REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 명 : 디지털 시스템
담당교수 : 김종태 교수님
학 과 : 전자전기공학과
학 년 : 3학년
학 번 : 2018312121
이 름 : 장재성
과 제 명 : Assignment3 보고서
제 출 일 : 2022/05/19 (목)

① 설계 목표

FSM(Finite State machine) 중에서 Moore State Machine 을 이용하여 입력이 CLK, Reset, Break, Left, Right 로, 출력이 8-bit 의 LED 로 구성된 Tail Light Controller(자동차 후미등 제어기)를 설계한다. 이 때, 최소 개의 state 를 사용한다. 구현하는 과정에서 State diegram 을 그린다. 구현 후에는 주어진 Test bench 로 시뮬레이션으르 진행하여 검증한다.

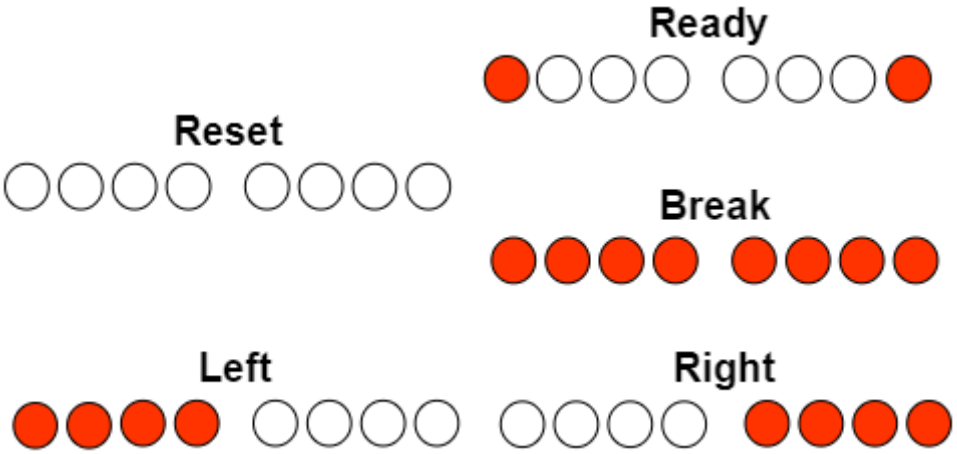
② 이론적 접근

(1) "Tail Light Controller" 작동 방식

Tail Light Controller 의 동작은 크게 다음 4 가지로 구성된다.

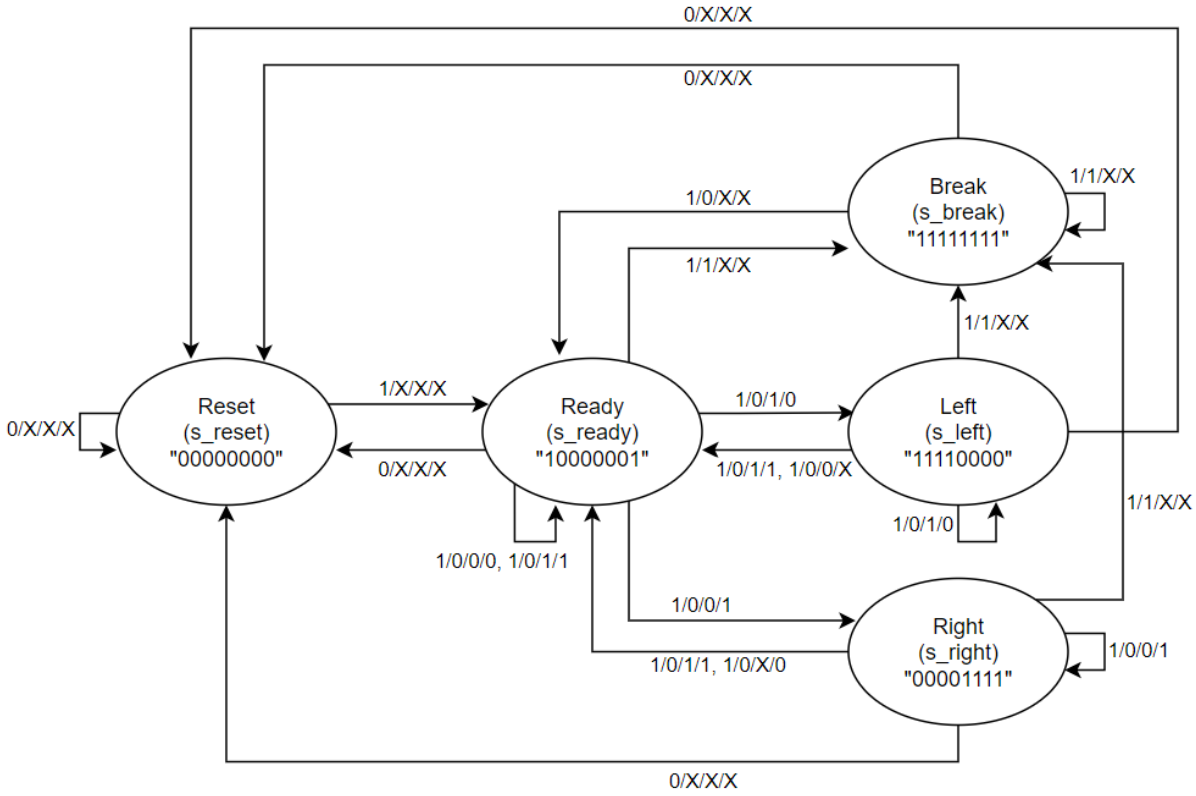
- Reset 동작을 제외한 모든 동작은 CLK 의 Rising Edge 에 맞춰 synchronous 하게 동작한다.
- Reset 은 0 이 입력되는 활성화되는 Active low 로, asynchronous 하게 동작한다.
- Break, Left, Right 는 1 이 입력되면 활성화되는 Active High 로, synchronous 하게 동작한다.
- 입력은 한 번에 여러 개가 동시에 입력될 수 있다. (예를 들어, left 와 right 가 동시에 입력이 가능하다)

