Students: Kevin (Kaiwen) Xue - 20814292, Anni Wang - 20849760

BME 393L: Lab 4

4.1 Binary clock-divider

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity binary_clock_divider is
   port(
         CLOCK_50_B5B:  in  std_logic ;     -- 50MHz clock on the board
         LEDR:       out std_logic_vector(9 downto 0)
      );
end entity binary_clock_divider;


architecture binary_clock_divider_architecture of binary_clock_divider is

signal counter: unsigned(29 downto 0);

begin

counter <= counter + 1 when rising_edge(CLOCK_50_B5B);
LEDR <= std_logic_vector(counter)(29 downto 20);

end architecture binary_clock_divider_architecture;
```

Figure 1: VHDL code with the entity and architecture declaration for the binary clock divider. Here, the 10 most significant bits are sent to the LEDR output group.

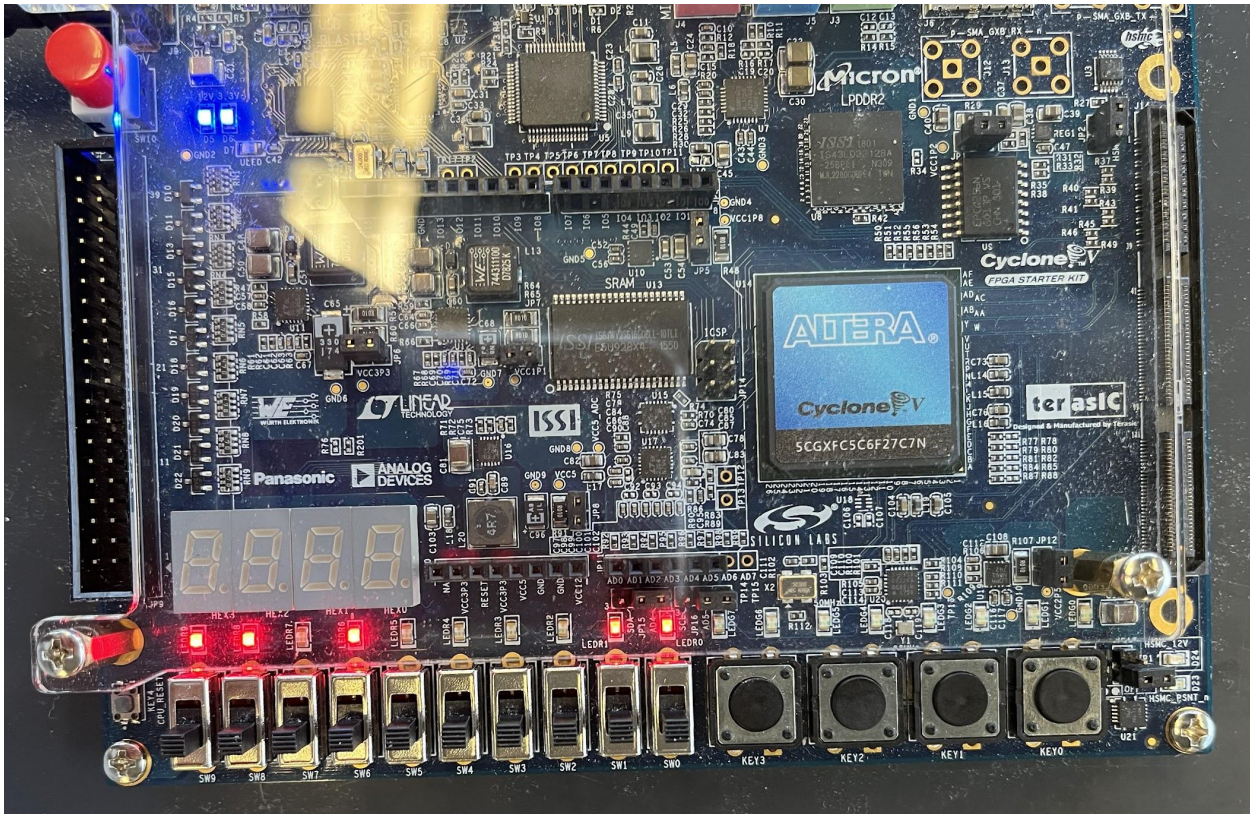

Figure 2: FPGA board with the binary clock-divider code

Video of the FPGA board with the binary clock-divider code working:
https://drive.google.com/file/d/1KJ2E2T6h4wLsR2lMo964I1HeSBbzzbQT/view?usp=sharing

4.2 Modulus clock-dividers

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity modulus_clock_divider is
   port(
         CLOCK_50_B5B:  in  std_logic ;     -- 50MHz clock on the board
         LEDG:       out std_logic_vector(9 downto 0)
      );
end entity modulus_clock_divider;


architecture modulus_clock_divider_architecture of modulus_clock_divider is

signal counter: unsigned(24 downto 0);
signal output: std_logic := '0';

begin

counting: process (CLOCK_50_B5B)
begin
 if rising_edge(CLOCK_50_B5B) then
  if (counter = to_unsigned(24999999, 25)) then
   counter <= to_unsigned(0, 25);
   output <= not output;
  else
   counter <= counter + 1;
  end if;
 end if;
end process;

LEDG(5) <= std_logic(output);

end architecture modulus_clock_divider_architecture;
```

