논리회로 실험 보고서

제출날짜	2018.06.14
학과	컴퓨터공학과
학번 이름	21622127 안종욱 21720903 조태식

목 차

제 1장

Ⅰ. 개요

○ 우리 실생활에서 유용하게 사용되고 있는 Traffic Controllor를 중심으로 하여금 수업시간에 실험을 통해 배운 Verilog와 Bread Board를 이용하여 프로젝트에 임하였다.

이 프로젝트는 2인 1조로 하여 진행하였으며 Bread Board, Verilog, 보고서 작성 등의 과제를 수행하였다.

제 2장

Ⅱ.팀원

1. 소개역할 및 계획

팀 구성원

학번	이름	학과
21622127	안종욱	컴퓨터공학과
21720903	조태식	컴퓨터공학과

팀원 역할

이름	역할
안종욱	보고서 틀 작성 ,bread board 제작 verilog 설계 및 코드 작성, logicworkds
조태식	설계 및 작성, 보고서 설명 작성, breadboard 설계 및 제작

< 계획 >

2. 팀 스케줄

○ 6월 1일 ~ 6월 5일

- 문제이해 : 프로젝트를 본 다음 수업시간에 배운 지식을 기반으로 각자 문제이해를 했음

○ 6월 6일 ~ 6월 8일

- 설계 및 회로작성 : 프로젝트를 설계한 다음 회로를 작성했음
- 6월 첫째 주에 한 문제 이해의 스위치 부분에 잇어서 오류를 찾고 다시 문제이해와 진리표 작성

○ 6월 9일 ~ 6월 11일

- Logicworks, Verilog, 빵판 : 각자 맡은 역할을 수행했음

○ 6월 11일 ~ 6월 14일

- 보고서 작성

제 3장

Ⅲ.문제이해

1. 과제 내용

* 교통 신호 제어기 설계

① 남북(NS), 동서(EW)의 교차로가 있는 교통신호제어기 디자인

: 남북(NS)을 주도로를 사용했으며 동서(EW)를 보조도로로 설정하였다.

② NS 도로가 교통 흐름이 많은 주도로이기 때문에 EW 도로에 차량이 없을 시 항상 녹색 신호 유지

: **스위치가 1일 때** : 보조도로에 차량이 있는 것으로 설정,
이 경우에는 정상적인 신호(그린->옐로우->레드->그린)로 반복하는 것을 원칙
스위치가 0일 때 : 주도로(NS)에서 녹색6초, 황색2초와
보조도로(EW)에서 녹색 3초, 황색2초가 지나고 난 후
다시 원점(NS=Green, EW=Red, Q3Q2Q1Q0=0000)으로 됐을 때

③ 카운터는 0000을 유지하되, NS은 green을 유지한다.
EW는 red를 유지하되, 다시 스위치가 1이 되면 정상적으로 카운터 작동되기 시작하면서 카운터에 따른 불빛으로 바뀜.

* 보조도로와 주도로가 Green이거나 Yellow일 경우 다른 한쪽은 무조건 Red이다.

* 다시 원점으로 가기 전까지 걸리는 시간은 주도로
NS 녹색 6초 + 황색 2초
EW 녹색 3초 + 황색 2초 = 총 13초

④ EW 도로에 차량이 있을 시 신호 바뀜

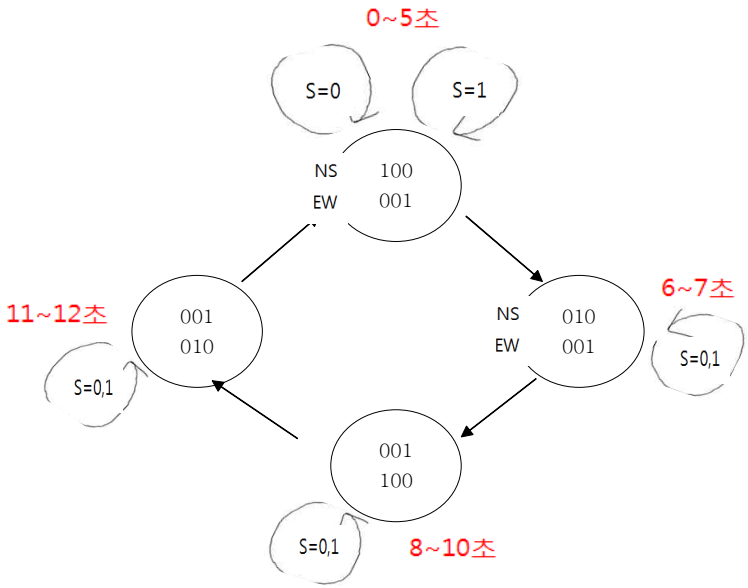
: 스위치가 0일 때 차량이 없다고 인지함(단, 차량이 없음을 유지하더라도 Q3 Q2 Q1 Q0=0000이 되기 전에 차량이 있다고 인지하여 스위치가 1이 되면 정상적인 신호를 계속 반복. (아무리 1~12초까지 스위치를 껐다 켜도 상관이 없고 단지 카운터가 0000 일때만 스위치가 꺼져있으면 차량이 없다고 인지를 한다는 것임)

⑤ Moore 로 설계

제 4장

Ⅳ.본론

4-1. 상태천이도 (S.T.T)



- 신호등 : 녹색, 황색, 적색 3가지

- NS 도로 신호 시간 : 녹색(6초) -> 황색(2초) -> 적색 (5초)

- EW 도로 신호 시간 : 녹색(3초) -> 황색(2초) -> 적색 (8초)

4-2. State Reduction

: 스위치가 0, 1일 때 상태가 어떻게 되는지 인지하지 못하였는데 정상적인 신호등일 때는 상태가 4개가 되는 것은 알고 있었다. 그럼로 4-2 단계를 넘어가되, SWITCH부분은 상태천이도에 추가하였다.

4-3. Binary Assignment

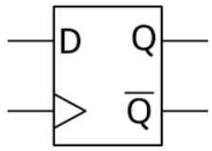
	G	Y	R
주도로, 보조도로	100	010	001

3가지 색깔(Green, Yellow, Red) 이므로 3bit 짜리로 하였다.

Switch	0,1
카운터	0000~1100

13초를 세어야 하였기에 0000(0초로 시작하여) 1100(12초로 끝)이므로 4bit 짜리로 하였다.

4-4. Choose a F/F (7474 ic chip)



<D Filp Flop>

D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

<True table>

D flip/flop은 다른 FlipFlop과 다르게 다음 상태를 그대로 가지기 때문에 선이나 여러 가지 측면에서 회로를 구성하기가 쉬웠다.

<입력>

D : 입력

S : Set에 1이 들어가면 Q는 1이다. (하지만 우리 팀이 쓴 칩은 negative 였고 우리는 사용하지 않기 때문에 1을 들어가야 했다.)
R : Reset에 1이 들어가면 Q는 0이다. (우리 팀이 쓴 칩은 set과 reset이 negative 이기 때문에 reset에 0이 들어가면 Q는 1이다. 그러므로 사용하지 않기 위해서는 reset에 1을 넣어야 했다.)

