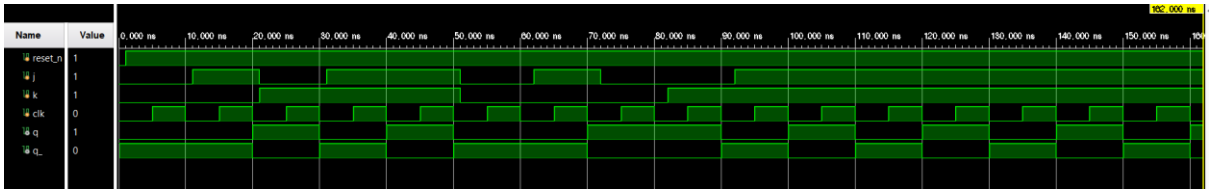
문제가 생기는 이유는 본 실습에서 만든 flip flop는 reset=1일 때는 master/slave flip flop랑 동일하다. 따라서 clock=1일 때 glitch가 발생하면 문제가 생기는 것은 당연하다. 하지만 glitch가 생기지 않고, j와 k가 값이 변하는 시간이 clock의 주기보다 길면, reset=1일 때 negative-edge-trigggered J-K FF과 동일한 결과 값이 나온다. 아래는 교재 chap6 p.14 negative-edge-trigggered J-K FF의 예시이다

<EX3>

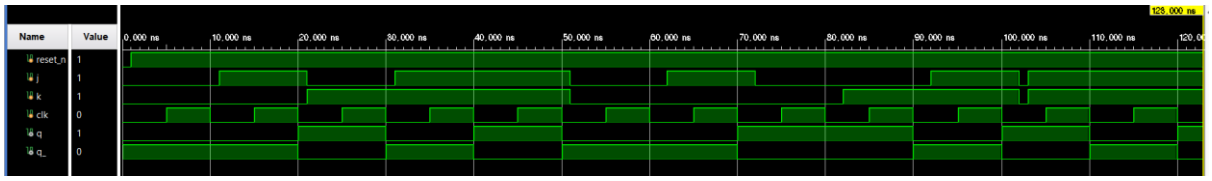


당연한 이야기지만, clock=0일 때 1에서 0으로 가는 glitch 역시 상관이 없다. 하지만 clock=1일때는 상관이 있다.

<EX3(#20 finish->#70 finish)>

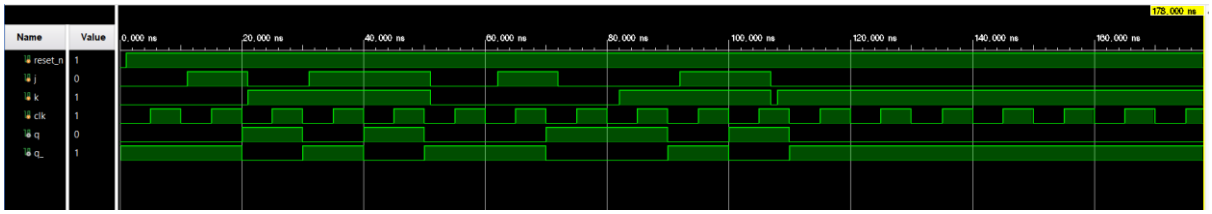


<EX4 Clock=0일 때 J 와 k에 glitch(1->0)이 동시에 발생한 경우>

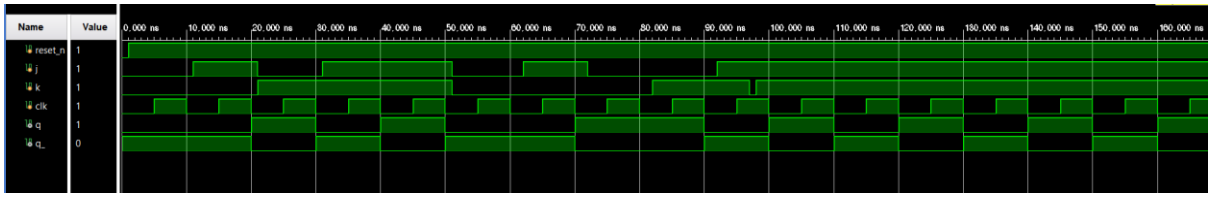


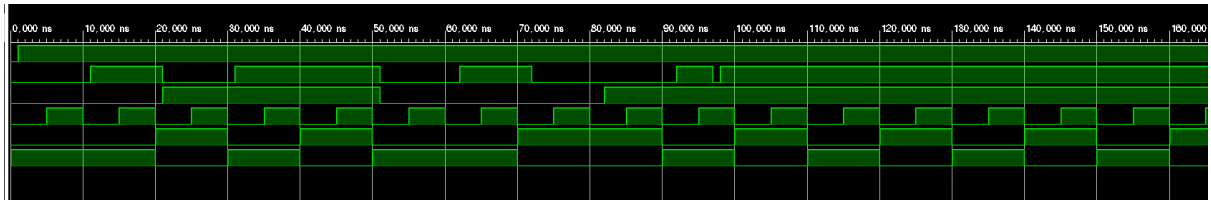
위와 동일함을 알 수 있다.

<EX5 Clock=1일 때 k에 glitch(1->0)가발생한 경우>



물론 아래 예시와 같이 문제가 없는 경우도 있으나, 이는 단순한 우연일 뿐이다(EX6, EX7)





모든 예시들은 주석 처리 해 놓았습니다

5. 논의 및 결론

처음으로 testbench를 구현해 보았다. 이번 testbench를 구현하면서 여러 예시를 접하였고, ALU와 그 안에 들어있는 Arithmetic unit, logic unit, multiplexer에 대해 좀더 깊이 이해할 수 있는 시간이 되었다. 특히 처음에 wrong 가 너무 많이 나와 당황했는데, 어떤 경우에 wrong가 뜨는지를 보고 adder

에 문제가 있다는 사실을 알고 고치는 등 다양한 시행착오를 통해 개념을 확실하게 잡은 계기가 되었다.

또한, Flip flop에서도 정말 다양한 예시를 생각해 보고 실제로 예측한 값과 비교하면서 개념을 확실히 잡을 수 있었다.