Figure 3: VHDL code with the entity and architecture declaration for the modulus clock divider. Here, an integer counter is incremented based on the ratio of input rising edges to output rising edges (25000000:1) to achieve a 1 Hz output clock frequency. The output clock value is sent to LEDG(5).

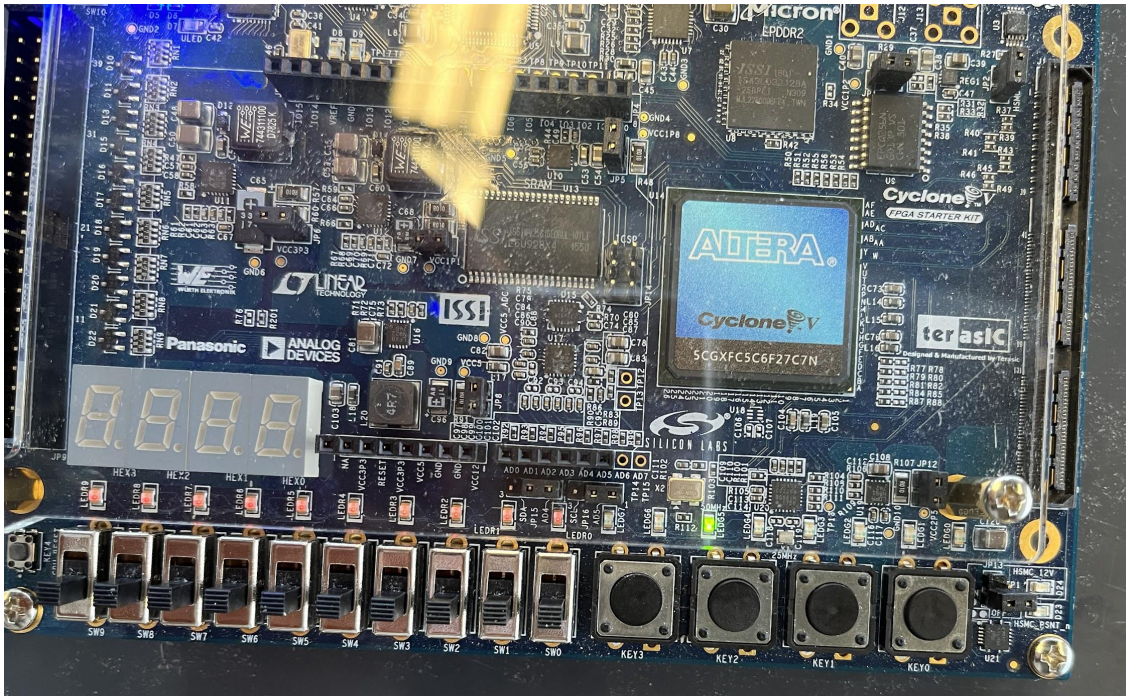

Figure 4: FPGA board with the modulus clock-dividers

Video of the FPGA board with the modulus clock-dividers working:
https://drive.google.com/file/d/1O3NuCMyPsO0UJusQR7189TFG5umQ1Er9/view?usp=sharing

4.3 Random number generator

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity random_number_generator is port(
        CLOCK_50_B5B:  in  std_logic ;    -- 50MHz clock on the board
        KEY:     in  std_logic_vector(3 downto 0);
        HEX3:    out std_logic_vector(6 downto 0);
        HEX2:    out std_logic_vector(6 downto 0);
        HEX1:    out std_logic_vector(6 downto 0);
        HEX0:    out std_logic_vector(6 downto 0)
        );
end entity random_number_generator;

architecture random_number_generator_architecture of random_number_generator is

component seven_segment is port (
      data_in: in std_logic_vector(3 downto 0);
      blank: in std_logic;
      data_out: out std_logic_vector(6 downto 0)
);
end component;

