컴퓨터 공학 기초 실험2 보고서

실험제목: Traffic Light Controller with/without Left Turns Signals

실험일자: 2022년 10월 11일 (화)

제출일자: 2022년 10월 18일 (화)

학 과: 컴퓨터정보공학부

담당교수: 공영호 교수님

실습분반: 화요일 0,1,2

학 번: 2019202050

성 명: 이강현

1. 제목 및 목적

A. 제목

Traffic Light Controller with/without Left Turns Signals

B. 목적

Finite State Machine(FSM)에 대해 이해하고 FSM의 예시로 Traffic Light Controller를 Verliog를 통하여 구현해본다. 저번 시간에 구현한 Combinational Logic을 가지고 구현에 사용하며 Traffic Light Controller를 이해하고 응용하여 state가 더 추가된 Traffic Light Controller with left signal도 Verlilog 코딩을 통하여 구현해본다.

2. 원리(배경지식)

<Finite State Machine(FSM)>

유한 상태 기계(FSM)란 하나의 상태만 취할 수 있는 기계를 말한다. 유한 상태 기계는 한 번에 하나의 상태를 가질 수 있으며 현재 상태는 외부로부터 입력을 받으면 조건에 따라 정해진 다른 상태로 변할 수 있다.

<Moore/Mearly FSM>

Moore/Mearly FSM은 FSM의 종류로 출력 값에 있어서 차이가 있다. Moore Machine은 출력이 현재 상태에만 의존하여 출력값이 정해진다는 점, Mearly Machine은 출력이 현재 상태와 입력에 의존하여 나온다는 점에서 차이가 있다. 아래의 그림과 같이 Moore는 output logic의 input 값으로 state 밖에 없는데 Mearly Machine의 input으로는 state와 input이 존재한다. 이번 traffic light controller에서는 Moore Machine을 이용하여 설계할 예정이다.

<Finite State Machine Encoding>

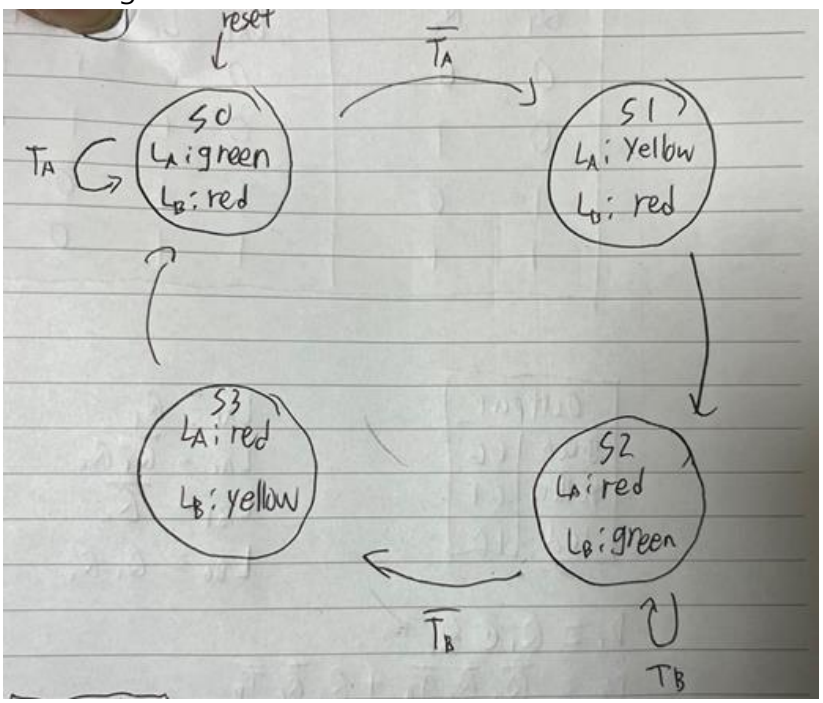
현재 상태, 다음 상태, input 등을 숫자로 표현하여 구분하기 위하여 Encoding을 하는데 Encoding에는 크게 Binary Encoding과 One-hot Encoding이 존재한다. Binary Encoding은 이진수로 표현하는 방법인데 예를 들어 state가 8가지라면 3-bit을 이용하여 000부터 111 까지 총 8가지의 상태를 표현할 수 있다. 따라서 State를 나타내는데 총 3개의 Flip-Flop 을 필요로 한다. One-hot Encoding은 한 순간에 하나의 Flip-Flop만 1이고 나머지 Flip- Flop은 모두 0이다. 따라서 8가지의 상태를 나타내기 위해서는 00000001,00000010,00000100 ~ 10000000과 같이 나타낸다. One-hot Encoding은 binary encoding에 비해 flip flop을 더 많이 사용하지만 상태가 변할 때 하나의 비트만 변하면 되므로 전력소비 측면에서 더욱 효율적이다.

