컴퓨터 공학 기초 실험2 보고서

실험제목: Traffic Light Controller with/without Left Turn Signals

실험일자: 2023년 10월 09일 (월)

제출일자: 2023년 10월 15일 (일)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 2022202065

성 명: 박나림

1. 제목 및 목적

A. 제목

Traffic Light Controller with/without Left Turn Signals

B. 목적

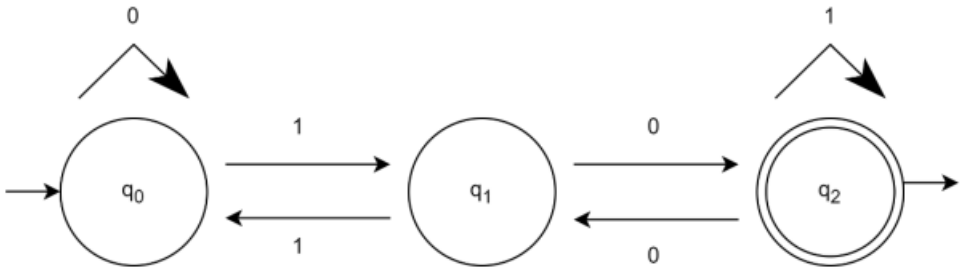
Moore FSM(Finite State Machine)의 기법을 공부하여 이해한다. 이를 이용하여 Traffic light controller를 베릴로그로 설계하도록 한다. 추가로 Left turn signal이 들어간 회로도 설계할 수 있도록 한다.

2. 원리(배경지식)

A. FSM (Finite State Machine)

유한 상태 기계라고도 하는 FSM은 주어진 시간안에서 유한한 수의 상태들 중 하나에만 있을 수 있는 추상적 기계이다. 즉, 일부 입력에 대한 응답으로 한 상태에서 다른 상태로 변경될 수 있다는 뜻이다. 이러한 상태 변경을 전환이라고 한다. 상태 목록과 초기 상태 및 각 전환들을 트리거 하는 입력으로 정의되는데, 여기에는 deterministic finite-state machines (결정적 유한 상태 기계)와 non- deterministic finite-state machines (비결정적 유한 상태 기계)가 있다. 이것들은 주로 오토마타 이론 쪽에서 연구된다. FSM의 설계 방법으로는, 상태 선도를 그리는 방법이 있다. State diagram, 상태 선도를 먼저 그리고 나서 state table, 다음 상태표를 만들고, 각 flip-flop의 출력 Q들의 변이에 따른 입력 값을 추가시킨다. 그 후 각 flip-flop입력들의 논리식을 구하고 그에 따른 회로를 구성하는 것으로 FSM을 설계할 수 있게 된다.

상태선도의 예시는 다음과 같다.

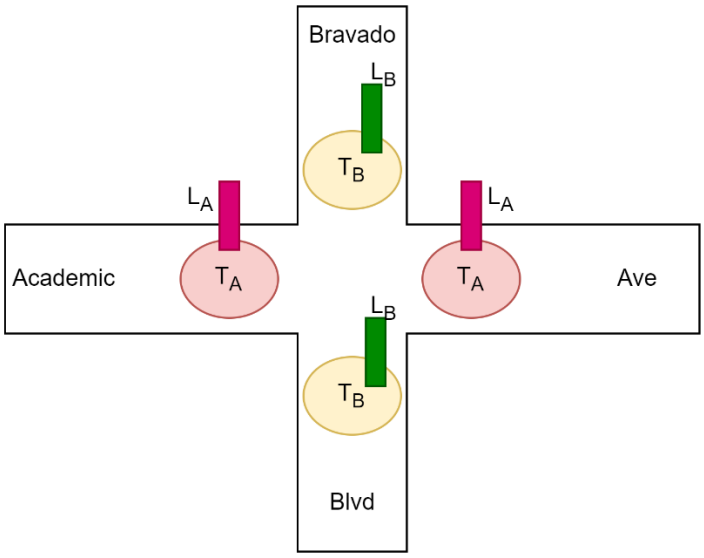


출력되는 것은 두 원으로 표시하고, 반복하여 실행되는 것은 화살표로 나타내게 된다.

B. Traffic light controller

주로 교통과 보행자의 원활하고 안전한 이동을 위해 신호등을 제어 및 조정하는 것을 말한다. 예를 들어, 녹색 단계에서는 모든 교통이 교차로를 통과하는 우선권을 가진다. 녹색, 황색, 붉은색같이 색의 필터 단계를 통해 교통을 조정하며, 운전자가 녹색 신호등의 진행 물결을 접할 수 있도록 교통 시스템의 신호 배치를 조정하기도 한다. 이러한 센서에서는 1상태가 되면 차량이 존재한다는 것을 뜻한다.

이번 실습에서는 이러한 신호등을 제어하는 logic을 구현하게 되며, 신호등은 L_A와 L_B가 있다. 각각 'Academic Ave', 'Bravado Blvd'의 차량통행을 제어하며, 시간에 따라 변하는 게 아니라 차량이 있을 때만 초록색이 되는 방식이다. 차량이 있음을 감지하는 센서로 각각 T_A와 T_B를 추가한다.