signal rndm: unsigned(7 downto 0);
signal prev_rndm: unsigned(7 downto 0);
signal hex0_input, hex1_input, hex2_input, hex3_input: std_logic_vector(3 downto 0);
signal blank_prev_rndm: std_logic := '1';
signal blank_rndm: std_logic := '1';

begin

hex0_inst: entity work.seven_segment(seven_segment_architecture) port map (hex0_input, blank_rndm, hex0);
hex1_inst: entity work.seven_segment(seven_segment_architecture) port map (hex1_input, blank_rndm, hex1);
hex2_inst: entity work.seven_segment(seven_segment_architecture) port map (hex2_input, blank_prev_rndm, hex2);
hex3_inst: entity work.seven_segment(seven_segment_architecture) port map (hex3_input, blank_prev_rndm, hex3);

rndm <= rndm + 1 when rising_edge(CLOCK_50_B5B);
```

Figure 5: VHDL code with the entity and architecture declaration for the random number generator. Here, the component declaration, signals and the port maps are defined.

```
rng_process: process (KEY(0))
begin

if rising_edge(KEY(0)) then
   if blank_rndm = '0' then
      blank_prev_rndm <= '0';
      hex3_input <= std_logic_vector(prev_rndm)(7 downto 4);
      hex2_input <= std_logic_vector(prev_rndm)(3 downto 0);
   else
      blank_rndm <= '0';
   end if;
   hex1_input <= std_logic_vector(rndm)(7 downto 4);
   hex0_input <= std_logic_vector(rndm)(3 downto 0);
   prev_rndm <= rndm;

end if;


end process;

