실험 6. 순차회로 – 계수기

20210774 김주은

1. 개요

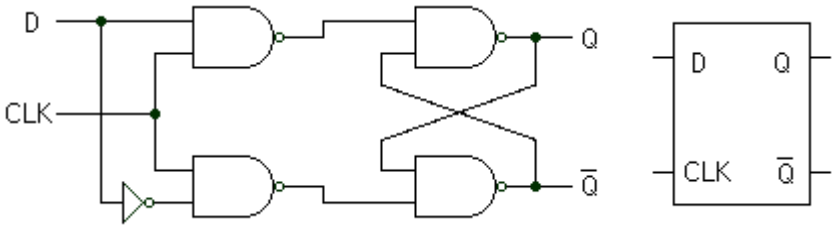
이번 실습의 목표는 순차회로의 대표적인 예시 중 하나인 계수기의 특성을 알아보고 다양한 계수기를 구현해본다. 계수기 중에서도 bcd 코드를 활용하여 돌아가는 계수기, 그리고 0~9까지 돌아가는 계수기 , 0~99 까지 돌아가는 계수기 등을 구현한다.

여기서 더 나아가 3 6 9 13 등의 순서로 돌아가는 계수기도 구현해보고, 이를 테스트하기 위한 테스트벤치도 구현하여 제대로 구현되어있는지 확인해본다.

2. 이론적 배경

1) D flip flop

D flip flop은 j k flip flop과 비슷하게 구현되지만 입력에 따라 output이 조금 다르다. D = Q+ 라는 식에 따라 D의 값에 따라 output의 값이 결정된다. 그리고 이러한 결과값은 clk의 신호에 따라서 output으로 나온다.



회로도는 이렇게 생겼고, 이에 따라 flip flop이 구동된다

2) 계수기(counter)

계수기란 순차회로의 일종이라고 할 수 있으며 정해진 패턴과 연속에 따라서 다음 state로 넘어가는 순차회로라고 할 수 있다. State들이 여러 개 구현되어 있고, 그 사이에서 state transition이 일어나는데, 이때 state transition은 clk의 신호와 미리 정해진 패턴에 따라 이루어진다. 그래서 이를 이용하여 counter을 구현할 수 있다. 사용자가 원하는 패턴을 디자인하고, 이에 따라 state transition이 일어나도록 flip flop까지 구현하면 원하는 기능을 하는 계수기가 만들어진다.

그리고 계수기가 동기 계수기와 비동기 계수기가 존재하고, 동기 게수기는 조합회로를 통해 클록 신호를 카운터의 모든 플립플롭에 동시에 넣는 계수기라고 할 수 있다. 비동기 계수기는 조합회로를 마찬가지로 이용하지만 클록 신호를 가장 최하단에 넣은 후에 그 상위 flip flop의 clk 신호에 그 전 플립플롭의 output을 넣는다. 그래서 이로 인해 작동 속도가 조금 더 느려질 수 있다.

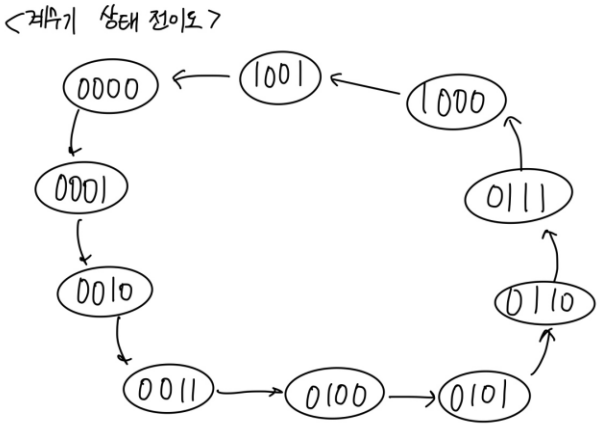
3) state transition diagram and state transition table

앞에서 말한 계수기 등 state transition이 일어나는 회로를 구현할 때 state transition diagram과 state transition table을 먼저 디자인해 주어야 한다. 먼저 처음 스테이트부터 어떻게 돌아갈 것 인지에 대해서 각 state에 대한 다음 state가 무엇인지에 대해 diagram을 그리는 데, 이는 state의 이름과, state 간의 화살표를 통해 표현한다.

그리고 state transition table을 디자인할 때는 state transition diagram를 참고하여 그린다. Present state와 next state에 대한 각 값을 다 입력하고, 각 present state에 대해 next state로 가기 위한 flip flop의 값을 같이 적어가며, 이 후 flip flop에 대한 식을 구하고, 회로도를 구현하는 데 까지 필요한 표를 그린다.