Q : 현재상태

Q+ : 현재상태에 따른 다음상태 (D와 현재상태 q에 따른 다음상태)

4-5. Logic Works Design

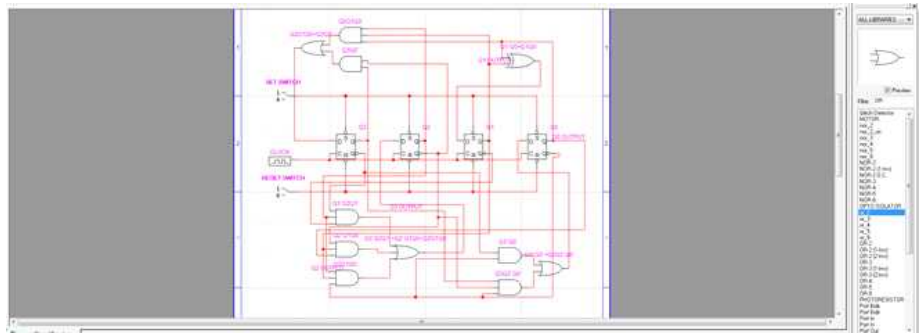
<Mod-13>

	Q 3	Q 2	Q 1	Q 0	Q 3+	Q 2+	Q 1+	Q 0+
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	1	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	0	1	0	0
13	X	X	X	X	X	X	X	X
14	X	X	X	X	X	X	X	X
15	X	X	X	X	X	X	X	X

: 표를 보다시피 Q3가 MSB이고 Q0가 LSB이며 Q3+ Q2+ Q1+ Q0+ 처럼 '+'가 달려있는 것은 다음 상태란 것이다. 굳이 up down 카운터 할 것 없이 그냥 단순히 up이 되고 0에서부터 12까지 하되, 12의 다음 상태는 0이 되도록 설계하였으며 4bit이므로 가짓수가 16개지만 그 외의 13, 14, 15 는 전부 다 무관항을 시켰다.

그리고 이것을 표로 만들어보면 위 그림같이 나오고 <4-6> 카르노맵이 나온다.

저기 회로도를 보면 알다시피 set과 reset은 쓰지 않으므로 각각 binary switch를 서서 1로 하였다.



준비물: D플립플롭 2개, 2input and gate 2개, 3input and gate 5개, XOR gate 1개, or gate 3개, 이거 reset switch 1개, set switch 1개 10ns clock 1개 그리고 모든 delay는 0NS로 해주었다.

결과 : 위의 그래프에 보다시피 0000 0001 ..... 1100 0000 0001.... 이 꾸준히 반복되는 13카운터를 만들었다.

4-6. karnaugh map

DC \ BA	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

$$Q_1 = \overline{Q}_1 Q_0 + Q_1 \overline{Q}_0$$

DC \ BA	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	0	X	X	X
10	0	0	1	0

$$Q_2 = \overline{Q}_3 Q_2 \overline{Q}_1 + \overline{Q}_2 Q_1 Q_0 + Q_2 Q_1 \overline{Q}_0$$

D=Q3, C=Q2, B=Q1, A=Q0 이다.

Q1 : 1과 돈케어 항끼리 묶은 후 2개의 식을 추출한다.

Q2 : 1과 1끼리 묶고, 1과 X끼리 묶은 후 3개의 값을 추출한다.

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	X	X	X
10	1	1	1	1

$$Q_3 = Q_3 \overline{Q}_2 + Q_2 Q_1 Q_0$$

DC \ BA	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	X	X	X
10	1	0	0	1

$$Q_0 = \overline{Q}_3 \overline{Q}_0 + Q_3 \overline{Q}_2 \overline{Q}_0$$

Q3 : 1끼리 묶고, 1과 X끼리 묶은 후 2개의 값을 추출한다.

Q0 : 1끼리 묶은 후 2개의 값을 추출한다.

32 \ 10	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	0	X	X	X
10	0	0	0	0

$$NS-G = \overline{Q}_3 \overline{Q}_2 + \overline{Q}_3 \overline{Q}_1$$

32 \ 10	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	X	X	X
10	0	0	0	0

$$NS-Y = Q_2 Q_1$$

D=Q3, C=Q2, B=Q1, A=Q0 이다.

NS_G (주도로 초록불) : 1과 돈케어 항끼리 묶은 후 2개의 식을 추출한다.

NS_Y (주도로 노란불) : 1과 X를 묶은 후 1개의 식을 추출한다.

32 \ 10	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	X	X	X
10	1	1	1	1

$$NS-R = Q_3$$

NS_R (주도로 빨간불) : 1과 X를 묶은 후 1개의 식을 추출한다.

32 \ 10	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	X	X	X
10	1	1	0	1

$$EW-G=Q_3\overline{Q}_2\overline{Q}_1+Q_3Q_1\overline{Q}_0$$

32 \ 10	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

$$EW-Y=Q_3Q_2+Q_3Q_1Q_0$$

D=Q3, C=Q2, B=Q1, A=Q0 이다.

EW_G (보조도로 초록불) : 1과 돈케어 항끼리 묶은 후 2개의 식을 추출한다.

EW_Y (보조도로 노랑불): 1과 X를 묶은 후 2개의 식을 추출한다.

32 \ 10	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	X	X	X
10	0	0	0	0

$$EW-R=\overline{Q}_3$$

EW_R (보조도로 빨간불) : 1끼리 묶은 후 1개의 식을 추출한다.

S98 Q2Q1Q0 \	000	001	011	010	110	111	101	100
00	0	0	0	1	1	0	0	1
01	1	0	0	1	X	X	X	X
11	1	0	0	1	X	X	X	X
10	1	0	0	1	X	0	0	1

$$D_0=Q_1\overline{Q}_0+\overline{Q}_3Q_2\overline{Q}_1\overline{Q}_0+Q_3\overline{Q}_2\overline{Q}_1\overline{Q}_0+S\overline{Q}_2\overline{Q}_1\overline{Q}_0$$

D=Q3, C=Q2, B=Q1, A=Q0 이다.

D0: 1과 돈케어 항끼리 묶은 후 3개의 식을 추출한다.

위의 표처럼 스위치는 D0에서만 영향을 끼치기 때문에 D3 D2 D1은 원래 카운터를 써도 되지만 D0은 스위치가 변수로 작용하기 때문에 추가할 필요가 있다. 다시 표를 그려보자면 위의 표처럼 나온다.

