UNIVERSITY OF
ALBERTA

ECE 511 DIGITAL ASIC DESIGN

Report of Lab 3: Traffic Intersection Controller

Student Name : Gangyi Li
Student Number : 1674819

December 10, 2021

Department of Electrical and Computer Engineering
University of Alberta

Abstract:

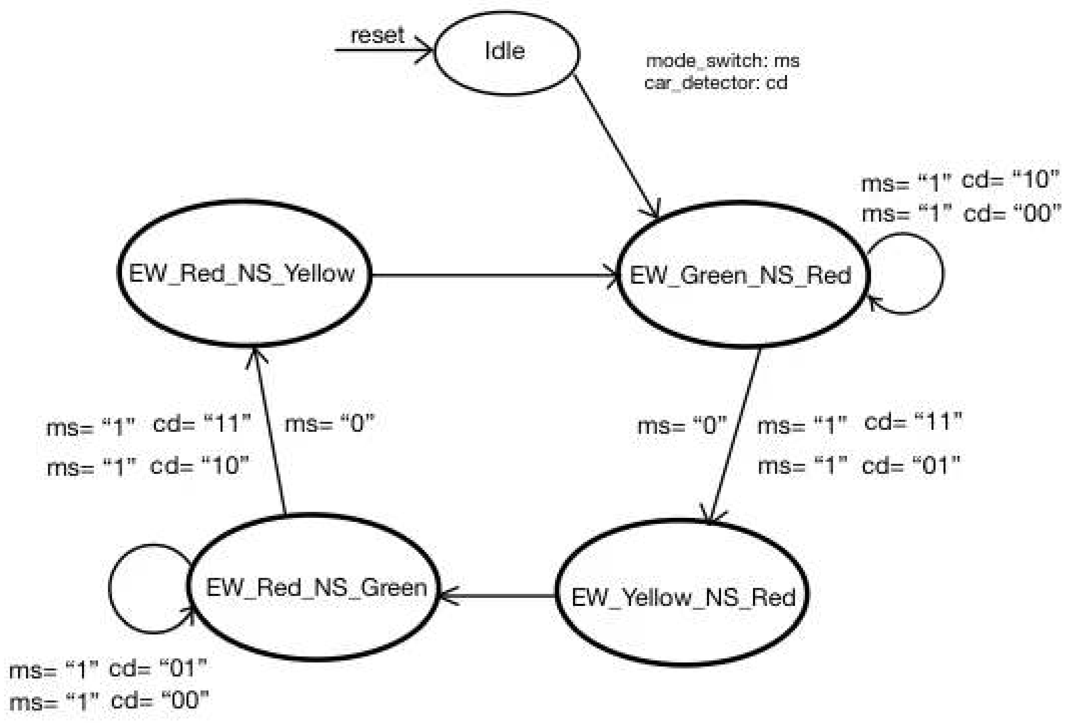
The purpose of this lab is to use the functionality of Xilinx Vivado to design and implement a traffic light controller for use in road intersections of normal and night modes. The function of red light camera and 7-segment display with a keypad can been realized in the finite state machine design.

1. Design

A traffic intersection controller has traffic lights on the two directions which are east-west (EW) and north-south (NS). The signal *"direction_select"* can change different directions on board. In order to involve all situations in the crossroads, there are five states in the design, which are *"idle", "EW_Red_NS_Green", "EW_Red_NS_Yellow", "EW_Yellow_NS_Red", "EW_Green_NS_Red"*. The state *"idle"* represents the initial state before changing, the other states represent the transition of light color. The signal *"state_signal"* shows these four states of light changing. In the normal mode, when the color of lights on one side changes from red to green, the transition of the lights on the other side is green->yellow-red. The delay for red, green and yellow lights is 20s, 20s and 2s respectively. The constant signals *"delay20s"* and *"delay2s"* show the count numbers. The signal *"count"* can be used in each state to realize different delays. To acquire 1Hz clock signal on board, the component *clk_divider_1Hz* can be used in the design. In the night mode, only a vehicle that approaches a traffic intersections and that encounters a red light, the light will change in the same rule of normal mode. immediately without delay. The signal "mode_switch" can be used to realize changing the night or normal modes. And the *"car_detector"* can show if there are vehicles on each direction in night mode. If any vehicle crosses an intersection despite a red light, the system can generate short duration pulse by the signal *"illegal_car_detector"* and activate *"external_camera"* signal to take picture. The specific meanings of each signal are shown in the list as below:

Signals with value	Meaning
direction_select = '1	north-south direction
direction_select = '0'	east-west direction
mode_switch = '1'	night mode
mode_switch = '0'	normal mode
car_detector = "00"	no vehicle
car_detector = "01"	vehicles on north-south direction
car_detector = "10"	vehicles on east-west direction
car_detector = "11"	vehicles on both directions
illegal_car_detector = "00"	none vehicles crossing in red light
illegal_car_detector = "01"	vehicles on north-south direction crossing in red light
illegal_car_detector = "10"	vehicles on east-west direction crossing in red light
illegal_car_detector = "11"	---
external_camera = "00"	none cameras taking pictures
external_camera = "01"	cameras on north-south direction taking pictures
external_camera = "10"	cameras on east-west direction taking pictures
external_camera = "11"	---
state_signal = "00"	State: EW_Green_NS_Red
state_signal = "01"	State: EW_Yellow_NS_Red
state_signal = "10"	State: EW_Red_NS_Green
state_signal = "11"	State: EW_Red_NS_Yellow

The schematic of state transition graph is shown in figure 1.



reset	mode_switch	car_detector	current state	next state
1	---	---	---	idle
0	---	---	idle	EW_Green_NS_Red
0	0	00	EW_Green_NS_Red	EW_Yellow_NS_Red
0	0	01	EW_Yellow_NS_Red	EW_Red_NS_Green
0	0	10	EW_Red_NS_Green	EW_Red_NS_Yellow
0	0	11	EW_Red_NS_Yellow	EW_Green_NS_Red
0	1	00	EW_Green_NS_Red	EW_Green_NS_Red
0	1	00	EW_Red_NS_Green	EW_Red_NS_Green
0	1	01	EW_Green_NS_Red	EW_Yellow_NS_Red
0	1	10	EW_Red_NS_Green	EW_Red_NS_Yellow
0	1	11	EW_Green_NS_Red	EW_Yellow_NS_Red
0	1	11	EW_Red_NS_Green	EW_Red_NS_Yellow

Figure1 : Schematic of State Transition Graph

The specific meanings of each process are shown in the list as below:

Process	Functions
normal_and_night_modes_change	Design the state transition with related delays in both modes
light_switch_and_camera_detector	Design the change of the traffic light colors and red light camera feature in each state
segment_show	Design 7-Segment display according to keypad
pedestrian_signal	Design pedestrian signals in each state

In the segment_show process, the content showing in both segments is decided by keypad. If the button "A" is pressed on the keypad, the 7-segment on left hand will show the current system state from 0 to 3 , and the 7-segment on the right hand will show the remaining time of current state from 8 to 0 or from 1 to 0. If the button "3" is pressed on the keypad, the 7-segment on the left hand will show the pedestrian_crosswalk_NS signal, and the 7-segment on the right will show the pedestrian_crosswalk_EW signal. These pedestrian signals are decided according to the green light in the final process. If there is a green light in the direction, the pedestrian signal will be '1'. If there are yellow and red lights in the direction, the pedestrian signal will be '0'. The button "A" and "3" are decided by the output signal in the design of the decoder of the keypad.

Design code for Traffic Intersection Controller

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

entity Traffic_system is
    Port(clk : in std_logic;
         reset : in std_logic;
         direction_select : in std_logic;
         mode_switch : in std_logic;
         car_detector : in std_logic_vector(1 downto 0);
         illegal_car_detector : in std_logic_vector(1 downto 0);
         external_camera : out std_logic_vector(1 downto 0);
         traffic_lights : out std_logic_vector(2 downto 0);
         state_signal : out std_logic_vector(1 downto 0);
         select_segment : out std_logic;
         display_sum : out std_logic_vector(6 downto 0);
         KeyPad_Row : inout std_logic_vector(3 downto 0);
         KeyPad_Col : inout std_logic_vector(3 downto 0)
         );
end entity;

