

VHDL code of the clock divider of 1 HZ:

```
-----  
-- Company: Department of Electrical and Computer Engineering, University of Alberta  
-- Engineer: Xinyue Chen  
-- Create Date: 2020/11/23 20:29:15  
-- Design Name: traffic intersection controller  
-- Module Name: clk_divider - Behavioral  
-- Project Name: traffic intersection controller  
-- Target Devices: ZYBO Z7-10  
-- Tool Versions:  
-- Description:  
-----
```

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

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

```
architecture Behavioral of clk_divider is  
    signal clock_out : std_logic := '0';  
    signal count : integer := 1;  
    constant cnt_number : integer := 62500000;  
    --avoid using magic number.  
begin  
    process(clk_in)  
    begin  
        if clk_in='1' and clk_in'event then  
            count <= count + 1;  
            if(count = cnt_number) then  
                --This will divide the system clock by 125000000 to generate a 1HZ clock  
                -- While in simulation process, we let count=1 to divide the simulation clock by 2.  
                clock_out <= NOT clock_out;  
                count <= 1;  
            end if;  
        end if;  
        clk_out <= clock_out;  
    end process;  
end Behavioral;
```

VHDL code of seven segment display:

```

-----
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/23 20:34:05
-- Design Name: traffic intersection controller
-- Module Name: seven_segment - Behavioral
-- Project Name: traffic intersection controller
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
-----

```

```

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

entity seven_segment is
    generic (clk_cnt_bits : integer:=8);
    port(clk7: in STD_LOGIC;
         rst: in STD_LOGIC;
         countdown: in STD_LOGIC_VECTOR(7 downto 0);
         presentstate: in STD_LOGIC_VECTOR(1 downto 0);
         pedestrian_crosswalk_EW: in STD_LOGIC;
         pedestrian_crosswalk_NS: in STD_LOGIC;
         --For pedestrian_crosswalk_sign demonstration.
         row: in STD_LOGIC_VECTOR(3 downto 0);
         col: out STD_LOGIC_VECTOR(3 downto 0);
         --kypd 4 rows and 4 columns to control which button to be pressed on keypad.
         chose_segement: out STD_LOGIC;
         seven_segment: out STD_LOGIC_VECTOR(6 downto 0));
end seven_segment;

architecture arch of seven_segment is

    subtype digit_type is integer range 0 to 10;
    type digits_type is array (1 downto 0) of digit_type;
    -- To creat an array of 2, each one is an integer range from 0 to 10.
    signal digit : digit_type;
    --The integer signal range from 0 to 10, used for seven segment display.
    signal digits : digits_type;
    signal clk_cnt :std_logic_vector(clk_cnt_bits-1 downto 0):="00000000";
    --We use the most significant bit (MSB) of clk_cnt to switch between left and right digit on seven segment display.

```

```

begin

count_proc : process(clk7) is
begin
    if rising_edge(clk7) then
        if rst = '1' then
            clk_cnt <= (others => '0');
        else
            clk_cnt <= clk_cnt + 1;
        end if;
    end if;
end process count_proc;

chose_segement <= clk_cnt(clk_cnt'high);
-- chose_segement is '0' when most significant bit of clk_cnt is '0'.
-- chose_segement is '1' when most significant bit of clk_cnt is '1'.
digit <= digits(0) when clk_cnt(clk_cnt'high) = '0' else digits(1);
-- digit is equal to digits(0) when most significant bit of clk_cnt is '0'.
-- digit is equal to digits(1) when most significant bit of clk_cnt is '1'.
col <="0111";
--Set the first column of keypad to be selected.

kypd_proc: process (countdown,presentstate,pedestrian_crosswalk_EW,row,pedestrian_crosswalk_NS) is
begin

    if( presentstate="00" and countdown=X"09" and row="1111") then
        --If no row of keypad is selected, we demonstrate presentstate and countdown from 9 to 0 on seven
        segment display.
        digits(1) <= 0;
        digits(0) <= 9;
    elsif( presentstate="00" and countdown=X"08" and row="1111") then
        digits(1) <= 0;
        digits(0) <= 8;
    elsif( presentstate="00" and countdown=X"07" and row="1111") then
        digits(1) <= 0;
        digits(0) <= 7;
    elsif( presentstate="00" and countdown=X"06" and row="1111") then
        digits(1) <= 0;
        digits(0) <= 6;
    elsif( presentstate="00" and countdown=X"05" and row="1111") then
        digits(1) <= 0;
        digits(0) <= 5;
    elsif( presentstate="00" and countdown=X"04" and row="1111") then
        digits(1) <= 0;