4-7. True Table

< MOD13 COUNTER 진리표 >

	Q 3	Q 2	Q 1	Q 0	Q 3+	Q 2+	Q 1+	Q 0+
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	1	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	0	1	0	0
13	X	X	X	X	X	X	X	X
14	X	X	X	X	X	X	X	X
15	X	X	X	X	X	X	X	X
16	X	X	X	X	X	X	X	X

< 최종 LOGICWORKS DESIGN 진리표 >

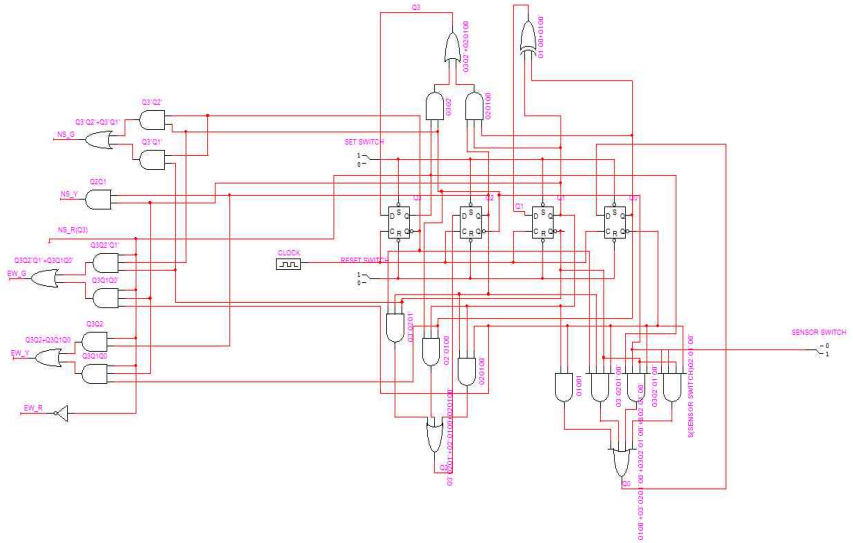
switch	Q	Q+	NS			EW		
	3210	3210	G	Y	R	G	Y	R
0	0000	0000	1	0	0	0	0	1
0	0001	0010	1	0	0	0	0	1
0	0010	0011	1	0	0	0	0	1
0	0011	0100	1	0	0	0	0	1
0	0100	0101	1	0	0	0	0	1
0	0101	0110	1	0	0	0	0	1
0	0110	0111	0	1	0	0	0	1
0	0111	1000	0	1	0	0	0	1
0	1000	1001	0	0	1	1	0	0
0	1001	1010	0	0	1	1	0	0
0	1010	1011	0	0	1	1	0	0
0	1011	1100	0	0	1	0	1	0
0	1100	0000	0	0	1	0	1	0
0	XXXX	XXXX	X	X	X	X	X	X
0	XXXX	XXXX	X	X	X	X	X	X
0	XXXX	XXXX	X	X	X	X	X	X
1	0000	0001	1	0	0	0	0	1
1	0001	0010	1	0	0	0	0	1
1	0010	0011	1	0	0	0	0	1
1	0011	0100	1	0	0	0	0	1
1	0100	0101	1	0	0	0	0	1
1	0101	0110	1	0	0	0	0	1
1	0110	0111	0	1	0	0	0	1
1	0111	1000	0	1	0	0	0	1
1	1000	1001	0	0	1	1	0	0
1	1001	1010	0	0	1	1	0	0
1	1010	1011	0	0	1	1	0	0
1	1011	1100	0	0	1	0	1	0
1	1100	0000	0	0	1	0	1	0
1	XXXX	XXXX	X	X	X	X	X	X
1	XXXX	XXXX	X	X	X	X	X	X
1	XXXX	XXXX	X	X	X	X	X	X

//위의 최종 logidcworkds design의 진리표에서의 초록색 표시는 무시하여도 된다,>

위의 카운터와 스위치라는 변수를 확인해서 하면 입력은 switch 한 개(카운터는 꾸준히 변하가기에 입력에서 배제함) 출력은 주도로 3개 보조도로 3개고,

그 출력 상태는 counter에 따라 계속 바뀌므로 그에 따른 표를 작성해보았다. 그러면 여기서 알다시피 siwitch가 0일 때 q=0000 일대 q+=0000이고 siwtch=1일 때 q=0000 일 때 q+=0001로 간다는 걸 알 수 있다. 물론 13-15 까지는 무관항으로 처리하였다. 그에 따라서 표를 만들어보았다.

그리고 현재상태에 따른 NS(주도로) G,Y,R 과 EW(보조도로) G,Y,R을 카노맵을 각각 그려서 표시를 해주었다.

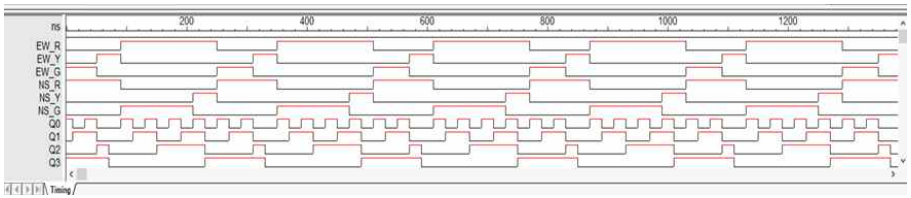


: 위의 표에서 Switch가 0일 때와 1일 때의 차이점은 Q+ 0000과 Q+0001의 차이고,

그에 따른 NS와 앞에서부터 순서대로 GYR, EW 앞에도 GYR이다.

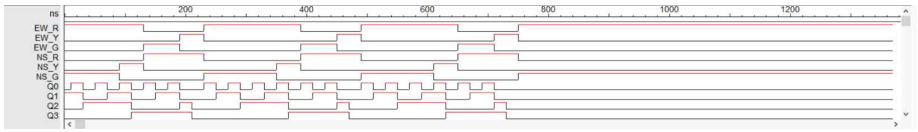
4-8. Simulation

아래의 그래프를 볼 수 있듯이 딜레이가 10ns씩 되는 것을 볼 수 있지만 신호등이 켜지긴 하였다. 모든 LOGICWORKDS GATE 딜레이를 0NS로 줬지만 딜레이가 생겼다. 아마 12초일때와 0초일때의 카노맵에서 진리표의 상태를 잘못 생각한 것으로 예상되었다.



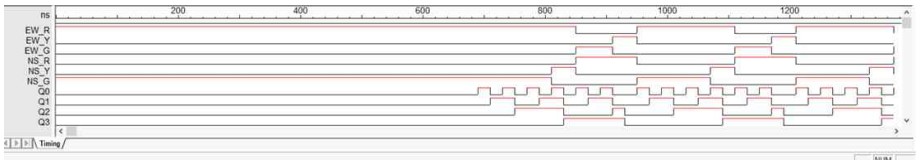
: **Switch = 1, 보조도로에 차가 있는 경우일 때이다.**

: 상태표처럼 0000이면 주도로 Green과 보조도로가 Red가 되어야 되지만, 실수로 인해 10ns씩 밀려서 1~6일 때 100 001, 7~8일 때 010 001, 9~11일 때 111 001 100, 12일 때 0까지 불이 켜지는 것을 확인할 수 있다.