3. 설계 세부사항

Traffic Light Controller

L_A, L_B 두가지의 신호등이 존재하는데 각각 다른 도로의 차량 통행을 제어하는 신호등이 다. 신호등의 원리는 빨간색, 초록색, 노란색이 존재하는데 도로에 차량이 있으면 초록색 이고 차량이 없으면 빨간색이다. 초록색에서 빨간색으로 바뀔 때 노란색을 무조건 거쳐야 한다. 또한 L_A , L_B 가 같은 색을 가지면 안된다.

1) State diagram



2) Encoding state

State	Encoding
S0	00
S1	01
S2	10
S3	11

Output	Encoding
Green	00
Yellow	01
Red	10

3) State Transition Table

Current State		Inputs		Next State	
Q_1	Q_0	T_A	T_B	D_1	D_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

진리표가 있기에 1을 찾아 논리식을 구하는 sop(sum of product)방법을 사용하여 논리식을 구하고 간소화하였다.

$D_1 = Q_1'Q_0 + Q_1Q_{0'} = Q_1 \oplus Q_0$

$D_0 = Q_1'Q_0'T_A' + Q_1Q_0'T'_B$

4) Output Logic

Current State		Outputs			
Q1	Q0	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

Output logic 또한 sop방법을 사용했다.

LA1=Q1

LA0 =Q1′Q0

LB1 =Q1′

LB0 = Q1Q0

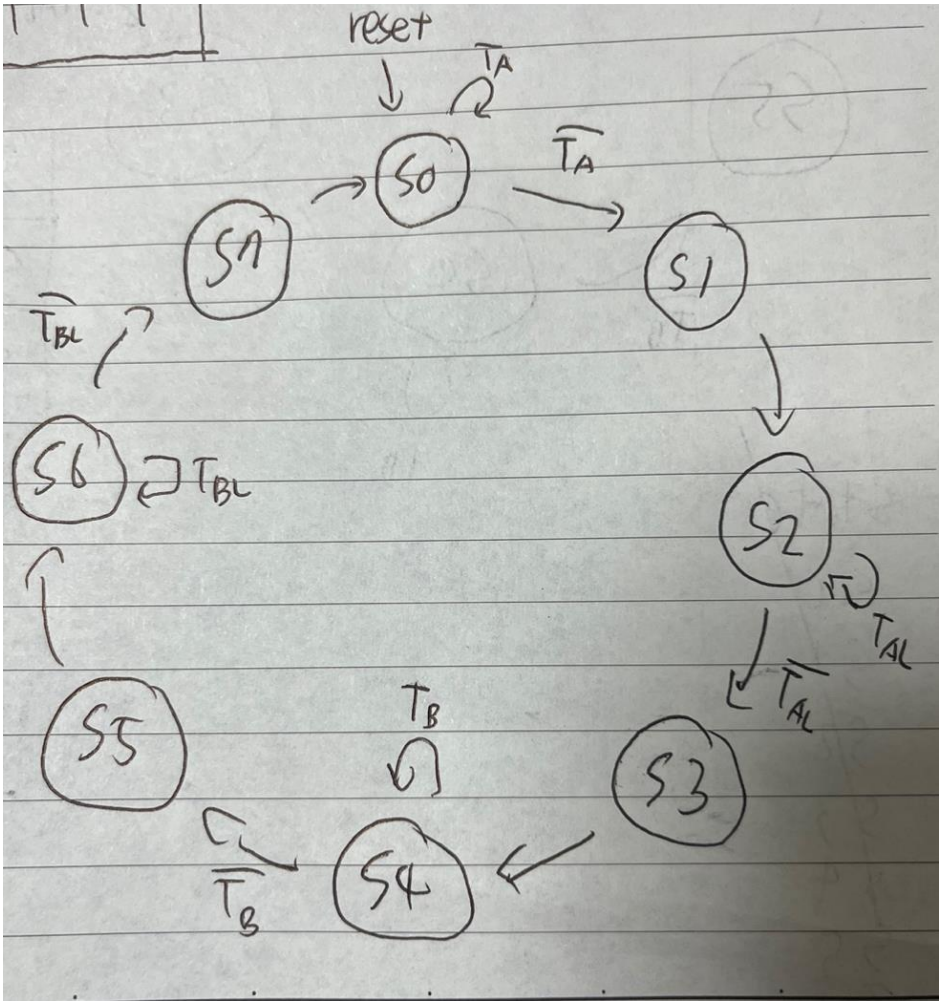
<module description>

