LAB REPORT
Laboratory exercise 3: Clocks and Timers

3220104688 杨佳昕

26 Dec, 2024

1. Problem Description:

Part I

Build a 3-digit BCD counter displayed on HEX2-0 which counts automatically at one-second intervals after reset button is cancelled. Use KEY0 to reset the counter to 0.

Part II

Use HEX7-2 to build a time-of-day clock, where SW15-0 are utilized to control the initial value of hour and minute. After reset is cancelled, the clock begins to count automatically.

Part III

SW7-0 are used to set the time (in seconds) between reset cancel and LEDR lighting. After the game start, the set time passed and LEDR lights. The player pressed KEY3 to make LEDR out, and HEX2-0 shows the player's reaction time in milliseconds.

2. Design Formulation

Part I

Use a variable to count how many rising edges has triggered. When reaching 50 million, i.e., one second passed, add 1 to the variable representing the HEX value. Utilize integer division and remainder expression to separate hundreds, tens and ones digit and display them on the hex display.

Part II

The second-counter is the same as that in Part I. When the seconds digit reaches 60, reset it to 0 and add 1 to the minutes digit. So as the minutes and hours digit (when hours digit reaches 24, just reset itself).

Part III

Define three states: before_light, before_pressed, after_pressed.

* **Before_light:** Before LEDR lights, copy the value of SW7-0 to a variable *light_time*. Every time one second passes, minus 1 to *light_time* itself. When *light_time* runs out, light LEDR and move to *before_pressed* state.

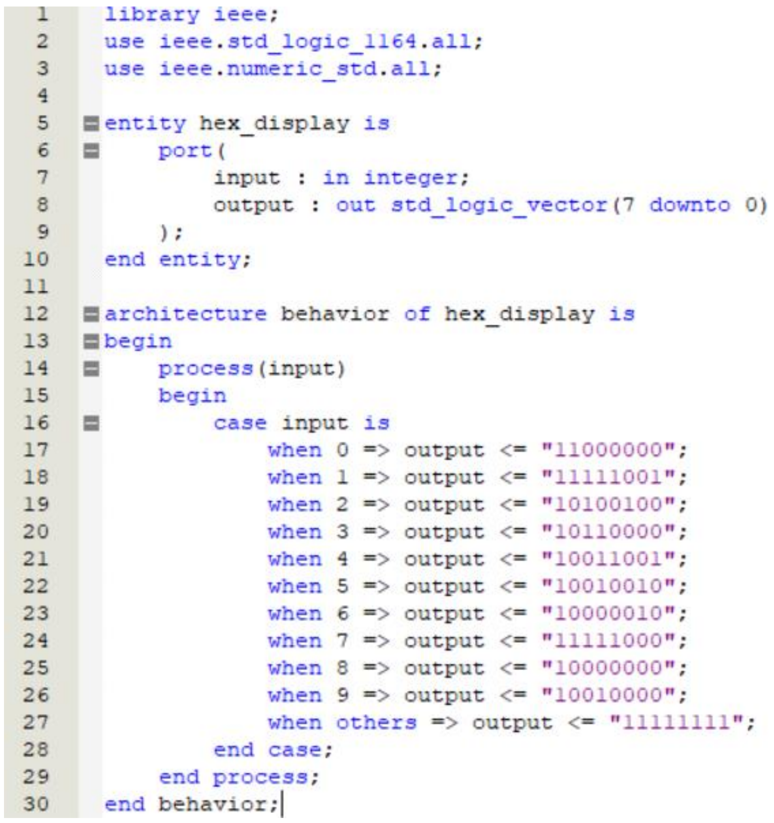
* **Before_pressed:** Before KEY3 is pressed, set counter to 0 and recount the time. Keep LEDR and counter on until KEY3 is pressed. Move to *after_pressed* state.

* **After_pressed:** Make LEDR off and copy the counter value to a new variable *reaction_time*. Separate hundreds, tens and

ones digit of *reaction_time* and display them on the hex display.

3. Design Entry

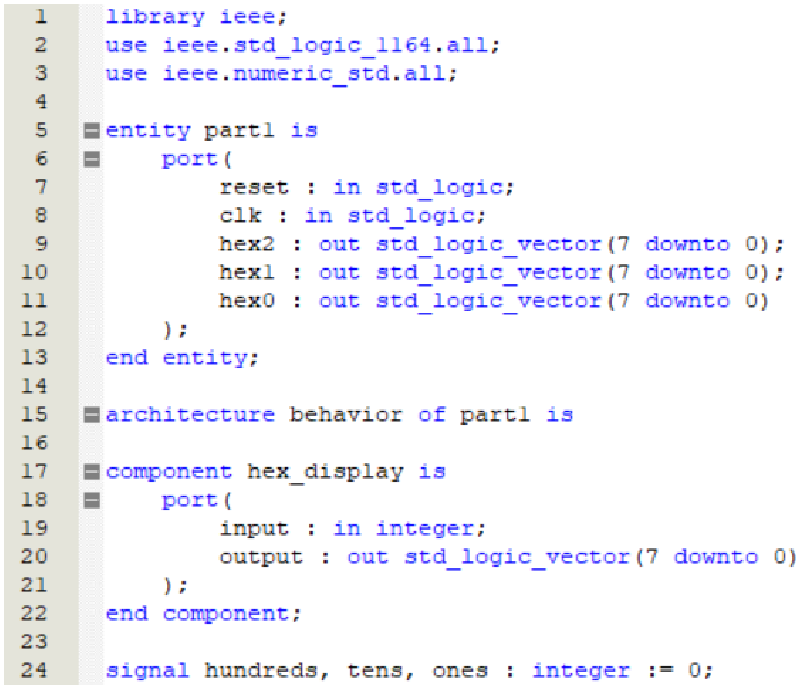
A VHDL file *hex_display.vhd* to change an integer to the form in hex display:



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

entity hex_display is
    port(
        input : in integer;
        output : out std_logic_vector(7 downto 0)
    );
end entity;

architecture behavior of hex_display is
begin
    process(input)
    begin
        case input is
            when 0 => output <= "11000000";
            when 1 => output <= "11111001";
            when 2 => output <= "10100100";
            when 3 => output <= "10110000";
            when 4 => output <= "10011001";
            when 5 => output <= "10010010";
            when 6 => output <= "10000010";
            when 7 => output <= "11111000";
            when 8 => output <= "10000000";
            when 9 => output <= "10010000";
            when others => output <= "11111111";
        end case;
    end process;
end behavior;
```

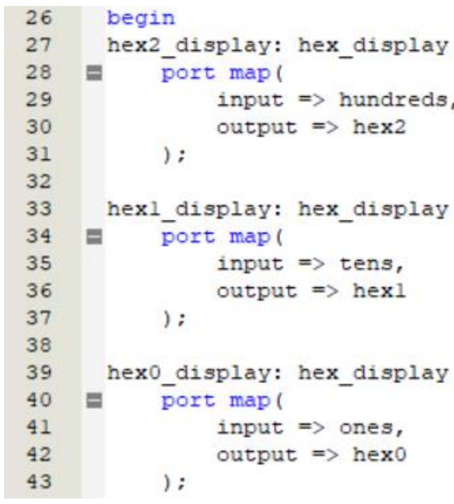
Part I

Entity, function import and signal form variable declaration:



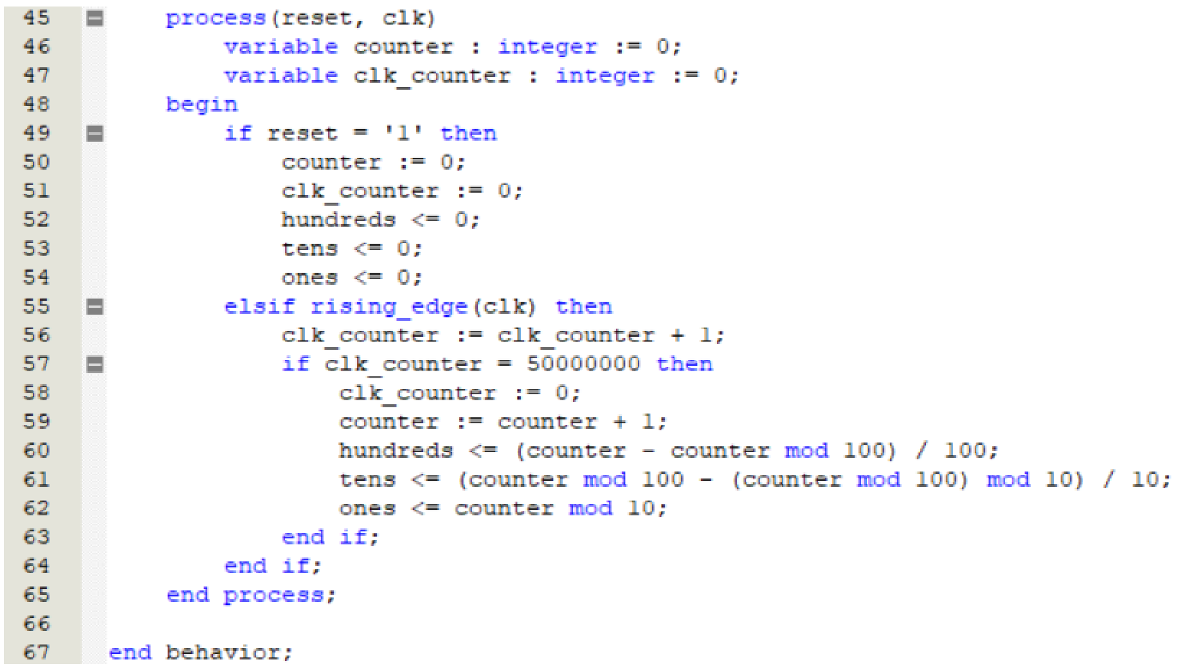
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity part1 is
    port(
        reset : in std_logic;
        clk : in std_logic;
        hex2 : out std_logic_vector(7 downto 0);
        hex1 : out std_logic_vector(7 downto 0);
        hex0 : out std_logic_vector(7 downto 0)
    );
end entity;

architecture behavior of part1 is

component hex_display is
    port(
        input : in integer;
        output : out std_logic_vector(7 downto 0)
    );
end component;

signal hundreds, tens, ones : integer := 0;
```

Hex display:



```vhdl
begin
hex2_display: hex_display
    port map(
        input => hundreds,
        output => hex2
    );

hex1_display: hex_display
    port map(
        input => tens,
        output => hex1
    );

hex0_display: hex_display
    port map(
        input => ones,
        output => hex0
    );
```

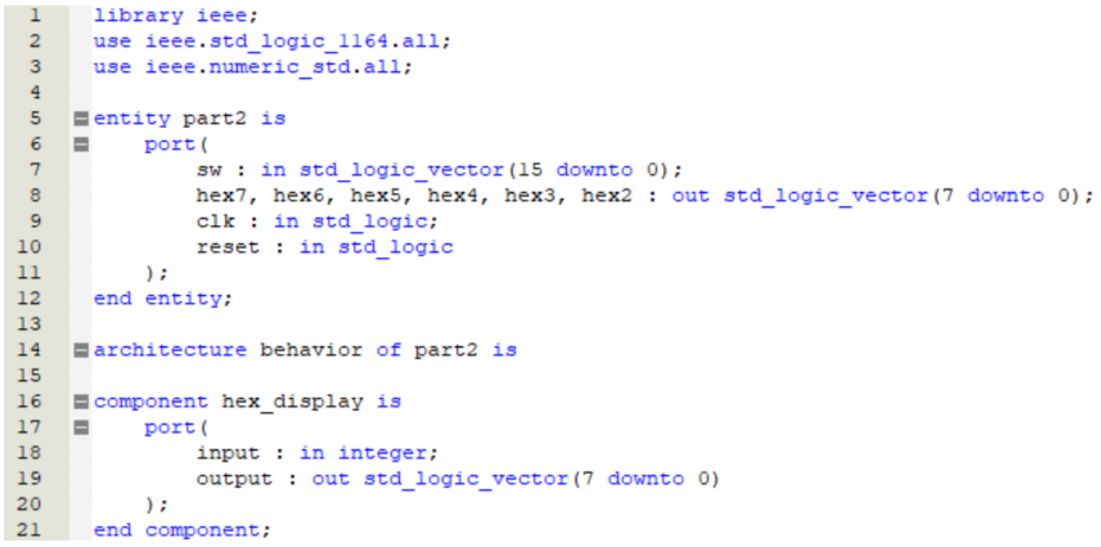
Process:



```vhdl
    process(reset, clk)
        variable counter : integer := 0;
        variable clk_counter : integer := 0;
    begin
        if reset = '1' then
            counter := 0;
            clk_counter := 0;
            hundreds <= 0;
            tens <= 0;
            ones <= 0;
        elsif rising_edge(clk) then
            clk_counter := clk_counter + 1;
            if clk_counter = 50000000 then
                clk_counter := 0;
                counter := counter + 1;
                hundreds <= (counter - counter mod 100) / 100;
                tens <= (counter mod 100 - (counter mod 100) mod 10) / 10;
                ones <= counter mod 10;
            end if;
        end if;
    end process;

end behavior;
```

