DOOR SECURITY SYSTEM

Bilkent University

Department of Electrical and Electronics Engineering

EE102 Term Project

Alp Tekin

22403597

Section 2

Video Presentation

https://www.youtube.com/watch?v=GS6-5niaZwo

Purpose

The purpose of the term project is to implement everything that we have learned in EE102 course using VHDL, Basys3 FPGA and external components. In this project, I aimed to design a door security system, using combinational logic, registers, a finite state machine (FSM) implemented in VHDL. From the outside, the door is protected by a 4-digit PIN code and alarm system is triggered when the password is entered incorrectly. Additionally, door can be opened from the inside when a person approaches the door.

Methodology

For the door security system, door can be opened in two ways:

First, from outside, the user should enter the correct password in order to open the door. For inputs a hand detection algorithm is used in Python. I used both OpenCV, MediaPipe and PySerial to implement the logic in Python code. The inputs are provided using the right hand, where each digit is represented by the number of extended fingers. When the hand remains stable and displays the same digit for two seconds, the system transfers the corresponding digit to the VHDL. This part is only used for inputs and all the main logic and implementations are made using VHDL. For the VHDL part an FSM logic is implemented to receive the data from Python. UART communication receives each digit and the system behaves as following:

- The received digit is displayed on the seven-segment display of the Basys3. Starting from the leftmost digit, each received input is shifted to the next display position. It allows the entered digits to appear sequentially from left to right.
- Once all four digits are entered, the system compares the input with predefined password.
    - If the entered password is correct, two SG90 Servo motors rotate to open the door and green LED indicates that the door is unlocked.
    - If the entered password is incorrect, servo motors remain stationary, a buzzer is activated to trigger an alarm and red LED blinks with same frequency as the buzzer.

In second state, corresponds to the exit operation from inside, the HC-SR04 ultrasonic distance sensor continuously monitors the proximity of a person to the door. The HC-SR04 ultrasonic distance sensor measures the distance to an object by using ultrasonic sound waves and the speed of sound. Thus, when the user approaches the door within a predefined distance threshold (20 cm), the sensor is triggered regardless of the current external state of the system. Once the signal is received from the sensor, it triggers the door to unlock and the green LED is activated to indicate that the door is opened. A reset button is used to reset the seven-segment display and return the entire system to its initial state.

Design Specifications

To implement the logic in VHDL, a modular structure was used. The whole system includes 1 top_module and 7 submodules for main controller, uart communication, seven-segment display, seven-segment decoder, servo driver, proximity sensor, alarm driver. The structure of the design can be seen in Figure 1.

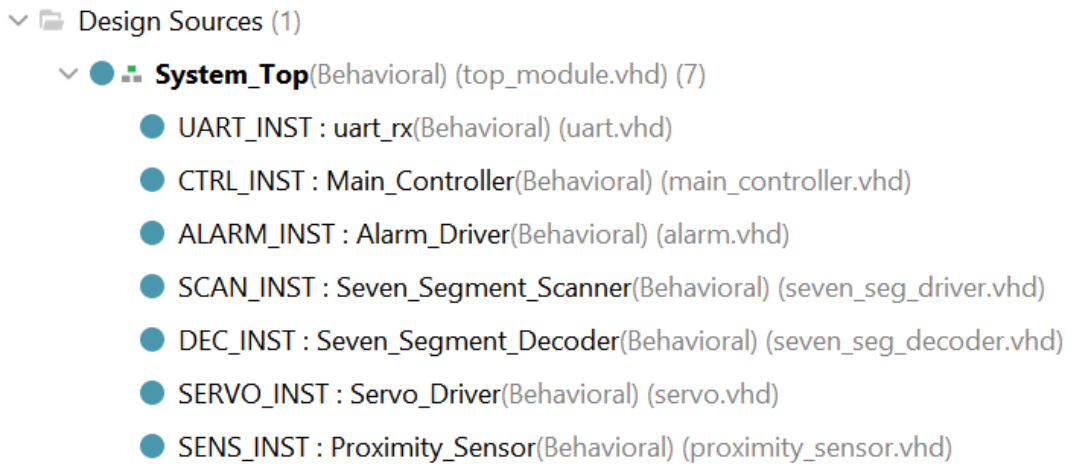

Figure 1: General Structure of the Design

In the top_module there are four input and eight outputs as following:

- clk : 100MHz internal clock signal of Basys3 (input)
- Rx_in : Received data input from UART module (input)
- btnC : Reset button to reset the whole system (input)
- seg : Cathodes for individual seven-segment display (output)
- an : Anodes to determine which digit is selected in seven-segment display (output)
- ja1_servo : For the first SG90 Servo Motor (output)
- ja2_servo : For the second SG90 Servo Motor (output)
- ja3_trig : Trigger pin for HC-SR04 ultrasonic sensor (output)

- ja4_echo : Received data echo pin for HC-SR04 ultrasonic sensor (input)
- jb1_buzzer : The alarm signal for the buzzer (output)
- jb2_led_green : Green LED to indicate that door is unlocked (output)
- jb3_led_red : Red LED to indicate that the entered password is incorrect (output)

The top_module ensures that the proper port map connections are made for each submodule.

The most significant part in this design is “main_controller.vhd” submodule since it controls how the system behaves based on the received inputs and sensor signals. At first, the code checks the reset (rst) input. When the reset signal is active, system resets the seven-segment display, stop the servo motors and turns off other components. If reset signal is not active, system checks the signal from HC-SR04 distance sensor input and confirms that the signal stays high for 40 ms to prevent noise related false detections. Furthermore, when the HC-SR04 distance sensor input is inactive, the system checks the data coming from the UART module and compares each received digit with the corresponding password digit. The predefined password is stored internally as constant values in this submodule. The door and buzzer states are stored using the door_latch and alarm_latch signals. If the door is unlocked either by entering the correct password or by a signal from the distance sensor, the door_latch signal is set to logic high. If an incorrect password is entered, the alarm_latch signal is set to logic high to activate the alarm. In Figure 2, RTL Schematic for main_controller submodule is shown.

