컴퓨터 공학 기초 실험2 보고서

실험제목: Latch&FlipFlop

TrafficLightController

실험일자: 2020년 10월 05일 (월)

제출일자: 2020년 10월 19일 (월)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2019202052

성 명: 김 호 성

1. 제목 및 목적

A. 제목

- Latch&FlipFlop
- Traffic Light Controller

B. 목적

- Latch를 구현한다.
- Flip-Flop을 구현한다.
- traffic light controller를 구현한다.

2. 원리(배경지식)

- SR (set / Reset) Latch

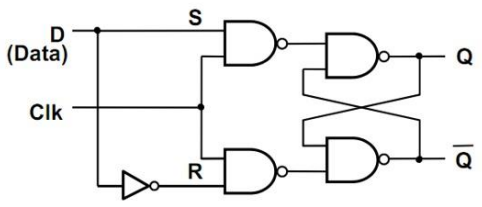
Sequential Circuit 중 가장 간단한 circuit이며, 2 개의 NOR gate로 구성되어 있다.
Input에 S, R이 있고 Output에 Q, Q bar가 있다.
S의 값을 통해 Q를 관리할 수 있고, R 값으로 reset할 수 있다.

- D Latch

SR Latch에서 S와 R이 동시에 입력될 경우 생기는 문제점과 output이 언제, 무엇으로 바뀌어야 하는지에 대해 구분을 못하는 단점을 보완한 Latch이다.
SR Latch의 입력 부분에 2개의 2 input AND gate가 됐다.
입력 값은 CLK와 D가 있고 둘 중 하나의 AND gate에 D bar 값이 들어가도록 inverter가 추가됐다.
D를 통해 다음 상태를 정할 수 있고 CLK를 통해 상태가 언제 변하는 지 조절할 수 있다.

- NAND gate 4개를 이용하여 D Latch를 구현하는 방법

SR Latch는 2개의 NOR gate 또는 NAND gate로 구현할 수 있다.
NAND gate와 NOR gate 둘 다 입력 값과 결과 값은 다르지 않으므로 회로를 설계하면 이런 모양이 나온다.



D에 inverter를 붙이는 과정은 같지만 AND

gate 대신 NAND gate를 사용함으로 써 기존의 SR NAND Latch에서 입력 값이 S bar와 R bar인 특성을 유지한다. 또한 AND gate는 트랜지스터를 6개 사용하지만 NAND gate는 4개만 사용하므로 더 효율적이기도 하다.

- D flip-flop
D Latch를 2개 이어 붙여 만든 회로다.
input에는 D와 CLK가 있고, output에는 Q와 Q bar가 있다.
첫 번째 D Latch에서 나온 Q의 값을 두 번째 D Latch의 D가 받아내고, 첫 번째 D Latch에는 LK에 inverter가 적용되어 입력된다.
D flip-flop의 종류로는 Enabled Flip-flop, Resettable Flip-flop등 종류가 다양하다.

- 실습에서 구현한 enabled D flip-flop이 아닌 다른 방법으로 구현하는 방법
실습에서는 Mux를 사용하여 D에 값을 받을지, 아니면 기존 값을 재사용 할지 선택했다.
다른 방법으로는 2 input AND gate를 사용하는 방법이 있다.
Input에 CLK와 EN을 입력해서 EN이 1이면 CLK이 정상적으로 회로의 toggle 기능을 할 수 있고, EN이 0이면 CLK 또한 0이 되어서 회로는 기존의 값을 재사용하게 된다.

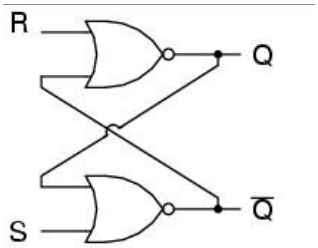
- Register
1개의 CLK을 공유하는 n개의 D Flip-flop으로 이루어진 일종의 저장장치이다.
CLK이 1개 이므로 register 내의 모든 값들은 동시에 바뀐다.
값들이 동시에 바뀐다는 것은 특정 방향으로 값들이 흐른다는 것이므로 하나의 data bus라고 할 수 있다.

- FSM
Finite state Machine의 약자로 Synchronous sequential circuit을 표현하는 방법 중 하나이다. 회로의 k개의 레지스터가 연결되어 있으면 2^k개의 특정한 상태를 도출할 수 있다. M개의 input, N개의 output, K개의 레지스터로 구성되어 있다.
FSM 내부는 2개의 combinational logic과 레지스터로 구성되어 있는데, 회로를 각각 next state logit과 output logic이라 부른다. 매 CLK edge마다 input 값을 기반으로 FSM의 다음 상태가 결정된다. FSM의 종류로는 Moore Machine과 Mealy machine이 있는데, Moore Machine은 결과 값이 현재의 상태에만 영향을 받고 Mealy machine은 현재의 상태와 input의 영향을 받는다.

3. 설계 세부사항

- SR Latch

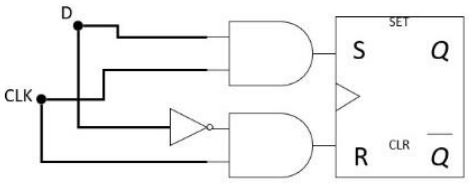
S	R	Q	Q_bar
0	0	Q_prev	Q_bar_pr
0	1	0	1
1	0	1	0
1	1	0	0



진리표와 회로 모양은 위와 같다. NOR gate는 1잉 하나라도 있을 경우 output이 0이 되므로 이를 이용하여 결과 값을 알아낼 수 있다. Input이 둘 다 0이면, 먼저 Q가 0이면 Qbar가 1인 경우와 Q가 1이면 Qbar가 0인 경우로 두 가지가 있다.

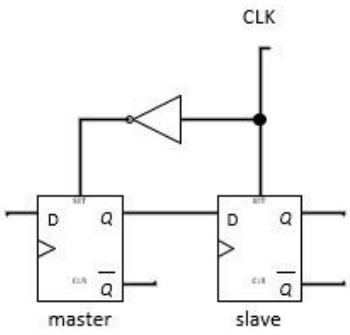
- D Latch

CLK	D	Q	Q_bar
0	X	Q_prev	Q_bar_pr
1	0	0	1
1	1	1	0



SR Latch의 input에 각각 AND gate를 추가했다. 입력 값은 CLK와 D이다. CLK가 0일 경우 입력 값은 D에 상관없이 모두 0이 되어서 결과 값 Q로 향하는 새로운 데이터를 막아버리는데, 이러한 상태를 opaque라고 한다. 결과 값은 이전 값을 유지한다. CLK가 1이면 회로의 모양에 따라 각각 알맞은 결과 값이 나온다. 이런 상태를 transparent라고 한다.

- D flip-flop

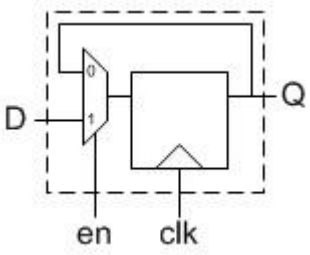


CLK	Q
Rising edge	D
Else	Q_prev

왼쪽의 D Latch를 L1이라 하고 오른쪽의 D Latch를 L2라 하자. 앞의 Latch는 master라 부르고 뒤의 Latch는 slave라 부른다. 그리고 두 래치 사이의 노드를 N1이라고 하자.

