CSED273 Lab 5 보고서

2022.05.12

20210643 김현준

1. 개요

본 Lab5 에서는 컴퓨터 연산에서 가장 기초가 된다고 할 수 있는 산술 논리 장치(Arithmetic Logic Unit) ALU를 구현하고, 정보를 저장할 수 있는 JK 플립플롭을 구현한다. 또한, 구현한 ALU와 Flip-Flop을, testbench를 작성하여 제대로 구현되었는지 확인하는 작업을 거쳐, 잘못된 부분이 있다면 수정하도록 한다.

특히 두 파트로 나뉘어 lab이 진행되는데, 우선 먼저 각 기능 별 모델을 구성하여 4 bit ALU를 구현하고, 이후 Master-slave JK Flip-Flop을 구현한다. Testbench도 직접 작성하여 정상적으로 작동하는지를 확인하도록 한다.

2. 이론적 배경

1) ALU (Arithmetic Logic Unit)

ALU는 입력에 대해서 여러 산술 연산과 논리 연산을 수행하는 회로이다. 산술(Arithmetic) 장치와 논리(Logic) 장치 두 부분으로 나눌 수 있다. 산술 장치는 사칙 연산(transfer, increment, add, 1's complement subtraction, 2's complement subtraction, decrement, transfer) 등을, 논리 장치는 Bitwise 논리 연산(AND, OR, XOR, NOT) 등을 수행한다.

2) 비동기 회로 / 동기 회로

모든 combinational logic 회로와 클록을 따르지 않는 순차 회로는 비동기 회로이다. 동기 회로는 다른 회로와 같은 순간에 맞춰 작동하기 위해 클록 신호를 보내 동기화시켜 작동하는 회로이다.

3) JK 래치 / JK 플립플롭

JK Latch는 SR Latch에 추가로 회로를 더하여, SR Latch의 문제점이었던, S와 R이 동시에 1이 입력되는 상황에서도 정상적으로 작동할 수 있도록 발전시킨 것이다. JK 래치에서 J와 K가 동시에 1일 경우 현재 상태에 상관없이 값을 반전시킬 수 있다. JK가 00이면 hold의 기능, 01이면 reset의 기능, 10이면 set의 기능, 11이면 toggle의 기능을 한다.

JK latch의 경우 입력이 바뀔 때 출력도 바로 바뀌는 비동기 회로이다. 반면, JK 플립플롭은 입력이 바뀌더라도 출력이 clock에 따라 맞추어 반영이 되도록 설계된 동기 회로이다. 즉, JK 플립플롭은 clock의 신호를 추가로 받아서 이에 맞추어 동기 회로의 기능을 하게 된다.

4) Master-slave JK 플립플롭

Master-slave JK Flip-Flop은 SR latch 두 개를 연결하여 만든 플립플롭으로, clock이 1인 동안 Master

latch를 활성화하여 입력을 임시적으로 저장한 후 clock이 0으로 떨어지는 순간 slave latch로 이를 전달하는 기능을 한다. 즉, Master latch가 활성화된 동안 잠깐 glitch가 발생하여 입력 값이 들어가면 Master에 저장되어 있다가 이후 Slave로 전파될 수 있는 문제가 있다. 이 문제를 해결하는 것은 clock이 0에서 1로, 1에서 0으로 바뀌는 순간에만 입력을 받는 Edge-trigger 회로를 사용하는 것인데, clock이 1인 동안 입력을 연속적으로 받기 때문에 문제가 생기기 때문에 이를 해결하는 방법이다.