C. Traffic light controller with Left turn signals

위에서 한 기본 신호등에 좌회전 신호까지 추가한 신호등이다. 차량이 있음을 감지하는 신호인 T_A와 T_B에 좌회전에 대한 차량을 감지하는 T_{AL}과 T_{BL}이 추가된다.

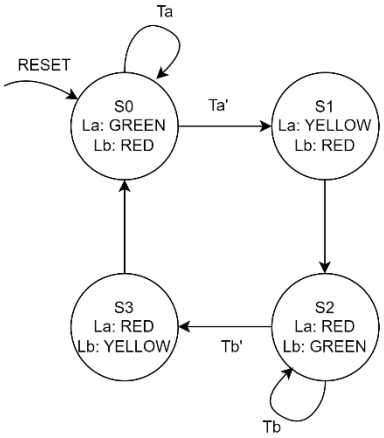
Traffic light는 교통이 없을 때 초록색, 노란색, 좌회전, 노란색, 빨간색의 단계를 거쳐가며 변화한다. 또한 좌회전하는 교통이 없더라도 좌회전 신호로 변화는 과정은 거쳐야 한다. 한쪽 신호가 초록색, 노란색, 좌회전일 동안에 반대쪽 신호는 반드시 빨간색을 유지하고 있어야 한다.

3. 설계 세부사항

3-1. Traffic Light Controller

A. Drawing the finite state diagram

1) Define states: S0, S1, S2, S3
2) Define inputs: Ta, Tb
3) Define outputs: La, Lb
4) Draw the diagram



-state transition table

Current State	Input		Next State
Q	T_A	T_B	D
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Current State		Input		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

$S'_1 = S_0 \oplus S_1$

$S'_0 = \overline{S_1 S_0 T_A} + S_1 \overline{S_0 T_B}$

-output table

Current State	Output	
	La	Lb
S0	GREEN	RED
S1	YELLOW	RED
S2	RED	GREEN
S3	RED	YELLOW

S_1	S_0	L_{a1}	L_{a2}	L_{b1}	L_{b2}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

$L_{a1} = s_1$

$L_{a2} = \overline{s_1} s_0$

$L_{b1} = \overline{s_1}$

$L_{b2} = s_1 s_0$

B. Encoding states

state	Encoding
S0	00
S1	01
S2	10
S3	11

C. Coding the module header

Clk, reset_n, Ta, Tb, La, Lb를 인자로 받아 I/O를 설정하고 코드의 가독성을 높이기 위해 parameter를 사용하여 state와 신호등 색을 설정한다.

D. Coding state registers(filp-flop) – sequential circuits

Flop-flop은 반드시 따로 always구문을 설정해야 한다. 이때 state와 next_state에 대해 2bits reg타입 변수로 선언하여 사용한다. 비동기 회로이므로 always구문 조건은 posedge clk과 negedge reset_n으로 한다. 만약 리셋이 활성화되면 state는 S0으로 초기화되며, 그 외에는 next_state로 변경시킨다. 이때 구문은 nonblocking(<=)으로 한다.

E. Coding combinational circuits

Combinational circuit part에서는 각각 always로 state tarnsiton condition과 state output generation을 작성한다. 이는 앞서 작성한 table의 값들을 그대로 사용하여 설계한다. casex문을 활용하며, 예외처리도 해주도록 한다.

3-2. Structural Traffic Light Controller

1) tl_cntr_struct: Traffic light controller의 top module로, 아래 sub module들을 각각 instance한다.
2) ns_logic: Traffic light controller의 next state를 결정하는 combinational logic로, 회로도에 맞춰서 gate들을 instance하여 설계한다.
3) _register2_r_async: 2-bit resettable register with active low asynchronous reset

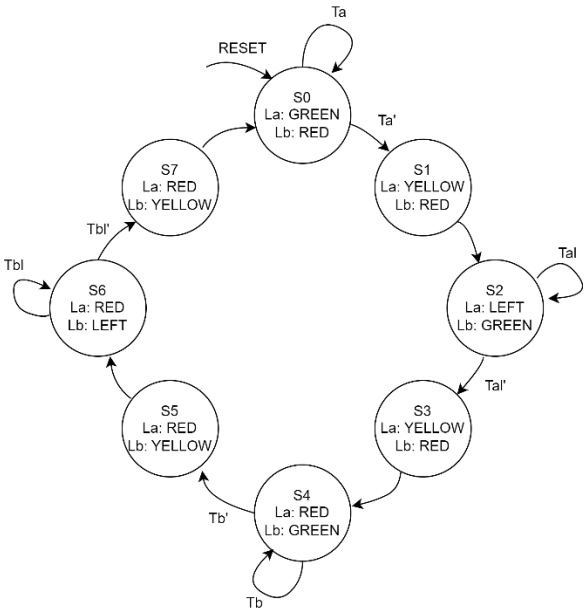
module로, 내부에 _dff_r_async를 2개 instance하여 2-bit을 저장할 수 있도록 한다.

