EEE102-Term Project Report

CONTROL OF CHANGING ROOM IN GYM

31.12.2023

CEM ALABAŞ-22102072

Video Presentation: https://youtu.be/Dx9rEi7VSdE

Introduction and Description

The project's concept of limiting entrance to the gym is to make sure that people change in a room before utilizing the gym session. Since using the changing room before using the gym is required, this is meant to encourage better behaviours. Those who wants to use gym must go to the changing room since the entrance of gym part is closed. After entering the changing room, person is getting a password which is created by the VHDL. After that people can use the password for entering the gym. The password can be entered through the switches which are on the Basys3. If password is correct, person can utilize the gym session ,otherwise, the buzzer beeps and the door does not open. Additionally, the number of person in the gym area is counted and if the total number of people in the gym is above 4, the red led turns on. It shows that no further entry will allowed. Even Though, the Basys3 gives a password, the door does not open and buzzer beeps when the red led turns on. Otherwise, when the total number of person in the gym is under 5, the green led turn on. It shows that the gym area is available.

Components

1) Ultrasonic Sensors (HC-SR04)

2) Jumper Cables

3) Breadboard

4) LCD Displays

5) A serve motor

6) Basys 3 FPGA boards

7) Buzzer

8) Green Led

9) Red Led

Design Methodology and Specifications

Firstly, the motion detection part was handled. For this purpose I utilized the HC SR04 sensor. When Basys3 send a signal to the trig, trig is sending a sound wave and the Basys3 clock starts to count. After the sound wave encounters with the object, it returns to the echo. Calculating the total time with the clock of the Basys 3 and knowing the speed of the sound propogation in air, I adjsuted the distance witth the 96 centimeter.The sensor can detect movements up to 96 centimeters. After that, I handled with the password part. I utilized the linear feedback shift register logic for creating a password and it is shown on the Basys3 7 segment. After the creating a password, the serve motor is added to the project for controlling the door. I have written the VHDL code for PWM module so that the servo turned until the 90 degrees. In order to achieve required frequency of servo motor, clock divider code is written for converting the clock of the Basys3 100 Mhz to the 64 Khz. This is required for the enhancing the accuracy of rotation. After that the first stage of the project is done.

After the completing the first stage of the project, I utilized the my motion detection code with the adding the another HC SR04 sensor to the exit. It was added for the purpose of counting the total number of person in the gym. When person enters the changing room the total number is increasing 1 on the 7 segment display which is independent from Basys3. If someone exits the gym area, the HC SR04 sensor which is on the exit area is detecting the motion and decreasing the total number of people 1. In the last part, I added to green led and

red led to show the crowdedness of the gym. If the green led actives, it shows that gym area is available for usage ,otherwise, the red led turns on and no other entries will allowed. I have restricted the total number of people in the gym with 4. When the red light turns on, even if the Basys3 gives a new code and the person enters the correct code, the door does not open and the buzzer beeps.

