디지털 시스템 설계 (CSED273) Lab 5. 이진수 연산 보고서

2023.04.27.

20220923 이윤혁

1. 개요

Lab5_1에서는 Arithmetic Unit과 Logic Unit을 구현하고, 이를 합쳐 ALU(Arithmetic Logic Unit)를 구현한다. Lab5_2에서는 SR latch를 구현하고, SR latch를 이용해 JK Flip-Flop을 구현한다.

2. 이론적 배경

2.1. ALU (Arithmetic Logic Unit)

ALU는 Arithmetic Unit과 Logic Unit을 Multiplexer로 연결한 Unit으로, Arithmetic Operation과 Logic Operation을 수행할 수 있다. Operation은 selection input으로 선택할 수 있다. Arithmetic Operation에는 사칙연산과 Increment, Decrement이 있고, Logic Operation에는 Bitwise And, Or, Xor, Not연산이 있다.

2.2. 비동기 회로 / 동기 회로

비동기 회로는 Clock 등의 동기화할 수 있는 신호가 부재하여, 출력이 동기화되지 않는 회로이다. 동기 회로는 Clock이 존재하여, 출력이 Clock에 동기화된 회로이다. 동기 회로는 주로 Clock이 1이거나 Clock의 상태가 바뀔 때 출력 값이 바뀌는 구조이다.

2.3. JK Latch / JK Flip-Flop

JK Latch는 SR Latch에 output을 input으로 다시 넣어준 회로이다. SR Latch의 forbidden input일 때, Racing이 일어나는 문제를 해결한다. JK Latch에서 SR latch의 forbidden state(J = 1, K = 1)가 Toggling state로 치환된다. Latch는 입력이 바뀌면 출력도 바로 바뀌는 비동기 회로이다. 반면 Flip-Flop은 Clock을 입력 받아 Clock에 출력을 동기화하는 동기 회로이다.

2.4. Master-Slave JK Flip Flop

Master-Slave JK Flip Flop은 SR Latch를 두개 있고, JK Latch처럼 output을 input으로 다시 넣어준 회로이다. Master-Slave Flip Flop을 구성하는 회로 중 입력을 받는 SR Latch를 Master Latch, 출력을 하는 Latch를 Slave Latch라고 한다. Clock이 1일 때, Master-Slave JK Flip-Flop에 입력이 들어오면 Master Latch에 임시로 입력이 저장된다. Clock이 0으로 떨어지는 순간 Master Latch의 출력이 Slave Latch에 입력되고, Slave Latch가 출력을 하면서 JK Flip-Flop의 출력이 Clock에 동기화되어 나온다. Master Slave는 Clock이 1일 때는 언제든지 입력 값을 저장하므로, Clock이 1일 때 생기는 Glitch에는 영향을 받는다는 문제점이 있다. 이는 Clock이 바뀌는 시점에서만 입력을 받는 Edge-

Trigger 회로를 이용해야 해결할 수 있다.

3. 실험 준비

3.1. Lab5_1

ALU의 Arithmetic Unit과 Logic Unit은 Selection input으로 분리할 수 있다. [표 1]은 Selection input에 따른 ALU의 동작을 분류한 표이다.

[표 1] ALU의 동작

	Select				동작	Adder 입력		
	S_3	S_2	S_1	S_0	out = A+B+C_in	A	B	C_in
산술장치	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+y_bar	y_bar	x	0
	0	1	0	1	x+y_bar+1	y_bar	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			
	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	Not x_i			

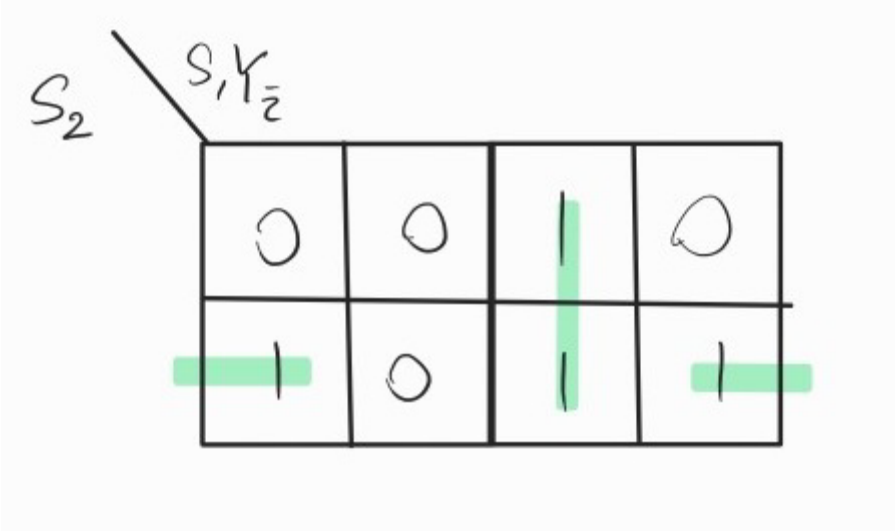
[표 2]는 Arithmetic Unit에서 Adder 입력은 selection input에 대해 나타낸 Truth Table이다.

[표 2] Adder 입력의 Truth Table

S_2	S_1	A
0	0	0000
0	1	y
1	0	y_bar
1	1	1111

[그림 1]은 [표 2]을 이용해 Adder input의 Bitwise K-map을 구성한 그림이다.

[그림 1] K-map of A_i

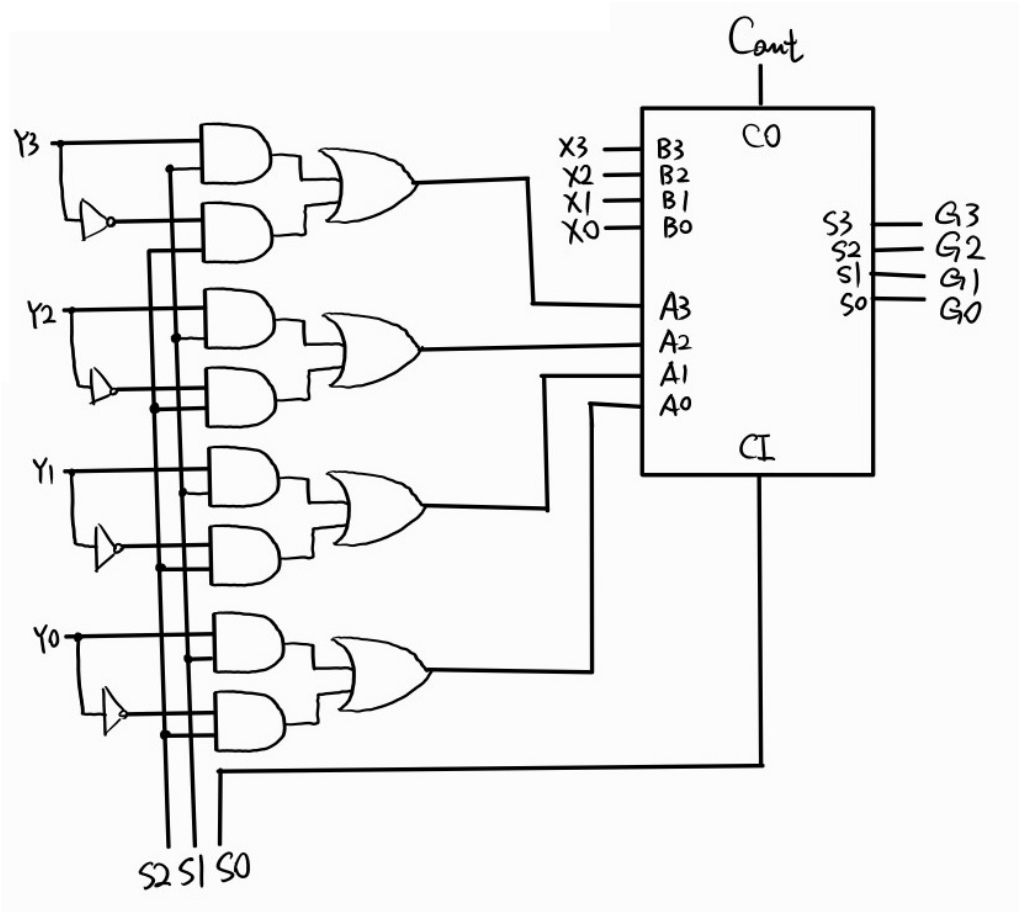


Simplification을 하면 다음과 같은 식을 얻을 수 있다.

$$A_i = S_2 y'_i + S_1 y_i$$

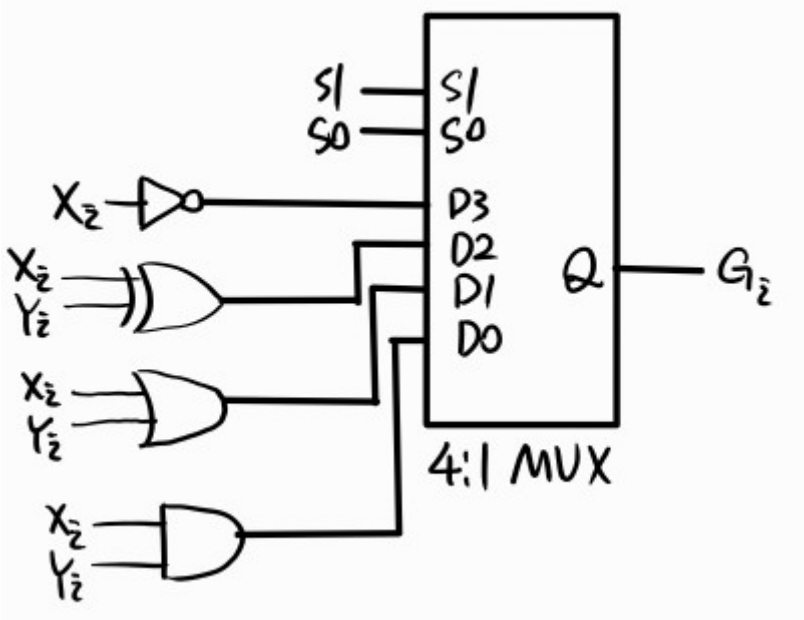
이 식을 토대로 Arithmetic Unit의 회로도를 그리면 [그림 2]와 같다.

[그림 2] Arithmetic Unit의 회로도



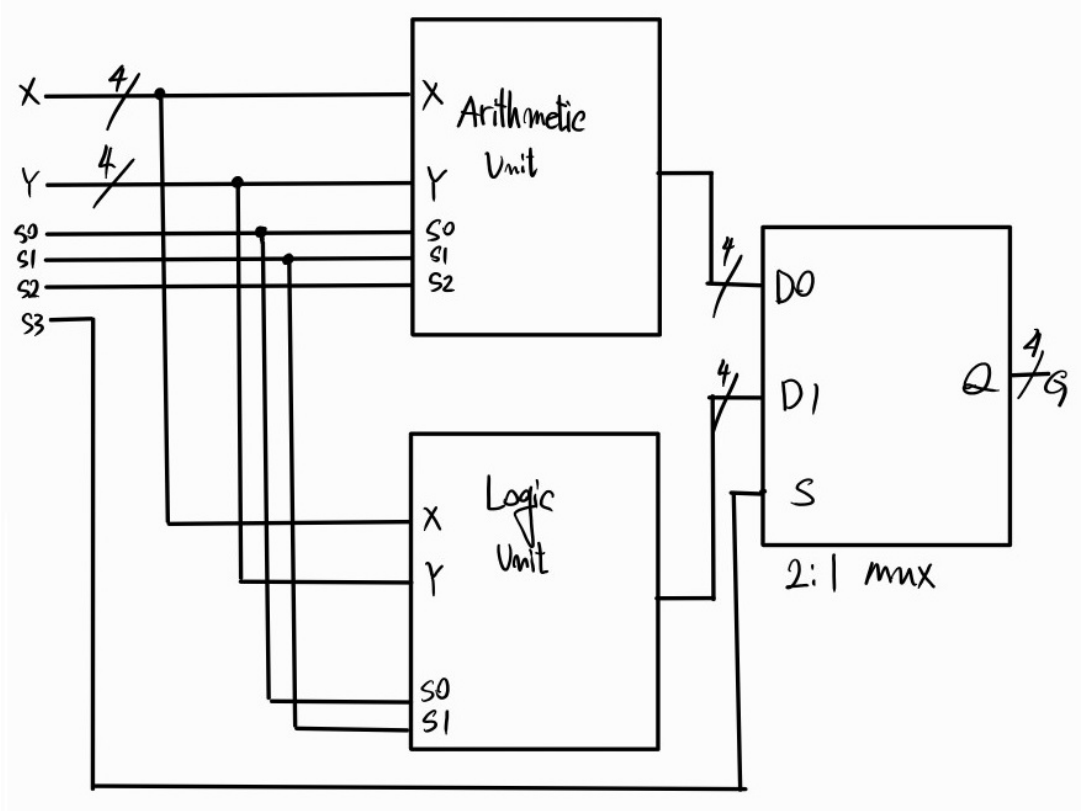
[표 1]을 토대로 4:1 MUX를 사용해 Logic Unit의 회로도를 그리면 [그림 3]과 같다.