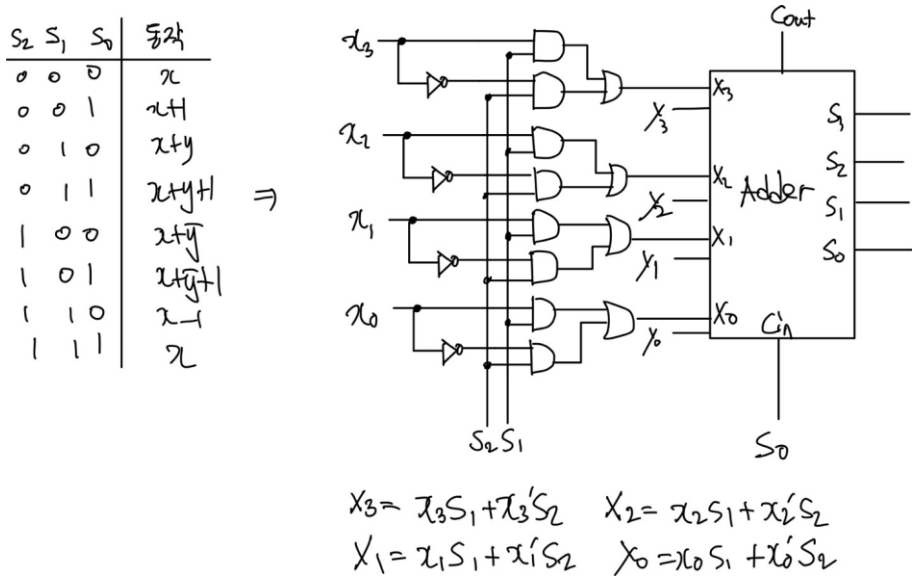
3. 실험 준비

1) ALU

	Select				동작	Adder 입력		
	S_3	S_2	S_1	S_0	$out = A + B + C_{in}$	A	B	C_{in}
산술 장치	0	0	0	0	x	0000	x	0
	0	0	0	1	$x+1$	0000	x	1
	0	0	1	0	$x+y$	y	x	0
	0	0	1	1	$x+y+1$	y	x	1
	0	1	0	0	$x+\bar{y}$	$\bar{y}$	x	0
	0	1	0	1	$x+\bar{y}+1$	$\bar{y}$	x	1
	0	1	1	0	$x-1$	1111	x	0
	0	1	1	1	x	1111	x	1

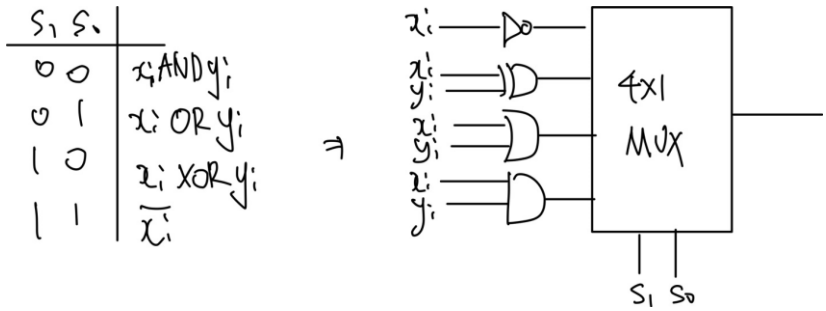
	Select				동작
	S_3	S_2	S_1	S_0	out_i
논리 장치	1	0	0	0	$x_i \ AND \ y_i$
	1	0	0	1	$x_i \ OR \ y_i$
	1	0	1	0	$x_i \ XOR \ y_i$
	1	0	1	1	$\overline{x_i}$

우선 4-bit ALU를 구현하기 위해서, S3 값에 따라 산술 장치와 논리 장치 두 부분으로 나누어 단순화 및 회로를 구성하였다. 위의 표와 같이, S3이 0일때는 Arithmetic, 1일때는 logic part가 실행되도록 하였다. 우선 Arithmetic 부분은 다음과 같이 구상하였다.

