컴퓨터 공학 기초 실험2 보고서

실험제목: Traffic Light Controller

with/without Left Turn Signals

실험일자: 2022년 10월 11일 (화)

제출일자: 2022년 10월 18일 (화)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

1. 제목 및 목적

A. 제목

Traffic Light Controller with/without Left Turn Signals

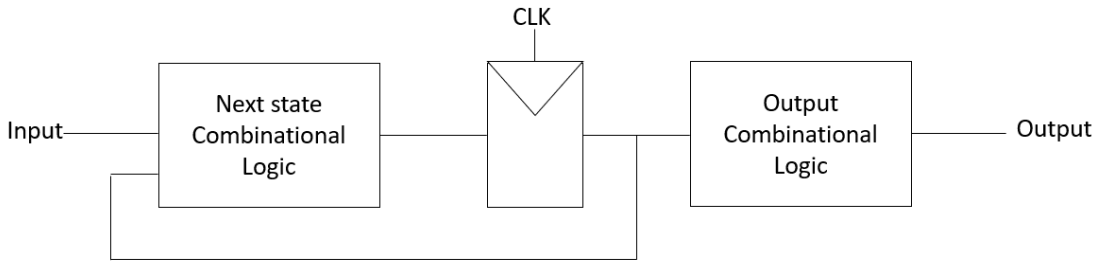
B. 목적

Finite State Machine, FSM의 기법 중 하나인 인 Moore FSM을 이용해서 Traffic Light Controller을 구현한다. Output이 현재 상태에 의해 결정되는 Moore FSM을 이해한다. 좌회전 신호가 있는 경우를 추가로 구현해보고, 좌회전 신호가 있는 경우와 없는 경우의 차이점을 알아본다.

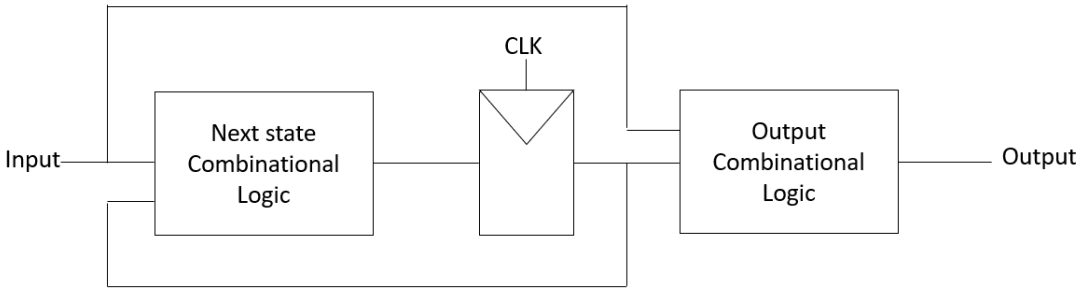
2. 원리(배경지식)

- FSM

Finite State Machine의 약자인 FSM은 유한 상태 기계를 의미하며 컴퓨터 프로그램과 논리 회로를 설계하는 데에 쓰인다. FSM은 현재 상태를 저장하는 register와 combination logic인 다음 상태를 결정하는 next logic, output을 결정하는 output logic으로 구성되어 있다. FSM은 현재 상태로부터 가능한 전이 상태, 그리고 이러한 전이를 유발시키는 조건들의 집합으로 정의된다. 여기서 전이는 특정 사건에 의해 다른 상태로 바뀌는 것을 의미한다. FSM의 종류로는 현재상태에 의해서만 output이 결정되는 Moore FSM과 현재 상태와 input에 의해서만 output이 결정되는 Mealy FSM이 있다.



위의 사진은 Moore machine의 설계도로 위에서 말했던 것처럼 현재상태와 input에 의해서만 output이 결정된다. 위의 사진에서 볼 수 있듯이 input과 output combinational logic이 연결되어 있지 않다.



위의 사진은 Mealy machine의 설계도로 위에서 말했던 것처럼 현재상태에 의해서만 output이 결정된다. 위의 사진에서 볼 수 있듯이 input과 output combinational logic이 연결되어 있다.

