ELECTRICAL AND ELECTRONICS ENGINNERING DEPARTMENT

EEE 102-Digital Logic Design

Section 2

Term Project Report

ALARM SYSTEM

Efe Demirdişer

22202638

Video Presentation: https://youtu.be/buTXW7I-7DY (It's also submitted to STARS.)

Purpose

This project aims to implement an extensive alarm system in VHDL using Basys 3 FPGA. The system is triggered by a motion sensor and an ultrasonic sensor, and the alarm turns off with the correct password input.

Methodology

The first step was to determine the properties of the alarm system. A design that is triggered by motion only under a certain distance condition is selected. Therefore, HC-SR04 (ultrasonic sensor) and PIR (motion sensor) sensors must be used together at the same time. The distance limit for the alarm to turn on was selected to be 10 cm. A 4x4 keypad was used for the password input. Furthermore, in order to make the system more advanced and increase the security, a Finite State Machine (FSM) logic that locks the system for 10 seconds if an incorrect password is entered 3 times in a row, is used.

After the design of the system had been finalized, the VHDL codes for the different parts of the system were created and tested on Basys 3 FPGA, such as the motion sensor module, distance detection module, keypad module, countdown module, and seven segment display modules. If the test of the corresponding part is successful, the next module at the top is generated. Therefore, it can be said that the VHDL code of this project was designed in a modular fashion.

DESIGN SPESIFICATIONS

The explanations of the inputs and the outputs of the design are as follows:

INPUTS:

- enable: It is assigned to a switch of Basys 3 and enables the system. When it is '0', the alarm does not turn on even if the conditions are satisfied.

- sensor: It becomes ‘1’ for a few seconds when PIR motion sensor detects motion.
- clk: It’s the 100 MHz internal clock frequency of Basys 3.
- resett: It is assigned to a button on Basys 3. It’s used to delete the password input on the seven segment display, in order to try a different password.
- ssd_select (2 bits): These inputs are assigned to 2 switches of Basys 3. They are the select inputs for the seven segment display. We can switch between different modes for the seven segment display with this 2 bit select input. (1.Mode shows the password input by the user, 2. Mode shows the distance measured by the ultrasonic sensor and the 3. Mode shows the number of incorrect password inputs and the lock countdown) with this 2-bit select input.)
- shift: It is assigned to a button of Basys 3 and shifts the password input to the left by 1 digit.
- enter: It is assigned to a button of Basys 3 and enters the password present on the seven segment display.
- echo: It’s the input from the ultrasonic sensor. The distance is calculated from the time this input stays high logic level.

OUTPUTS:

- trigger: It’s the signal that triggers the ultrasonic sensor.
- an (4 bits): The signals for the anodes of seven segment display.
- seg (7 bits): The signals for the cathodes of seven segment display.
- Q: It becomes ‘1’ when alarm turns on and stays ‘1’ until the correct password is entered. It’s connected to the red lead of the RGB LED and the buzzer on the breadboard.
- Q_not: It’s the compliment of Q and connected to the green lead of the RGB LED.
- is_locked: When the system is locked, this output stays ‘1’ and goes back to ‘0’ when the 10 seconds is passed. It’s connected to the yellow LED on the breadboard.
- is_enabled: It shows whether the system is activated or not and it is connected to the blue LED on the breadboard.

INOUT:

- JA (8 bits): This is the input/output of the 4x4 keypad. The most significant 4 bits represent the rows and the least significant 4 bits represent the columns. The reason why

this is declared as inout is that we have to scan the columns of the keypad (output) and receive the row information of the pressed key (input) at the same time.

The hierarchy of the VHDL modules can be found in the below figure (Figure 1).

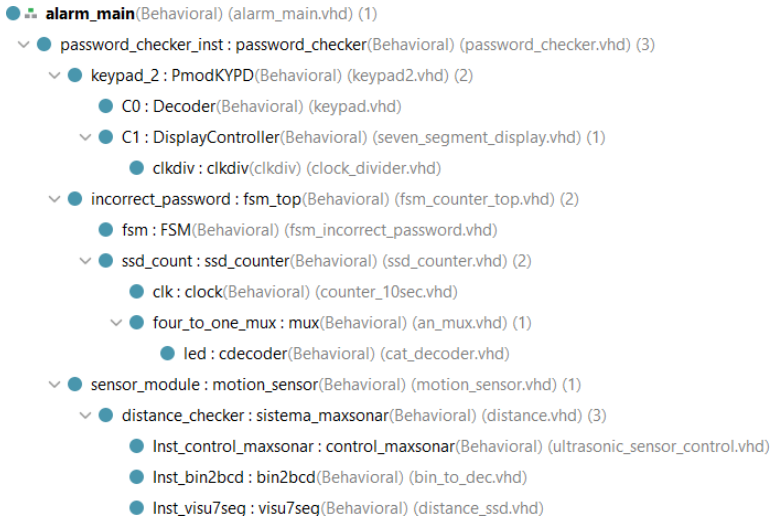

Figure 1: VHDL Hierarchy.

- alarm_main module takes the outputs from the password_checker module and assigns the corresponding values to Q, Q_not and is_enabled.

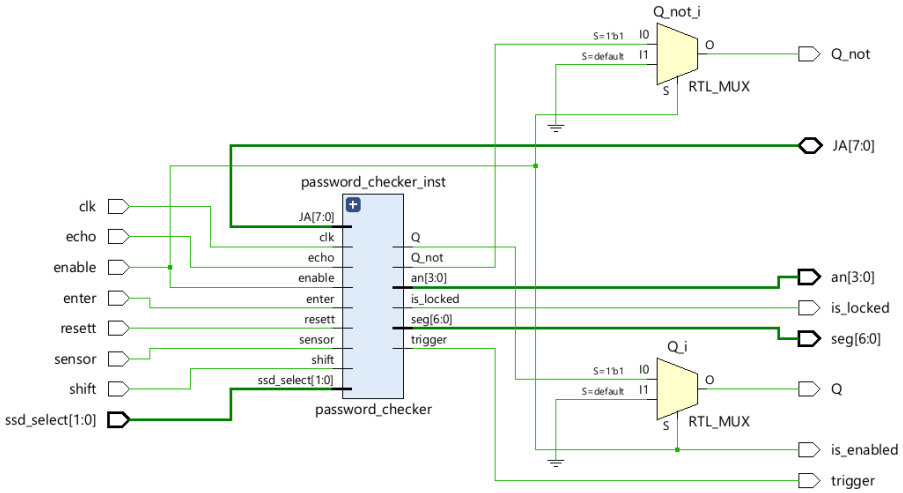

Figure 2: RTL Schematic of alarm_main module.

- password_checker module receives the sensor inputs from the motion_sensor module, 4-digit keypad input (32 bits in total) from PmodKYPD module and checks whether the password entered is correct, sends the check result to fsm_top module, receives the reset signal that will stop the alarm from fsm_top and sends it to motion_sensor module. Also, it involves the multiplexer for the 3 different seven segmet display options. The correct password (A7C9) is initialized in this module.

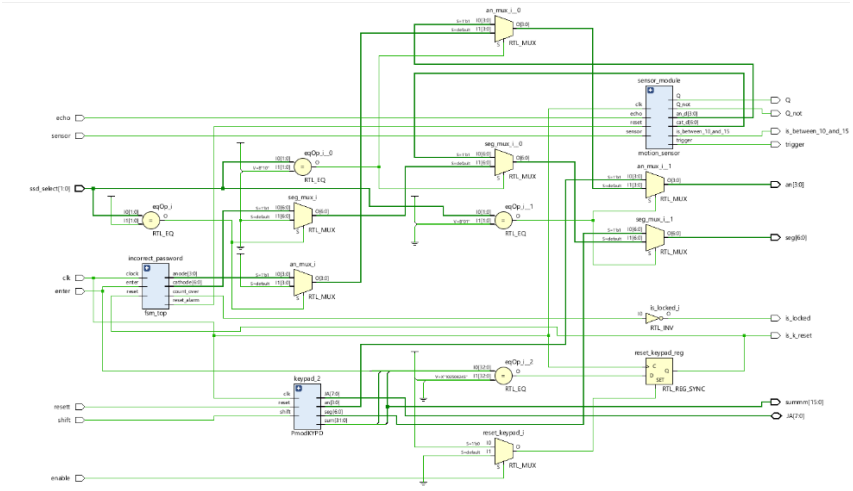

Figure 3: RTL Schematic of password_checker module.

