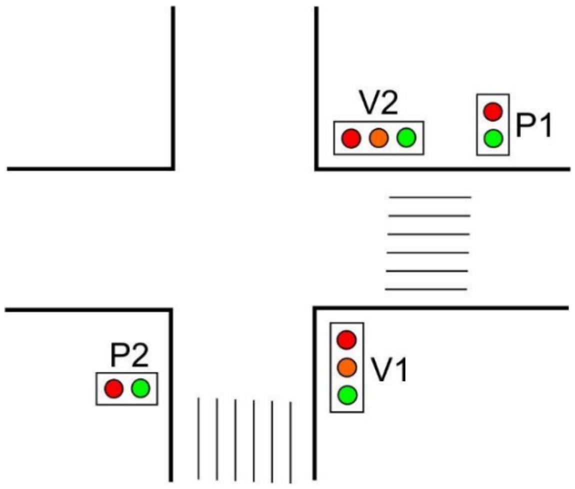
Light Controller for Crossroads

This is the second homework for the course of Programmable Logic Devices in the 2021-2022 Spring semester of West Pomeranian University of Technology.

> Author: **Gökhan Koçmarlı** (gokhan-kocmarli@zut.edu.pl)

In the homework, we're asked to create a controller device for such a crossroad, prepare its state diagram and code the FPGA with using VHDL. One should notice that in the assignment's instructions document, we have no information about the light status of the pedestrians if the vehicle lights in the state of green and yellow is on. I made a decision to make pedestrians stop (red) in that states, since it is much more safer to do it.



This design does not care about the traffic itself. Therefore, it is known as fixed-sequence (which means always with the same delay translation) traffic lights system.