```

```

digits(0) <= 4;
elseif( presentstate="00" and countdown=X"03" and row="1111") then
digits(1) <= 0;
digits(0) <= 3;
elseif( presentstate="00" and countdown=X"02" and row="1111") then
digits(1) <= 0;
digits(0) <= 2;
elseif( presentstate="00" and countdown=X"01" and row="1111") then
digits(1) <= 0;
digits(0) <= 1;
elseif( presentstate="00" and countdown=X"00" and row="1111") then
digits(1) <= 0;
digits(0) <= 0;

elseif(presentstate="01" and row="1111") then
digits(1) <= 1;
--No need for demonstration of countdown/time_remain in this state.
digits(0) <= 10;

elseif(presentstate="10" and countdown=X"09" and row="1111") then
digits(1) <= 2;
digits(0) <= 9;
elseif( presentstate="10" and countdown=X"08" and row="1111") then
digits(1) <= 2;
digits(0) <= 8;
elseif( presentstate="10" and countdown=X"07" and row="1111") then
digits(1) <= 2;
digits(0) <= 7;
elseif( presentstate="10" and countdown=X"06" and row="1111") then
digits(1) <= 2;
digits(0) <= 6;
elseif( presentstate="10" and countdown=X"05" and row="1111") then
digits(1) <= 2;
digits(0) <= 5;
elseif( presentstate="10" and countdown=X"04" and row="1111") then
digits(1) <= 2;
digits(0) <= 4;
elseif( presentstate="10" and countdown=X"03" and row="1111") then
digits(1) <= 2;
digits(0) <= 3;
elseif( presentstate="10" and countdown=X"02" and row="1111") then
digits(1) <= 2;
digits(0) <= 2;
elseif( presentstate="10" and countdown=X"01" and row="1111") then

```

```

digits(1) <= 2;
digits(0) <= 1;
elsif( presentstate="10" and countdown=X"00" and row="1111") then
digits(1) <= 2;
digits(0) <= 0;

elsif(presentstate="11" and countdown=X"01" and row="1111") then
digits(1) <= 3;
digits(0) <= 1;
elsif(presentstate="11" and countdown=X"00" and row="1111") then
digits(1) <= 3;
digits(0) <= 0;

elsif (pedestrian_crosswalk_EW='0'and pedestrian_crosswalk_NS='0' and row="0111") then
--If first row of keypad is selected, we demonstrate pedestrain walk/stop sign on seven segment display.
digits(0) <= 0;
digits(1) <= 0;
elsif(pedestrian_crosswalk_EW='1' and pedestrian_crosswalk_NS='0' and row="0111") then
digits(0) <= 1;
digits(1) <= 0;
elsif (pedestrian_crosswalk_EW='0'and pedestrian_crosswalk_NS='1' and row="0111") then
digits(0) <= 0;
digits(1) <= 1;
else
digits(0) <= 10;
digits(1) <= 10;
end if;
end process kypd_proc;

display_PROC : process(digit) is
begin
case digit is
--display the digit on seven segment display.
when 0 => seven_segment <= "0111111";
when 1 => seven_segment <= "0000110";
when 2 => seven_segment <= "1011011";
when 3 => seven_segment <= "1001111";
when 4 => seven_segment <= "1100110";
when 5 => seven_segment <= "1101101";
when 6 => seven_segment <= "1111101";
when 7 => seven_segment <= "0000111";
when 8 => seven_segment <= "1111111";
when 9 => seven_segment <= "1101111";
when 10=> seven_segment <= "0000000";