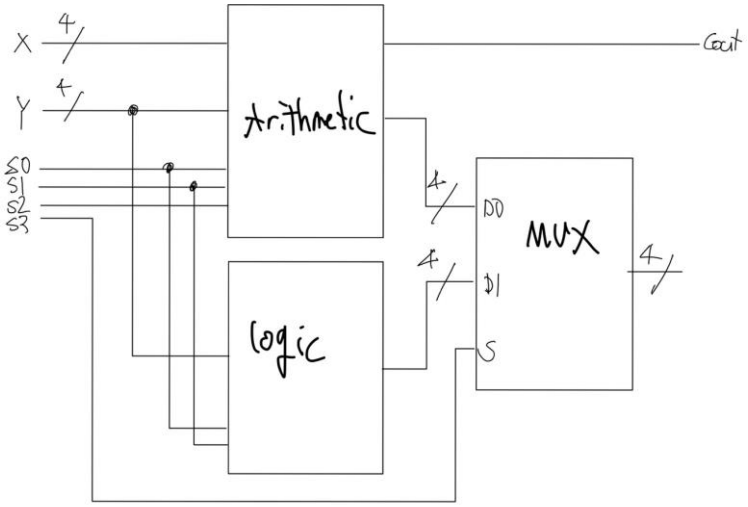


S0는 +1의 여부를 결정하므로, Carry in으로 adder에 입력시켰고, S1과 S2는 각각 +y를 하는지, +y'를 하는지 결정하기 때문의 위의 식과 같이 연결하였다. (왼쪽 그림 동작에서 x, y와 회로의 x, y가 반전되어 있음)

다음으로 logic part의 회로를 아래와 같이 구상하였다. 우선 각 논리 실행에 대해서 각 gate를 이용하여 구현하고자 하였고, S1, S0을 selection input으로 하여 4x1 MUX를 통과시켜 마무리하였다.



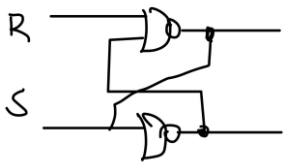
마지막으로 두 모듈을 2:1 MUX로 묶어 전체 ALU의 회로드를 그린 결과, 아래와 같이 표현할 수 있었다. Arithmetic part와 logic part가 S3에 따라 나뉘어 기능이 수행되도록 되어 있다.



2:1 MUX 4개를 이용해서, S3를 MUX의 selection input으로 사용하여 arithmetic 회로와 logic 회로 부분을 선택하도록 하였다. 이렇게 하여 전체 ALU를 구현하면 될 것이다.

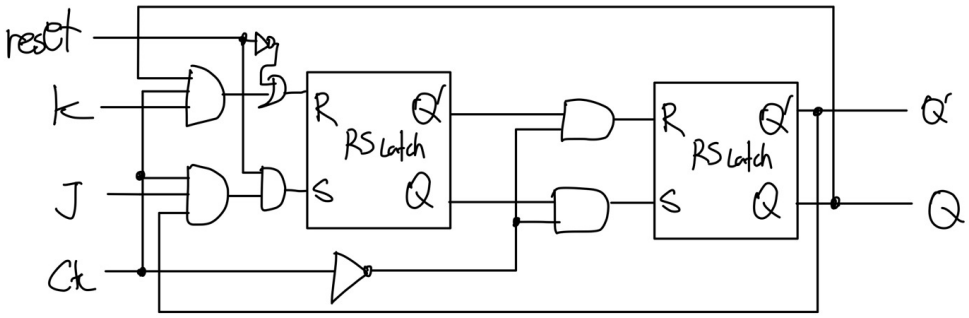
2) Master-slave JK Flip-Flop

우선 JK Flip-Flop를 만들기 위해서 SR래치를 먼저 구현하여야 한다. NOR 게이트를 이용하여 SR latch를 구현한 결과 아래와 같이 구상할 수 있었다.



또한, SR래치를 두개 연결하여 JK Flip-Flop을 만들 수 있었다. 이는 아래와 같은데, 이때 우리는 Negative reset을 Master-slave JK 플립플랩을 만들기 위해 입력 앞에도 게이트를 넣어 놓았다. reset_n이 0일 경우 Master latch에 입력되는 S와 R 값을 변경하여 reset 동장을 수행할 수 있도록

and gate와 or gate, not gate를 배치하였다.

