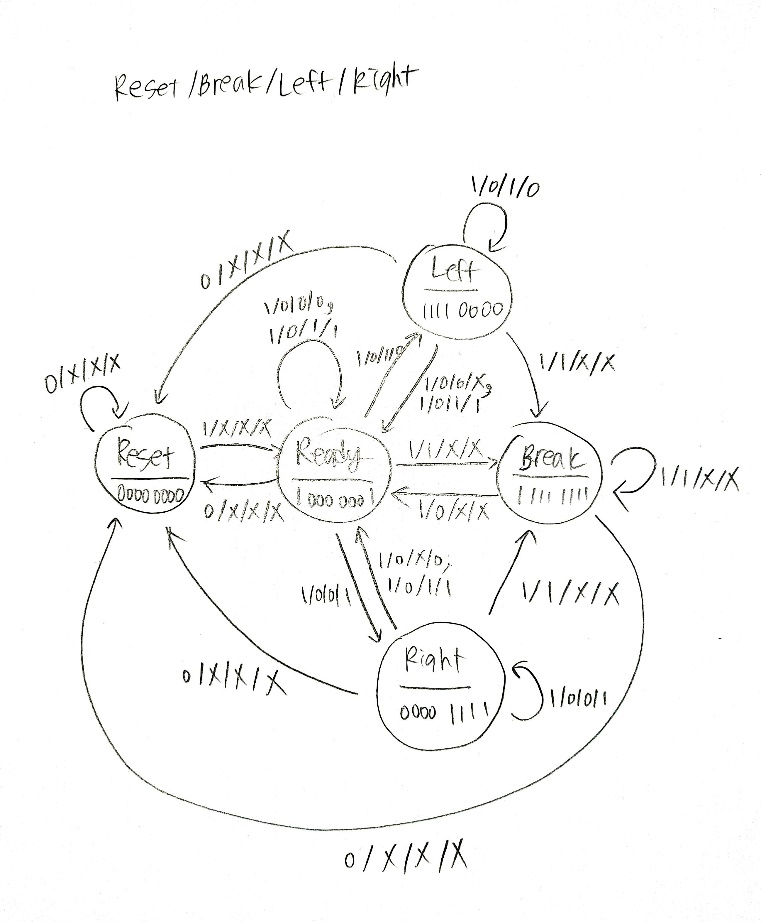
Tail Light Controller

Copyright 2022. Hyeonsu Lee All rights reserved.

(1) 설계

Reset이 0일 때는 무조건 Reset state로, Reset=1일 때는 spec에 맞게 동작하도록 state diagram을 그렸다. 또한 무어 머신이기 때문에 state와 output을 같이 썼다.(각각에 해당하는 출력 기입/ 어떤 led가 on되는지를 알 수 있다.)

또한 과제 pdf에서 설명되어 있는 대로 특정 상태에서 ~입력과 상관없이라는 말이 있으면 don’t care로, ‘~만 입력되었을 때’ 라는 말이 있으면 그 입력만 1이고 다른 입력은 0을 인가하였다. 그대로 state diagram을 그려보면 좌측과 같은 결과를 얻을 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| State | State assignment | 의미 | Present  Output |
| Reset | 000 | Reset이 활성화된 상태(Reset=’0’) / 모든 LED off | 00000000 |
| Ready | 001 | Reset이 비활성화되었고, left, right, break가 아닌 상태(양 끝 LED만 on된 상태) | 10000001 |
| Left | 010 | Reset 비활성화, 왼쪽 4개의 LED가 on된 상태 | 11110000 |
| Right | 011 | Reset 비활성화, 오른쪽 4개의 LED가 on된 상태 | 00001111 |
| Break | 100 | Reset 비활성화, 모든 LED가 on된 상태 | 11111111 |

Tail light 제어기 동작 원리

-asynchronous reset을 이용하기 때문에 reset을 제외한 다른 입력에 상관없이 reset이 0이 되면 reset state로 돌아가고 출력 또한 ‘00000000’이 된다.

-reset이 비활성화된 상태에서 Break 입력이 1이 되면 left, right 입력에 상관없이 break 상태가 되고 그 말은 left, right보다 우위에 있음을 의미한다.