CLK = 0 일 때 master는 transparent이고 slave는 opaque이다. 그러므로 이 때는 입력 값에 상관없이 D값이 N1을 통해 흐른다. CLK = 1이면 master는 opaque이고 slave는 transparent이다. 이러면 N1에서 Q로 가는 길은 열려 있고, D로 가는 길은 막혀 있다.

- EN flip-flop

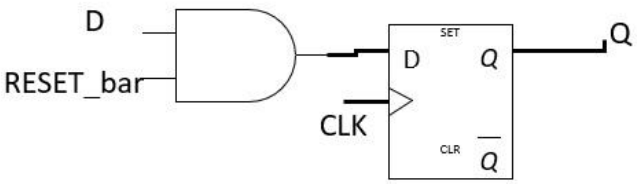


Input			Output
EN	D	CLK	Q
0	X	Rising	Q_{prev}
1	0	Rising	0
1	1	Rising	1
1	X	Falling or 1 or 0	Q_{prev}

D flip-flop의 input에 mux를 추가한 회로다.

EN이 1이면 회로는 기존의 D flip-flop처럼 작동하고 EN이 0이면 회로는 CLK를 무시하고 이전 값을 유지한다. EN flip-flop은 모든 CLK edge에서 작동하는 게 아니라 원할 때만 값을 바꿀 수 있게 조작할 수 있다.

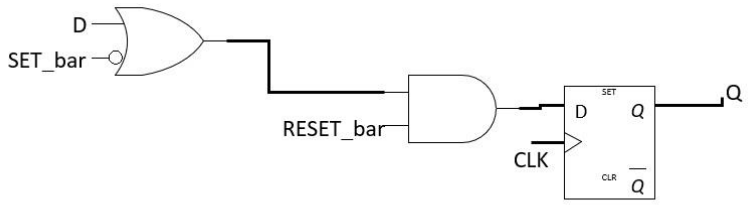
- Resettable flip-flop



Input			Output
RESET_bar	D	CLK	Q
0	X	X	0
1	0	Rising	0
1	1	Rising	1
1	X	Falling or 1 or 0	Q_{prev}

D flip-flop의 입력 값에 D에 AND gate를 추가한 회로다. AND gate에는 RESET이라 불리는 새로운 입력 값이 추가됐는데, 값이 0이면 기존의 D flip flop처럼 작동하고 값이 1이면 D를 무시하고 결과 값을 0으로 재설정한다. 이러한 성질로 인해 resettable flip-flop이라는 이름이 붙었다. 이 회로는 시스템 내의 복잡한 회로를 초기화 해야 할 때 유용하다.

- Synchronous Set/Resettable D flip-flop



Synchronous는 CLK이 rising edge일 때에만 reset을 수행한다는 뜻이다. Set이 0인 경우 invertor로 인해 1이 입력되고 D의 값에 관계없이 AND gate에 1이 입력되어 reset을 통해 정상적으로 reset을 수행할 수 있다. Set이 1일 경우 0이 입력되고 D의 값에 따라 set의 수행 여부를 따질 수 있다.

- Register

N bit register는 N개의 D flip-flop을 하나의 CLK에 연결하여 만든 저장장치이다.

- Async/sync Set/Resettable D flip-flop

Sync는 CLK이 rising edge일 경우에만 동작하고, Async는 CLK이 rising edge가 아니어도 동작한다.

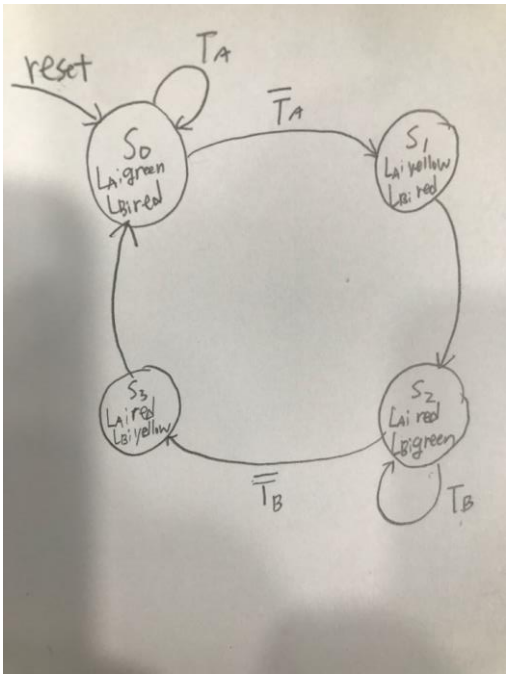
- Traffic Light Controller

주어진 그림에 따르면 총 4가지 상태가 있는데, 아래와 같다.

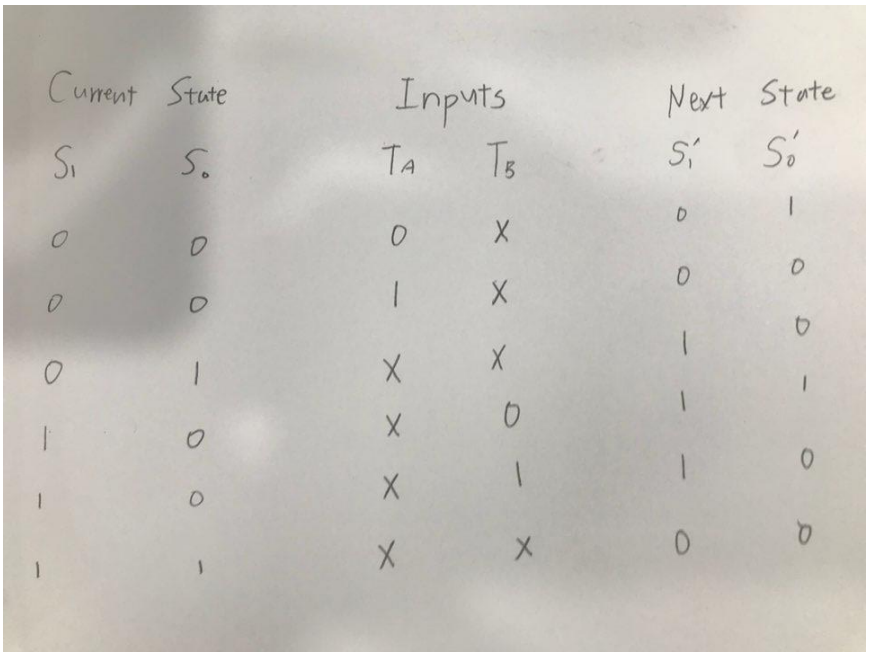
State	La	Lb
S0	green	red
S1	yellow	red
S2	red	green
S3	red	yellow

각각의 신호 Ta와 Tb는 해당하는 위치의 길에 사람이 있으면 1을, 없으면 0을 출력한다. La와 Lb는 입력 값에 따른 결과 값이다. 이 회로가 실제로 동작하기 위해서는 상태와 출력 값이 2진수로 인코딩되어야 한다. (binary encoding)