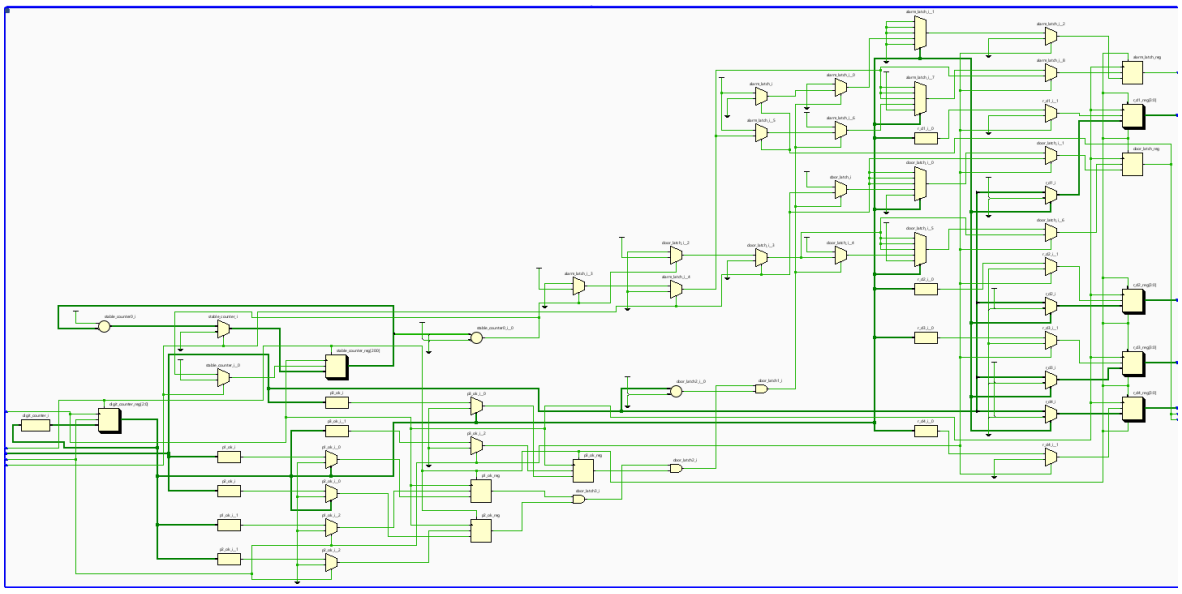

Figure 2: RTL Schematic for “main_controller.vhd” submodule

A Finite State Machine (FSM) is implemented in the “UART.vhd” module. The UART module receives serial data through the micro-USB port from a Hand-Detection Python Algorithm. This module includes two inputs (clk, rx_pin_in) and two outputs (rx_data_out, rx_done_tick). FSM has 5 states, IDLE, START, DATA, STOP, DONE. In IDLE state, the receiver looks for a logic “0” input which indicates the start bit. Since the UART remains at logic “1” when idle, once logic “0” is received the FSM transitions to START state. In the START state, the FSM verifies the validity of the start bit by sampling the RX line at the midpoint of the bit period. The midpoint of a bit period is calculated using the internal 100 MHz clock and the baud rate of 9600. If the start bit is valid so the system moves on to the DATA state; otherwise, the system turns back into IDLE state. In Data state, the code reads 8-bit input bit by bit, which corresponds to the entered digit. This 8-bit data is stored in a register. During the STOP state, the FSM counts one bit duration (10416 clock cycles) to allow the stop bit to be received, and then transitions to the DONE state. The DONE state signals that a complete byte has been successfully received and makes the data available to the system. After DONE state, FSM returns back to IDLE state and the received output rx_data_out is made available to other submodules. A RTL Schematic for UART module can be seen in Figure 3.

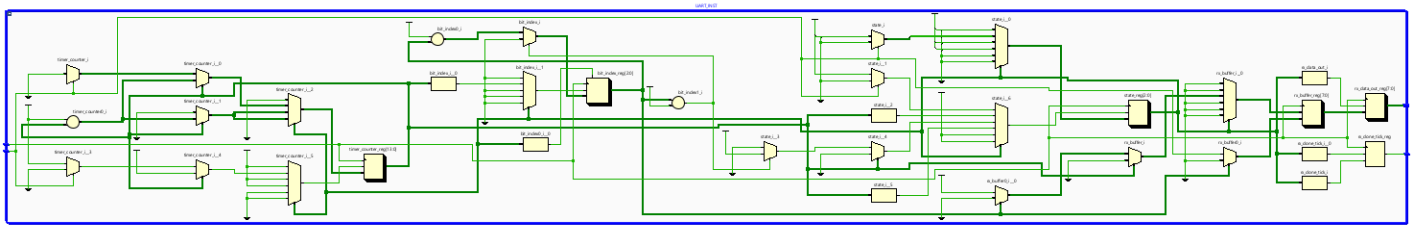

Figure 3: RTL Schematic for “UART.vhd” submodule

The “seven_segment_scanner.vhd” module is used to select which digit of the seven-segment display will be active at a given time. The module uses a counter to switch between the four digits very quickly. At each moment, only one digit is turned on, but since the switching is fast enough, all digits appear to be on at the same time. This module takes 6 inputs (clk, rst, in_d1, in_d2, in_d3, in_d4) and 2 outputs (o_mux_data, o_anode). Each input digit corresponds to a fixed position on the display. This allows that the entered digits to be shown from left to right. After this process is completed, the o_mux_data is transmitted to seven_segment_decoder submodule to generate the corresponding segment pattern. The RTL schematic of the submodule is shown in Figure 4.

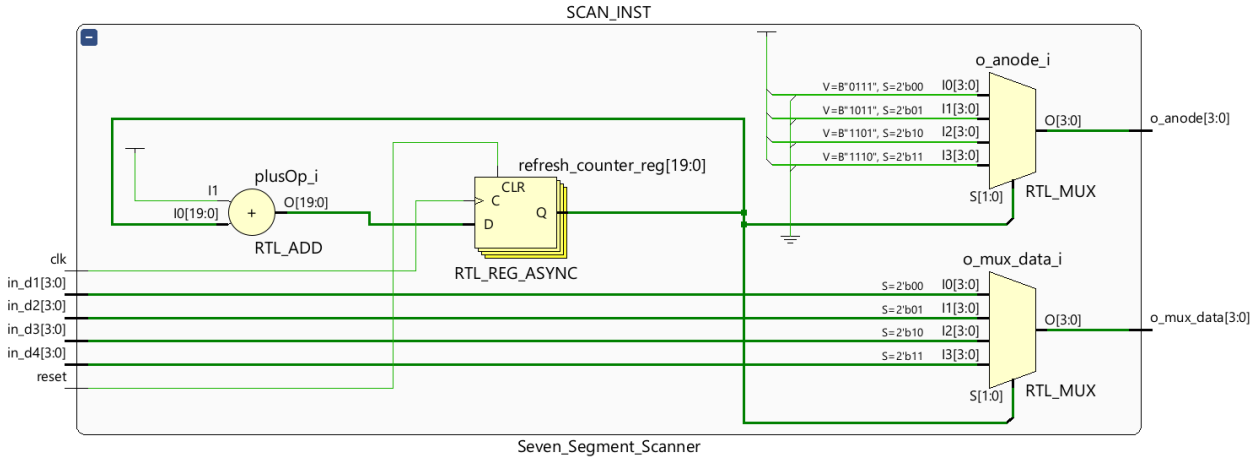

Figure 4: RTL Schematic for "seven_segment_scanner.vhd" submodule

The 4-bit output from "seven_segment_scanner.vhd" module is forwarded to "seven_segment_decoder.vhd" submodule. Seven-segment decoder takes one input i_hex and one output o_seg. In this module, the received 4-bit data is converted into the corresponding seven-segment pattern using a decoder. For digits that have not yet been entered, the seven-segment display is programmed to show a dash "—". In Figure 5, RTL Schematic for decoder submodule is shown.

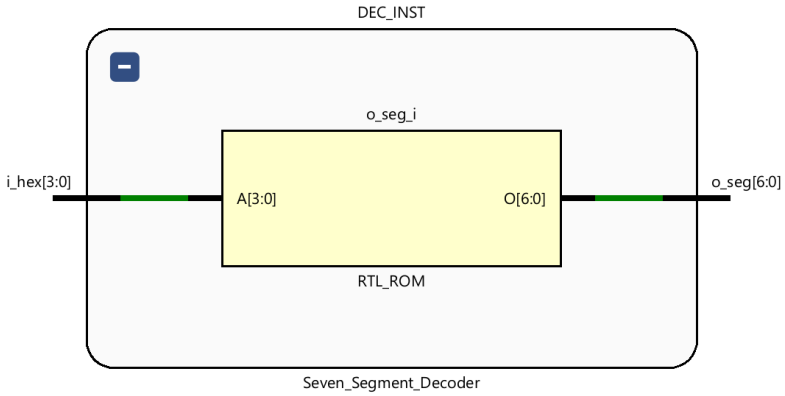

Figure 5: RTL Schematic for "seven_segment_decoder.vhd" submodule

The proximity_sensor.vhd module is designed to detect the presence of an object using an HC-SR04 ultrasonic sensor. This module takes two inputs (clk, echo) and gives two outputs (trig, obj_det) The sensor operates using two main signals: TRIG and ECHO. The module sends a trigger signal to the TRIG pin of the HC-SR04 sensor. After the trigger signal is sent, the sensor returns an ECHO signal. The duration of the ECHO signal changes depending on how far the object is. If the measured ECHO duration is below a predefined threshold corresponding to approximately 20 cm, the module sets the obj_det signal to indicate that an object is detected. RTL schematic for this submodule is illustrated in Figure 6.