end architecture random_number_generator_architecture;
```

Figure 6: VHDL code where the main random number process is defined.

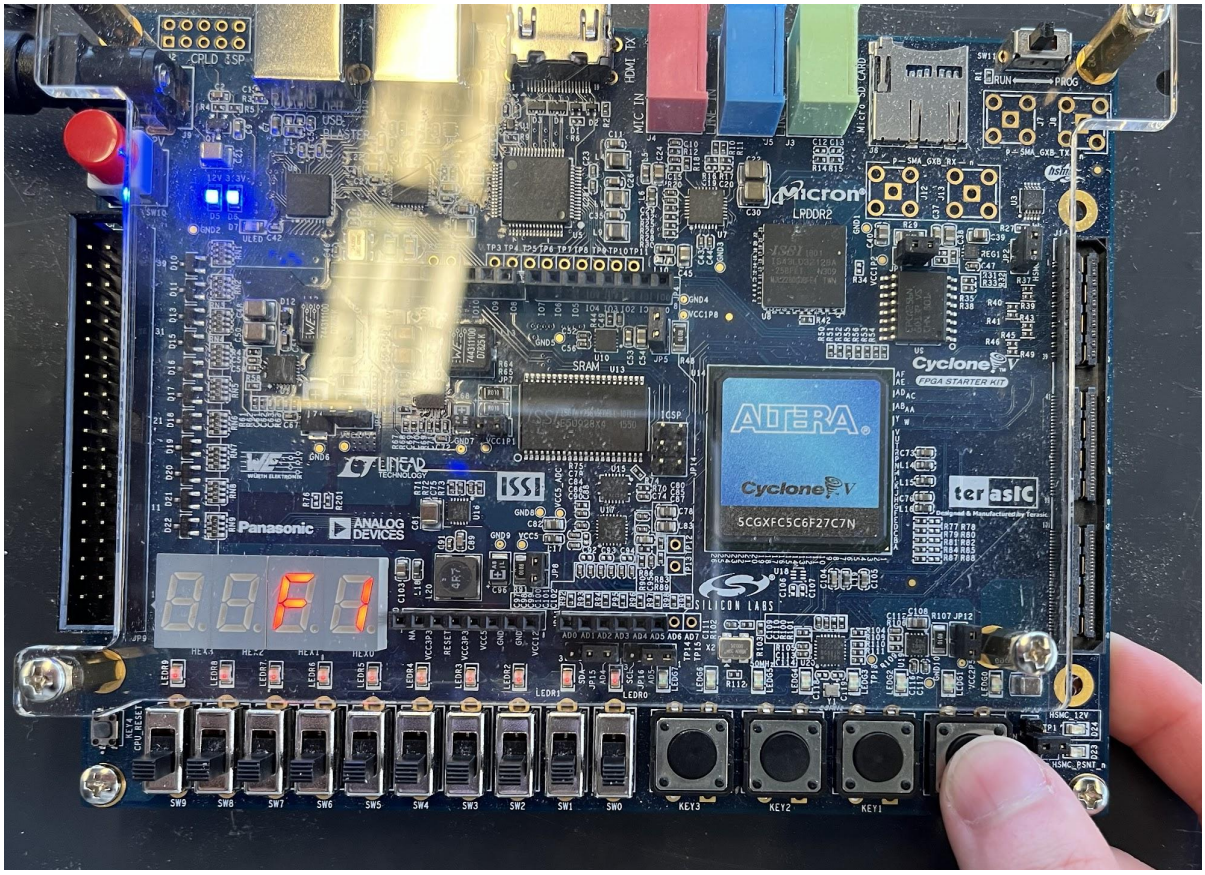

Figure 7: FPGA board with the random number generator code. Demonstrating the first click with HEX2 and HEX3 blank.

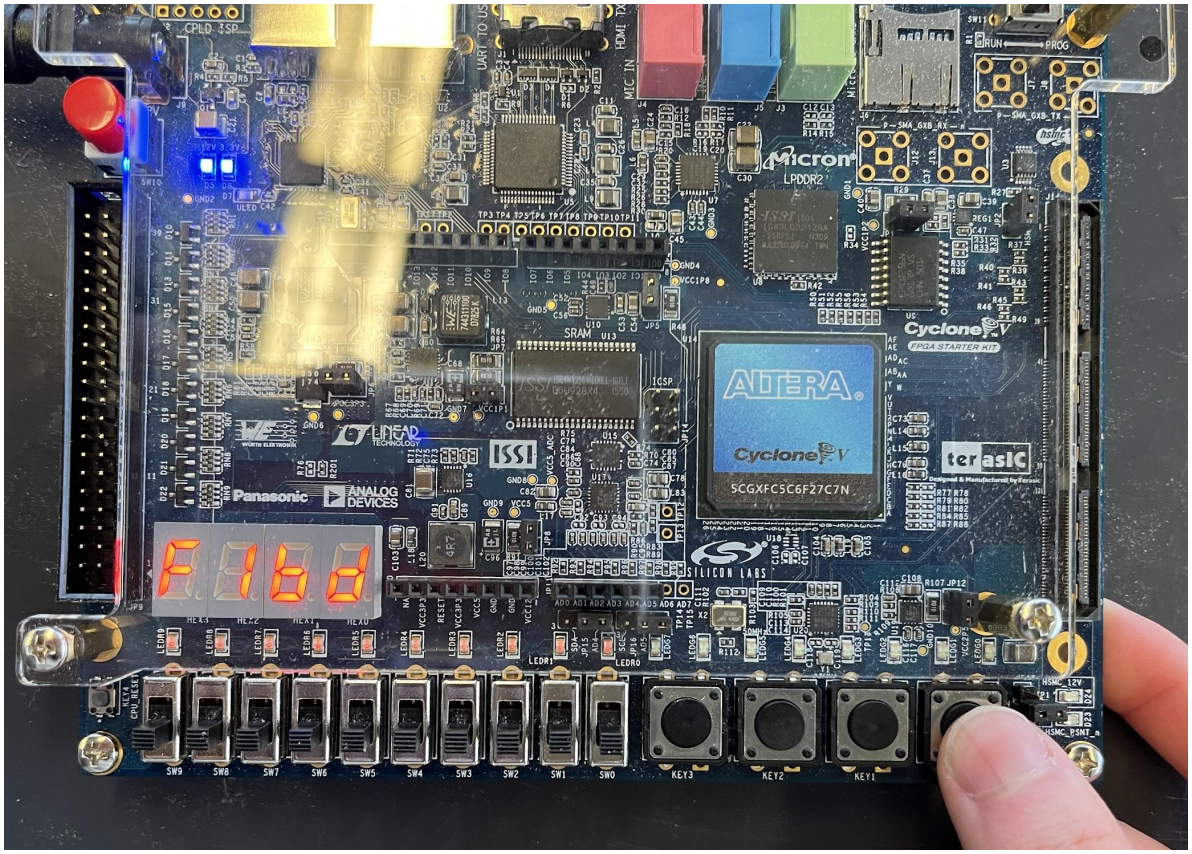

Figure 8: FPGA board with the random number generator code. Demonstrating the second click.

Video of the FPGA board with the random number generator code working:
https://drive.google.com/file/d/1D8jWy08NvJYuhq7Anl6ALFRSFK0xifEx/view?usp=sharing

4.4 Traffic light - State machine implementation