```

```
        end case;
    end process display_PROC;
end arch;
```

VHDL code of traffic light controller:

```
-----
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/23 20:18:06
-- Design Name: traffic intersection controller
-- Module Name: traffic_light_controller - Behavioral
-- Project Name: traffic intersection controller
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
-----

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

entity traffic_light_controller is
    Port (view_NS_light: in STD_LOGIC;
          clk:in STD_LOGIC;
          reset: in STD_LOGIC;
          carcrossing_EW: in STD_LOGIC;
          carcrossing_NS: in STD_LOGIC;
          carpresence_EW: in STD_LOGIC;
          carpresence_NS: in STD_LOGIC;
          night_mode: in STD_LOGIC;
          keypad_hold_in: inout STD_LOGIC_VECTOR(3 downto 0);
          keypad_hold_out: inout STD_LOGIC_VECTOR(3 downto 0);
          --keypad_hold_in and keypad_hold_out are connected to row and col of component seven_segment.
          --The two entity ports represents the 4 rows and 4 columns of Pmod kypd.
          FSM_currentstate: out STD_LOGIC_VECTOR(1 downto 0);
          redlight_flash_EW: out STD_LOGIC;
          redlight_flash_NS: out STD_LOGIC;
          --Demonstrating 1 sec pulse if there is car crossing when redlight_NS or redlight_EW on.
          traffic_light: out STD_LOGIC_VECTOR(2 downto 0);
          select_segment: out STD_LOGIC;
          segment_display: out STD_LOGIC_VECTOR(6 downto 0)
    );
end traffic_light_controller;
```

architecture arch of traffic_light_controller is

component clk_divider is

Port (clk_in : in STD_LOGIC;

clk_out : out STD_LOGIC);

-- The 1HZ colck divider we use to demonstrate the result on FPGA board.

end component clk_divider;

component seven_segment is

port(clk7: in STD_LOGIC;

rst: in STD_LOGIC;

countdown: in STD_LOGIC_VECTOR(7 downto 0);

presentstate: in STD_LOGIC_VECTOR(1 downto 0);

pedestrian_crosswalk_EW: in STD_LOGIC;

pedestrian_crosswalk_NS: in STD_LOGIC;

row: in STD_LOGIC_VECTOR(3 downto 0);

col: out STD_LOGIC_VECTOR(3 downto 0);

chosed_segement: out STD_LOGIC;

seven_segment: out STD_LOGIC_VECTOR(6 downto 0));

end component seven_segment;

type state_type is

(EW_green_NS_red,EW_yellow_NS_green,EW_red_NS_green,EW_red_NS_yellow,EW_yellow_NS_red);

--Normal mode: transit from EW_green_NS_red to EW_yellow_NS_green to EW_red_NS_green to EW_red_NS_yellow

--Night mode: transit from EW_green_NS_red to EW_yellow_NS_red to EW_red_NS_green to EW_red_NS_yellow

--There are five states in total.

--However, at each mode, there are four states respectively.

--Thus the current state can be demonstrated on LED 1 and LED 2 in different mode.

signal current_state, next_state: state_type;

signal clock_out: std_logic;

signal count_time: std_logic_vector(7 downto 0):=X"00";

--To count the time delay at each state.

signal remain_time: std_logic_vector(7 downto 0):=X"14";

--To count the time remain at current state and is used for red/green light countdown on seven segment display.

signal red_flash_time_EW: std_logic_vector(7 downto 0):=X"00";

signal red_flash_time_NS: std_logic_vector(7 downto 0):=X"00";

-- This is used as sign signal to let the redlight_flash_EW/Ns <= '1' last for 1 sec.

signal state_signal: std_logic_vector(1 downto 0) := "00";

signal pedestrian_sign_EW: std_logic := '0';

```

signal pedestrian_sign_NS: std_logic := '0';
-- This is used to demonstrate the pedestrain walk/stop for '1' and '0' on seven segment display.
signal S0_light_enable: std_logic := '0';
signal S1_light_enable: std_logic := '0';
signal S2_light_enable: std_logic := '0';
signal S3_light_enable: std_logic := '0';
signal S0_delay_20s: std_logic := '0';
signal S1_delay_2s: std_logic := '0';
signal S2_delay_18s: std_logic := '0';
signal S3_delay_2s: std_logic := '0';
-- This is used as sign signal to let the FSM stay in current state for certain time period.
Begin

clock_divide: component clk_divider port map ( clk_in => clk,
                                             clk_out => clock_out);
segment_demo: component seven_segment port map (clk7 => clk,
                                             rst => reset,
                                             countdown => remain_time,
                                             presentstate => state_signal,
                                             pedestrian_crosswalk_EW => pedestrian_sign_EW,
                                             pedestrian_crosswalk_NS => pedestrian_sign_NS,
                                             col => keypad_hold_out(3 downto 0),
                                             row => keypad_hold_in(3 downto 0),
                                             chose_segement => select_segment,
                                             seven_segment => segment_display);

FSM_currentstate <= state_signal;

sequ_logic: process(reset,clock_out) is
begin
if (reset='1') then
current_state <= EW_green_NS_red;
elsif(rising_edge(clock_out)) then
current_state <= next_state;
end if;
end process sequ_logic;

delay_process:process(clock_out,S0_light_enable,S1_light_enable,S2_light_enable,S3_light_enable,count_time,remain_time,view_NS_light) is
begin
if(rising_edge(clock_out)) then
if(S0_light_enable='1' or S1_light_enable='1' or S2_light_enable='1' or S3_light_enable='1') then
count_time <= count_time + X"01";
remain_time <= remain_time - X"01";

