논리회로 실험

Term Project

실험제목 : 교통 신호 제어기 설계

제출일자 : 2018 - 06 - 05

소속 : 통계학과

학번 : 21410785, 21410781

이름 : 황 희, 최영용

<제목 차례>

<표 차례>

<그림 차례>

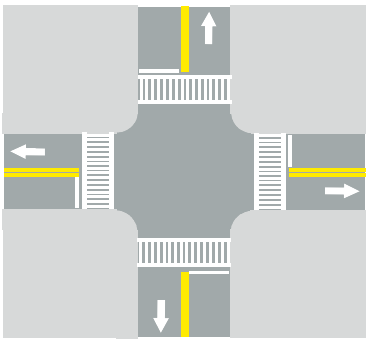
<수식 차례>

1. 서론

2.1. 실험목적 및 내용

◆ 실험목적 : 논리회로에서 배운 이론을 바탕으로 실생활에 적용 가능한 논리회로를 설계하고, 구현함으로써 논리회로에 대한 이해도를 높인다.

◆ 실험내용 : 다음과 같이 남북(NS), 동서(EW)의 교차로가 있는 교통신호 제어기 디자인
단, EW도로에는 차량이 있는 경우에만 신호가 바뀌도록 한다.
남북(NS)도로의 신호 순서
: 녹색(6초) -> 황색(2초) -> 적색
동서(EW)도로의 신호 순서
: 녹색(3초) -> 황색(2초) -> 적색



2.2. 팀원 역할

최영용: 회로설계(상태천이도 및 카노맵), verilog구현 및 주석처리
황 희 : logicworks 회로구현, 보고서작성, 브래드보드 구현

2. 본론

2.1. 문제이해

◆ 문제이해 : 차량이 있는 경우를 Binary 스위치로 두고, 스위치가 1인경우를 차량이 있는 경우, 스위치가 0인경우를 차량이 없는 경우로 생각한다. 초기값은 남북(NS)도로가 녹색신호, 동서(EW)도로가 적색신호인 경우로 두고, 스위치는 0인경우로 둔다. 만일 스위치가 0에서 1로(EW도로에 차량 도착) 바뀐다면 1로 바뀐 순간부터 남북(NS)도로는 6초간 녹색 신호를 유지하다가 2초간 황색신호, 그다음 적색신호가 점등됨과 동시에 동서(EW)도로는 초록불이 점등된다. EW도로의 녹색신호는 3초, 황색신호는 2초를 유지한 후 EW->적색, NS->녹색으로 점등된다. 스위치가 1인경우의 점등순서와 시간은 다음의 표와 같다.

표 1 NS, EW 신호 점등 순서

초(SEC)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NS														
EW														

따라서, 스위치가 1인경우, NS도로의 적색신호는 총 5초, EW도로의 적색신호는 총 8초로 생각할 수 있고, 13초마다 순환함을 알 수 있다.
만일 스위치가 1에서 0으로 바뀌는 경우에는 현재 신호가 NS -> 녹색 EW -> 적색이 점등 될 때까지 스위치 1에서의 순서대로 신호가 진행되다가 스위치에 1이 들어올 때까지 NS, EW가 각각 녹색, 적색인 상태를 유지하면 된다.

입력과 출력은 다음과 같이 지정해주었다.

표 2 INPUT

INPUT	0	EW도로에 차가 없는 경우
	1	EW도로에 차가 있는 경우

표 3 OUTPUT

OUTPUT	NS_R	NS도로 적색
	NS_Y	NS도로 황색
	NS_G	NS도로 녹색
	EW_R	EW도로 적색
	EW_Y	EW도로 황색
	EW_G	EW도로 녹색

그리고 총 순환이 13초 주기로 이루어지므로 MOD-13 counter를 만들어 각 초에 맞는 상태를 NS,EW값에 할당 해줄 것이다.
우리팀은 무어 머신으로 설계하기로 했다.

2.2. 상태천이도 - State Transition Diagram(S.T.D.)