Switch = 1, 보조도로에 차가 있는 경우일 때이다.

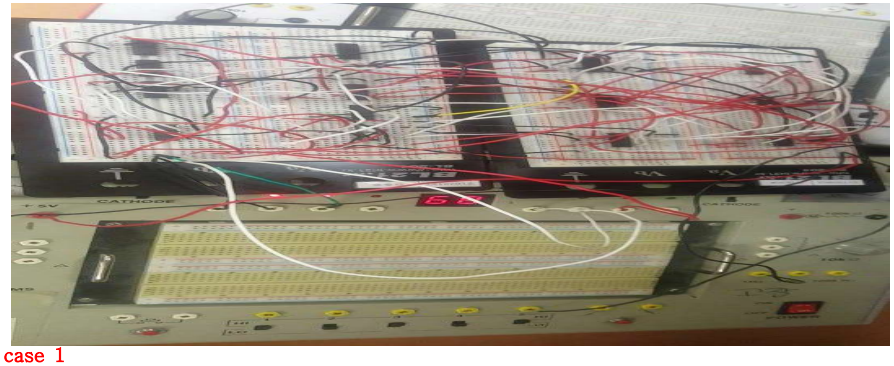
: switch가 1인 상태에서 0으로 바뀔 때 (센서가 인식하였음) 바로 100 001로 상태가 바뀌지 않고 13카운터를 모두 돈 다음, 다시 0000에서 switch가 1인지 0인지 확인하고 0이면 계속 100 001을 돌린다.



Switch = 0, 보조도로에 차가 없는 경우일 때이다.

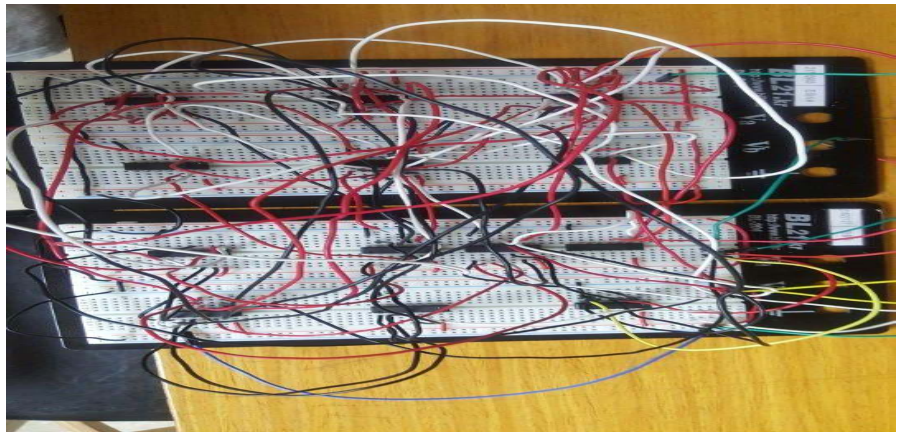
: switch가 0인 상태에서 1로 바뀔 때 바로 카운터가 0000부터 돌기 시작하며, 신호등처럼 다시 switch가 0이 되기 전까지 정상적인 신호등을 유지하고 있다.

4-9. Bread Board



case 1

위의 사진처럼 브래드보드를 만들 때 필요한 칩은 (번호랑 개수) 다 체크해서 아래의 표로 간단하게 보여준다음 완성사진을 보여준다. 위의사진은 and gate를 썼을 때 (안쓰는 pin에 1을 넣어줬었다) 하지만 결과는 다른 주도로 G Y R과 보조도로 G R은 정상적으로 작동하였으나 보조도로 Y가 계속 켜져있는 현상이 발생하였다.



case2 :

이점에 대해서는 다시 선이 복잡하게 연결되었던 점과 칩의 순서가 뒤죽박죽이었던 점을 감안하여서 다시 d, d, 2a, 2a, 3a, 3a, 2xor, 2or,2or,2or 4a 4a로 순서를 다시 정하고 하였으나 이번엔 불이 하나만 계속 켜져있고 다른 건 안되는 걸 봐서 무슨 오류가 있는지를 파악하고 다시 카노맵을 확인해 보았다. 또한 4and에서의 안쓰는 핀에 1을 꼽기보다는 그냥 아예 안쓰도록 하였고, 이 2번째 경우도 실패하자 아예 nor gate나 4and 게이트를 안쓰고 다 2and로 대체하였으면 칩의 가열된 것을 다 갈아서 다시 회로를 짜보았다.

아래는 Bread Board에서 필요한 IC칩들을 간추린 표이다.

<Case1 Bread Board>

GATE / 개수	사용할 수 있는 개수	수량
D FlipFlop	4번	2개
2input gate	7번	2개
3input gate	8번	2개
2XOR	1번	1개
2OR	10번	3개
4input gate	3번	2개

< Case2 Bread Board>

GATE / 개수	사용할 수 있는 개수	수량
D FlipFlop	4번	2개
2input gate	8번	2개
3input gate	8번	2개
2OR	11번	3개
4input gate	3번	2개

4-10. Verilog code

Case:1 (IF문절)