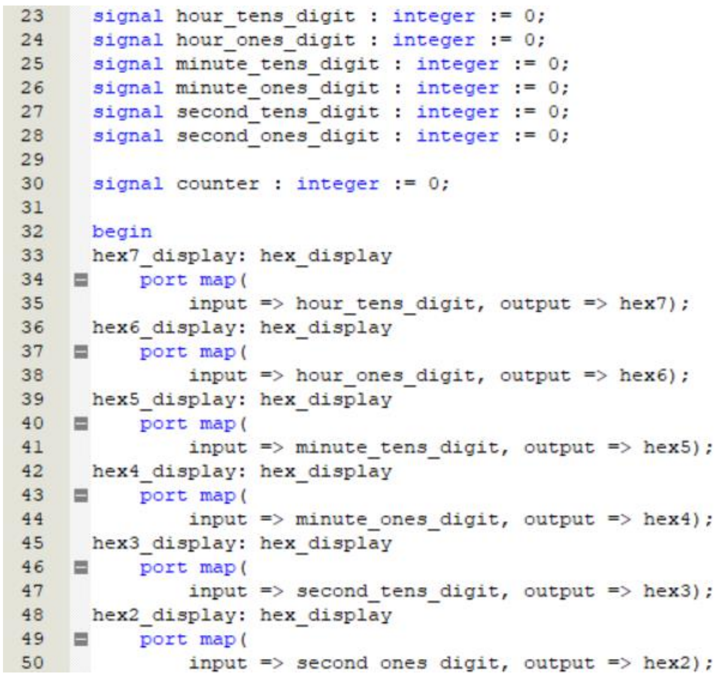
Part II

Entity and function import:



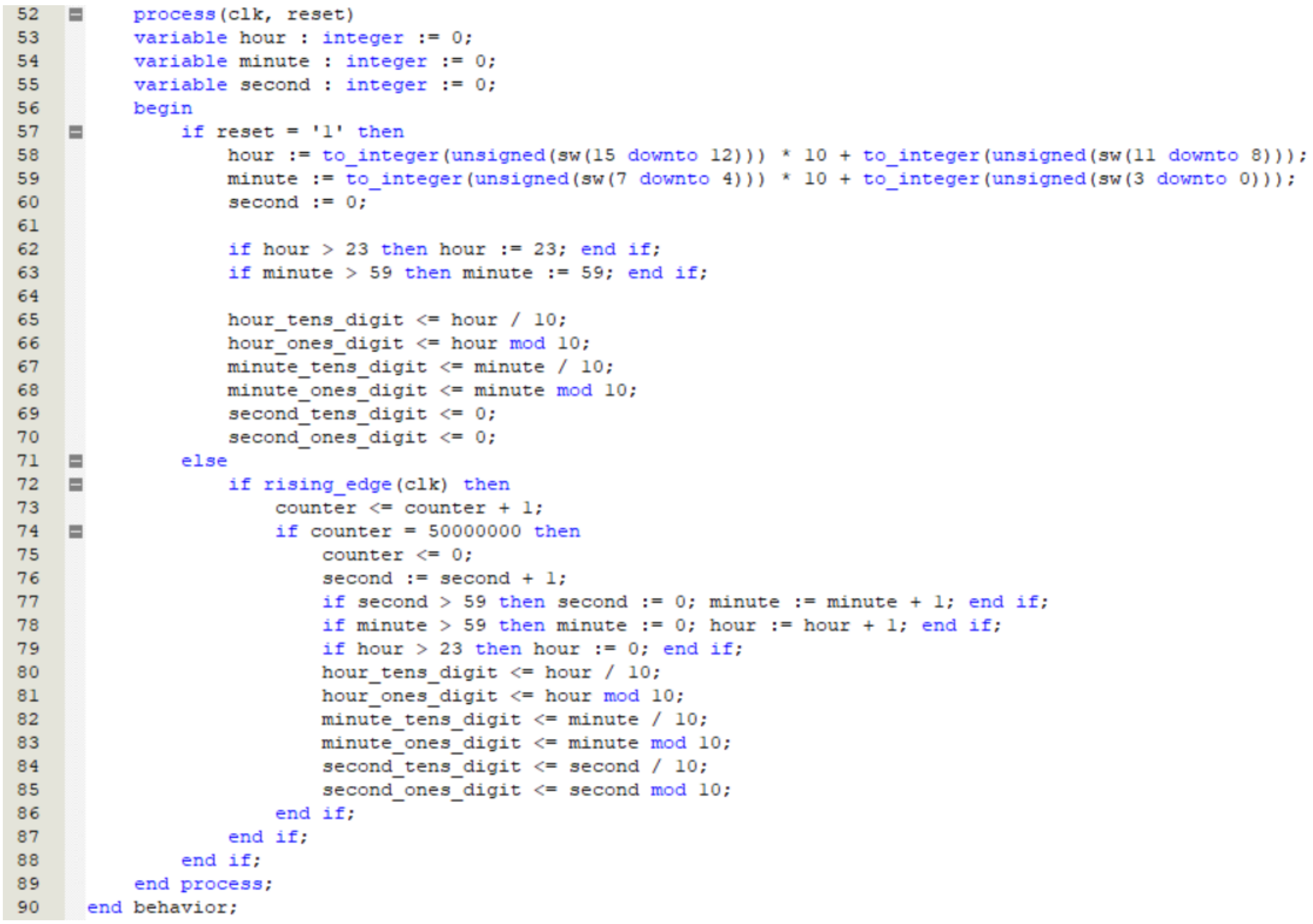
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity part2 is
    port(
        sw : in std_logic_vector(15 downto 0);
        hex7, hex6, hex5, hex4, hex3, hex2 : out std_logic_vector(7 downto 0);
        clk : in std_logic;
        reset : in std_logic
    );
end entity;

architecture behavior of part2 is

component hex_display is
    port(
        input : in integer;
        output : out std_logic_vector(7 downto 0)
    );
end component;
```

Hex display:



```vhdl
signal hour_tens_digit : integer := 0;
signal hour_ones_digit : integer := 0;
signal minute_tens_digit : integer := 0;
signal minute_ones_digit : integer := 0;
signal second_tens_digit : integer := 0;
signal second_ones_digit : integer := 0;

signal counter : integer := 0;