Modules

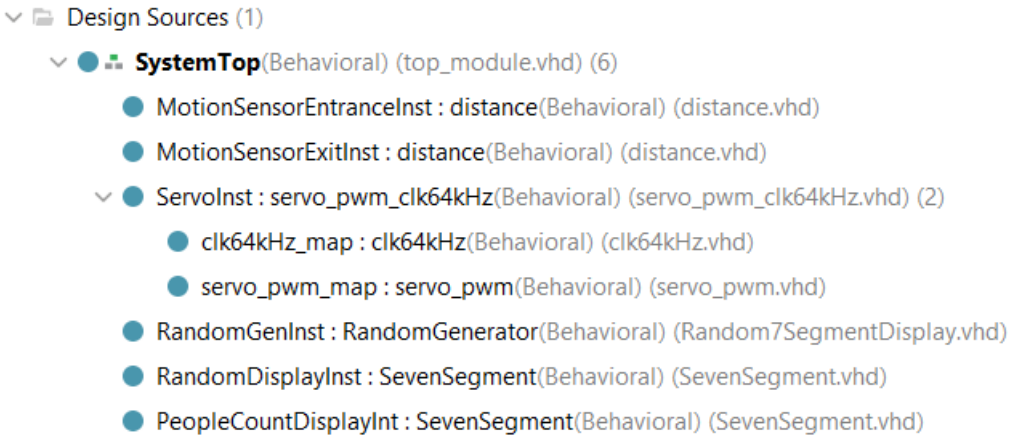

Figure 1 : All Modules

1) **Top Module:** All of the modules is connected here. This section deals with the activation of the servo motor depending on whether a password is given after motion is detected and whether the password is entered correctly or not. In addition, counting the total number of people and activating the red or green light accordingly is also handled in this section.
2) **Dıstance:** This code is written for the motion detection and it detects the motions in the range of 0-96 centimeter.
3) **Servo_pwm_clk64kHz:** This code is written fort he purpose of controlling the servo motor. First pulse width modulation part is handled after that the clock of the Basys 3 converted from 100 Mhz to 64 Khz. It is required for the accuracy of rotation.

4) RandomGenerator: It cannot generate randomly number, ıt uses the linear feedback shift register logic for giving a password.

5) SevenSegment: The seven segment codes is taken from the Lab 5.

The schematic of top module:

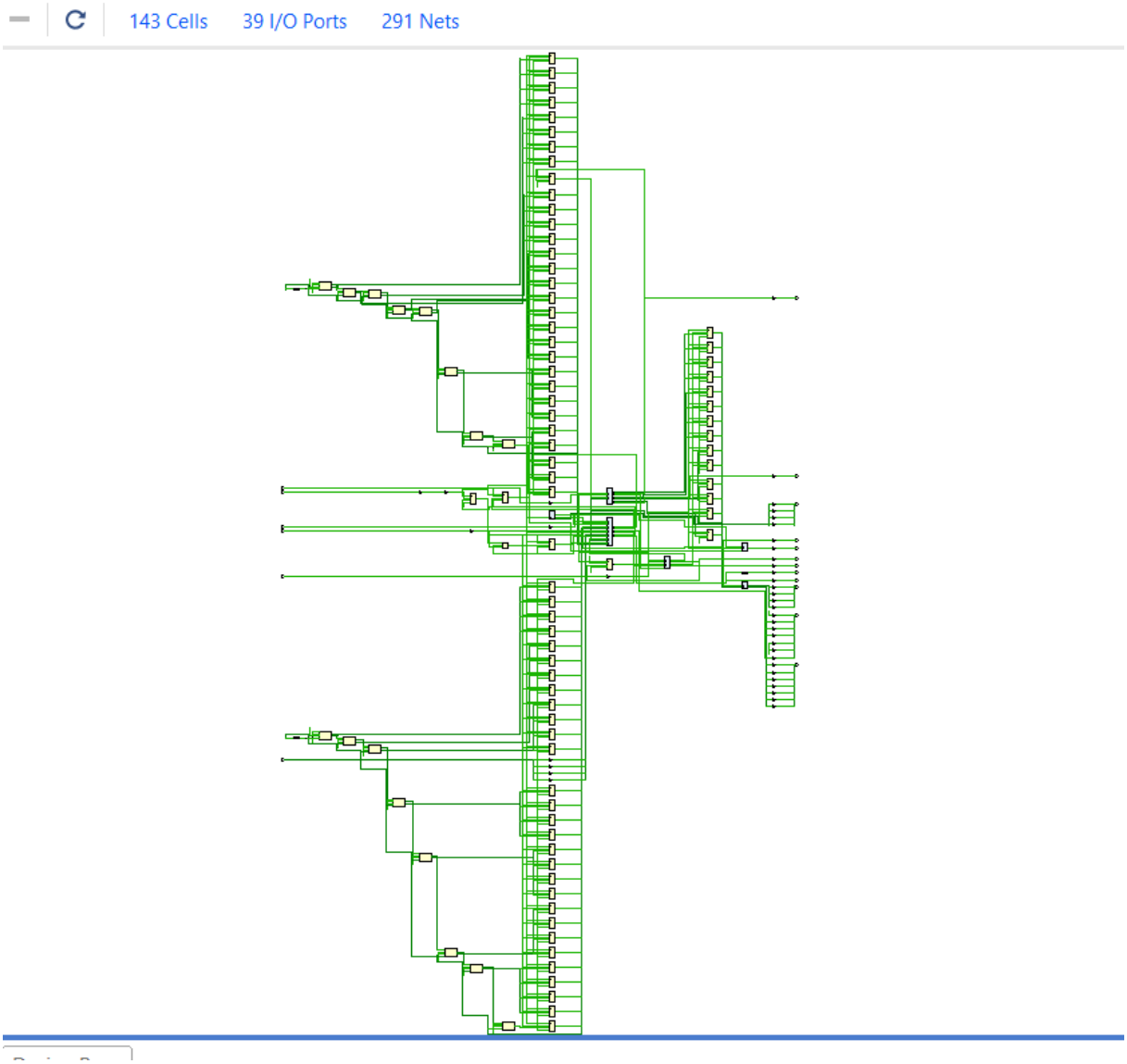

Figure 2: RTL Schematic of Project

Results

As previously said, in order to begin this process, I first wrote the VHDL codes for each submodule individually. Then I tested each code to see whether it functions. After the able to operate all of my codes, I combined the all of the codes on the top module.

In the figure 3.1, the motion detected and password is given on the Basys3 board. The red led turns on and The total number is increased 1.

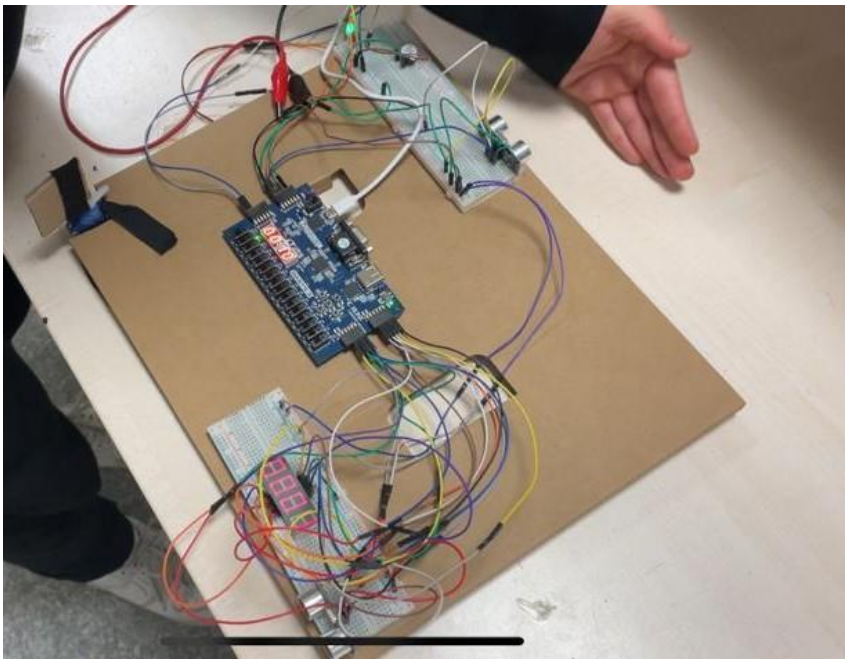

Figure 3.1

In figure 3.2, the password is entered correctly and the door is opening.