4) _dff_r_async: Resettable D flip-flop with active low asynchronous reset이다.
5) o_logic: 현재 state값에 따라 output값을 결정하는 combinational logic으로, 회로도에 맞춰서 gate들을 instance하여 설계한다.

3-3. Traffic Light Controller with Left Turn signals

A. Drawing the finite state diagram

1) Define states: S0, S1, S2, S3, S4, S5, S6, S7
2) Define inputs: Ta, Tal, Tb, Tbl
3) Define outputs: La, Lb
4) Draw the diagram



-state transition table

Current State	Input				Next State
Q	T_A	T_{AL}	T_B	T_{BL}	D
S0	0	X	0	0	S1
S0	1	X	X	X	S0
S1	X	X	X	X	S2
S2	X	0	X	X	S3
S2	X	1	X	X	S2
S3	X	X	X	X	S4
S4	X	X	0	X	S5

S4	X	X	1	X	S4
S5	X	X	X	X	S6
S6	X	X	X	0	S7
S6	X	X	X	1	S6
S7	X	X	X	X	S0

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	0	0	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

$D_2 = \bar{Q}_2 Q_1 Q_0 + Q_2\overline{Q_1 Q_0 T_B} + Q_2\overline{Q_1 Q_0}T_B + Q_2\overline{Q_1}Q_0 + Q_2 Q_1\overline{Q_0 T_{BL}} + Q_2 Q_1 \bar{Q}_0 T_{BL}$

$= \overline{Q_2} Q_1 Q_0 + Q_2\overline{Q_1} + Q_2 Q_1\overline{Q_0}$

$D_1 = \overline{Q_2 Q_1} Q_0 + \overline{Q_2} Q_1\overline{Q_0 T_{AL}} + \overline{Q_2} Q_1\overline{Q_0} T_{AL} + Q_2\overline{Q_1} Q_0 + Q_2 Q_1\overline{Q_0 T_{BL}} + Q_2 Q_1\overline{Q_0} T_{BL}$

$= \overline{Q_2 Q_1} Q_0 + Q_1\overline{Q_0} + Q_2\overline{Q_1} Q_0$

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

-output table

Current State	Output	
	La	Lb
S0	GREEN	RED
S1	YELLOW	RED
S2	LEFT	RED
S3	YELLOW	RED
S4	RED	GREEN
S5	RED	YELLOW
S6	RED	LEFT
S7	RED	YELLOW

S_2	S_1	S_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

$L_{a1} = \overline{Q_2}Q_1\overline{Q_0} + Q_2\overline{Q_1Q_0} + Q_2\overline{Q_1}Q_0 + Q_2Q_1\overline{Q_0} + Q_2Q_1Q_0 = Q_1\overline{Q_0} + Q_2$

$L_{a0} = \overline{Q_2Q_1}Q_0 + \overline{Q_2}Q_1Q_0 + Q_2\overline{Q_1Q_0} + Q_2\overline{Q_1}Q_0 + Q_2Q_1\overline{Q_0} + Q_2Q_1Q_0 = Q_0 + Q_2$

$L_{b1} = \overline{Q_2Q_1Q_0} + \overline{Q_2Q_1}Q_0 + Q_2Q_1\overline{Q_0} + \overline{Q_2}Q_1Q_0 + Q_2Q_1\overline{Q_0} = \overline{Q_2} + Q_1\overline{Q_0}$

$L_{b0} = \overline{Q_2Q_1Q_0} + \overline{Q_2Q_1}Q_0 + \overline{Q_2}Q_1\overline{Q_0} + \overline{Q_2}Q_1Q_0 + Q_2\overline{Q_1}Q_0 + Q_2Q_1Q_0 = \overline{Q_2} + Q_0$

B. Encoding states

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

C. Coding the module header

Clk, reset_n, Ta, Tal, Tb, Tbl, La, Lb를 인자로 받아 I/O를 설정하고 코드의 가독성을 높이기 위해 parameter를 사용하여 state와 신호등 색을 설정한다.

D. Coding state registers(flip-flop)-sequential circuits

앞서 설계한 내용과 비슷하지만, 이번엔 state가 3-bits로 표현된다는 점이 바뀐다.

E. Coding combinational circuits

위 설계 내용과 같은 방식으로 table의 값들을 이용하여 진행한다.

3-4. Structural Traffic Light Controller with Left Turn signals

1) tl_cntr_w_left_struct: Traffic light controller의 top module로, 아래 sub module들을 각각 instance한다.
2) ns_logic: Traffic light controller의 next state를 결정하는 combinational logic로, 회로도에 맞춰서 gate들을 instance하여 설계한다.
3) _register3_r: 3-bit resettable register with active low asynchronous reset module로, 내부에 _dff_r_async를 3개 instance하여 3-bit을 저장할 수 있도록 한다.

4) _dff_r_async: Resettable D flip-flop with active low asynchronous reset이다.