architecture behaviour of Traffic_system is
    type stateType is (idle, EW_Red_NS_Green, EW_Red_NS_Yellow,
EW_Yellow_NS_Red, EW_Green_NS_Red);
    signal current_state : stateType;
    signal clk_1Hz : std_logic;
    signal clk_1000Hz : std_logic;
    signal count : integer := 1;
    signal Decode: STD_LOGIC_VECTOR (3 downto 0);
    signal pedestrian_crosswalk_EW : std_logic;
    signal pedestrian_crosswalk_NS : std_logic;
    constant delay20s : natural := 20;
    constant delay2s : natural := 2;
    component clk_divider is
        Port( clk_in : in std_logic;
              clk_out : out std_logic);
    end component;
    component clk_divider_1000Hz is
        Port( clk_in : in std_logic;
              clk_out : out std_logic);
    end component;
    component Decoder is
        Port (clk : in  STD_LOGIC;
              Row : in  STD_LOGIC_VECTOR (3 downto 0);
              Col : out  STD_LOGIC_VECTOR (3 downto 0);
              DecodeOut : out  STD_LOGIC_VECTOR (3 downto 0));
    end component;
begin
    label_clk_divider_1hz : clk_divider_1hz port map (clk_in => clk, clk_out =>
clk_1Hz);
    label_clk_divider_1000hz : clk_divider_1000hz port map (clk_in => clk,
clk_out => clk_1000Hz);
    C0: Decoder port map (clk=>clk, Row =>KeyPad_Row, Col=>KeyPad_Col,
DecodeOut=> Decode);
    normal_and_night_modes_change : process(reset, clk_1Hz, mode_switch,
car_detector, current_state) is
    begin
    if reset = '1' then
        current_state <= idle;
    elsif clk_1Hz'event and clk_1Hz = '1' then
        case current_state is
            when idle =>
                current_state <= EW_Green_NS_Red;

            when EW_Green_NS_Red =>
                if (mode_switch = '0') then
                    if(count = delay20s) then
                        count <= 1;
                        current_state <= EW_Yellow_NS_Red;
                    else
                        count <= count + 1;
                        current_state <= EW_Green_NS_Red;
                end if;
            else
                if (car_detector = "01") then
                    current_state <= EW_Yellow_NS_Red;
                elsif (car_detector = "11") then
                    if(count = delay2s) then
                        count <= 1;
                        current_state <= EW_Yellow_NS_Red;
                    else
                        count <= count + 1;
                        current_state <= EW_Green_NS_Red;
                    end if;
                else
                    current_state <= EW_Green_NS_Red;
                end if;
            end if;
            when EW_Yellow_NS_Red =>
                if(mode_switch = '0') then
                    if (count = delay2s) then
                        count <= 1;
                        current_state <= EW_Red_NS_Green;
                    else
                        count <= count + 1;
                        current_state <= EW_Yellow_NS_Red;
                    end if;
                else
                    current_state <= EW_Red_NS_Green;
               end if;

            when EW_Red_NS_Green =>
                if (mode_switch = '0') then
                    if (count = delay20s) then
                       count <= 1;
                       current_state <= EW_Red_NS_Yellow;
                    else
                        count <= count + 1;
                        current_state <= EW_Red_NS_Green;
                    end if;
                else
                    if (car_detector = "10") then
                        current_state <= EW_Red_NS_Yellow;
                    elsif (car_detector = "11") then
                        if(count = delay2s) then
                            count <= 1;
                            current_state <= EW_Red_NS_Yellow;
                        else
                            count <= count + 1;
                            current_state <= EW_Red_NS_Green;
                        end if;
                    else
                        current_state <= EW_Red_NS_Green;
                    end if;
                end if;

            when EW_Red_NS_Yellow =>
                if(mode_switch = '0') then
                    if (count = delay2s) then
                        count <= 1;
                        current_state <= EW_Green_NS_Red;
                    else
                        count <= count + 1;
                        current_state <= EW_Red_NS_Yellow;
                    end if;
               else
                    current_state <= EW_Green_NS_Red;
              end if;
        end case;
      end if;
    end process;

    light_switch_and_camera_detector : process(current_state, direction_select,
illegal_car_detector) is
    begin
```

```
        case(current_state) is
            when idle =>
                traffic_lights <= "000";
                external_camera <= "00";
                state_signal <= "--";

            when EW_Green_NS_Red =>  state_signal <= "00";
                if(illegal_car_detector(0) = '1') then
                    external_camera <= "01";
                else
                    external_camera <= "00";
                end if;
                if(direction_select = '1') then
                    traffic_lights <= "100";
                else
                    traffic_lights <= "010";
                end if;

            when EW_Yellow_NS_Red =>  state_signal <= "01";
                if(illegal_car_detector(0) = '1') then
                    external_camera <= "01";
                else
                    external_camera <= "00";
                end if;
                if(direction_select = '1') then
                    traffic_lights <= "100";
                else
                    traffic_lights <= "001";
                end if;

            when EW_Red_NS_Green => state_signal <= "10";
                if(illegal_car_detector(1) = '1') then
                    external_camera <= "10";
                else
                    external_camera <= "00";
                end if;
                if(direction_select = '1') then
                    traffic_lights <= "010";
                else
                    traffic_lights <= "100";
                end if;

            when EW_Red_NS_Yellow =>  state_signal <= "11";
                if(illegal_car_detector(1) = '1') then
                    external_camera <= "10";
                else
                    external_camera <= "00";
                end if;
                if(direction_select = '1') then
                    traffic_lights <= "001";
                else
                    traffic_lights <= "100";
                end if;
        end case;
    end process;

    Segment_show : process(clk, Decode, count, current_state,
pedestrian_crosswalk_EW) is
```

```
      begin
      if (clk_1000Hz = '1') then
          select_segment <= '0';
          if Decode = "1111" then
          case(current_state) is
              when idle =>
                 display_sum <= "0000000";
              when EW_Green_NS_Red =>
                  if (count = 1) then
                      display_sum <= "1111111";
                  elsif (count = 2) then
                      display_sum <= "0000111";
                  elsif (count = 3) then
                      display_sum <= "1111101";
                  elsif (count = 4) then
                      display_sum <= "1101101";
                  elsif (count = 5) then
                      display_sum <= "1100110";
                  elsif (count = 6) then
                      display_sum <= "1001111";
                  elsif (count = 7) then
                      display_sum <= "1011011";
                  elsif (count = 8) then
                      display_sum <= "0000110";
                  elsif (count = 9) then
                      display_sum <= "0111111";
                  else
                      display_sum <= "0000000";
                  end if;

              when EW_Yellow_NS_Red =>
                  if (count = 1) then
                      display_sum <= "0000110";
                  elsif (count = 2) then
                      display_sum <= "0111111";
                  else
                      display_sum <= "0000000";
                  end if;

              when EW_Red_NS_Green =>
                  if (count = 1) then
                      display_sum <= "1111111";
                  elsif (count = 2) then
                      display_sum <= "0000111";
                  elsif (count = 3) then
                      display_sum <= "1111101";
                  elsif (count = 4) then
                      display_sum <= "1101101";
                  elsif (count = 5) then
                      display_sum <= "1100110";
                  elsif (count = 6) then
                      display_sum <= "1001111";
                  elsif (count = 7) then
                      display_sum <= "1011011";
                  elsif (count = 8) then
                      display_sum <= "0000110";
                  elsif (count = 9) then
                      display_sum <= "0111111";
                  else
                      display_sum <= "0000000";
                  end if;

              when EW_Red_NS_Yellow =>
                  if (count = 1) then
                      display_sum <= "0000110";
                  elsif (count = 2) then
                      display_sum <= "0111111";
                  else
                      display_sum <= "0000000";
                  end if;
              end case;
          elsif  Decode = "0111" then
             if(pedestrian_crosswalk_EW = '0') then
                  display_sum <= "0111111";
             else
                  display_sum <= "0000110";
             end if;
          else
              display_sum <= "0000000";
          end if;
      elsif(clk_1000Hz = '0') then
          select_segment <= '1';
          if Decode = "1111" then
          case(current_state) is
              when idle =>
                 display_sum <= "0000000";
              when EW_Green_NS_Red =>
                  display_sum <= "0111111";
              when EW_Yellow_NS_Red =>
                  display_sum <= "0000110";
              when EW_Red_NS_Green =>
                  display_sum <= "1011011";
              when EW_Red_NS_Yellow =>
                  display_sum <= "1001111";
              end case;

          elsif  Decode = "0111" then
             if(pedestrian_crosswalk_NS = '0') then
                  display_sum <= "0111111";
             else
                  display_sum <= "0000110";
             end if;
          else
              display_sum <= "0000000";
          end if;
      end if;
      end process;

      pedestrian_signal : process(current_state) is
      begin
           case(current_state) is
              when idle =>
                  pedestrian_crosswalk_EW <= '0';
                  pedestrian_crosswalk_NS <= '0';

              when EW_Green_NS_Red =>
```

```
                pedestrian_crosswalk_EW <= '1';
                pedestrian_crosswalk_NS <= '0';

            when EW_Yellow_NS_Red =>
                pedestrian_crosswalk_EW <= '0';
                pedestrian_crosswalk_NS <= '0';
            when EW_Red_NS_Green =>
                pedestrian_crosswalk_EW <= '0';
                pedestrian_crosswalk_NS <= '1';
            when EW_Red_NS_Yellow =>
                pedestrian_crosswalk_EW <= '0';
                pedestrian_crosswalk_NS <= '0';

        end case;
    end process;
end architecture;
```

Design code for Clock_Divider_1hz

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


entity clk_divider_1hz is
    Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC);
end clk_divider_1hz;

architecture Behavioral of clk_divider_1hz is
signal clock_out : std_logic := '0';
signal count : integer := 1;
constant divider_1hz : natural := 62500000;
begin
    process(clk_in, clock_out)
    begin
        if clk_in='1' and clk_in'event then
            count <= count + 1;
            if(count = divider_1hz) then
                clock_out <= NOT clock_out;
                count <= 1;
            end if;
        end if;
    clk_out <= clock_out;
    end process;
end Behavioral;
```