<state transition diagram>



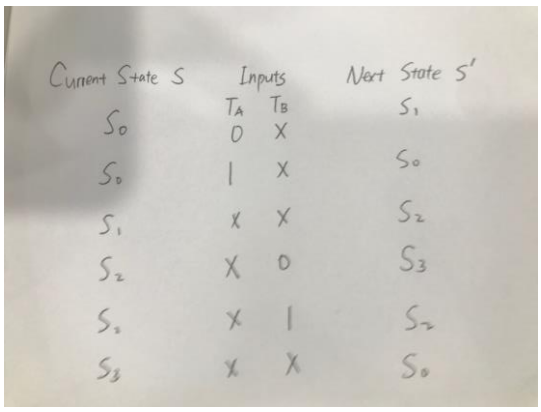
<state transition table with binary encoding>



Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S_1'	S_0'
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

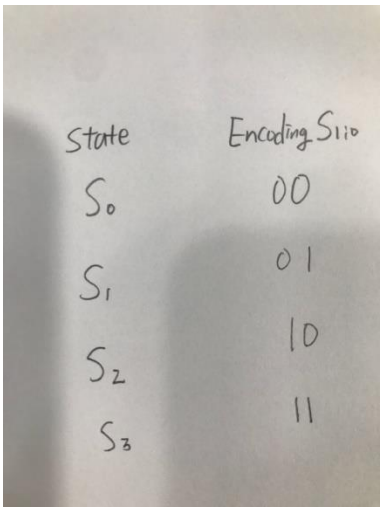
Binary encoding을 했을 때의 입력 값, 현재 상태, 다음 상태이다. State transition diagram을 토대로 하나씩 table을 살펴보면

<state transition table>



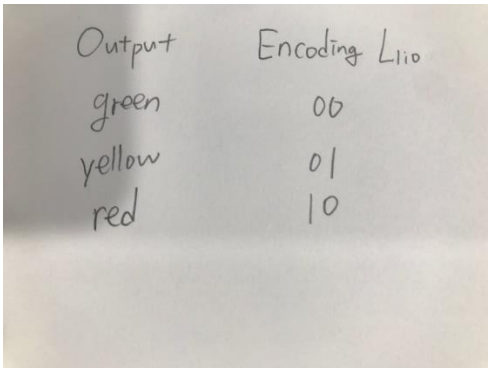
Current State S	Inputs T_A	Inputs T_B	Next State S'
S_0	0	X	S_1
S_0	1	X	S_0
S_1	X	X	S_2
S_2	X	0	S_3
S_2	X	1	S_2
S_3	X	X	S_0

<state encoding>



State	Encoding $S_{1:0}$
S_0	00
S_1	01
S_2	10
S_3	11

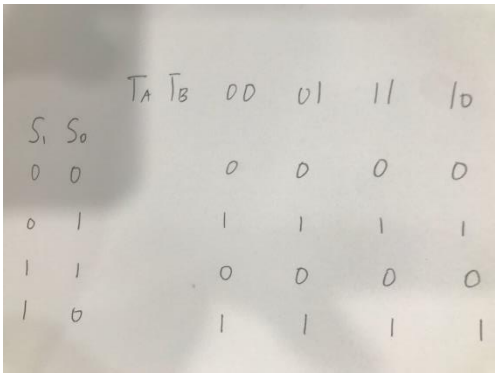
<output encoding>



Output	Encoding $L_{1:0}$
green	00
yellow	01
red	10

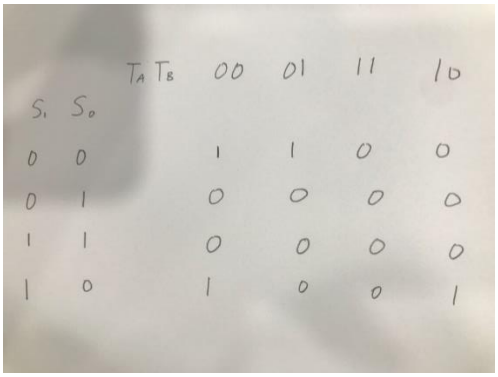
State transition table에 대한 카르노 맵을 그려보면

<Karnaugh map S1>



S_1	S_0	$T_A T_B$ 00	01	11	10
0	0	0	0	0	0
0	1	1	1	1	1
1	1	0	0	0	0
1	0	1	1	1	1

<Karnaugh map S2>



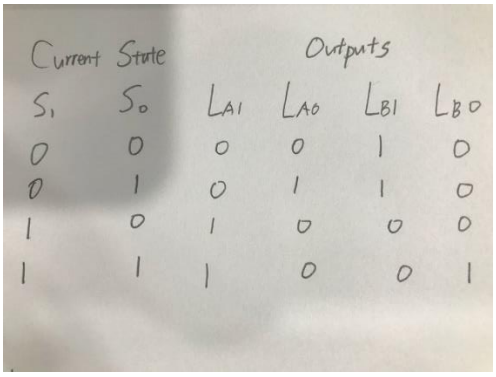
S_1	S_0	$T_A T_B$ 00	01	11	10
0	0	1	1	0	0
0	1	0	0	0	0
1	1	0	0	0	0
1	0	1	0	0	1

이를 계산하면

$S1`= S1 \oplus S0$이고

$S0`= S_1 S_0 T_a + S1 S_0 T_b$가 된다.

이를 이용해서 output의 table을 그려보면 다음과 같다.



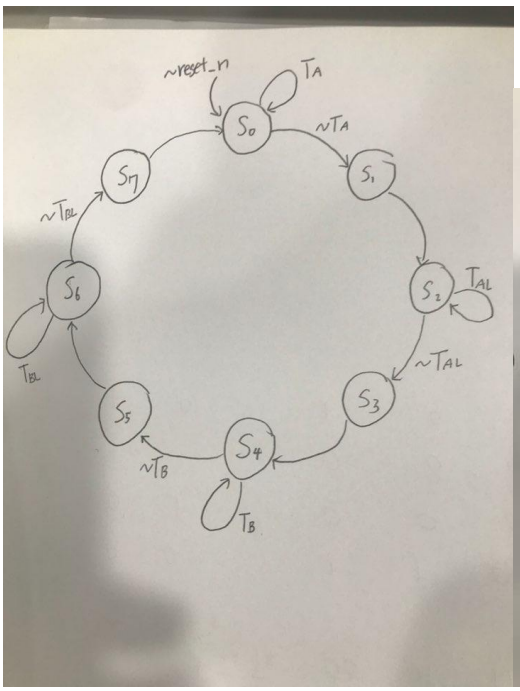
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

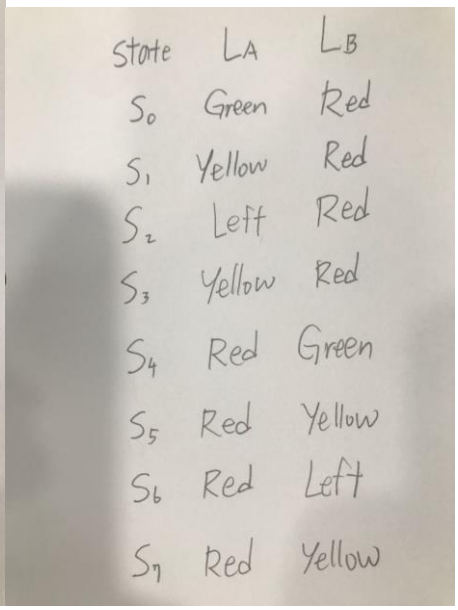
- Traffic Light Controller With Left Turn Signal

순서는 Traffic Light Controller와 같다.