begin
hex7_display: hex_display
    port map(
        input => hour_tens_digit, output => hex7);
hex6_display: hex_display
    port map(
        input => hour_ones_digit, output => hex6);
hex5_display: hex_display
    port map(
        input => minute_tens_digit, output => hex5);
hex4_display: hex_display
    port map(
        input => minute_ones_digit, output => hex4);
hex3_display: hex_display
    port map(
        input => second_tens_digit, output => hex3);
hex2_display: hex_display
    port map(
        input => second_ones_digit, output => hex2);
```

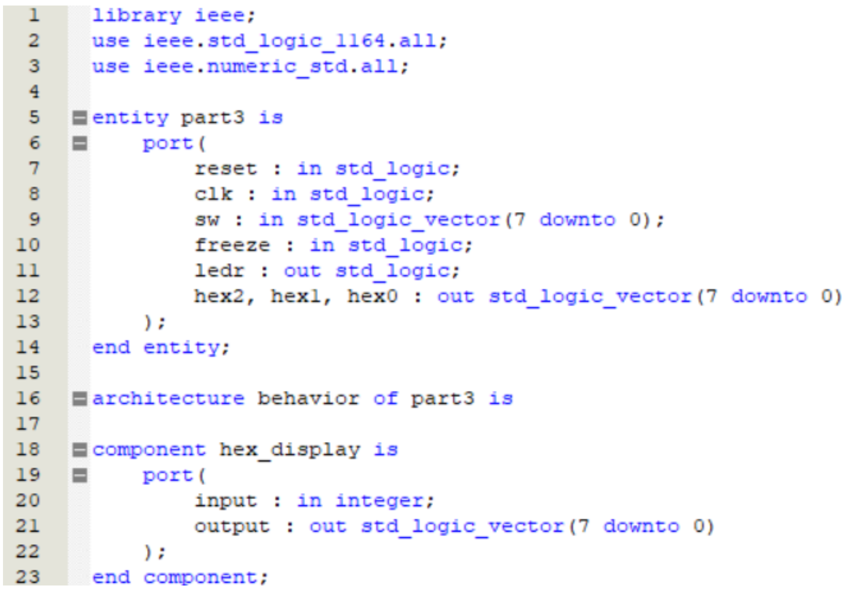
Process:



```vhdl
    process(clk, reset)
    variable hour : integer := 0;
    variable minute : integer := 0;
    variable second : integer := 0;
    begin
        if reset = '1' then
            hour := to_integer(unsigned(sw(15 downto 12))) * 10 + to_integer(unsigned(sw(11 downto 8)));
            minute := to_integer(unsigned(sw(7 downto 4))) * 10 + to_integer(unsigned(sw(3 downto 0)));
            second := 0;

            if hour > 23 then hour := 23; end if;
            if minute > 59 then minute := 59; end if;

            hour_tens_digit <= hour / 10;
            hour_ones_digit <= hour mod 10;
            minute_tens_digit <= minute / 10;
            minute_ones_digit <= minute mod 10;
            second_tens_digit <= 0;
            second_ones_digit <= 0;
        else
            if rising_edge(clk) then
                counter <= counter + 1;
                if counter = 50000000 then
                    counter <= 0;
                    second := second + 1;
                    if second > 59 then second := 0; minute := minute + 1; end if;
                    if minute > 59 then minute := 0; hour := hour + 1; end if;
                    if hour > 23 then hour := 0; end if;
                    hour_tens_digit <= hour / 10;
                    hour_ones_digit <= hour mod 10;
                    minute_tens_digit <= minute / 10;
                    minute_ones_digit <= minute mod 10;
                    second_tens_digit <= second / 10;
                    second_ones_digit <= second mod 10;
                end if;
            end if;
        end if;
    end process;
end behavior;
```

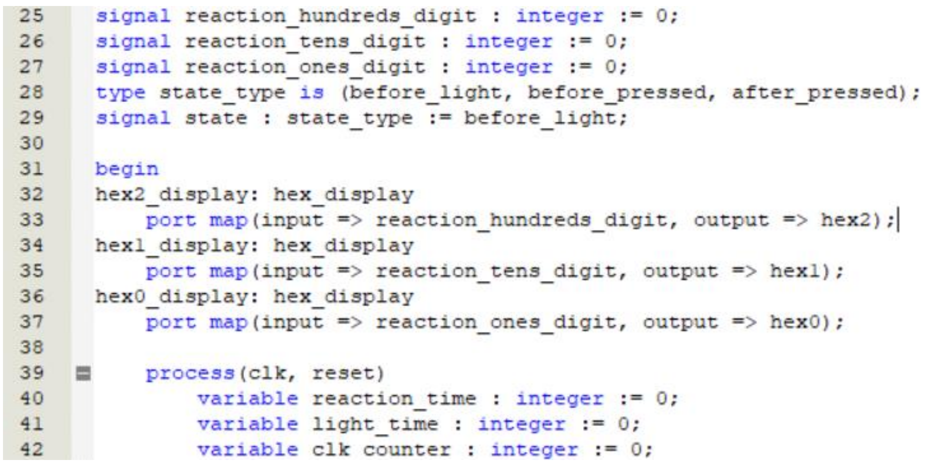
Part III

Entity and function import:



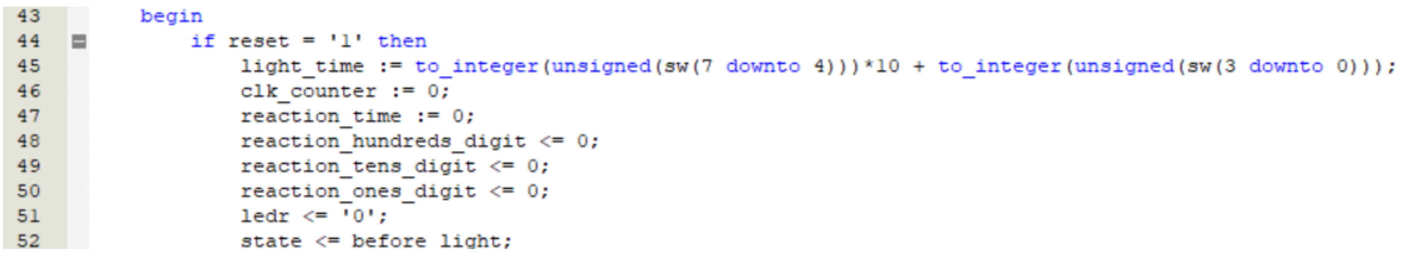
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity part3 is
    port(
        reset : in std_logic;
        clk : in std_logic;
        sw : in std_logic_vector(7 downto 0);
        freeze : in std_logic;
        ledr : out std_logic;
        hex2, hex1, hex0 : out std_logic_vector(7 downto 0)
    );
end entity;

architecture behavior of part3 is

component hex_display is
    port(
        input : in integer;
        output : out std_logic_vector(7 downto 0)
    );
end component;
```

Signal and *variable* form variable declaration, and hex display:



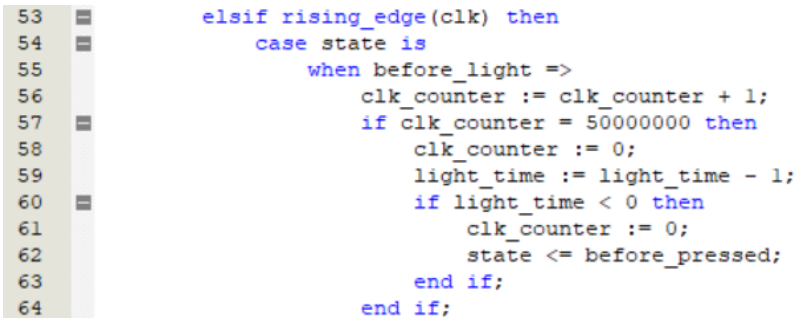
```vhdl
signal reaction_hundreds_digit : integer := 0;
signal reaction_tens_digit : integer := 0;
signal reaction_ones_digit : integer := 0;
type state_type is (before_light, before_pressed, after_pressed);
signal state : state_type := before_light;

begin
hex2_display: hex_display
    port map(input => reaction_hundreds_digit, output => hex2);
hex1_display: hex_display
    port map(input => reaction_tens_digit, output => hex1);
hex0_display: hex_display
    port map(input => reaction_ones_digit, output => hex0);

    process(clk, reset)
        variable reaction_time : integer := 0;
        variable light_time : integer := 0;
        variable clk_counter : integer := 0;
```

Reset:



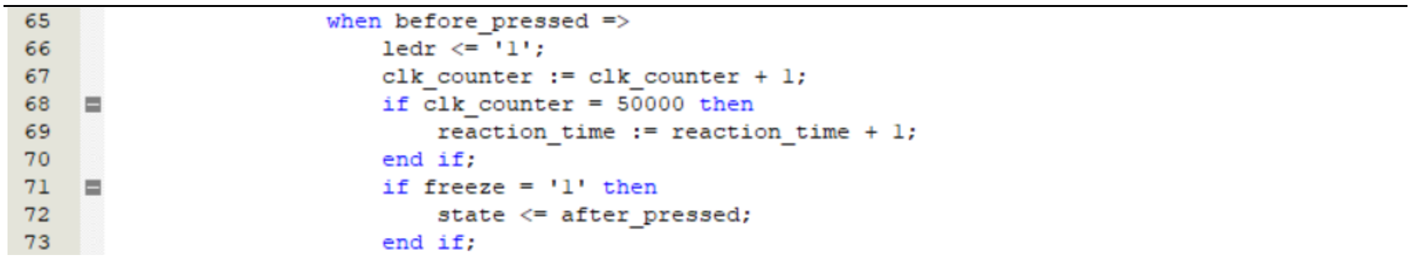
```vhdl
    begin
        if reset = '1' then
            light_time := to_integer(unsigned(sw(7 downto 4)))*10 + to_integer(unsigned(sw(3 downto 0)));
            clk_counter := 0;
            reaction_time := 0;
            reaction_hundreds_digit <= 0;
            reaction_tens_digit <= 0;
            reaction_ones_digit <= 0;
            ledr <= '0';
            state <= before_light;
```

Before_light state:



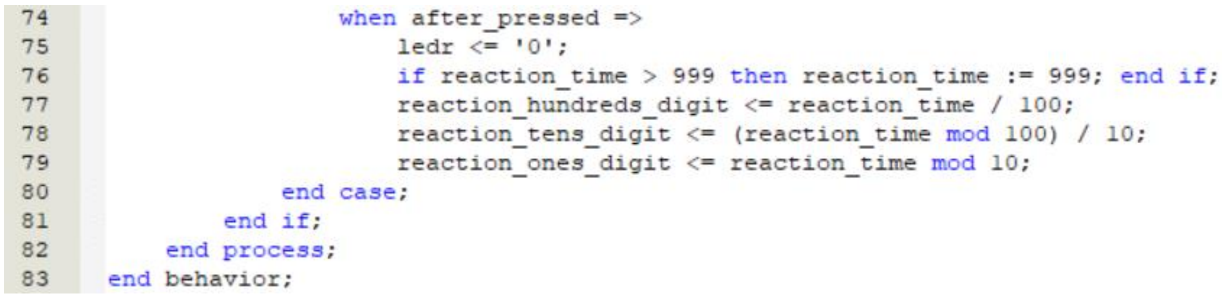
```vhdl
        elsif rising_edge(clk) then
            case state is
                when before_light =>
                    clk_counter := clk_counter + 1;
                    if clk_counter = 50000000 then
                        clk_counter := 0;
                        light_time := light_time - 1;
                        if light_time < 0 then
                            clk_counter := 0;
                            state <= before_pressed;
                        end if;
                    end if;
```

Before_pressed state:



```
                when before_pressed =>
                    ledr <= '1';
                    clk_counter := clk_counter + 1;
                    if clk_counter = 50000 then
                        reaction_time := reaction_time + 1;
                    end if;
                    if freeze = '1' then
                        state <= after_pressed;
                    end if;
```

After_pressed state:



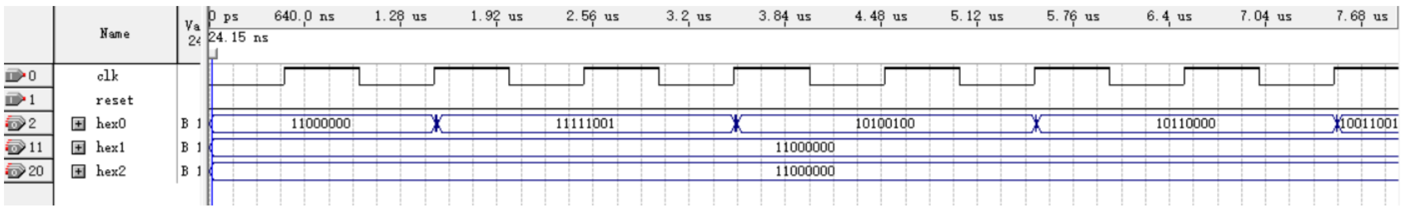
```
                when after_pressed =>
                    ledr <= '0';
                    if reaction_time > 999 then reaction_time := 999; end if;
                    reaction_hundreds_digit <= reaction_time / 100;
                    reaction_tens_digit <= (reaction_time mod 100) / 10;
                    reaction_ones_digit <= reaction_time mod 10;
            end case;
        end if;
    end process;
end behavior;
```

4. Simulation Result:

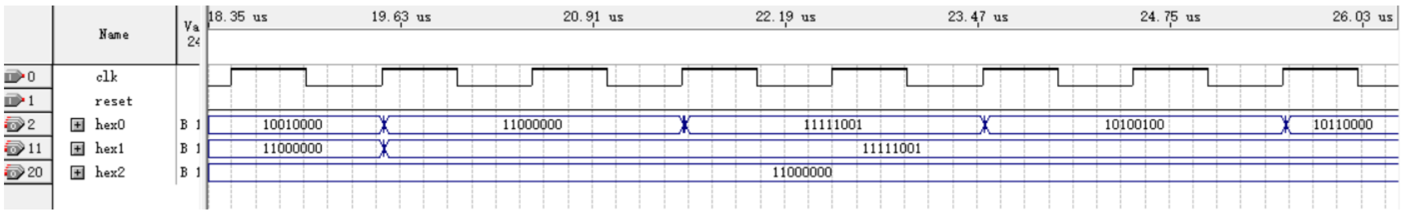
Part I

It is impossible to make counter work when 50 million rising edges pass in simulation, and thus I changed *50 million* to *2*.