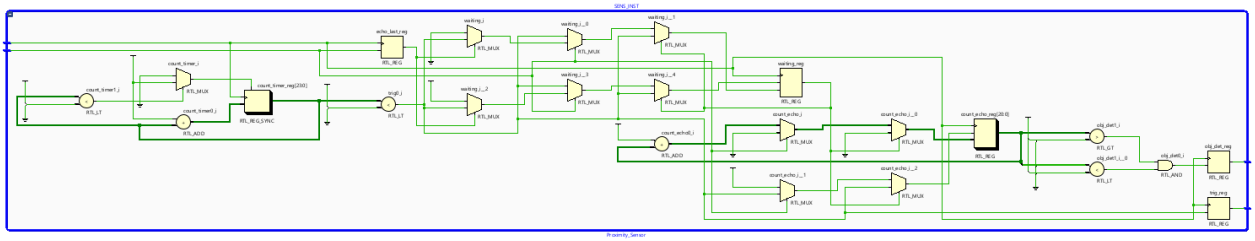

Figure 6: RTL Schematic for “proximity_sensor.vhd” submodule

The servo.vhd module is written to open the door when the correct password is entered or when the proximity sensor is triggered. Although this system includes two servo motors, only one servo sub module is used since both motors are activated using the same control signal. This module has two inputs (clk, servo_en) and one output (servo_pwm). The servo motor is controlled using a PWM signal with a period of 20 ms. The servo operates with a 180 degree rotation range. When a 0.5 ms signal is applied, the servo motors remain in the closed position. When a 2.5 ms signal is applied, the servo motors rotate fully and the door is opened. While the door remains open, the servo motors continuously receive the 2.5 ms PWM signal to maintain the open position. Similarly, when the door is closed, the Basys3 FPGA continuously sends a 0.5 ms PWM signal to keep the servo motors in the closed position. The RTL schematic for servo.vhd module can be seen in Figure 7.

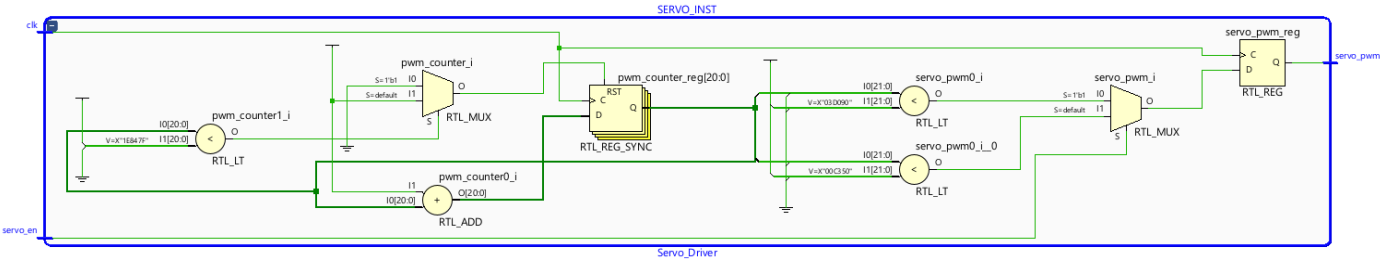

Figure 7: RTL Schematic for "servo.vhd" submodule

The alarm.vhd submodule is designed to generate visual and audible warnings when the password is entered incorrectly. The module has two inputs (clk, enable) and two outputs (led_out, buzzer_out). When the enable signal is active, the red LED blinks and the buzzer generates a sound. The blinking LED is generated using a counter, which toggles the LED state at a fixed interval. Similarly, the buzzer output is produced using another counter to generate a square-wave signal. The buzzer is activated only when the alarm is enabled and the LED is in the active state. Both the blinking LED and the buzzer operate at a frequency of 2 Hz during this period. In addition, the buzzer tone is generated using a separate signal while the buzzer is active. The frequency of this signal determines whether the buzzer sound is low-pitched or high-pitched. In Figure 8, the RTL schematic for this submodule is shown.

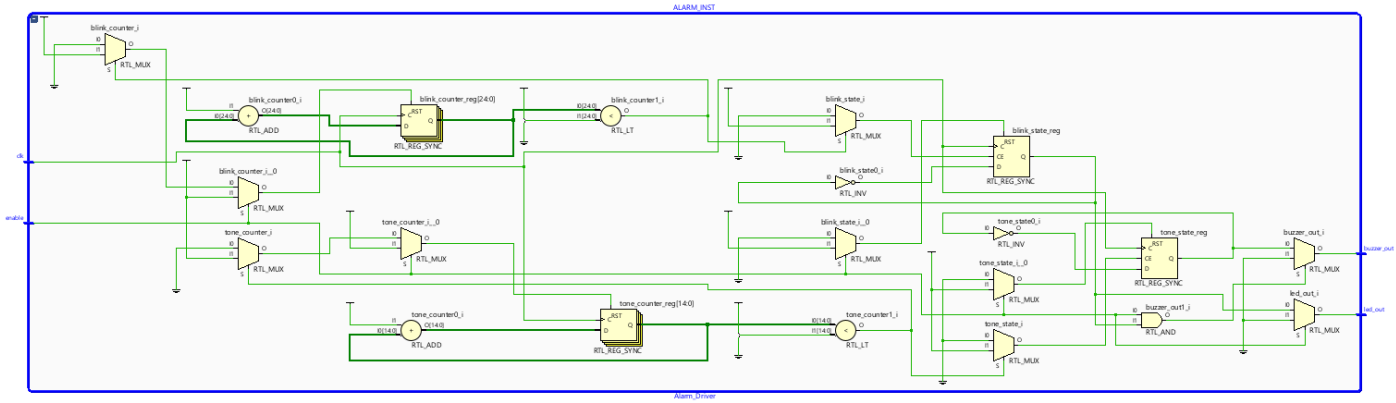

Figure 8: RTL Schematic for "alarm.vhd" submodule

For the hardware implementation, since the buzzer operates with 5 V, BC547 transistor was used to ensure that the buzzer receives the signal from Basys3. This configuration allows the control signal to be provided by the Basys3 while the buzzer is powered by an external 5 V power supply. Also, for the HC-SR04 ultrasonic distance sensor, a voltage divider is used to scale down the output signal and ensure that a 3.3 V is transferred to the FPGA. The hardware implementation of the system can be seen in the following Figure 9-11.

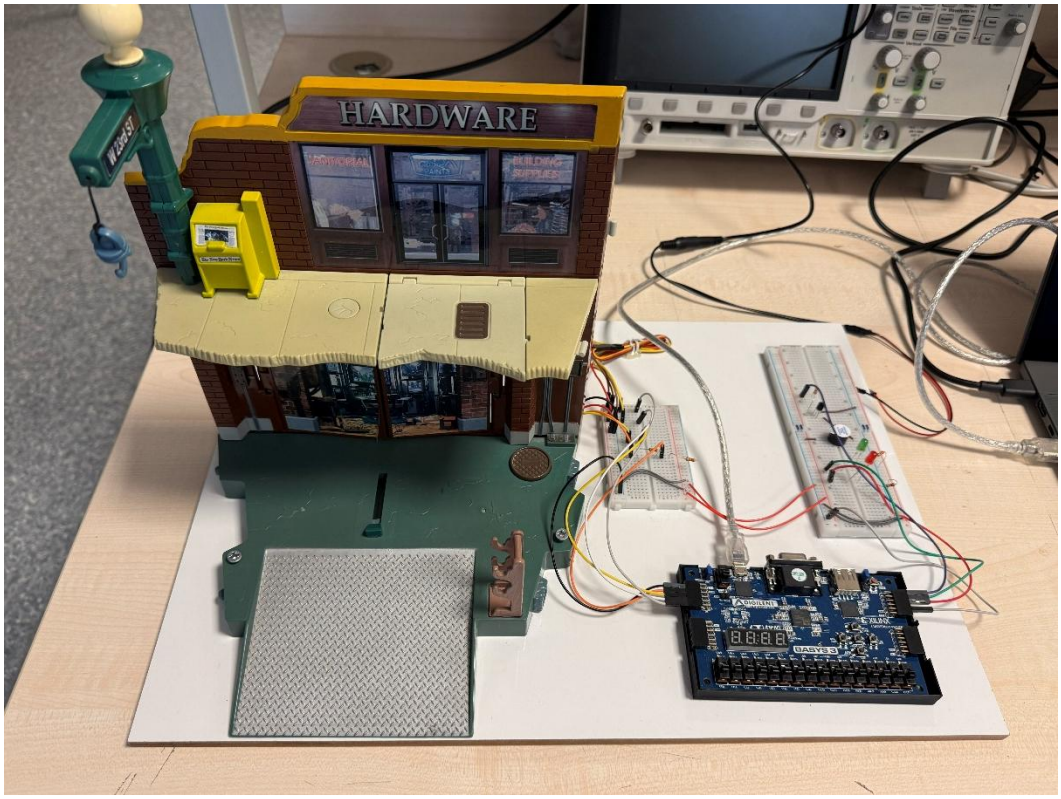

Figure 9: Physical setup of the door security system (front view)