먼저 상태와 입출력 값을 정의해준 다음 그에 대한 다이어그램을 그린다. 신호 Ta와 Tb는 직진하는 차를 확인하고, Tal과 Tbl은 좌회전하는 차를 확인한다. 다이어그램 및 상태와 출력 값 변화는 다음과 같다.

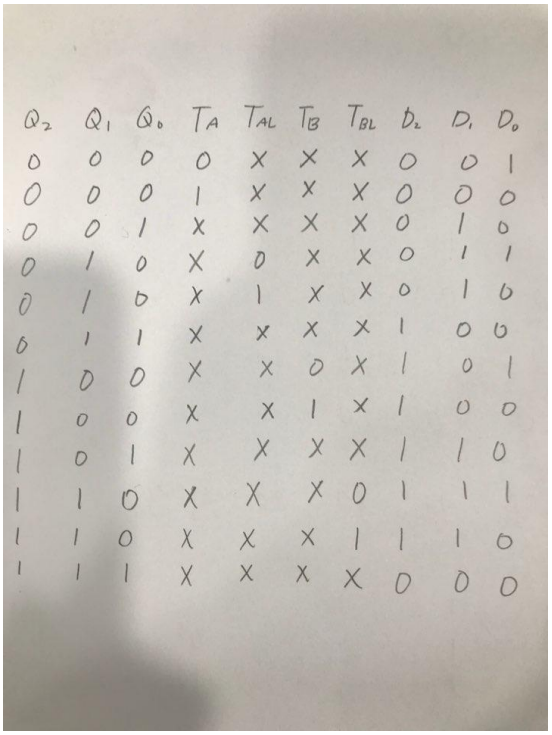
<state transition diagram>





S0 상태에서 A도로에 차가 있으면 Ta - 1이므로 상태를 유지하고, 차가 없으면 ~Ta이므로 S1상태로 이동한다. 이 때는 S1이 Yellow이므로, 바로 S2로 넘어가게 된다. S2에서는 좌회전 차량을 확인하는데 이전 과정과 같은 방식으로 좌회전 차량이 있으면 상태 유지, 없으면 다음 상태인 S3으로 넘어간다. 이 상태 또한 yellow이므로 S4로 넘어가게 된다. 이런 방식으로 S7까지 진행한 후 S0으로 돌아와서 이러한 과정을 반복하게 된다. 그리고 이 회로도 binary encoding방식을 이용한다.

<state transition table with binary encoding?>



Q_2	Q_1	Q_0	T_A	T_{AL}	T_B	T_{BL}	D_2	D_1	D_0
0	0	0	0	X	X	X	0	0	1
0	0	0	1	X	X	X	0	0	0
0	0	1	X	X	X	X	0	1	0
0	1	0	X	0	X	X	0	1	1
0	1	0	X	1	X	X	0	1	0
0	1	1	X	X	X	X	1	0	0
1	0	0	X	X	0	X	1	0	1
1	0	0	X	X	1	X	1	0	0
1	0	1	X	X	X	X	1	1	0
1	1	0	X	X	X	0	1	1	1
1	1	0	X	X	X	1	1	1	0
1	1	1	X	X	X	X	0	0	0

여기서 D0, D1, D2의 Boolean equation을 구하면

D2 = Q2`Q1Q0 + Q2Q1` + Q2Q2Q0'

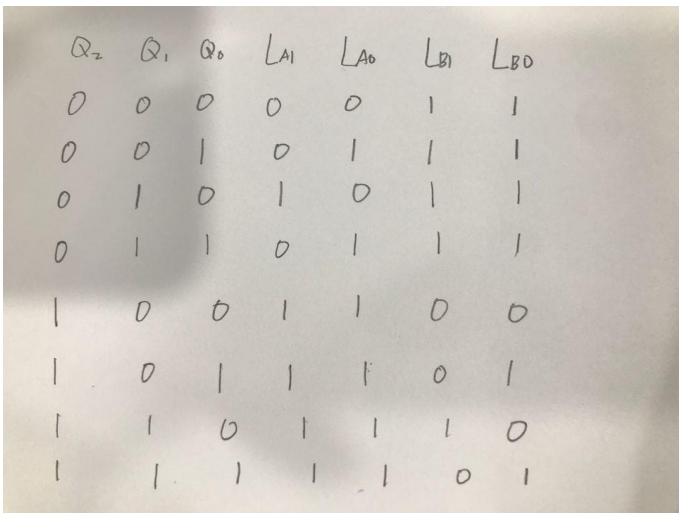
D1 = Q2`Q1`Q0 + Q1Q0` + Q2Q1`Q0

D0 = Q2`Q1`Q0`Ta` + Q2`Q1Q0`Tal` + Q2Q1`Q0`Tb` + Q2Q1Q0`Tbl`

이다.

Binary encoding에 의해 green = 00 yellow = 01, turn_left = 10, red = 11이라고 한다면 출력 값의 table은 다음과 같다.

<output encoding>



Q_2	Q_1	Q_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	0	1	1
0	0	1	0	1	1	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	1	1	0	0
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	1	0	1

이를 토대로 각 결과의 Boolean equation을 구하면 다음과 같다.

La1 = Q2`Q1Q0` + Q2Q1`Q0` + Q2Q1Q0` + Q2Q1Q0

La0 = Q2`Q1`Q0 + Q2`Q1Q0 + Q2Q1`Q0` + Q2Q1`Q0 + Q2Q1Q0` + Q2Q1Q0

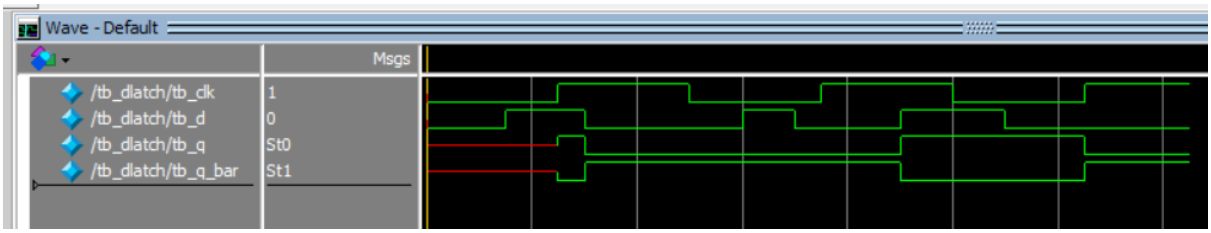
Lb1 = Q2`Q1`Q0` + Q2`Q1`Q0 + Q2`Q1Q0` + Q2`Q1Q0 + Q2Q1Q0`