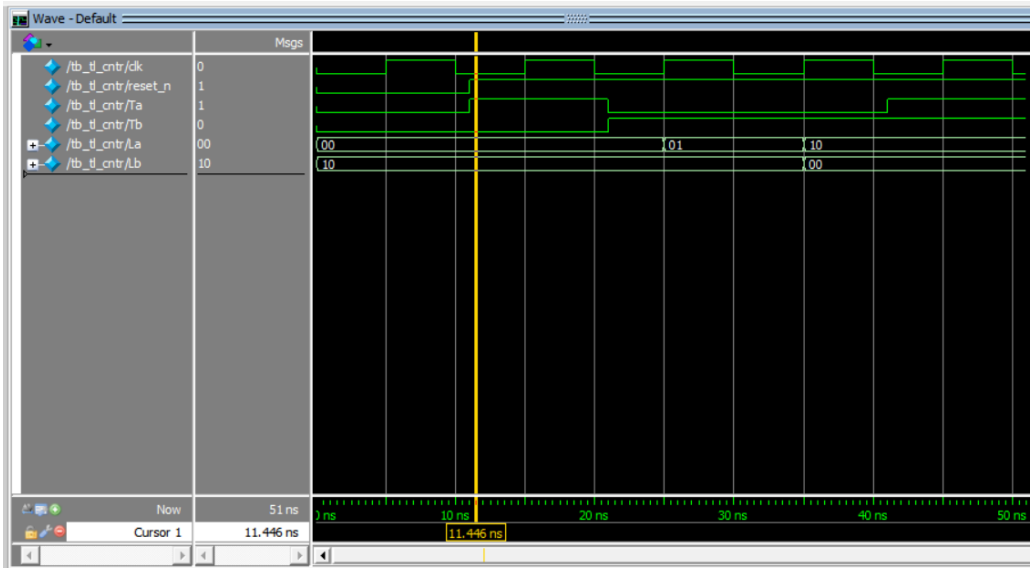
5) o_logic: 현재 state값에 따라 output값을 결정하는 combinational logic으로, 회로도에 맞춰서 gate들을 instance하여 설계한다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

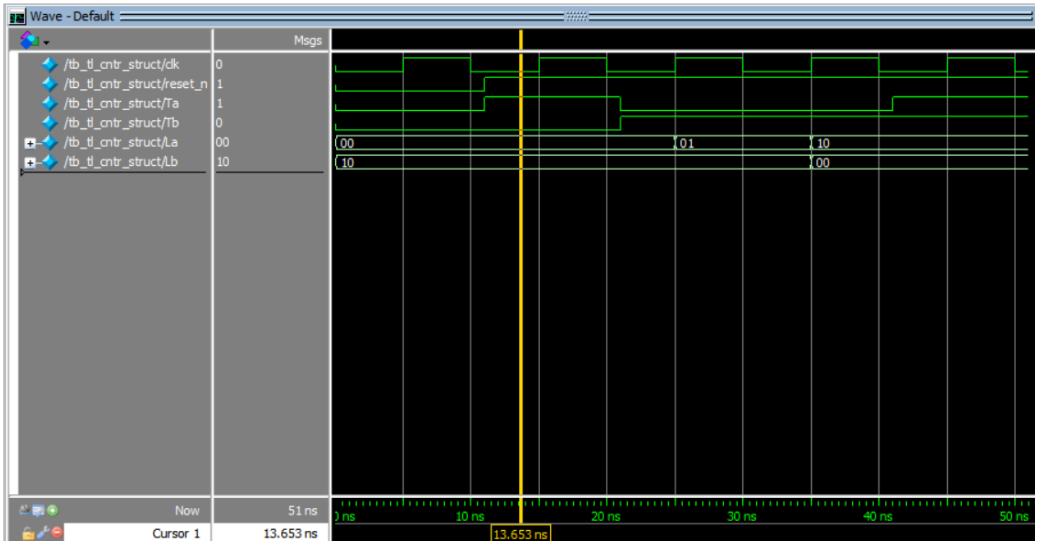
1) tl_cntr

처음에 reset이 0인 동안에는 비동기식 회로이므로 clock에 관계없이 무조건 S0의 상태가 된다. (La: GREEN(00), Lb: RED(10)) 그 다음 reset이 비활성화 되면서 clock이 상승하는 순간 Ta와 Tb는 S0의 상태를 유지하므로 결과 값은 동일하다. 그리고 Ta가 비활성화 되고 Tb가 켜지면 우선 La는 YELLW(01), Lb는 RED(10)이 되고, 다음 clock cycle에서 온전히 La가 RED(10), Lb가 GREEN(00)이 된다.