Design code for Clock_Divider_1000hz

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


entity clk_divider_1000Hz is
    Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC);
end clk_divider_1000Hz;

architecture Behavioral of clk_divider_1000Hz is
signal clock_out : std_logic := '0';
signal count : integer := 1;
constant divider_1000hz : natural := 62500;
begin
    process(clk_in, clock_out)
    begin
        if clk_in='1' and clk_in'event then
            count <= count + 1;
            if(count = divider_1000hz) then
                clock_out <= NOT clock_out;
                count <= 1;
            end if;
        end if;
    clk_out <= clock_out;
    end process;
end Behavioral;
```

Design code for KeyPad_Decoder

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

entity Decoder is
    Port (clk : in  STD_LOGIC;
          Row : in  STD_LOGIC_VECTOR (3 downto 0);
          Col : out  STD_LOGIC_VECTOR (3 downto 0);
          DecodeOut : out  STD_LOGIC_VECTOR (3 downto 0));
end Decoder;

architecture Behavioral of Decoder is

signal sclk :STD_LOGIC_VECTOR(19 downto 0);
begin
    process(clk)
        begin
        if clk'event and clk = '1' then
            -- 1ms
            if sclk = "00011000011010100000" then
                --C1
                Col<= "0111";
                sclk <= sclk+1;
            -- check row pins
            elsif sclk = "00011000011010101000" then
                --R1
                if Row = "0111" then
                    DecodeOut <= "0001";    --1
                --R2
                elsif Row = "1011" then
                    DecodeOut <= "0100"; --4
                --R3
                elsif Row = "1101" then
                    DecodeOut <= "0111"; --7
                --R4
                elsif Row = "1110" then
                    DecodeOut <= "0000"; --0
                end if;
                sclk <= sclk+1;
            -- 2ms
            elsif sclk = "00110000110101000000" then
                --C2
                Col<= "1011";
                sclk <= sclk+1;
            -- check row pins
            elsif sclk = "00110000110101001000" then
                --R1
                if Row = "0111" then
                    DecodeOut <= "0010"; --2
                --R2
                elsif Row = "1011" then
                    DecodeOut <= "0101"; --5
                --R3
                elsif Row = "1101" then
                    DecodeOut <= "1000"; --8
                --R4
                elsif Row = "1110" then
                    DecodeOut <= "1111"; --F
                end if;
                sclk <= sclk+1;
            --3ms
            elsif sclk = "01001001001111100000" then
                --C3
                Col<= "1101";
                sclk <= sclk+1;
            -- check row pins
            elsif sclk = "01001001001111101000" then
                --R1
                if Row = "0111" then
                    DecodeOut <= "0011"; --3
                --R2
                elsif Row = "1011" then
                    DecodeOut <= "0110"; --6
                --R3
                elsif Row = "1101" then
                    DecodeOut <= "1001"; --9
                --R4
                elsif Row = "1110" then
                    DecodeOut <= "1110"; --E
                end if;
                sclk <= sclk+1;
            --4ms
            elsif sclk = "01100001101010000000" then
                --C4
                Col<= "1110";
                sclk <= sclk+1;
            -- check row pins
            elsif sclk = "01100001101010001000" then
                --R1
                if Row = "0111" then
```

```
                    DecodeOut <= "1010"; --A
                --R2
                elsif Row = "1011" then
                    DecodeOut <= "1011"; --B
                --R3
                elsif Row = "1101" then
                    DecodeOut <= "1100"; --C
                --R4
                elsif Row = "1110" then
                    DecodeOut <= "1101"; --D
                end if;
                sclk <= "00000000000000000000";
            else
                sclk <= sclk+1;
            end if;
        end if;
    end process;

end Behavioral;
```

2. Simulations

Hence there are two modes in design, involving normal mode and night mode. The part has two testbenches to show the two modes' results with the situations of illegal cars crossing the road. The count number in the clock divider changes to 1 to show full simulation results. The simulations of the red light camera feature will be shown in both modes. In the normal mode testbench, the states count the clock time as the delay time in design. At first, the direction signal is '0', the traffic light signal shows the light change in EW direction. During this period, When an illegal_car_detector signal shows '1' in related directions, the external_camera signal will show '1'. After that, the direction signal is '1' and repeats the red light feature again. In the night mode testbench, the direction changes as in the normal mode to show the traffic light change. The state only changes when the car_detector shows '1' in related directions. The change has no delay time and will remain the new state during the period. The waveform also shows that red light cameras also work in the night mode.

Normal mode Testbench

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

entity tb_traffic is
end entity;

architecture tb_behaviour of tb_traffic is
    component Traffic_system is
    Port(clk : in std_logic;
         reset : in std_logic;
         direction_select : in std_logic;
         mode_switch : in std_logic;
         car_detector : in std_logic_vector(1 downto 0);
         illegal_car_detector : in std_logic_vector(1 downto 0);
         external_camera : out std_logic_vector(1 downto 0);
         traffic_lights : out std_logic_vector(2 downto 0);
```

```
            state_signal : out std_logic_vector(1 downto 0)
            );
    end component;

    component clk_divider
           Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC);
    end component;
    signal clk_in_tb : std_logic;
    signal clk_out_tb : std_logic;
    signal reset_tb : std_logic;
    signal direction_select_tb : std_logic;
    signal mode_switch_tb : std_logic;
    signal car_detector_tb : std_logic_vector(1 downto 0);
    signal illegal_car_detector_tb : std_logic_vector(1 downto 0);
    signal external_camera_tb : std_logic_vector(1 downto 0);
    signal traffic_lights_tb : std_logic_vector(2 downto 0);
    signal state_signal_tb : std_logic_vector(1 downto 0);
    constant clock_period_tb : time := 4ns;
begin
    vending_tb : Traffic_system port map(clk => clk_in_tb, reset => reset_tb,
direction_select => direction_select_tb,
                                   mode_switch => mode_switch_tb, car_detector
=> car_detector_tb, illegal_car_detector => illegal_car_detector_tb,
                                  external_camera => external_camera_tb,
traffic_lights => traffic_lights_tb, state_signal => state_signal_tb);
    divider : clk_divider port map(clk_in => clk_in_tb, clk_out => clk_out_tb);

    clock : process
    begin
        clk_in_tb <= '0';
        wait for clock_period_tb/2;
        clk_in_tb <= '1';
        wait for clock_period_tb/2;
    end process;

    stimulus : process
    begin
        reset_tb <= '1';
        wait for 2ns;
        reset_tb <= '0';
        mode_switch_tb <= '0';
        direction_select_tb <= '0';
        illegal_car_detector_tb <= "00";
        wait for 12ns;
        illegal_car_detector_tb <= "01";
        wait for 6ns;
        illegal_car_detector_tb <= "00";
        wait for 64ns;
        illegal_car_detector_tb <= "10";
        wait for 6ns;
        illegal_car_detector_tb <= "00";
        direction_select_tb <= '1';
        wait for 10ns;
        illegal_car_detector_tb <= "01";
        wait for 4ns;
        illegal_car_detector_tb <= "00";
        wait for 48ns;
```

```
            illegal_car_detector_tb <= "10";
            wait for 6ns;
            illegal_car_detector_tb <= "00";
            wait for 20ns;