Lb0 = Q2`Q1`Q0` + Q2`Q1`Q0 + Q2`Q1Q0` + Q2`Q1Q0 + Q2Q1`Q0 + Q2Q1Q0

4. 설계 검증 및 실험 결과

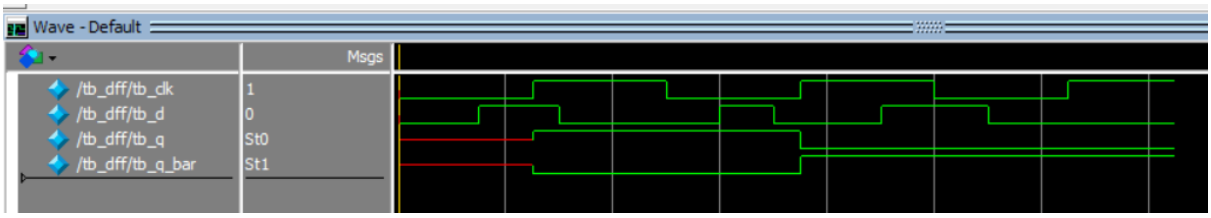
A. 시뮬레이션 결과

- D Latch



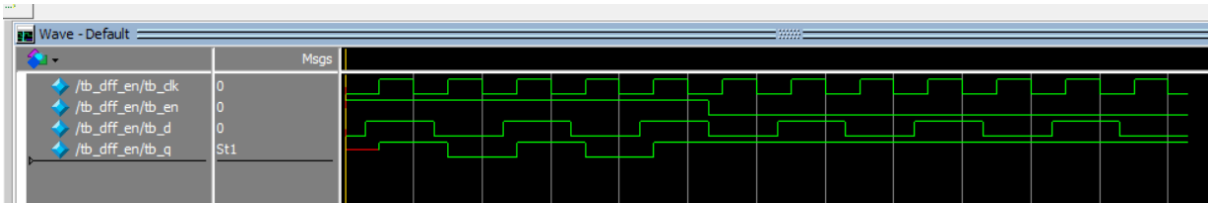
CLK는 단위시간 * 25마다 바뀌도록 설정했고 D값은 임의로 설정했다. 입력 값에 따른 출력 값이 나오는 걸 볼 수 있다.

- D flip-flop



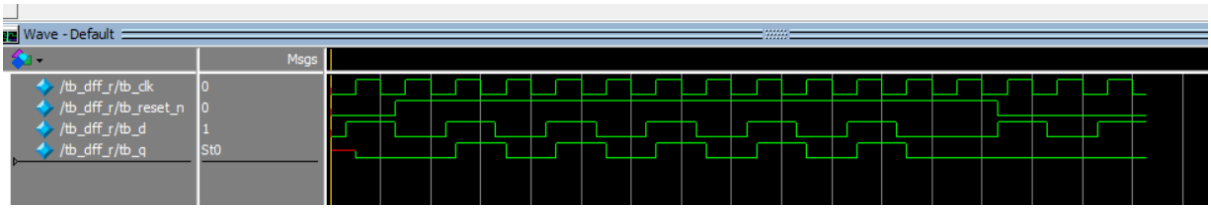
D Latch에서와 같은 시간에 CLK를 변하도록 설정했다. Rising edge일 때 출력 값이 변하는 걸 볼 수 있다.

- Enabled D flip-flop



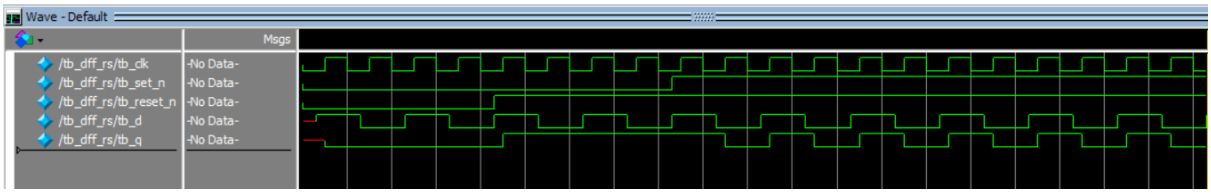
CLK와 D를 반복시키고 EN값에 변화를 주었다. EN이 0이 되면 결과 값이 이전 값을 계속 유지하는 것을 확인할 수 있다.

- Resettable D flip-flop



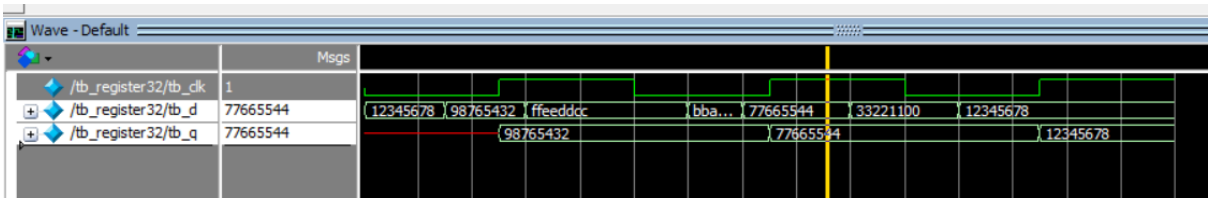
CLK와 D를 반복시켰다. Reset이 1일 때 CLK의 변화에 따라 Q의 값이 변하는 걸 확인할 수 있다. Reset이 0이면 Q의 값이 변하지 않는다.

- Synchronous Set/Resettable D flip-flop



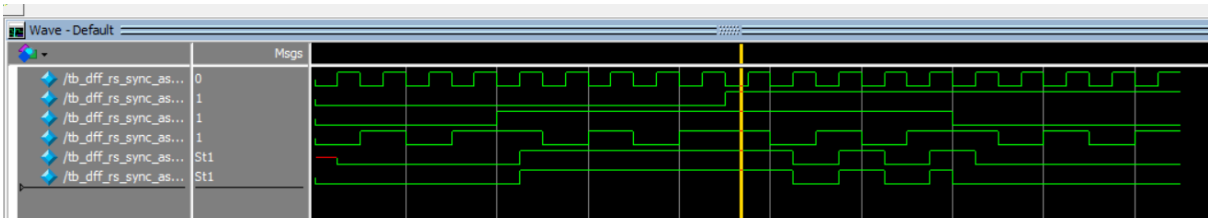
단위시간 * 10으로 STEP을 설정하고 STEP/6마다 CLK 값이 변하도록 설정했다. Set과 reset이 둘 다 0일 때 결과는 0 값을 유지하다가 reset이 1이 되면 flip-flop에 들어가는 입력 값이 1이므로 q가 1인걸 볼 수 있다. Set과 reset이 둘 다 1이 되면 d의 변화에 맞춰 q값이 변하는 걸 볼 수 있다.

- Register



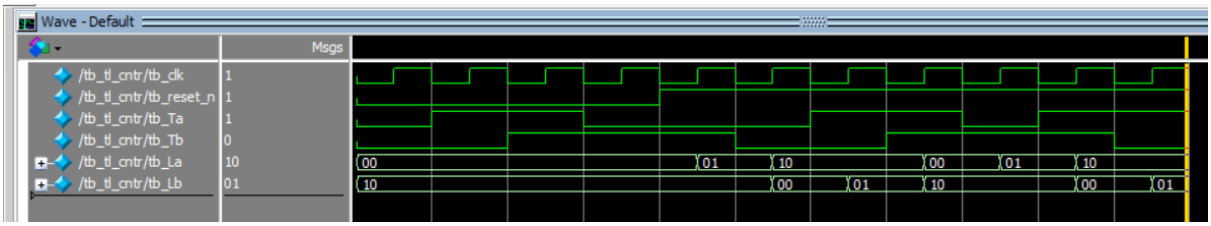
CLK을 주기적으로 변하도록 하고 D에 32비트 16진수 값을 입력했다. 그에 따라 Q값이 변화하는 것을 볼 수 있다.