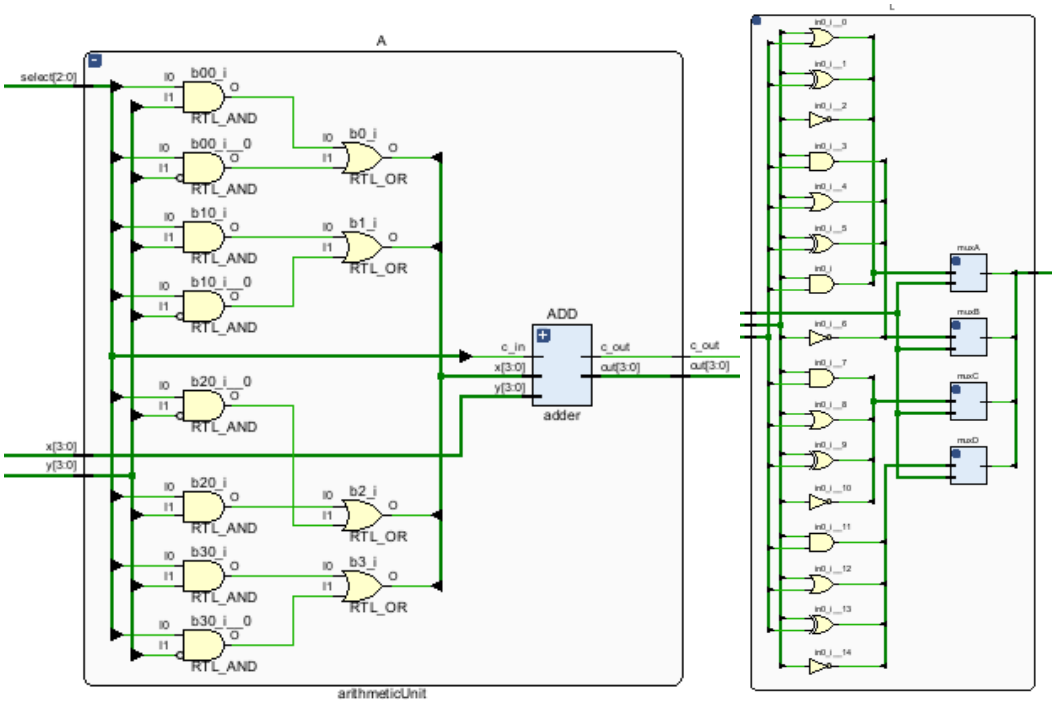


다음으로, SR 래치와 비교하여 Master-slave JK 플립플롭은 clock이 0인 경우에 들어오는 글리치에 대해서는 해결할 수 있지만, clock이 1인 경우에 들어오는 글리치에 대해서는 해결할 수 없을 것으로 예상하였다. 이는, clock이 1인 동안은 Master 래치로 입력이 연속적으로 받아지기 때문에 그때 들어오는 글리치도 입력으로 들어오기 때문이다. 반면에, clock을 적절히 활용하여 JK FF를 형성함으로써, SR 래치와는 다르게 clock이 0인 경우는 글리치가 발생하지 않는 것을 알 수 있다.