-left, right는 각각 reset과 break가 비활성화된 상태에서만 해당 입력만이 1일 때 활성화된다.

(2) 코드

\*tail light 제어기 코드

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

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;

* Led는 8bit이기 때문에 std\_logic\_vector(7 downto 0)를 이용하였다.

architecture Behavioral of TAIL\_LIGHT is

signal state : std\_logic\_vector(2 downto 0); --state의 개수가 총 5개이기 때문에 0~7까지 나타낼 수 있도록 3bit std\_logic\_vector 이용

signal next\_state : std\_logic\_vector(2 downto 0); --Flipfolp 3개 이용한다는 의미

* State, next state를 이전 과제와 달리 직접 쓰는 것이 아니라 인코딩 후, 즉 state를 2진수로 바꿔서 사용하기 위해 std\_logic\_vector(2 downto 0)를 이용했다.
* State 5개이므로 3bit를 이용한 것이고, flipflop또한 3개를 이용한다.

begin

--state transition

process(CLK, RESET) --state reg는 sequential logic이기 때문에 CLK필요, asynchronous reset

begin

if(RESET='0') then

state<="000"; --RESET state

elsif(CLK='1' and CLK'event) then

state<=next\_state;

end if;

end process;

-> state의 transition은 clk의 에지에서 일어나기 때문에 sequential logic이다. Sensitivity list에 clk를 포함했고, asyn reset이기 때문에 마찬가지로 sensitivity list에 포함시켰다.(clk에 따라 reset이 되는 것이 아니어서)

--led out

process(state) --moore machine은 출력이 오직 현재 상태에 의해서 결정

begin

if(state="000") then --RESET state

LED<="00000000"; -- all led off

elsif(state="001") then --ready state

LED<="10000001"; --양 끝 led on

elsif(state="010") then -- left state

LED<="11110000"; -- 왼쪽 4개 led on

elsif(state="011") then -- right state

LED<="00001111"; --오른쪽 4개 led on

else

LED<="11111111"; --break state / all led on

end if;

end process;

-> 설계 과정에서 assignment한 각각의 state에 따른 출력을 설정하였다.

-> 무어 머신은 오직 현재 상태에 의해서만 출력이 결정되기 때문에 sensitivity list에 state를 포함시켰다.

--next state

process(RESET, BREAK, LEFT, RIGHT, state) --현재 상태에서 어떤 입력을 받는지에 따라 다음 상태 결정

begin

--reset state

if(state="000")then

if(RESET='0') then next\_state<="000"; --다음 상태 reset state

else next\_state<="001"; --다음 상태 ready state

end if;

--ready state

elsif(state="001") then

if(RESET='0') then next\_state<="000"; --다음 상태 reset state

elsif(BREAK='1') then next\_state<="100"; --다음 상태 break state

elsif(LEFT='1' and RIGHT='0') then next\_state<="010"; --다음 상태 left state

elsif(LEFT='0' and RIGHT='1') then next\_state<="011"; --다음 상태 right state

else next\_state<="001"; --다음 상태 ready state

end if;

--left state

elsif(state="010") then

if(RESET='0') then next\_state<="000"; --다음 상태 reset state

elsif(BREAK='1') then next\_state<="100"; --다음 상태 break state

elsif(LEFT='1' and RIGHT='0') then next\_state<="010"; --다음 상태 left state

else next\_state<="001"; --다음 상태 ready state

end if;

--right state

elsif(state="011") then

if(RESET='0') then next\_state<="000"; --다음 상태 reset state

elsif(BREAK='1') then next\_state<="100"; --다음 상태 break state

elsif(LEFT='0' and RIGHT='1') then next\_state<="011"; --다음 상태 right state

else next\_state<="001"; --다음 상태 ready state

end if;