- Async/Sync Set/Resettable D flip-flop



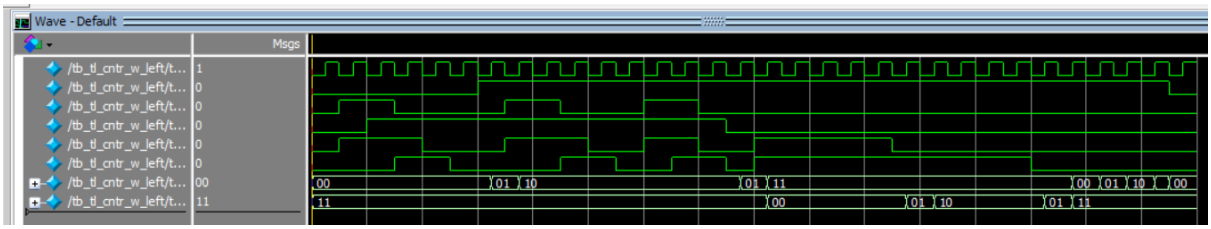
Set과 reset을 4가지 경우로 나누어 진행했다. 입력 값의 변화에 맞추어 sync와 async가 변하다가 마지막에 set = 1, reset = 0일 때 async는 testbench의 조건에 의해 즉시 변하는 것에 비해 sync는 CLK이 rising edge가 되었을 때 변하는 것을 볼 수 있다.

- Traffic Light Controller



매 5초마다 clk 값을 반복하도록 설정했다. Tb_reset_n이 0일 때는 출력 값이 0인걸 볼 수 있다. Tb_reset_b이 1이 되면 입력 값의 변화에 맞춰 tb_La, tb_Lb가 변하는 걸 볼 수 있다.

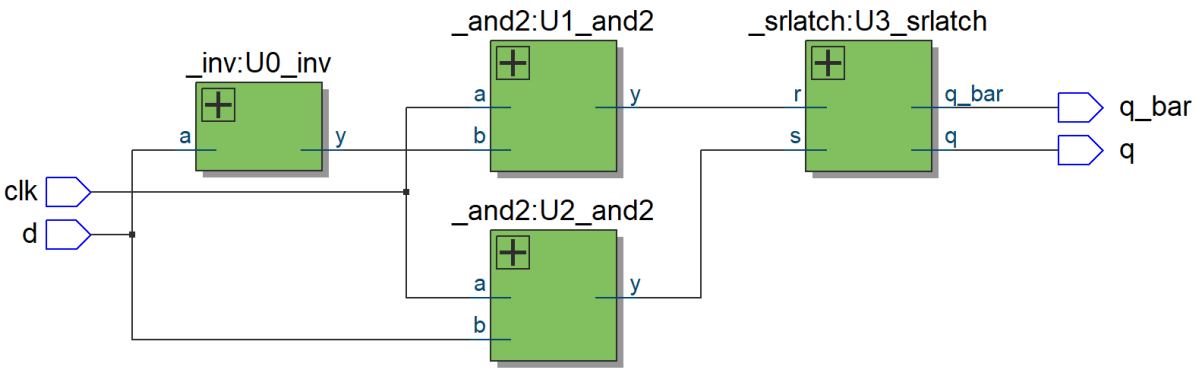
- Traffic Light Controller With Left Turn Signal



위와 마찬가지로 매 5초마다 clk 값이 반복된다. Tb_reset_n이 0이면 출력 값이 0이고 tb_reset_n = 1일 때부터 testbench에서 설정한 입력 값에 맞춰서 결과값이 나오는 걸 볼 수 있다.