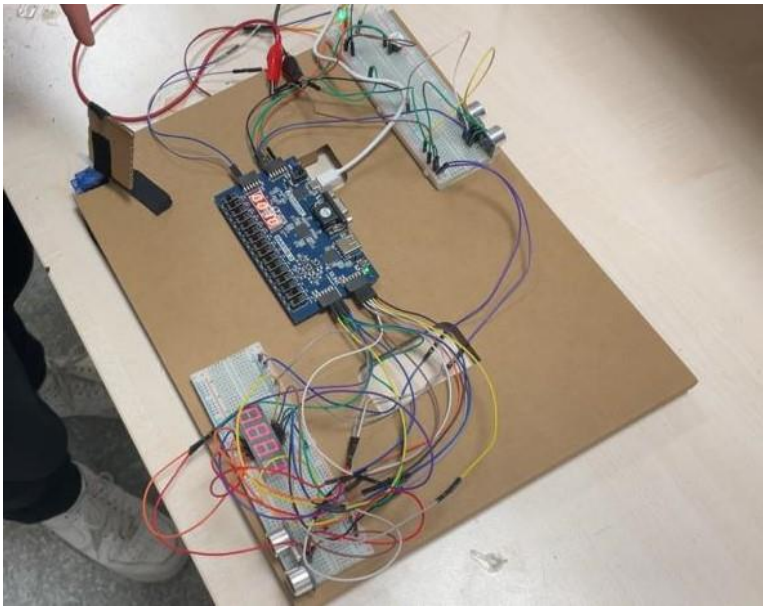

Figure 3.2

In figure 3.3, the total number of person in the gym is exceeded and the red led turns on.

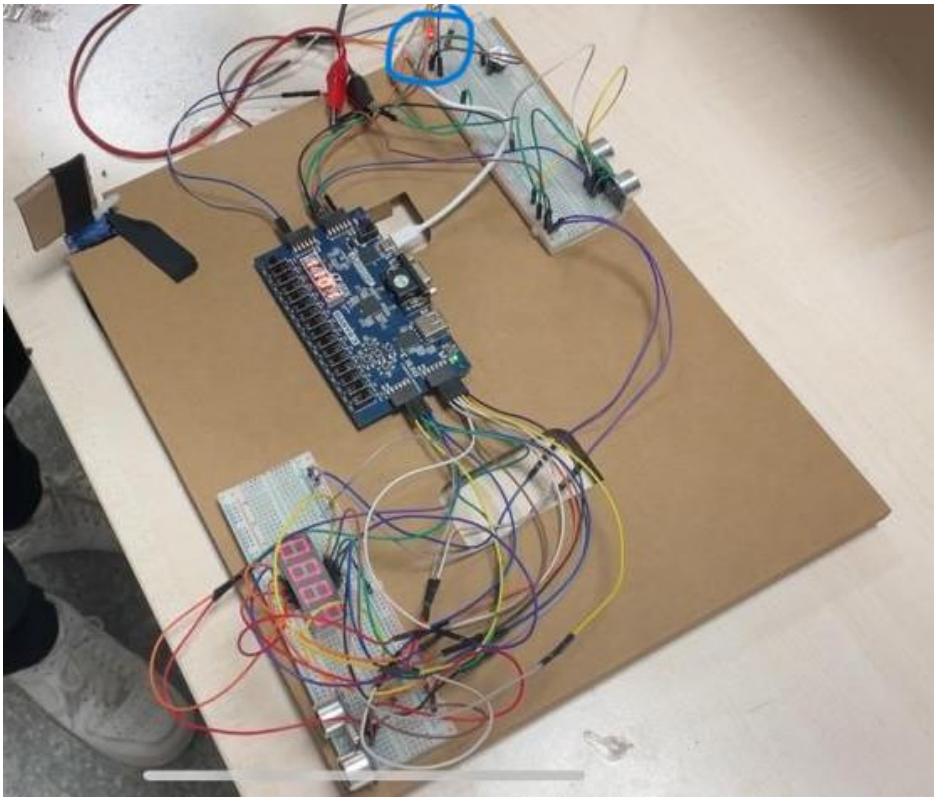

Figure 3.3

Conclusion

This project was aimed to regulate the acces to the gym, making it mandatory for individuals to utilize the changing room befor entering the workout area. This initiative aims to foster healthier practices by ensuring that gym users do not bypass the changing room prior to their exercise sessions. Ultrasonic sensor codes and servo motor codes was utilized for this project. Additionally, the research about the creating a random password is done and utilized with the linear feedback shift register. The utilized sources are on the references part. This project gave me important experience on FPGA board and VHDL coding.

Works Cited

Riley, Sean . “Random Numbers with LFSR (Linear Feedback Shift Register) - Computerphile.” *Www.youtube.com*, Computerphile , Sept. 10AD, www.youtube.com/watch?v=Ks1pw1X22y4. Accessed Dec. 1AD.