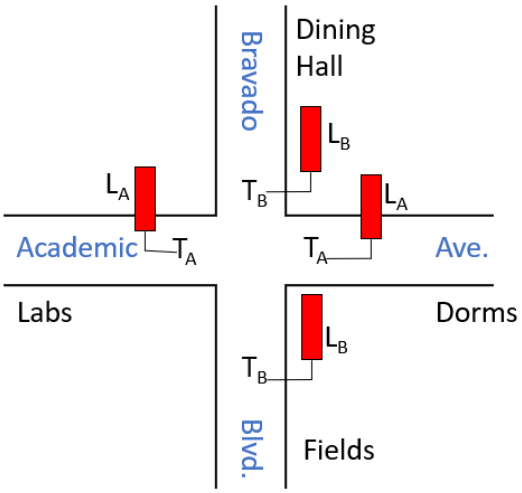
다음으로 FSM 설계에 대해 말해보면 다음과 같다.

1. Finite state diagram을 그린다.
    1-1. states 정의.
    1-2. inputs 정의.
    1-3. outputs 정의.
    1-4. diagram 그리기.
2. Encoding states
3. module header 코딩하기.
4. state registers(flip-flops) 코딩하기. -sequential circuits
5. combinational circuits 코딩하기.

이때 Encoding states를 할 때에는 binary encoding or one-hot coding이 가능하다. 여기서 one-hot encoding으로 encoding을 하는 경우 4bit로 표현해야 한다. One-hot encoding의 경우 사용자가 보기 편하다는 장점이 있지만 같은 개수의 state를 encoding 하기 위해서는 더 많은 bit가 필요하다는 단점이 있다.

3. 설계 세부사항

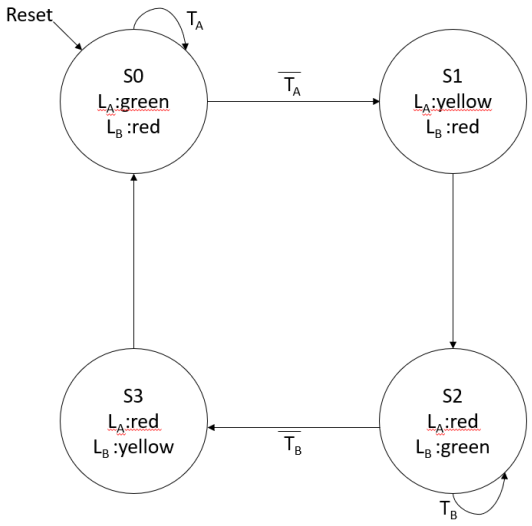
- tl_cntr



신호등 L_A는 Academic Ave의 차량 통행을 제어하는 신호등이고, L_B는 Bravado Blvd의 차량 통행을 제어하는 신호등이다. 차량이 거리에 있다면 신호등은 초록색이 되며, 없으면 빨간색이 된다. 차량이 있음을 감지하기 위해서 traffic sensor를 각각의 거리에 설치하고 T_A, T_B라고 한다. Traffic light은 교통이 없다면 초록색, 노란색, 빨간색을 차례로 거치며, 한쪽 traffic light가 초록색이나 노란색이면 다른 traffic light는 빨간색이다.

(1) Drawing the finite state diagram

State : S0, S1, S2, S3, Input : T_A, T_B output : L_A, L_B



(2) Encoding states

- Encode states

State	Encode

S0	00
S1	01
S2	10
S3	11

- 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

	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$D_1=Q_1 \oplus Q_0$

	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$D_0=\overline{Q_1}\ \overline{Q_0}\ \overline{T_A} + Q_1\overline{Q_0}\ \overline{T_B}$

- output table

Current state		**Outputs**			
Q_1	**Q_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

	0	1
0	0	1
1	0	0

$L_{A0}=\overline{Q_1}Q_0$

	0	1
0	0	0
1	1	1

$L_{A1}=Q_1$

	0	1
0	0	0
1	0	1

$L_{B0}=Q_1Q_0$

	0	1
0	1	1
1	0	0

$L_{B1}=\overline{Q_1}$

- 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 $D_1=Q_1 \oplus Q_0$ $D_0=\overline{Q_1}\,\overline{Q_0}\,\overline{T_A}+Q_1\overline{Q_0}\,\overline{T_B}$ 식을 기반으로 함
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 $L_{A1}=Q_1$ $L_{A0}=\overline{Q_1}Q_0$ $L_{B1}=\overline{Q_1}$ $L_{B0}=Q_1Q_0$ 식을 기반으로 함