[그림 3] Logic Unit의 회로도



Arithmetic Unit과 Logic Unit을 2:1 MUX를 이용해 합쳐 ALU의 회로도를 구성하면 [그림 4]와 같다.

[그림 4] ALU의 회로도

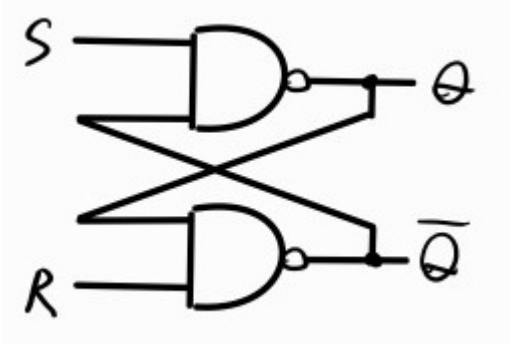


3.2. Lab5_2

Negative reset Master-slave JK Flip-Flop을 구현하기 위해선 SR Latch를 먼저 구현해야 한다. 일반 SR Latch와는 다르게 NAND 게이트로 SR Latch를 구현하므로 SR Latch의 동작이 일반 SR Latch와는 다르다. 그러나 SR latch 2개를 이용해 Flip-Flop을 구성하므로 이 SR Latch로 구현한 JK Flip-Flop은 입력과 출력이 일반 JK Flip-Flop과 동일하다.

NAND로 구현한 SR Latch의 회로도는 [그림 5]이고, [그림 5]의 excitation table은 [표 3]이다.

[그림 5] NAND로 구현한 SR Latch의 회로도



[표 3] NAND로 구현한 SR Latch의 Exitation table

	S(t)	R(t)	Q(t)	Q′(t)	Q(t+Δ)	Q′(t+Δ)	Q(t+2Δ)	Q′(t+2Δ)
Forbidden	0	0	0	1	1	1	1	1
Forbidden	0	0	1	0	1	1	1	1
Set	0	1	0	1	1	1	1	0
Set	0	1	1	0	1	0	1	0
Reset	1	0	0	1	0	1	0	1
Reset	1	0	1	0	1	1	0	1
Hold	1	1	0	1	0	1	0	1
Hold	1	1	1	0	1	0	1	0

Q(t+2Δ)항까지 추가한 이유는 미세한 딜레이 후 바뀐 출력 값이 SR Latch의 state를 또 바꾸기 때문이다.

이 SR Latch는 일반 SR Latch의 Set state에서 Reset, Reset State에서 Set이 작동하고, Hold state는 Forbidden, Forbidden state는 Hold가 작동한다. SR Latch가 input이 bitwise not을 취한 것처럼 작동하므로, 이 SR Latch를 2개를 연결하여 Negative Reset Master-Slave JK Flip-Flop을 구성한다면, Negative Reset Master-Slave JK Flip-Flop은 일반적인 Master-Slave JK Flip-Flop과 똑같이 작동하게 된다.

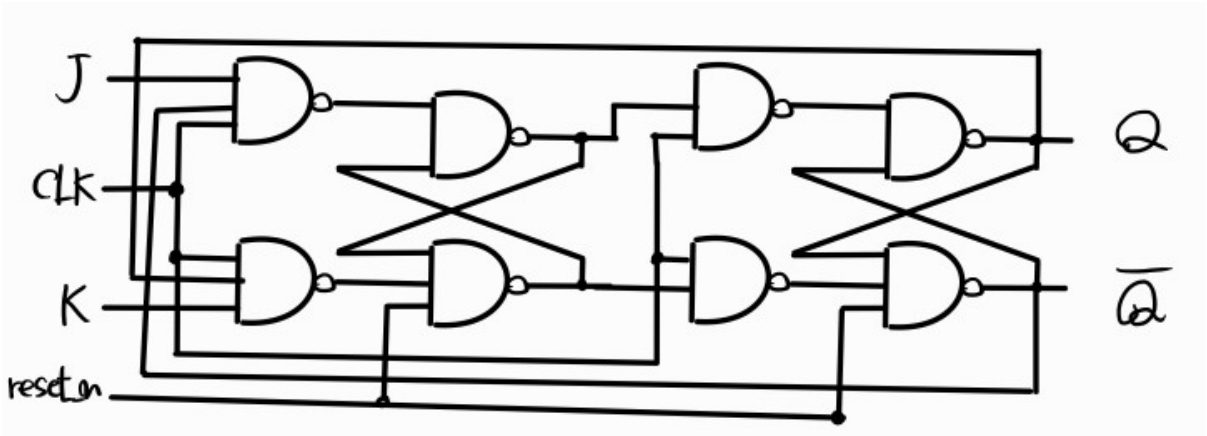
따라서 Negative Reset Master-Slave JK Flip-Flop의 Excitation table은 [표 4]와 같다.

[표 4] Negative Reset Master-Slave JK Flip-Flop의 Excitation table

	S(t)	R(t)	Q(t)	Q′(t)	Q(t+Δ)	Q′(t+Δ)
Hold	0	0	0	1	0	1
Hold	0	0	1	0	1	0
Reset	0	1	0	1	0	1
Reset	0	1	1	0	0	1
Set	1	0	0	1	1	0
Set	1	0	1	0	1	0
Toggle	1	1	0	1	1	0
Toggle	1	1	1	0	0	1

[그림 5]의 SR Latch를 이용해 Negative Reset Master-Slave JK Flip-Flop의 회로도를 구성하면 [그림 6]과 같다.

[그림 6] Negative Reset Master-Slave JK Flip-Flop의 회로도



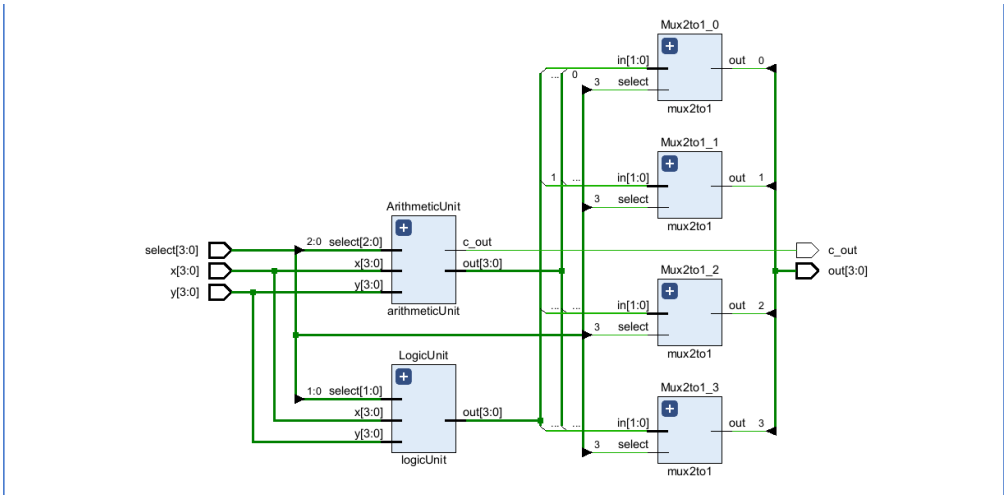
SR Latch는 Clock이 없으므로, input의 Glitch에 output이 무조건 영향을 받게 된다. 따라서 모든Glitch를 해결할 수 없다. 그러나 JK Flip-Flop은 Clock이 1일 때만 Input의 영향을 받으므로, Clock이 0일 때의 Glitch를 해결할 수 있다. 그러나 Clock이 1일 때 생긴 input의 Glitch(1's Catch)는 여전히 해결할 수 없다.

4. 결과

4.1. Lab5_1

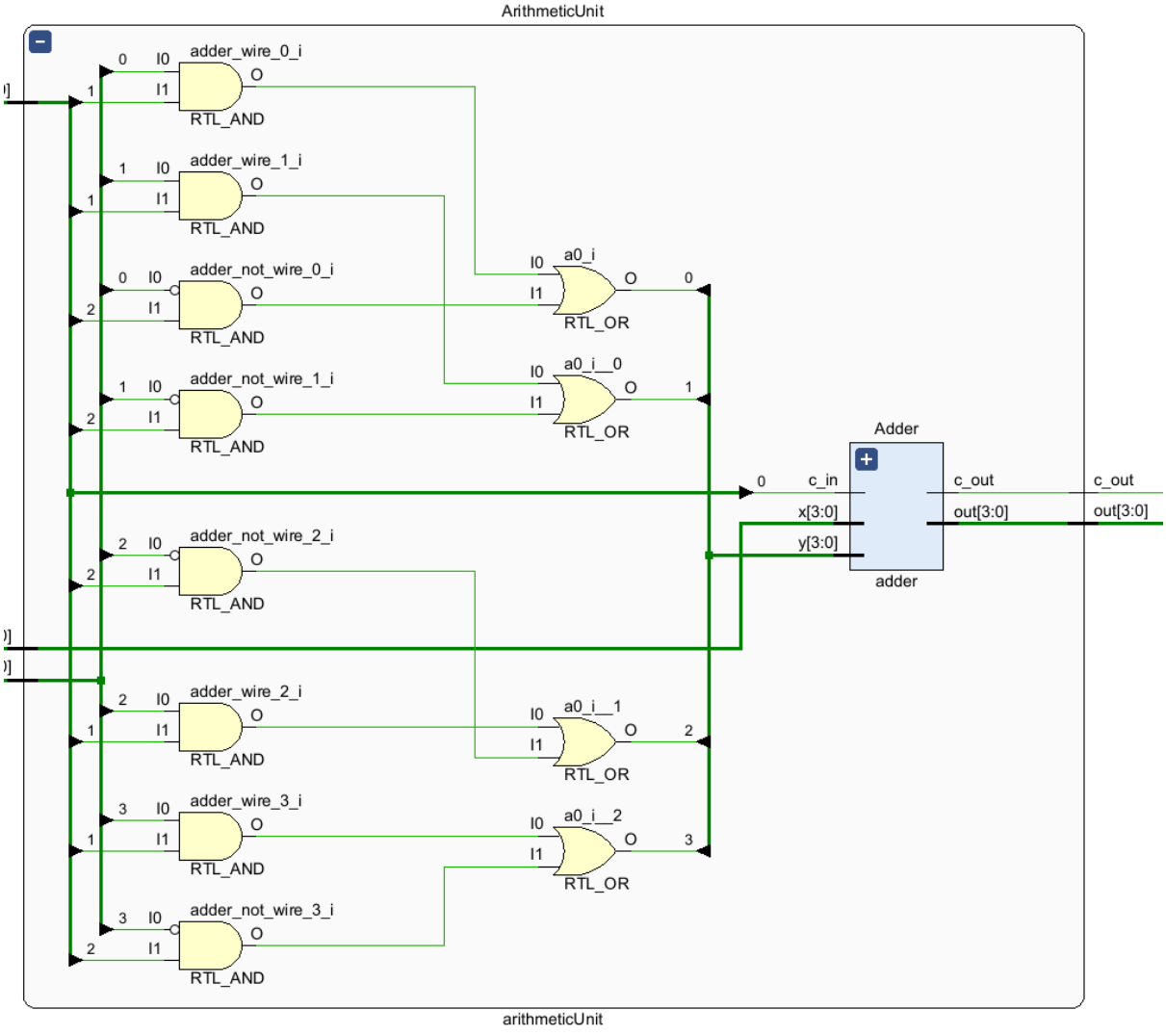
[그림 7]은 ALU의 Schematic이다.