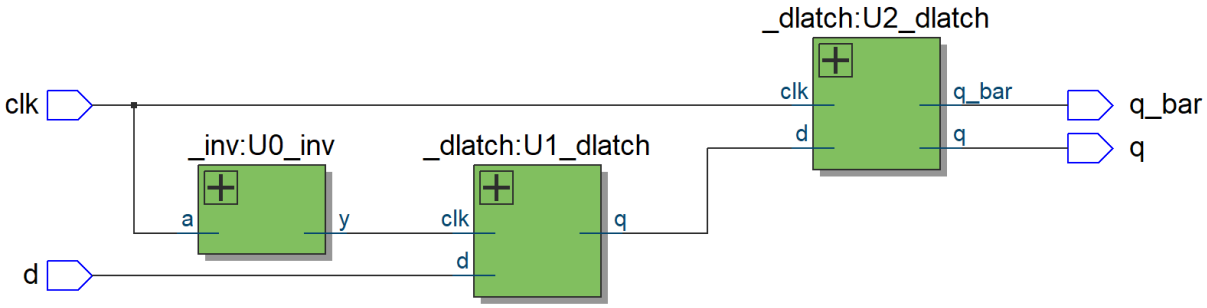
B. 합성(synthesis) 결과

- RTL viewer
  - D Latch



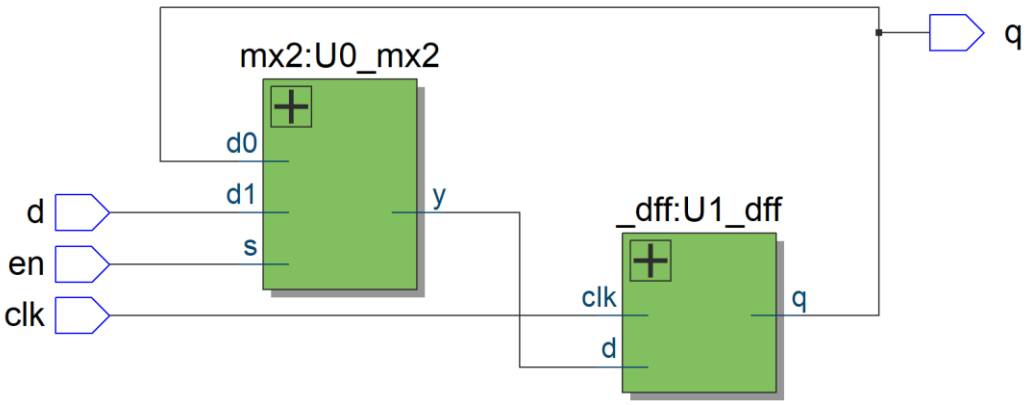
SR Latch + 2*2input AND gate

- D flip-flop



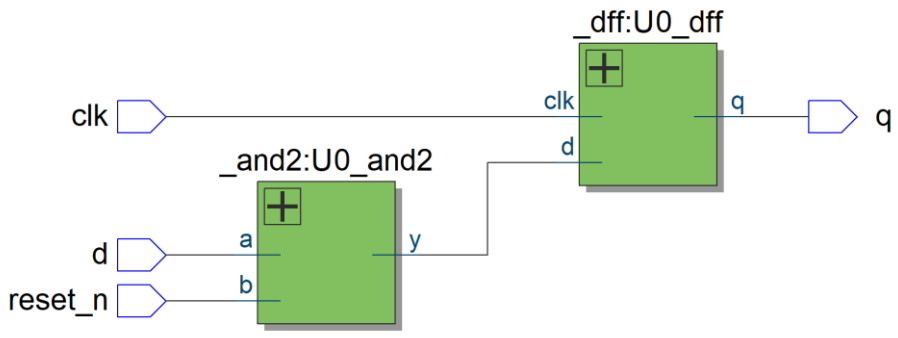
2* D Latch + Inverter

- Enabled D flip-flop



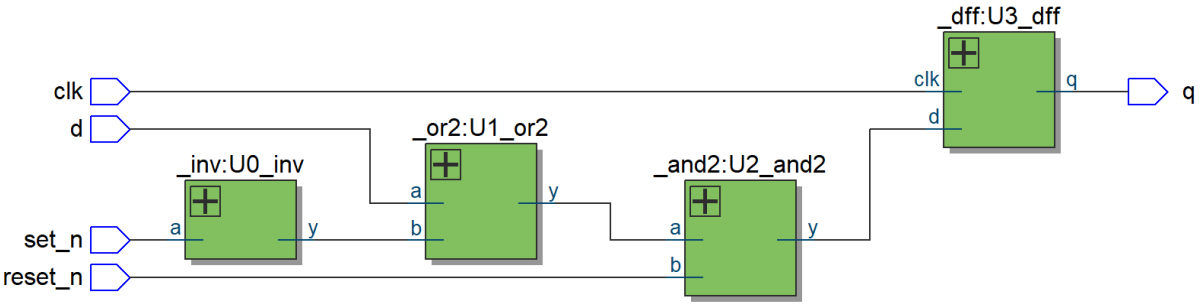
D flip-flop + 2 to 1 mux

- Resettable D flip-flop



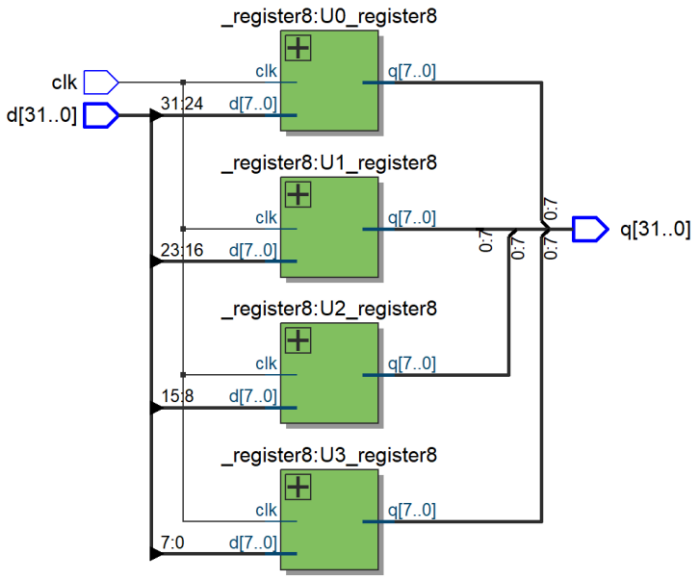
D flip-flop + AND gate

- Synchronous Set/Resettable D flip-flop



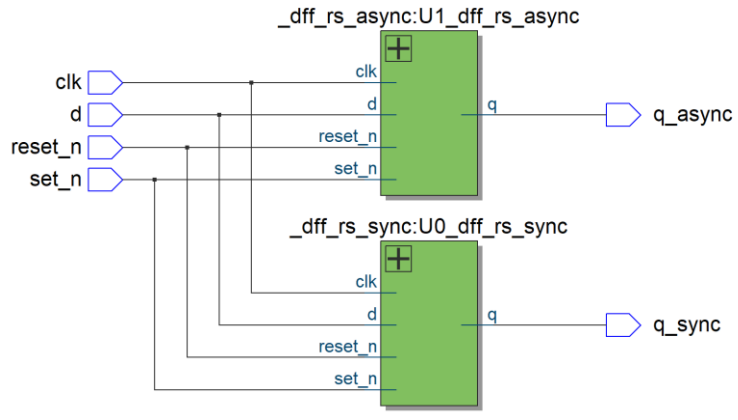
Resettable D flip-flop + 2input OR gate + Inverter

- Register



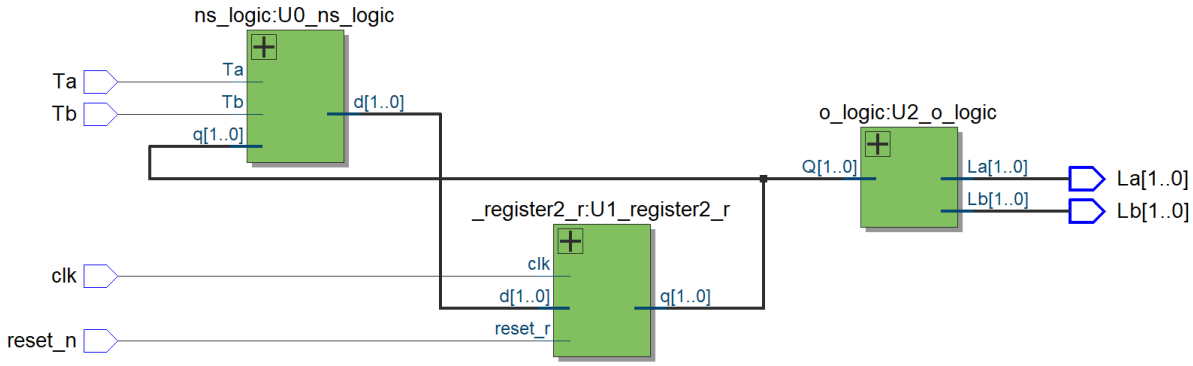
8비트 레지스터를 4개 연결했다. 하나의 CLK을 공유하고 있고 모든 데이터가 동시에 한 방향으로 흐른다.

- Async/Sync Set/Resettable D flip-flop



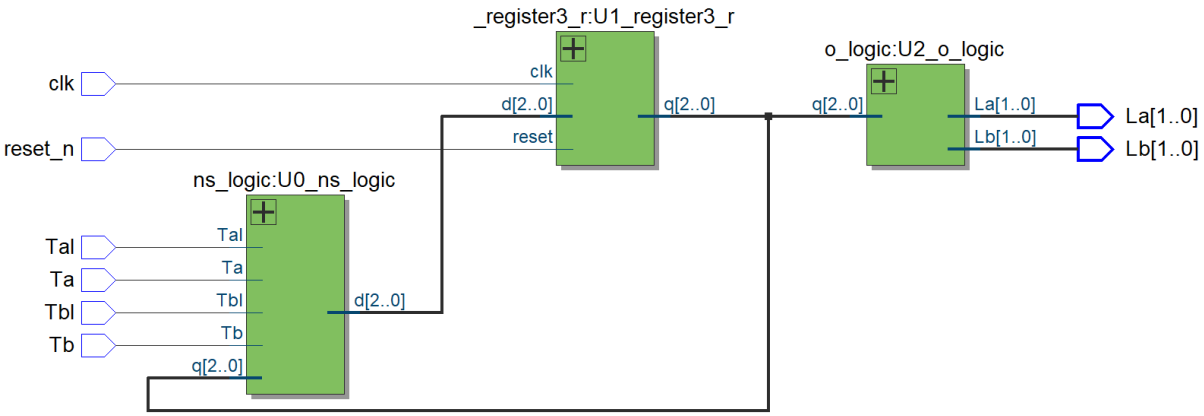
Sync와 Async를 구분해주기 위해 각각의 D flip-flop을 따로 설치했다.