“Servomotor Control with PWM and VHDL.” *CodeProject*, 20 Dec. 2012, www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL. Accessed 31 Dec. 2023.

VHDL Code for Seven-Segment Display on Basys 3 FPGA. www.fpga4student.com/2017/09/vhdl-code-for-seven-segment-display.html.

“Vhdl-Ultrasonic Sensor(Hc-Sr04).” *Stack Overflow*, stackoverflow.com/questions/27580741/vhdl-ultrasonic-sensorhc-sr04. Accessed 31 Dec. 2023.

Appendices

top_module.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity SystemTop is
    Port ( clk: in std_logic;

        check: in std_logic;
        echoEntrance: in std_logic;
        trigEntrance: buffer std_logic;
        echoExit: in std_logic;
        trigExit: buffer std_logic;
        reset : in STD_LOGIC;
        code_switches : in std_logic_vector (3 downto 0);
        SS_Cathode_1 : out std_logic_vector (6 downto 0);
        SS_Anode_1: out std_logic_vector(3 downto 0);
        SS_Cathode_2 : out std_logic_vector (6 downto 0);
        SS_Anode_2: out std_logic_vector(3 downto 0);
        redLed: out STD_LOGIC;
        greenLed: out STD_LOGIC;
        exitLed: out STD_LOGIC;
        enterLed: out STD_LOGIC;
        buzzer: out STD_LOGIC;
```

```
        servo: out STD_LOGIC);
end SystemTop;

architecture Behavioral of SystemTop is

    signal increment_people_count : STD_LOGIC;
    signal object_detected_entrance : STD_LOGIC;
    signal decrement_people_count : STD_LOGIC;
    signal object_detected_exit : STD_LOGIC;
    signal random_gen_activator  : STD_LOGIC;
    signal is_checked : STD_LOGIC;
    signal buzzer_active : STD_LOGIC;
    signal random_4_bit :  std_logic_vector(3 downto 0);
    signal random_4_bit_0 :  std_logic_vector(3 downto 0);
    signal random_4_bit_1 :  std_logic_vector(3 downto 0);
    signal random_4_bit_2 :  std_logic_vector(3 downto 0);
    signal random_4_bit_3 :  std_logic_vector(3 downto 0);
    signal servoPos :  std_logic_vector(6 downto 0);
    constant BUZZER_WAIT_CYCLE : integer := 300000000; --buuzer will beep 3 second.
    constant SERVO_WAIT_CYCLE : integer := 500000000;-- servo motor turns on 5 second.
    signal buzzer_counter : integer range 0 to BUZZER_WAIT_CYCLE;
    signal servo_counter : integer range 0 to SERVO_WAIT_CYCLE;
    signal people_counter : integer range 0 to 9999;
    signal  people_counter_0 :  std_logic_vector(3 downto 0);
    signal  people_counter_1 :  std_logic_vector(3 downto 0);
```

```
    signal  people_counter_2 :  std_logic_vector(3 downto 0);
    signal  people_counter_3 :  std_logic_vector(3 downto 0);
     signal last_enter_state : STD_LOGIC := '0';
     signal last_exit_state : STD_LOGIC := '0';

    constant PEOPLE_COUNTER_START_WAIT_CYCLE : integer := 50000000;
    signal people_counter_start_wait_counter : integer range 0 to
          PEOPLE_COUNTER_START_WAIT_CYCLE;
begin

    MotionSensorEntranceInst : entity work.distance
        Port map (
            clk => clk,
            trig => trigEntrance,
            echo => echoEntrance,
            data => object_detected_entrance
        );

         c : entity work.distance
        Port map (
            clk => clk,
            trig => trigExit,
            echo => echoExit,
            data => object_detected_exit
        );
```

```
-- I have two different motion detect entrance and exit so ı defined
   MotionSensorEntranceInst and MotionSensorEntranceInst


ServoInst : entity work.servo_pwm_clk64kHz
  Port map (
    clk => clk,
    reset => reset,
    pos => servoPos,
    servo => servo
  );