```



```

if(count_time= X"13" and S0_light_enable='1') then
S0_delay_20s <='1';
--Time is up in S0 and is able to move to the next state.
S1_delay_2s <='0';
S2_delay_18s <='0';
S3_delay_2s <='0';
count_time <= X"00";
--Count to 19 secs to move to the next state, and the assignment current_state <= next_state takes 1 sec.
-- total time delay is 20 secs.
elsif(remain_time= X"00" and S0_light_enable='1') then
remain_time <= X"01";
--reset remain_time in next_state.
elsif(count_time= X"01" and S1_light_enable='1') then
S0_delay_20s <='0';
S1_delay_2s <='1';
--Time is up in S1 and is able to move to the next state.
S2_delay_18s <='0';
S3_delay_2s <='0';
count_time <= X"00";
--Count to 1 secs to move to the next state, and the assignment current_state <= next_state takes 1 sec.
-- total time delay is 2 secs.
elsif (remain_time= X"00" and S1_light_enable='1' and view_NS_light ='1') then
remain_time <= X"11";
elsif (remain_time= X"00" and S1_light_enable='1' and view_NS_light ='0') then
remain_time <= X"13";
--The reset remain_time in next_state for NS direction is 17 secs, while for EW direction is 19 sec.
elsif(count_time= X"11" and S2_light_enable='1') then
S0_delay_20s <='0';
S1_delay_2s <='0';
S2_delay_18s <='1';
--Time is up in S2 and is able to move to the next state.
S3_delay_2s <='0';
count_time <= X"00";
--Count to 17 secs to move to the next state, and the assignment current_state <= next_state takes 1 sec.
-- total time delay is 18 secs.
elsif(remain_time= X"00" and S2_light_enable='1') then
remain_time <= X"01";
elsif(count_time= X"01" and S3_light_enable='1') then
S0_delay_20s <='0';
S1_delay_2s <='0';
S2_delay_18s <='0';
S3_delay_2s <='1';
--Time is up in S3 and is able to move to the next state.
count_time <= X"00";

```

--Count to 1 secs to move to the next state, and the assignment current_state <= next_state takes 1 sec.

-- total time delay is 2 secs.

elsif(remain_time= X"00" and S3_light_enable='1') then

remain_time <= X"13";

else

S0_delay_20s <='0';

S1_delay_2s <='0';

S2_delay_18s <='0';

S3_delay_2s <='0';

end if;

end if;

if (carcrossing_EW='1') then

red_flash_time_EW <= red_flash_time_EW + X"01";

else

red_flash_time_EW <= X"00";

end if;

if (carcrossing_NS='1') then

red_flash_time_NS <= red_flash_time_NS + X"01";

else

red_flash_time_NS <= X"00";

end if;

--When carcrossing='1', at the rising_edge of clk_out signal, red_flash_time <= red_flash_time + 1.

-- This is used as sign signal to let the redlight_flash_EW/Ns <= '1' last for 1 sec.

end if;

end process delay_process;

comb_logic:

process(current_state,view_NS_light,night_mode,red_flash_time_NS,red_flash_time_EW,carpresence_NS,carpresence_EW,

S0_delay_20s,S1_delay_2s,S2_delay_18s,S3_delay_2s) is

begin

case current_state is

when EW_green_NS_red =>

state_signal <= "00";

pedestrian_sign_EW <= '0';

pedestrian_sign_NS <= '1';

S0_light_enable <= '1';

S1_light_enable <= '0';

S2_light_enable <= '0';