```
entity traffic_light_controller is
   port(
         CLOCK_50_B5B:  in  std_logic;     -- 50MHz clock on the board
         HEX0:     out std_logic_vector(6 downto 0);
         HEX3:     out std_logic_vector(6 downto 0);
         LEDR:     out   std_logic_vector(9 downto 0);     -- Red LEDs
         LEDG:     out   std_logic_vector(7 downto 0)      -- Green LEDs
      );
end entity traffic_light_controller;


architecture traffic_light_controller_architecture of traffic_light_controller is

component clock_divider is
   port (
            clock_in: in std_logic;
            num_of_rising_edges: in integer;
            clock_out: out std_logic
);
end component;

type light_state_enum is (f_green, s_green, f_red, s_red);


signal light_state:  light_state_enum := f_green;
signal light_next_state:  light_state_enum := f_green;
signal light_state_encoding: std_logic_vector(1 downto 0);

type direction_state_enum is (NS, EW);
signal direction_state:  direction_state_enum := NS;
signal direction_next_state:  direction_state_enum := NS;

signal fast_clock, slow_clock: std_logic;
signal time_counter: unsigned(2 downto 0) := "000";
signal state_encoding: std_logic_vector(3 downto 0);

begin
fast_clock_inst: entity work.clock_divider(clock_divider_architecture) port map (CLOCK_50_B5B, 2500000, fast_clock); -- 10 Hz clock
slow_clock_inst: entity work.clock_divider(clock_divider_architecture) port map (fast_clock, 5, slow_clock); -- 1 Hz clock

hex0_inst: entity work.seven_segment(behavioral) port map ('0' & std_logic_vector(time_counter), '0', hex0);
hex3_inst: entity work.seven_segment(behavioral) port map (state_encoding, '0', hex3);
```

Figure 9: VHDL code with the entity and architecture declaration for the traffic light controller. Here, the component declaration, state enum, signals, clocks, and port maps are defined.

```
traffic_light_controller_next_state_process: process (slow_clock)
begin

if rising_edge(slow_clock) then
   case light_state is
      when f_green =>
         if time_counter = "001" then
            light_state <= s_green;
            time_counter <= "000";
         else
            time_counter <= time_counter + 1;
         end if;
      when s_green =>
         if time_counter = "100" then
            light_state <= f_red;
            time_counter <= "000";
         else
            time_counter <= time_counter + 1;
         end if;
      when f_red =>
         if time_counter = "010" then
            light_state <= s_red;
            time_counter <= "000";
         else
            time_counter <= time_counter + 1;
         end if;
      when s_red =>
         if time_counter = "000" then
            light_state <= f_green;
            time_counter <= "000";
            if direction_state = NS then
               direction_state <= EW;
            else
               direction_state <= NS;
            end if;
         else
            time_counter <= time_counter + 1;
         end if;
      when others =>
            null;
   end case;

end if;

end process;
```

Figure 10: VHDL code where the main traffic light controller next state process is defined.

```
traffic_light_controller_state_encoding_process: process (light_state, direction_state)
begin
   case light_state is
      when f_green =>
         light_state_encoding <= "00";
      when s_green =>
         light_state_encoding <= "01";
      when f_red =>
         light_state_encoding <= "10";
      when others =>
         light_state_encoding <= "11";
   end case;
   case direction_state is
      when NS =>
         state_encoding <= "00" & light_state_encoding;
      when EW =>
         state_encoding <= "01" & light_state_encoding;
      when others =>
         state_encoding <= "1000";
   end case;
end process;

with state_encoding select
      LEDG(7) <= fast_clock when "0000",
                 '1' when "0001",
                 '0' when others;

with state_encoding select
      LEDR(0) <= fast_clock when "0010",
                 '0' when "0000",
                 '0' when "0001",
                 '1' when others;

with state_encoding select
      LEDG(4) <= fast_clock when "0100",
                 '1' when "0101",
                 '0' when others;

with state_encoding select
      LEDR(4) <= fast_clock when "0110",
                 '0' when "0100",
                 '0' when "0101",
                 '1' when others;