- Traffic Light Controller



다음 상태를 결정해주는 ns_logic 회로를 제일 앞에 설계하고 결과 값이 레지스터에 들어가도록 한다. 새 결과 값을 받기 위해 reset기능이 있는 레지스터를 중간에 설치하고 결과값을 출력하는 o_logic 회로를 마지막에 설치한다. 레지스터는 clk 및 다른 input값에도 영향을 받는 asynchronous를 사용한다.

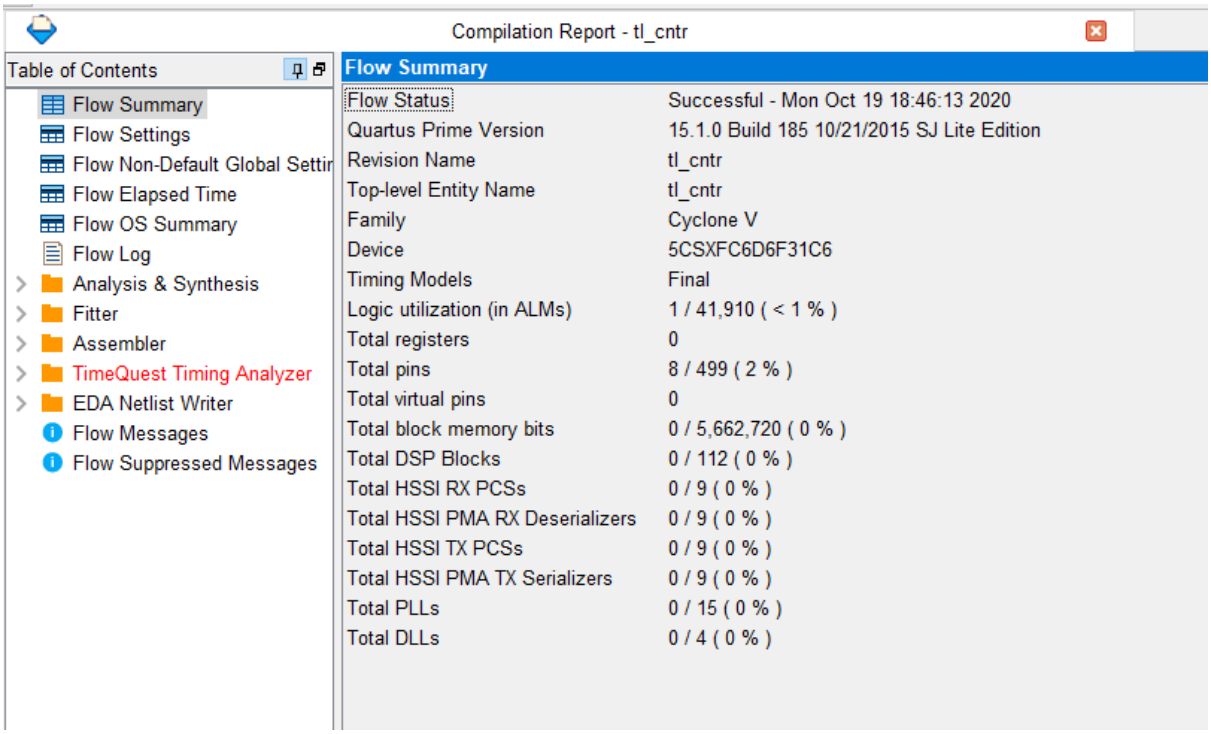
- Traffic Light Controller With Left Turn Signal



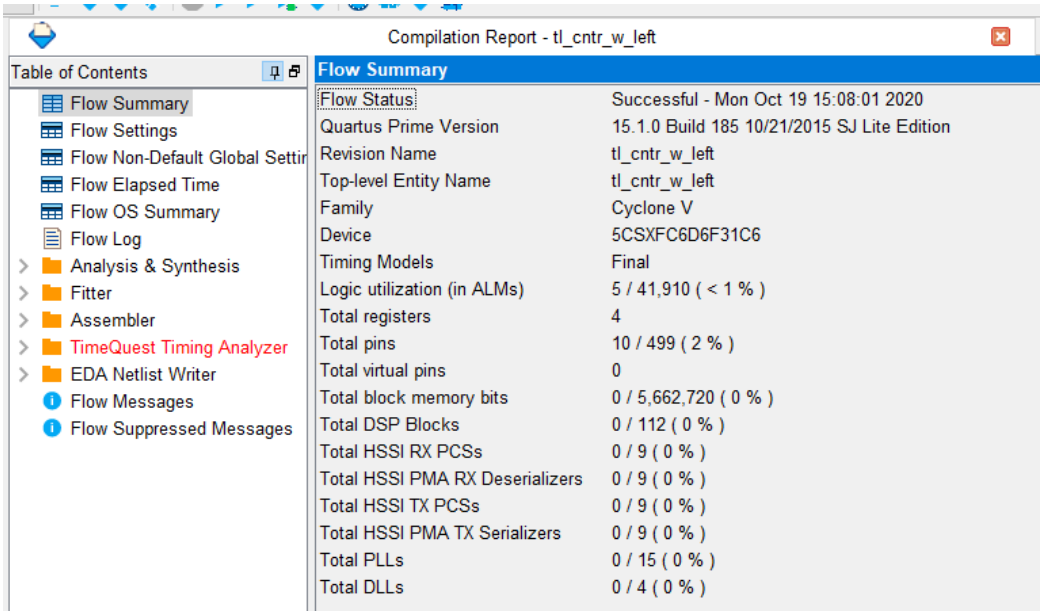
직진 또는 좌회전을 감지하는 기능을 추가한 next state logic이 제일 앞에 붙는다. 이 다음은 이전 회로와 같이 reset 기능이 있는 레지스터를 가운데 붙여주고, 결과 값을 출력하는 output logic을 붙여준다.

- Flow Summary

  - Traffic Light Controller



  - Traffic Light Controller With Left Turn Signal



5. 고찰 및 결론

A. 고찰

- D FF with active-low synchronous reset and set과 D FF with active-low asynchronous reset and set의 차이

Sync 회로는 CLK의 변화에 즉각적으로 영향을 받고, Async 회로는 CLK의 변화에 큰 영향을 받지 않는다. Synchronous는 "동시에 일어나는" 이라는 뜻을 가지고 있는데, CLK값 과 결과 값이 동시에 움직여야 하는 회로라고 볼 수 있다. Asynchronous는 반대로 "동시에 일어나지 않는" 이라는 뜻이다. 즉 CLK 값이 변하더라도 결과 값이 그대로 일 수 있다.

- FSM

컴퓨터 공학 기초 실험 1 시간 FSM을 조직한 적이 있었는데, 그 때 보다 계산해야 할 양이 많았던 것 같다. 특히 Karnaugh map을 계산하는 과정에서 하나라도 실수할 경우 모든 값이 이상해지는 결과가 나오기 때문에 더욱 집중해서 계산했던 것 같다.

B. 결론

Traffic Light Controller With Left Turn Signal (이하 LTC)의 FSM Encoded State Transition Table만 중 D2만 구한다고 하더라도 계산해야 할 양이 꽤 많았다. 계산 값에 따라 logic 이 달라지기 때문에 코딩도 코딩이지만 코딩하기 이전 계산 과정도 매우 중요하다.

6. 참고문헌

이준환/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2020
공영호/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2020