그리고 Tail Light 는 현재 어떤 상태인지(동작) (Reset 인지, Ready 인지, Break 인지, Left 인지, Right 인지) LED 를 통해 나타낸다. 각 상태에 따른 LED 표현 방식은 다음과 같다.



(2) Tail Light Controller 무어 상태기계

현재 상태에 의해서만 결과가 결정되는 Moore State machine 의 State diagram 에서 각 상태는 고유의 값을 지니며, State transition 을 나타낼 때, State 와 함께 output(Z)을 표시한다. Tail Light Controller 의 동작을 고려하여 Moore State machine 의 State diagram 을 그리면 총 5 개 State 로 구성할 수 있다. Tail Light Controller 의 Moore State machine State diagram 은 다음과 같다.



각 상태에 대한 설명은 다음과 같다.

Reset(s_reset) : Reset 상태는 어떤 상태에서든 RESET 입력 신호가 0 인 경우 transition 되는 "Initial State"에 해당한다. 즉, Reset 이 일어날 때의 경우를 뜻한다. RESET 상태에 대한 동작은 다음과 같다.

- Reset 은 CLK 에 관계없이(asynchronous) 동작한다.
- Reset 이 활성화되면 RESET 상태에 머무른다.
- Reset 이 비활성화되면 다른 입력 값에 관계없이 Ready(s_ready)상태가 된다. (synchronous)

Reset 상태는 위 동작을 따르며, Moore machine 에서 값은 "00000000"이다. 이는 실제 자동차 후미등 전부가 꺼진 상태와 유사하다.

Ready(s_ready) : Ready 상태는 Reset 상태에서 Reset 입력 신호가 1 인 경우, Break 상태에서 Break 입력 신호가 0 인 경우, Left 상태에서 Reset 신호를 제외한 입력 신호 중 Left 입력 신호만 1 이 아닌 경우, Right 상태에서 Reset 신호를 제외한 입력 신호 중 Right 입력 신호만 1 이 아닌 경우에 transition 되는 상태이다. Ready 상태에서는 어떤 입력을 받느냐에 따라

"Reset", "Break", "Left", "Right" 어떤 상태로든 transition 이 일어날 수 있다. 즉, 어떤 상태 혹은 어떤 동작으로든 바뀔 수 있는 준비가 되어있는 경우를 뜻한다. Ready 상태에 대한 동작은 다음과 같다.

- Break 가 입력되면 Reset 을 제외한 다른 입력 값에 상관없이 Break 상태가 된다.
- Left 만 입력되면 Left 상태가, Right 만 입력되면 Right 상태가 된다.
- 다른 경우에서는 Ready 상태에 머무른다.

Ready 상태는 위 동작을 따르며, Moore machine 에서 값은 "10000001"이다.

Break(s_break) : Break 상태는 Ready 상태, Break 상태, Left 상태, Right 상태에서 Break 신호가 1 인 경우에 transition 되는 상태이다. Break 신호가 계속 1 을 유지한다면, Left, Right 신호의 값에 상관없이 계속 Break 상태를 유지한다. 실제 자동차 후미등의 경우, 자동차의 브레이크를 밟았을 때 나타나는 경우이다. Break 상태에 대한 동작은 다음과 같다.

- Break 가 입력되면 Reset 을 제외한 다른 입력 값에 상관없이 break 상태에 머무른다.
- Break 가 입력되지 않으면 Reset 을 제외한 다른 입력 값에 상관없이 Ready 상태가 된다.

Break 상태는 위 동작을 따르며, Moore machine 에서 값은 "11111111'이고, 이는 실제 자동차 후미등 전부가 켜진 상태와 유사하다.

Left(s_left) : Left 상태는 Ready 상태, Left 상태에서 Reset 신호를 제외하고 나머지 신호 중 Left 신호만 1 인 경우에 transition 되는 상태이다. Left 신호가 계속 1 을 유지한다면, 계속 Left 상태를 유지한다. 실제 자동차 후미등의 경우, 좌회전 신호를 넣었을 때 나타나는 경우이다. Left 상태에 대한 동작은 다음과 같다.