구분	이름	설명
Top module	tl_cntr	Traffic light controller의 top module
Sub module	ns_logic	Traffic light controller의 next state를 결정하는 combinational logic
Sub module	_register2_r_async	내부에 dff_r_async를 instance -현재 state의 값을 저장하고 있다..
Sub module	_dff_r_async	Resettable D flip-flop with active low asynchronous reset
Sub module	o_logic	현재 state의 값을 바탕으로 output 값을 결정하는 combinational logic

Traffic Light Controller with Left Turn Signals

Traffic Light Controller에 좌회전 신호를 추가한 것으로 좌회전에 대한 차량 감시 신호인T_{AL}, T_{BL}이 추가되었다. State는 이전과 유사하지만 좌회전 차량이 있는 경우 state를 유지 해주고 좌회전 차량이 없을 시 다음 state로 넘어간다.

1) State Diagram



2) Encoding States

State	L_A	L_B
S0	Green	Red
S1	Yellow	Red
S2	Left	Red
S3	Yellow	Red
S4	Red	Green
S5	Red	Yellow
S6	Red	Left
S7	Red	Yellow

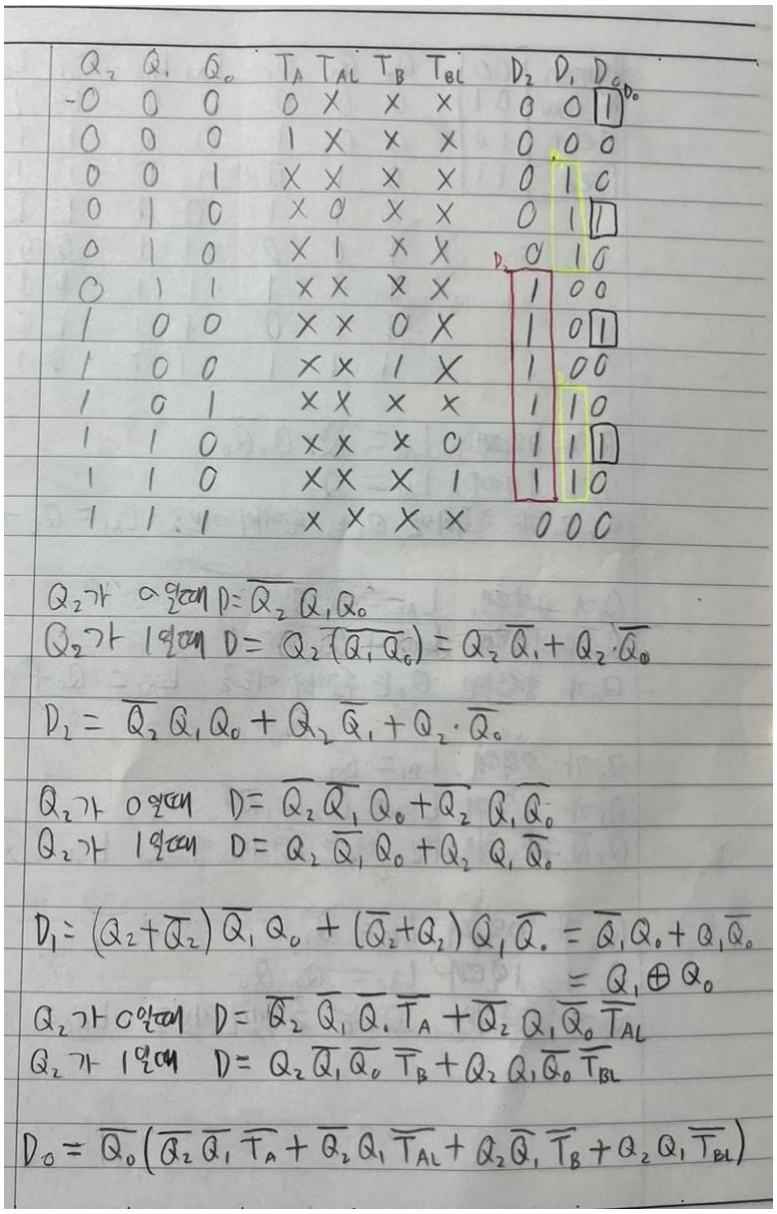
State	Encoding
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