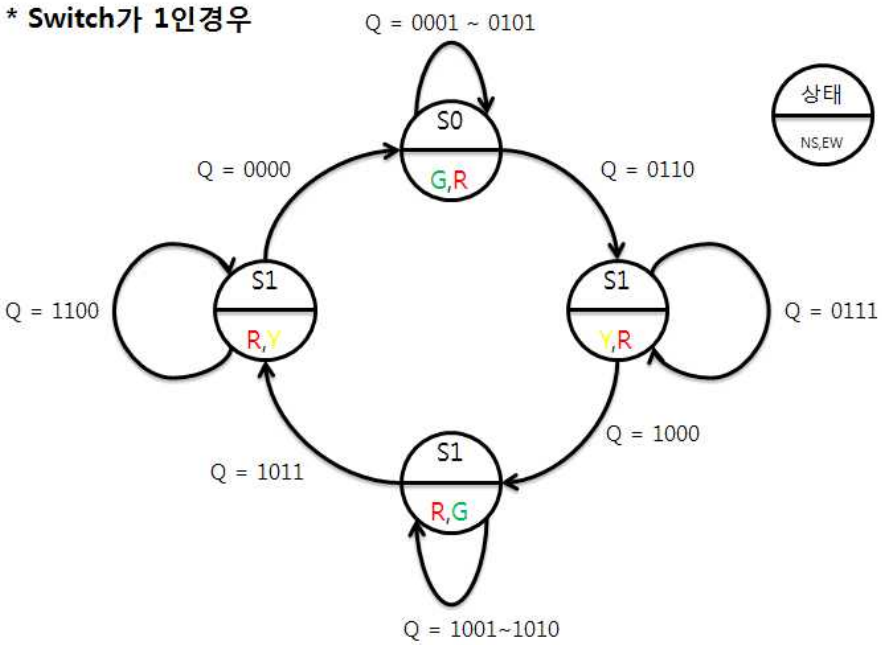


그림 1 상태천이도(State Transition Diagram)

◆ Q는 MOD-13 counter의 현재상태(PS)이다. 신호의 특성 상 시간의 흐름에 따라 상태가 변하므로 Switch의 상태는 따로 고려하기로 하였다. Switch가 0인 경우는 2.4.의 플립플롭 선택에서 다루도록 하겠다.

2.3. 상태천이표 - State Transition Table(S.T.T.)

표 4 State Transition Table

PS				NS				OUTPUT					
Q[3]	Q[2]	Q[1]	Q[0]	Q[3]	Q[2]	Q[1]	Q[0]	NS_G	NS_Y	NS_R	EW_G	EW_Y	EW_R
0	0	0	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	0	1	0	1	0	0	0	0	1
0	0	1	0	0	0	1	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	1	0	0	0	0	1
0	1	0	0	0	1	0	1	1	0	0	0	0	1
0	1	0	1	0	1	1	0	1	0	0	0	0	1
0	1	1	0	0	1	1	1	0	1	0	0	0	1
0	1	1	1	1	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	1	0	0	1	0	0	0
1	0	0	1	1	0	1	0	0	0	1	1	0	0
1	0	1	0	1	0	1	1	0	0	1	1	0	0
1	0	1	1	1	1	0	0	0	0	1	1	1	0
1	1	0	0	0	0	0	0	0	0	1	0	1	0
1	1	0	1	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

◆ 위의 표는 스위치 1에서의 MOD-13 counter 상태표와 그 상태에 따른 OUTPUT이다. OUTPUT은 NS도로의 적,녹,황색신호와 EW도로의 적,녹,황색 신호로 이루어져있다.