RandomGenInst : entity work.RandomGenerator
  Port map (
    clk => clk,
    reset => reset,
    button => random_gen_activator ,
    random4Digit => random_4_bit
  );


RandomDisplayInst : entity work.SevenSegment
  Port map (
    clk => clk,
    isAnode => '1',
    digit0 => random_4_bit_0,
    digit1 => random_4_bit_1,
```

```
        digit2 => random_4_bit_2,
        digit3 => random_4_bit_3,
        SS_Cathode =>  SS_Cathode_1,
        SS_Anode => SS_Anode_1

    );



PeopleCountDisplayInt : entity work.SevenSegment
    Port map (
        clk => clk,
        isAnode => '0',
        digit0 => people_counter_0 ,
        digit1 => people_counter_1,
        digit2 => people_counter_2,
        digit3 => people_counter_3,
        SS_Cathode =>  SS_Cathode_2,
        SS_Anode => SS_Anode_2
    );
  random_4_bit_0 <= (3 downto 1 => '0', 0 => random_4_bit(0));
  random_4_bit_1 <= (3 downto 1 => '0', 0 => random_4_bit(1));
  random_4_bit_2 <= (3 downto 1 => '0', 0 => random_4_bit(2));
  random_4_bit_3 <= (3 downto 1 => '0', 0 => random_4_bit(3));
  people_counter_0 <= std_logic_vector(to_unsigned(people_counter mod 10, 4)); --first
      digit
```

```
people_counter_1 <= std_logic_vector(to_unsigned(people_counter/10 mod 10, 4)); --
      second digit

people_counter_2 <= std_logic_vector(to_unsigned(people_counter/100 mod 10, 4)); --
      third digit

people_counter_3 <= std_logic_vector(to_unsigned(people_counter/1000 mod 10, 4));--
      fourth digit

buzzer <= buzzer_active;

redLed <= '1' when people_counter >= 5 else '0';

greenLed <= '1' when people_counter < 5 else '0';

-- I restricted the number the purpose of showing easily the different conditions, when
      number under 5, the green led will turn on.

random_gen_activator <= object_detected_entrance  when people_counter < 5 else '0';



   enterLed <= object_detected_entrance;

    exitLed <= object_detected_exit;







   process(clk)

    variable enter_action : boolean;

    variable exit_action : boolean;



begin



   if rising_edge(clk) then
```

```
enter_action := (object_detected_entrance = '1' and last_enter_state = '0');

last_enter_state <= object_detected_entrance;

-- I encountered same problem here (the random number part), so I defined a
last_enter_state, trying to synchronize the situations.

-- When I did not this, the number was increased 2 or 3 three times or decreased same
way.


exit_action := (object_detected_exit = '1' and last_exit_state = '0');

last_exit_state <= object_detected_exit;


if(enter_action) then

    people_counter <= people_counter + 1;

end if;


if(exit_action) then

   if ( people_counter > 0) then

    people_counter <= people_counter - 1; end if;

end if;



if (buzzer_active  = '1') then

   buzzer_counter <= buzzer_counter + 1;

   if (buzzer_counter = BUZZER_WAIT_CYCLE) then

      buzzer_active <= '0';

   end if;

else
```

```
    if (is_checked = '0' and check = '1') then
        is_checked <= '1';



        if ((random_4_bit xor code_switches) = "0000" and (people_counter < 5) ) then
            servoPos <= "1000000"; --The servo motor is adjusted to 90 degrees.
            servo_counter <= 0;
        else
            buzzer_active <= '1';
            buzzer_counter <= 0;
        end if;



    elsif (is_checked = '1' and check = '0') then
        is_checked <= '0';
    end if;
end if;

if (servoPos = "1000000") then
        servo_counter <= servo_counter + 1;
    if (servo_counter = SERVO_WAIT_CYCLE) then
        servoPos <= "0000000";
        end if;
    end if;
end if;
```