- PmodKYPD module is the connection of Decoder module (keypad decoder) and DisplayController module (seven segment display module for the keypad input).

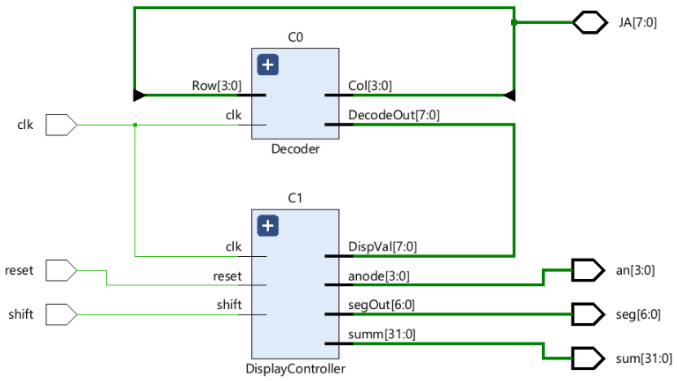

Figure 4: RTL Schematic of PmodKYPD module.

- Decoder module (keypad decoder) scans the columns of the keypad with the clock signal and outputs the corresponding 8-bit signal for the pressed key.

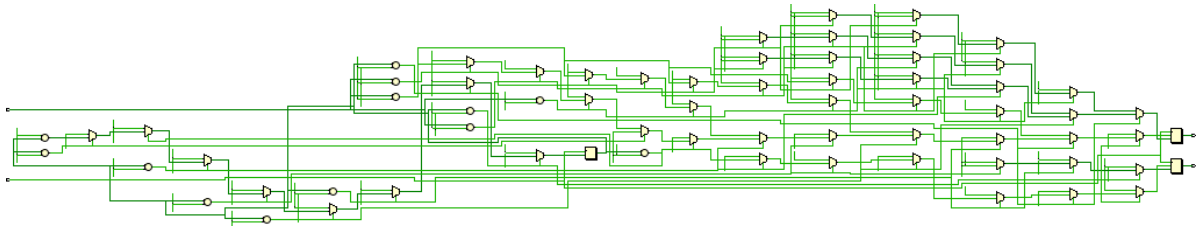

Figure 5: RTL Schematic of Decoder module.

- The DisplayController module takes the new clock from clkdiv module and uses it as clock, and outputs the corresponding anode and cathode outputs that will display the keypad input. A shift-register is used in order to show multiple digits of the password.

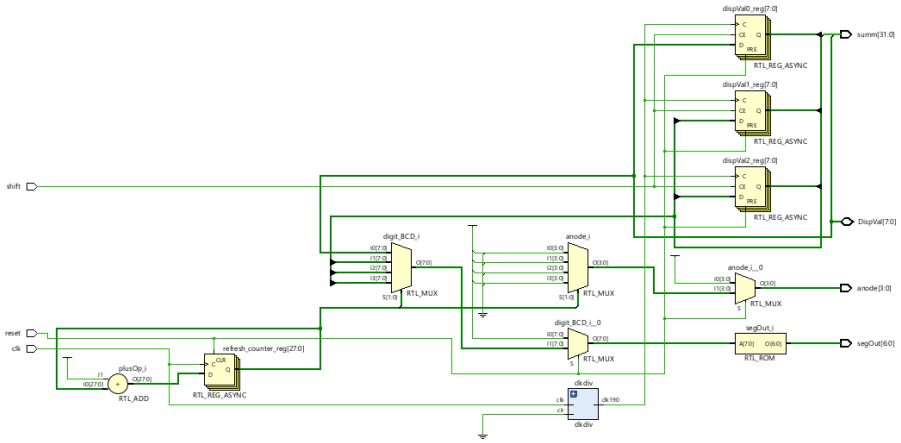

Figure 6: RTL Schematic of DisplayController module.

- Since the default internal clock frequency of Basys 3 is too high for the shift operation, a slower clock was generated in the clkdiv module. (Shift operation is implemented in every 0.5 second.)

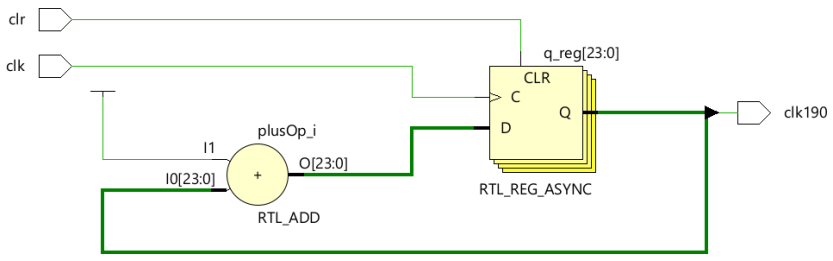

Figure 7: RTL Schematic of clkdiv module.

- fsm_top module connects the FSM and ssd_counter modules and outputs the reset signal of the alarm.

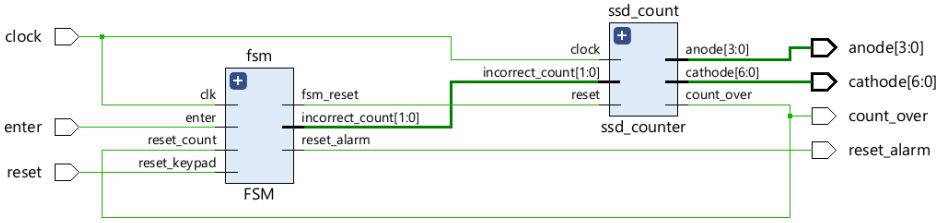

Figure 8: RTL Schematic of fsm_top module.

- FSM module traces the incorrect input count and when incorrectly entered password count reaches 3, it outputs a signal that will start the 10 second lock. Initially, 4 states were designed, which are state_0, state_1, state_2 and wait_lock. However, the transition from state_2 to state_0 was problematic. Since there was a timing issue between enter and reset_keypad signals, when the system is in state_2 and the password is entered correctly, it used to go to wait_lock state instead of state_0. Therefore, an additional state_2_hold state was added for self-correction.

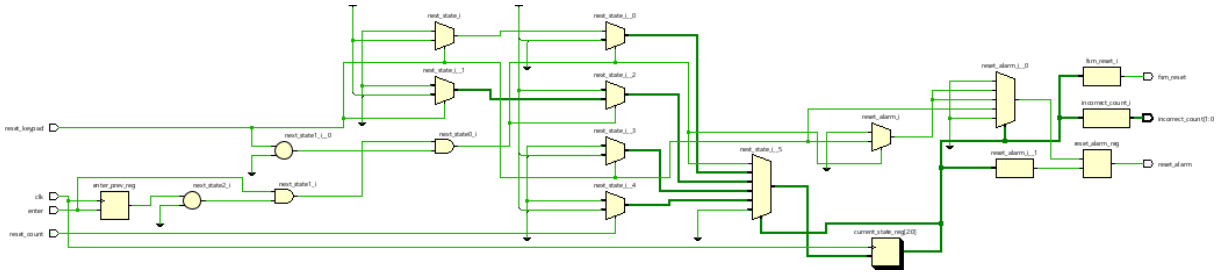

Figure 9: RTL Schematic of FSM module.

- ssd_counter module connects counter_10sec module (the module that increments the number of seconds passed in every second until it reaches 10 seconds) and the anode multiplexer module for seven segment display of the counter and the number of incorrect password inputs.