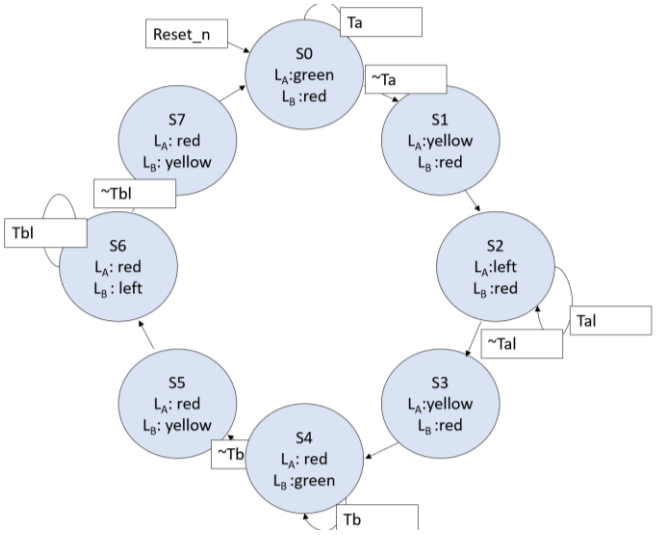
- tl_cntr_w_left

지난 신호등에 좌회전 신호를 추가하여 구현한다. 신호등 L_A는 Academic Ave의 차량 통행을 제어하는 신호등이고, L_B는 Bravado Blvd의 차량 통행을 제어하는 신호등이다. 차량이 거리에 있다면 신호등은 초록색 또는 좌회전 신호가 되며, 없으면 빨간색이 된다. 차량이 있음을 감지하기 위해서 traffic sensor을 각각의 거리에 설치하고 T_A, T_B라고 명명한다. T_{AL}과 T_{BL}은 좌회전 신호에 대해서 차량의 유무를 감지하는 역할을 한다.

Traffic light는 교통이 없을 때 초록색, 노란색을 거쳐 자회전으로 변한다. 교통이 없을 경우 좌회전에서 노란색을 거쳐 빨간색으로 변한다. 좌회전하는 교통이 없어도 초록색에서 좌회전 신호로 우선 변해야 한다. 한 쪽 traffic light가 초록색, 노란색, 좌회전이라면 반대 신호는 빨간색이어야 한다.

(1) Drawing the finite state diagram

State : S0, S1, S2, S3, S4, S5, S6, S7 Input : T_A, T_B,T_{AL}, T_{BL} output : L_A, L_B



(2) Encoding states

- Encode states

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

- state transition table

Current State	Input	Next State

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

	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
010	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
110	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
100	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1

$D_0 = Q2' * Q1' * Q0' * Ta' + Q2' * Q1 * Q0' * Tal' + Q2 * Q1' *Q0 '* Tb' + Q2 * Q1 * Q0' * Tbl'$

	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
001	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
010	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
110	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$D_1 = Q2 '* Q1' * Q0 + Q1 * Q0' + Q2 * Q1' * Q0$

	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
011	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
010	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

110	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
101	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

D_2 = Q2′ * Q1 * Q0 + Q2 * Q1′ + Q2 * Q1 * Q0′

- output table

Current State			**Outputs**			
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