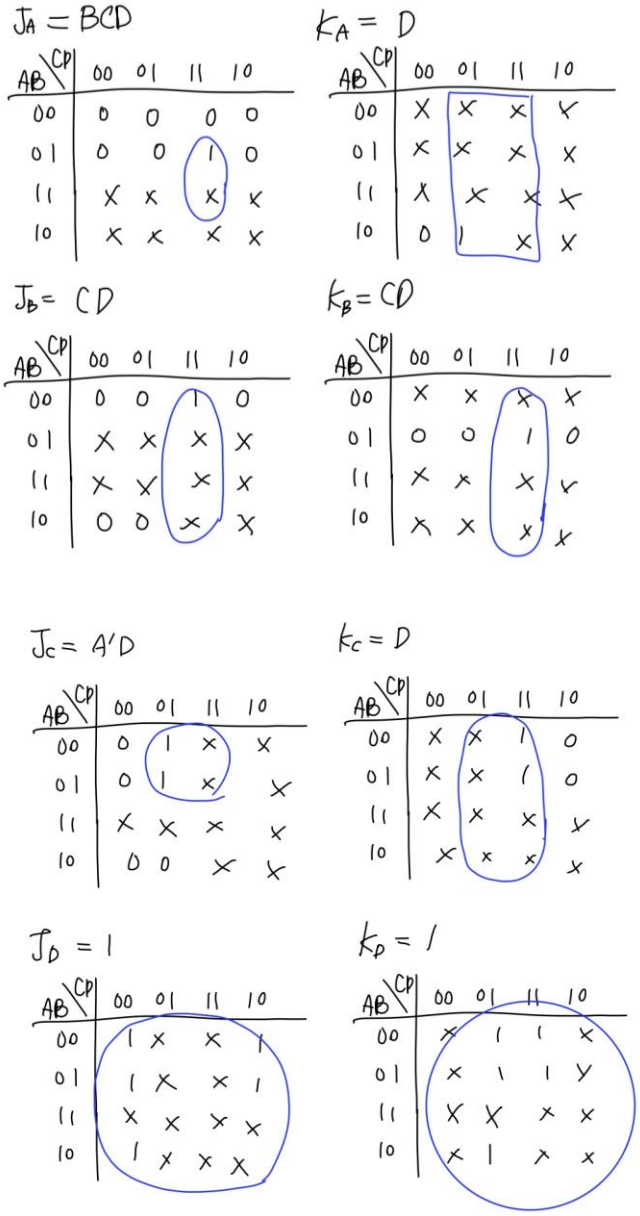
3. 실험 준비

Lab 6-1)



Present state	Next state	J(A)	K(A)	J(B)	K(B)	J(C)	K(C)	J(D)	K(D)
0000	0001	0	X	0	X	0	X	1	X
0001	0010	0	X	0	X	1	X	X	1
0010	0011	0	X	0	X	X	0	1	X
0011	0100	0	X	1	X	X	1	X	1
0100	0101	0	X	X	0	0	X	1	X
0101	0110	0	X	X	0	1	X	X	1
0110	0111	0	X	X	0	X	0	1	X
0111	1000	1	X	X	1	X	1	X	1
1000	1001	X	0	0	X	0	X	1	X
1001	0000	X	1	0	X	0	X	X	1
1010	xxxx	X	X	X	X	X	X	X	X
1011	Xxxx	X	X	X	X	X	X	X	X