S3_light_enable <= '0';

redlight_flash_EW <='0';

if (view_NS_light = '0') then

```

traffic_light <= "010";
-- To switch between NS light and EW light.
else
traffic_light <= "001";
end if;
if (red_flash_time_NS = X"01") then
redlight_flash_NS <='1';
-- Let redlight_flash_NS <='1' last for 1 sec.
else
redlight_flash_NS <='0';
end if;
if (night_mode = '0') then
if (S0_delay_20s = '1') then
next_state <= EW_yellow_NS_green;
-- When S0_delay_20s = '1', the time in current_state is up and we move to next state.
else
next_state <= EW_green_NS_red;
end if;
elsif(carpresence_NS = '1') then
-- Starting night_mode function.
--We move to next_state when there is car presence in NS direction.
next_state <= EW_yellow_NS_red;
else
next_state <= EW_green_NS_red;
end if;

when EW_yellow_NS_green =>
state_signal <= "01";
pedestrian_sign_EW <= '0';
pedestrian_sign_NS <= '0';
S0_light_enable <= '0';
S1_light_enable <= '1';
S2_light_enable <= '0';
S3_light_enable <= '0';

redlight_flash_EW <='0';
redlight_flash_NS <='0';
if (view_NS_light = '0') then
traffic_light <= "100";
else
traffic_light <= "010";
end if;
if (S1_delay_2s = '1') then
next_state <= EW_red_NS_green;

```

```

else
next_state <= EW_yellow_NS_green;
end if;

when EW_red_NS_green =>
state_signal <= "10";
pedestrian_sign_EW <= '1';
pedestrian_sign_NS <= '0';
S0_light_enable <= '0';
S1_light_enable <= '0';
S2_light_enable <= '1';
S3_light_enable <= '0';

redlight_flash_NS <='0';
if (view_NS_light = '0') then
traffic_light <= "001";
else
traffic_light <= "010";

end if;
if (red_flash_time_EW = X"01") then
redlight_flash_EW <='1';
-- Let redlight_flash_EW <='1' last for 1 sec.
else
redlight_flash_EW <='0';
end if;
if (night_mode = '0') then
if (S2_delay_18s = '1') then
next_state <= EW_red_NS_yellow;
else
next_state <= EW_red_NS_green;
end if;
elsif(carpresence_EW = '1') then
next_state <= EW_red_NS_yellow;
else
next_state <= EW_red_NS_green;
end if;

when EW_red_NS_yellow =>
state_signal <= "11";
pedestrian_sign_EW <= '1';
pedestrian_sign_NS <= '0';
S0_light_enable <= '0';
S1_light_enable <= '0';

```

```

S2_light_enable <= '0';
S3_light_enable <= '1';

redlight_flash_NS <='0';
if (view_NS_light = '0') then
traffic_light <= "001";

else
traffic_light <= "100";
end if;
if (red_flash_time_EW = X"01") then
redlight_flash_EW <='1';
else
redlight_flash_EW <='0';
end if;
if (night_mode = '0') then
if (S3_delay_2s= '1') then
next_state <= EW_green_NS_red;
else
next_state <= EW_red_NS_yellow;
end if;
else
next_state <= EW_green_NS_red;
end if;

when EW_yellow_NS_red =>
pedestrian_sign_EW <= '0';
pedestrian_sign_NS <= '1';
S0_light_enable <= '0';
S1_light_enable <= '0';
S2_light_enable <= '0';
S3_light_enable <= '0';

redlight_flash_EW <='0';
if (view_NS_light = '0') then
traffic_light <= "100";
else
traffic_light <= "001";
end if;
if (red_flash_time_NS = X"01") then
redlight_flash_NS <='1';
else
redlight_flash_NS <='0';
end if;

```

```

if (night_mode = '1') then
state_signal <= "01";
next_state <=  EW_red_NS_green;
else
state_signal <= "--";
next_state <=  EW_yellow_NS_red;
end if;
end case;

end process comb_logic;
end arch;

```

VHDL testbench of traffic light controller:

```

-----
-- Company: Department of Electrical and Computer Engineering, University of Alberta
-- Engineer: Xinyue Chen
-- Create Date: 2020/11/25 18:03:31
-- Design Name: traffic intersection controller
-- Module Name: traffic_light_tb - Behavioral
-- Project Name: traffic intersection controller
-- Target Devices: ZYBO Z7-10
-- Tool Versions:
-- Description:
-----

```

```

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

```

```

entity traffic_light_tb is
end traffic_light_tb;

```

```

architecture Behavioral of traffic_light_tb is
component traffic_light_controller is