[그림 7] Schematic of ALU



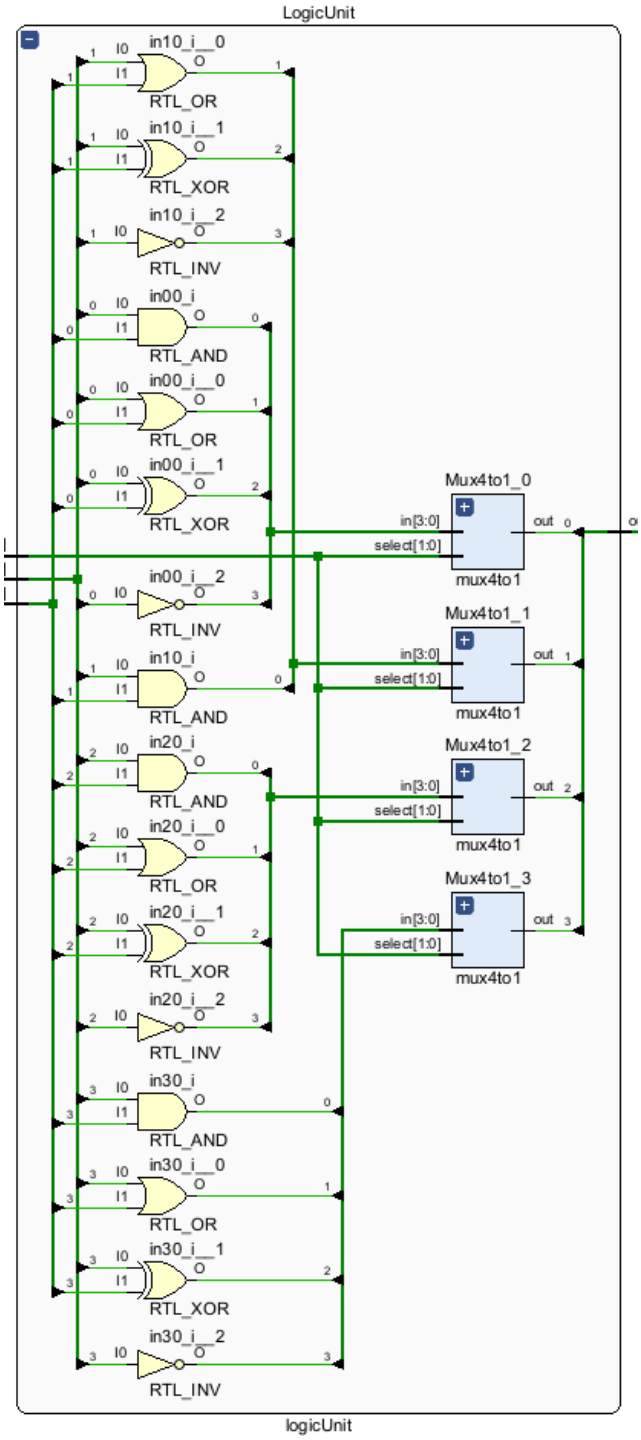
[그림 8]은 Arithmetic unit의 Schematic이다.

[그림 8] Schematic of Arithmetic unit



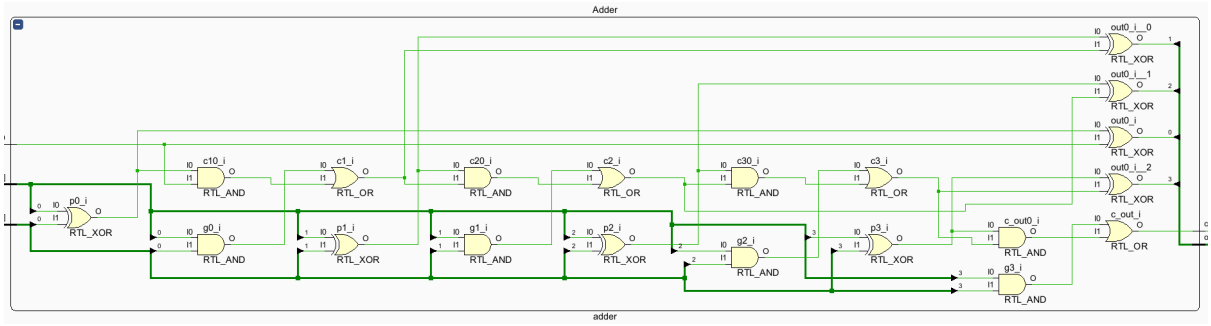
[그림 9]는 Logic unit의 Schematic이다.

[그림 9] Schemetic of Logic unit



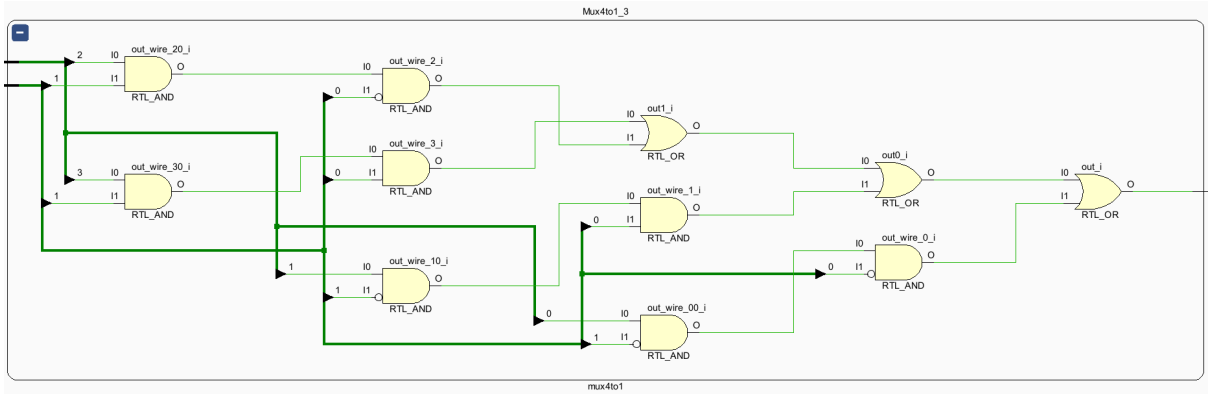
[그림 10]은 Adder의 Schematic이다.

[그림 10] Schematic of Adder



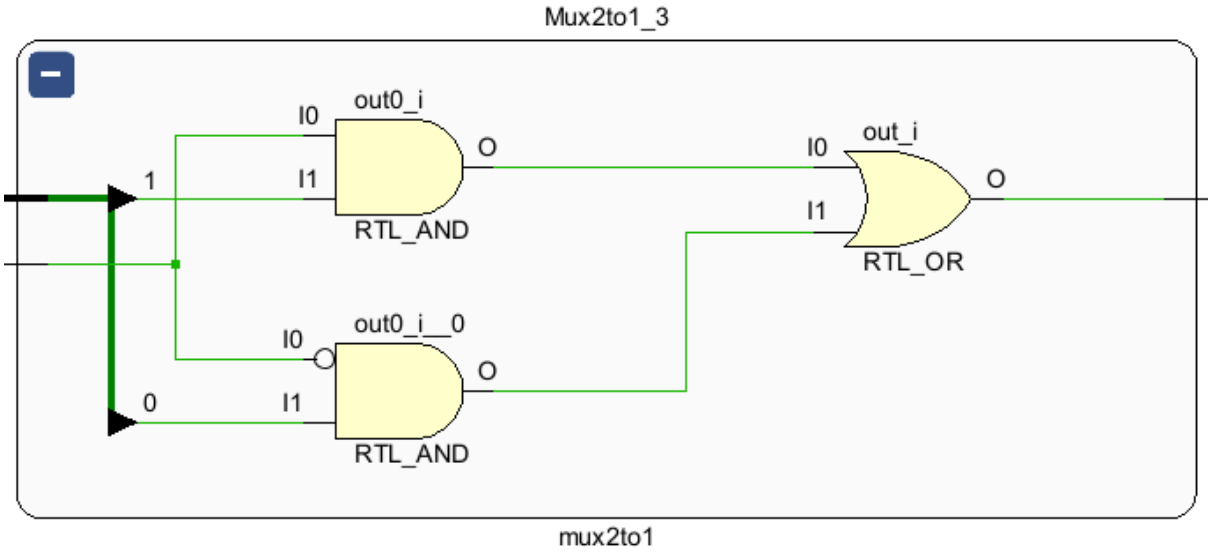
[그림 11]은 4 to 1 MUX의 Schematic이다.

[그림 11] 4 to 1 MUX



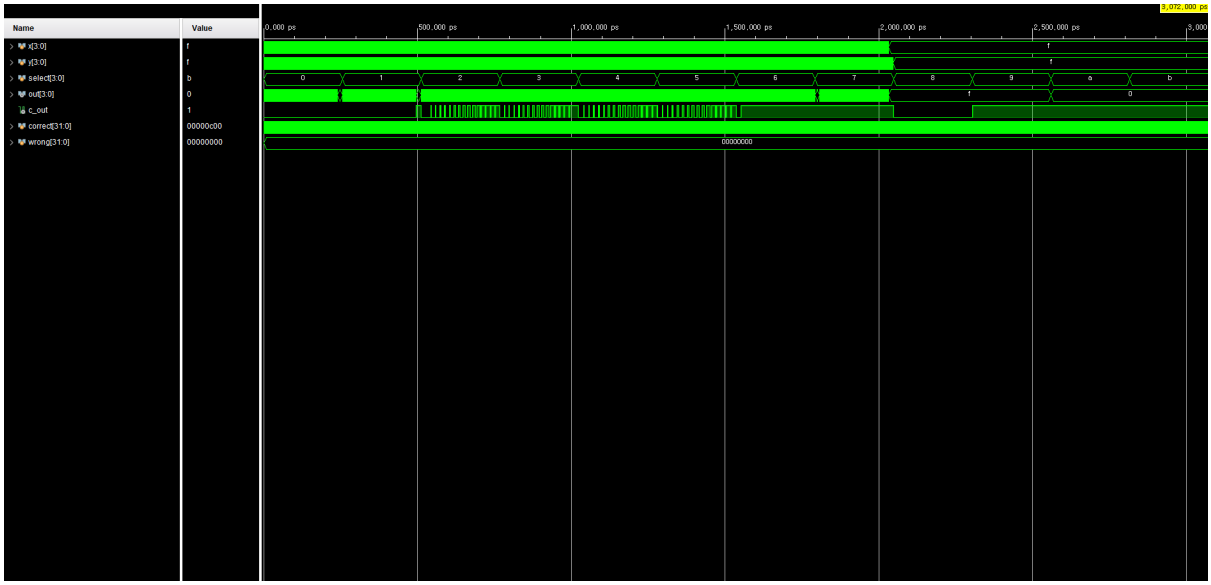
[그림 12]은 2 to 1 MUX의 Schematic이다.

[그림 12] 2 to 1 MUX



[그림 13]은 ALU의 Simulation이다.