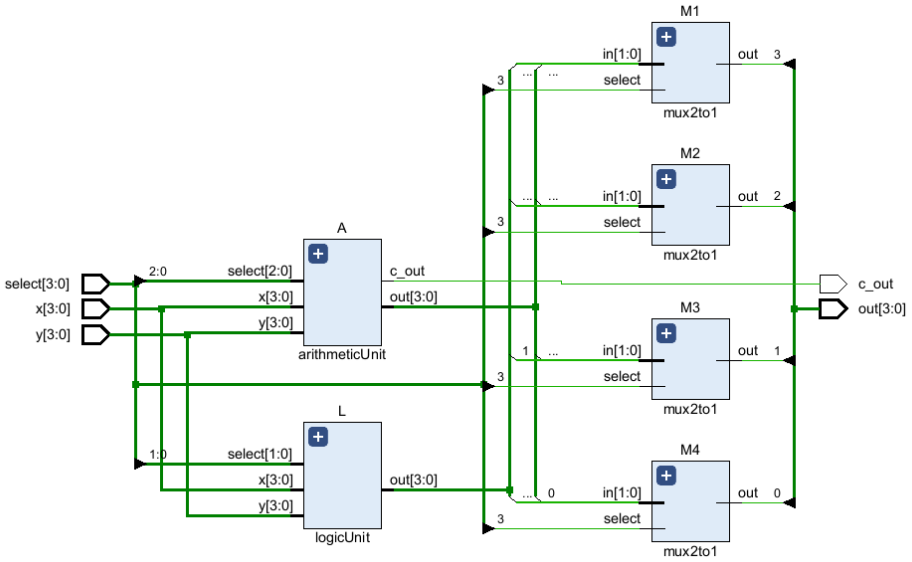
4. 결과

1) ALU

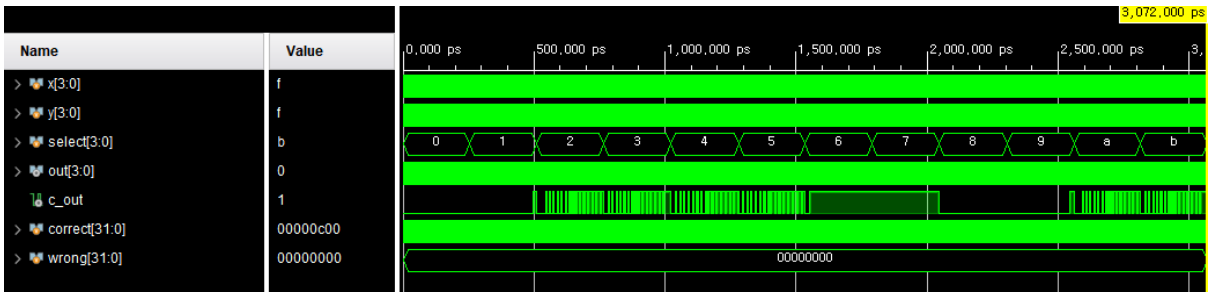
실험 준비 과정에서 구상하였던 대로 verilog로 코드를 작성할 수 있었다. 우선, ALU를 만드는 데에 필요한 MUX와 Adder 등을 모두 모듈로 구현하였고 이 모듈들을 이용하여 Arithmetic 부분과 logic 부분의 기능을 구현하고, 이를 MUX로 합쳐 연결하여 전체 ALU를 형성할 수 있었다. 우선 산술 장치와 논리 장치의 회로는 각각 아래와 같이 나오는 것을 확인할 수 있었다.



구현된 회로를 확인해 보면, Selection input S에 대해 산술 장치의 구현이 제대로 된 것을 확인할 수 있다. 또한, 각 게이트와 MUX로 논리 장치의 회로 또한 제대로 완성된 것을 확인할 수 있었다. 최종적으로 이 두 장치를 합친 전체 회로는 아래와 같이 Schematic으로 확인할 수 있었다.