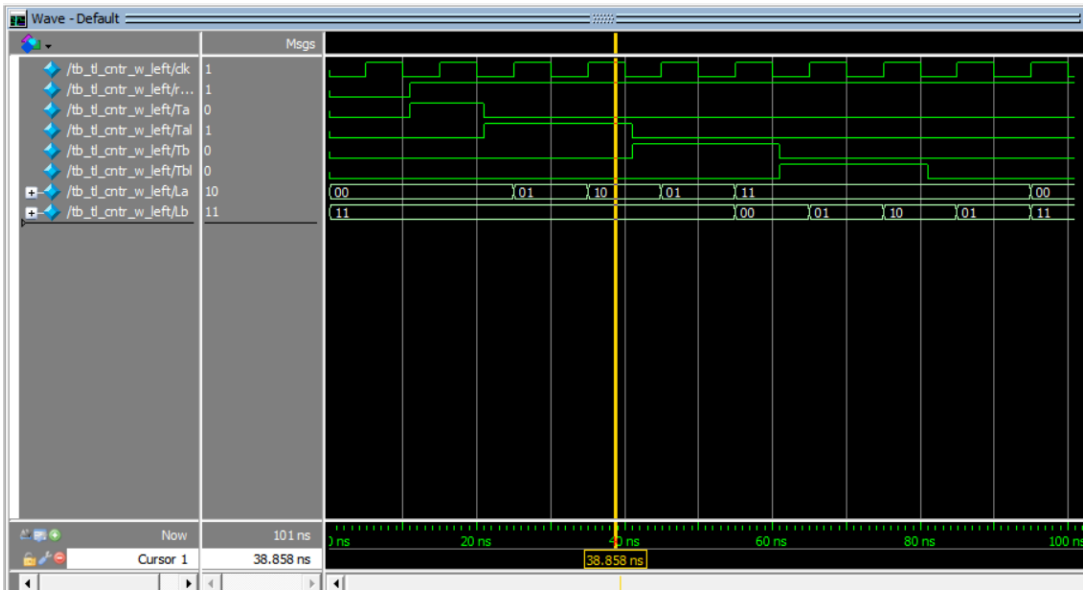
2) tl_cntr_struct

tl_cntr과 구조적으로는 다르게 설계되지만 그 결과는 동일하게 나오는 걸 확인할 수 있다.



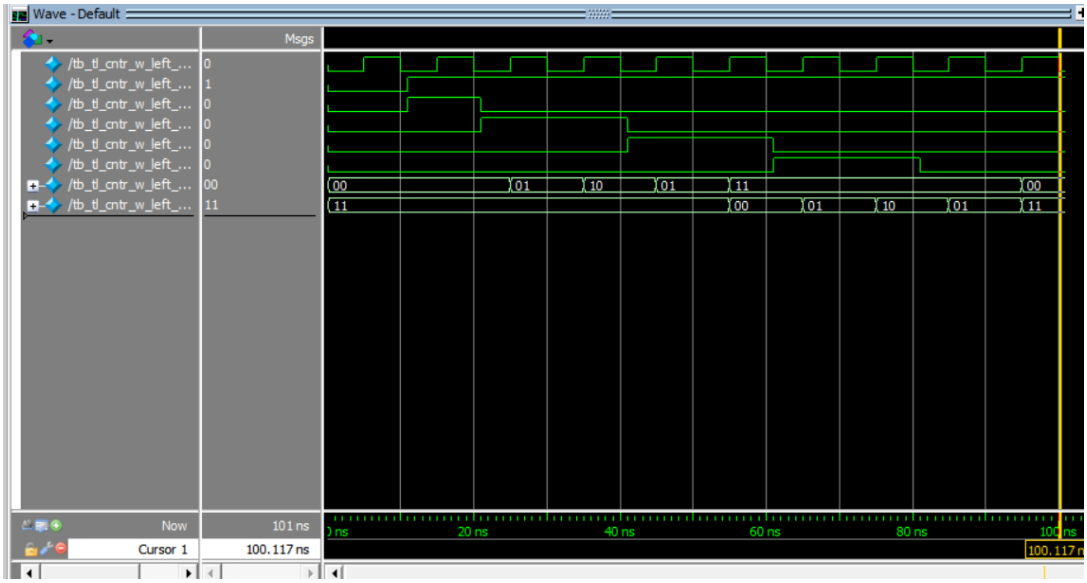
3) tl_cntr_w_left

처음 reset이 활성화되면 S0상태가 되고, 이후 모든 state에 대해 차례대로 테스트한 결과이다. Reset이 비활성화 된 후 첫번째로는 Ta가 켜진 S0이므로 결과는 동일하게 나오다가 S2로 Tal이 켜지면 S1으로 La가 YELLOW(01)를 거친 다음 S2로 LEFT(10)가 된다. 그 다음 Tb가 켜지면 S3로 La가 YELLOW(01)를 거친 다음 S4로 RED(11)가 되면서 Lb가 GREEN(00)이 된다. 그 후에 Tbl이 켜지면 같은 방식으로 동작하여 순서대로 YELLOW(01), LEFT(10), YELLOW(01), RED(11)로 변하는 것을 볼 수 있다.