Color	Code
Green	00
Yellow	01
Left	10
Red	11

3) State Transition Table

Current State			Inputs				Next State		
Q2	Q1	Q0	T_A	T_{AL}	T_B	T_{BL}	D2	D1	D0
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

위의 진리표를 가지고 sop(sum of product)를 사용하고 식을 간소화하였다.



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

Q_2가 0일때 $D=\overline{Q_2}Q_1Q_0$

Q_2가 1일때 $D=Q_2(\overline{Q_1Q_0})=Q_2\overline{Q_1}+Q_2\cdot\overline{Q_0}$

$D_2=\overline{Q_2}Q_1Q_0+Q_2\overline{Q_1}+Q_2\cdot\overline{Q_0}$

Q_2가 0일때 $D=\overline{Q_2}\,\overline{Q_1}Q_0+\overline{Q_2}Q_1\overline{Q_0}$

Q_2가 1일때 $D=Q_2\overline{Q_1}Q_0+Q_2Q_1\overline{Q_0}$

$D_1=(Q_2+\overline{Q_2})\overline{Q_1}Q_0+(\overline{Q_2}+Q_2)Q_1\overline{Q_0}=\overline{Q_1}Q_0+Q_1\overline{Q_0}$

$=Q_1\oplus Q_0$

Q_2가 0일때 $D=\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}\,\overline{T_A}+\overline{Q_2}Q_1\overline{Q_0}\,\overline{T_{AL}}$

Q_2가 1일때 $D=Q_2\overline{Q_1}\,\overline{Q_0}\,\overline{T_B}+Q_2Q_1\overline{Q_0}\,\overline{T_{BL}}$

$D_0=\overline{Q_0}(\overline{Q_2}\,\overline{Q_1}\,\overline{T_A}+\overline{Q_2}Q_1\overline{T_{AL}}+Q_2\overline{Q_1}\,\overline{T_B}+Q_2Q_1\overline{T_{BL}})$