2.4. 플립플롭 선택

◆ 선택 플립플롭 : JK-F/F

표 5 mod-13 counter J,K F/F 할당

PS				NS				J, K							
Q[3]	Q[2]	Q[1]	Q[0]	Q[3]	Q[2]	Q[1]	Q[0]	J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	1	0	1	0	X	0	0	X	1	X	X	1
1	0	1	0	1	0	1	1	X	0	0	X	X	0	1	X
1	0	1	1	1	1	0	0	X	0	1	X	X	1	X	1
1	1	0	0	0	0	0	0	X	1	X	1	0	X	0	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

◆ 스위치가 0일때의 MOD-13 counter의 상태 해결방안 : Switch가 0일때는 EW도로에 차가 없을 때의 초기 신호 형식을 그대로 유지하면 되므로 Q[3],Q[2],Q[1],Q[0]가 4'b0000일 때부터 계속적으로 4'b0000을 유지 할 수 있게 하면 된다. JK F/F의 ClearN신호는 Negative Edge일 때 작동하는 Active Low 방식이므로 다음의 식을 만족하게 만들면 해결이 가능하다.

$$(\overline{Switch}\,\overline{Q[3]}\,\overline{Q[2]}\,\overline{Q[1]}\,\overline{Q[0]})' = 0$$
$$= Switch + Q[3] + Q[2] + Q[1] + Q[0]$$

이와같은 식을 만족하게 한다면

수식 1 Switch 0 -> Q = 4'b0000 만족식

그림 2와 같이 어떤 상태에서라도 S0에 도달하게 되면 S0을 계속 유지하도록 만들 수 있다.

따라서, Switch를 별도의 INPUT으로 넣어주지 않았고, INPUT은 오직 Q의 값으로만 하여 J, K, OUTPUT의 식을 도출하였다.

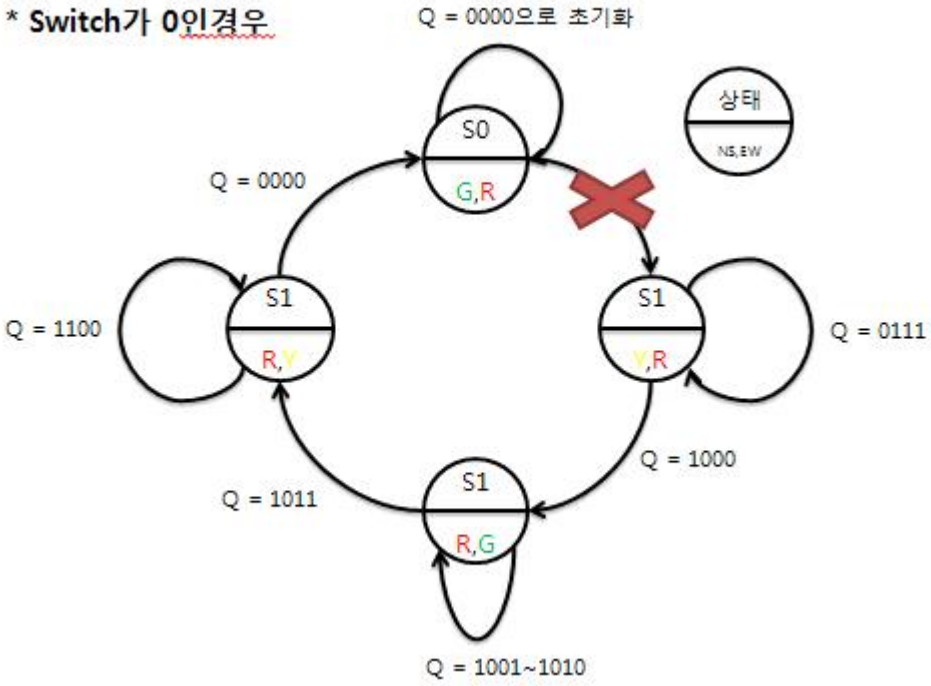


그림 2 Switch가 0인경우의 상태천이도

표 6 J_D의 카노맵

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

표 7 J_C의 카노맵

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

표 8 J_B의 카노맵

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

표 9 J_A의 카노맵

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

표 10 K_D의 카노맵

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

표 11 K_C의 카노맵

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

표 12 K_B의 카노맵

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

표 13 K_A의 카노맵

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

표 14 NS_G의 카노맵

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

표 15 NS_Y의 카노맵

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

표 16 NS_R의 카노맵

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

표 18 EW_G의 카노맵

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

표 17 EW_Y의 카노맵

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

표 19 EW_R의 카노맵

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

◆ MSB가 D, LSB가 A인 4비트 13카운터로 J,K의 입력식과 OUTPUT의 결과식을 카노맵을 이용해 나타내면 다음과 같다.