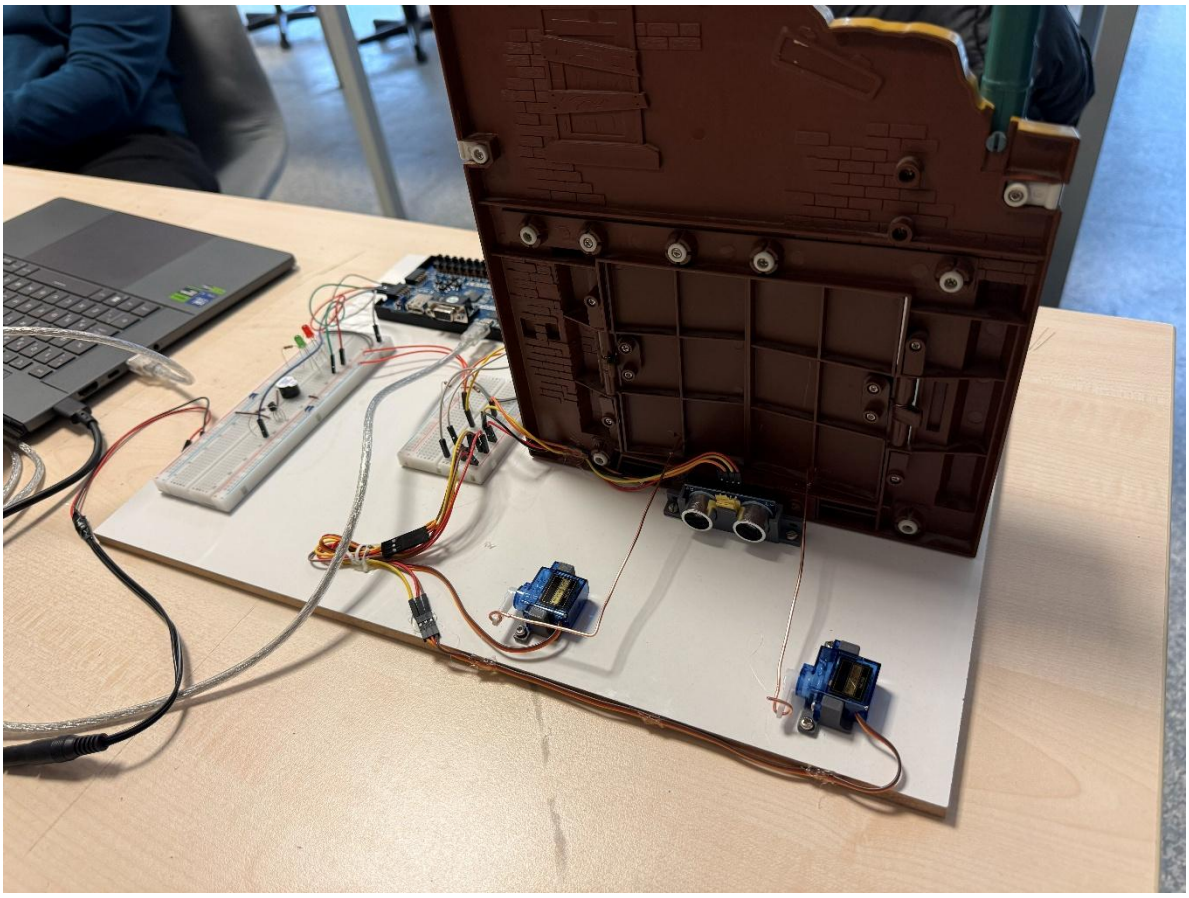

Figure 10: Physical setup of the door security system (rear view)

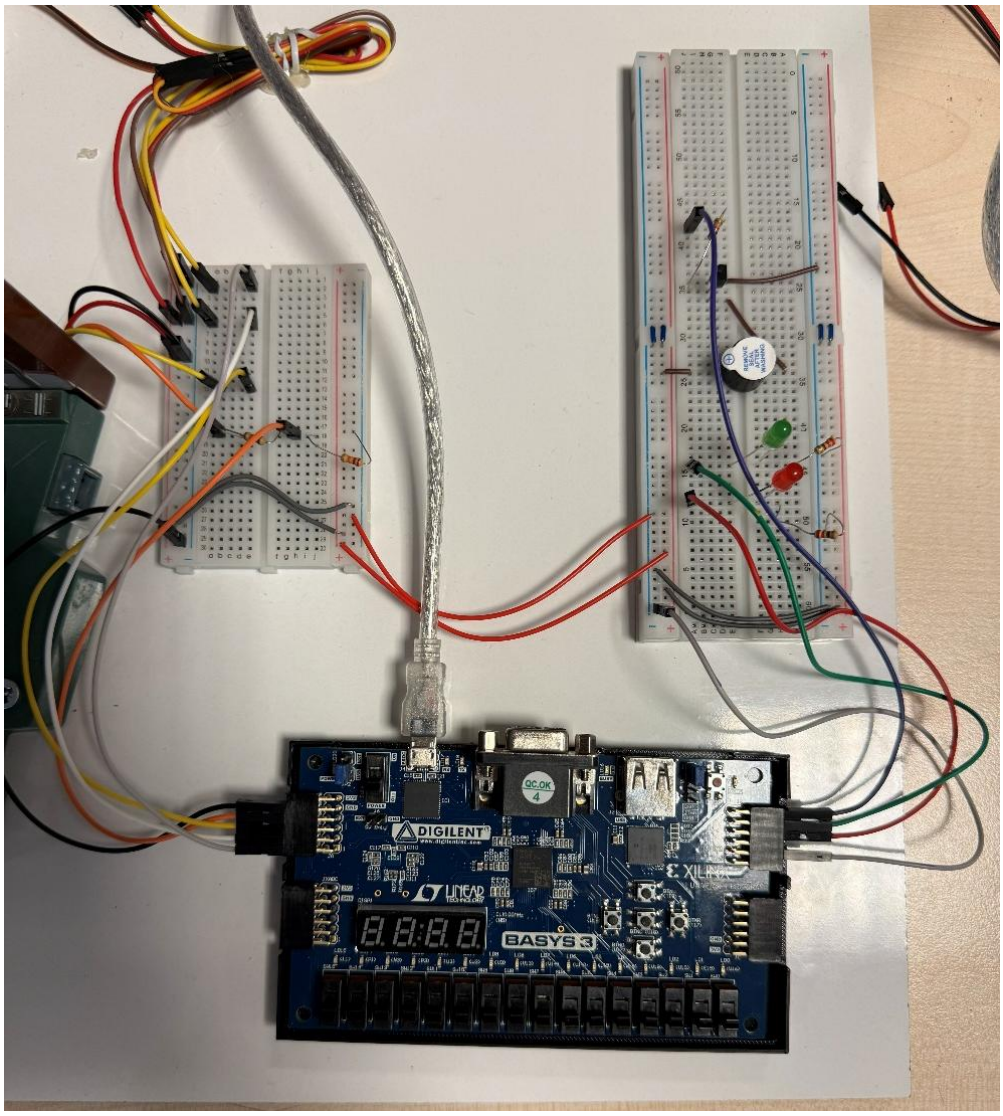

Figure 11: Basys3 FPGA and breadboard implementations of the system components

Results

After the physical implementation and uploading the VHDL code to the Basys3 FPGA, the behavior of the system was tested. First, the system is initial state where the buzzer and LEDs remain inactive while the proximity sensor continuously checks for an object. Figure 12 shows the initial state of the system after power is applied to the Basys3 FPGA. The system waits for a password input or a signal from the proximity sensor.