```
    end process;


end Behavioral;
```

distance.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity distance is
  Port (clk: in std_logic;
        echo: in std_logic;
        trig: buffer std_logic;
        data: out std_logic
        );
end distance;


architecture Behavioral of distance is


signal time_echo: integer:=0;
--echo signal is defined the purpose of calculating the distance.
begin
process(clk, echo)
variable constant1,constant2, constant3:integer:=0;
```

```
variable y :std_logic:='0';
begin



if rising_edge(clk) then
if(constant1=0) then
   trig<='1';


elsif(constant1=10000) then
   trig<='0';
   y:='1';
--in the basys3 and trig signal, we have 10 microsecond delay so that this constant is defined.
elsif(constant1=10000000) then
   constant1:=0;
   trig<='1';
--giving enough time for the sound wave which is coming out from trig signal and coming
        back to the echo.
end if;
constant1:=constant1+1;


if(echo = '1') then
   constant2:=constant2+1;
end if;


if(echo = '0' and y='1') then
   time_echo<= constant2;
```

```
    constant2:=0;
    y:='0';
end if;



if  (time_echo < 280000)then
    data <= '1';
--echoo = 280000, this calculation represent the 96.3 cm it means that echo signal can
          recognize the targets under 96.3 cm
else
    data <= '0';
end if;
end if;

end process;
end Behavioral;
```

servo_pwm_clk64kHz.vhd

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


entity servo_pwm_clk64kHz is
    Port (
        clk  : in  STD_LOGIC;
        reset: in  STD_LOGIC;
        pos  : in  STD_LOGIC_VECTOR(6 downto 0);
```

```
        servo: out STD_LOGIC
    );
end servo_pwm_clk64kHz;

architecture Behavioral_alt of servo_pwm_clk64kHz is
    Component clk64kHz_comp
        Port (
            clk    : in  STD_LOGIC;
            reset  : in  STD_LOGIC;
            clk_out: out STD_LOGIC
        );
    End Component;

    Component servo_pwm_comp
        Port (
            clk   : in  STD_LOGIC;
            reset : in  STD_LOGIC;
            pos   : in  STD_LOGIC_VECTOR(6 downto 0);
            servo : out STD_LOGIC
        );
    End Component;

    signal modulated_clk : STD_LOGIC := '0';
begin
    clock_generator: clk64kHz_comp port map(
```

```
        clk => clk,
        reset => reset,
        clk_out => modulated_clk
    );

    pwm_controller: servo_pwm_comp port map(
        clk => modulated_clk,
        reset => reset,
        pos => pos,
        servo => servo
    );
end Behavioral_alt;
```

clk64kHz.vhd

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

entity clk64kHz is
    Port (
```

```
        clk    : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end clk64kHz;
 --Becausd of the pulse width modulation(PWM), we need to change the clock frequency
          since we need a particular frequency for precise control of the motor position.
architecture Behavioral of clk64kHz is
    signal temp: STD_LOGIC;
    signal counter : integer range 0 to 780 := 0;
begin
    divid_frequency: process (reset, clk) begin
        if (reset = '1') then
            temp <= '0';
            counter  <= 0;
        elsif rising_edge(clk) then
            if (counter = 780) then
```

```
            temp <= NOT(temp);
            counter  <= 0;
        else
            counter <= counter + 1;
        end if;
      end if;
   end process;

   clk_out <= temp;
end Behavioral;
```

servo_pwm.vhd

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

entity servo_pwm_alt is
   Port (
      clk   : in  STD_LOGIC;
      reset : in  STD_LOGIC;
      pos   : in  STD_LOGIC_VECTOR(6 downto 0);
```

```
        servo : out STD_LOGIC
    );
end servo_pwm_alt;

architecture Behavioral_alt of servo_pwm_alt is
    -- the constants are defined because of the clearity.
    constant MAX_CNT: unsigned(10 downto 0) := to_unsigned(1279, 11);
    constant OFFSET: unsigned(7 downto 0) := to_unsigned(32, 8);

    signal cnt : unsigned(10 downto 0) := (others => '0');
    signal pwm_signal: unsigned(7 downto 0);
begin
    -- PWM signal computation
    pwm_signal <= unsigned('0' & pos) + OFFSET;