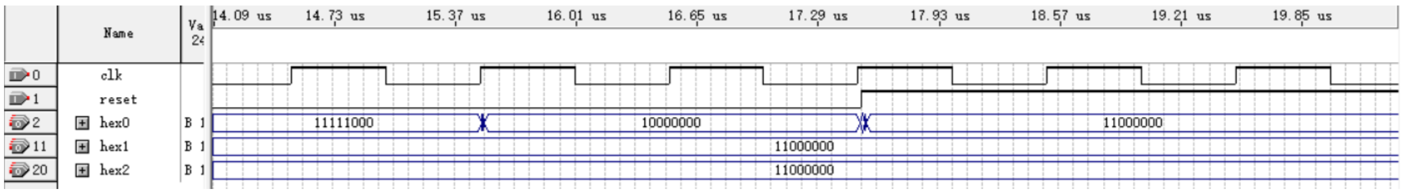
Timer:



Case:



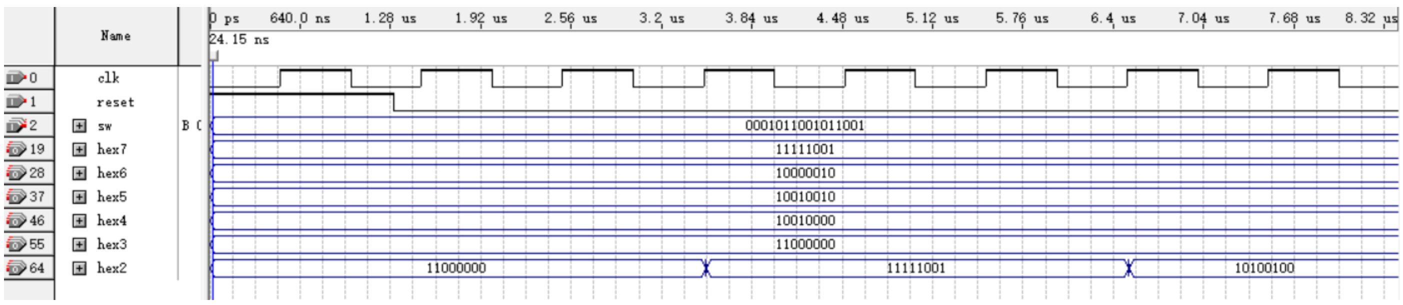
Reset:



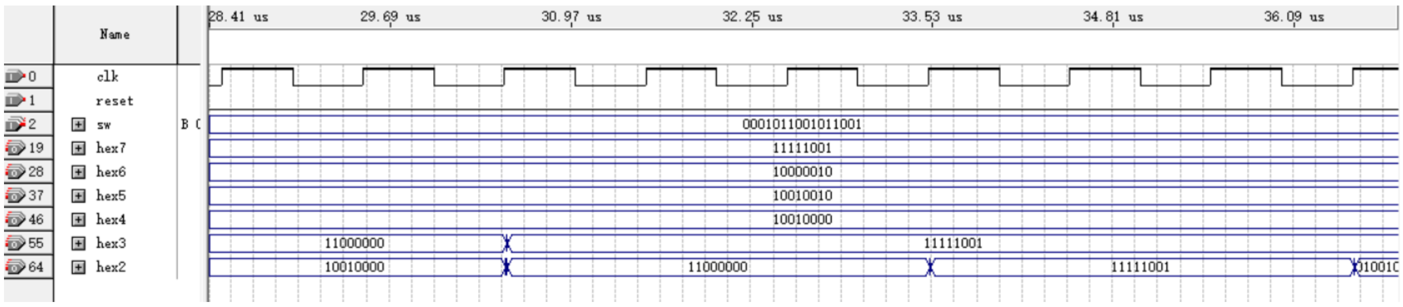
Part II

Change *50 million* to *2*:

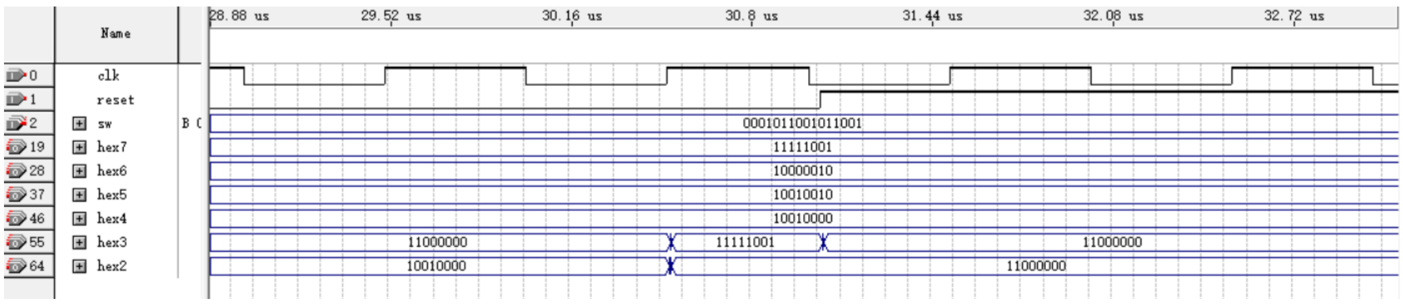
Timer:



Case:



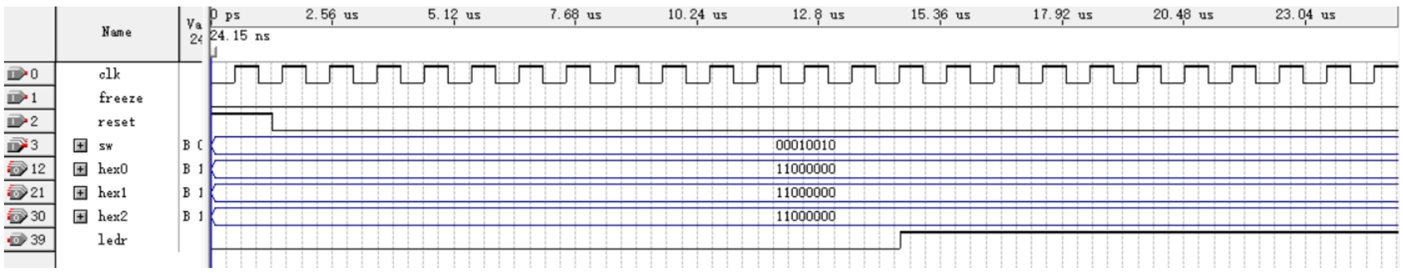
Reset:



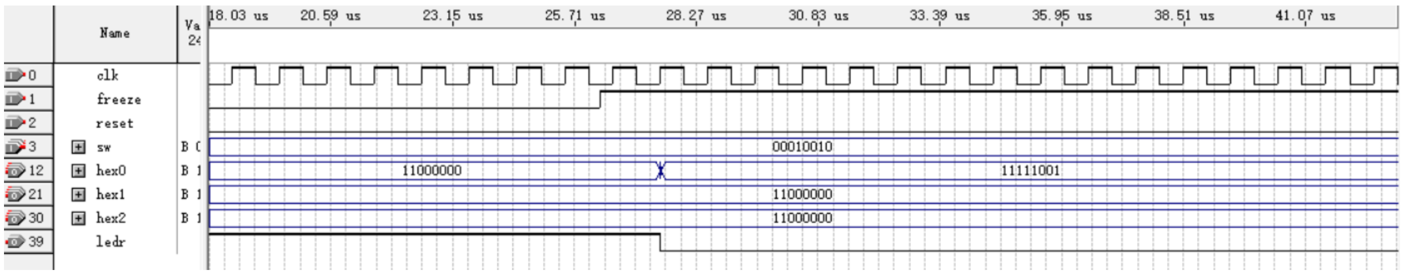
Part III

Change both *50 million* and *50 thousand* in the code to *1*:

LEDR lights:

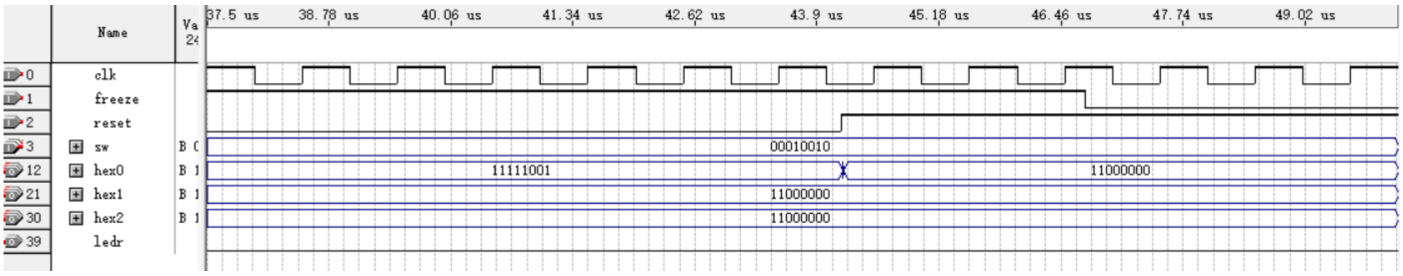


When KEY3 is pressed:



Due to the fact that the hex output is in the form of signal type variables *reaction_cundreds_digit*, *reactions_tens_digit* and *reactions_ones_digit*, while signal type variables do not update before the end of a process, resulting in an unavoidable delay of approximately one to two clock rising edges. However, this doesn't matter when applicating in real, because we need 50 thousand clock rising edges to add 1 to the variable *reaction_time* itself, which is a dramatically large number comparing with 1 or 2.

Reset:

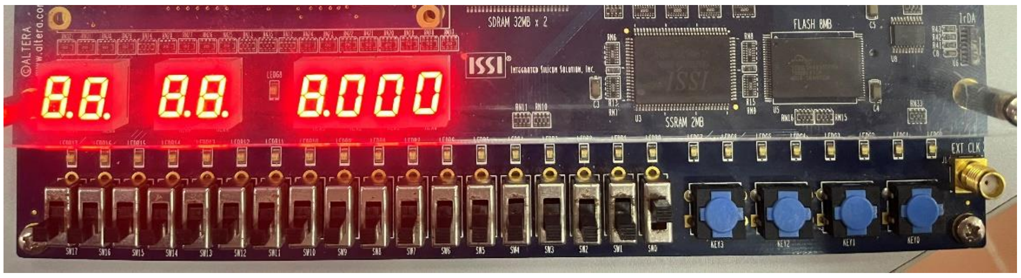


5. Experimental Results:

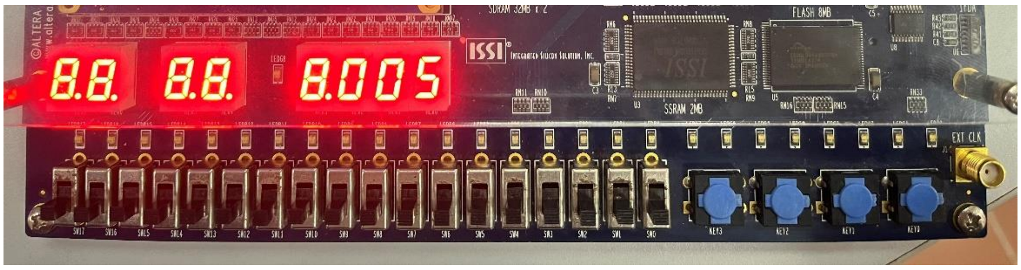
Part I

Note: Change KEY0 to SW0 as reset button.

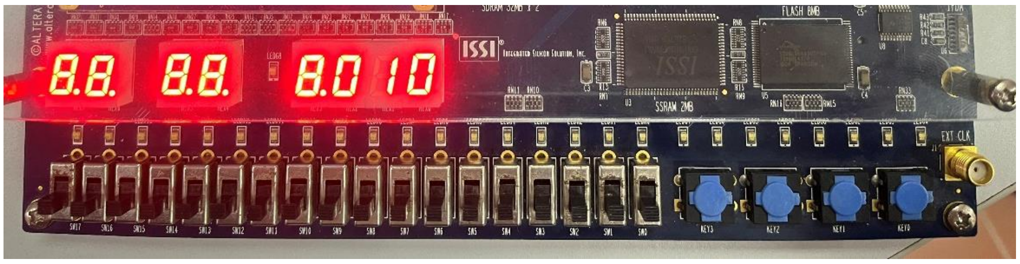
Reset:



Timer:



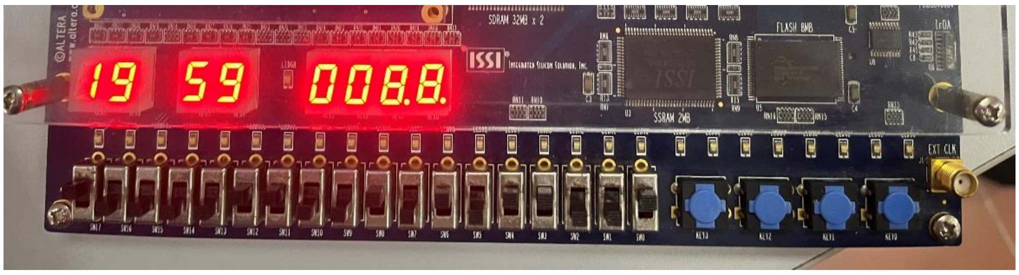
Case:



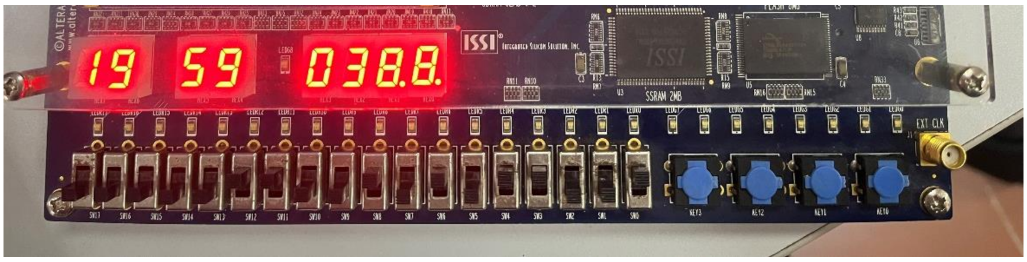
Part II

Start time: 19:59:00.