D2=Q2′Q1Q0+Q2Q1′ + Q2Q0′

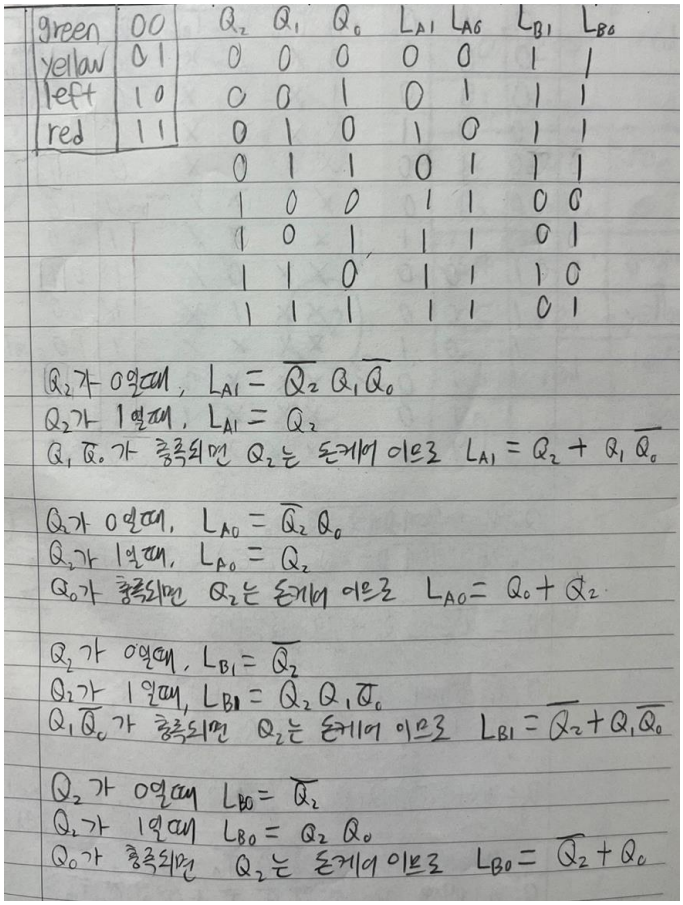
D1=Q1 ⊕ Q0

D0=(Q2′Q1′TA′+ Q2′Q1T_{AL}′ + Q2Q1′Tb′+ Q2Q1T_{BL}′)Q0

4) Output Table

Current State			Outputs			
Q2	**Q1**	**Q0**	**LA1**	**La0**	**Lb1**	**Lb0**
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	0	0
1	1	1	1	1	0	1

위의 표를 통하여 Boolean equation을 구하면 아래와 같이 나온다.



따라서 논리식은

LA1 = Q2 + Q1Q0'

LA0 = Q0 + Q2

LB1 = Q2′ + Q1Q0′

LB0 = Q2′ + Q0

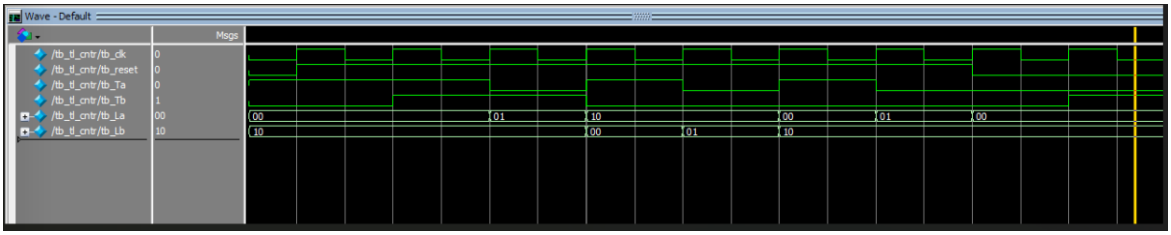
<module description>

구분	이름	설명
Top module	tl_cntr_w_left	Traffic light controller의 top module
Sub module	ns_logic	Traffic light controller의 next state를 결정하는 combinational logic
Sub module	_register3_r	내부에 dff_r_async를 instance -현재의 state 값을 저장.
Sub module	_dff_r_async	Resettable D flip-flop with active low asynchronous reset
Sub module	o_logic	현재 state의 값에 기반하여 output 값을 결정하는 combinational logic

4. 설계 검증 및 실험 결과

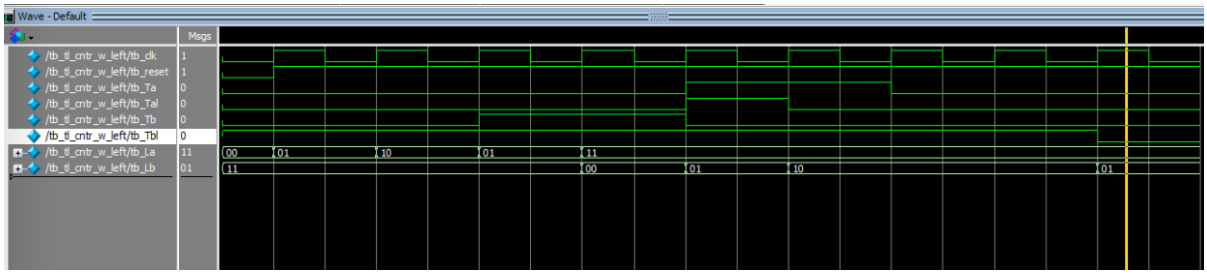
A. 시뮬레이션 결과

Traffic Light Controller



위의 시뮬레이션 결과를 보면 초기상태는 La가 00(green) Lb가 10(red)로 되어있는 것을 확인할 수 있다. Reset이 끝나고 첫 rising edge를 만났으나 Ta가 1,즉 차가 도로에 있기 때문에 La는 green을 유지하는 것을 확인할 수 있다. 그 이후 다음 rising edge에서 Ta가 0 즉, 차가 없으니 01(yellow)를 거쳐 10(red)로 변하면서 Lb가 00(green)으로 변한다. Lb는 Tb가 0이므로 바로 다음 rising edge에서 01(yellow)를 거쳐 10(red)로 변한다.