- $J_D = CBA$
- $J_C = BA$
- $J_B = A$
- $J_A = \overline{D} + \overline{C}$
- $K_D = C$
- $K_C = BA + D$
- $K_B = A$
- $K_A = 1$
- $NS\_G = \overline{D}\overline{B} + \overline{D}\overline{C}$
- $NS\_Y = CB$
- $NS\_R = D$
- $EW\_G = D\overline{C}\overline{B} + D\overline{C}\overline{A}$
- $EW\_Y = DC + DBA$
- $EW\_R = \overline{D}$

2.5. 구현 (Implementation) - Verilog

◆ Verilog 코드

▸ Traffic_Signal 모듈의 코드

```
//JK플립플롭에대한 모듈이다.
module JK(Q,j,k,setn,resetn,clock);

input clock,setn,resetn,j,k;
output reg Q;

always @(posedge clock)
begin
	if(resetn==0)
		Q<=0;
	else if (setn==0)
		Q<=1;
	else if (j==0 && k==0)
		Q<=Q;
	else if (j==0 && k==1)
		Q<=0;
	else if (j==1 && k==0)
		Q<=1;
	else if (j==1 && k==1)
		Q<=~Q;
end
endmodule

//13초 카운터에대한 모듈이다
module mod13_counter(Q,Setn, Resetn, Clock);
	input Setn, Resetn, Clock;		//Setn,Resetn,Clock을 input으로 두었다.
	output [3:0] Q;				//13초 카운터이기 때문에 4비트이다.
	reg high;				//신호 1을 할당하기위한 변수 high생성

	initial
		high = 1;			//1을 할당해준다.

	//0000->0001->0010->0011->0100->0101->0110->0111->1000->1001->1010->1011->1100->0000 을
반복하는 JK플립플롭 식이다.

	JK JKFF3(Q[3],(Q[2]&Q[1]&Q[0]),Q[2],Setn,Resetn,Clock);
	JK JKFF2(Q[2],(Q[1]&Q[0]),((Q[1]&Q[0])|Q[3]) ,Setn, Resetn, Clock);
	JK JKFF1(Q[1], Q[0],Q[0], Setn, Resetn, Clock);
	JK JKFF0(Q[0],~Q[3]|~Q[2], high, Setn, Resetn, Clock);

endmodule

//EW도로, NS도로를 구현한 코드이다
module Traffic_signal(EW,NS,Q,Switch,Setn,Resetn,Clock);
	input Switch,Setn,Resetn,Clock;		//13초카운터에 넣을 input이다.
```

```
output [3:0] Q;                                  //13초카운터에서 출력하는 값을 받기위한 output이다.
output reg EW_G,EW_Y,EW_R,NS_G,NS_Y,NS_R; //OUTPUT 각각의 변수는 0또는 1의 값을 가진다.

// NS_ = 남북 도로, EW_ = 동서 도로
// _G = 녹색, _Y = 황색, _R = 적색
// 1이면 점등, 0이면 소등인 상태이다.

//switch 0 -> EW에 차가 없다.
//switch 1 -> EW에 차가 있다. 를 표현했다.
initial
begin
  NS_G=1;            // 초기값으로 NS에 녹색신호, EW에 적색신호인 상태를 주었다.
  NS_Y=0;
  NS_R=0;
  EW_G=0;
  EW_Y=0;
  EW_R=1;
end


mod13_counter mod13(Q,Setn,Resetn,Clock); //13초 카운터를 불러온다.

always @(posedge Clock)     //clock이 posedge 일때만 작동하게 한다.
begin
//스위치가 0이고 Q가 0000일때 즉 EW에 차가 없고 EW신호가 적색으로 바뀌었을때를 의미한다
if(Switch==0&&Q==4'b0000)
            begin
              NS_G=1;
              NS_Y=0;
              NS_R=0;
              EW_G=0;
              EW_Y=0;
              EW_R=1;
            end
//그외 다른경우는 카노맵에서 도출한 식을 바탕으로 신호가 정상작동하게한다.
//NS : 녹색(6초) -> 황색(2초) -> 적색(3초) -> 적색(3초) -> 녹색 순
//EW : 적색(6초) -> 적색(2초) -> 녹색(3초) -> 황색(2초) -> 적색 순
else
            begin
            NS_G=(~Q[3]&~Q[1])|(~Q[3]&~Q[2]);
              NS_Y=Q[2]&Q[1];
              NS_R=Q[3];
              EW_G=(Q[3]&~Q[2]&~Q[1])|(Q[3]&~Q[2]&~Q[0]);
              EW_Y=(Q[3]&Q[2])|(Q[3]&Q[1]&Q[0]);
              EW_R=~Q[3];
            end

end

endmodule
```