1100	Xxxx	X	X	X	X	X	X	X	X
1101	Xxxx	X	X	X	X	X	X	X	X
1110	Xxxx	X	X	X	X	X	X	X	X
1111	xxxx	X	X	X	X	X	X	X	X

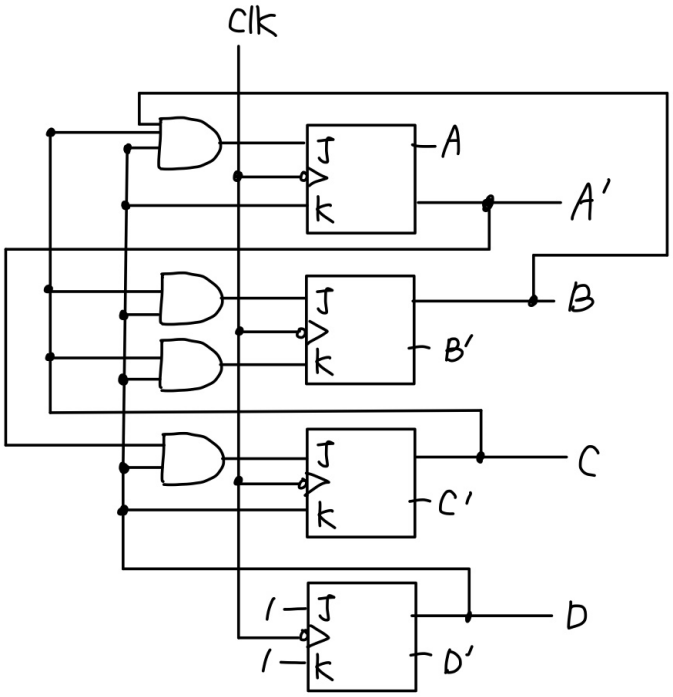


J(A) = BCD, K(A) = D

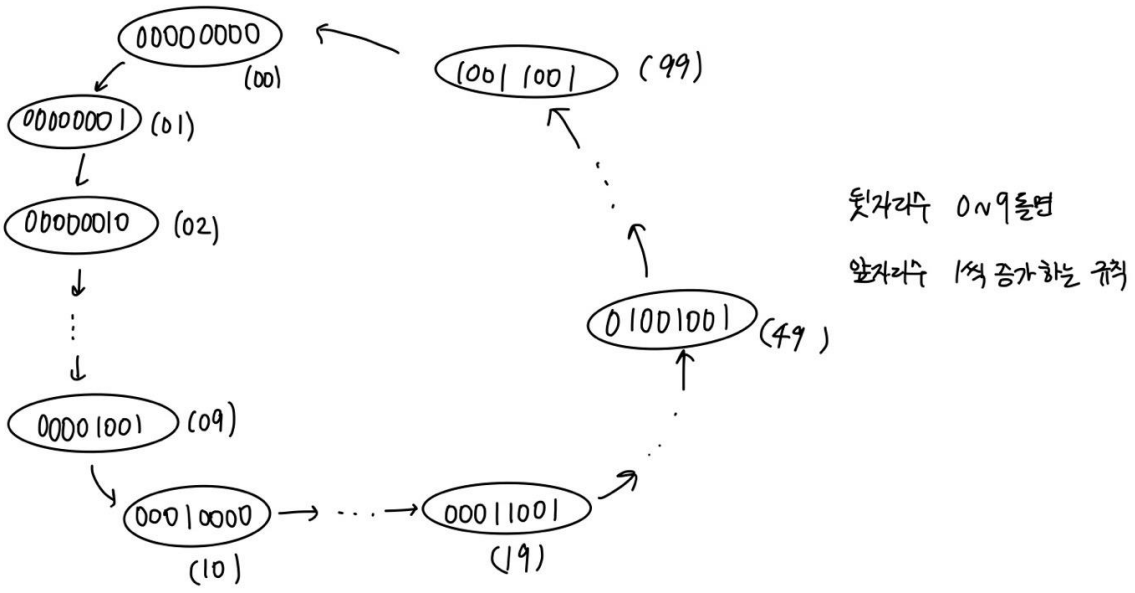
J(B) = CD, K(B) = CD

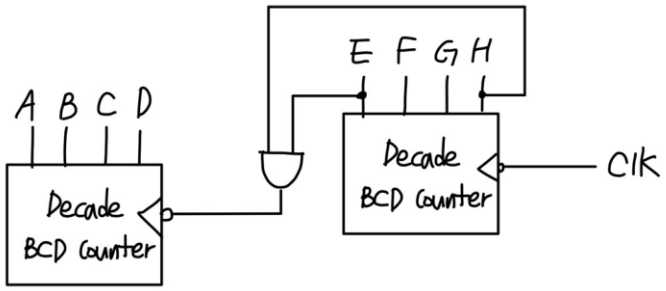
J(C) = A'D, K(C)=D

J(D) = 1, K(D) = 1

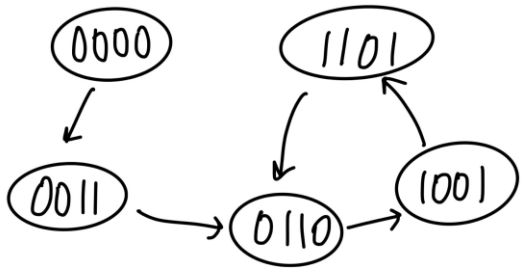

Clk
A
A'
B
B'
C
C'
D
D'
1
1
J
K

Lab 6-2)

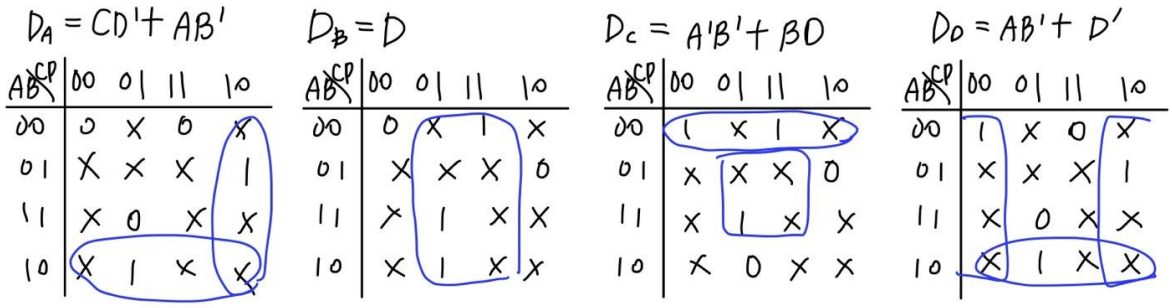

00000000 (00)
00000001 (01)
00000010 (02)
00001001 (09)
00010000 (10)
00011001 (19)
01001001 (49)
00101001 (99)
윗자리수 0~9돌면
윗자리수 1씩 증가하는 규칙