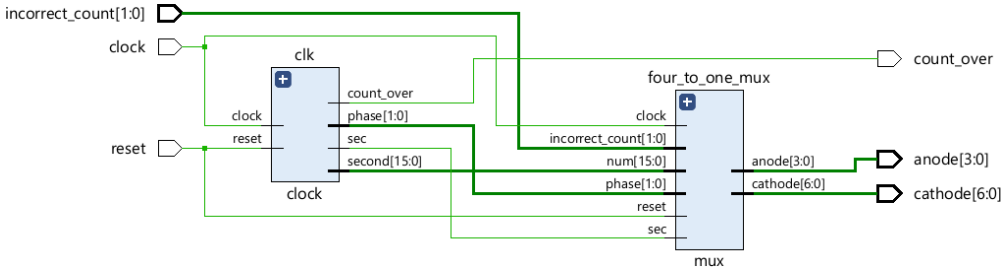

Figure 10: RTL Schematic of ssd_counter module.

- counter_10sec module is the modified version of the counter module that I used in EEE 102 Lab 5. The modification is that the counter stops when it reaches to 10, and outputs count_over signal which is used in FSM module for the state transition of the wait_lock state.

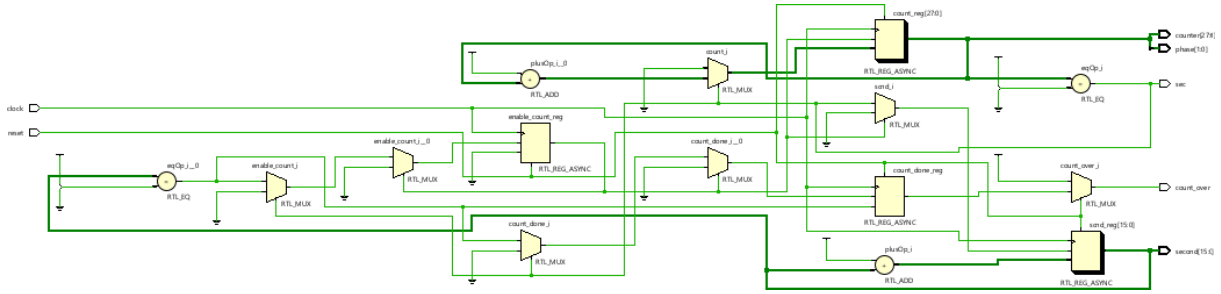

Figure 11: RTL Schematic of counter_10sec module.

- mux module involves the anode multiplexer and the cathode decoder module. It's the modified version of the anode multiplexer module that I used in EEE 102 Lab 5. The modification is that the first digit always shows the number of incorrect password input, whereas the last 2 digits are used for the countdown when current state is wait_lock.

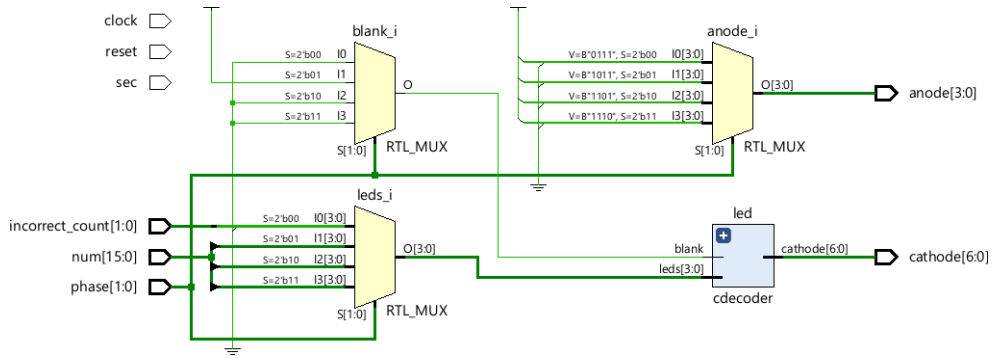

Figure 12: RTL Schematic of mux module.

- cdecoder module is the decoder for the cathode of seven segment display. It's the exact same code I used in EEE 102 Lab 5.

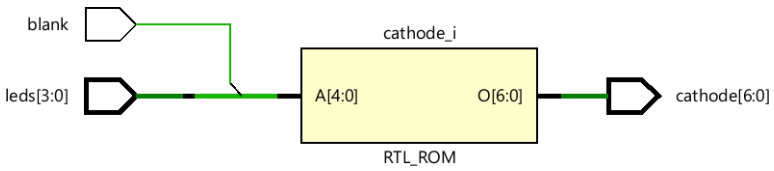

Figure 13: RTL Schematic of cdecoder module.

- motion_sensor module involves an FSM with 0 and 1 states in order to turn on the alarm. Since the alarm should stay on even after the motion stops, a latch-like behavior was

needed and an FSM was chosen in order to achieve that. This module also contains the submodule sistema_maxsonar. It takes the AND of is_less_than_10 and sensor signals to trigger the alarm.

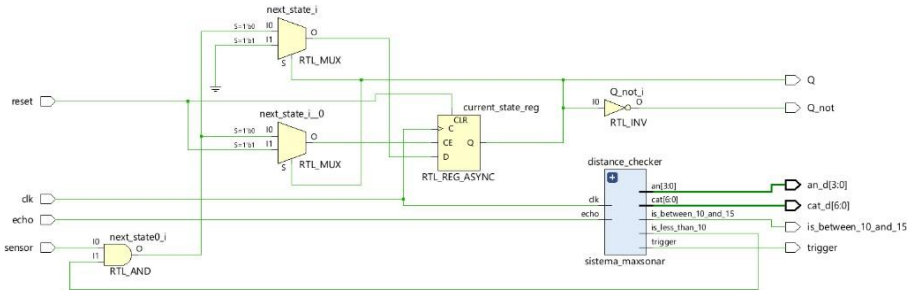

Figure 14: RTL Schematic of motion_sensor module.

- sistema_maxsonar module updates the measured distance value every time the distance changes and sends it to bin2bcd module, and then connects the bin2bcd and vsiu7seg modules together. It also involves the control_maxsonar submodule.

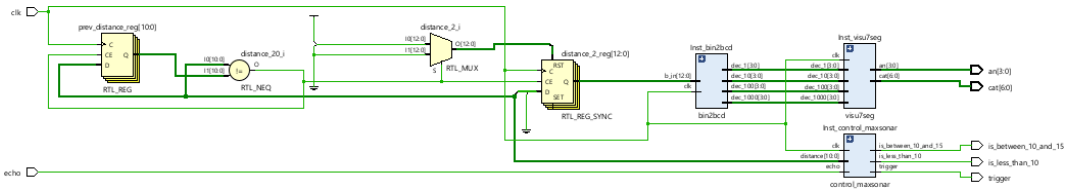

Figure 15: RTL Schematic of sistema_maxsonar module.

- control_maxsonar module triggers the HC-SR04 sensor, calculates the distance from the echo signal's length, and outputs is_less_than_10 signal that enables the alarm to turn on only when the distance is less than or equal to 10 cm. The AND operation of is_less_than_10 signal and the signal from the motion sensor enables the alarm to turn on, as mentioned in the description of motion_sensor module.

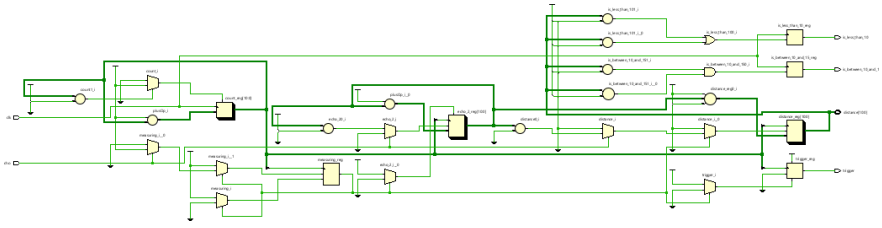

Figure 16: RTL Schematic of control_maxsonar module.

- visu7seg module is the seven segment display module for the distance calculated by control_maxsonar module. It receives the digits of the distance value separately from the bin2bcd module.