[그림 13] Simulation of ALU

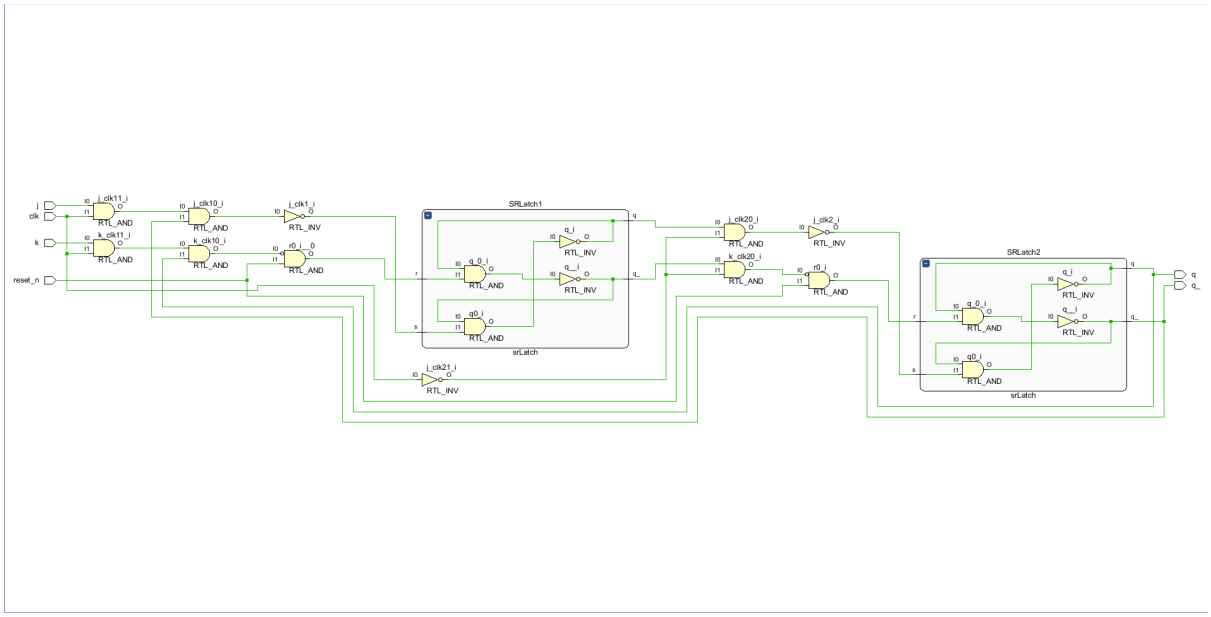


시뮬레이션 결과 Wrong이 0번 발생했으므로, 구현이 제대로 되었음을 알 수 있다.

4.2. Lab5_2

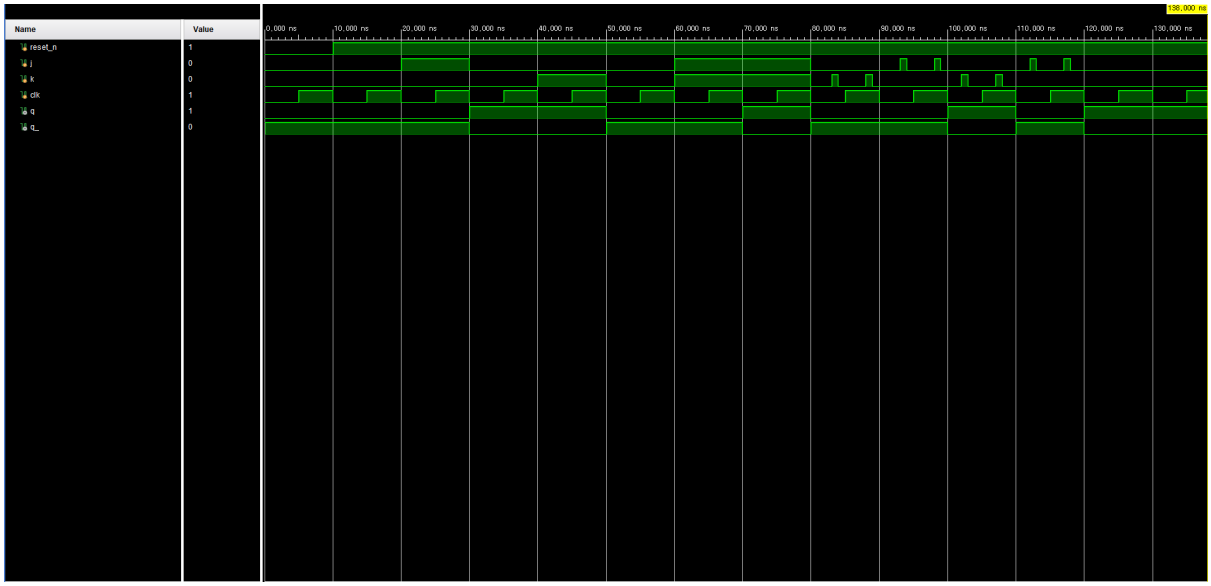
[그림 14]는 Negative Reset Master-Slave JK Flip-Flop의 Schematic이다.

[그림 14] Schematic of Negative Reset Master-Slave JK Flip-Flop



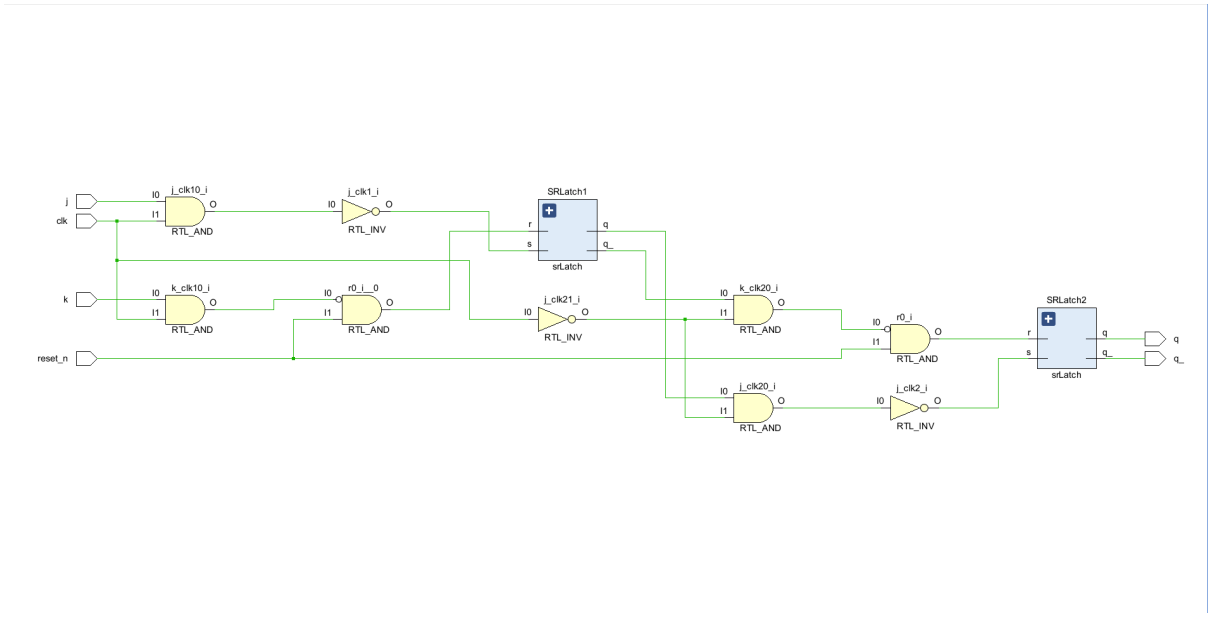
[그림 15]는 Negative Reset Master-Slave JK Flip-Flop의 Simulation 결과이다. Excitation이 정상적으로 작동하며, 0's Catch의 Glitch는 해결하지만 1's Catch의 Glitch는 해결하지 못하는 것을 알 수 있다.

[그림 15] Simulation of Negative Reset Master-Slave JK Flip-Flop



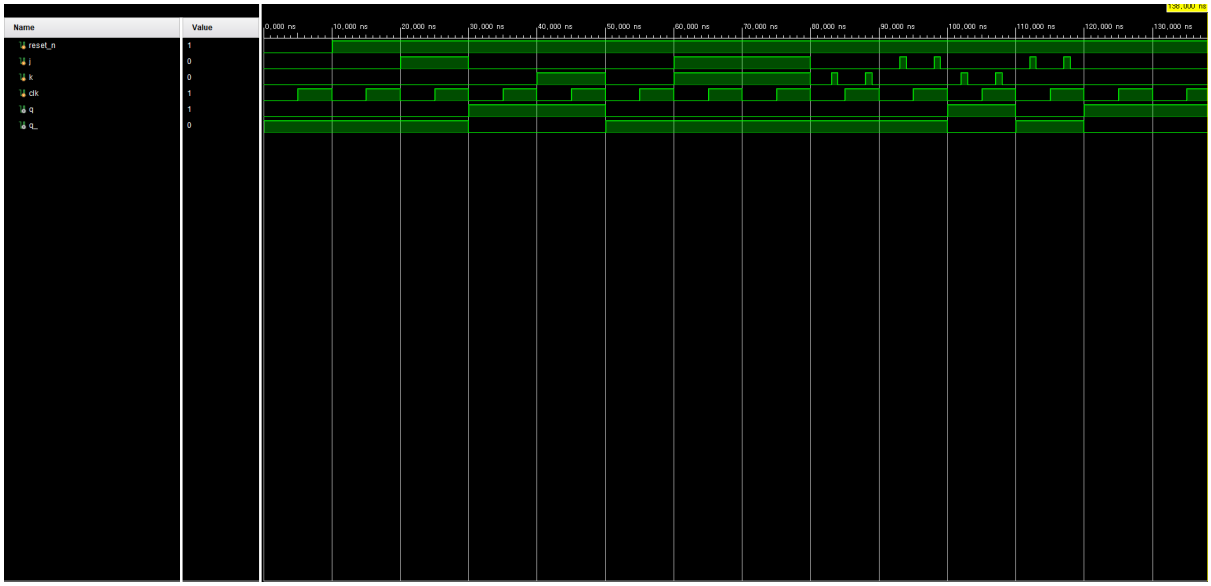
[그림 16]은 2개의 SR latch를 연결한 회로의 Schematic이다. (공지사항의 첫 번째 모듈)

[그림 16] Schematic of 2-SR latch connected circuit



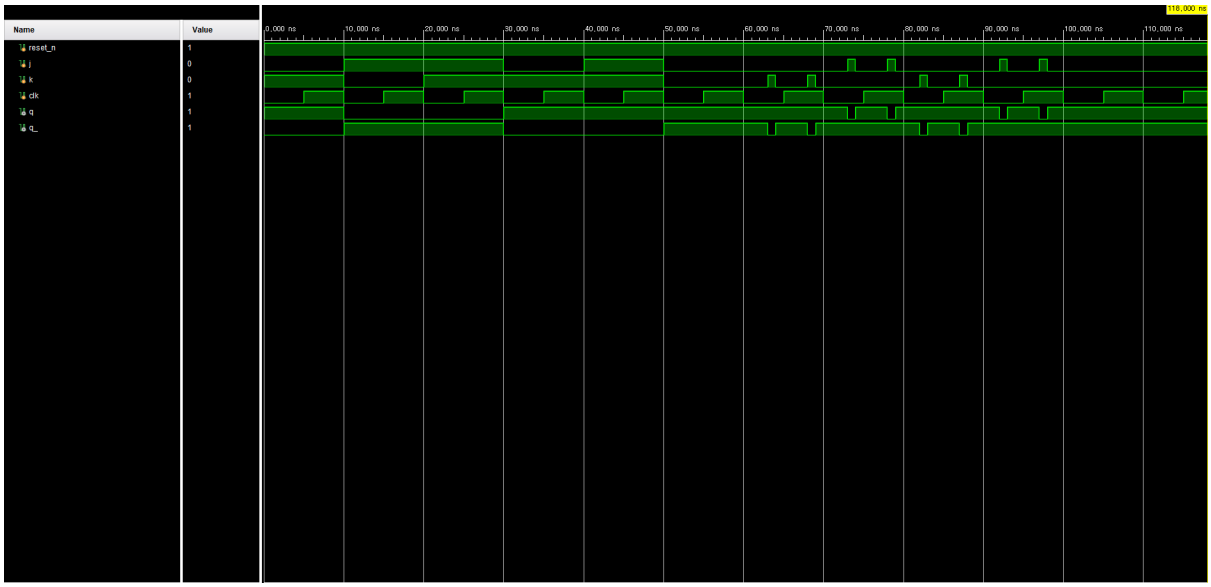
[그림 17]은 2개의 SR latch를 연결한 회로의 Simulation이다. 이 회로는 Negative Reset Master-Slave JK Flip-Flop처럼 0's Catch의 Glitch는 해결하고 1's Catch의 Glitch는 해결하지 못한다. Negative Reset Master-Slave JK Flip-Flop과 다른 점은 Toggling state에서 Toggling이 작동하지 않고 Hold가 작동하는 Glitch가 있다는 것이다.