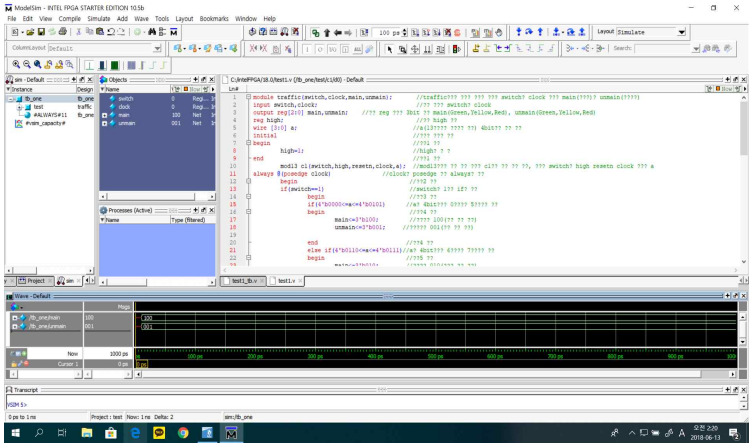
```
module traffic(switch,clock,main,unmain);//traffic이라는 이름의 모듈에 인자는 switch와 clock 그리고
        main(주도로)와 unmain(보조도로)
input switch,clock;                                    //입력 인자는 switch와 clock
output reg[2:0] main,unmain;    //출력 reg 인자는 3bit 짜리 main(Green,Yellow,Red),
unmain(Green,Yellow,Red)
reg high;                                              //변수 high 선언
wire [3:0] a;                                          //a(13카운터를 사용하기 위함) 4bit짜리 변수 선언
initial                                                //시그널 초기화 정의
begin                                                  //괄호1 시작
        high=1;                                        //high는 참 값
end                                                    //괄호1 닫기
        mod13 c1(switch,high,resetn,clock,a);  //mod13이라고 하는 모듈 이름을 c1이라 정의 하여
사용, 인자는 switch와 high resetn clock 그리고 a
always @(posedge clock)                                //clock이 posedge 일때 always문 시작
        begin                                          //괄호2 시작
        if(switch==1)                                  //switch가 1일때 if문 시작
                begin                                  //괄호3 시작
                if(4'b0000<=a<=4'b0101)        //a가 4bit이므로 0이상이고 5이하일때 시작
                begin                                  //괄호4 시작
                        main<=3'b100;          //주도로에 100(그린 신호 적용)
                        unmain<=3'b001;        //보조도로에 001(레드 신호 적용)

                end                                    //괄호4 닫기
                else if(4'b0110<=a<=4'b0111)//a가 4bit이므로 6이상이고 7이하일때 시작
                begin                                  //괄호5 시작
                        main<=3'b010;          //주도로에 010(옐로우 신호 적용)
                        unmain<=3'b001;        //보조도로에 001(레드 신호 적용)
                end                                    //괄호5 닫기
                else if(4'b1000<=a<=4'b1010)//a가 4bit이므로 8이상이고 10이하일때 시작
                begin                                  //괄호6 시작
                        main<=3'b001;          //주도로에 001(레드 신호 적용)
                        unmain<=3'b100;        //보조도로에 100(그린 신호 적용)
                end                                    //괄호6 닫기
                else                                   //a가 4bit이므로 11이상이고 12이하일때 시작
                begin                                  //괄호7 시작
                        main<=3'b001;          //주도로에 001(레드 신호 적용)
                        unmain<=3'b010;        //보조도로에 010(옐로우 신호 적용)
```

```
        end                             //괄호7 닫기
    end                                 //괄호3 닫기,switch가 1일때 if문 닫기
        else                            //switch가 0일때 시작
        begin                           //괄호8 시작
            main<=3'b100;               //주도로에 100(그린 신호 적용)
            unmain<=3'b001;             //보조도로에 001(레드 신호 적용)
        end                             //괄호8 닫기
end                                     //괄호2 닫기
endmodule                               //모듈 종료

module mod13(switch,setn,resetn,clock,q);   //mod13이라는 이름의 모듈에 인자는 switch와 setn
그리고 resetn과 clock 과 q
input switch,setn,resetn,clock;             //입력 인자는 switch와 setn(무조건 출력 q를 1로 만듬),
resetn(무조건 출력 q를 0으로 만듬) 그리고 clock
output wire [3:0] q;                        //카운터 q가 13까지 세어야 하므로 4bit짜리 output
reg high;                                   //high 라는 변수 선언
initial                                     //시그널 초기화 정의
begin                                       //괄호1 시작
    high=1;                                 //high에 참 값 정의
end                                         //괄호1 닫기
    flipflop_d d3((q[3]&~q[2])||(q[2]&q[1]&q[0]),high,resetn,clock,q[3]); //flipflop_d이란
모듈 d3에 카운터를 설계한 것처럼 순서대로 d3에는 카노맵에서 추출한 값을 집어넣고, setn에서는
high를 출력값 q엔 q[3]를 넣어준다.
    flipflop_d
d2((~q[3]&q[2]&~q[1])||(~q[2]&q[1]&q[0])||(q[2]&q[1]&~q[0]),high,resetn,clock,q[2]);
//flipflop_d이란 모듈 d2에 카운터를 설계한 것처럼 순서대로 d2에는 카노맵에서 추출한 값을
집어넣고, setn에서는 high를 출력값 q엔 q[2]를 넣어준다.
    flipflop_d d1((~q[1]&q[0])||(q[1]&~q[0]),high,resetn,clock,q[1]); //flipflop_d이란 모듈
d1에 카운터를 설계한 것처럼 순서대로 d1에는 카노맵에서 추출한 값을 집어넣고, setn에서는 high를
출력값 q엔 q[1]를 넣어준다.
    flipflop_d
d0(((q[1]&~q[0])||(~q[3]&q[2]&~q[1]&~q[0])||(q[3]&~q[2]&~q[1]&~q[0])||(switch&~q[2]&~
q[2]&~q[0])),high,resetn,clock,q[0]); //flipflop_d이란 모듈 d0에 카운터를 설계한 것처럼 순서대로
d0에는 카노맵에서 추출한 값을 집어넣고, setn에서는 high를 출력값 q엔 q[0]를 넣어준다.
endmodule //모듈 종료

module flipflop_d(d,setn,resetn,clock,q); //flipflop_d 이라는 이름의 모듈에 인자는 d,setn,resetn
그리고 clock 과 q
input d,setn,resetn,clock;                  //입력 인자는 d(입력)과 setn(무조건 출력을 1로 만듬)
그리고 resetn(무조건 출력을 0로 만듬) 그리고 clock
output reg q;                               //q를 출력
always @(posedge clock)                     //clock이 1로 posedge가 될때에만 실행
begin                                       //괄호1 시작
    if(resetn==0 && setn==1)                //reseten이 0이고 setn이 1일 때 시작
```



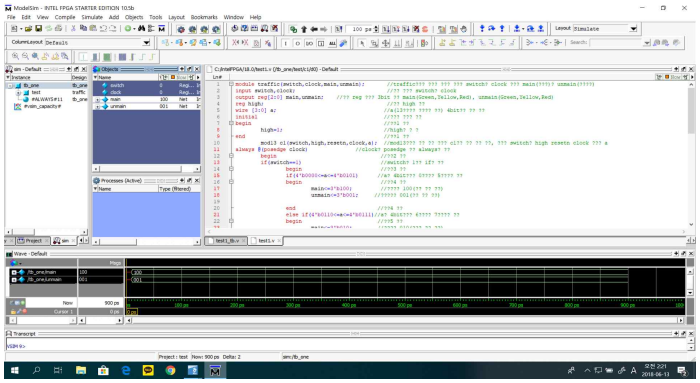
```
        q<=0;                       //q는 0을 출력
    else if(resetn==1 && setn==0)   //reseten이 1 이고 setn이 0일 때 시작
        q<=1;                       //q는 1을 출력
    else if(resetn==1 && setn==1)   //reseten이 1 이고 setn이 1일 때 시작
        q<=d;                       //q는 d의 값을 출력
end                                 //괄호1 닫기
endmodule                           //모듈 종료
`

`timescale 10ps/1ps                 //10ps의 시간단위이고 1ps의 오차범위
module tb_one;                      //tb_one으로 모듈 이름
    reg switch,clock;               //switch와 clock reg 선언
    wire [2:0] main,unmain;         //3bit짜리 main(주도로) unmain(보조도로) wire 선언
    traffic test(switch,clock,main,unmain);   //traffic이란 모듈을 test 인자는 traffic 그대로
가져옴
    initial                         //시그널 초기화 사용
    begin                           //괄호1 시작
        clock<=0;                   //클락에 0 초기화
        switch<=0;                  //switch에 0 초기화
    end                             //괄호 1 닫기
    always                          //always 항상 실행
#1 clock <=~clock;                  //clock은 #1마다 0,1,0,1.... 바뀜
endmodule                           //모듈 종료
```

Case:1 (if문)



Switch == 0

위의 그래프에 보다시피 100, 001이 꾸준히 유지되는 것을 볼 수 있다. 하지만 #3 switch<=1 문을 넣었을 때에는 카운터는 돌아가지만 신호등은 작용을 하지 않았다.