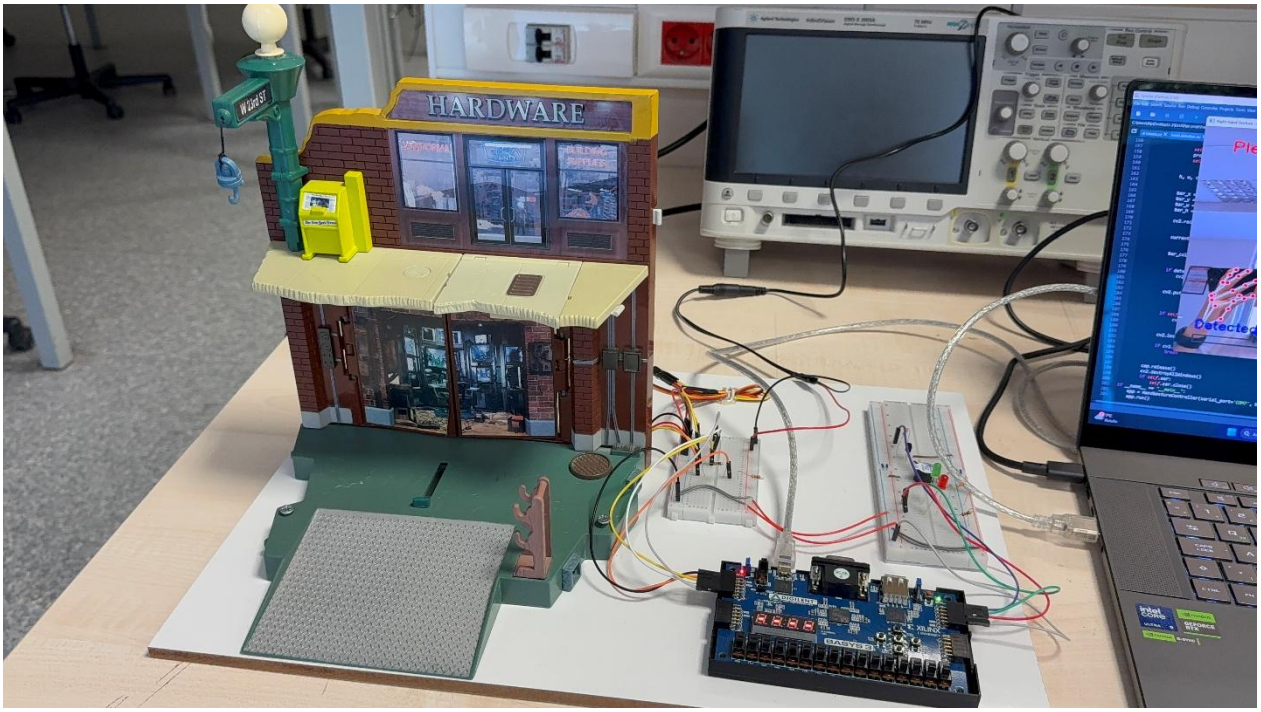

Figure 12: Initial state of the Door Security System

At first, the system was tested for the outside entry scenario, where access to the door requires entering the correct password. Using password, the digits were entered sequentially through Python hand detection module. As it can be observed in Figure 13, the seven-segment display initially shows "- - - -" since no digit is entered through the Python code. As shown in Figures 14–16, the first three digits were entered and displayed on the seven-segment display.