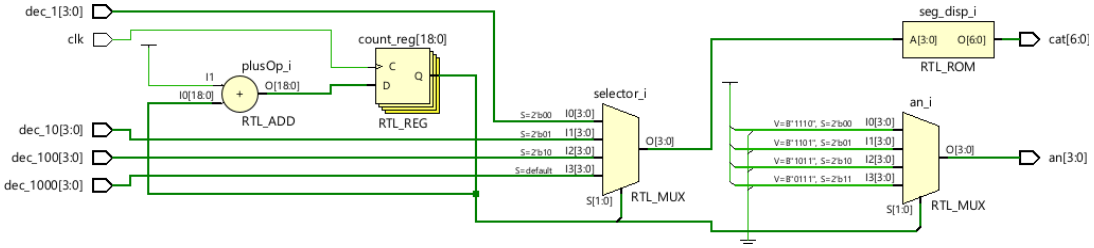

Figure 17: RTL Schematic of visu7seg module.

- bin2bcd module converts the binary distance value to binary coded decimal (BCD).

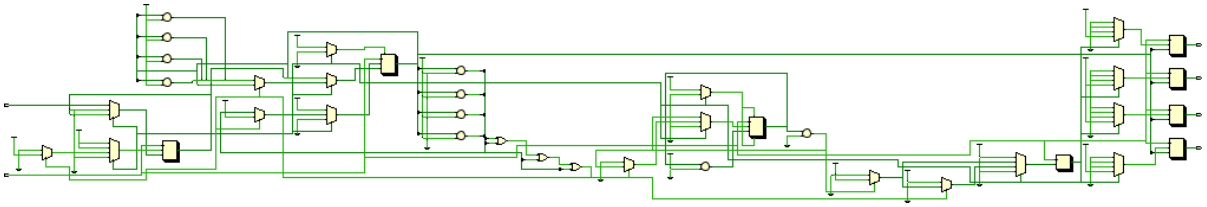

Figure 18: RTL Schematic of bin2bcd module.

RESULTS

After the VHDL codes for the design had been generated, the project was tested. The results of the project for all of the components were as expected. The results can be seen in the below examples.

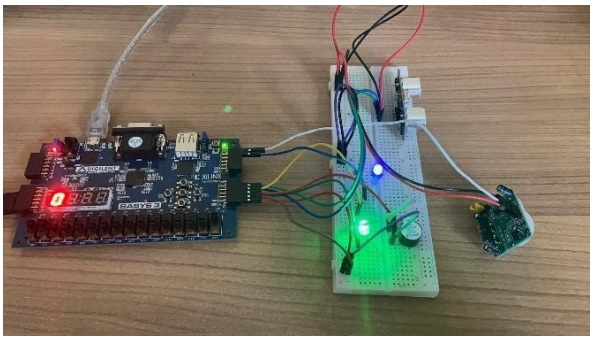

Figure 19: The alarm is in OFF state.

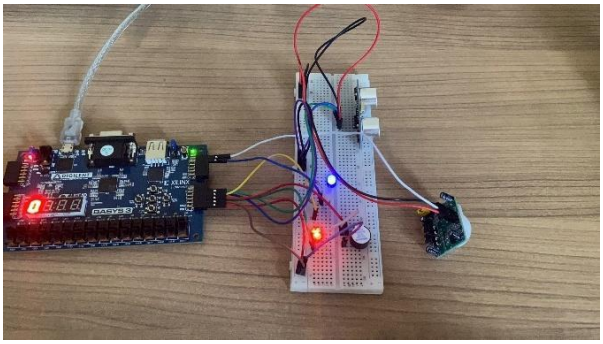

Figure 20: The alarm is in ON state.

Figure 21: The correct password is displayed on the seven segment display.

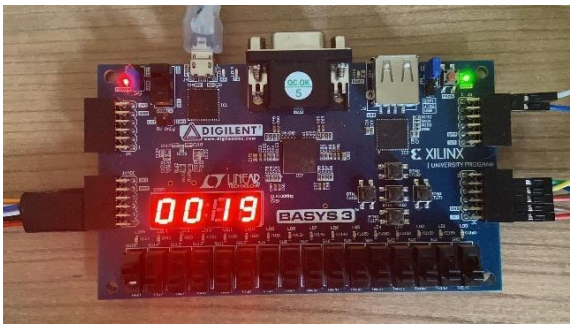

Figure 22: Current distance is displayed on the seven segment display (in the unit of cm).

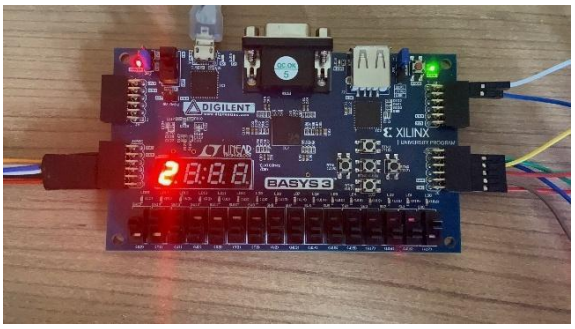

Figure 23: The number of incorrect password inputs is displayed on the seven segment display. Since it's 2, the count hasn't started.

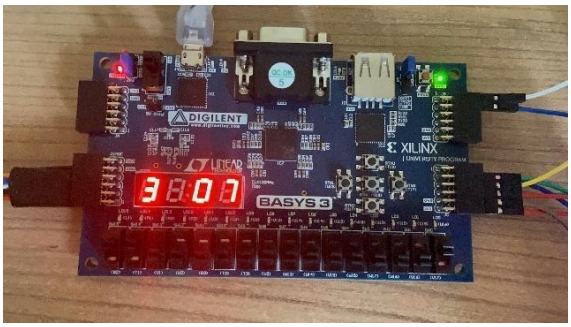

Figure 24: The number of incorrect password inputs is displayed on the seven segment display. The count has started and 7 seconds has passed.

CONCLUSION

In this project, I designed an extensive alarm system on Basys 3 FPGA using VHDL. I also used some additional components such as a PIR motion sensor, an HC-SR04 ultrasonic sensor, a 4x4 keypad, a buzzer, some LEDs, and a breadboard. Compared to a usual alarm system, my project had some additional properties like a distance condition together with the motion sensor, a limit for entering the password incorrectly (3 times in a row), and multiple seven segment display options such as the password input display, distance display, and the number of incorrect password inputs display. The project was successful since I was able to perform all of the properties that I stated in the proposal. In addition, this project was a great opportunity for me to implement the objectives covered in the lectures.

REFERANCES

[1] "Binary to BCD and BCD to Binary," Real Digital, [Online]. Available: https://www.realdigital.org/doc/6dae6583570fd816d1d675b93578203d. [Accessed 02 12 2024].

[2] "HC-SR501 PIR Sensor," Components 101, 18 07 2021. [Online]. Available: https://components101.com/sensors/hc-sr501-pir-sensor. [Accessed 21 11 2024].

[3] P. Marian, "HC-SR04: Datasheet, Specs, and More," [Online]. Available: https://www.electroschematics.com/hc-sr04-datasheet/. [Accessed 02 12 2024].

Appendix

Code 1: alarm_main.vhd

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


entity alarm_main is
    Port (
        enable     : in STD_LOGIC;
        sensor     : in STD_LOGIC;
        clk        : in STD_LOGIC;
        resett     : in STD_LOGIC;
        ssd_select : in STD_LOGIC_VECTOR(1 downto 0);
        shift      : in STD_LOGIC;
        enter      : in STD_LOGIC;   --
        JA         : inout STD_LOGIC_VECTOR (7 downto 0);
        echo : in  STD_LOGIC;
        trigger : out  STD_LOGIC;
        an     : out STD_LOGIC_VECTOR (3 downto 0);
        seg    : out STD_LOGIC_VECTOR (6 downto 0);
        Q      : out STD_LOGIC;
        Q_not  :  out  STD_LOGIC;
        is_locked : out STD_LOGIC;
        is_enabled : out STD_LOGIC
    );
end alarm_main;
```

```
architecture Behavioral of alarm_main is
signal Q_internal       : STD_LOGIC;
signal Q_not_internal   : STD_LOGIC;
signal summm_internal   : STD_LOGIC_VECTOR (15 downto 0);
signal a: STD_LOGIC;
signal b: STD_LOGIC;
signal c: STD_LOGIC;
signal d: STD_LOGIC;

begin

password_checker_inst : entity work.password_checker
    port map (
        clk    => clk,
        enter => enter,
        echo => echo,
        trigger => trigger,
        shift  => shift,
        ssd_select => ssd_select,
        sensor => sensor,
        Q      => Q_internal,
        Q_not  => Q_not_internal,
        summm  => summm_internal,
        JA     => JA,
        is_locked => is_locked,
        an     => an,
        seg    => seg,
        enable => enable,
        resett => resett,
        is_between_10_and_15 => c,
        is_k_reset => d
    );