- Break 가 입력되면 reset 을 제외한 다른 입력 값에 상관없이 Break 상태로 간다.
- Left 만 입력되면 Left 상태에 머무른다.
- 다른 경우에서는 Ready 상태로 간다.

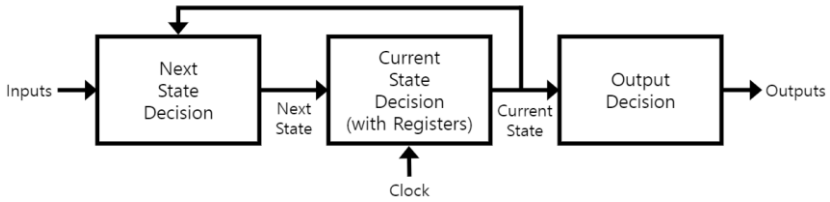
Left 상태는 위 동작을 따르며, Moore machine 에서 값은 "11110000"이고, 이는 실제 자동차 후미등에서 왼쪽 부분만 켜진 상태와 유사하다.

Right(s_right) : Right 상태는 Ready 상태, Right 상태에서 Reset 신호를 제외하고 나머지 신호 중 Right 신호만 1 인 경우에 transition 되는 상태이다. Right 신호가 계속 1 을 유지한다면, 계속 Right 상태를 유지한다. 실제 자동차 후미등의 경우, 우회전 신호를 넣었을 때 나타나는 경우이다. Right 상태에 대한 동작은 다음과 같다.

- Break 가 입력되면 Reset 을 제외한 다른 입력 값에 상관없이 Break 상태로 간다.
- Right 만 입력되면 Right 상태에 머무른다.
- 다른 경우에서는 Ready 상태로 간다.

Right 상태는 위 동작을 따르며, Moore machine 에서 값은 "00001111"이고, 이는 실제 자동차 후미등에서 오른쪽 부분만 켜진 상태와 유사하다.

Moore machine 은 'Next State Decision', 'Current State Decision', 'Output Deicision' 총 3 가지 부분으로 나뉜다. 여기서 'Next State Decision'과 'Output Decision' 부분은 Combinational logic 이고, 'Current State Decision' 부분은 CLK Event 에 동작하는 Sequential logic 이다. 전체적인 구성은 다음과 같다.



VHDL 로 Tail Light Controller 의 Moore machine 구현은 위와 같이 2 개의 Combinational logic circuit 과 1 개의 Sequential logic circuit 을 위에서 그린 Moore machine 의 state diagram 을 참고하여 구현함으로써 이루어진다. 이에 대해서는 추후 'VHDL 구현' 부분에 자세히 설명한다.

③ VHDL 구현(Source Code 설명)

(1) Tail Light Controller 무어 상태기계 VHDL 구현

우선, Input 은 1-bit 의 CLK, RESET, BREAK, LEFT, RIGHT 을 선언하고, Output 은 8-bit 의 LED 를 선언한다. 모든 input, output port 는 'std_logic' type 으로 선언한다.

```
entity TAIL_LIGHT is
    port(
      CLK   : in STD_LOGIC;
      RESET  : in STD_LOGIC;
      BREAK  : in STD_LOGIC;
      LEFT  : in STD_LOGIC;
      RIGHT  : in STD_LOGIC;
      LED : out STD_LOGIC_VECTOR(7 downto 0)
    );
end TAIL_LIGHT;
```

그 후, state 에 대한 type 선언과 state 에 대한 정보를 담을 내부 signal 을 선언해야 한다. Tail Light Controller 의 Moore machine state diagram 에 따르면 state 는 s_reset(RESET / Initial state), s_ready(READY), s_break(BREAK), s_left(LEFT), s_right(RIGHT) 총 5 개로 구성된다. 이 5 개의 state 에 대해 'states'라는 하나의 type 을 선언하고, state 에 대한 정보를 담을 변수를 앞서 선언한 states type 으로 현재 상태를 나타내는 'state'와 다음 상태를 나타내는 'next_state' 두 개 선언한다.

```
-- Declaration architecture
architecture Behavioral of TAIL_LIGHT is
    -- Declaration 5 states for implementing Tail Light Controller using Moore machine
    -- Refer to Tail Light Controller State diagram from 20183121121_JangJaeSeong_HW3.docx
    type states is (s_reset, s_ready, s_break, s_left, s_right);

    -- Declaration of internal signal state and next_state which is used for expressing states.
    signal state, next_state : states;
```