	00	01	11	10
0	0	1	1	0
1	1	1	1	1

L_{A0} = Q_0 + Q_2

	00	01	11	10
0	0	0	0	1
1	1	1	1	1

L_{A1}= Q_1* $\overline{Q_0}$+ Q_2

	00	01	11	10
0	1	1	1	1
1	0	1	1	0

L_{B0} = Q_0 + $\overline{Q_2}$

	00	01	11	10
0	1	1	1	1
1	0	0	0	1

L_{B1} = Q_1* $\overline{Q_0}$ + $\overline{Q_2}$

- module description

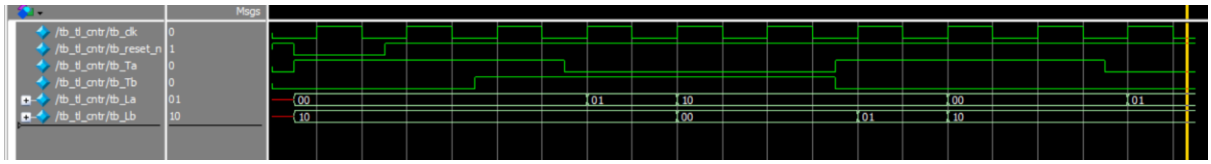
구분	**이름**	**설명**

Top module	tl_cntr_w_left	Traffic light controller의 top module
Sub module	ns_logic	Traffic light controller의 next state를 결정하는 combinational logic D_0 = Q2′ * Q1′ * Q0′ * Ta′ + Q2′ * Q1 * Q0′ * Tal′ + Q2 * Q1′ *Q0 ′* Tb′ + Q2 * Q1 * Q0′ * Tbl′ D_1 = Q2 ′* Q1′ * Q0 + Q1 * Q0′ + Q2 * Q1′ * Q0 D2 = Q2′ * Q1 * Q0 + Q2 * Q1′ + Q2 * Q1 * Q0′식을 기반으로 함
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 $L_{A0} = Q_0 + Q_2$ $L_{A1} = Q_1 * \overline{Q_0} + Q_2$ $L_{B0} = Q_0 + \overline{Q_2}$ $L_{B1} = Q_1 * \overline{Q_0} + \overline{Q_2}$ 위 식에 따라 각 LOGIC을 설계

4. 설계 검증 및 실험 결과

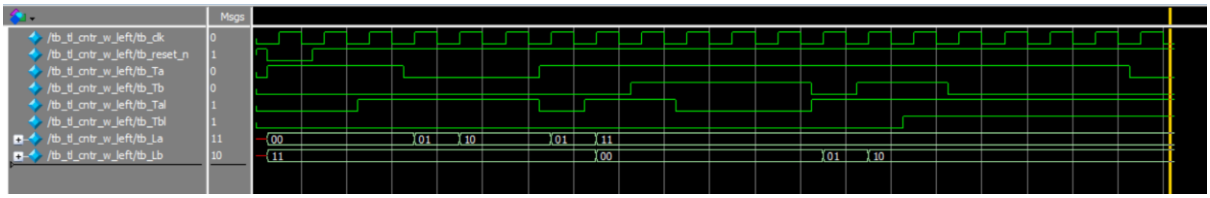
A. 시뮬레이션 결과

- tl_cntr



La가 노란색이거나 초록색이면 Lb가 빨간색이고, Lb가 노란색이거나 초록색이면, La는 빨간색인 것을 확인할 수 있다.

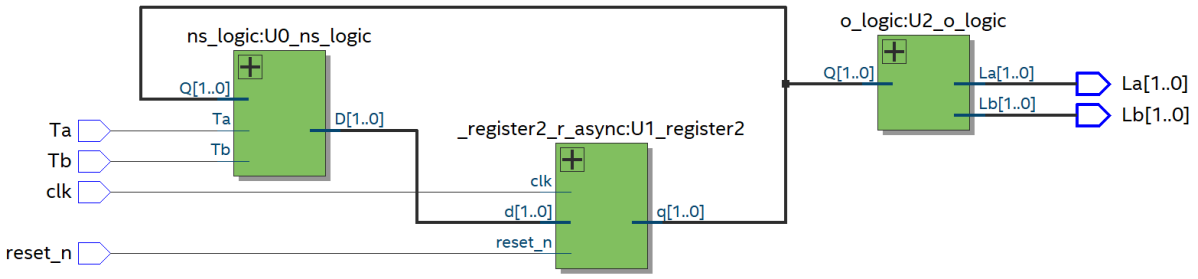
- tl_cntr_w_left



La가 좌회전이고, 노란색이거나 초록색이면 Lb가 빨간색이고, Lb가 좌회전이고, 노란색이거나 초록색이면, La는 빨간색인 것을 확인할 수 있다.

B. 합성(synthesis) 결과

- tl_cntr



tl_cntr의 rtl viewer로, ns_logic, _register2_r_async, o_logic이 연결되어 설계된 것을 확인할 수 있다.