Q      <= Q_internal when enable = '1' else '0';
Q_not  <= Q_not_internal when enable = '1' else '0';
```

```
is_enabled <= enable;

end Behavioral;
```

Code 2: password_checker.vhd

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

entity password_checker is
    Port (
        clk    : in STD_LOGIC;
        enable     : in STD_LOGIC;
        echo : in  STD_LOGIC;
        ssd_select : in STD_LOGIC_VECTOR(1 downto 0);
        trigger : out  STD_LOGIC;
        enter : in STD_LOGIC;
        shift  : in std_logic;
        resett  : in std_logic;
        sensor : in STD_LOGIC;
        Q : out STD_LOGIC;
        Q_not : out STD_LOGIC;
        is_locked : out STD_LOGIC;
        summm : out STD_LOGIC_VECTOR(15 downto 0);
        JA     : inout STD_LOGIC_VECTOR (7 downto 0);
        an     : out STD_LOGIC_VECTOR (3 downto 0);
        seg    : out STD_LOGIC_VECTOR (6 downto 0);
        is_between_10_and_15 : out STD_LOGIC;
        is_k_reset: out STD_LOGIC
    );
end password_checker;

architecture Behavioral of password_checker is

signal sum : std_logic_vector (32 downto 0);
```

```
signal reset: std_logic;
signal reset_keypad: std_logic;
signal reset_count: std_logic;
signal count_competed: std_logic;
signal an_dis: STD_LOGIC_VECTOR (3 downto 0);
signal cat_dis: STD_LOGIC_VECTOR (6 downto 0);
signal an_kp: STD_LOGIC_VECTOR (3 downto 0);
signal cat_kp: STD_LOGIC_VECTOR (6 downto 0);
signal an_count: STD_LOGIC_VECTOR (3 downto 0);
signal cat_count: STD_LOGIC_VECTOR (6 downto 0);
signal an_mux: STD_LOGIC_VECTOR (3 downto 0);
signal seg_mux: STD_LOGIC_VECTOR (6 downto 0);

begin
is_k_reset <= reset_keypad;

process(ssd_select, an_kp, cat_kp, an_dis, cat_dis, an_count, cat_count)
begin
    if ssd_select = "01" then
        an_mux <= an_kp;
        seg_mux <= cat_kp;
    elsif ssd_select = "10" then
        an_mux <= an_dis;
        seg_mux <= cat_dis;
    elsif ssd_select = "11" then
        an_mux <= an_count;
        seg_mux <= cat_count;
    else
        an_mux <= "1111";
        seg_mux <= "0000000";
    end if;
end process;

an <= an_mux;
seg <= seg_mux;
```

```
keypad_2 : entity work.PmodKYPD port map(
            clk     => clk,
            shift   => shift,
            sum     => sum(32 downto 1),
            reset   => resett,
            JA      => JA,
            an      => an_kp,
            seg     => cat_kp);

sum(0) <= enter;

summm <= sum(31 downto 16);
process(clk)
begin
   if rising_edge(clk) then
      if enable = '0' then
         reset_keypad <= '1';
      elsif enable = '1' then
         if sum = "10000001001010000010000101001000101" then --A7C9,enter
            reset_keypad <= '1';
         else
            reset_keypad <= '0';
         end if;
   end if;
   end if;
end process;

incorrect_password : entity work.fsm_top port map(clock => clk,
                                    reset =>reset_keypad,
                                    enter =>enter,
                                    count_over => count_competed,
                                    anode => an_count,
                                    cathode => cat_count,
                                    reset_alarm => reset
                                        );
```

```
sensor_module: entity work.motion_sensor port map(
                clk => clk,
                echo => echo,
                trigger => trigger,
                sensor => sensor,
                reset  => reset,
                Q      => Q,
                Q_not  =>  Q_not,
                an_d =>an_dis,
                cat_d =>cat_dis,
                is_between_10_and_15 => is_between_10_and_15);
is_locked <= not count_competed;
end Behavioral;
```

Code 3: keypad_2.vhd

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

entity PmodKYPD is
   Port (
      clk    : in STD_LOGIC;
      shift  : in std_logic;
      reset  :in std_logic;
      sum    : out std_logic_vector (31 downto 0);
      JA     : inout STD_LOGIC_VECTOR (7 downto 0);
      an     : out STD_LOGIC_VECTOR (3 downto 0);
      seg    : out STD_LOGIC_VECTOR (6 downto 0)
   );
end PmodKYPD;

architecture Behavioral of PmodKYPD is
   signal Decode, dispVal, dispVal0, dispVal1, dispVal2 : STD_LOGIC_VECTOR (7 downto 0);
   signal summ : std_logic_vector (31 downto 0);

begin
```

```
    C0: entity work.Decoder port map (
        clk       => clk,
        Row       => JA(7 downto 4),
        Col       => JA(3 downto 0),
        DecodeOut => Decode
    );

    C1: entity work.DisplayController port map (
        clk       => clk,
        shift     => shift,
        summ      => sum,
        reset     => reset,
        DispVal   => Decode,
        anode     => an,
        segOut    => seg
    );
end Behavioral;
```

Code 4: keypad.vhd

```
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(7 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
            if sclk = "00011000011010100000" then
                -- C1
                Col <= "0111";
                sclk <= sclk + 1;

            elsif sclk = "00011000011010101000" then
                -- R1
                if Row = "0111" then
                    DecodeOut <= "10001000"; -- 1
                -- R2
                elsif Row = "1011" then
                    DecodeOut <= "01001000"; -- 4
                -- R3
                elsif Row = "1101" then
                    DecodeOut <= "00101000"; -- 7
                -- R4
                elsif Row = "1110" then
                    DecodeOut <= "00011000"; -- *
                end if;
                sclk <= sclk + 1;

            elsif sclk = "00110000110101000000" then
                -- C2
                Col <= "1011";
                sclk <= sclk + 1;


            elsif sclk = "00110000110101001000" then
                -- R1
                if Row = "0111" then
                    DecodeOut <= "10000100"; -- 2
                -- R2
```

```
    elsif Row = "1011" then
        DecodeOut <= "01000100"; -- 5
    -- R3
    elsif Row = "1101" then
        DecodeOut <= "00100100"; -- 8
    -- R4
    elsif Row = "1110" then
        DecodeOut <= "00010100"; -- 0
    end if;
    sclk <= sclk + 1;

elsif sclk = "01001001001111100000" then
    -- C3
    Col <= "1101";
    sclk <= sclk + 1;


elsif sclk = "01001001001111101000" then
    -- R1
    if Row = "0111" then
        DecodeOut <= "10000010"; -- 3
    -- R2
    elsif Row = "1011" then
        DecodeOut <= "01000010"; -- 6
    -- R3
    elsif Row = "1101" then
        DecodeOut <= "00100010"; -- 9
    -- R4
    elsif Row = "1110" then
        DecodeOut <= "00010010"; -- #
    end if;
    sclk <= sclk + 1;

elsif sclk = "01100001101010000000" then
    -- C4
    Col <= "1110";
```

```
            sclk <= sclk + 1;

        elsif sclk = "01100001101010001000" then
            -- R1
            if Row = "0111" then
                DecodeOut <= "10000001"; -- A
            -- R2
            elsif Row = "1011" then
                DecodeOut <= "01000001"; -- B
            -- R3
            elsif Row = "1101" then
                DecodeOut <= "00100001"; -- C
            -- R4
            elsif Row = "1110" then
                DecodeOut <= "00010001"; -- D
            end if;
            sclk <= "00000000000000000000";