[그림 17] Simulation of 2개의 SR latch를 연결한 회로



[그림 18]은 SR Latch의 Simulation이다.

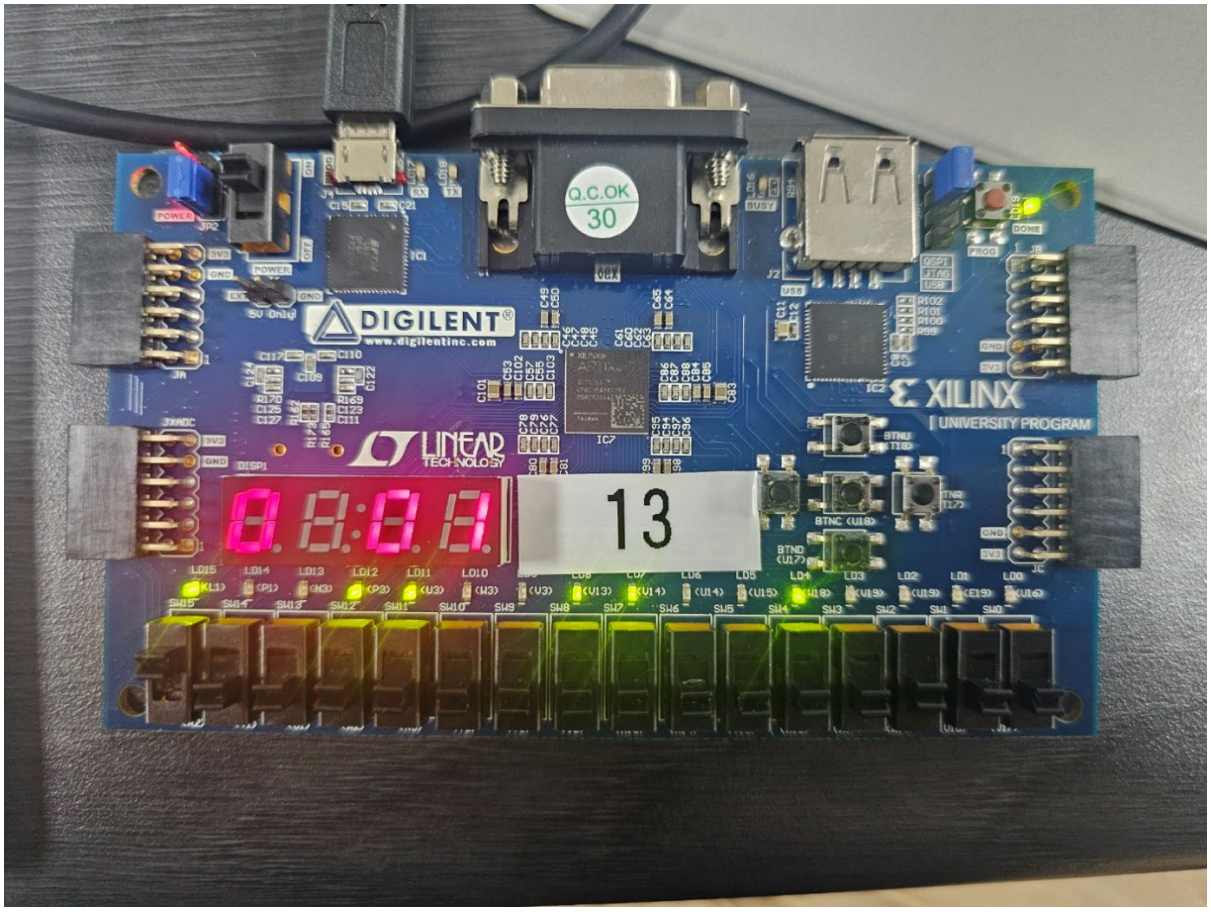
[그림 18] Simulation of SR Latch



SR Latch는 1's Catch와 0's Catch의 Glitch 모두 해결하지 못하며, Forbidden state가 존재한다.

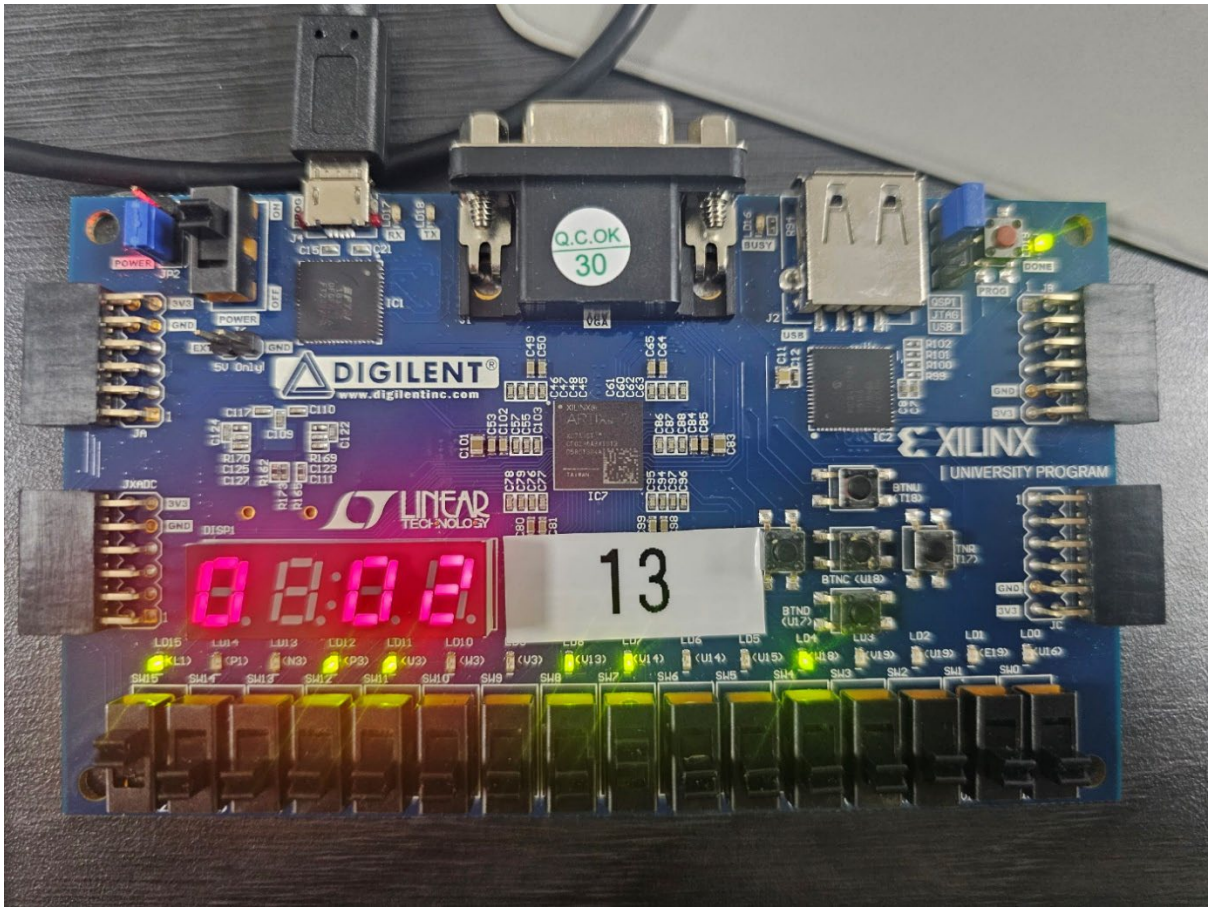
[그림 19]는 FPGA로 ALU의 Transfer를 구현한 사진이다.

[그림 19] FPGA ALU Transfer, x = 1, result = 1



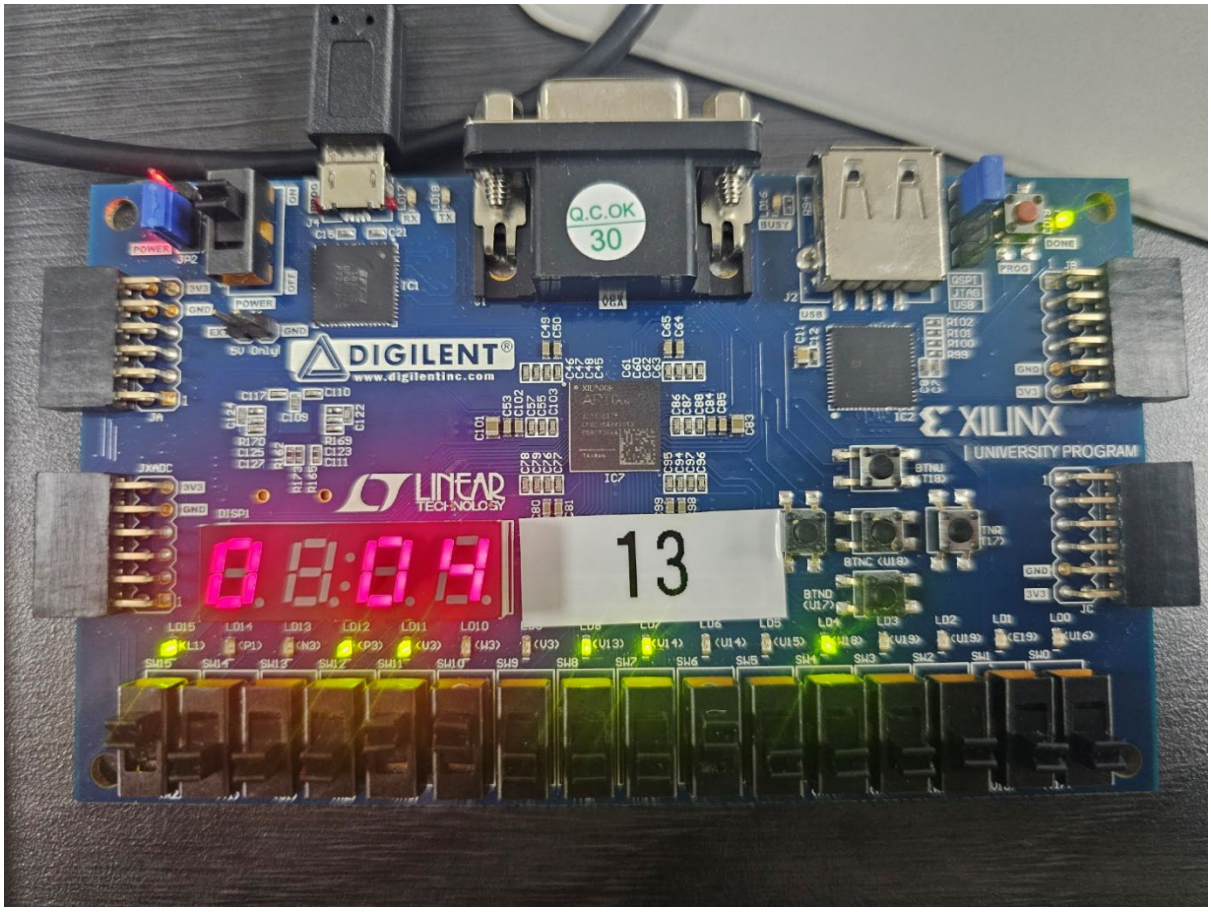
[그림 20]는 FPGA로 ALU의 increment를 구현한 사진이다.