--break state

else

if(RESET='0') then next\_state<="000"; --다음 상태 reset state

elsif(BREAK='1') then next\_state<="100"; --다음 상태 break state

else next\_state<="001"; --다음 상태 ready state

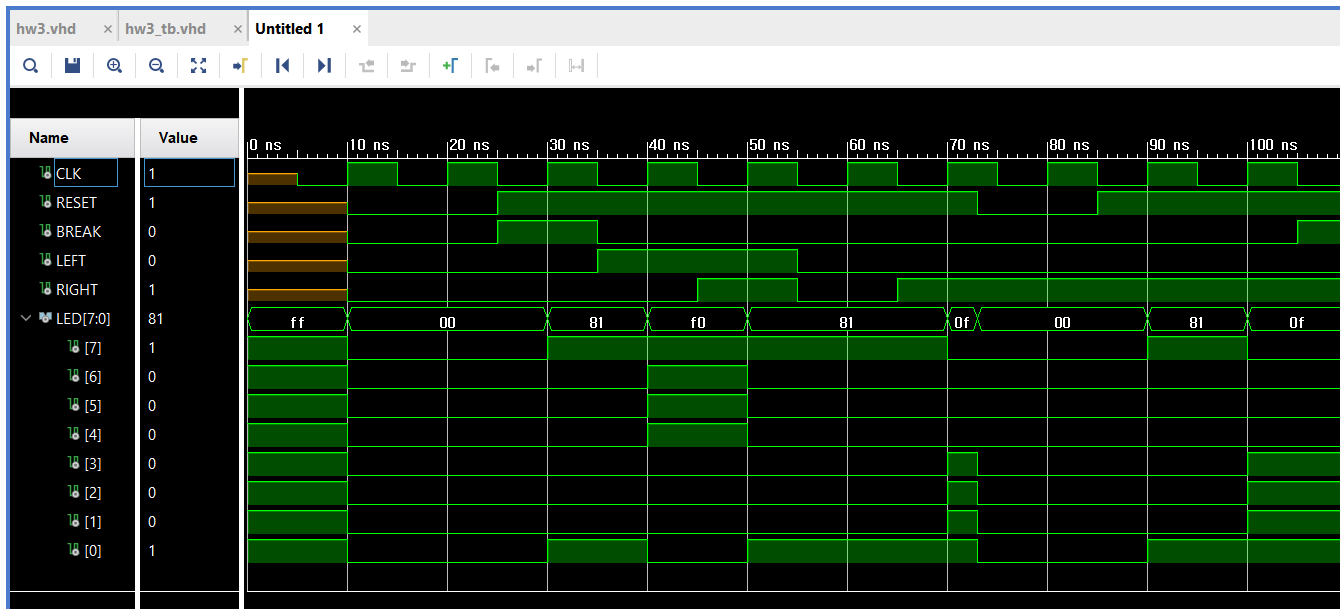
end if;

end if;

end process;

-> state diagram을 참고하여 각각의 state에서 어떤 입력을 받고 다음 상태가 되는지를 파악하여 코드를 작성하였다.

(2) waveform 결과 및 분석



모든 입력이 인가되는 시점은 10ns이기 때문에 그 이후부터 분석하려고 한다.

1/1/1/1 이런 식으로 쓸 때가 있다면, Reset/Break/Left/Right의 입력을 의미한다.