그 후, Moore machine 을 구성하는 3 가지 부분을 구현한다.
첫 번째로, 'Current State Decision' 부분은 Reset 신호가 valid('0')한 경우, State 를 Initial State(s_reset)로 transition 하고, Reset 신호가 invalid('1')한 경우, CLK 에 따라 state 가 다음 state 로 transition 하도록 구현한다. if 문을 이용하여 RESET 이 '0'인 경우(if (RESET = '0')), state 은 s_reset 를, RESET 이 1 이고 CLK 이 rising edge 인 경우(elsif (CLK = '1' and CLK'event)), state 는 next_state 를 할당하도록 설정한다. Process 의 sensitivity list 는 해당 프로세스에서 사용하는 입력이 CLK 과 RESET 이므로, 'process(CLK, RESET)'로 작성한다.

```
process(CLK, RESET)
  begin
    if(RESET = '0') then
      state <= s_reset;
    elsif(CLK = '1' and CLK'event) then
      state <= next_state;
    end if;
end process;
```

두 번째로, 'Output Decision' 부분은 Tail Light Controller Moore State diagram 을 참고한다. Case 구문을 이용하여 모든 state 에 대하여 각 state 인 경우의 출력(LED)에 앞서 그렸던 Moore state diagram 을 참고하여 알맞은 값을 할당하면 된다. 해당 부분은 combinational logic 이고, Mealy machine 은 현재 상태와 입력에 따라 Output 이 결정되므로 always sensitivity 에 CLK 이 들어가지 않고, 다른 입력인 state 만 쓰이면서 구현된다.

```
process(state)
  begin
    case state is
      when s_reset => LED <= "00000000"; -- The value of s_reset is "00000000".
      when s_ready => LED <= "10000001"; -- The value of s_ready is "10000001".
      when s_break => LED <= "11111111"; -- The value of s_break is "11111111".
      when s_left => LED <= "11110000";  -- The value of s_left is "11110000".
      when s_right => LED <= "00001111"; -- The value of s_right is "00001111".
    end case;
end process;
```

세 번째로, 'Next state' 부분은 Tail Light Controller Moore State diagram 을 참고한다. Case 구문을 이용하여 모든 state 에 대하여 각 state 인 경우 다양한 입력에 따른 다음 상태를 앞에서 그렸던 Tail Light Controller Moore state diagram 을 참고하여 알맞은 값을 할당하면 된다. 각 state 마다 RESET 을 제외한 입력 BREAK, LEFT, RIGHT 신호가 0 인지 1 인지에 따라 next_state 의 값이 달라지므로, 각 state 를 서술할 때마다 if 구문을 사용하여 BREAK, LEFT, RIGHT 를 Tail Light Controller 의 동작을 참고하여 0 과 1 의 경우로 나누어 next_state 에 값을 다르게 할당해야 한다. 해당 부분은 combinational logic 이고, Moore machine 은 현재 상태에 따라서 Output 이 결정되므로 always sensitivity 에 CLK 이 들어가지 않고, 다른 입력인 state, BREAK, LEFT, RIGHT 만 쓰이면서 구현된다.

```
process(state, BREAK, LEFT, RIGHT)
  begin
    case state is
      -- s_reset(RESET state) : next state will be s_ready when RESET deactivates and CLK is trigger edge.
      when s_reset =>
        next_state <= s_ready;
when s_ready =>
  if(Break = '1') then next_state <= s_break;
  elsif(LEFT = '1' and RIGHT = '0') then next_state <= s_left;
  elsif(LEFT = '0' and RIGHT = '1') then next_state <= s_right;
  else next_state <= s_ready;
  end if;
when s_break =>
  if(Break = '1') then next_state <= s_break;
  else next_state <= s_ready;
  end if;
when s_left =>
  if(Break = '1') then next_state <= s_break;
  elsif(LEFT = '1' and RIGHT = '0') then next_state <= s_left;
  else next_state <= s_ready;
  end if;
      when s_right =>
        if(Break = '1') then next_state <= s_break;
        elsif(LEFT = '0' and RIGHT = '1') then next_state <= s_right;
        else next_state <= s_ready;
        end if;
    end case;
end process;
```

(각 State 에 대한 설명에 대한 주석은 VHDL 코드에 쓰여있다)

④ 검증 계획 (Test bench code 설명)

Tail Light Controller 검증은 앞서 구현한 entity(TAIL_LIGHT)에 대한 component 선언 후 port out 하고, 입력들(CLK, RESET, BREAK, LEFT, RIGHT)을 지연 시간을 활용하여 알맞은 시간에 알맞은 값으로 할당하여 그에 따른 결과 LED 가 알맞게 출력되는지 확인함으로써 이루어진다. 이에 대한 Test bench code 를 만들어서 검증을 해야 한다. Test Bench Code 는 다음과 같다.