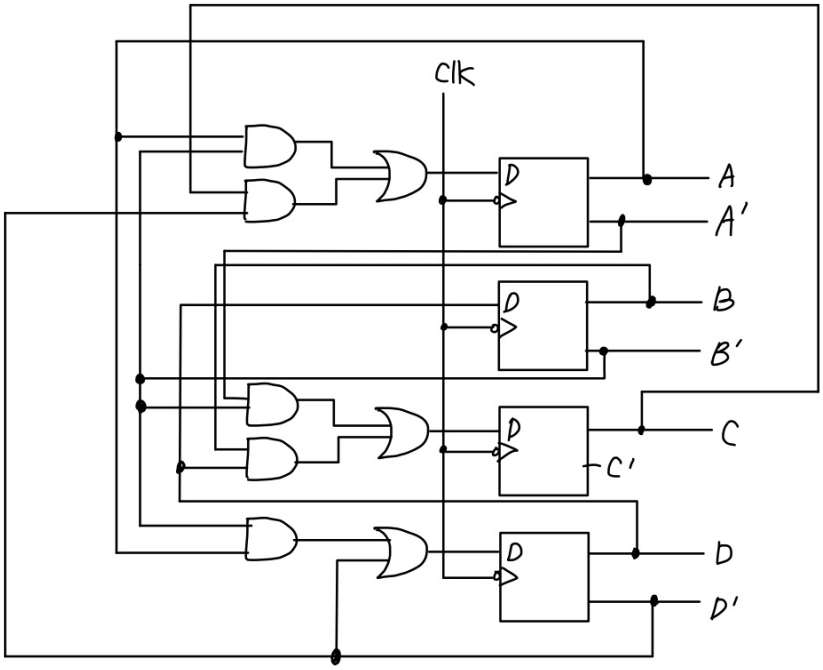

A B C D
E F G H
Decade
BCD Counter
Decade
BCD Counter
CIK

Lab 6-3)



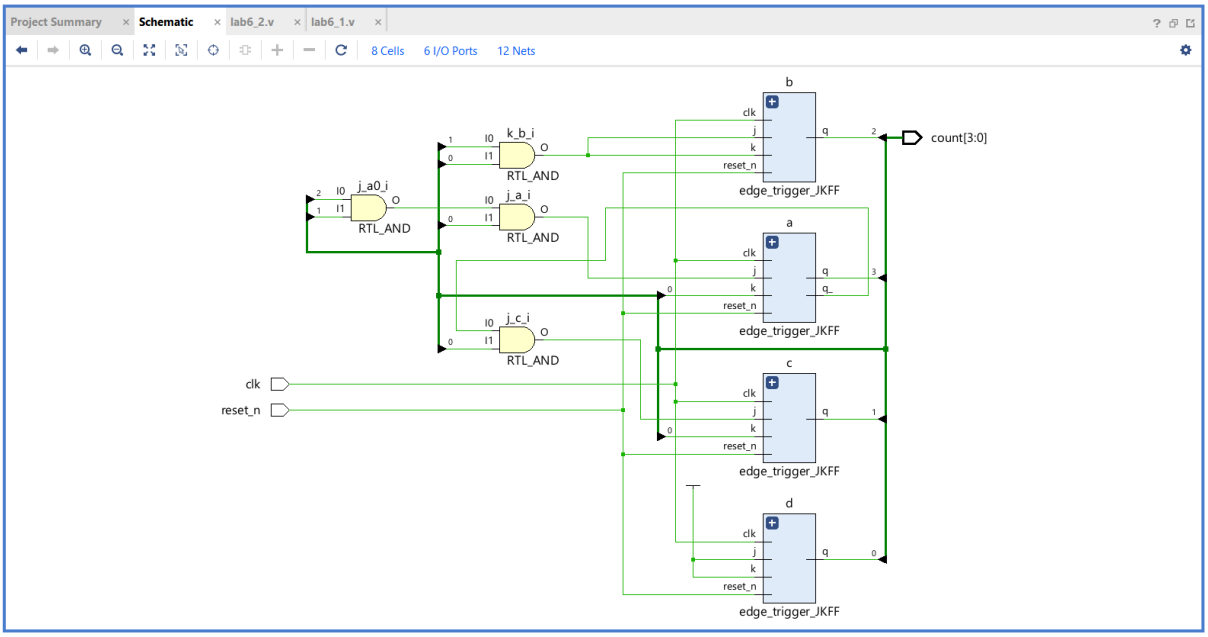
Present state	Next state	D(A)	D(B)	D(C)	D(D)
0000	0011	0	0	1	1
0001	XXXX	X	X	X	X
0010	XXXX	X	X	X	X
0011	0110	0	1	1	0
0100	XXXX	X	X	X	X
0101	XXXX	X	X	X	X
0110	1001	1	0	0	1
0111	XXXX	X	X	X	X
1000	XXXX	X	X	X	X
1001	1101	1	1	0	1
1010	XXXX	X	X	X	X
1011	XXXX	X	X	X	X
1100	XXXX	X	X	X	X
1101	0110	0	1	1	0
1110	XXXX	X	X	X	X
1111	XXXX	X	X	X	X