Switch == 1

switch가 1일 때는 else 구문인 nw=100과 ew=001구문만 작용을 하였으며 처음 카운터가 0000일 때의 100과 001만 작용하는 것으로 보아 [3:0] a라고 선언한 카운터 변수가 잘 실행이 되지 않는다고 생각을 하였다.

<실패 이유>

개인적인 생각

위의 두 결과사진처럼 switch가 0인 경우는 정상적으로 작용하고 switch가 1인 경우일때에도 if와 else if, else if문을 제외하고 else문은 정상적으로 작용을 하였지만, 혹은 젤 처음 if문인 nw 100과 ew 001은 작용을하였지만 카운터가 돌아가지는 않았따. 이 두 경우를 제외하고는 안되는 것을 봐서 test bench에서 a라는 카운터 변수가 정확히 bit를 인지하지 못하는 것으로 파악되었다. 그에 따라서 4'b0000 bit단위로 표기했던 방법을 10진수 숫자로 표현하기도 하고 case문으로도 해 보았지만 무슨 이유때문인지 3가지 경우에서는 되지 않았다. 우리는 그 경우를 wire과 reg를 잘 사용하지 못한 점에 있다고 생각을 하였고 wire문과 reg의 개념을 다시 살펴보았다.

Case:2 (assign 할당)

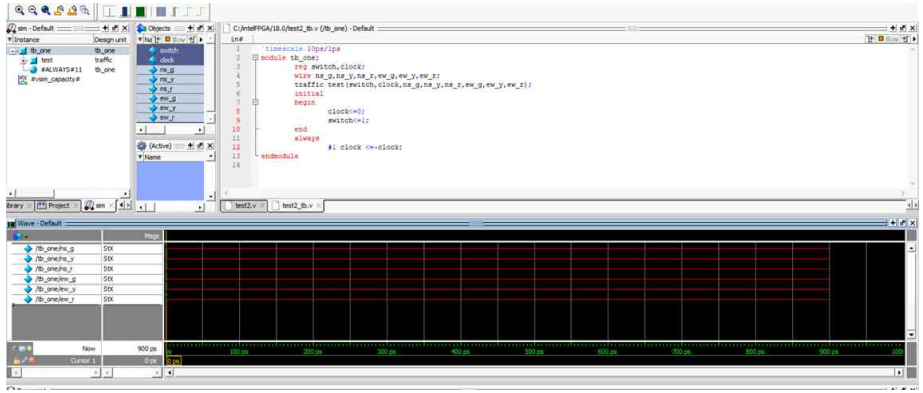
```
module traffic(switch,clock,ns_g,ns_y,ns_r,ew_g,ew_y,ew_r);       //traffic이라는 이름의 모듈에
        인자는 switch와 clock 그리고 ns_g,ns_y,ns_r,ew_g,ew_y,ew_r
input switch,clock;                                               //입력 인자는 switch와 clock
output reg ns_g,ns_y,ns_r,ew_g,ew_y,ew_r;                         //출력 reg 인자는
        ns_g,ns_y,ns_r,ew_g,ew_y,ew_r
reg high;                                                         //변수 high 선언
reg low;                                                          //변수 low 선언
wire [3:0] a;                                                     //a(13카운터를 사용하기 위함)
        4bit짜리 변수 선언
initial                                                  //시그널 초기화 정의
begin                                                    //괄호1 시작
    high=1;                                              //high는 참 값
    low=0;                                               //low는 거짓 값
end                                                      //괄호1 닫기
    mod13 c1(switch,high,resetn,clock,a);                //mod13이라고 하는 모듈 이름을 c1이라
        정의 하여 사용, 인자는 switch와 high resetn clock 그리고 a
always @(*)                                              //항상 실행되는 always문
begin                                                    //괄호2 시작
if(switch==1)                                            //switch가 1일때 시작
begin                                                    //괄호3 시작
assign ns_g=((~a[3]&~a[2])||(~a[3]&~a[1]));              //진리표에서 구한 값을 ns_g에 할당
assign ns_y=(a[2]&a[1]);                                          //진리표에서 구한 값을 ns_y에
        할당
assign ns_r=a[3];                                        //진리표에서 구한 값을 ns_r에 할당
assign ew_g=((a[3]&~a[2]&~a[1])||(a[3]&a[1]&~a[0]));     //진리표에서 구한 값을 ns_g에
        할당
assign ew_y=((a[3]&a[2])||(a[3]&a[1]&a[0]));             //진리표에서 구한 값을 ns_y에 할당
assign ew_r=~a[3];                                       //진리표에서 구한 값을 ns_r에 할당
end                                                      //괄호3 닫기
else                                                     //switch가 0일때
begin                                                    //괄호4 시작
assign ns_g=high;                                        //1을 ns_g에 할당
assign ns_y=low;                                         //0을 ns_g에 할당
assign ns_r=low;                                         //0을 ns_g에 할당
assign ew_g=low;                                         //0을 ns_g에 할당
```



```
assign ew_y=low;                                         //0을 ns_g에 할당
assign ew_r=high;                                        //1을 ns_g에 할당
end                                                      //괄호4 닫기
end                                                      //괄호2 닫기
endmodule                                                //모듈 종료

module mod13(switch,setn,resetn,clock,q);                //mod13이라는 이름의 모듈에 인자는
        switch와 setn 그리고 resetn과 clock 과 q
input switch,setn,resetn,clock;                          //입력 인자는 switch와 setn, resetn
        그리고 clock
output wire [3:0] q;                                     //4bit짜리 13카운터로 정의
reg high;                                                //high 변수 선언
initial                                                  //initial 초기화 정의
begin                                                    //괄호5 시작
    high=1;                                              //high에 참값 저장
end                                                      //괄호5 닫기
    flipflop_d d3(switch,(q[3]&~q[2])||(q[2]&q[1]&q[0]),high,resetn,clock,q[3]); //flipflop_d이란
        모듈 d3에 카운터를 설계한 것처럼 순서대로 d3에는 카노맵에서 추출한 값을 집어넣고,
        setn에서는 high를 출력값 q엔 q[3]를 넣어준다.
    flipflop_d
        d2(switch,(~q[3]&q[2]&~q[1])||(~q[2]&q[1]&q[0])||(q[2]&q[1]&~q[0]),high,resetn,cl
        ock,q[2]); //flipflop_d이란 모듈 d2에 카운터를 설계한 것처럼 순서대로 d2에는 카노맵에서
        추출한 값을 집어넣고, setn에서는 high를 출력값 q엔 q[2]를 넣어준다.
    flipflop_d d1(switch,(~q[1]&q[0])||(q[1]&~q[0]),high,resetn,clock,q[1]); //flipflop_d이란 모듈
        d1에 카운터를 설계한 것처럼 순서대로 d1에는 카노맵에서 추출한 값을 집어넣고, setn에서는
        high를 출력값 q엔 q[1]를 넣어준다.
    flipflop_d
        d0(switch,((q[1]&~q[0])||(~q[3]&q[2]&~q[1]&~q[0])||(q[3]&~q[2]&~q[1]&~q[0])||(
        switch&~q[2]&~q[2]&~q[0])),high,resetn,clock,q[0]); //flipflop_d이란 모듈 d0에 카운터를
        설계한 것처럼 순서대로 d0에는 카노맵에서 추출한 값을 집어넣고, setn에서는 high를 출력값
        q엔 q[0]를 넣어준다.
endmodule //모듈 종료

module flipflop_d(switch,d,setn,resetn,clock,q);  //flipflop_d 이라는 이름의 모듈에 인자는
        switch,d,setn,resetn 그리고 clock 과 q
input switch,d,setn,resetn,clock; // 입력 인자는 switch setn resetn clock
output reg q;                    /출력 인자는 reg 형식의 q
always @(posedge clock)          //posedge clock일때만 항상문 시작
begin             //괄호6 시작
```

```
    if(resetn==0 && setn==1)    //resetn이 0이고 setn이 1일 때 시작
        q<=0;                   //출력값 0
    else if(resetn==1 && setn==0)       //resetn이 1이고 setn이 0일 때 시작
        q<=1;                   //출력값 1
    else if(resetn==1 && setn==1)       //resetn이 1 이고 setn이 1일 때
        q<=d;                   //출력값 d
end                             //괄호6 닫기
endmodule                       //모듈 끝


`timescale 10ps/1ps     //10ps의 사간단위이고 1ps의 오차범위
module tb_one;          //tb_one 모듈 생성
        reg switch,clock;       //reg형의 switch 와 clcok
        wire ns_g,ns_y,ns_r,ew_g,ew_y,ew_r;     //wire형의 ns_g,ns_y,ns_r,ew_g,,ew_y,ew_r 출력
        6개
        traffic test(switch,clock,ns_g,ns_y,ns_r,ew_g,ew_y,ew_r); //traffic이란 모듈을 test 인자는
        traffic 그대로 가져옴
        initial   //시그널 초기화 정의
        begin   //괄호7 시작
                clock<=0;       //clock에 0 시작
                switch<=0;      //switch 0 시작
        end                     //괄호7 닫기
        always                  //계속 유지되는 항상문
                #1 clock <=~clock;      //#1마다 clock은 ~clock
endmodule                       //모듈 종료
```