▸ Traffic_Signal 테스트벤치 모듈의 코드

```
//교통신호의 테스트벤치 코드이다.
`timescale 10ps/1ps
module tb_Traffic_signal;
	//교통신호 코드를 테스트해주기위한 reg와 wire이다.
	reg Switch,Setn,Resetn,Clock;  //교통신호코드의 input에 들어갈 변수.
	//교통신호에 output에 들어갈 변수이다.
	wire [3:0] Q;
	wire [1:0] EW;
	wire [1:0] NS;

	//교통신호를 테스트 하기위한 구문.
	Traffic_signal test(EW,NS,Q,Switch,Setn,Resetn,Clock);

	initial
	begin
		//초기 값을 Switch->1(EW에 차가있다.)로 주었다.
		Switch<=1;
		Setn <= 1;
		Resetn <= 0;
		Clock <= 0;

		#1 Resetn <= 1;
		#40 Switch <= 0;
		#40 Switch <= 1;
	end

	always
		#1 Clock = ~Clock;

	always
	begin

//스위치가 0이고 Q가 0000일때 즉 EW에 차가 없고 EW신호가 적색으로 바뀌었을때를 의미한다

//Resetn을 0으로 계속 할당하는 이유는 EW에 차가없는데 카운터를 증가시켜주면 신호가 바뀌기 때문이다.

		//Switch가 1로 바뀌었을때에 카운터를 0000부터 시작하게해주기위한 조건문이다.
		if(Switch==0&&Q==4'b0000)
			#1 Resetn<=0;
		else
			#1 Resetn<=1;
	end


endmodule
```