[그림 20] FPGA ALU increment, x = 1, result = 2



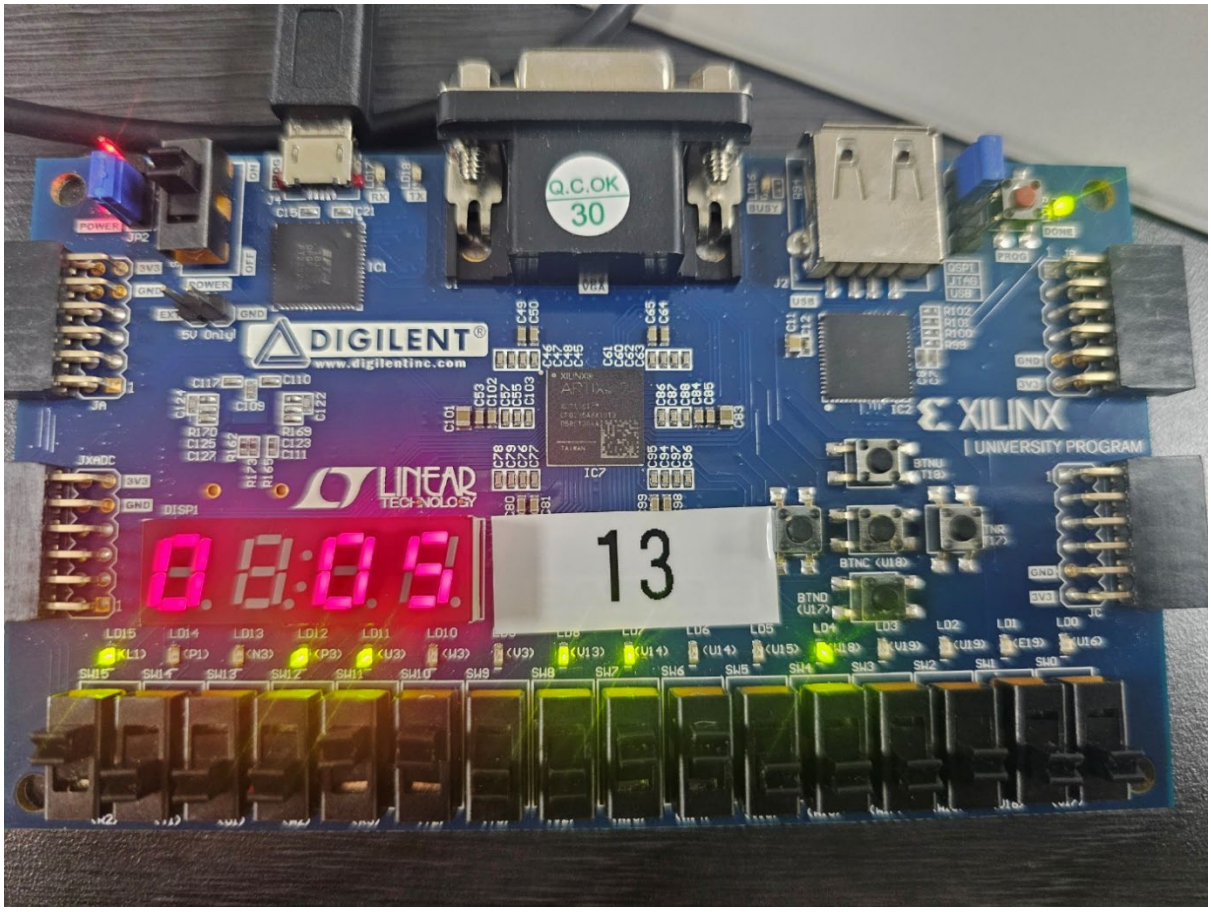
[그림 21]는 FPGA로 ALU의 Addition을 구현한 사진이다.

[그림 21] FPGA ALU addition, x = 1, y = 3, result = 4



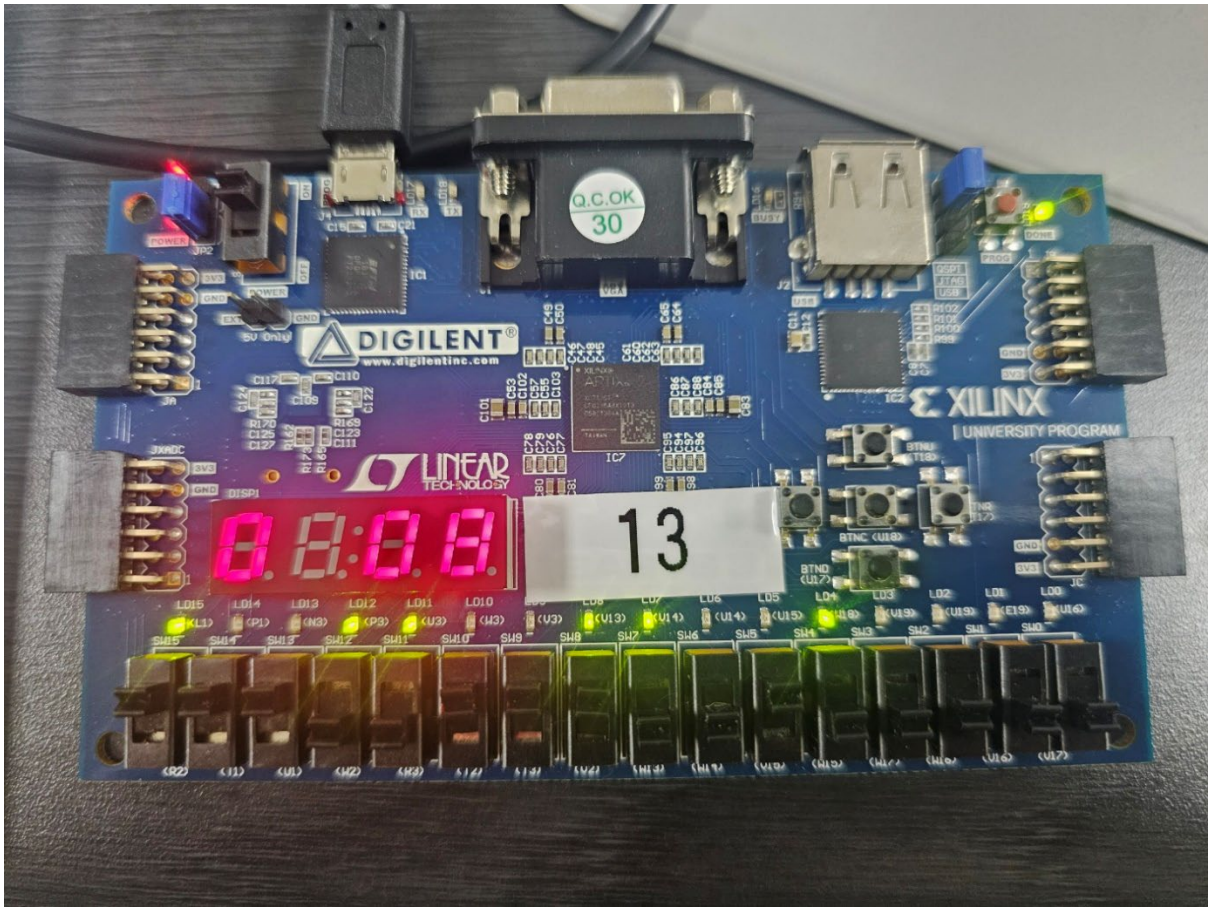
[그림 22]은 FPGA로 ALU의 x+y+1을 구현한 사진이다.

[그림 22] FPGA ALU x+y+1, x = 1, y = 3, result =5



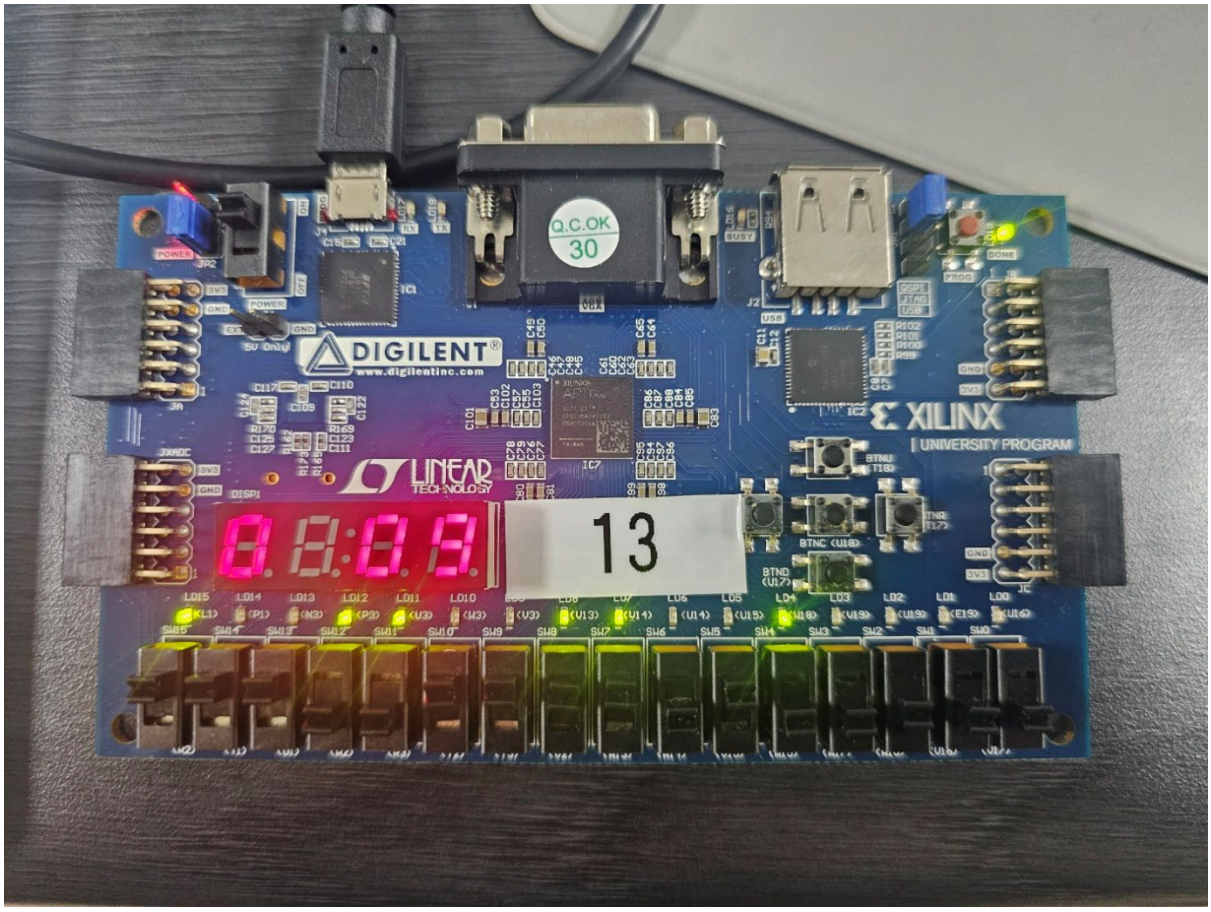
[그림 23]은 FPGA로 1C subtraction을 구현한 사진이다.