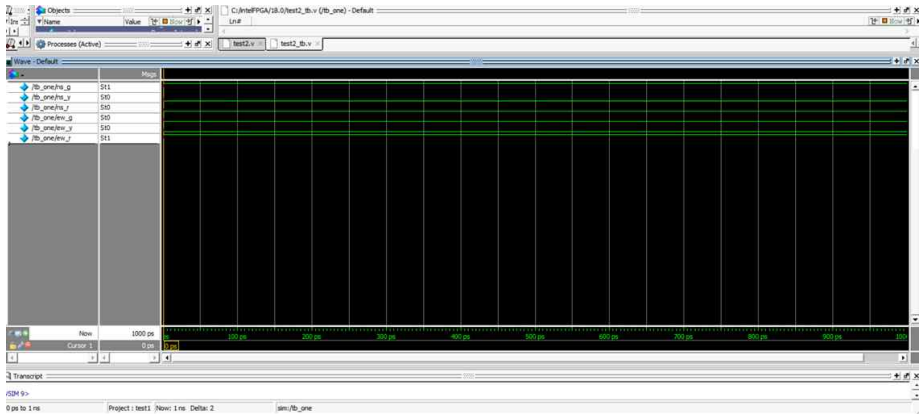

Case:2 (assign 할당)



Switch == 1

6개의 출력 인자들이 출력이 되지 않고 있었다.

stx는 상태가 표시되지 않는 값으로 안되는 이유를 알지 못했다.



Switch == 0

하지만 스위치를 껐을 때에는 100과 001이 유지되는 걸로 봐서

스위치가 꺼졌을 때에는 잘 회로가 작동하였다.

Case:3 (mod-13)

```
module mod13(setn,resetn,clock,q); //mod13이름의 인자는 setn과 resetn과 clock과 출력 q
input setn,resetn,clock;          //입력 인자 setn과 resetn 그리고 clock
output wire [3:0] q;              //출력 인자 4bit짜리 q 출력
reg high;                         //reg high 변수 정의
initial                           //시그널 초기화 정의
begin                             //괄호1 시작
   high=1;                        //high에 참값 넣기
end              //괄호 1 닫기
   flipflop_d d3((q[3]&~q[2])||(q[2]&q[1]&q[0]),high,resetn,clock,q[3]);          //flipflop_d이란
        모듈 d3에 카운터를 설계한 것처럼 순서대로 d3에는 카노맵에서 추출한 값을 집어넣고,
        setn에서는 high를 출력값 q엔 q[3]를 넣어준다.
flipflop_d d2((~q[3]&q[2]&~q[1])||(~q[2]&q[1]&q[0])||(q[2]&q[1]&~q[0]),high,resetn,clock,q
2]);      //flipflop_d이란 모듈 d2에 카운터를 설계한 것처럼 순서대로 d2에는 카노맵에서 추출한 값을
        집어넣고, setn에서는 high를 출력값 q엔 q[2]를 넣어준다.
   flipflop_d d1((~q[1]&q[0])||(q[1]&~q[0]),high,resetn,clock,q[1]);  //flipflop_d이란 모듈 d1에
        카운터를 설계한 것처럼 순서대로 d1에는 카노맵에서 추출한 값을 집어넣고, setn에서는
        high를 출력값 q엔 q[1]를 넣어준다.
   flipflop_d d0((~q[3]&~q[0])||(q[3]&~q[2]&~q[0]),high,resetn,clock,q[0]);   //flipflop_d이란
        모듈 d0에 카운터를 설계한 것처럼 순서대로 d0에는 카노맵에서 추출한 값을 집어넣고,
        setn에서는 high를 출력값 q엔 q[0]를 넣어준다.
endmodule //모듈 종료

module flipflop_d(d,setn,resetn,clock,q); //flipflop_d 이라는 이름의 모듈에 인자는 d,setn,resetn
        그리고 clock 과 q
input d,setn,resetn,clock;        //입력 인자는 d,setn,resetn,clock;
output reg q;                     //출력 인자는 reg형의 q
always @(posedge clock)           //posedge의 clock일 때 항상문
begin                             //괄호2 시작
   if(resetn==0 && setn==1)       //resetn이 0이고 setn이 1일 때 시작
      q<=0;                       //출력에 0넣기
   else if(resetn==1 && setn==0)       //resetn이 1이고 setn이 0일때 시작
      q<=1;                       //출력에 1넣기
   else if(resetn==1 && setn==1)       //resetn이 1이고 setn이 1일 때 시작
      q<=d;                       //출력에 d넣기
end                               //괄호2 닫기
```



```
endmodule                         //모듈 종료
```

Q3, Q2, Q1, Q0가 msb부터 lsb 까지는
0000부터 1100까지 되었다.