        else
            sclk <= sclk + 1;
        end if;
    end if;
  end process;
end Behavioral;
```

Code 5: seven_segment_display_keypad.vhd

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

entity DisplayController is
  Port (
    clk   :  in  std_logic;
    shift  :  in  std_logic;
    reset    : in std_logic;
    summ     : out std_logic_vector(31 downto 0);
```

```
        DispVal   : inout STD_LOGIC_VECTOR(7 downto 0);
        anode     : out std_logic_vector(3 downto 0);
        segOut    : out STD_LOGIC_VECTOR(6 downto 0)
    );
end DisplayController;

architecture Behavioral of DisplayController is
    signal digit_BCD        : std_logic_vector(7 downto 0);
    signal digit_counter    : std_logic_vector(1 downto 0);
    signal refresh_counter  : std_logic_vector(27 downto 0);
    signal dispVal0, dispVal1, dispVal2, dispVal3 : std_logic_vector(7 downto 0);
    signal clk190           : std_logic;

begin
process(clk)
begin
    if reset = '1' then
        refresh_counter <= (others => '0');
    elsif (rising_edge(clk)) then
        refresh_counter <= refresh_counter + 1;
    end if;
end process;

digit_counter <= refresh_counter(19 downto 18);

clkdiv: entity work.clkdiv port map (clk, '0', clk190);

process
begin
    if reset = '1' then
        dispVal0 <= "11111111"; -- Dash
        dispVal1 <= "11111111"; -- Dash
        dispVal2 <= "11111111"; -- Dash
    elsif rising_edge(clk190) and shift = '1' then
        dispVal0 <= DispVal;
        dispVal1 <= dispVal0;
```

```
        dispVal2 <= dispVal1;
    end if;
end process;

process(digit_counter, reset)
begin
    if reset = '1' then
        anode <= "1111"; -- Disable all digits
        digit_BCD <= "11111111"; -- Dash
    else
        case digit_counter is
            when "00" =>
                anode <= "1110";
                digit_BCD <= DispVal;
            when "01" =>
                anode <= "1101";
                digit_BCD <= dispVal0;
            when "10" =>
                anode <= "1011";
                digit_BCD <= dispVal1;
            when "11" =>
                anode <= "0111";
                digit_BCD <= dispVal2;
        end case;
    end if;
end process;

summ <= dispVal2 & dispVal1 & dispVal0 & DispVal;

with digit_BCD select
    segOut <= "1111111" when "11111111", -- Dash
              "1000000" when "00010100", -- 0
              "1111001" when "10001000", -- 1
              "0100100" when "10000100", -- 2
              "0110000" when "10000010", -- 3
              "0011001" when "01001000", -- 4
```

```
            "0010010" when "01000100", -- 5
            "0000010" when "01000010", -- 6
            "1111000" when "00101000", -- 7
            "0000000" when "00100100", -- 8
            "0010000" when "00100010", -- 9
            "0001000" when "10000001", -- A
            "0000011" when "01000001", -- B
            "1000110" when "00100001", -- C
            "0100001" when "00010001", -- D
            "0111111" when others;
end Behavioral;
```

Code 6: clock_divider.vhd

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

entity clkdiv is
    Port (
        clk     : in STD_LOGIC;
        clr     : in STD_LOGIC;
        clk190  : out STD_LOGIC
    );
end clkdiv;

architecture clkdiv of clkdiv is
    signal q : std_logic_vector(23 downto 0);
begin
    process(clk, clr)
    begin
        if clr = '1' then
            q <= x"000000";
        elsif clk'event and clk = '1' then
            q <= q + 1;
        end if;
    end process;
```

```
    clk190 <= q(23);

end clkdiv;
```

Code 7: fsm_counter_top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity fsm_top is
 Port (    clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           enter : in STD_LOGIC;
           count_over: out STD_LOGIC;
           reset_alarm      : out STD_LOGIC;
           anode : out STD_LOGIC_VECTOR(3 downto 0);
           cathode : out STD_LOGIC_VECTOR(6 downto 0)
            );
end fsm_top;

architecture Behavioral of fsm_top is
signal incorrect: STD_LOGIC_VECTOR(1 downto 0);
signal resett: STD_LOGIC;
signal fsm_rreset: STD_LOGIC;

begin
count_over <= resett;

fsm : entity work.FSM port map (clk =>clock,
                                reset_keypad=>reset,
                                reset_count=>resett ,
                                reset_alarm => reset_alarm,
                                incorrect_count=> incorrect,
                                fsm_reset => fsm_rreset,
                                enter => enter
                                );

ssd_count : entity work.ssd_counter port map (clock => clock,
                                    reset => fsm_rreset,
```

```
                    incorrect_count => incorrect,
                    count_over => resett,
                    anode => anode,
                    cathode => cathode
                    );
end Behavioral;
```

Code 8: fsm_incorrect_password.vhd

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

entity FSM is
    Port (
        clk           : in  STD_LOGIC;
        reset_keypad  : in  STD_LOGIC;
        reset_count   : in  STD_LOGIC;
        enter         : in  STD_LOGIC;
        reset_alarm      : out STD_LOGIC;
        incorrect_count : out STD_LOGIC_VECTOR(1 downto 0);
        fsm_reset     : out STD_LOGIC
    );
end FSM;

architecture Behavioral of FSM is
    type state_type is (state_0, state_1, state_2, state_2_hold, wait_lock);
    signal current_state, next_state : state_type;
    signal enter_prev : STD_LOGIC := '0';
begin

    process(clk)
    begin
        if rising_edge(clk) then
            current_state <= next_state;
            enter_prev <= enter;
        end if;
    end process;
```

```
process(current_state, reset_keypad, reset_count, enter, enter_prev)
begin
    -- default
    fsm_reset <= '1';
    incorrect_count <= "00";

    case current_state is
        when state_0 =>
            if enter = '1' and enter_prev = '0' and reset_keypad = '0' then
                next_state <= state_1;
                reset_alarm <= '0';
            else
                next_state <= state_0;
                reset_alarm <= '0';
            end if;

        when state_1 =>
            incorrect_count <= "01";
            if enter = '1' and enter_prev = '0' and reset_keypad = '0' then
                next_state <= state_2;
                reset_alarm <= '0';
            elsif reset_keypad = '1' then
                next_state <= state_0;
                reset_alarm <= '1';
            else
                next_state <= state_1;
                reset_alarm <= '0';
            end if;

        when state_2 =>
            incorrect_count <= "10";
            if enter = '1' and enter_prev = '0' and reset_keypad = '0' then
                next_state <= state_2_hold;
                reset_alarm <= '0';
            elsif reset_keypad = '1' then
                next_state <= state_0;
```

```
                reset_alarm <= '1';
            else
                next_state <= state_2;
                reset_alarm <= '0';
            end if;

        when state_2_hold =>   -- this state is added for self-correction (there is a timing issue between enter and reset_keypad)
            incorrect_count <= "10";
            if reset_keypad = '1' then
                next_state <= state_0;
                reset_alarm <= '1';
            else
                next_state <= wait_lock;
                reset_alarm <= '0';
            end if;

        when wait_lock =>
            incorrect_count <= "11";
            fsm_reset <= '0';
            if reset_count = '1' then
                next_state <= state_0;
                reset_alarm <= '0';
            else
                next_state <= wait_lock;
                reset_alarm <= '0';
            end if;

        when others =>
            next_state <= state_0;
           -- reset_alarm <= '1';
      end case;
   end process;
end Behavioral;
```

Code 9: ssd_counter.vhd

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity ssd_counter is
    Port ( clock : in STD_LOGIC;
           reset : in STD_LOGIC;
           incorrect_count: in STD_LOGIC_VECTOR(1 downto 0);
           count_over: out STD_LOGIC;
           anode : out STD_LOGIC_VECTOR(3 downto 0);
           cathode : out STD_LOGIC_VECTOR(6 downto 0));
end ssd_counter;

architecture Behavioral of ssd_counter is

signal phase: STD_LOGIC_VECTOR(1 downto 0);
signal sec: STD_LOGIC;
signal num: STD_LOGIC_VECTOR(15 downto 0);

begin

clk: entity work.clock port map(clock => clock, reset => reset, phase => phase, sec => sec, second => num, count_over => count_over);
four_to_one_mux: entity work.mux port map(clock => clock, reset => reset, phase => phase, num => num, sec => sec, anode => anode, cathode => cathode, incorrect_count => incorrect_count);

end Behavioral;
```