        end process;
    end architecture;
```

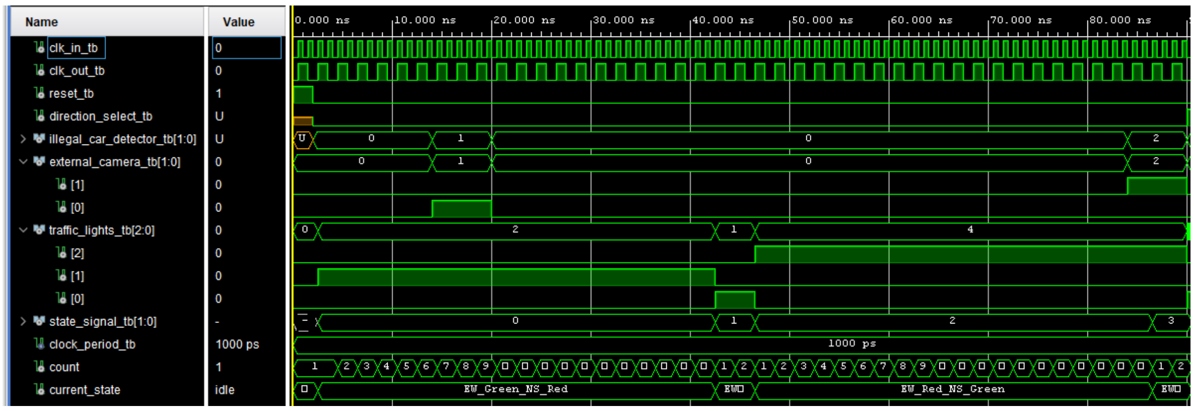


Figure2 : Simulation waveform of normal mode when direction_select = '0'

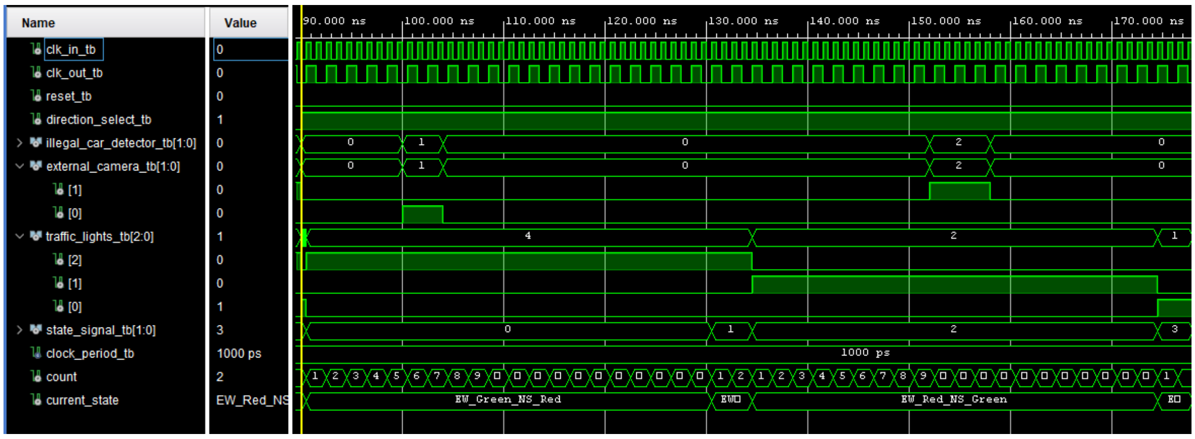


Figure3 : Simulation waveform of normal mode when direction_select = '1'

Night mode Testbench

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

entity tb_traffic is
end entity;

architecture tb_behaviour of tb_traffic is
    component Traffic_system is
    Port(clk : in std_logic;
         reset : in std_logic;
         direction_select : in std_logic;
         mode_switch : in std_logic;
         car_detector : in std_logic_vector(1 downto 0);
         illegal_car_detector : in std_logic_vector(1 downto 0);
         external_camera : out std_logic_vector(1 downto 0);
         traffic_lights : out std_logic_vector(2 downto 0);
         state_signal : out std_logic_vector(1 downto 0)
         );
    end component;

    component clk_divider
           Port ( clk_in : in STD_LOGIC;
           clk_out : out STD_LOGIC);
    end component;
    signal clk_in_tb : std_logic;
    signal clk_out_tb : std_logic;
    signal reset_tb : std_logic;
    signal direction_select_tb : std_logic;
    signal mode_switch_tb : std_logic;
    signal car_detector_tb : std_logic_vector(1 downto 0);
    signal illegal_car_detector_tb : std_logic_vector(1 downto 0);
    signal external_camera_tb : std_logic_vector(1 downto 0);
    signal traffic_lights_tb : std_logic_vector(2 downto 0);
    signal state_signal_tb : std_logic_vector(1 downto 0);
    constant clock_period_tb : time := 4ns;
begin
    vending_tb : Traffic_system port map(clk => clk_in_tb, reset => reset_tb,
direction_select => direction_select_tb,
                                   mode_switch => mode_switch_tb, car_detector
=> car_detector_tb, illegal_car_detector => illegal_car_detector_tb,
                                  external_camera => external_camera_tb,
traffic_lights => traffic_lights_tb, state_signal => state_signal_tb);
    divider : clk_divider port map(clk_in => clk_in_tb, clk_out => clk_out_tb);

    clock : process
    begin
        clk_in_tb <= '0';
        wait for clock_period_tb/2;
        clk_in_tb <= '1';
        wait for clock_period_tb/2;
    end process;

    stimulus : process
    begin
        reset_tb <= '1';
        wait for 2ns;
        reset_tb <= '0';
        mode_switch_tb <= '1';
        direction_select_tb <= '0';
        car_detector_tb <= "00";
        illegal_car_detector_tb <= "00";
        wait for 40ns;
        car_detector_tb <= "01";
        wait for 15ns;
        car_detector_tb <= "10";
        illegal_car_detector_tb <= "10";
        wait for 6ns;
        car_detector_tb <= "00";
        illegal_car_detector_tb <= "00";
        wait for 20ns;
        direction_select_tb <= '1';
        car_detector_tb <= "00";
        illegal_car_detector_tb <= "00";
        wait for 40ns;
        car_detector_tb <= "01";
        wait for 15ns;
        car_detector_tb <= "10";
        illegal_car_detector_tb <= "10";
        wait for 6ns;
        car_detector_tb <= "00";
        illegal_car_detector_tb <= "00";
        wait for 20ns;

    end process;
end architecture;
```

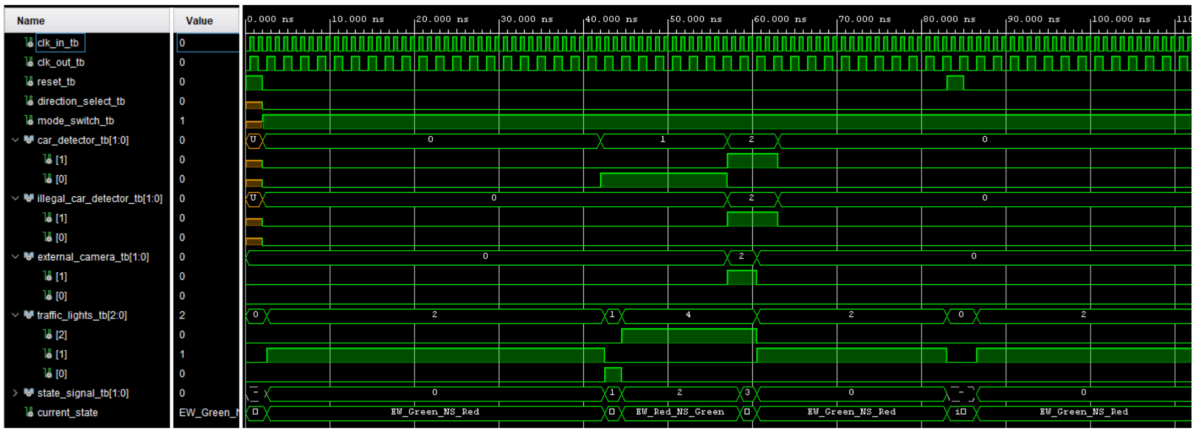


Figure4 : Simulation waveform of night mode when direction_select = '0'