end architecture traffic_light_controller_architecture;
```

Figure 11: VHDL code with the state encoding process and the output logic implemented using concurrent statements.

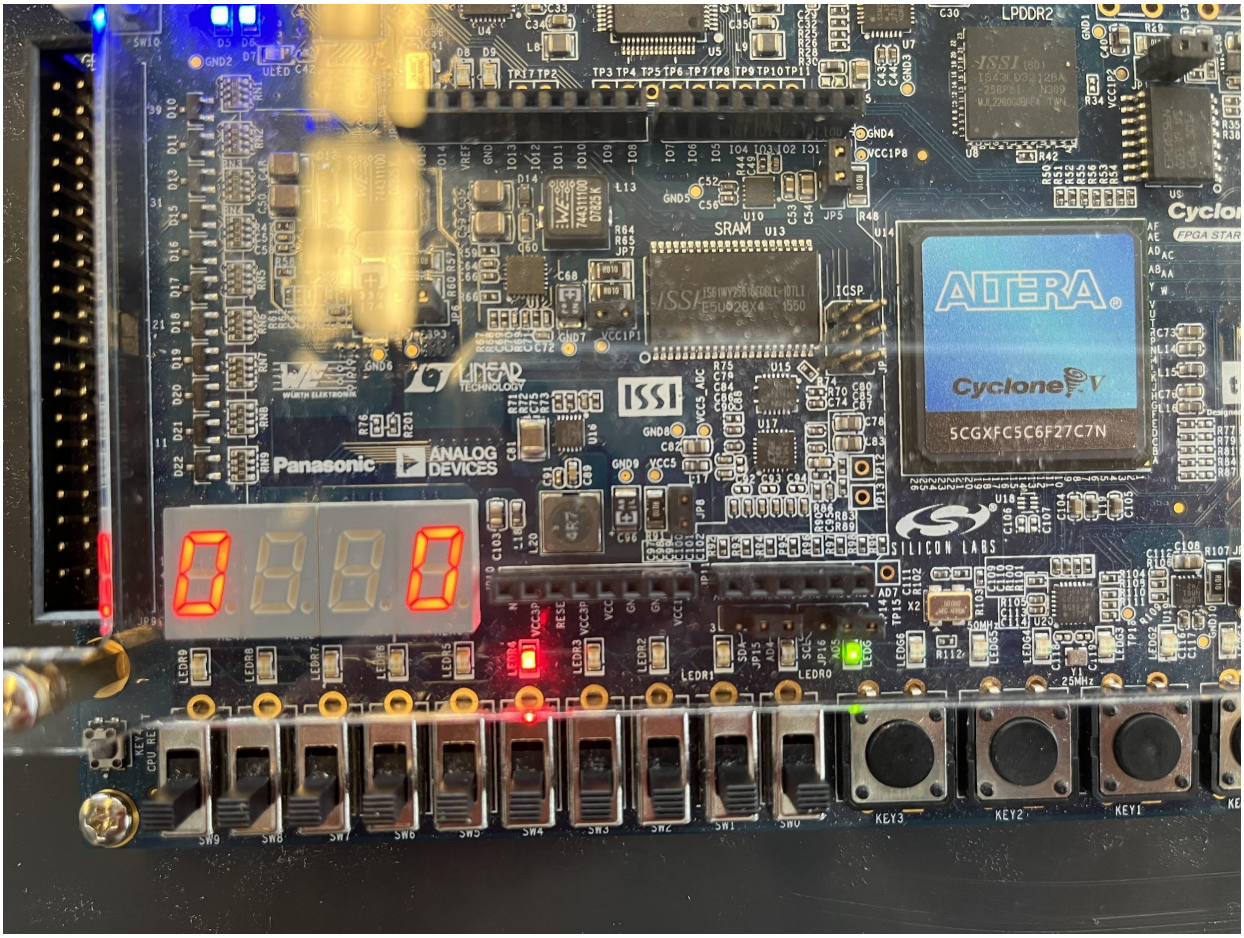

Figure 12: FPGA board with the traffic light code. Demonstrating state 0 at time 0s.

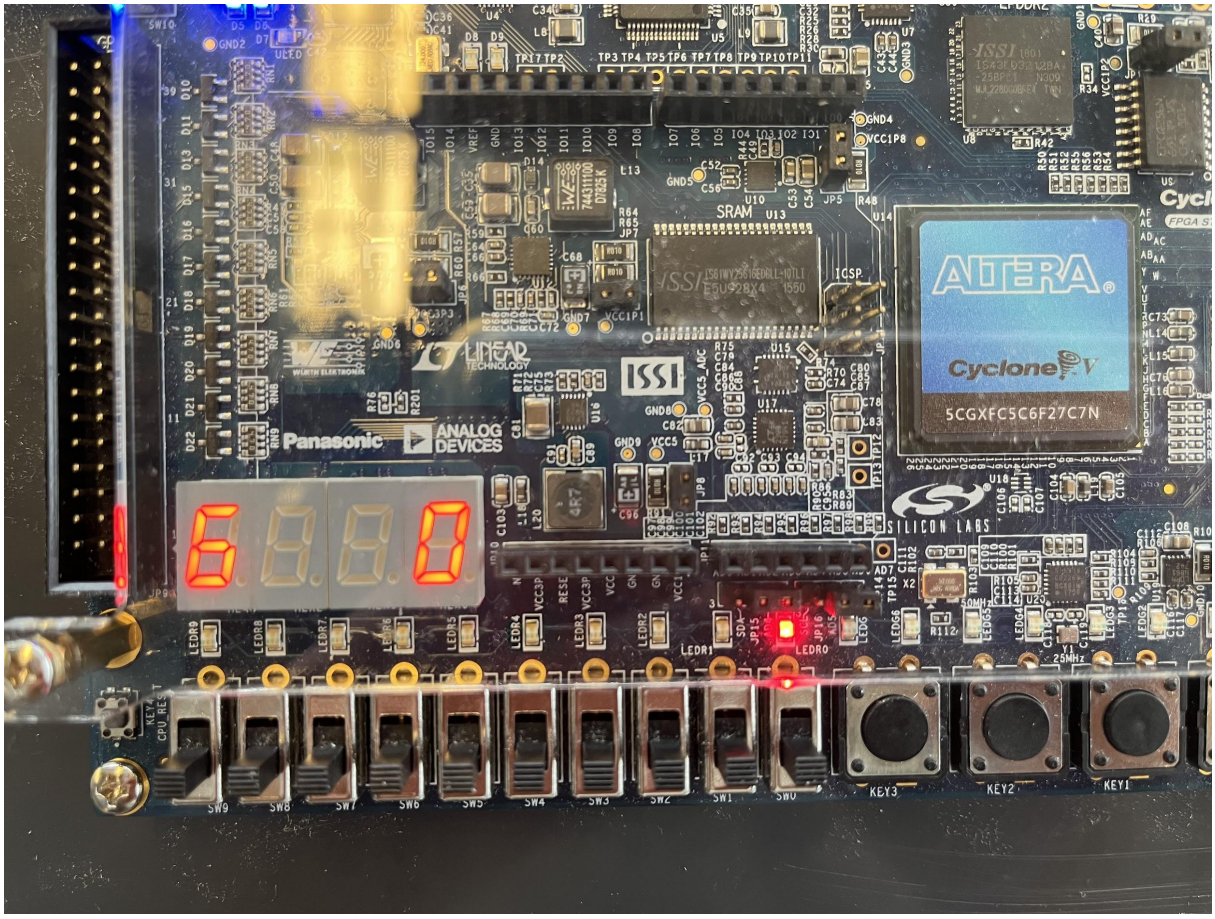

Figure 13: FPGA board with the traffic light code. Demonstrating state 6 at time 0s.

Video of the FPGA board with the traffic light code working:
https://drive.google.com/file/d/1zidasvdYhwy7ukYMzLXPYmKa9tfka3aS/view?usp=sharing

5. Post-lab