(1) Tail Light Controller 무어 상태기계 Test Bench Code

Test Bench Code 는 외부 I/O signal 을 선언하지 않고, 모두 내부 signal 을 선언하고 값을 설정함으로써 이루어진다. 즉, Test 에 대한 entity 를 선언할 때 I/O port 부분을 작성하지 않는다. 그 후, architecture 부분에 앞서 구현한 Tail Light Controller entity(TAIL_LIGHT)를 대상으로 component 선언을 한다.

```
-- Declaration entity and Port.
-- There is no I/O port in Test Bench Code.
entity TL_tb is
end TL_tb;

-- Declaration architecture
architecture Behavioral of TL_tb is
  -- Declaration of component of Tail Light Controller(TAIL_LIGHT)
  component TAIL_LIGHT is
    port(
    CLK     : in STD_LOGIC;
    RESET   : in STD_LOGIC;
    BREAK   : in STD_LOGIC;
    LEFT    : in STD_LOGIC;
    RIGHT   : in STD_LOGIC;
    LED : out STD_LOGIC_VECTOR(7 downto 0)
  );
  end component;
```

입력에 대한 값을 설정하고 출력에 대한 값을 저장할 내부 signal 을 선언한다. 그 후, 앞서 선언한 Tail Light Controller Moore machine 의 component 인 TAIL_LIGHT 를 해당 component 의 I/O port 와 내부 signal 을 알맞게 연결하여 Port map(Instantiation)한다.

```
-- Declaration of internal signal for setting input signal used for simulation.
-- All signal except LED is declared as std_logic type.
-- LED is declared as 8bit std_logic_vector type.
signal CLK   : STD_LOGIC;
signal RESET  : STD_LOGIC;
signal BREAK  : STD_LOGIC;
signal LEFT  : STD_LOGIC;
signal RIGHT  : STD_LOGIC;
signal LED : STD_LOGIC_VECTOR(7 downto 0);
begin
  -- Port out(Instantiation) Tail Light Controller Moore machine (TAIL_LIGHT) for simulation.
  -- Connect I/O port with internal signal which is declared in Test Bench.
  t_l : TAIL_LIGHT port map (
      CLK => CLK,
      RESET => RESET,
      BREAK => BREAK,
      LEFT => LEFT,
      RIGHT => RIGHT,
      LED => LED);
```

그 후, RESET, CLK, BREAK, LEFT, RIGHT(Input)에 대한 값들을 설정한다. 각각의 입력에 대한 설정은 각 입력마다 process 구문을 활용하고 해당 process 문 안에서 'wait for' 구문을 활용하여 원하는 시간에 원하는 값이 할당되도록 한다.

우선, RESET 의 입력 값은 10ns 일 때 '0', 25ns 일 때 '1', 73ns 일 때 '0', 85ns 일 때 '1'이 되도록 설정한다. 중간에 'wait for' 구문을 통해 지연 시간을 설정함으로써 위의 test process 를 구성할 수 있다. 이 부분에 대한 RESET 의 Test process 와 이에 대한 VHDL 코드만 다음과 같이 나타낸다.

소스로 주어진 RESET 의 Test process

Time(ns)	0	10	25	73	85
RESET	'X'	'0'	'1'	'0'	'1'

```
-- RESET signal test process.
rst_t : process
begin
  wait for 10ns;
  RESET <= '0';  -- RESET signal will be '0' at 10ns.
  wait for 15ns;
  RESET <= '1';  -- RESET signal will be '1' at 25ns.
  wait for 48ns;
  RESET <= '0';  -- RESET signal will be '0' at 73ns.
  wait for 12ns;
  RESET <= '1';  -- RESET signal will be '1' at 85ns.
  wait;
end process;
```

다음으로, CLK 은 주기를 10ns 로 설정한다. 이를 설정하기 위해 처음 5ns 동안에는 CLK 값이 '1'로 설정하고, 다음 5ns 동안에는 CLK 값이 '0'으로 설정한다. 이 과정이 Simulation 동안 반복이 되어 주기가 10ns 인 CLK 이 만들어진다. 이에 대한 VHDL 코드는 다음과 같다.

```
-- CLK signal test process.
-- The CLK signal period is 10ns.
clk_t : process
begin
  wait for 5ns;
  clk <= '0';
  wait for 5ns;
  clk <= '1';
end process;
```

BREAK, LEFT, RIGHT(Input)에 대한 값 설정은 RESET 의 값 설정과 동일한 방식으로 이루어지므로 자세한 설명은 RESET 의 값 설정 부분을 참고한다. 이 부분에 대한 BREAK, LEFT, RIGHT 의 Test process 와 이에 대한 VHDL 코드만 다음과 같이 나타낸다.

BREAK 의 Test process

Time(ns)	0	10	25	35	105	115
BREAK	'U'	'0'	'1'	'0'	'1'	'0'

LEFT 의 Test process

Time(ns)	0	10	35	55
X	'U'	'0'	'1'	'0'