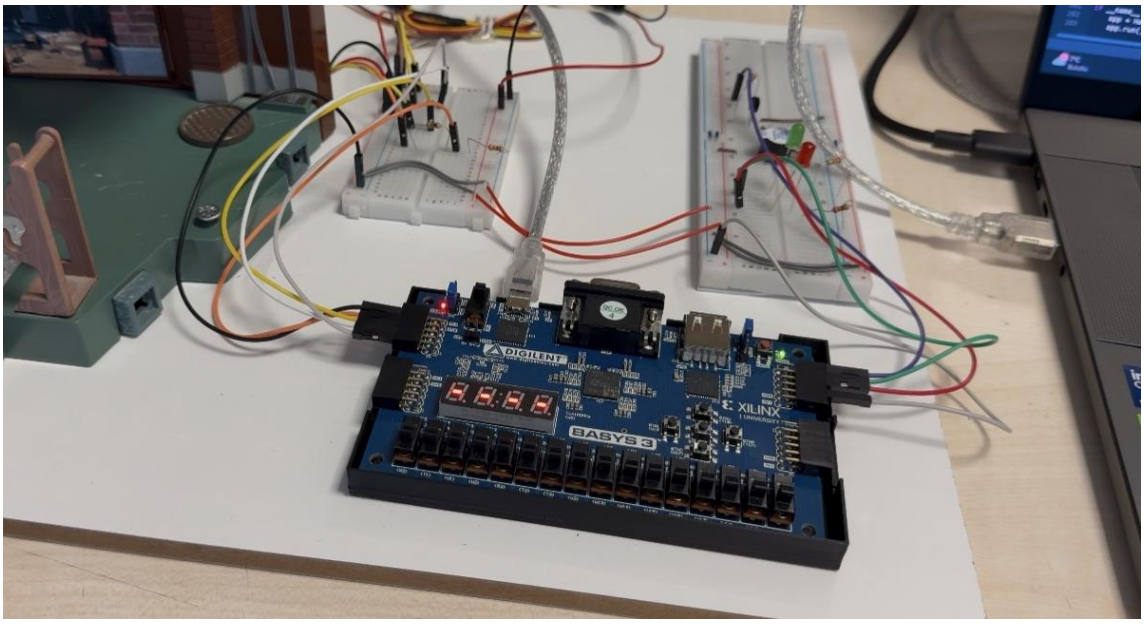

Figure 13: Initial state of the Seven Segment Display

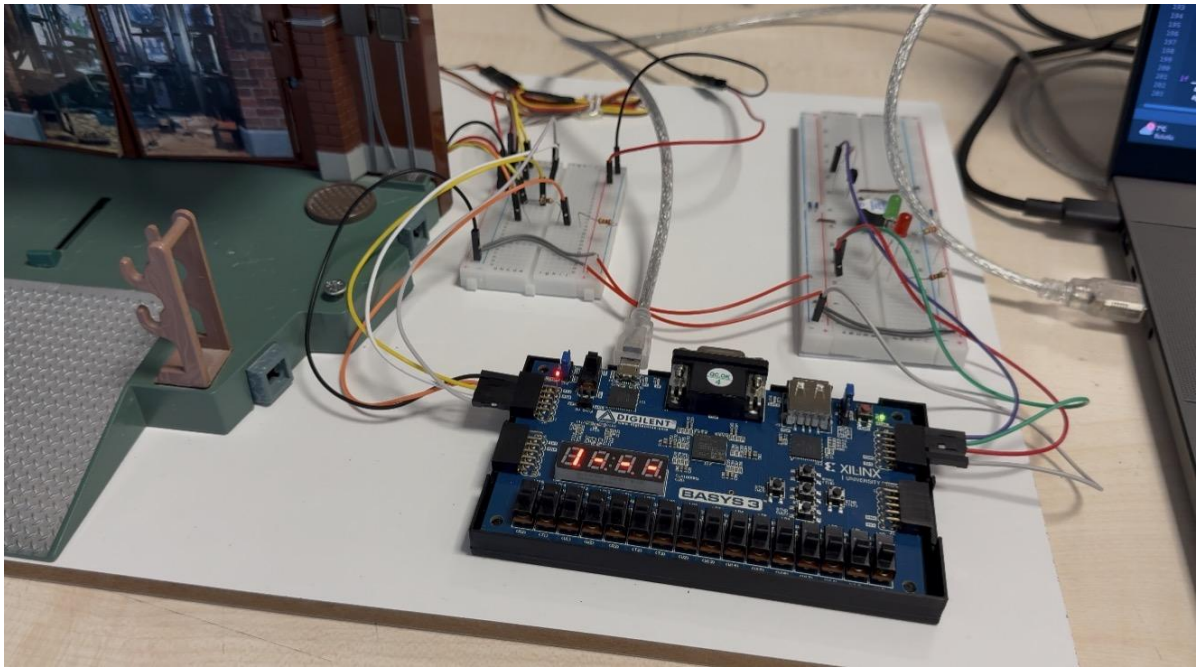

Figure 14: Seven Segment Display when the first digit is entered

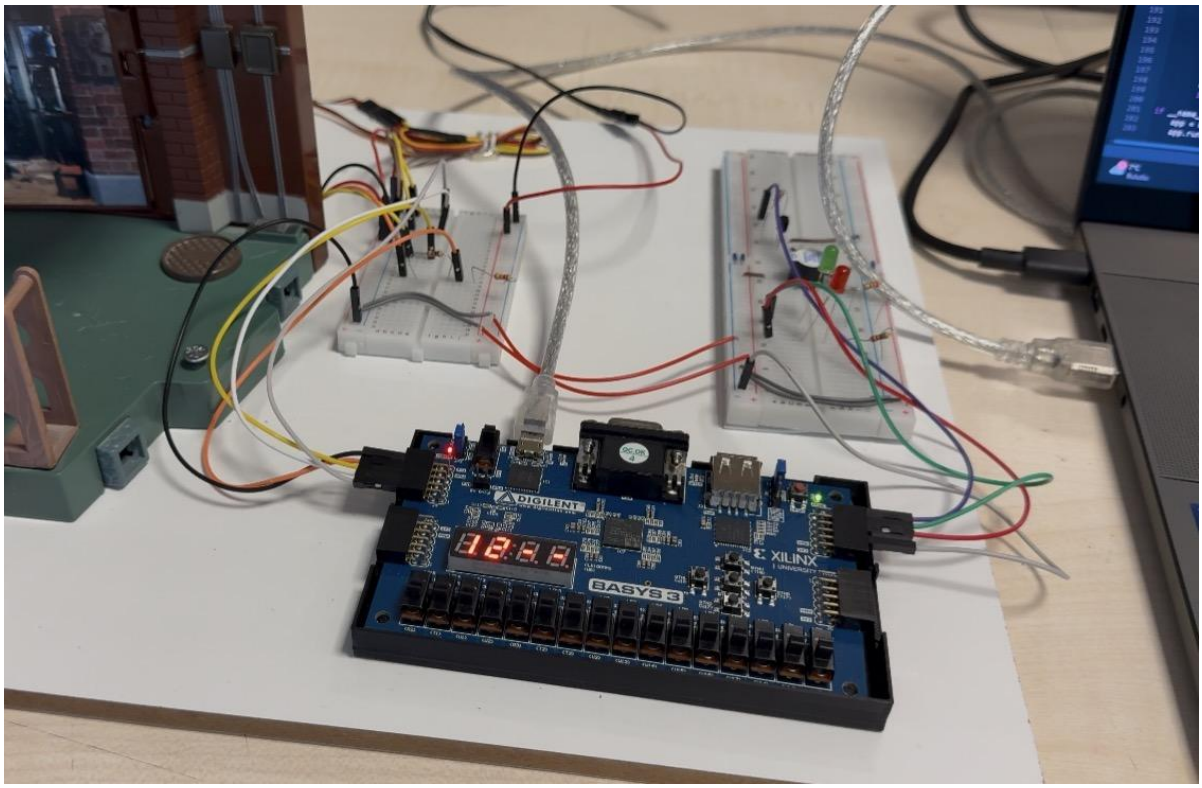

Figure 15: Seven Segment Display when the second digit is entered

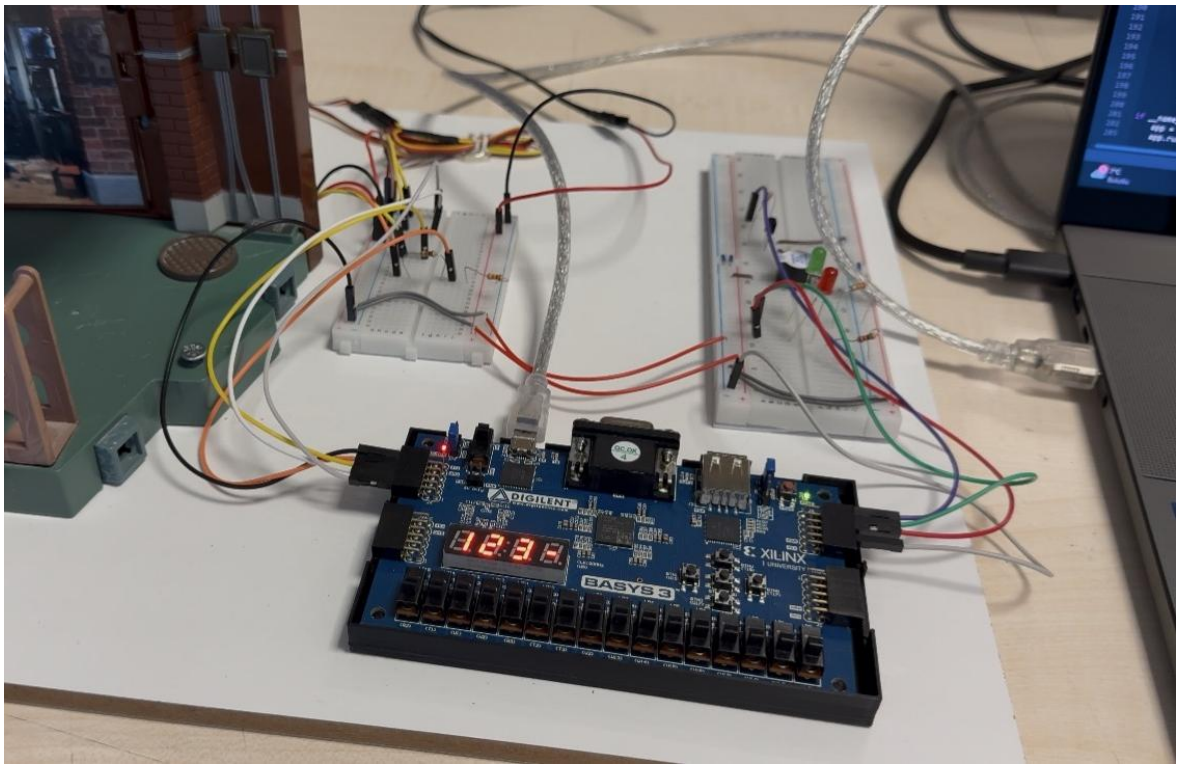

Figure 16: Seven Segment Display when the third digit is entered

After the correct password “1234” is entered, the system automatically opens the door and activates the green LED, as shown in Figure 17. The correct password activated the two servo motors, and the system behaved as expected.