Traffic Light Controller with left signal

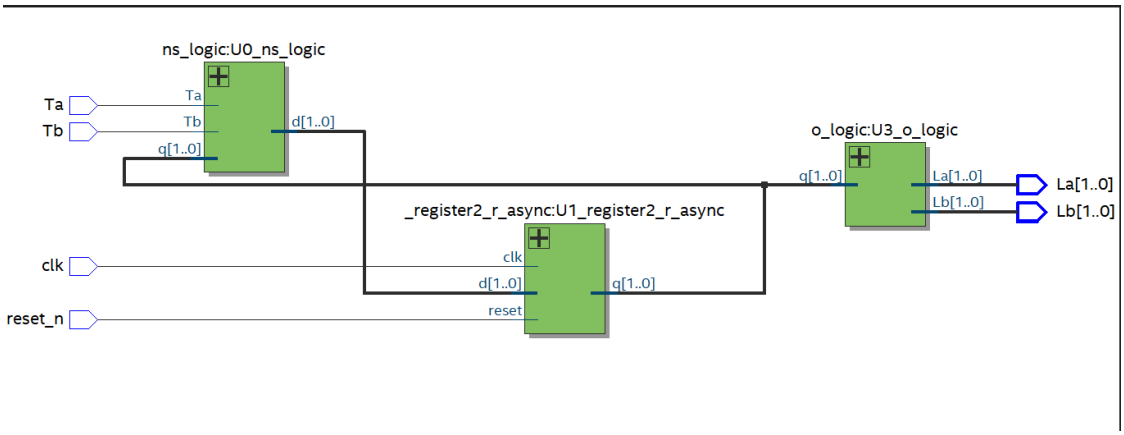


위의 wave를 보면 La는 00(green)에서 시작하여 Ta가 0이므로 01(yellow)를 통해 10(left)로 바뀐다. 그 후에 Tal이 0이므로 다시 01(yellow), 11(red)로 변화한다. 그와 동시에 Lb또한 00(green)으로 변화한다. La가 11(red)로 변화하기전 Lb는 11(red)를 유지하는 것을 확인할 수 있다. 그 후 Tb가 0이됨과 동시에 rising edge를 만나 Lb는 01(yellow)로 변화하며 그 이후 10(left)가 된다. Tbl이 1을 유지하고 있으므로 Lb는 10(left)에서 변화하지 않고 값을 유지하고 있으며 Tbl이 0이 되자마자 rising edge를 만나 Lb또한 01(yellow)로 변하는 것을 확인 할 수 있다.

B. 합성(synthesis) 결과

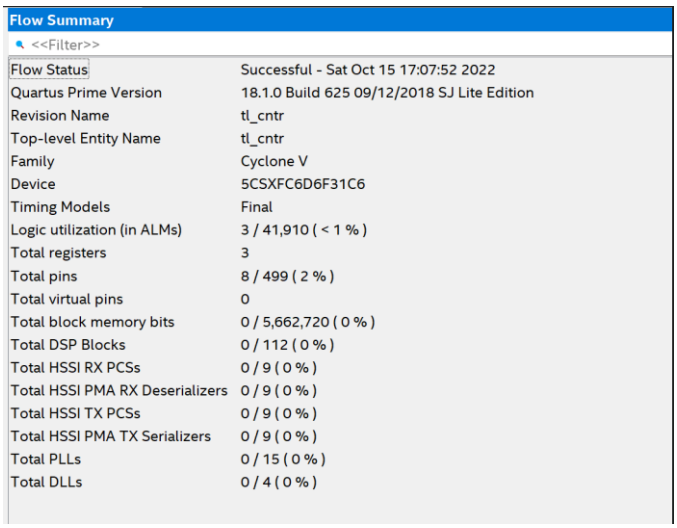
Traffic Light Controller

<RTL Viewer>



Ns logic을 통과한 값들이 register의 입력값으로 o logic을 통과한 값들이 출력값과 ns logic의 입력으로 잘 연결되어 회로가 잘 형성된 것을 확인할 수 있다.