4) tl_cntr_w_left_struct

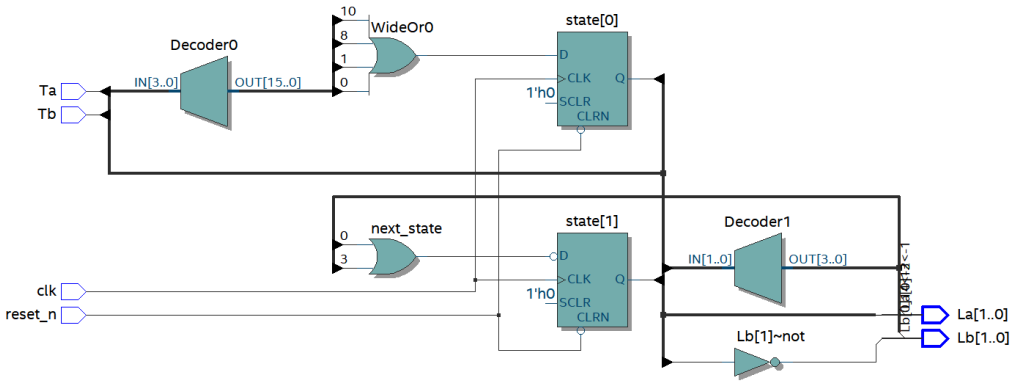
tl_cntr_w_left_struct와 구조적으로는 다르게 설계되지만 그 결과는 동일하게 나오는 걸 확인할 수 있다.



B. 합성(synthesis) 결과

1) tl_cntr

tl_cntr의 RTL viewer와 Flow summary이다. 회로의 모습과 크기를 확인할 수 있다.



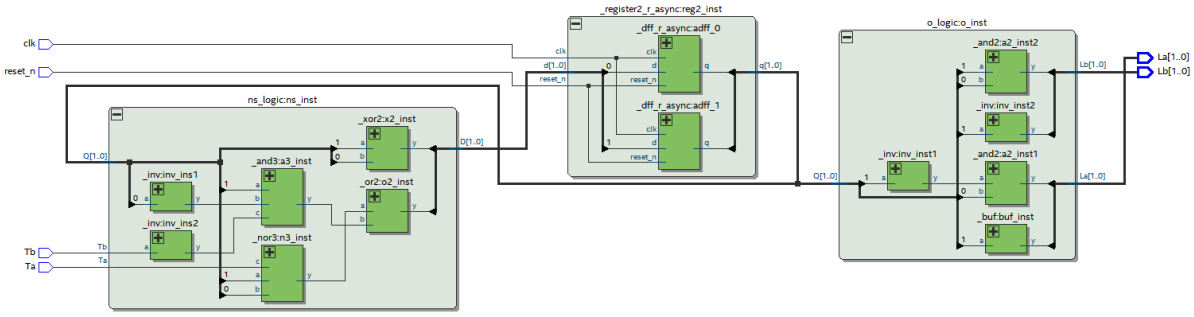
Flow Summary

<<Filter>>

Flow Status	Successful - Sat Oct 14 14:22:04 2023
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 %)

2) tl_cntr_struct

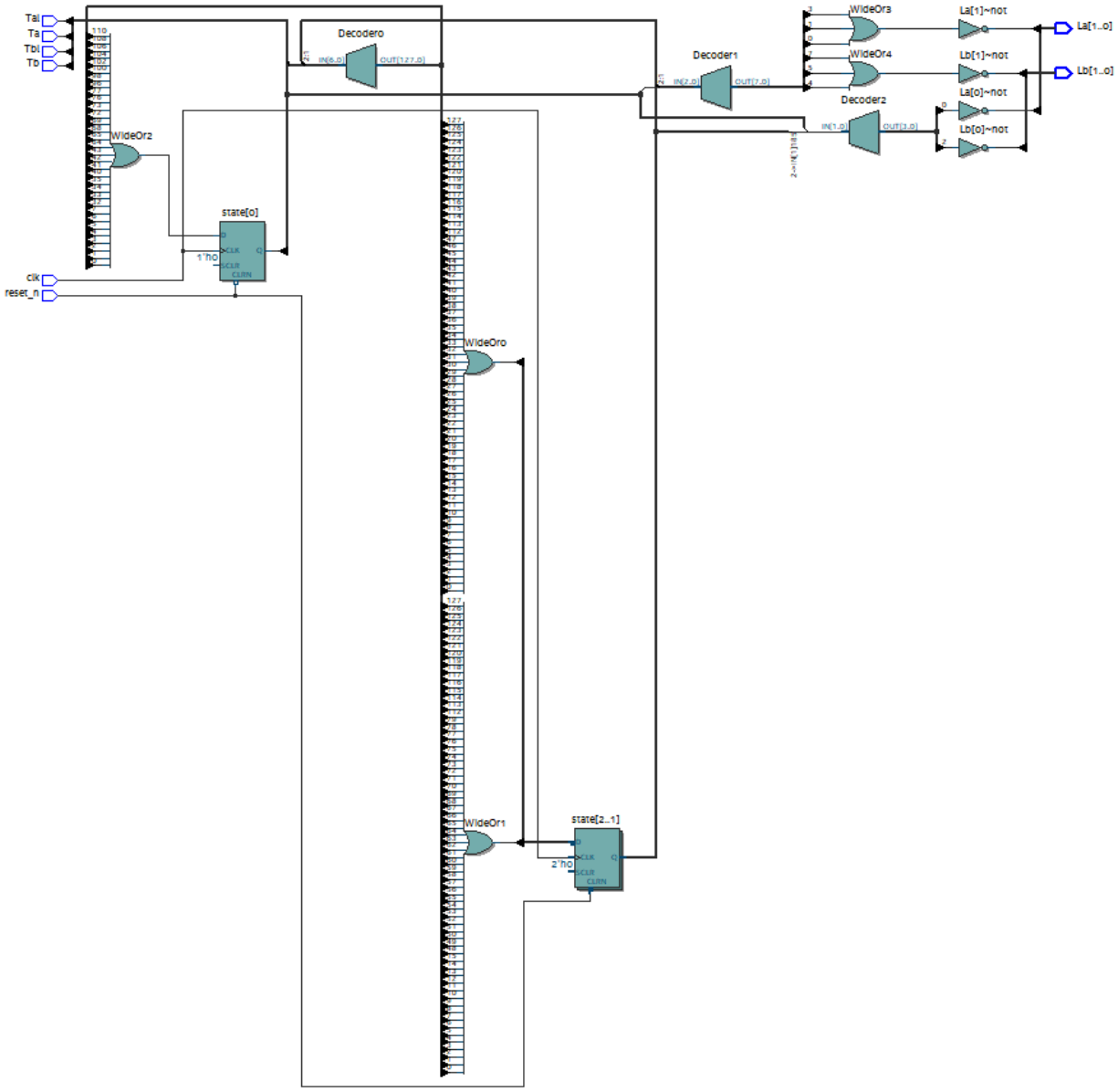
tl_cntr_struct의 RTL viewer와 Flow summary이다. tl_cntr과 구조적으로 회로도는 다르지만 회로의 크기는 동일함을 확인할 수 있다.