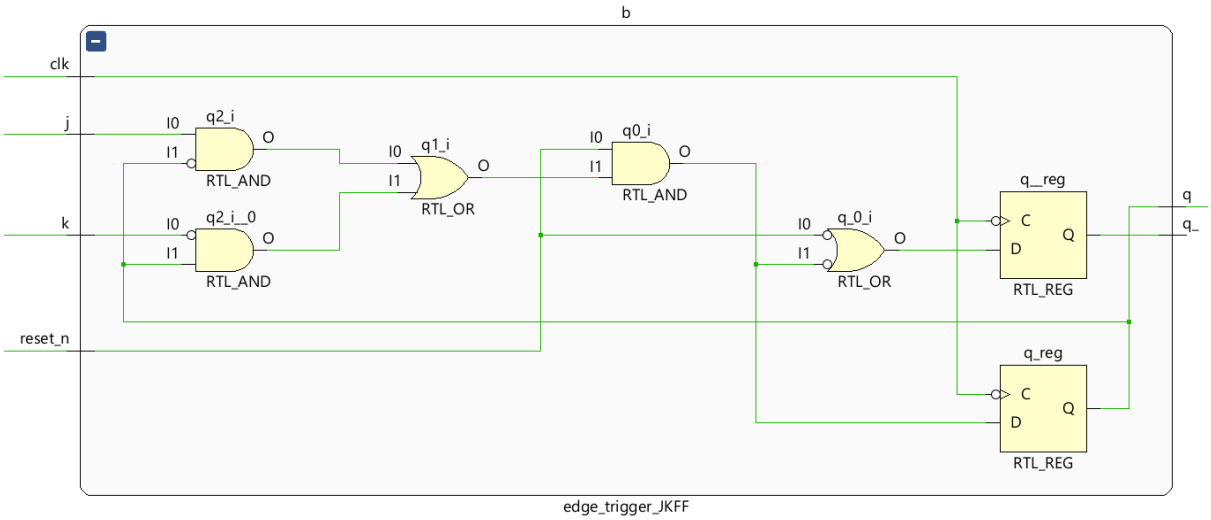


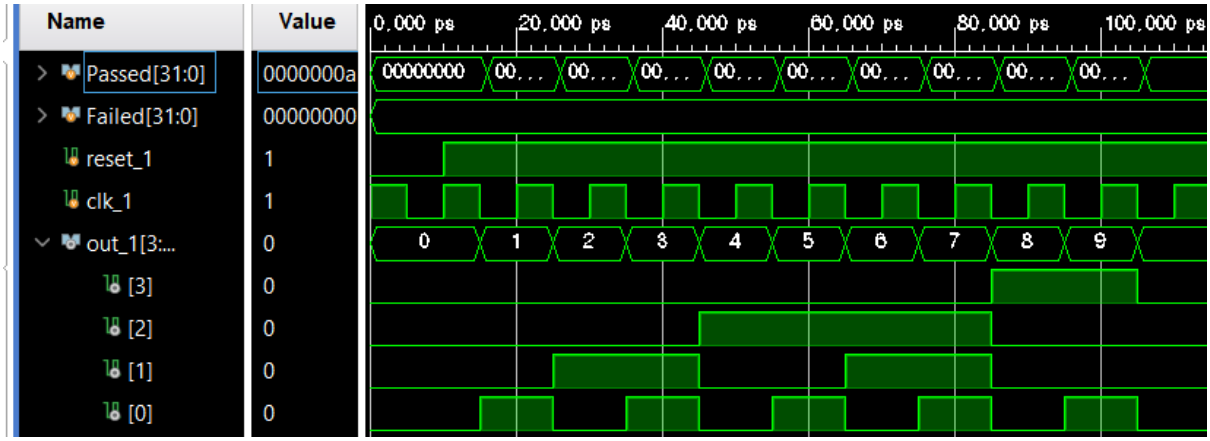
4. 결과

Lab 6-1)