◆ 실행 결과

▸ Switch가 1인 경우

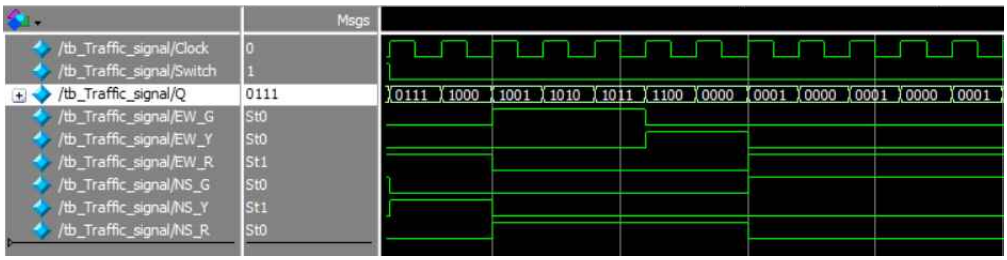


그림 3 모델심 결과 (Switch = 1) 1

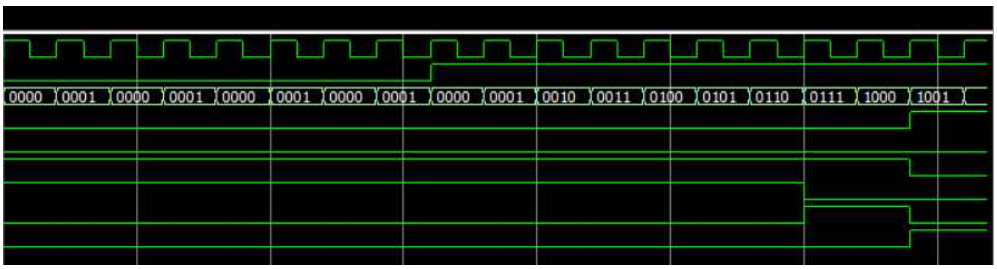


그림 4 모델심 결과 (Switch = 1) 2

▸ Switch가 0인 경우

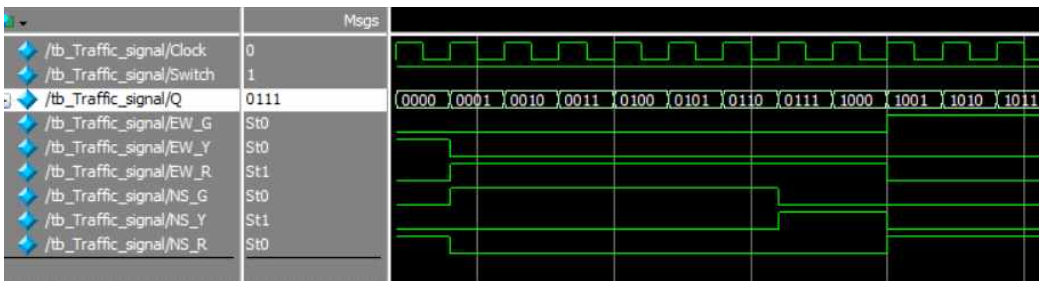


그림 5 모델심 결과 (Switch = 0) 1

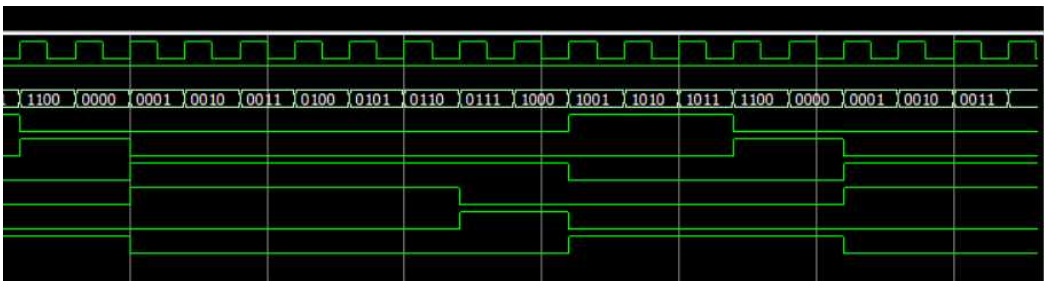


그림 6 모델심 결과 (Switch = 0) 2

2.6. 구현 (Implementation) - Logicworks

◆ 회로구현

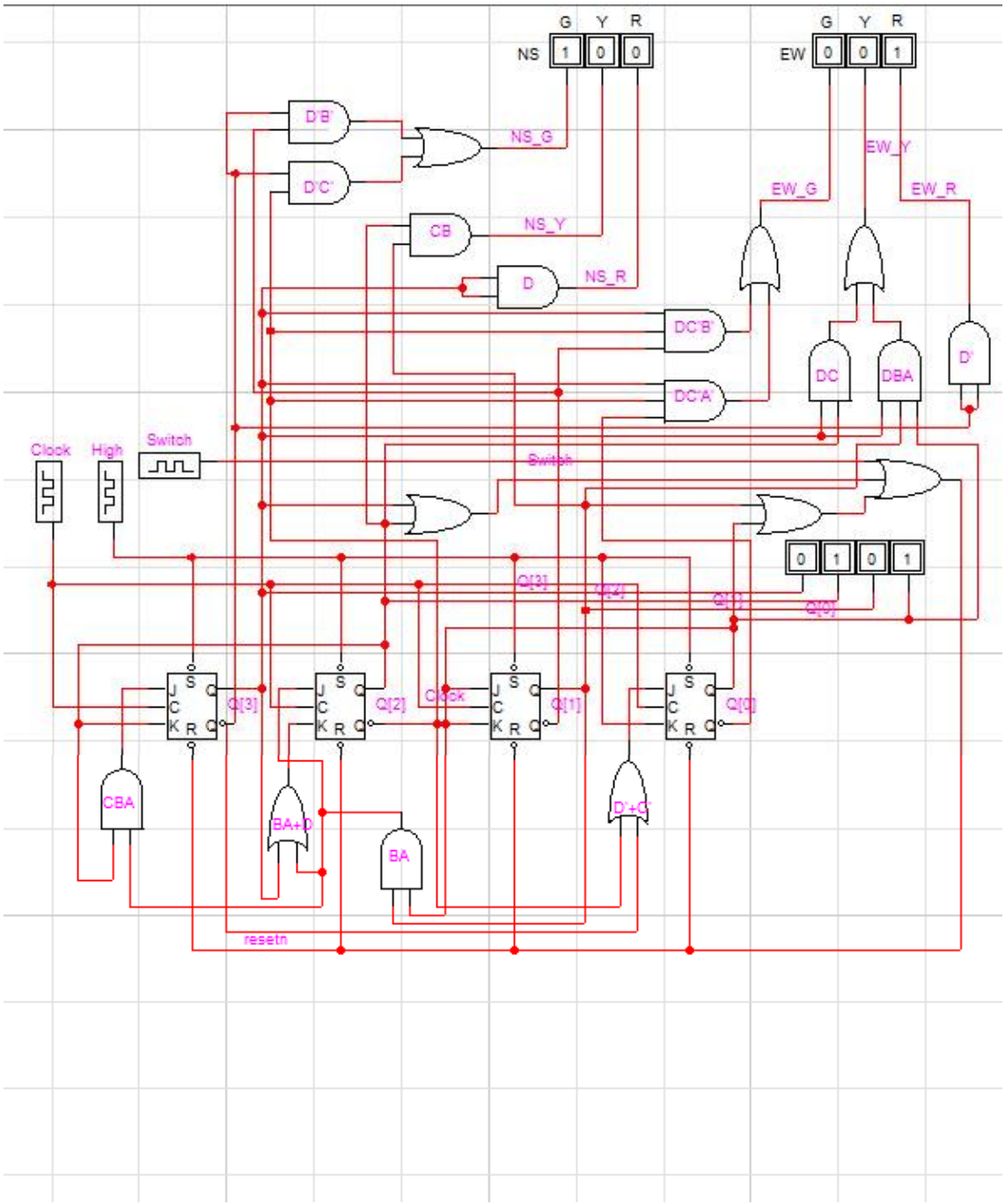