```

```

    Port (view_NS_light: in STD_LOGIC;
          clk:in STD_LOGIC;
          reset: in STD_LOGIC;
          carcrossing_EW: in STD_LOGIC;
          carcrossing_NS: in STD_LOGIC;
          carpresence_EW: in STD_LOGIC;
          carpresence_NS: in STD_LOGIC;
          night_mode: in STD_LOGIC;
          keypad_hold_in: inout STD_LOGIC_VECTOR(3 downto 0);

```

```

        keypad_hold_out: inout STD_LOGIC_VECTOR(3 downto 0);
        FSM_currentstate: out STD_LOGIC_VECTOR(1 downto 0);
        redlight_flash_EW: out STD_LOGIC;
        redlight_flash_NS: out STD_LOGIC;
        traffic_light: out STD_LOGIC_VECTOR(2 downto 0);
        select_segment: out STD_LOGIC;
        segment_display: out STD_LOGIC_VECTOR(6 downto 0)
    );
end component traffic_light_controller;

--input
signal view_NS_light_tb:std_logic:='0';
signal clk_tb:std_logic:='0';
signal reset_tb:std_logic:='0';
signal carcrossing_EW_tb:std_logic:='0';
signal carcrossing_NS_tb:std_logic:='0';
signal carpresence_EW_tb:std_logic:='0';
signal carpresence_NS_tb:std_logic:='0';
signal night_mode_tb:std_logic:='0';
signal keypad_hold_in_tb:std_logic_vector(3 downto 0):="0000";

--output
signal keypad_hold_out_tb:std_logic_vector(3 downto 0);
signal FSM_currentstate_tb:std_logic_vector(1 downto 0);
signal redlight_flash_EW_tb:std_logic;
signal redlight_flash_NS_tb:std_logic;
signal traffic_light_tb:std_logic_vector(2 downto 0);
signal select_segment_tb:std_logic;
signal segment_display_tb:std_logic_vector(6 downto 0);
constant time_period:time:= 4 ns;

begin
component_tlc: component traffic_light_controller port map
    (view_NS_light => view_NS_light_tb,
     clk => clk_tb,
     reset => reset_tb,
     carcrossing_EW => carcrossing_EW_tb,
     carcrossing_NS => carcrossing_NS_tb,
     carpresence_EW => carpresence_EW_tb,
     carpresence_NS => carpresence_NS_tb,
     night_mode => night_mode_tb,
     keypad_hold_in=> keypad_hold_in_tb,
     keypad_hold_out=> keypad_hold_out_tb,
     FSM_currentstate => FSM_currentstate_tb,
     redlight_flash_EW => redlight_flash_EW_tb,

```

```

        redlight_flash_NS => redlight_flash_NS_tb,
        traffic_light => traffic_light_tb,
        select_segment => select_segment_tb,
        segment_display => segment_display_tb);

--clock process
clk:process
begin
    clk_tb <='0';
    wait for time_period/2;
    clk_tb <='1';
    wait for time_period/2;
end process clk;

stimu_proc: process
begin
    night_mode_tb <= '0';
    view_NS_light_tb <= '0';
    reset_tb <= '1';
    carcrossing_EW_tb <= '0';
    carcrossing_NS_tb <= '0';
    carpresence_EW_tb <= '0';
    carpresence_NS_tb <= '0';
    keypad_hold_in_tb <= "1111";
    wait for 16 ns;
    reset_tb <= '0';
    wait for 24 ns;
    carcrossing_NS_tb <= '1';
    --Test illegal crossing camera on NS direction.
    wait for 16 ns;
    carcrossing_NS_tb <= '0';
    wait for 160 ns;
    carcrossing_EW_tb <= '1';
    --Test illegal crossing camera on EW direction.
    wait for 16 ns;
    carcrossing_EW_tb <= '0';
    wait for 48 ns;
    keypad_hold_in_tb <= "0111";
    --Press '1' on kypd to display pedestrian crosswalk EW/NS.
    wait for 72 ns;

    view_NS_light_tb <= '1';
    -- view NS direction light.
    wait for 24 ns;
    carcrossing_NS_tb <= '1';