Code 10: counter_10sec.vhd

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

entity clock is
    Port (
        clock      : in  STD_LOGIC;
        reset      : in  STD_LOGIC;
        count_over : out STD_LOGIC;
```

```
        counter   : out STD_LOGIC_VECTOR(27 downto 0);
        phase     : out STD_LOGIC_VECTOR(1 downto 0);
        second    : out STD_LOGIC_VECTOR(15 downto 0);
        sec       : out STD_LOGIC
    );
end clock;

architecture Behavioral of clock is
    signal count: STD_LOGIC_VECTOR(27 downto 0) := (others => '0');
    signal scnd: STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
    signal enable_count: STD_LOGIC := '1';
    signal count_done: STD_LOGIC := '0';
begin
    process(clock, reset)
    begin
        if reset = '1' then
            count <= (others => '0');
            scnd <= (others => '0');
            enable_count <= '1';
            count_done <= '0';
        elsif rising_edge(clock) then
            if enable_count = '1' then
                count <= count + 1;
                if count = x"5F5E0FF" then
                    count <= (others => '0');
                    scnd <= scnd + 1;
                    if scnd = "0000000000001001" then
                        enable_count <= '0';
                        count_done <= '1';
                    end if;
                end if;
            end if;
        end if;
    end process;

    count_over <= '1' when reset = '1' else count_done;
```

```
    sec <= '1' when count = x"5F5E0FF" else '0';
    phase <= count(19 downto 18);
    counter <= count;
    second <= scnd;
end Behavioral;
```

Code 11: an_mux.vhd

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

entity mux is
    Port (
        clock : in STD_LOGIC;
        reset : in STD_LOGIC;
        phase : in STD_LOGIC_VECTOR(1 downto 0);
        num : in STD_LOGIC_VECTOR(15 downto 0);
        sec : in STD_LOGIC;
        incorrect_count: in STD_LOGIC_VECTOR(1 downto 0);
        anode : out STD_LOGIC_VECTOR(3 downto 0);
        cathode : out STD_LOGIC_VECTOR(6 downto 0)
    );
end mux;

architecture Behavioral of mux is
    signal leds: STD_LOGIC_VECTOR(3 downto 0);
    signal blank: STD_LOGIC;
begin
    process(phase)
    begin
        case phase is
            when "00" =>
                anode <= "0111"; -- 1st digit
                leds <= "00" & incorrect_count;
                blank <= '0';
            when "01" =>
```

```
            anode <= "1011"; -- 2nd digit
            leds <= num(11 downto 8);
            blank <= '1';
        when "10" =>
            anode <= "1101"; -- 3rd digit
            leds <= num(7 downto 4);
            blank <= '0';
        when "11" =>
            anode <= "1110"; -- 4th digit
            leds <= num(3 downto 0);
            blank <= '0';
        when others =>
            anode <= "1111";
    end case;
  end process;

led: entity work.cdecoder port map(leds => leds, cathode => cathode, blank=>blank);

end Behavioral;
```

Code 12: cat_decoder.vhd

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

entity cdecoder is
  Port (
    leds : in STD_LOGIC_VECTOR(3 downto 0);
    blank : in STD_LOGIC;
    cathode : out STD_LOGIC_VECTOR(6 downto 0)
  );
end cdecoder;

architecture Behavioral of cdecoder is
begin
  process(leds)
  begin
    case leds & blank is
```

```
            when "00000" => cathode <= "1000000"; -- 0
            when "00010" => cathode <= "1111001"; -- 1
            when "00100" => cathode <= "0100100"; -- 2
            when "00110" => cathode <= "0110000"; -- 3
            when "01000" => cathode <= "0011001"; -- 4
            when "01010" => cathode <= "0010010"; -- 5
            when "01100" => cathode <= "0000010"; -- 6
            when "01110" => cathode <= "1111000"; -- 7
            when "10000" => cathode <= "0000000"; -- 8
            when "10010" => cathode <= "0010000"; -- 9
            when "10100" => cathode <= "0001000"; -- A
            when "10110" => cathode <= "0000011"; -- b
            when "11000" => cathode <= "1000110"; -- C
            when "11010" => cathode <= "0100001"; -- d
            when "11100" => cathode <= "0000110"; -- E
            when "11110" => cathode <= "0001110"; -- F
            when others => cathode <= "1111111"; -- Blank
        end case;
    end process;
end Behavioral;
```

Code 13: sensors.vhd

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


entity motion_sensor is
    Port (
        clk : in  STD_LOGIC;
        echo : in  STD_LOGIC;
        trigger : out  STD_LOGIC;
        sensor : in STD_LOGIC;
        reset  : in STD_LOGIC;
        Q_not  : out STD_LOGIC;
        Q      : out STD_LOGIC;
        an_d : OUT std_logic_vector(3 downto 0);
        cat_d : out  STD_LOGIC_VECTOR(6 downto 0);
        is_between_10_and_15 : out STD_LOGIC
```

```
    );
end motion_sensor;

architecture Behavioral of motion_sensor is

    signal Q_internal : STD_LOGIC := '0';
    signal is_less : STD_LOGIC;
    signal current_state, next_state : STD_LOGIC;

begin

distance_checker : entity work.sistema_maxsonar port map(
    clk => clk,
    echo => echo,
    trigger => trigger,
    an => an_d,
    cat => cat_d,
    is_less_than_10 => is_less,
    is_between_10_and_15 => is_between_10_and_15
);

process (clk, reset)
begin
    if reset = '1' then
        current_state <= '0';
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;

process (current_state, sensor, is_less, reset)
begin

    next_state <= current_state;
    Q_internal <= '0';
```

```
    case current_state is
        when '0' =>
            if sensor = '1' and is_less = '1' then
                next_state <= '1';
            end if;

        when '1' =>
            Q_internal <= '1';
            if reset = '1' then
                next_state <= '0';
            end if;
    end case;
end process;

Q <= Q_internal;
Q_not <= not Q_internal;

end Behavioral;
```

Code 14: distance.vhd

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

entity sistema_maxsonar is
    Port ( clk : in  STD_LOGIC;
           echo : in  STD_LOGIC;
           trigger : out std_logic;
           an : out std_logic_vector(3 downto 0);
           cat : out  STD_LOGIC_VECTOR(6 downto 0);
           is_less_than_10 : out std_logic;
           is_between_10_and_15: out std_logic
           );
end sistema_maxsonar;
```

```
architecture Behavioral of sistema_maxsonar is
  COMPONENT control_maxsonar
   PORT(
      clk : IN std_logic;
      echo : IN std_logic;
      trigger : OUT std_logic;
      distance : INOUT std_logic_vector(10 downto 0);
      is_less_than_10 : out STD_LOGIC;
      is_between_10_and_15: out STD_LOGIC
      );
   END COMPONENT;

  COMPONENT bin2bcd
   PORT(
      clk : IN std_logic;
      b_in : IN std_logic_vector(12 downto 0);
      dec_1 : OUT std_logic_vector(3 downto 0);
      dec_10 : OUT std_logic_vector(3 downto 0);
      dec_100 : OUT std_logic_vector(3 downto 0);
      dec_1000 : OUT std_logic_vector(3 downto 0)
      );
   END COMPONENT;

  COMPONENT visu7seg
   PORT(
      clk : IN std_logic;
      dec_1 : IN std_logic_vector(3 downto 0);
      dec_10 : IN std_logic_vector(3 downto 0);
      dec_100 : IN std_logic_vector(3 downto 0);
      dec_1000 : IN std_logic_vector(3 downto 0);
      an : OUT std_logic_vector(3 downto 0);
      cat : out  STD_LOGIC_VECTOR(6 downto 0)
      );
   END COMPONENT;
```

```
    signal dec_1, dec_10, dec_100, dec_1000 : std_logic_vector(3 downto 0);
    signal distance : std_logic_vector(10 downto 0);
    signal distance_2 : std_logic_vector(12 downto 0);
    signal prev_distance : std_logic_vector(10 downto 0) := (others => '0');

begin

    Inst_control_maxsonar: control_maxsonar PORT MAP(
        clk => clk,
        echo => echo,
        trigger => trigger,
        distance => distance,
        is_less_than_10 => is_less_than_10,
        is_between_10_and_15 => is_between_10_and_15
    );