Flow Summary	
<<Filter>>	
Flow Status	Successful - Sat Oct 14 22:02:51 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	tl_cntr_struct
Top-level Entity Name	tl_cntr_struct
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 %)

3) tl_cntr_w_left

tl_cntr_w_left의 RTL viewer와 Flow summary이다. 회로의 모습과 크기를 확인할 수 있다. 좌회전 신호가 들어가지 않은 기본 신호등 회로보다 회로의 크기가 더 큰 것을 알 수 있다.



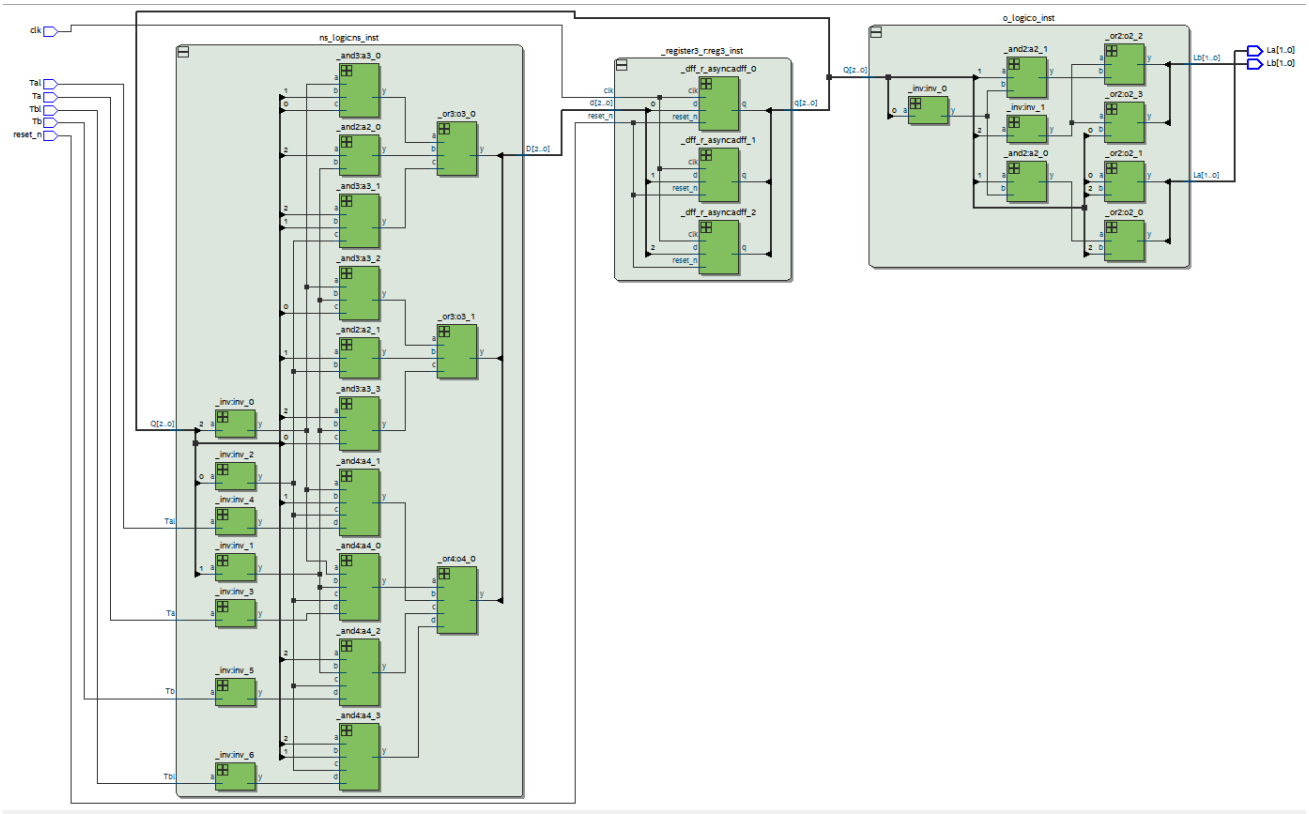
Flow Summary

<<Filter>>

Flow Status	Successful - Sat Oct 14 17:54:21 2023
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)	5 / 41,910 (< 1 %)
Total registers	4
Total pins	10 / 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 %)

4) tl_cntr_w_left_struct

tl_cntr_w_left_struct의 RTL viewer와 Flow summary이다. tl_cntr_w_left과 구조적으로 회로도는 다르지만 회로의 크기는 동일함을 확인할 수 있다.



Flow Summary	
<<Filter>>	
Flow Status	Successful - Sun Oct 15 00:15:23 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	tl_cntr_w_left_struct
Top-level Entity Name	tl_cntr_w_left_struct
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	5 / 41,910 (< 1 %)
Total registers	4
Total pins	10 / 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 %)

5. 고찰 및 결론

A. 고찰

처음에 module들을 설계할 때, tl_cntr과 tl_cntr_struct의 차이점이 구분되지 않아 단순히 tl_cntr의 역할들을 여러 module로 분리하여 설계하는 식으로 하였다. 그러다 문제를 다시 파악하고 회로도에 맞게 gate들을 이용하여 구조적으로 설계해야 한다는 점을 이해하고 다시 작성하였다.