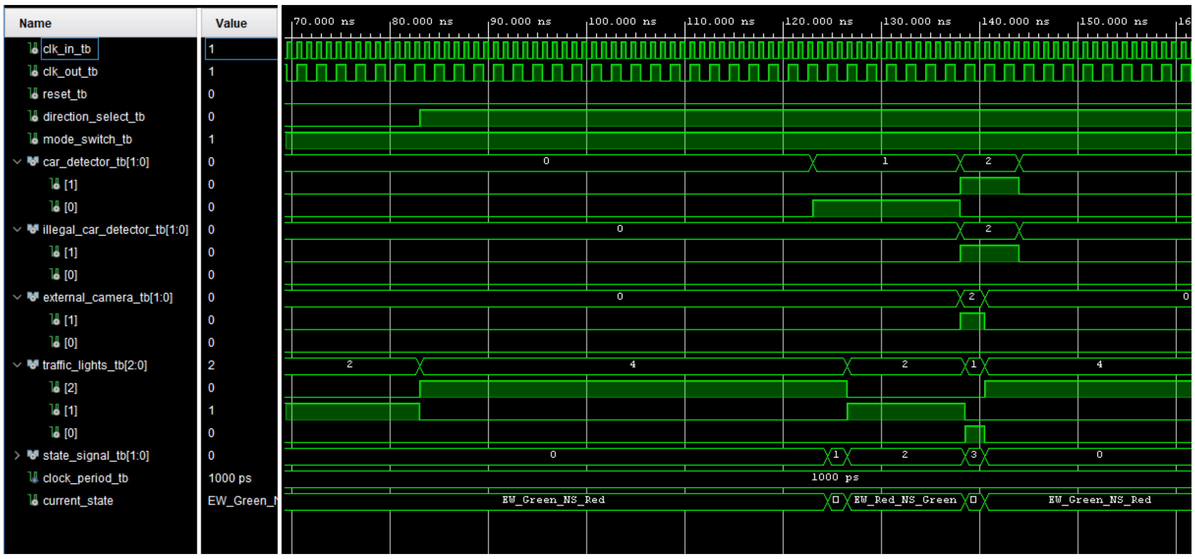
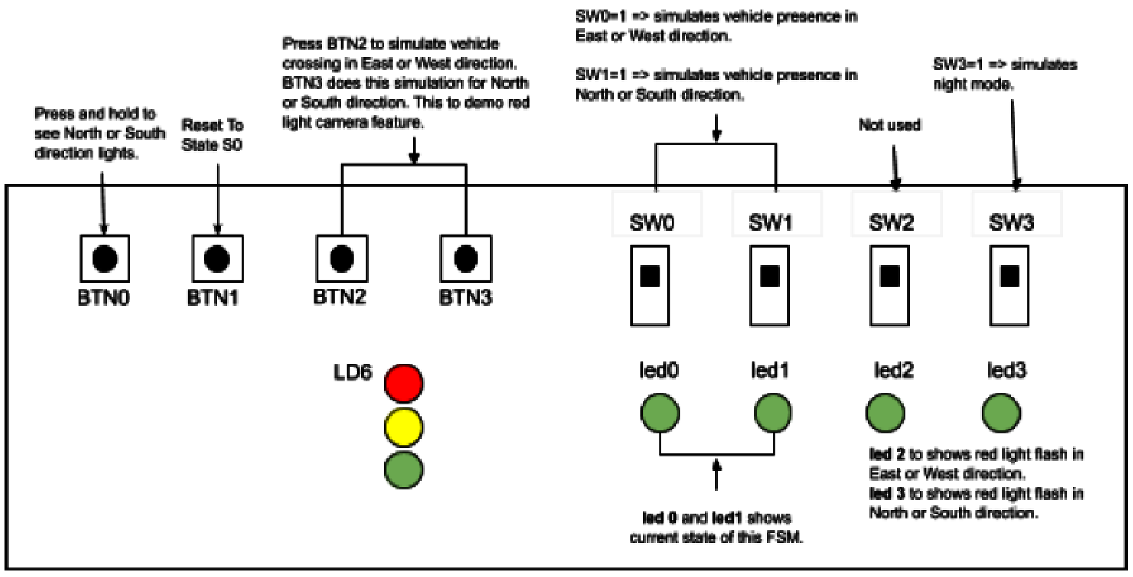


Figure5 : Simulation waveform of normal mode when direction_select = '1'

3. Constraints and Tests

Constraints

The constraint file assigns all signals to different pin according to the figure shown as below:



Press and hold to see North or South direction lights.
Reset To State S0
Press BTN2 to simulate vehicle crossing in East or West direction. BTN3 does this simulation for North or South direction. This to demo red light camera feature.
SW0=1 => simulates vehicle presence in East or West direction.
SW1=1 => simulates vehicle presence in North or South direction.
Not used
SW3=1 => simulates night mode.
BTN0
BTN1
BTN2
BTN3
SW0
SW1
SW2
SW3
LD6
led0
led1
led2
led3
led 0 and led1 shows current state of this FSM.
led 2 to shows red light flash in East or West direction.
led 3 to shows red light flash in North or South direction.

```
set_property -dict {PACKAGE_PIN K17 IOSTANDARD LVCMOS33} [get_ports { clk }];
create_clock -period 8.000 -name sys_clk_pin -waveform {0.000 4.000} -add
[get_ports clk];

set_property -dict { PACKAGE_PIN K18 IOSTANDARD LVCMOS33 } [get_ports {
direction_select }]; #btn[0]
set_property -dict {PACKAGE_PIN P16 IOSTANDARD LVCMOS33} [get_ports { reset }];
#btn[1]

set_property -dict {PACKAGE_PIN K19 IOSTANDARD LVCMOS33} [get_ports {
illegal_car_detector[1]}]; #btn[2]
set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports {
illegal_car_detector[0]}];

set_property -dict {PACKAGE_PIN G15 IOSTANDARD LVCMOS33} [get_ports {
car_detector[1] }];
set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports {
car_detector[0] }];

set_property -dict {PACKAGE_PIN T16 OSTANDARD LVCMOS33 } [get_ports {
mode_switch }];

set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports
{traffic_lights[2] }]; #led6_r
set_property -dict {PACKAGE_PIN F17 IOSTANDARD LVCMOS33} [get_ports
{traffic_lights[1] }]; #led6_g
set_property -dict {PACKAGE_PIN M17 IOSTANDARD LVCMOS33} [get_ports
{traffic_lights[0] }]; #led6_b

set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports
state_signal[0]}];#led0
set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33} [get_ports
{state_signal[1]}];#led1
set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports
{external_camera[1]}];#led2
set_property -dict {PACKAGE_PIN D18 IOSTANDARD LVCMOS33} [get_ports
{external_camera[0]}];#led3
```

```
set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {
display_sum[0] }];
set_property -dict {PACKAGE_PIN W15 IOSTANDARD LVCMOS33} [get_ports {
display_sum[1] }];
set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports {
display_sum[2] }];
set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports {
display_sum[3] }];
set_property -dict {PACKAGE_PIN T14 IOSTANDARD LVCMOS33} [get_ports {
display_sum[4] }];
set_property -dict {PACKAGE_PIN T15 IOSTANDARD LVCMOS33} [get_ports {
display_sum[5] }];
set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {
display_sum[6] }];
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {
select_segment }];


set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Row[0] }];
set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Row[1] }];
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Row[2] }];
set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Row[3] }];
set_property -dict { PACKAGE_PIN V13 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Col[0] }];
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Col[1] }];
set_property -dict { PACKAGE_PIN T17 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Col[2] }];
set_property -dict { PACKAGE_PIN Y17 IOSTANDARD LVCMOS33 } [get_ports {
KeyPad_Col[3] }];
```