```
entity reaction_time_calculator is
   port(
         CLOCK_50_B5B:  in  std_logic;     -- 50MHz clock on the board
         KEY:      in    std_logic_vector(3 downto 0);     -- Push buttons
         HEX0:     out std_logic_vector(6 downto 0);
         HEX1:     out std_logic_vector(6 downto 0);
         HEX2:     out std_logic_vector(6 downto 0);
         HEX3:     out std_logic_vector(6 downto 0)
      );
end entity reaction_time_calculator;


architecture reaction_time_calculator_architecture of reaction_time_calculator is

component clock_divider is
   port (
            clock_in: in std_logic;
            num_of_rising_edges: in integer;
            clock_out: out std_logic
);
end component;

type reaction_time_calculator_state_enum is (idle, key0_initiated, key1_initiated, finished_calculation);
signal current_reaction_time_calculator_state:  reaction_time_calculator_state_enum := idle;
signal clock: std_logic;
signal time_counter_previous: unsigned(15 downto 0) := "0000000000000000";
signal time_counter_current: unsigned(15 downto 0) := "0000000000000000";
signal blank: std_logic := '1';
begin
clock_inst: entity work.clock_divider(clock_divider_architecture) port map (CLOCK_50_B5B, 25000, clock); -- 1000 Hz clock

hex0_inst: entity work.seven_segment(behavioral) port map (std_logic_vector(time_counter_previous(3 downto 0)), blank, hex0);
hex1_inst: entity work.seven_segment(behavioral) port map (std_logic_vector(time_counter_previous(7 downto 4)), blank, hex1);
hex2_inst: entity work.seven_segment(behavioral) port map (std_logic_vector(time_counter_previous(11 downto 8)), blank, hex2);
hex3_inst: entity work.seven_segment(behavioral) port map (std_logic_vector(time_counter_previous(15 downto 12)), blank, hex3);
```

Figure 14: VHDL code with the entity and architecture declaration for the reaction time calculator. Here, the component declaration, state enum, signals, clocks, and port maps are defined.