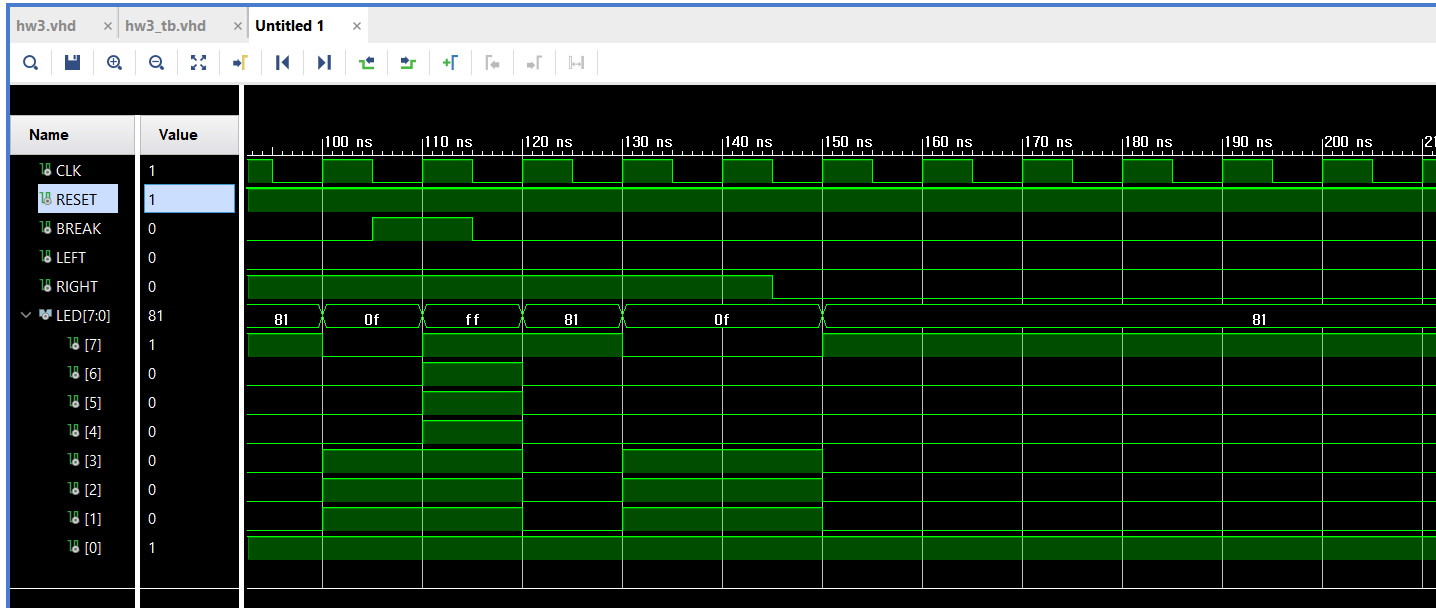
무어머신이기 때문에 클락의 엣지에서 출력이 바로 변한다.

10~25ns, 73~85ns에서는 reset이 0이기 때문에 다른 입력이 무엇이든 상관없이 출력이 0임을 확인할 수 있다.

25ns~73ns - reset이 1이기 때문에 비활성화된다.

30ns에서 break는 1이 되지만 reset state에서 ready state로 변화하는 과정이기 때문에 출력은 ‘10000001’이 된다. 40ns에서 Ready state인데 reset, left는 1이고 나머지 입력은 0이기 때문에 left state가 되고 출력은 ‘11110000’이 되는 것을 확인할 수 있다. 50ns에서 left state인데 reset은 1, break는 0이고 left=right=1이기 때문에 state diagram을 참고하면 ready state로 돌아가는 것을 확인할 수 있고 실제 waveform에서도 ready state가 된 것을 출력을 통해 확인할 수 있다. 이때 출력은 ‘10000001’이다. 60ns에서 ready state인데 입력이 1/0/0/0이므로 ready state에 머물고 출력 또한 ‘10000001’이다. 70ns에서 ready state인데 입력이 1/0/0/1이기 떄문에 right state가 되고 출력은 ‘00001111’이 되는 것을 확인할 수 있다. 하지만 이내 reset이 0이 되어 출력은 ‘00000000’이 된다.

Reset이 되었기 때문에 다시 reset state로 돌아갔고, 90ns에서 ready state가 됐음을 알 수 있다. 따라서 출력은 ‘10000001’이다. 100ns에서 ready state에서 1/0/0/1의 입력을 받았으므로 다시 right state가 되며 출력은 ‘00001111’이 되는 것을 볼 수 있다.



110ns -> right 상태에서 1/1/0/1의 입력을 받았으므로 right, left의 입력에 상관없이 break 상태가 되는 것을 알 수 있다. 따라서 출력은 ‘11111111’이다.

120ns -> break state에서 reset은 1일 때 break는 0이 되었기 때문에 다시 ready state로 돌아가고 출력은 ‘10000001’이 된다.

130ns -> ready state에서 1/0/0/1의 입력을 받아서 right state로 가고 출력은 ‘00001111’이 되는 것을 알 수 있다.

140ns -> right state에서 여전히 reset=1인 상태에서 right을 제외한 모든 입력이 0이기 때문에 right state에 머물고 출력 또한 바로 전과 같다.

150ns -> right state에서 입력이 1/0/0/0이기 때문에 ready state가 되고 출력은 ‘10000001’임을 확인할 수있다. 이 이후로 입력의 변화가 없기 때문에 ready state에 머물고 출력 또한 ‘10000001’인 것을 확인할 수 있다.