<Flow Summary>



Flow Summary

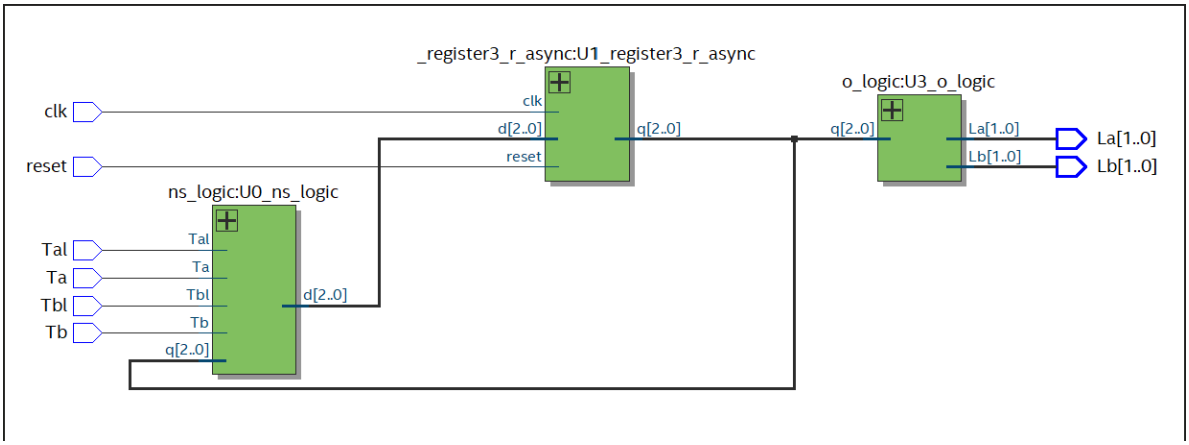
<<Filter>>

Flow Status	Successful - Sat Oct 15 17:07:52 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	tl_cntr
Top-level Entity Name	tl_cntr
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	3 / 41,910 (< 1 %)
Total registers	3
Total pins	8 / 499 (2 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

구현하는데 사용된 logic elements는 3개이며, pin은 8개가 사용된 것을 볼 수 있다.

Traffic Light Controller with left signals

<RTL Viewer>



Traffic Light Controller에 state만 추가 되어 state들이 3비트로 형성되었고 회로가 잘 연결된 것을 확인할 수 있다.

<Flow Summary>

Flow Summary

<<Filter>>

Flow Status	Successful - Sat Oct 15 20:27:15 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	tl_cntr_w_left
Top-level Entity Name	tl_cntr_w_left
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	3
Total pins	10
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Flow summary에 성공적인 결과를 확인할 수 있다.

5. 고찰 및 결론

A. 고찰

이번 실험은 코드를 구현하는 것보다 각 단계를 설계하는 것에서 시간을 많이 소요했다. 각 state를 어떻게 정의할 것인지를 생각해야하고 논리식을 정말 신중하게 구해내야했다. 조금만 잘못되어도 출력값이 변하지 않는다던지 단계가 변하지 않는다던지등의 문제들이 생겼기 때문이다. 실제로 이번실험에서 2-input and gate를 잘못 사용하여 값들이 전혀 변하지 않아 출력값을 얻지 못하는 경우도 생겼다. 이처럼 FSM을 설계할때는 입력논리, 출력논리를 정확하게 구해내고 어떻게 설계할 것인지가 제일 중요하다는 것을 배웠다.

B. 결론

이번 실험에서는 FSM(finite state machine)을 traffic light controller를 구현해보는 것을 통해 설계하는 법과 verilog로 구현하는 법을 배웠다.
유한상태머신으로 각 상황을 state로 생각하고 encoding하고 이를 논리식을 통해서 조건을 만족하면 다음 단계로 넘어가는 것을 이용한다면 일상생활에서 이루어지는 것들을 모두 설계할 수 있을 것 같고 앞으로 유용하게 사용할 수 있는 개념을 배운 것 같다.

6. 참고문헌

공영호 교수님/컴퓨터공학기초실험2/2022