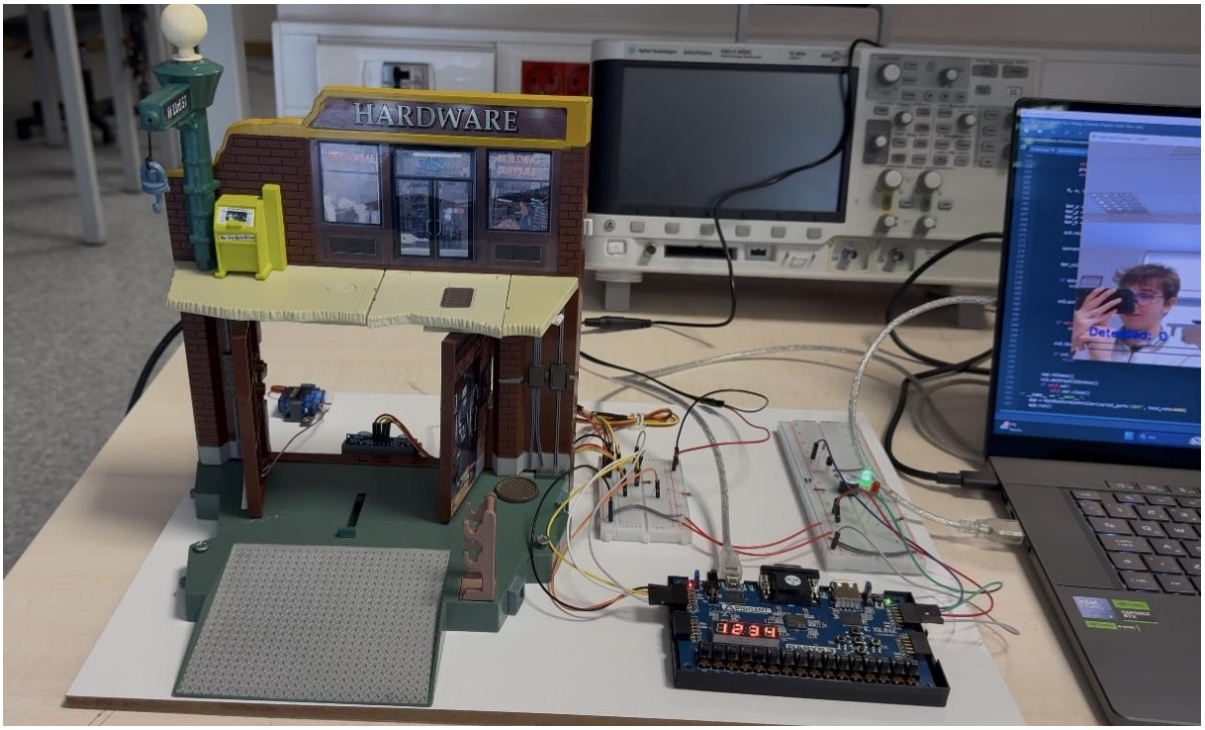

Figure 17: Door opened after the correct password is entered

After that, the system was tested with an invalid password to ensure that the alarm system was working properly. As shown in Figures 18–19, the incorrect password was entered sequentially through the hand-detection module. The incorrect password triggers the buzzer and the red LED, indicating an alarm condition.

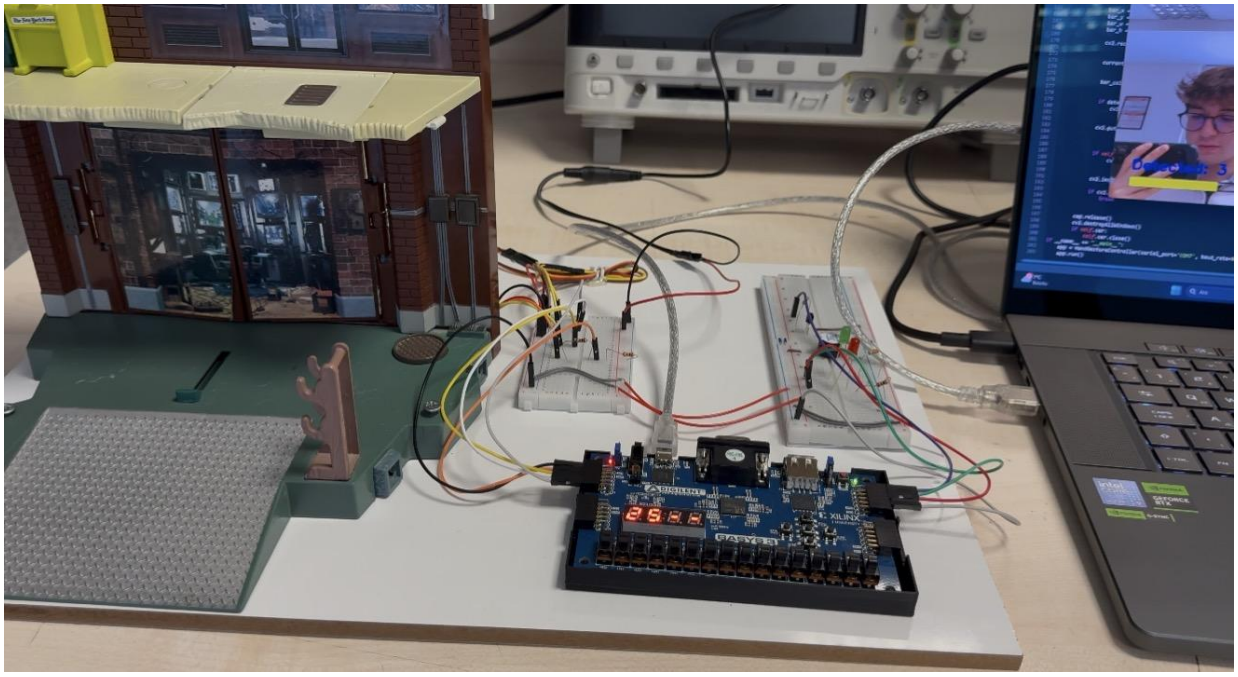

Figure 18: Seven Segment Display when the two digits are entered

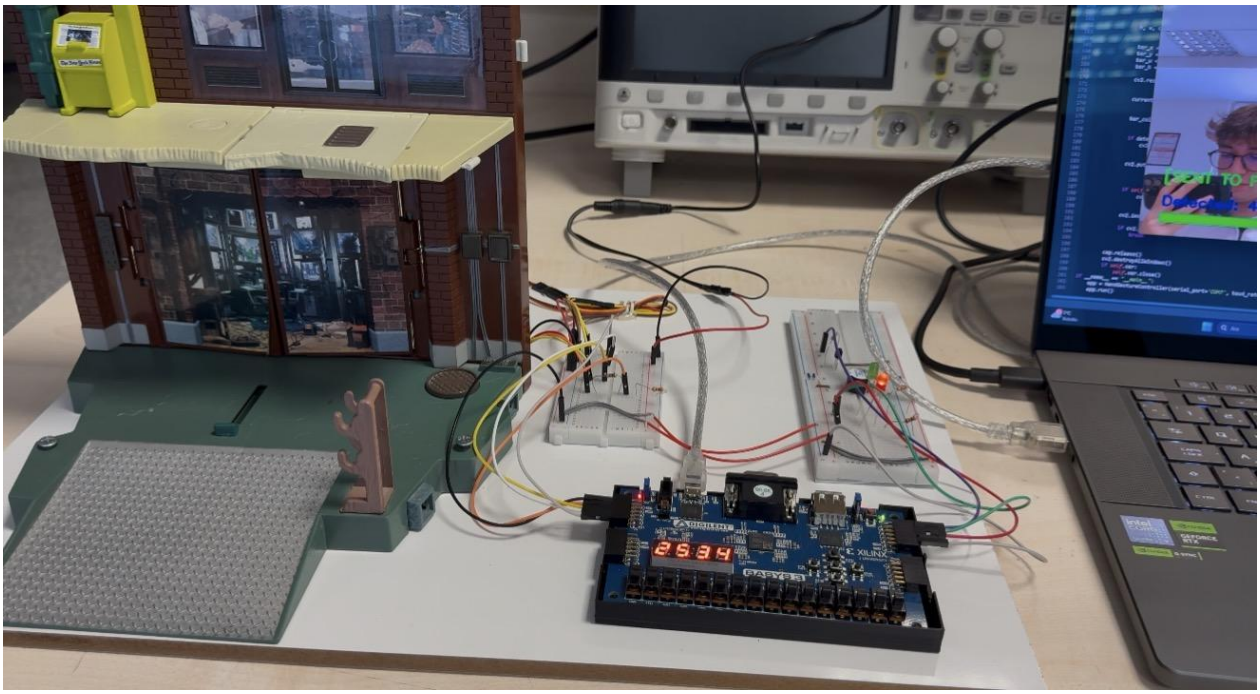

Figure 19: The alarm system is activated since the password was incorrect

Finally, the system was tested for the inside exit scenario. When no object is detected by the proximity sensor, the door remains in the closed position. As an object approaches the door, the proximity sensor is activated and the door is opened automatically. These results can be seen in Figure 20-21.