```
reaction_time_calculator_next_state_process: process (KEY, clock)
begin
if rising_edge(clock) then
   case current_reaction_time_calculator_state is
      when key0_initiated =>
         if KEY(1) = '0' then
            current_reaction_time_calculator_state <= finished_calculation;
            time_counter_previous <= time_counter_current;
            time_counter_current <= "0000000000000000";
            blank <= '0';
         else
            time_counter_current <= time_counter_current + 1;
         end if;
      when key1_initiated =>
         if KEY(0) = '0' then
            current_reaction_time_calculator_state <= finished_calculation;
            time_counter_previous <= time_counter_current;
            time_counter_current <= "0000000000000000";
            blank <= '0';
         else
            time_counter_current <= time_counter_current + 1;
         end if;
      when idle =>
         if KEY(0) = '0' then
            current_reaction_time_calculator_state <= key0_initiated;
         elsif KEY(1) = '0' then
            current_reaction_time_calculator_state <= key1_initiated;
         else
            null;
         end if;
      when finished_calculation =>
         if KEY(0) = '1' and KEY(1) = '1' then
            current_reaction_time_calculator_state <= idle;
         else
            null;
         end if;
   end case;
end if;

end process;

end architecture reaction_time_calculator_architecture;
```

Figure 15: VHDL code with the next state process for the reaction time calculator.

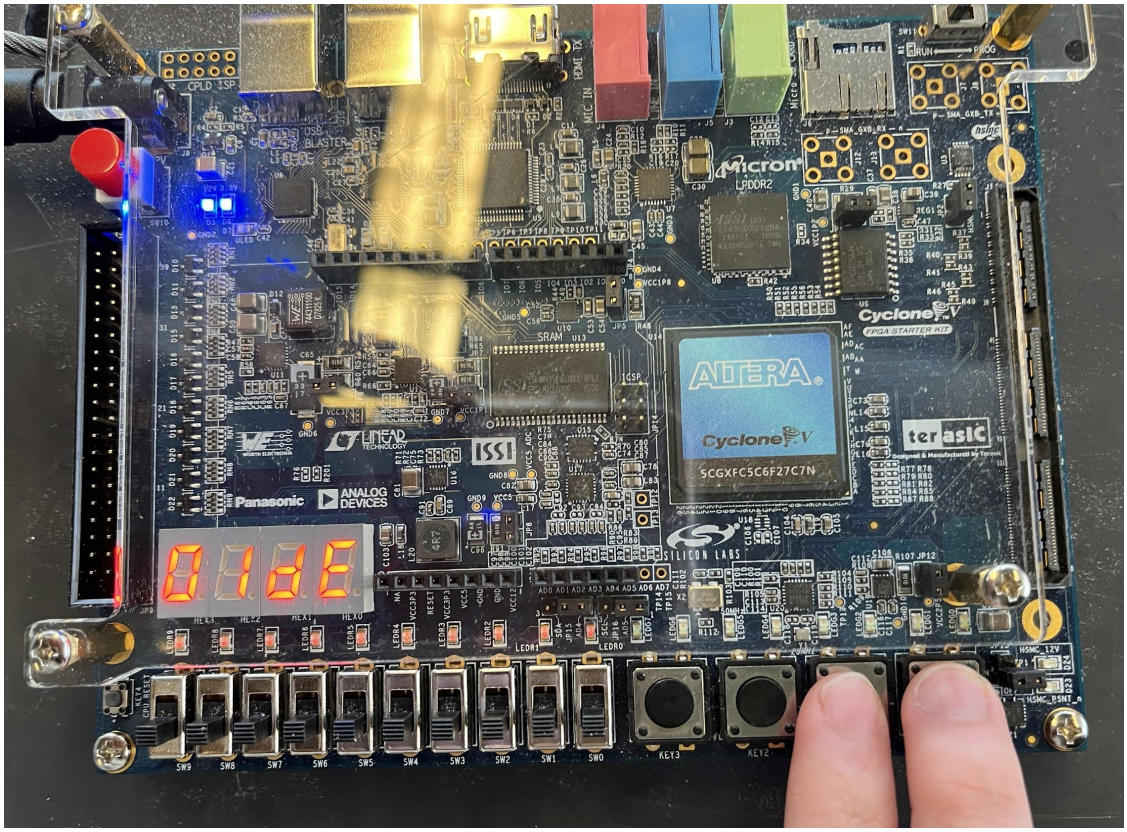

Figure 16: FPGA board with the post-lab code

Video of the FPGA board with the post-lab code working:
https://drive.google.com/file/d/1iFzFvboZ7AGAybsf2cLDmDUcGF0RwX5z/view?usp=sharing