그림 7 회로 구현

◆ 결과

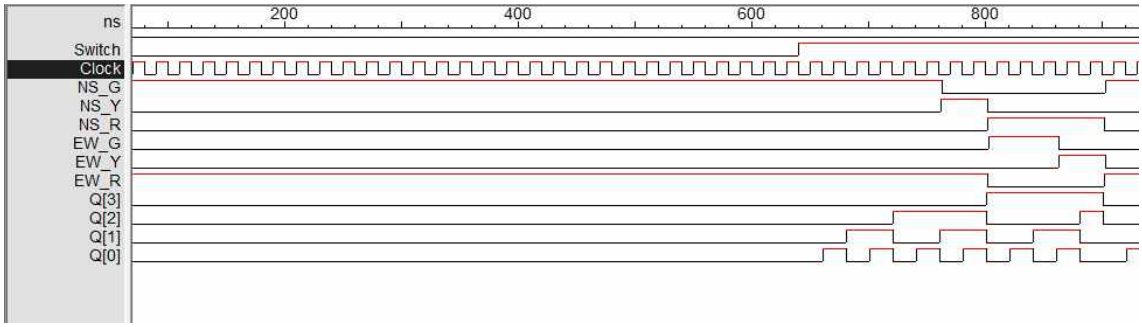


그림 8 스위치가 0에서 1로 바뀌는 경우

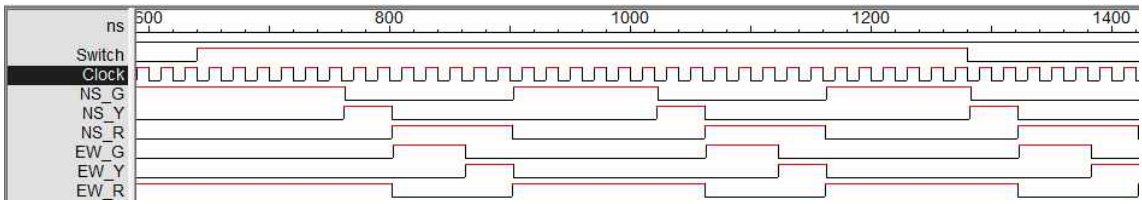


그림 9 스위치 1인경우의 신호 변화

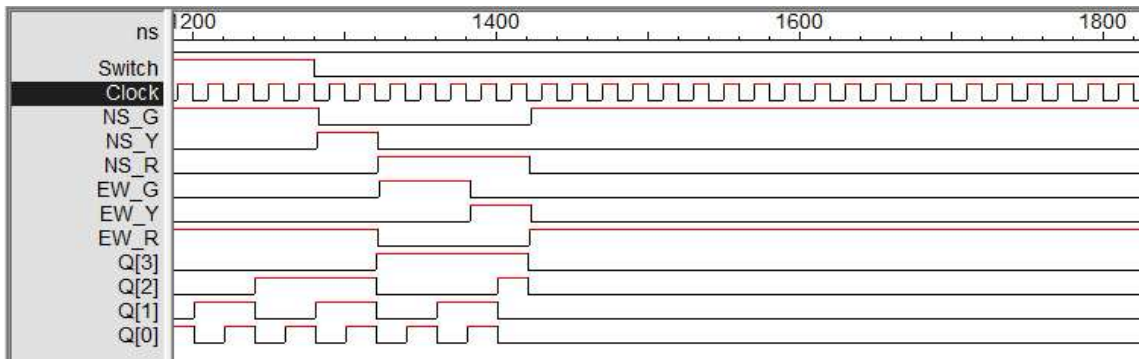


그림 10 스위치가 1에서 0으로 바뀌는 경우

3. 결론

3.1. 부족한 점

◆ 모델심으로 구현할 때에 클락이 맞지 않아서인지 딜레이 때문인지 알 수 없는 원인으로 인해 Switch를 0으로 주면 Q[0]가 0을 계속 유지하고 있어야하는데 0과 1을 주기적으로 반복하는 현상이 나타났다. 이 점에 대해서 보완하면 좋았을 것 같고, Logicworks에서 구현을 할때 게이트의 딜레이 때문에 클락이 Positive Edge에서 회로가 동작함에도 불구하고 Negative Edge에서 회로가 동작하는 것처럼 보이는 현상이 있는데 딜레이를 없앨 수 있었다면 더 좋았을 것이라는 생각을 하였다.

3.2. 느낀 점

◆ 논리회로를 배우기 전, 어떤 사물(신호등, 자판기 등)의 작동원리를 궁금해 할 때는 막연하고 추상적으로 이런 식으로 작동 하겠지, 라는 생각을 가졌었는데, 논리회로를 배운 후로는 작동원리를 구체적으로 생각해보고 설계까지 생각 해 볼수 있게 되었다. 또한, 컴퓨터의 구조와 동작을 이해하는데 있어서도 많은 도움이 되었던 것 같고 한학기 동안 논리회로와 논리회로 실험을 가르쳐 주신 교수님들께 감사의 말씀을 드립니다.

- 끝 -