tl_cntr_struct를 설계할 때, 회로도를 바탕으로 gate들을 instance하여 만들었다. 여기서 ns_logic부분에서 3-to-1 NOR gate와 등가인 것을 잘못 해석하여 NAND gate로 설계하였다가 문법 오류는 없지만 웨이브폼이 잘못 나오는 상황이 발생하였다. 다시 gate들을 확인해서 수정하니 제대로 된 값이 나올 수 있었다.

tl_cntr_w_left_struct를 설계하면서 마찬가지로 웨이브폼이 잘못 나왔었는데, 이 역시 다시 코드를 살펴보니 wire를 선언한 이름과 실제 사용한 wire의 이름이 서로 달라서 값이 제대로 전달되지 않아 잘못 나왔던 것이었다. 수정하고 다시 실행해보니 정상적으로 값이 나올 수 있었다.

B. 결론

이러한 fsm 회로를 설계하는 과정에서 직접 테스트벤치를 실행해보면서, Ta에서 Tb로 바로 전환하여도 clock cycle이 도는 동안 중간 과정인 Yellow상태를 반드시 거쳐가는 것을 확인할 수 있었다. 이는 바로 바뀌지 않고 clock이 상승하는 순간에 Yellow로 되었다가, 다음 clock이 상승하는 부분에서 Red로 바뀐다는 것이다. 또한 이런 식으로 바뀌기 위해서는 최소 clock이 2번 이상 주기가 바뀌어야 하니, 테스트벤치에서 시간을 할당할 때 여유를 두고 값을 설정해야 올바른 결과값이 나온다는 점도 깨달았다. 이후 다른 신호를 추가하여 state 수가 늘어나도 같은 방식으로 설계하면 될 것 같다는 생각이 들었다.

또한 struct로 구현하는 것과 behavioral하게 구현하는 것에 대한 차이점을 알 수 있었다. RTL viewer를 보면 struct로 할 땐 단순히 gate들이 연결된 모습이지만 behavioral은 decoder가 들어가는 등 전혀 다른 회로의 모습처럼 보이기도 했다. 기본 신호등에선 후자가 더 간단해 보였지만 좌회전 신호가 들어간 신호등에선 오히려 gate로 하는 회로가 더 오류를 확인하기 편했다. 물론 이렇게 확인할 때는 확실히 볼 수 있어 좋지만 직접 설계해보니 state 수가 늘어날수록 struct는 식을 하나하나 다 계산해야 하니 전체적으로 보면 behavioral하게 구현하는 게 더 효율성이 높은 것 같다.

이 둘은 다른 회로처럼 보여도 Flow summary를 보면 회로의 크기가 동일하게 나오고, 같은 테스트벤치로 실행하면 그 결과 값도 같게 나오는 등 둘은 같은 회로를 다른 방식으로 설계했을 뿐이라는 점을 확실히 깨달을 수 있었다.

6. 참고문헌

David Money Harris and Sarah L. Harris / Digital Design and Computer Architecture / Elsevier / 2007