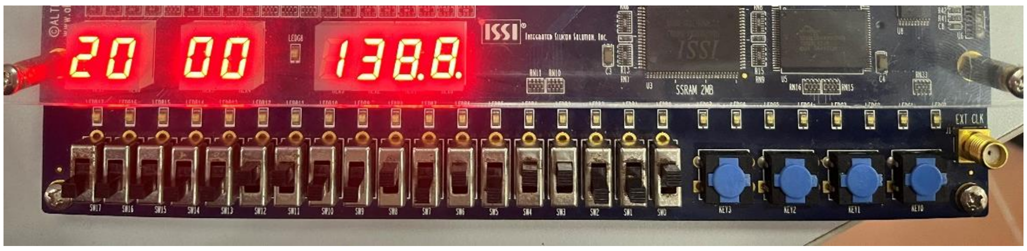
Reset (change KEY0 to SW17 as reset):



Timer:

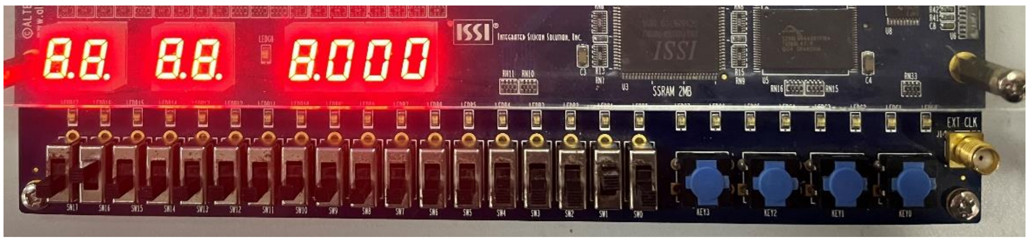


Case of second-bit, minute-bit and hour-bit:

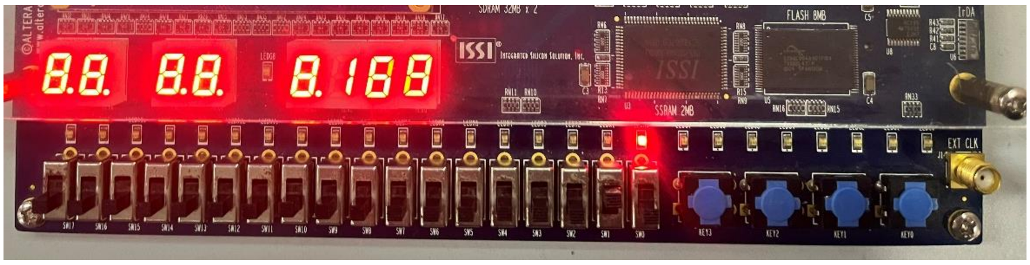


Part III

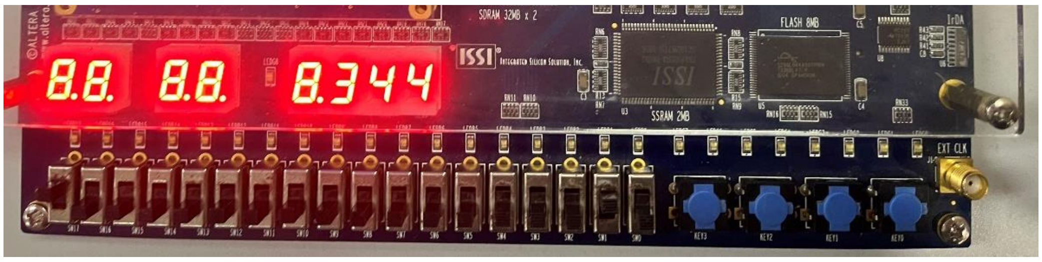
Reset (change KEY0 to SW16 as reset):



LEDR triggers:



Freeze (change KEY3 to SW17 as freezing button):

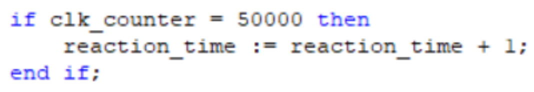


6. Discussion and Conclusion:

Discussion:

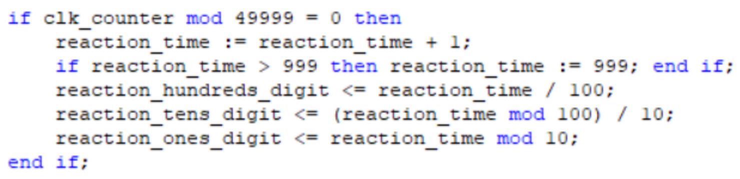
A code fault in Part III, line 68.

Change:



```
if clk_counter = 50000 then
    reaction_time := reaction_time + 1;
end if;
```

To:



```
if clk_counter mod 49999 = 0 then
    reaction_time := reaction_time + 1;
    if reaction_time > 999 then reaction_time := 999; end if;
    reaction_hundreds_digit <= reaction_time / 100;
    reaction_tens_digit <= (reaction_time mod 100) / 10;
    reaction_ones_digit <= reaction_time mod 10;
end if;
```

Three mistakes have been corrected:

(1) 50000->49999: This is a tiny mistake because one clock rising edge is ignorable comparing with one second.

(2) Add code that calls back to 0 when counter reaches 50000, otherwise the if statements in the first code cannot be triggered.

(3) Add code for hex display when LEDR lights and timer runs.

Conclusion:

In this lab exercise on Clocks and Timers, we successfully implemented a 3-digit BCD counter, a time-of-day clock, and a reaction time game using VHDL. The BCD counter and clock displayed accurate time progression on HEX displays after respective resets. For the reaction time game, LEDR lit after a set delay, measuring player response in milliseconds. Adjustments were made to simulation parameters for practical testing. Minor code corrections improved functionality, ensuring counters reset properly and HEX displays updated accurately. This exercise demonstrated practical applications of timing circuits and reinforced VHDL programming skills.