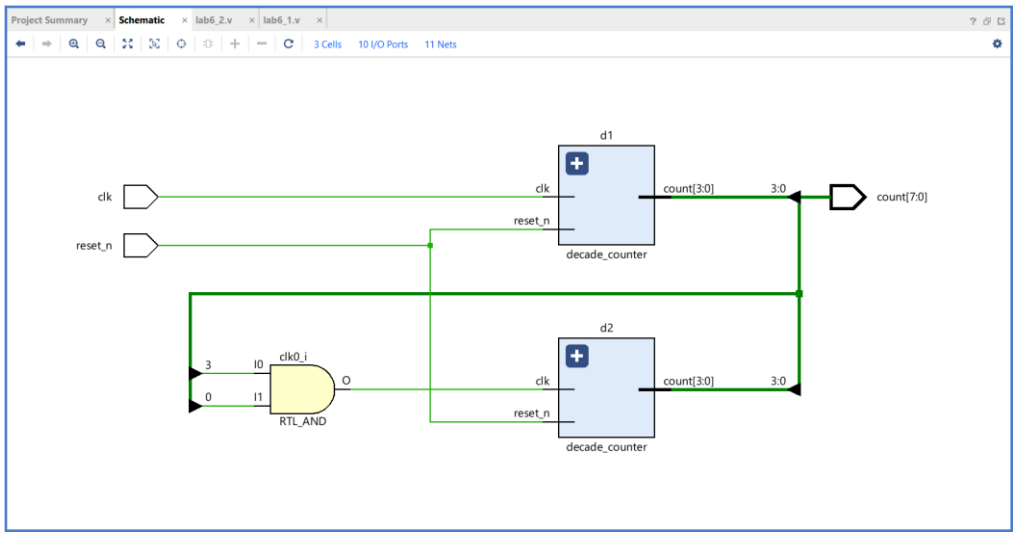
And gate가 실험 준비과정에서 직접 그린 회로도와 마찬가지로 4개로 구성되어 있고, 모든 구현이 똑같다.



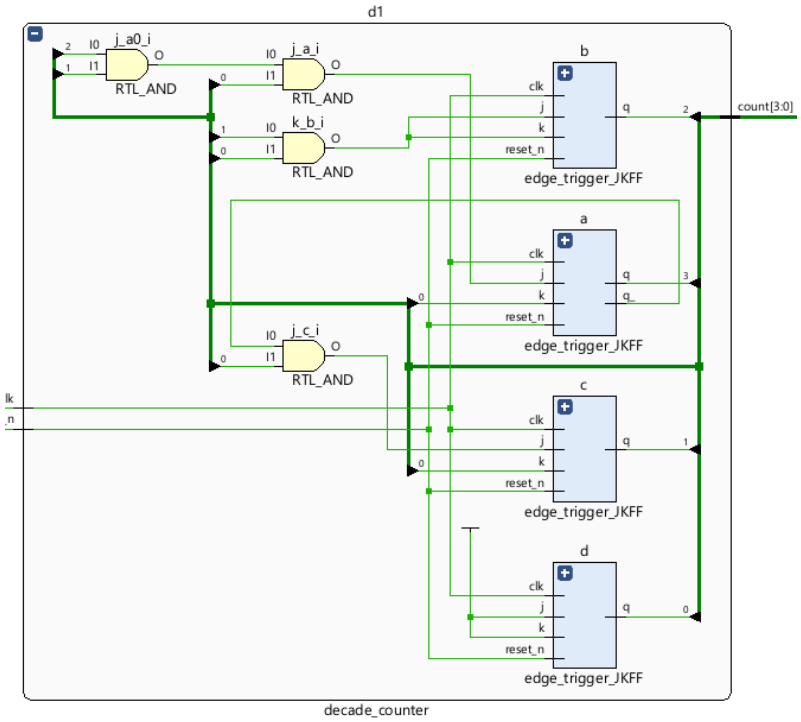


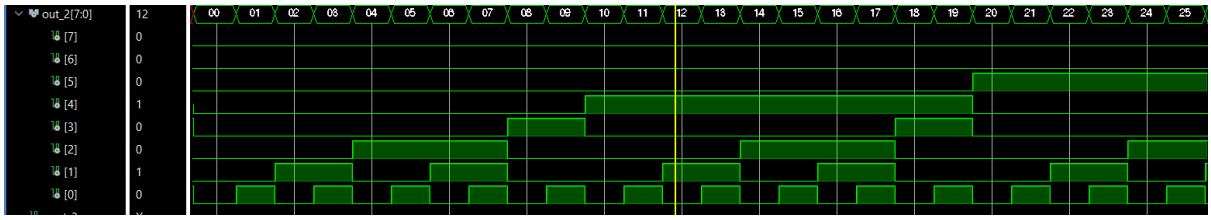
처음에 구현하고자 했던 0~9까지 clk 신호에 따라 하나씩 증가한다. 이로써 제대로 구현했는지 알 수 있었다.

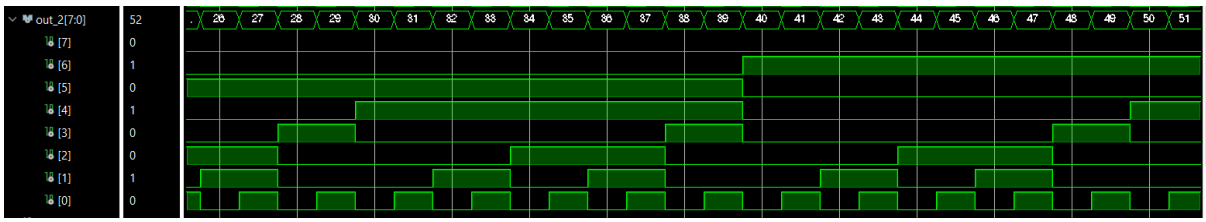
Lab 6-2)