    -- Counter process
    pwm_counter: process(clk, reset)
    begin
        if reset = '1' then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            cnt <= (cnt = MAX_CNT) ? (others => '0') : cnt + 1;
        end if;
    end process;
```

```
    -- controlling the servo signal
    servo_control: process(cnt, pwm_signal)
    begin
        servo <= '1' when cnt < pwm_signal else '0';
    end process;
end Behavioral_alt
```

RandomGenerator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;

entity RandomGenerator is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           button : in STD_LOGIC; -- Button input
           random4Digit: out std_logic_vector (3 downto 0));

end RandomGenerator;

architecture Behavioral of RandomGenerator is
```

```
    signal lfsr : STD_LOGIC_VECTOR (3 downto 0) := "0000"; --lfsr represents the linear shift feedback register.

    signal last_button_state : STD_LOGIC := '0';



    begin



    random4Digit <= lfsr;




    process(clk, reset)

    variable button_pressed : boolean;

    -- for the purpose of synchronizing the clock signal and the button, ı am defining a new variable.



    begin



        if reset = '1' then

            lfsr <= "0000";

        elsif rising_edge(clk) then



            button_pressed := (button = '1' and last_button_state = '0');

            last_button_state <= button;

-- if button is pressed and last state is 0 then the number will be generated ,otherwise, ı encountered such problem that the number genereta 2 or 3 times when ı pressed te button

            if last_button_state = '1' and button = '0' and lfsr <= "0000" then
```

```
            lfsr <= "0001"; --defining initial condition.
        end if;
        if button_pressed then

            lfsr(3 downto 1) <= lfsr(2 downto 0);
            lfsr(0) <= lfsr(3) xor lfsr(2);
            --shifting operation.
        end if;
      end if;
   end process;
end Behavioral;
```

SevenSegment.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;


entity SevenSegment is
   Port ( clk : in STD_LOGIC;
        isAnode: in STD_LOGIC;
        digit0: in std_logic_vector (3 downto 0);
```

```
        digit1: in std_logic_vector (3 downto 0);
        digit2: in std_logic_vector (3 downto 0);
        digit3: in std_logic_vector (3 downto 0);
        SS_Cathode : out std_logic_vector (6 downto 0);
        SS_Anode: out std_logic_vector(3 downto 0));
end SevenSegment;

architecture Behavioral of SevenSegment is

   signal SSCase : integer range 0 to 3;
   constant SS_COUNT : integer := 100000;
   signal SSCounter : integer range 0 to SS_COUNT;
   signal digit : integer range 0 to 9;

   -- Random Number Generator Process
   begin
   SS_Anode <=  std_logic_vector(not to_unsigned(2**SSCase, 4)) when isAnode = '1' else
std_logic_vector(to_unsigned(2**SSCase, 4));

   process(SSCase)
   begin
      case SSCase is
   when 0 =>  digit <= to_integer(unsigned(digit0));
   when 1 =>  digit <= to_integer(unsigned(digit1));
   when 2 =>  digit <= to_integer(unsigned(digit2));
```

```
when 3 =>  digit <= to_integer(unsigned(digit3));
 end case;
end process;

process(clk)
begin

   if rising_edge(clk) then
       SSCounter <= SSCounter +1;
      if SSCounter = SS_COUNT then
         SSCase <= SSCase+1;
         SSCounter <= 0;
      end if;



   end if;
end process;


process(digit)
 begin
   case digit is
      when 0 =>
```

```
            SS_Cathode <=  "0000001";
        when 1 =>
            SS_Cathode <=  "1001111";
        when 2 =>
            SS_Cathode <=  "0010010";
        when 3 =>
            SS_Cathode <=  "0000110";
        when 4 =>
            SS_Cathode <=  "1001100";
        when 5 =>
            SS_Cathode <=  "0100100";
        when 6 =>
            SS_Cathode <=  "0100000";
        when 7 =>
            SS_Cathode <=  "0001111";
        when 8 =>
            SS_Cathode <=  "0000000";
        when 9 =>
            SS_Cathode <=  "0000100";
    end case;
  end process;
end Behavioral;
```