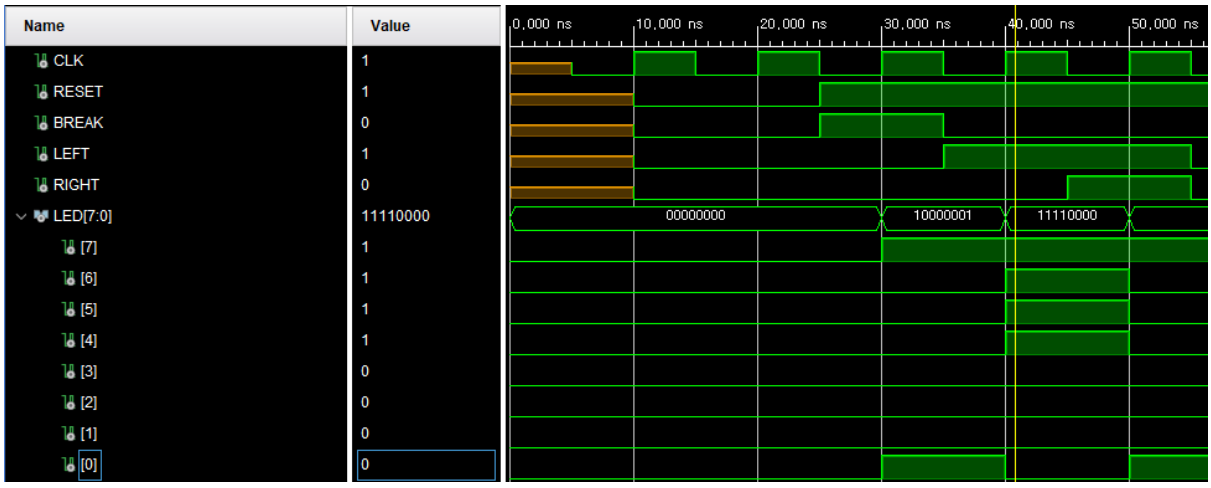
RIGHT 의 Test process

Time(ns)	0	10	45	55	65	145
X	'U'	'0'	'1'	'0'	'1'	'0'

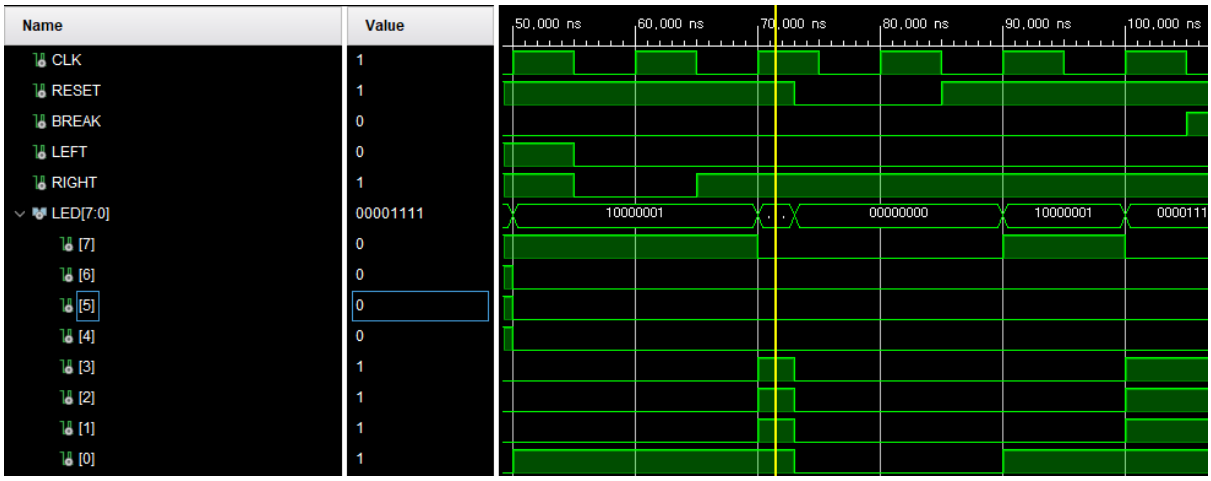
```
-- BREAK, LEFT, RIGHT(input) signal test process.
tl_t : process
  begin
    wait for 10ns;
    BREAK <= '0';  -- BREAK signal will be '0' at 10ns.
    LEFT <= '0';   -- LEFT signal will be '0' at 10ns.
    RIGHT <= '0';  -- RIGHT signal will be '0' at 10ns.
    wait for 15ns;
    BREAK <= '1';  -- BREAK signal will be '1' at 25ns.
    wait for 10ns;
    BREAK <= '0';  -- BREAK signal will be '0' at 35ns.
    LEFT <= '1';   -- LEFT signal will be '1' at 35ns.
    wait for 10ns;
    RIGHT <= '1';  -- RIGHT signal will be '1' at 45ns.
    wait for 10ns;
    LEFT <= '0';   -- LEFT signal will be '0' at 55ns.
    RIGHT <= '0';  -- RIGHT signal will be '0' at 55ns.
    wait for 10ns;
    RIGHT <= '1';  -- RIGHT signal will be '1' at 65ns.
    wait for 40ns;
    BREAK <= '1';  -- BREAK signal will be '1' at 105ns.
    wait for 10ns;
    BREAK <= '0';  -- BREAK signal will be '0' at 115ns.
    wait for 30ns;
    RIGHT <= '0';  -- RIGHT signal will be '0' at 145ns.
    wait;
end process;
```