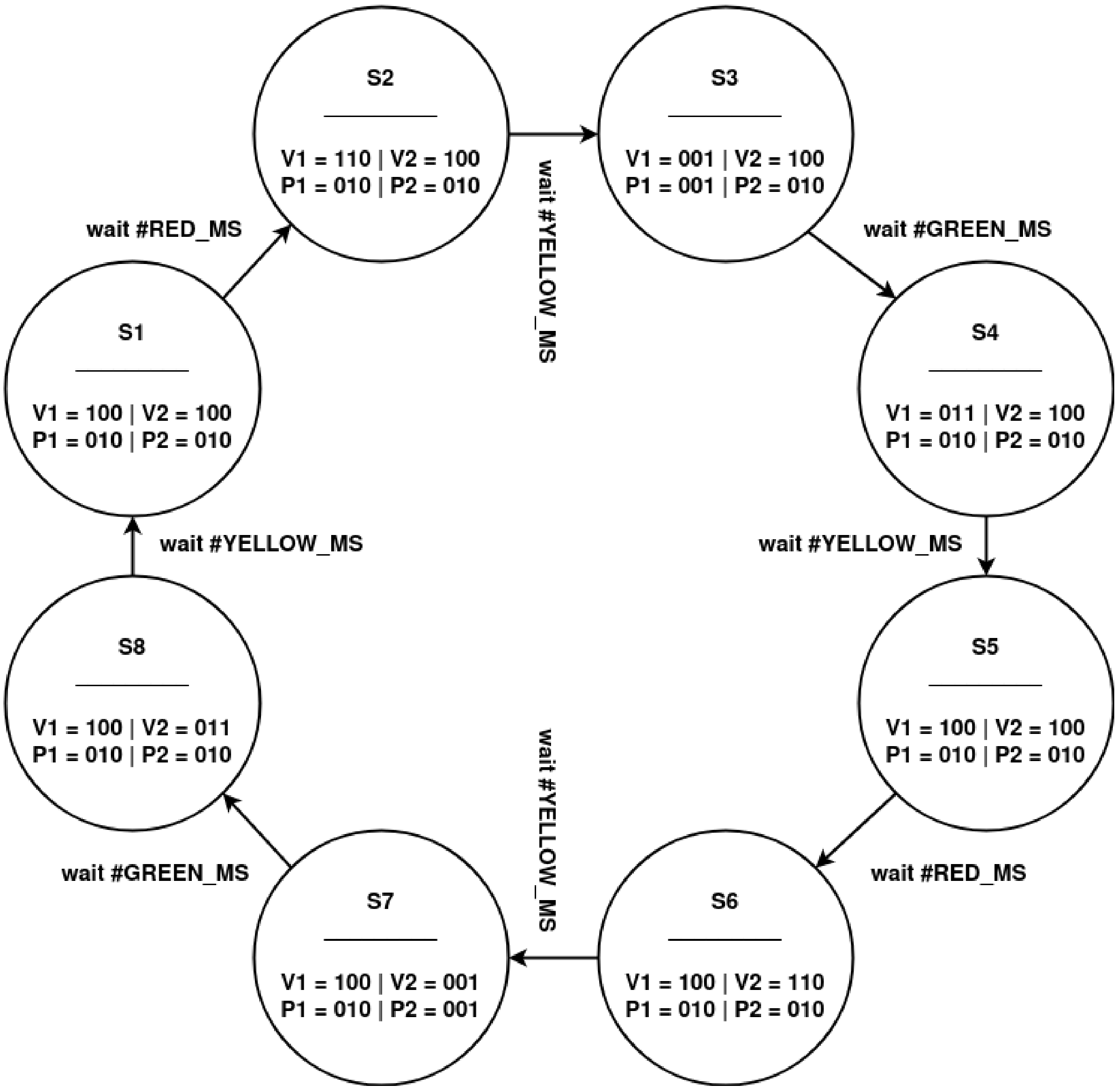
Task 1. Creating a State Table

There are eight different state that we can write for this example. After the state `S8`, the mechanism will move to the beginning, to `S1`.

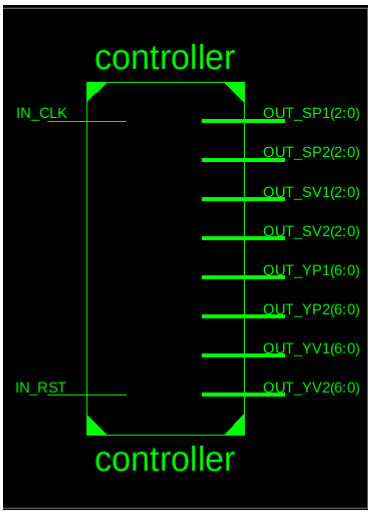
Number	State	Vehicle V1	Pedestrian P1	Vehicle V2	Pedestrian P2	Description
1	S1	100	010	100	010	All are read light.
2	S2	110	010	100	010	Vehicles on V1 starts to go.
3	S3	001	001	100	010	Vehicles on V1 goes, Pedestrains on P1 goes.
4	S4	011	010	100	010	Vehicles on V1 start to stop, Pedestrains on P1 stops.
5	S5	100	010	100	010	All are read light.
6	S6	100	010	110	010	Vehicles on V2 starts to go.
7	S7	100	010	001	001	Vehicles on V2 goes, Pedestrains on P2 goes.
8	S8	100	010	011	010	Vehicles on V2 start to stop, Pedestrains on P2 stops.

Task 2. Represent the States with a Diagram

In the second figure, you can find the state diagram for our traffic light design. Notice that changing between states only depends on the timer, which is shown as `#LIGHT_MS` concept. In the assignment paper, it is said to us that `#RED_MS = #YELLOW_MS = #GREEN_MS = 1000 ms`.



Task 3. Implement it using the VHDL



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity controller is
    port (
        -- Inputs
        IN_RST, IN_CLK                      : in std_logic;
        -- Outputs for Showing States
        OUT_SV1, OUT_SV2, OUT_SP1, OUT_SP2  : out std_logic_vector(2 downto 0);
        -- Outputs for 7 Segment Display
        OUT_YV1, OUT_YV2, OUT_YP1, OUT_YP2  : out std_logic_vector(6 downto 0)

    );
end controller;

architecture Behavioral of controller is
    -- For much readable code, create state light definitations.
    constant S_V_RED            : std_logic_vector(2 downto 0) := "100";
    constant S_V_RED_YELLOW     : std_logic_vector(2 downto 0) := "110";
    constant S_V_GREEN          : std_logic_vector(2 downto 0) := "001";
    constant S_V_GREEN_YELLOW   : std_logic_vector(2 downto 0) := "011";
    constant S_P_RED            : std_logic_vector(2 downto 0) := "010";
    constant S_P_GREEN          : std_logic_vector(2 downto 0) := "001";
    -- Also, create 7 segment light definitations.
    constant Y_V_RED            : std_logic_vector(6 downto 0) := "1000000";
    constant Y_V_RED_YELLOW     : std_logic_vector(6 downto 0) := "1000001";
    constant Y_V_GREEN          : std_logic_vector(6 downto 0) := "0001000";
    constant Y_V_GREEN_YELLOW   : std_logic_vector(6 downto 0) := "0001001";
    constant Y_P_RED            : std_logic_vector(6 downto 0) := "0000001";
    constant Y_P_GREEN          : std_logic_vector(6 downto 0) := "0001000";
    -- To travel throught states, create a type.
    type states is (S1, S2, S3, S4, S5, S6, S7, S8);
    signal current_state : states;

begin
        -- Travel the states in each cycle of the clock.
        state_travel: process(IN_RST, IN_CLK)
            begin
```