    Inst_bin2bcd: bin2bcd PORT MAP(
        clk => clk,
        b_in => distance_2,
        dec_1 => dec_1,
        dec_10 => dec_10,
        dec_100 => dec_100,
        dec_1000 => dec_1000
    );

    Inst_visu7seg: visu7seg PORT MAP(
        clk => clk,
        dec_1 => dec_1,
        dec_10 => dec_10,
        dec_100 => dec_100,
        dec_1000 => dec_1000,
        an => an,
        cat => cat
    );

    process(clk)
```

```
  begin
    if rising_edge(clk) then
      if distance /= prev_distance then
        distance_2 <= "00" & distance;
        prev_distance <= distance;
      end if;
    end if;
  end process;
end Behavioral;
```

Code 15: ultrasonic_sensor_control.vhd

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

entity control_maxsonar is
  Port (
    clk : in  STD_LOGIC;
    echo : in  STD_LOGIC;
    trigger : out  STD_LOGIC;
    distance : inout  STD_LOGIC_VECTOR (10 downto 0);
    is_less_than_10 : out STD_LOGIC;
    is_between_10_and_15: out STD_LOGIC
  );
end control_maxsonar;

architecture Behavioral of control_maxsonar is
  signal count : unsigned(10 downto 0) := (others => '0');
  signal clk_2 : std_logic;
  signal echo_2 : unsigned(10 downto 0) := (others => '0');
  signal measuring : std_logic := '0';

begin

  p_clk: process(clk)
  begin
    if rising_edge(clk) then
```

```
        if count < 1450 then
            count <= count + 1;
        else
            count <= (others => '0');
        end if;
    end if;
end process;
clk_2 <= count(10);

process(clk_2)
begin
    if rising_edge(clk_2) then

        if measuring = '0' then
            trigger <= '1';
            echo_2 <= (others => '0');
            measuring <= '1';
        else
            trigger <= '0';
            if echo = '1' then
                if echo_2 < 1600 then
                    echo_2 <= echo_2 + 1;
                end if;
            else
                if echo_2 > 0 then
                    distance <= std_logic_vector(echo_2 / 4);
                end if;
                measuring <= '0';
            end if;
        end if;
    end if;
end process;
process(clk)
begin
    if rising_edge(clk) then
        if unsigned(distance) < 10 or unsigned(distance) = 10 then
```

```
            is_less_than_10 <= '1';
        else
            is_less_than_10 <= '0';
        end if;
    end if;
  end process;
   process(clk)
  begin
    if rising_edge(clk) then
        if unsigned(distance) > 10 and unsigned(distance) <= 15 then
            is_between_10_and_15 <= '1';
        else
            is_between_10_and_15 <= '0';
        end if;
    end if;
  end process;
end Behavioral;
```

Code 16: distance_ssd.vhd

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

entity visu7seg is
   Port ( clk : in  STD_LOGIC;
          dec_1 : in  STD_LOGIC_VECTOR (3 downto 0);
          dec_10 : in  STD_LOGIC_VECTOR (3 downto 0);
          dec_100 : in  STD_LOGIC_VECTOR (3 downto 0);
          dec_1000 : in  STD_LOGIC_VECTOR (3 downto 0);
          an : out  STD_LOGIC_VECTOR (3 downto 0);
          cat : out  STD_LOGIC_VECTOR(6 downto 0));
end visu7seg;

architecture Behavioral of visu7seg is
  signal count : unsigned (18 downto 0);
  signal seg_disp : std_logic_vector (6 downto 0);
  signal selector : std_logic_vector (3 downto 0);
```

```
begin
  P1_count:process(clk)
  begin
    if rising_edge(clk) then
      count <= count + 1;
    end if;
  end process;

  with count(18 downto 17) select
   an <= "1110" when "00",  -- 0
        "1101" when "01",  -- 1
        "1011" when "10",  -- 2
        "0111" when "11",  -- 3
        "1111" when others;  -- turned off
  with count(18 downto 17) select
  selector <=  dec_1 when "00",
        dec_10 when "01",
        dec_100 when "10",
        dec_1000 when others;
   with selector select
   seg_disp <= "1000000" when "0000",  -- 0
             "1111001" when "0001",  -- 1
             "0100100" when "0010",  -- 2
             "0110000" when "0011",  -- 3
             "0011001" when "0100",  -- 4
             "0010010" when "0101",  -- 5
             "0000010" when "0110",  -- 6
             "1111000" when "0111",  -- 7
             "0000000" when "1000",  -- 8
             "0010000" when "1001",  -- 9
             "1111111" when others;  -- turned off
 cat <= seg_disp;
end Behavioral;
```

Code 17: bin_to_dec.vhd

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity bin2bcd is
    Port ( clk : in  STD_LOGIC;
           b_in : in  STD_LOGIC_VECTOR (12 downto 0);
           dec_1 : out  STD_LOGIC_VECTOR (3 downto 0);
           dec_10 : out  STD_LOGIC_VECTOR (3 downto 0);
           dec_100 : out  STD_LOGIC_VECTOR (3 downto 0);
           dec_1000 : out  STD_LOGIC_VECTOR (3 downto 0));
end bin2bcd;
architecture Behavioral of bin2bcd is
  signal dec_1_i, dec_10_i, dec_100_i, dec_1000_i: unsigned(3 downto 0);
  signal b_in_reg: std_logic_vector(12 downto 0);
  signal bcd4: unsigned(15 downto 0);
  type state_type is (load, replace, compr, ready);
  signal comp_u, comp_d, comp_c, comp_m: std_logic;
  signal sum: std_logic;
begin
  P1:process(clk)
     variable i: integer range 0 to b_in'high;
     variable state: state_type;
  begin
     if rising_edge(clk) then
        case state is
           when load =>
              b_in_reg <= b_in;
              bcd4 <= (others => '0');
              i := b_in'high;
              state := replace;
           when replace =>
              bcd4(15 downto 1) <= bcd4(14 downto 0);
              bcd4(0) <= std_logic(b_in_reg(12));
              b_in_reg <= b_in_reg(11 downto 0) & '0';
              if i > 0 then
                 i := i - 1;
                 state := compr;
```

```
      else
        state := ready;
      end if;
    when compr =>
      if (sum = '1') then
        if (comp_u = '1') then
          bcd4( 3 downto  0) <= bcd4( 3 downto  0) + 3;
        end if;
        if (comp_d = '1') then
          bcd4( 7 downto  4) <= bcd4( 7 downto  4) + 3;
        end if;
        if (comp_c = '1') then
          bcd4(11 downto  8) <= bcd4(11 downto  8) + 3;
        end if;
        if (comp_m = '1') then
          bcd4 (15 downto 12) <= bcd4 (15 downto 12) + 3;
        end if;
        state := replace;
      else
        bcd4(15 downto 1) <= bcd4(14 downto 0);
        bcd4(0) <= std_logic(b_in_reg(12));
        b_in_reg <= b_in_reg(11 downto 0) & '0';
        if i > 0 then
          i := i - 1;
          state := compr;
        else
          state := ready;
        end if;
      end if;
    when ready =>
      dec_1  <= STD_LOGIC _VECTOR(bcd4( 3 downto  0));
      dec_10 <= STD_LOGIC_VECTOR(bcd4( 7 downto  4));
      dec_100 <= STD_LOGIC_VECTOR(bcd4(11 downto  8));
      dec_1000 <= STD_LOGIC _VECTOR(bcd4(15 downto 12));
      state := load;
  end case;
```

```
    end if;
  end process;
  comp_u <= '1' when bcd4( 3 downto  0) > 4 else '0';
  comp_d <= '1' when bcd4( 7 downto  4) > 4 else '0';
  comp_c <= '1' when bcd4(11 downto  8) > 4 else '0';
  comp_m <= '1' when bcd4(15 downto 12) > 4 else '0';
  sum <= comp_u OR comp_d OR comp_c OR comp_m;
end Behavioral;
```