Test Pictures

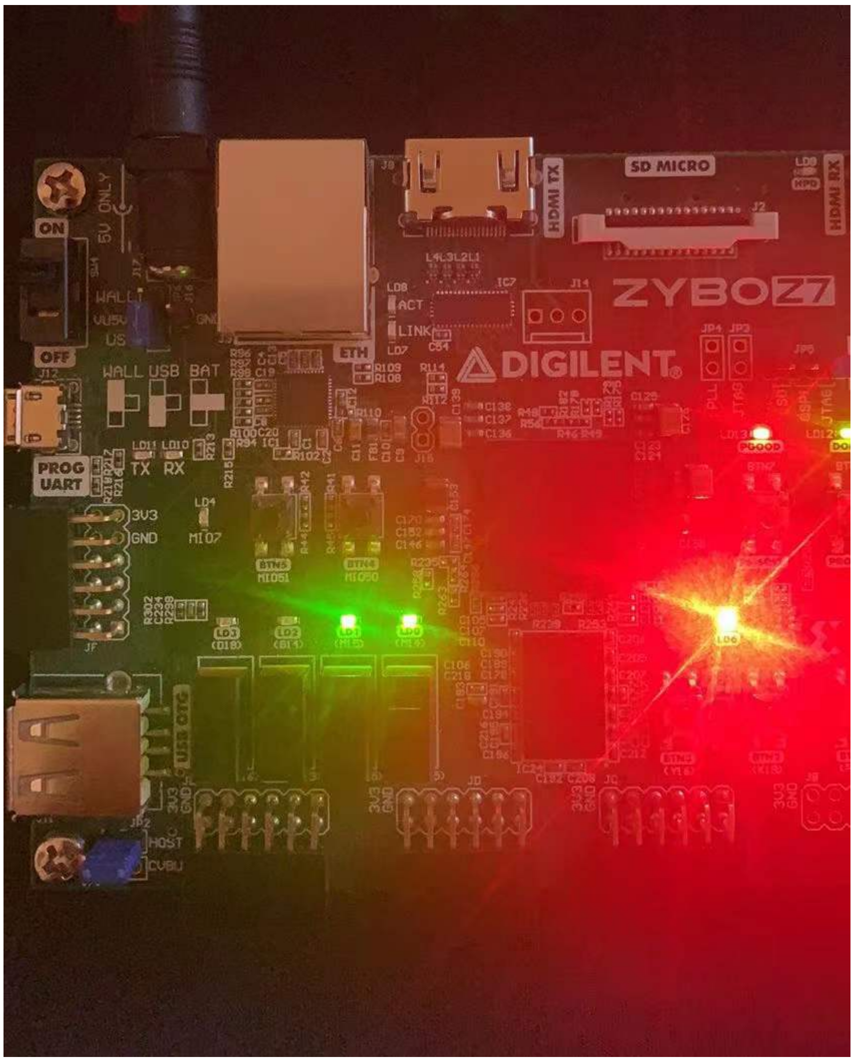


Figure6: Red traffic light

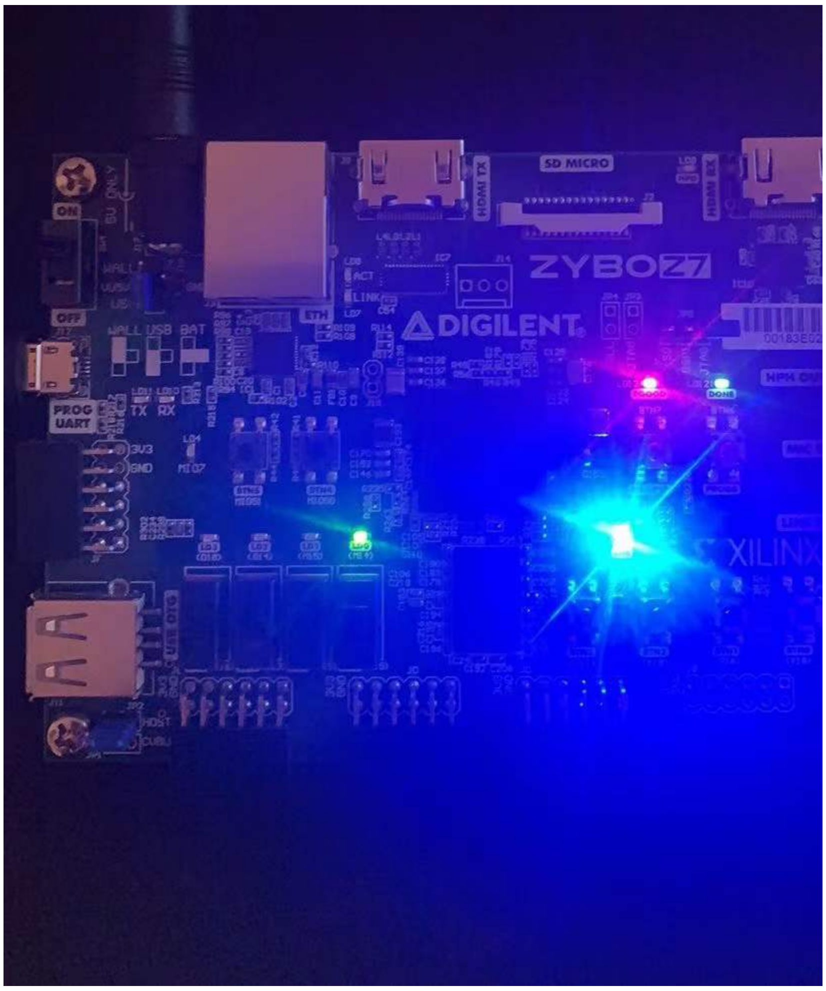


Figure7: blue traffic light

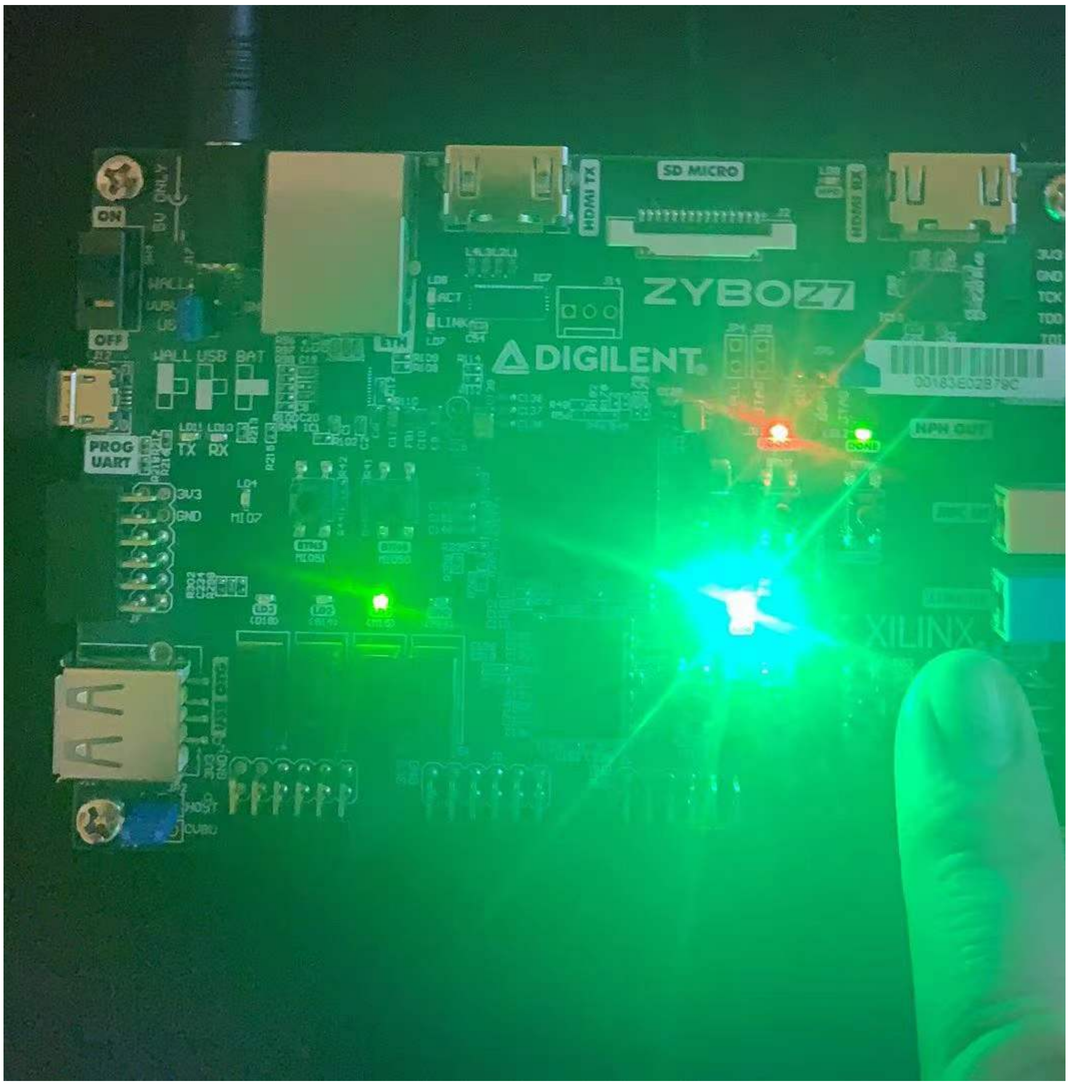


Figure8: Green traffic light with "direction_select"='1'