```vhdl
            -- Set the reset condition.
            if (IN_RST = '1') then
                current_state <= S1;
            elsif (rising_edge(IN_CLK)) then
                -- Set the cases of translation to next states.
                case current_state is
                    when S1 => current_state <= S2;
                    when S2 => current_state <= S3;
                    when S3 => current_state <= S4;
                    when S4 => current_state <= S5;
                    when S5 => current_state <= S6;
                    when S6 => current_state <= S7;
                    when S7 => current_state <= S8;
                    when S8 => current_state <= S1;
                end case;
            end if;
        end process;

    -- Give the exact outputs for the 7 segment display.
    do_state_work: process(IN_CLK)
        begin
            if (rising_edge(IN_CLK)) then
                -- All are read light.
                if (current_state = S1) then
                    -- State Outputs
                    OUT_SV1 <= S_V_RED;
                    OUT_SV2 <= S_V_RED;
                    OUT_SP1 <= S_P_RED;
                    OUT_SP2 <= S_P_RED;
                    -- 7 Segment Outputs
                    OUT_YV1 <= Y_V_RED;
                    OUT_YV2 <= Y_V_RED;
                    OUT_YP1 <= Y_P_RED;
                    OUT_YP2 <= Y_P_RED;

                -- Vehicles on V1 start to go.
                elsif (current_state = S2) then
                    -- State Outputs
                    OUT_SV1 <= S_V_RED_YELLOW;
                    -- 7 Segment Outputs
                    OUT_YV1 <= Y_V_RED_YELLOW;

                -- Vehicles on V1 goes, Pedestrains on P1 goes.
                elsif (current_state = S3) then
                    -- State Outputs
                    OUT_SV1 <= S_V_GREEN;
                    OUT_SP1 <= S_P_GREEN;
                    -- 7 Segment Outputs
                    OUT_YV1 <= Y_V_GREEN;
                    OUT_YP1 <= Y_P_GREEN;

                -- Vehicles on V1 start to stop, Pedestrains on P1 stops.
                elsif (current_state = S4) then
                    -- State Outputs
                    OUT_SV1 <= S_V_GREEN_YELLOW;
```

```vhdl
                        OUT_SP1 <= S_P_RED;
                        -- 7 Segment Outputs
                        OUT_YV1 <= Y_V_GREEN_YELLOW;
                        OUT_YP1 <= Y_P_RED;

                    -- All are read light.
                    elsif (current_state = S5) then
                        -- State Outputs
                        OUT_SV1 <= S_V_RED;
                        -- 7 Segment Outputs
                        OUT_YV1 <= Y_V_RED;

                    -- Vehicles on V2 starts to go.
                    elsif (current_state = S6) then
                        -- State Outputs
                        OUT_SV2 <= S_V_RED_YELLOW;
                        -- 7 Segment Outputs
                        OUT_YV2 <= Y_V_RED_YELLOW;

                    -- Vehicles on V2 goes, Pedestrains on P2 goes.
                    elsif (current_state = S7) then
                        -- State Outputs
                        OUT_SV2 <= S_V_GREEN;
                        OUT_SP2 <= S_P_GREEN;
                        -- 7 Segment Outputs
                        OUT_YV2 <= Y_V_GREEN;
                        OUT_YP2 <= Y_P_GREEN;

                    -- Vehicles on V2 start to stop, Pedestrains on P2 stops.
                    elsif (current_state = S8) then
                        -- State Outputs
                        OUT_SV2 <= S_V_GREEN_YELLOW;
                        OUT_SP2 <= S_P_RED;
                        -- 7 Segment Outputs
                        OUT_YV2 <= Y_V_GREEN_YELLOW;
                        OUT_YP2 <= Y_P_RED;

                    end if;
                end if;
            end process;
end Behavioral;
```