[그림 23] FPGA ALU 1C subtraction, x=7, y = 14, result = 8



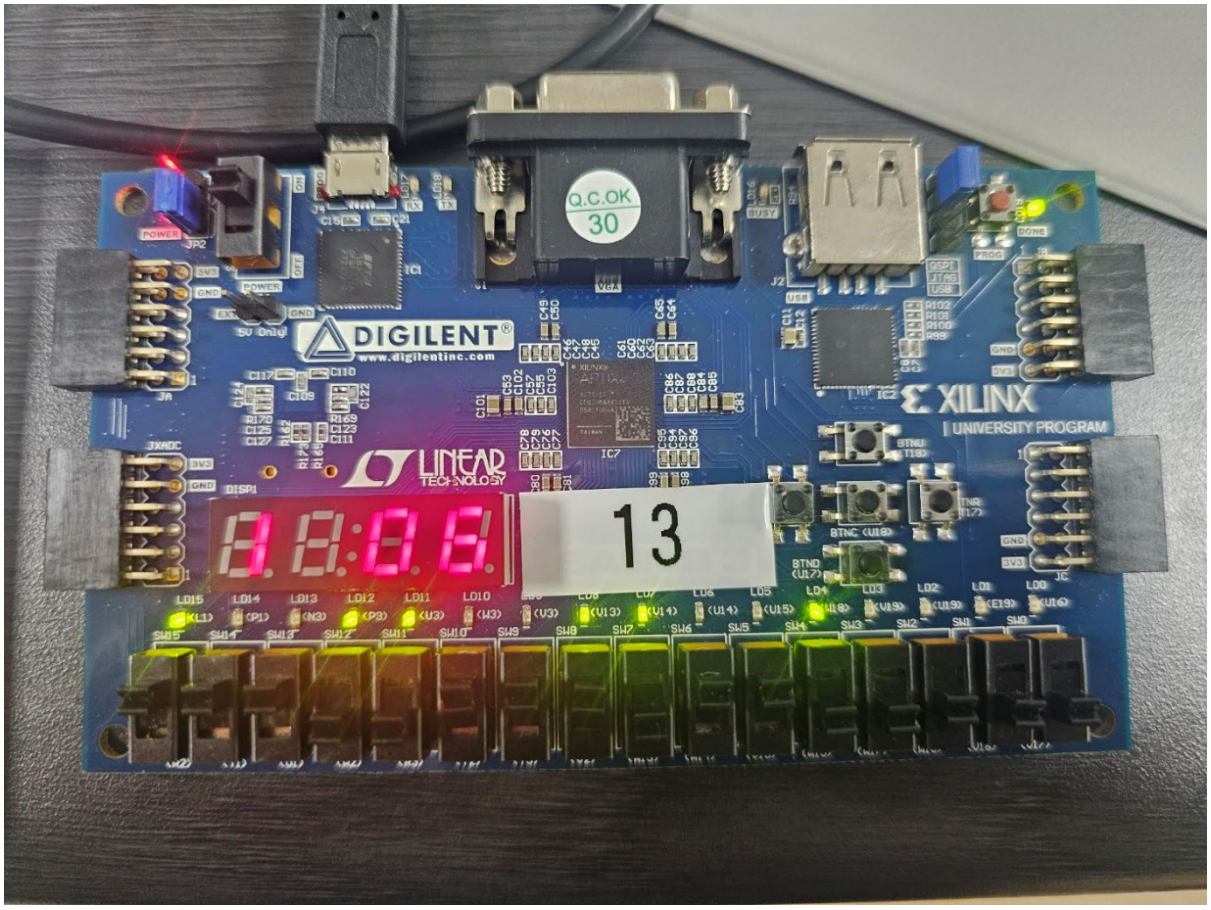
[그림 24]은 FPGA로 2C subtraction을 구현한 사진이다.

[그림 24] FPGA ALU 2C subtraction, x = 7, y = 14, result = 9



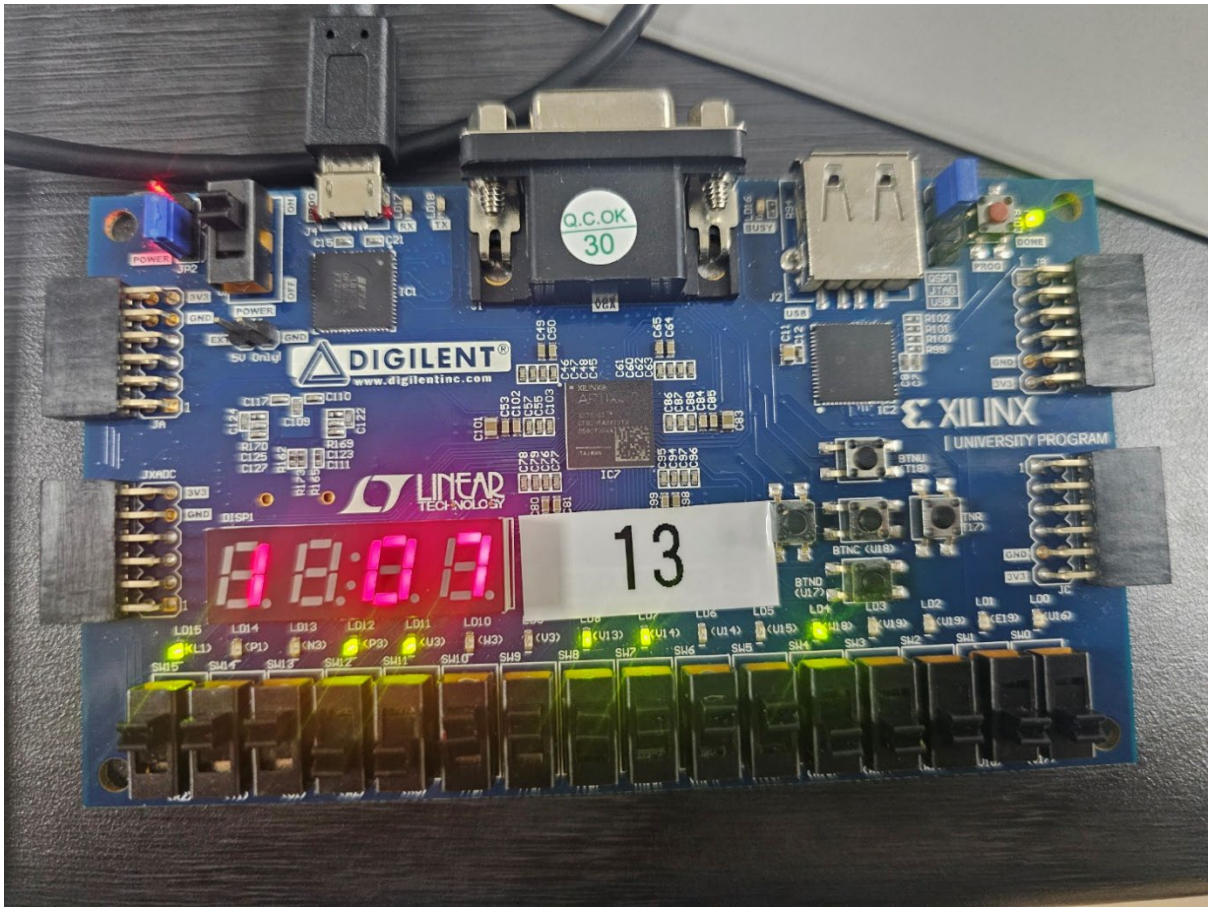
[그림 25]은 FPGA로 decrement을 구현한 사진이다.

[그림 25] FPGA ALU decrement x = 7, result = 6



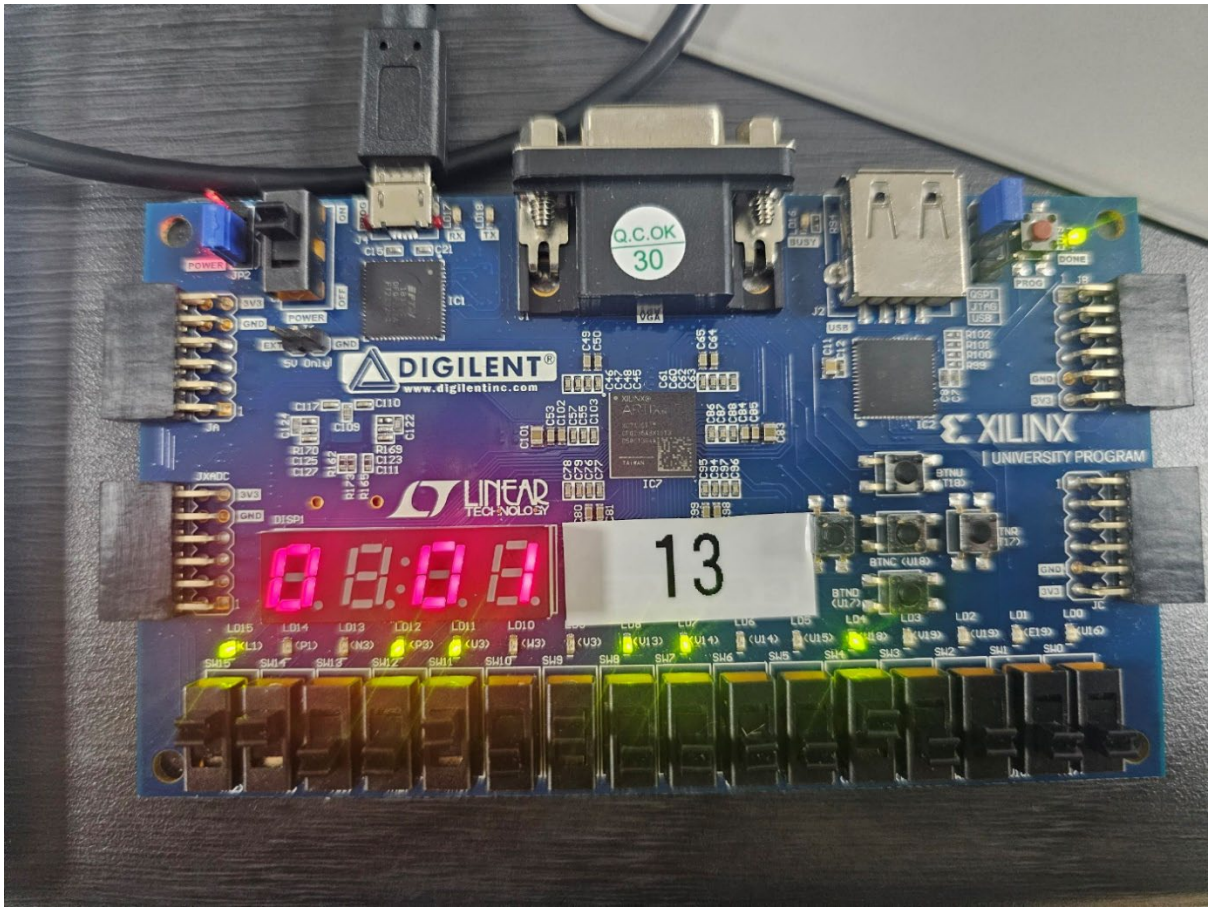
[그림 26]은 FPGA로 transfer를 구현한 사진이다.