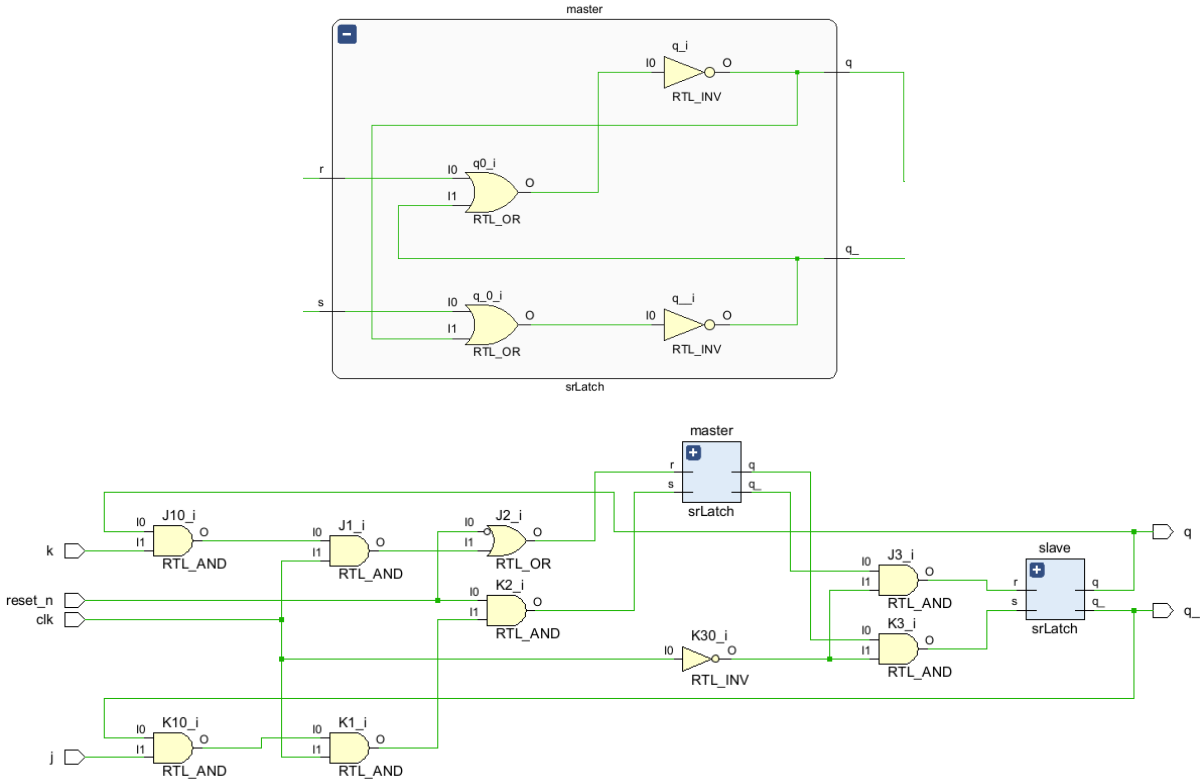
결과적으로 실험 준비 과정에서 구상했던 회로 그대로 vivado 환경에서 구현이 된 것을 확인할 수 있었다. 다음으로, 완성된 ALU의 모든 기능을 확인하기 위해서 testbench의 작성을 완료하고 이를 실행시켜보았다. ALU의 모든 기능(transfer, increment, add, 1's complement subtraction, 2's complement subtraction, decrement, transfer, AND, OR, XOR, NOT)을 각각 테스트하는 코드를 작성하고 이를 실행하였다. 그 결과는 아래와 같이 나왔다.



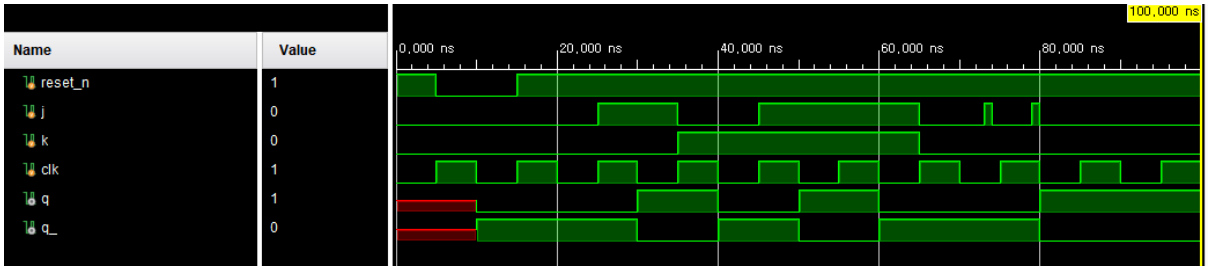
결과를 확인해 보면, correct는 00000c00, wrong은 00000000으로, 모든 case에서 출력이 제대로 나왔음을 확인할 수 있다. 즉, testbench에서 작성한 코드에 따라 계산된 값과 실제 구현한 ALU를 통과한 값이 모든 상황에서 일치하여 correct만이 증가되어 나타난 것이다. 그러므로 구현한 ALU는 제대로 그 기능을 하는 것으로 결론지을 수 있다.