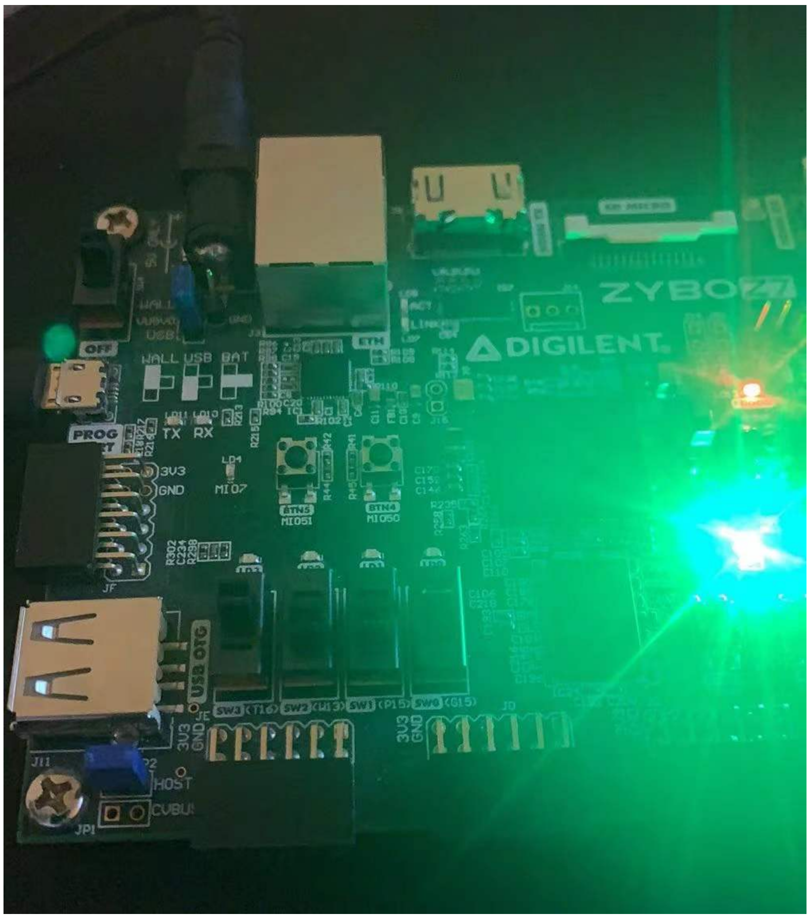


Figure9: Vehicle approaching the EW direction in night mode

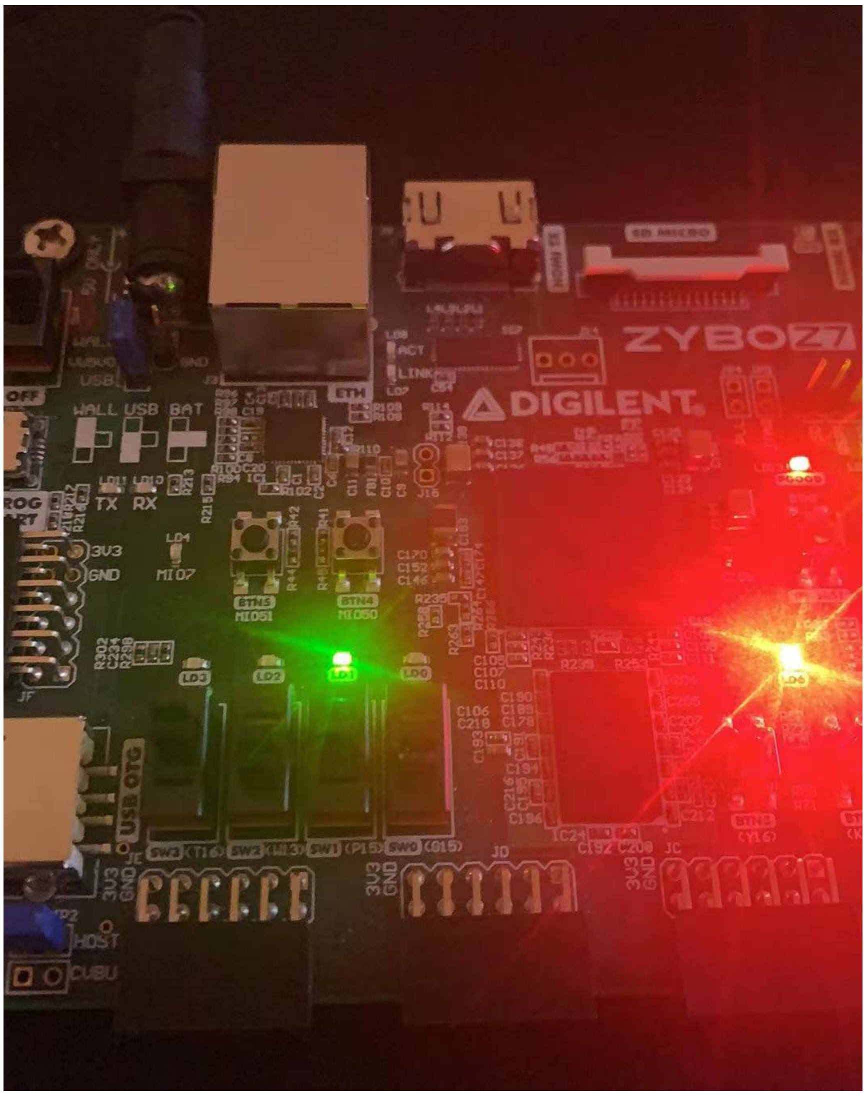


Figure10: Vehicle approaching the NS direction in night mode

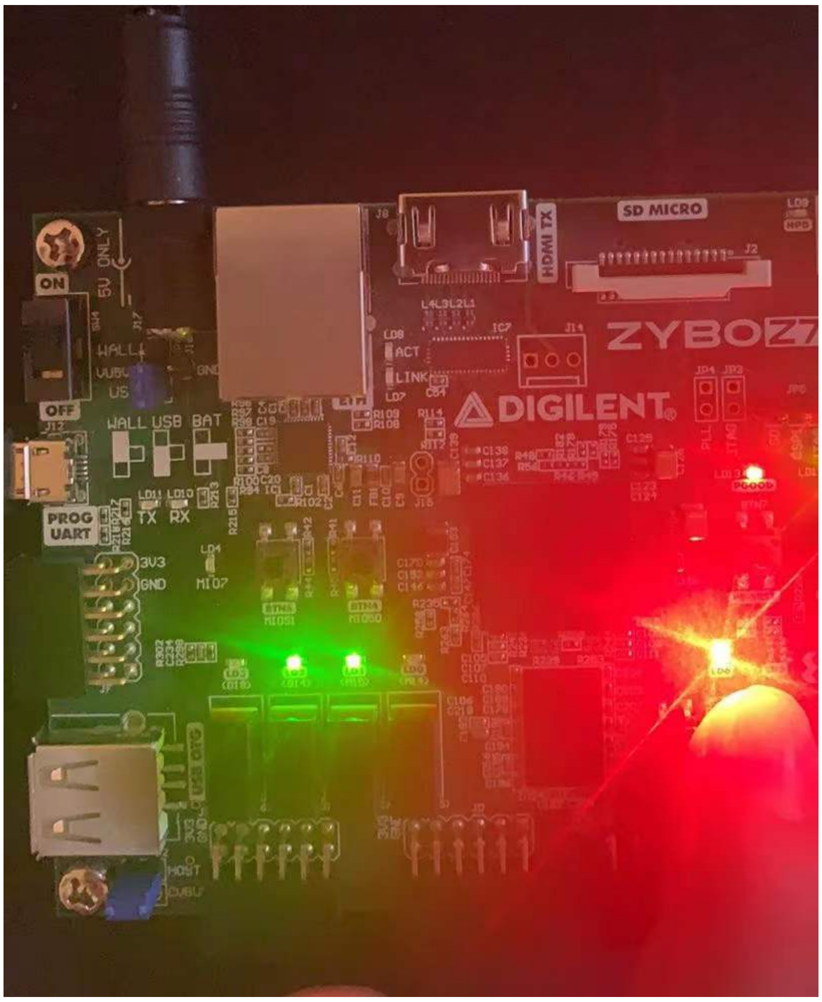


Figure11: Vehicle approaching the EW direction and encountering a red light

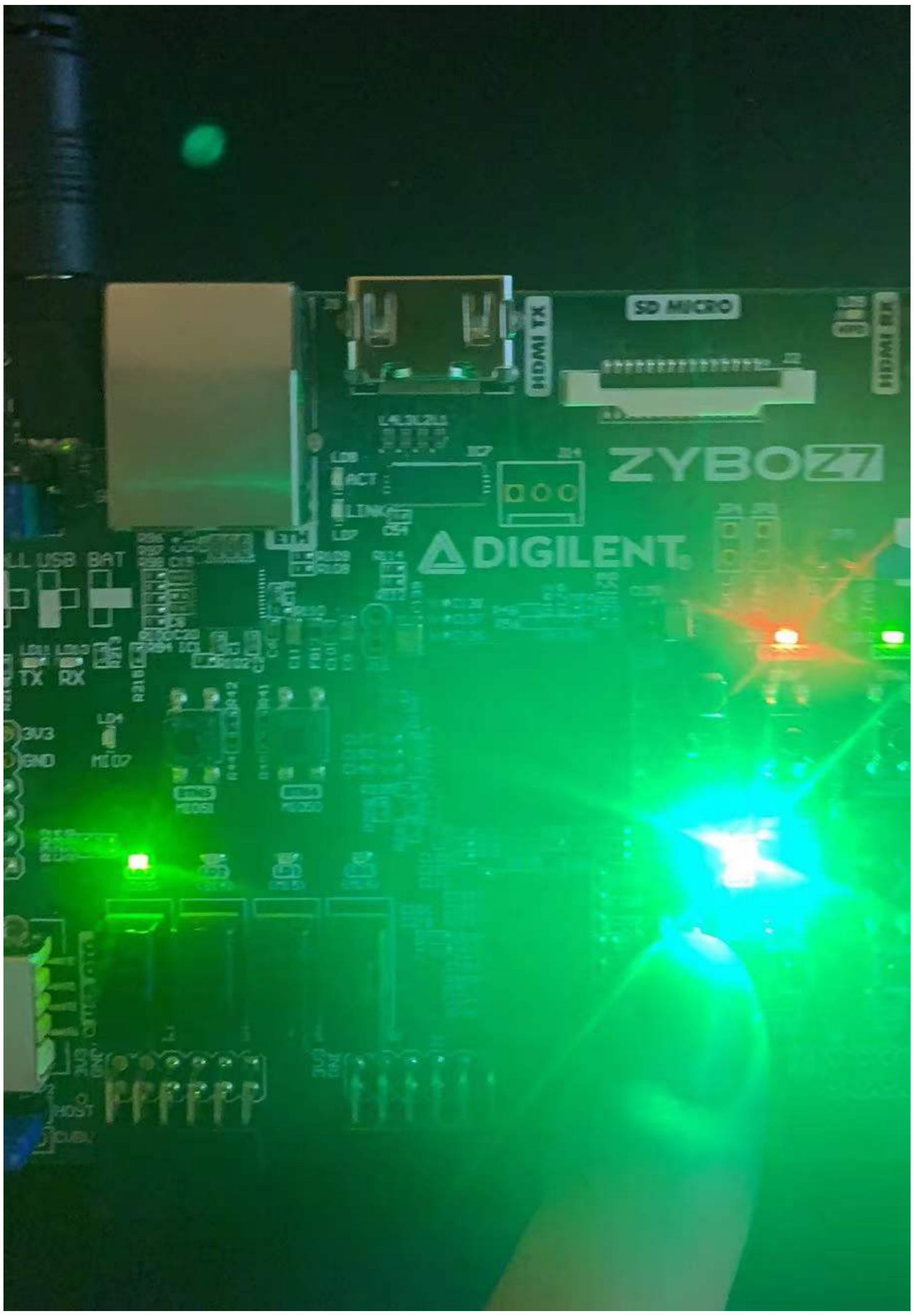


Figure12: Vehicle approaching the NS direction and encountering a red light

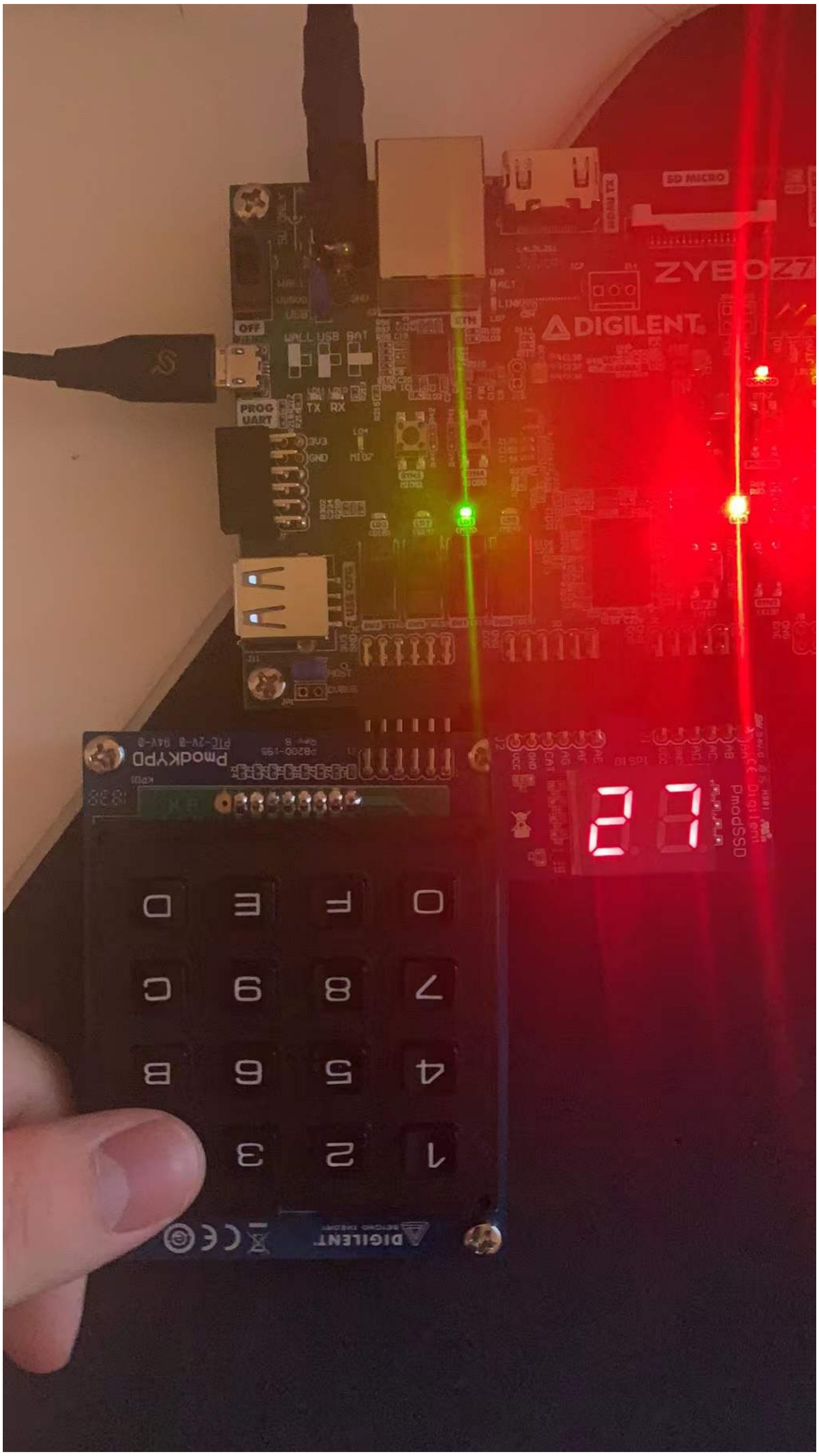