```



```

wait for 16 ns;
carcrossing_NS_tb <= '0';
wait for 128 ns;
keypad_hold_in_tb <= "1111";
wait for 32 ns;
carcrossing_EW_tb <= '1';
wait for 16 ns;
carcrossing_EW_tb <= '0';
wait for 120 ns;
view_NS_light_tb <= '0';
night_mode_tb <= '1';
--Test the night mode function.
wait for 8 ns;
carcrossing_NS_tb <= '1';
wait for 8 ns;
carcrossing_NS_tb <= '0';
wait for 8 ns;
carpresence_NS_tb <= '1';
--Send the carpresence_NS signal for next state transition.
wait for 8 ns;
carpresence_NS_tb <= '0';
wait for 16 ns;
view_NS_light_tb <= '1';
carcrossing_EW_tb <= '1';
wait for 8 ns;
carcrossing_EW_tb <= '0';
wait for 8 ns;
carpresence_EW_tb <= '1';
--Send the carpresence_EW signal for next state transition.
wait for 8 ns;
carpresence_NS_tb <= '0';
wait for 16 ns;
wait;
end process stimu_proc;
end Behavioral;

```

Constraint file of the traffic intersection controller:

In constraint file, each entity port is mapped to the Zybo Z7 board as shown below:

```

##Clock signal
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 { }];
##Switches
set_property -dict {PACKAGE_PIN G15 IOSTANDARD LVCMOS33} [ get_ports { carpresence_EW }];

```

```

#Sch=sw[0]
set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { carpresence_NS }];
#Sch=sw[1]
set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { night_mode }];
#IO_L9P_T1_DQS_34 Sch=sw[3]

##Buttons
set_property -dict { PACKAGE_PIN K18 IOSTANDARD LVCMOS33 } [get_ports { view_NS_light }];
#IO_L12N_T1_MRCC_35 Sch=btn[0]
set_property -dict { PACKAGE_PIN P16 IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L24N_T3_34 Sch=btn[1]
set_property -dict { PACKAGE_PIN K19 IOSTANDARD LVCMOS33 } [get_ports { carcrossing_EW }];
#IO_L10P_T1_AD11P_35 Sch=btn[2]
set_property -dict { PACKAGE_PIN Y16 IOSTANDARD LVCMOS33 } [get_ports { carcrossing_NS }];
#IO_L7P_T1_34 Sch=btn[3]

##LEDs
set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { FSM_currentstate[0] }];
set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { FSM_currentstate[1] }];
set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports { redlight_flash_EW }];
set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { redlight_flash_NS }];

##RGB LED 6
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { traffic_light[0] }];
#IO_L18P_T2_34 Sch=led6_r
set_property -dict { PACKAGE_PIN F17 IOSTANDARD LVCMOS33 } [get_ports { traffic_light[1] }];
#IO_L6N_T0_VREF_35 Sch=led6_g
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports { traffic_light[2] }];
#IO_L8P_T1_AD10P_35 Sch=led6_b

##Pmod Header JC
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { segment_display[0] }];
#IO_L10P_T1_34 Sch=jc_p[1]
set_property -dict { PACKAGE_PIN W15 IOSTANDARD LVCMOS33 } [get_ports { segment_display[1] }];
#IO_L10N_T1_34 Sch=jc_n[1]
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { segment_display[2] }];
#IO_L1P_T0_34 Sch=jc_p[2]
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { segment_display[3] }];
#IO_L1N_T0_34 Sch=jc_n[2]

##Pmod Header JD
set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { segment_display[4] }];
#IO_L5P_T0_34 Sch=jd_p[1]
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { segment_display[5] }];

```

```

#IO_L5N_T0_34 Sch=jd_n[1]
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { segment_display[6] }];
#IO_L6P_T0_34 Sch=jd_p[2]
set_property -dict { PACKAGE_PIN R14 IOSTANDARD LVCMOS33 } [get_ports { select_segment }];
#IO_L6N_T0_VREF_34 Sch=jd_n[2]

##Pmod Header JE
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_out[0] }];
#IO_L4P_T0_34 Sch=je[1]
set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_out[1] }];
#IO_L18N_T2_34 Sch=je[2]
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_out[2] }];
#IO_25_35 Sch=je[3]
set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_out[3] }];
#IO_L19P_T3_35 Sch=je[4]
set_property -dict { PACKAGE_PIN V13 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_in[0] }];
#IO_L3N_T0_DQS_34 Sch=je[7]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_in[1] }];
#IO_L9N_T1_DQS_34 Sch=je[8]
set_property -dict { PACKAGE_PIN T17 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_in[2] }];
#IO_L20P_T3_34 Sch=je[9]
set_property -dict { PACKAGE_PIN Y17 IOSTANDARD LVCMOS33 } [get_ports { keypad_hold_in[3] }];
#IO_L7N_T1_34 Sch=je[10]

```