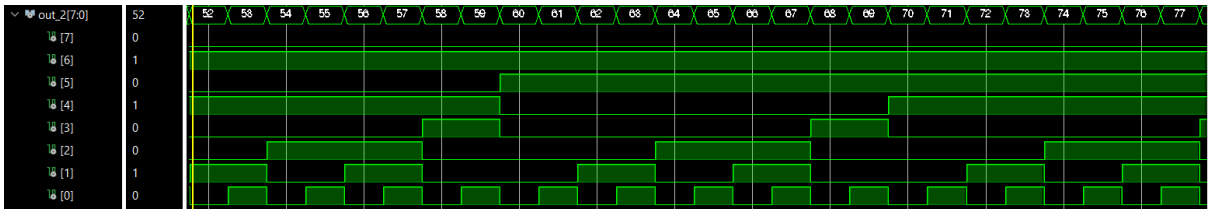


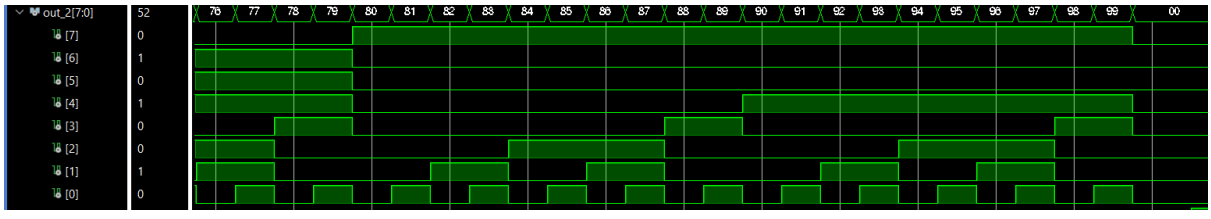
실험 준비과정에서 직접 그린 회로도와 마찬가지로 and gate로 연결된 두개의 counter로 구성되어 있고, 모든 구현이 똑같다.