제 5장

Ⅴ.결론

1. Logicworks와 Simulation은 10ns delay가 있지만 나름대로 열심히 하여 작동이 되었다. 그 이유는 아마도 스위치가 0일때와 스위치가 1일 때 0000과 00001로 가는 그 차이에 있어서 clock 만큼 생각해보면 카운터를 작동 안하게 하는 것이 아니라 카운터를 0000을 유지하게 하는 것이므로 스위치를 껐다 킬 때마다 10ns delay가 될 수도 있다고 생각을 하였다.

2. **Breadboard를** 6~7번 정도 꼽아보고 칩을 갈거나, 여러 가지 칩을 사용해보면서 꼽아보았고 또한 안될때마다 계속 디버그를 찾아가면서 꼽아보았다. 또한 오래 사용할수록 ic chip들이 가열이 되었고 꼽을 때마다 집중이 흐려지기도 하였으나 꾸준히 여러번을 해보았지만 완성은 잘 되지 않았다.

3. modelsim에서 verilog 코드를 할 때 assign문이나 if문 혹은 case문으로 계속 돌려 보았지만 위에서 설명한 3가지 경우에서만 오류가 자꾸 발생하였다. 특히나 begin end처럼 괄호를 잘 맞춰주는 것이 어려웠고 무엇보다도 reg wire을 구분하는 것이 매우 어려웠다.

안종욱 조원의 느낀점 : 올해 “영남대 컴퓨터공학과”에 편입학을 하였다. 학문을 더 넓히자는 이유였다. 3월부터 적응하는 기간이 있어 중간고사 등 모든 과제에 있어서 많이 뒤쳐졌다.

학교에서 편입생을 위한 프로그램이 형식상만 있을 뿐 따로 모이거나 친해지는 계기가 없어 다가가기 힘들었다. 이 것이 모든 과목 팀 프로젝트에 악영향을 줄 거라곤 생각을 못했다.

“논리회로 실험 팀 프로젝트”는 2인 1조가 되어 프로젝트를 수행하는 것인데 아는 사람도 없고 친구도 없어서 혼자 남겨졌다. 조가 없는 사람은 교수님이 별도로 조를 짜주셨다. 어느 정도 얘기해본 같은 반 동생이랑 같은 조가 되었다.

미안했다. 전적대에서도 동일계열이었지만 영상 쪽을 배웠기에 이 과목에 대한 지식이 없었기 때문이다. 도움을 주지 못할망정 민폐만 끼칠 것만 같았다. 같은 조가 되어 서로 상의를 하는 도중, 의견이 많이 엇갈렸다. 중간 중간에 위기도 있었다. 형으로서 리드를 해야되고 도움을 많이 주고 싶었지만 초반에 그게 뜻대로 되지 않았다. 형으로서 미안했다. 누구나 의견이 다를 수 있고 생각이 다를 수 있다고 생각한다.

오랜 대화를 통해 서로의 문제점을 풀었고 각자 맡은 임무를 수행했다. 서로 잘하는 부분을 담당하였고 같이 밤까지 새면서 열심히 하였다.

팀프로젝트 중반에서 후반까지 슬슬 잘 풀리기 시작하여 의견도 잘 맞았다.

과제는 Logic works는 잘 되었지만 Bread board가 잘되지 않았다는 것에 대해 많이 안타까웠다. 하지만 둘이서 열심히 노력을 했다. 어떤 조는 분명 검사를 맡았을 것이고, 맡지 않은 조도 있을 것이다.

검사를 맡지 않은 조는 포기해서가 아니라 열심히 하였는데 뜻대로 되지 않았을 뿐, 모두 고생한 프로젝트였다. 묵묵히 열심히 도와준 조태식 조원에게 고맙다고 말하고 싶다.

조태식 조원의 느낀점 : 지금까지는 항상 친한 친구와 짝을 하여서 아무 다툼 없이 무난하게 팀프로젝트를 진행하였지만, 이번에는 친한 친구와 하기보다는 본격적인 team project 였기에 모르는 사람과 한번 해보려고 조를 짜지 않고 가만히 있어서 나보다 나이가 많은 종욱이 형이랑 함께 하게 되었다.

처음에는 서로 의견도 잘 맞지 않고 하는 방법도 다르고 둘 다 프로그래밍이나 bread board를 어려워하였기에 힘들었지만, 다른 사람들의 도움을 받지 않고 우리끼리 찬찬히 해보는 과정에 있어서 많은 실패과정이 있었지만 결국 logic works가 성공했을 때의 성취감을 아직도 잊을 수가 없었다. 그 와중에 delay 10ns가 생기는 것을 발견하고 여러 가지 방안을 대안 해보았지만 찾을 수가 없어, 그 상태로 빵판을 해 보았다. 빵판에서 되게 오래 걸리긴 하였지만 8번 반복해본 결과 완전한 신호등은 완성하지 못하였고, 최고로 성공했을 때의 신호등은 다른 것은 작동하되 보조도로의 yellow가 항상 켜져 있는 신호등이었다. verilog도 다 완성은 하였으나, 코드나 카운터를 읽어 들이는 부분에서 무엇인가 잘 되지 않아서 힘들었다. 그래도 꾸준히 2주동안 알차게 혹은 열심히 시간을 투자해서 여기까지 보고서를 쓴 것 만으로도 성취감을 많이 느끼게 되었다. 보고서를 마치고 집으로 가는 중 엘리베이터를 타면서 다음 과제는 엘리베이터를 “설계하시오” 라고 나왔으면 좋겠다고 생각을 하였다.

이 과제를 마치면서 지금까지 많이 힘들었을 종욱이형이랑 무사히 과제를 마치게 되어 형한테 고마웠고, 2학기 때도 마이크로페서란 과목에서도 같이 만나서 그때는 완성을 했으면 좋겠다는 바람이 있다.

제 6장

Ⅴ. 참고문헌

http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0ahUKEwjK8fH8qtHbAhVJwbwKHWorDQ4QFgg3MAM&url=http%3A%2F%2Fcfile6.uf.tistory.com%2Fattach%2F1818D34850A49032210498&usg=AOvVaw3sVy8pKh7n_uDP1UBJVLaC