[그림 26] FPGA ALU transferm x = 7, result = 7



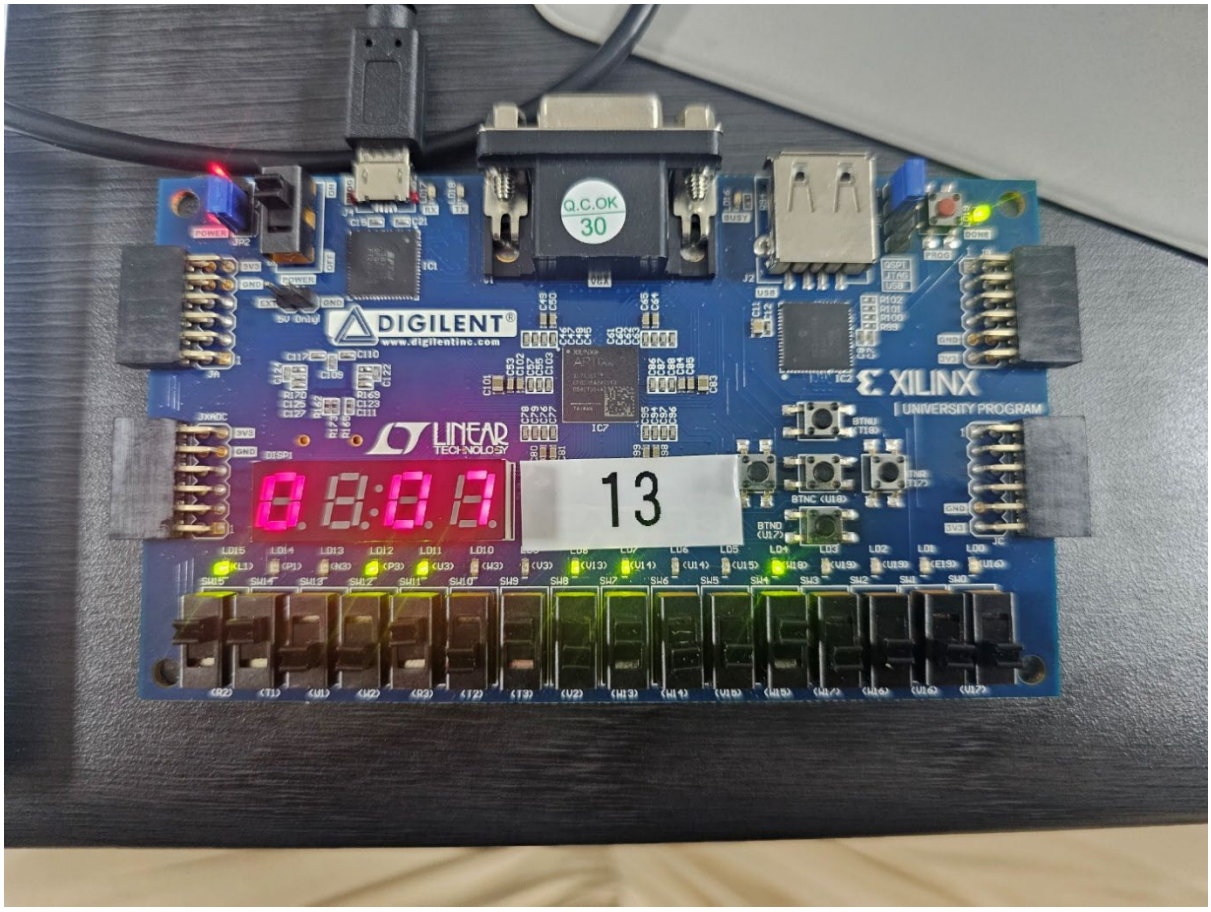
[그림 27]은 FPGA로 Bitwise And를 구현한 사진이다.

[그림 27] FPGA ALU Bitwise And, x = 0011, y = 0101, result = 0001 (1)



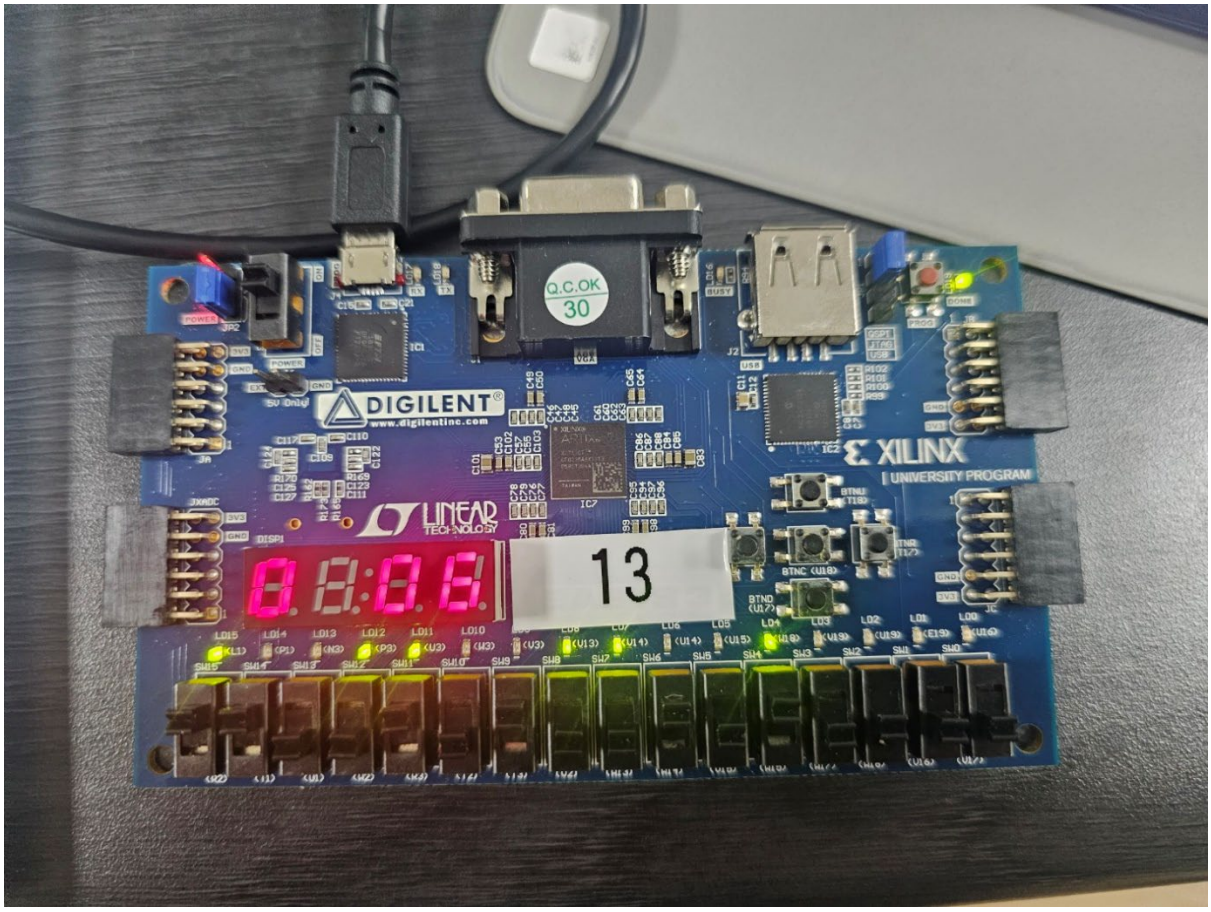
[그림 28]은 FPGA로 Bitwise Or를 구현한 사진이다.

[그림 28] FPGA ALU Bitwise Or, x = 0011, y = 0101, result = 0111 (7)



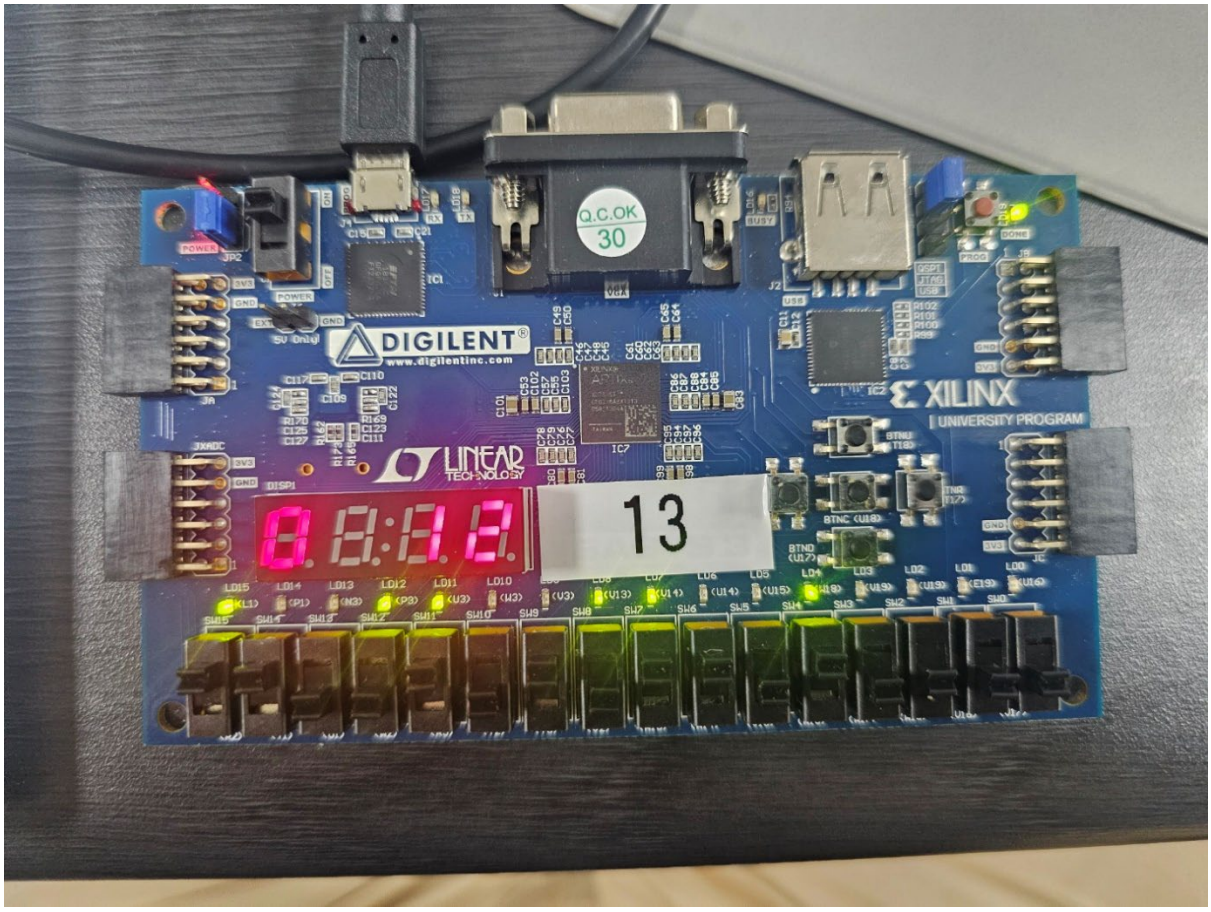
[그림 29]은 FPGA로 Bitwise Xor를 구현한 사진이다.

[그림 29] FPGA ALU Bitwise Xor, x = 0011, y = 0101, result = 0110 (6)



[그림 30]는 FPGA로 Bitwise Not을 구현한 사진이다.

[그림 30] FPGA ALU Bitwise Not, x= 0011, result = 1100 (12)



5. 논의

Lab5_1에서는 ALU의 Arithmetic Unit과 Logic Unit을 단순화시키고, 직접 구현할 수 있었다. Lab5_2에서는 JK Flip-Flop을 구현하면서 직접 Excitation Table을 작성하고, 회로를 구성할 수 있었다. 이번 Lab5는 기능이 조금씩 추상화되어 설계를 하는 것이 복잡했는데, 배웠던 여러 회로유닛을 이용하여 컴퓨터의 기본이 되는 연산을 할 수 있는 ALU와, 값을 저장할 수 있는 Flip-Flop을 구현해보니 디지털 시스템에 대한 이해가 많이 향상된 것 같다.

이번 Lab에서 SR Latch를 NAND로 구현하는 것이 가장 어려웠었다. Excitation Table을 작성할 때 Latch의 output이 두 번씩 바뀌는 경우가 있어서 회로 구현에 많은 시간을 쏟았었다. 이런 경우도 있음을 깨달을 수 있었다.