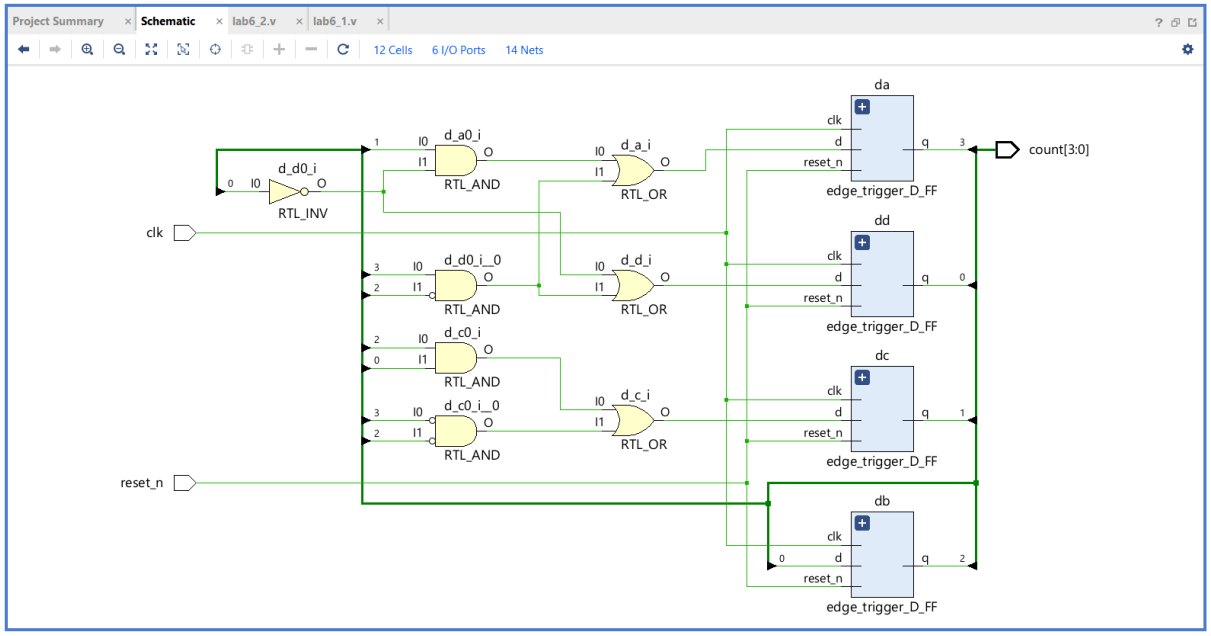


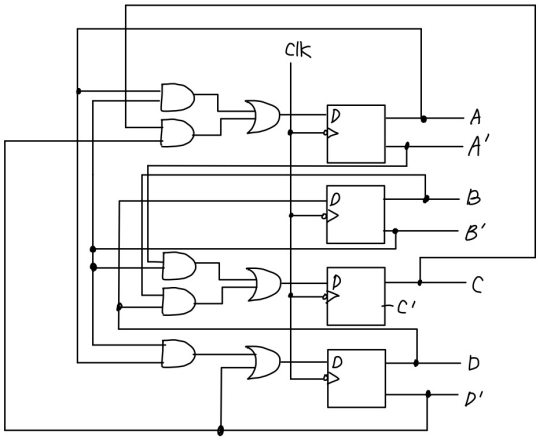




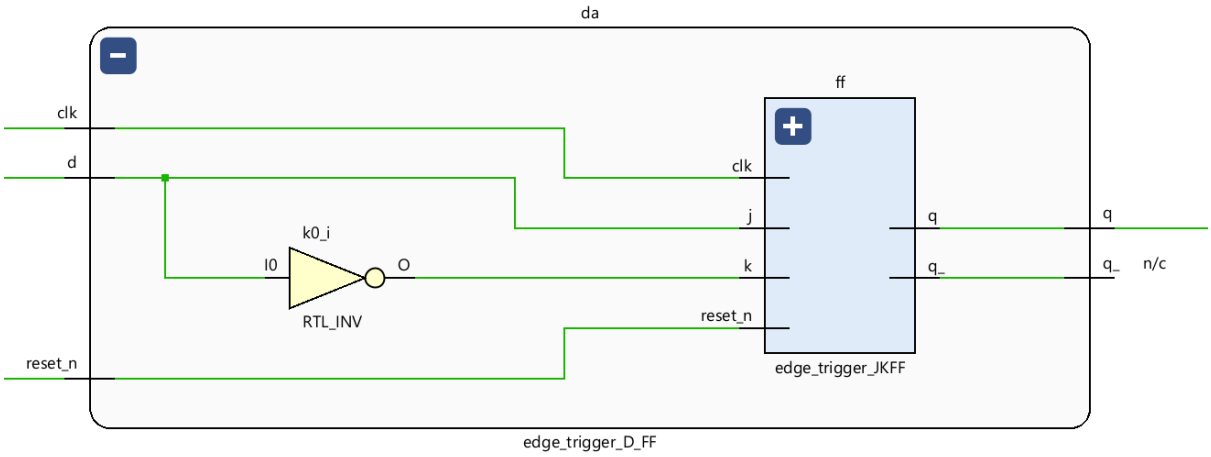
이도 마찬가지로 0~99까지 clock 신호에 따라 하나씩 증가한다.

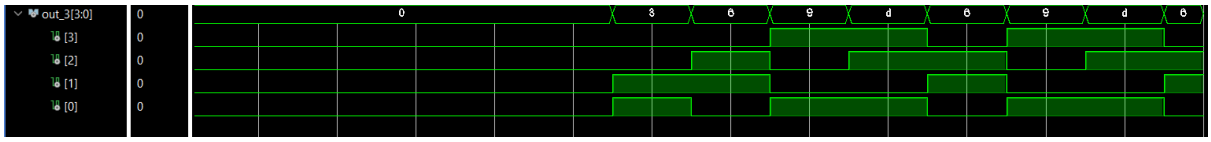
Lab 6-3)





앞에서 그린 회로도와 똑같이 구현되므로 제대로 코드로 구현했음을 알 수 있다.





이도 마찬가지로 3-6-9-13-3-6 과 같은 순서대로 clock 신호에 따라 하나씩 증가한다.

```
# run 1000ns
lab6_1_test
lab6_2_test
lab6_3_test
Lab6 Passed = 118, Failed = 0
```

마지막 console 창에 나와있는 passed와 failed의 개수를 통해 제대로 구현된 것을 확인할 수 있었다.

5. 논의

이번 랩도 저번 랩과 마찬가지로 내가 직접 테스트벤치를 구현하는 과제였다. 저번 랩에 테스트벤치를 작성하면서 베릴로그 문법을 익혔지만 이번에도 새로운 문제가 많았다.

Lab6_1의 테스트 벤치에서는 out_expected를 세팅하는 과정은 저번 랩과 비슷했기 때문에 어렵지 않았다. 그리고 clk_1을 5초 단위로 반복하면서 껐다 키는 세팅을 했다. 그 후 먼저 clk_1이 첫 번째로 0으로 내려가기 전에 reset_1을 0으로 설정해주었다. 이 후 clk_1이 된 후 out_expected를 다 0000으로 세팅하고 reset_1을 1로 세팅해준다. 이후 10번을 반복하며 out_expected를 1을 더해주고 if 조건문을 넣어서 out_1과 out_expected를 비교해서 같으면 passed에 1을 더해주고, 다르면 failed을 넣어주기로 했다.

하지만 Lab6_2부터 문제가 생긴 게 있었다. 8비트 중에서는 7~4 비트는 초기화가 되지 않아서문제가 생겼다. 그래서 고민을 한 후 처음 빠르게 0~9까지 반복하여 7~4비트를 초기화해주고, 이후에 0~99까지 반복문을 돌리면서 lab6_1과 비슷하게 구현하니 성공할 수 있었다.

세번째 lab6_3은 0 3 6 9 13 6 9 13 을 반복하는 것이라서 앞에서 반복했던 것과 마찬가지로 반복할 수 없었지만 쉽게 구현을 할 수 있었다.