FIgure13: Display the time remaining and the current system state

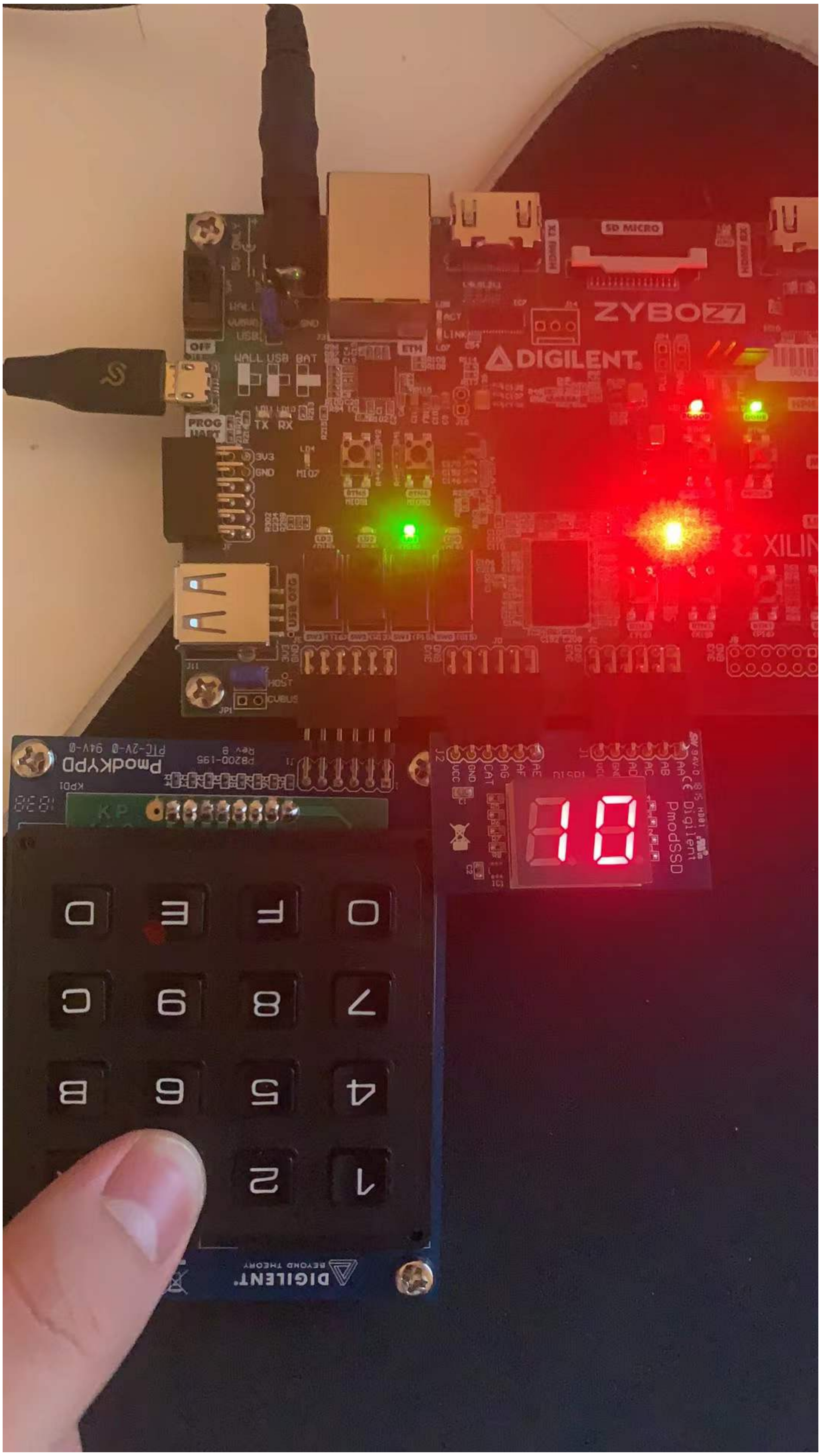


FIgure14: Display the pedestrian crosswalk signals

4. Conclusion

The objective of this project was to design a Traffic Light Controller for use in road intersections, to implement the design by using the IEEE Standard VHDL Language, to simulate the system using the Vivado software, and to be able to understand and explain the output results. The report clearly shows that the project was successful in finishing six lab3 requirements.