⑤ 시뮬레이션 결과 및 분석

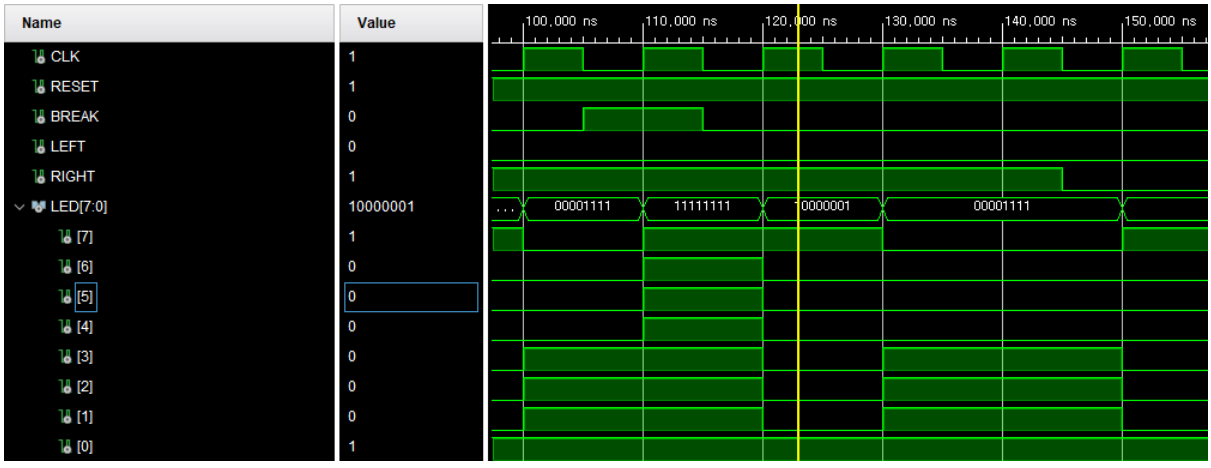
(1) Tail Light Controller 무어 상태기계 시뮬레이션 결과 및 분석



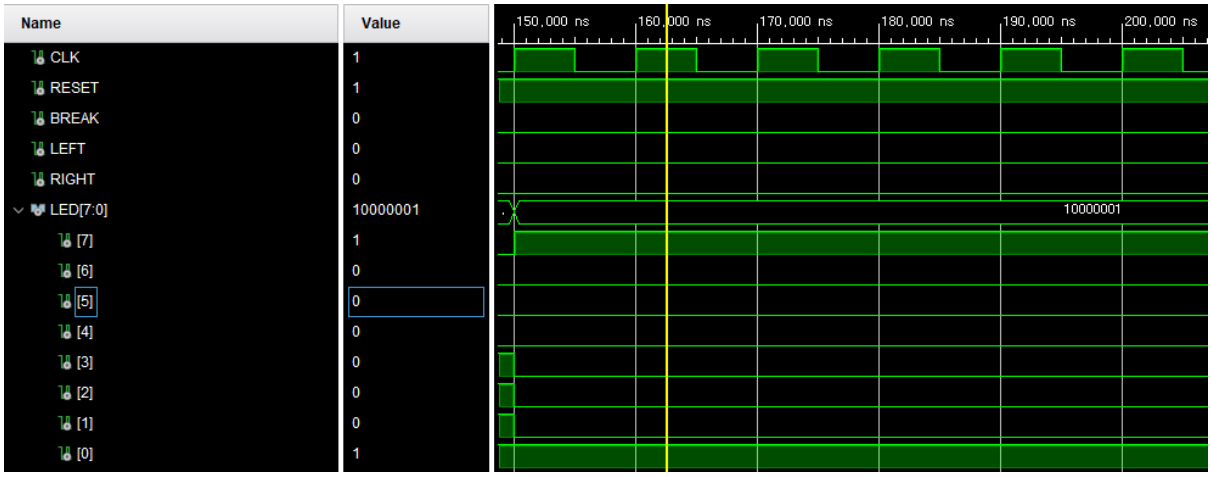
<Tail Light Controller Simulation 0ns ~ 50ns 의 결과>



<Tail Light Controller Simulation 50ns ~ 100ns 의 결과>



<Tail Light Controller Simulation 100ns ~ 150ns 의 결과>



<Tail Light Controller Simulation 150ns ~ 200ns 의 결과>

위의 Simulation waveform 을 보면 CLK 의 주기가 10ns 인 것을 볼 수 있다. 그리고, 처음 0ns 에는 모든 입력들에 대한 값이 설정되지 않아 'U'로 설정됨을 볼 수 있다. 10ns 부터 25ns 까지 RESET 값이 '0'이므로, 이 때, Tail Light Controller 는 RESET 동작을 하며, 상태는 초기 상태인 s_reset(RESET) 상태가 설정되며 출력도 "00000000"으로 설정된다. 25ns 이후 RESET 값이 '1'이 되면서 RESET 동작은 일어나지 않는 대신 CLK 신호가 trigger(rising edge)되는 시점마다 상태는 입력 BREAK, LEFT, RIGHT 값에 따라 알맞은 상태로 transition 이 일어나고, 현재 상태에 따라 출력 LED 의 8 bit 출력 값이 결정된다. 25ns 이후 RESET 신호가 다시 '0'이 되는 75ns 까지 입력 BREAK, LEFT, RIGHT, 상태(S), 그리고 출력(LED)에 대한 표를 그려보면 다음과 같다. 중간에 상태와 값이 바뀌는 부분은 CLK 이 rising edge trigger 되었을 때 나타나는 변화이다.