2) Master-slave JK Flip-Flop

다음으로 Master-slave JK FF의 경우에는 기본이 되는 SR 래치 모듈을 구현한 뒤, 실험 준비 단계에서 설계했던 대로, Negative reset Master-slave JK 플립플롭을 완성할 수 있었다. Verilog로 모듈 설계를 마치고 schematic 기능으로 회로를 확인해본 결과 아래와 같이 나타났다.



위의 사진은 SR 래치가 구현된 모습이고, 아래는 SR래치 모듈을 이용해 전체 Master-slave JK FF를 구현한 모습이다. 회로도를 보면, 실험 준비 단계에서 설계했던 것과 똑같이 나왔음을 알 수 있다. 특히, reset_n을 다루는 회로도 부분의 and gate와 or gate 또한, 설계했던 대로 나오는 것을 확인할 수 있었다. k와 clock q가 AND를 통과한 것이 reset_n의 NOT를 통과한 것과 OR 게이트로 묶여 r의 input으로 들어가고, j와 clock, q'이 AND를 통과한 것이 reset_n과 AND 게이트로 묶이는 것을 schematic에서 확인할 수가 있다. 회로에 대한 확인을 마치고 나서 testbench를 완성하여 이 FF의 동작을 확인하였다. testbench는 특히 글리치에 대한 내용을 확인할 수 있도록 설계하였다. testbench 실행의 결과는 아래와 같다.



이 결과를 확인해 보면, 우선 앞부분에서 FF의 기능이 제대로 수행됨을 확인할 수가 있다. J, K 값에 따라서 clock이 1에서 0으로 변하는 순간에 set, reset의 기능이 제대로 수행됨을 확인할 수가 있고, 맨 앞부분에서 reset_n 값에 따라서도 제대로 reset이 이루어짐을 확인할 수가 있다. set과 reset 외에도 hold, toggle의 기능도 제대로 수행된 것을 위의 그림에서 확인할 수이 있다. 즉, FF이 실험 준비 과정에서 구상한 대로 제대로 완성되었고 그 기능을 제대로 수행하고 있다는 것을 알 수 있다.

다음으로 test의 마지막 부분에 두 개의 글리치를 만들어 놓았다. 이때, clock이 0일 때 나타난 glitch에 대해서는 아무 변화가 일어나지 않았다. 하지만 clock이 1일 때 나타난 glitch에 대해서는 clock이 0으로 다시 바뀔 때 q의 변화가 나타나는 것을 알 수 있다. 따라서, Master-slave JK Flip-Flop은 SR래치였으면 해결이 불가능한 글리치 하나(clock이 0일때의 글리치)를 커버하였지만, clock이 1일때의 글리치에 대해서는 커버를 하지 못한다는 것을 확인할 수 있었다. 이는 실험 준비 과정에서 예상했던 바와 일치하는 내용이다.

5. 논의

직접 ALU를 구현해 봄으로써 여러 MUX와 ADDER 모듈 등을 사용하여 하나의 장치를 만드는 데에 더욱 익숙해질 수 있었고, ALU에 대한 이해를 높일 수 있었다. 또한, 처음에 ALU 구현 코드를 작성하고 testbench를 돌렸을 때, wrong이 0이 아닌 문제가 발생하였는데, 이는 모델 구현 상에서 코드 작성 실수 때문이었고 이를 바로 수정하여 문제를 해결할 수 있었다.

또한, SR 래치와 Master-slave JK FF를 구현하면서 수업시간에 배운 내용에 대해 더 복습할 수 있었고, 특히 실험 준비 과정에서 reset_n을 어떻게 처리해야 할 것인지에 대해 고민이 많았지만, 상황에 맞게 gate를 연결하여 전체 FF를 설계할 수 있었다.

전체적으로 이번 lab5를 수행하면서 처음으로 testbench를 작성해 보았고, 관련된 문법 등이 처음에는 익숙하지 않았지만 다양한 자료를 찾아보면서 공부할 수 있었다. 특히, lab4에서 주어졌던 testbench를 다시 열어보고 공부하면서 힌트를 얻어 lab5의 testbench를 작성할 수 있었다.