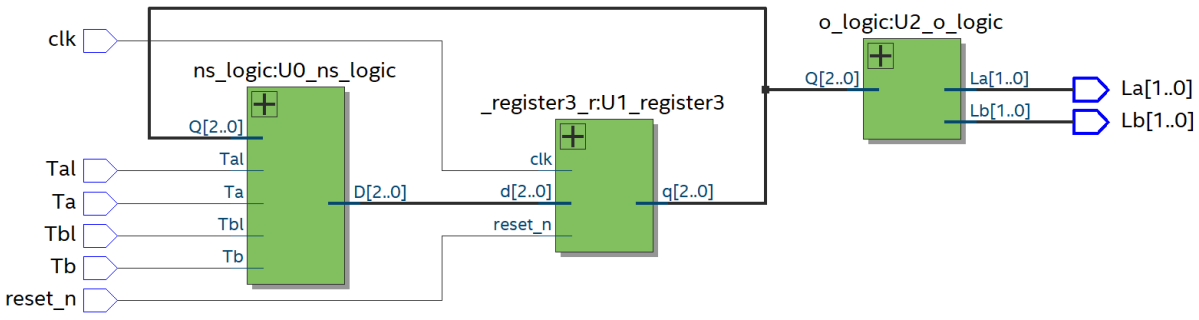
Flow Summary

<<Filter>>

Flow Status	Successful - Mon Oct 17 17:51:33 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)	N/A
Total registers	2
Total pins	8
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

tl_cntr의 flow summary이다. 총 2개의 register와 8개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- tl_cntr_w_left



tl_cntr_w_left의 rtl viewer로, ns_logic, _register3_r, o_logic이 연결되어 설계된 것을 확인할 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Oct 17 17:52:56 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

tl_cntr_w_left의 flow summary이다. 총 3개의 register와 10개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

5. 고찰 및 결론

A. 고찰

Sequential logic 설계 단계를 1-1부터 5까지의 단계를 따라서 하나씩 직접 설계하고 그리다 보니 어떻게 해야 할지 감을 잡을 수 있었다. 시간이 다른 실습보다 오래 걸리기는 했지만 하나씩 해결해 나가니, 시간이 아깝지 않았고 성취감도 느낄 수 있었다. tl_cntr 실습 부분은 1학기 때 해본 적이 있어 쉽게 설계할 수 있었지만, tl_cntr_w_left는 식이 복잡해 시간이 오래 걸렸다.

또한 1학기 디지털논리회로1 시간에 quine-mccluskey method를 배우지 않아 이번 실습의 ppt에 나와있는 quine-mccluskey method를 이해하는데 시간이 오래 걸렸던 것 같다.

verilog를 이용해 코드를 짜고 설계해 구현해 내는 것이 어렵게만 느껴지다가, 점점 verilog에 익숙해지고, 이론으로만 배웠던 것을 직접 구현해 보니 그때 미처 이해하지 못했던 부분이 이해되는 것 같아 의미 있었던 것 같다.

B. 결론

1학기 디지털논리회로1 시간에 FSM, 그 중 Moore FSM과 Mealy FSM에 대해 배웠었다. 이번 실습을 통해 이론으로만 배우고 이해했던 내용들을 복습할 수 있었다. 또한 직접 코드를 작성하고 Traffic Light Controller을 구현해보니 FSM의 원리에 대해 보다 잘 이해하고 알게 된 것 같다. tl_cntr

실습은 1학기 디지털논리회로1 시간에 과제로 진행한 경험이 있어, 쉽게 설계하고 코드를 작성할 수 있었다. 하지만 tl_cntr_w_left, 좌회전 신호가 하나 추가됐을 뿐인데 설계단계가 처음 실습보다 더 복잡했던 것 같다. 하나하나 직접 설계해서 구현해야 하다 보니 시간이 많이 걸렸지만, 설계를 제대로 하고 코드를 작성한다면 시간도 절약할 수 있고, 더 나아가 복잡한 Moore FSM도 구현할 수 있을 것 같다.

6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교 컴퓨터 공학과 2022 강의자료

공영호 교수님/컴퓨터공학기초실험2/광운대학교/2022 강의자료