Time(ns)	25~35	35~45	45~55	55~65	65~75
BREAK	'1'	'0'	'0'	'0'	'0'
LEFT	'0'	'1'	'1'	'0'	'0'
RIGHT	'0'	'0'	'1'	'0'	'1'
State	s_reset -> s_ready	s_ready -> s_left	s_left -> s_ready	s_ready -> s_ready	s_ready -> s_right
LED	"00000000" -> "10000001""	"10000001" -> "11110000"	"11110000" -> "10000001"	"10000001" -> "10000001"	"10000001" -> "00001111"

그 후, 75ns 부터 85ns 까지 RESET 신호가 '0'이기 때문에 Tail Light Controller 는 RESET 동작이 일어나 초기 상태(s_reset)로 바뀌며, 출력 LED 도 "00000000"으로 설정된다. RESET 신호가 다시 '1'이 되는 85ns 부터 200ns 까지 입력 BREAK, LEFT, RIGHT, State, 그리고 출력(LED)에 대한 표를 그려보면 다음과 같다.

Time(ns)	85~95	95~105	105~115	115~125	125~140	140~145	145~150	150~200
BREAK	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'0'
LEFT	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
RIGHT	'1'	'1'	'1'	'1'	'1'	'1'	'0'	'0'
State	s_reset -> s_ready	s_ready -> s_right	s_right -> s_break	s_break -> s_ready	s_ready -> s_right	s_right	s_right -> s_ready	s_ready
LED	"00000000" -> "10000001"	"10000001" -> "00001111"	"00001111" -> "11111111"	"11111111" -> "10000001"	"10000001" -> "00001111"	"00001111"	"00001111" -> "10000001"	"10000001"

위에서 분석한 결과들을 보면, Tail Light Controller 의 RESET 동작과 CLK 과 입력 BREAK, LEFT, RIGHT 에 따른 상태변화가 State diagram 에 맞게 작동하여 결과적으로 Tail Light Controller 의 동작을 제대로 수행함을 알 수 있다. 자세히는 다음과 같다.

- s_reset 으로 transition 인 경우(RESET 동작) 검증 : 10ns~25ns / 75ns ~ 85ns
- s_ready 로 transition 인 경우(READY 동작) 검증 : 25ns~35ns / 45ns ~ 55ns / 55ns~65ns / 85ns ~ 95ns / 115ns~125ns / 145ns~150ns
- s_break 로 transition 인 경우(BREAK 동작) 검증 : 105ns~115ns
- s_left 로 transition 인 경우(LEFT 동작) 검증 : 35ns~45ns
- s_right 로 transition 인 경우(RIGHT 동작) 검증 : 65ns~75ns / 95ns~105ns / 125ns~140ns

⑥ 고찰

이번 과제에서는 Tail Light Controller 의 작동 방식을 이해하고 이를 바탕으로 Moore machine 을 Tail Light Controller 를 대상으로 VHDL 로 구현하고, 구현한 Moore machine 을 호출(Port Map)하여 Tail Light Controller 를 구현하였다. 구현 과정 중에서는 Tail Light Controller 의 Moore machine 에 대한 State diagram 을 그려서 작동 방식에 대해 먼저 설계하였다. 특히 이번 설계 과제는 실생활에 적용되는 동작에 대한 구현으로, 전자 부품에 들어있는 디지털 회로를 설계하기 위해서 이런 과정을 거쳐 설계되고 구현된다는 것을 느낄 수 있었다. 그리고, FSM 의 개념을 실생활과 연관지어 구성함으로써 FSM 의 개념은 단순히 하드웨어적, 소프트웨어적 개념에 국한되는 것이 아니라 굉장히 다양한 분야에 쓰일 수 있음을 알 수 있었다.
이번 과제를 통해서 추후 실생활과 관련된 동작에 대해 디지털 회로를 설계할 때 동작에 대한 상세한 이해, 동작을 구현하기 전에 해당 동작에 대한 FSM 과 같은 논리적 개념을 이용하여 이론적으로 나타내는 부분의 중요성을 깨달았고, 추후 어떤 하드웨어를 구현할 때마다 단순히 코드 그 자체에만 집중하기 보다 코드를 짜기 전 이론적 부분에 집중해야겠다는 생각이 들었다. 이 생각은 하드웨어나 소프트웨어를 설계할 때 많은 도움이 될 것이다.