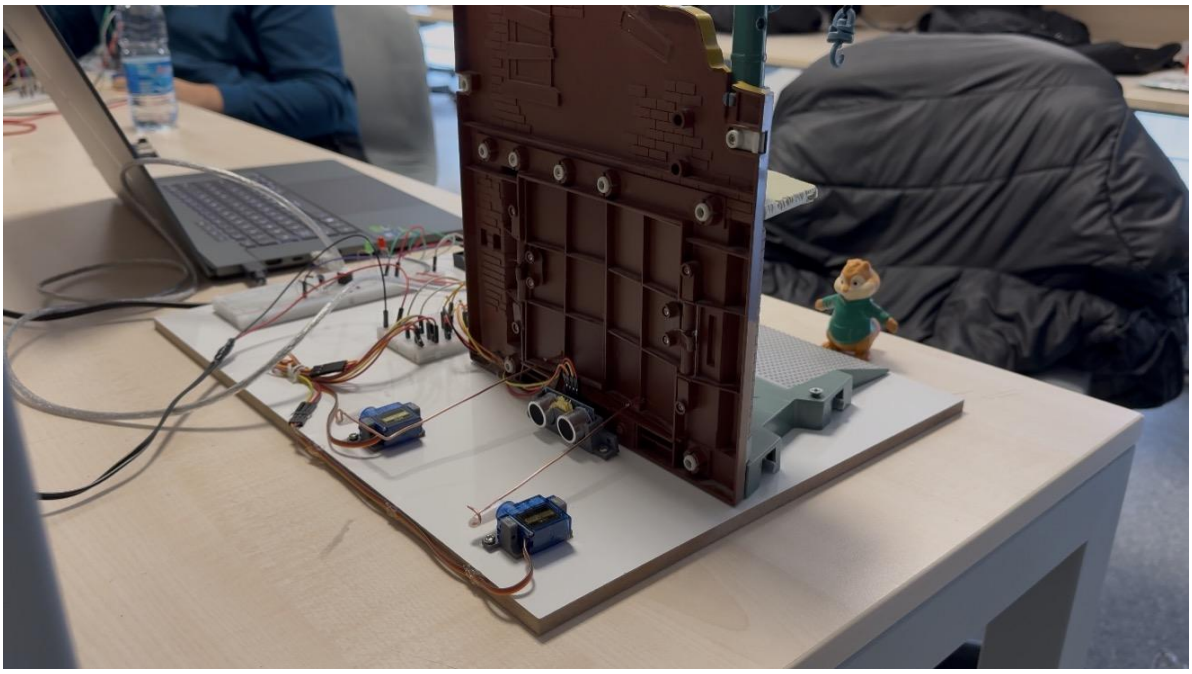

Figure 20: When Theodore is not detected, the door remains closed

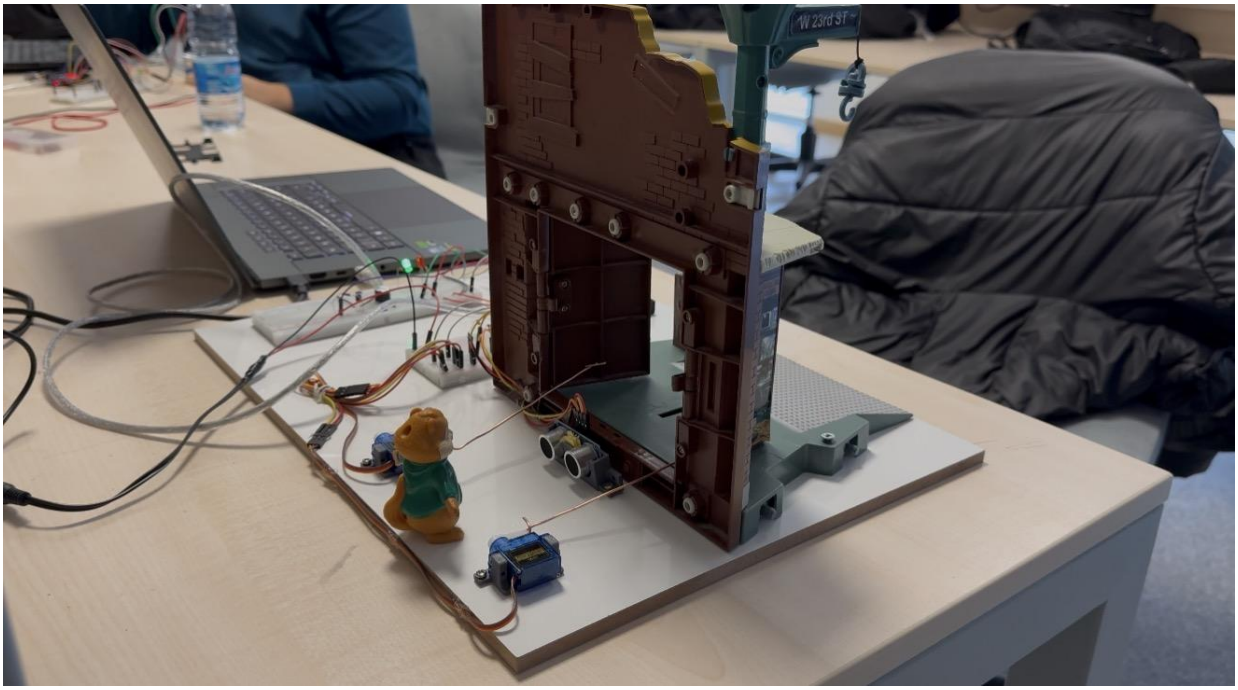

Figure 21: When Theodore approaches the door, the door is opened

The reset button is used to return the system to its initial state and it operated correctly in all tests as well. Overall, the system behaved as expected and the implementation was completed successfully. All possible operating scenarios were tested, and the system responded correctly in each case.

Conclusion

In this project, I aimed to design a fully functional password protected door security system. To achieve this, I used a Basys3 FPGA. I implemented VHDL code into it to handle all the decision making processes of the security system. I designed a main controller module to control the core logic, where all system operations are determined. Since multiple states must be considered, designing and debugging the main controller was one of the most challenging parts of this project. For the physical setup, I used external components such as SG90 servo motors, HC-SR04 ultrasonic sensor, buzzer and LEDs. In this way, I learned how to integrate external components with Basys3.

For the password logic, I used Python hand-detection module to receive input from the user. The detected input is then directly sent to Basys3 FPGA through UART module. Therefore, this project mainly operates using VHDL logic on Basys3 FPGA. I utilized external software modules and hardware components. Through this process, I learned how the Basys3 FPGA can be effectively used and how different hardware and software components can be integrated into a fully functional system. I developed the project step by step at each stage. In the final stage, I added a proximity sensor to allow exiting from the inside. This extension makes the door security system fully functional.

Although these were the main benefits for me, I also encountered many challenges while making the project. Understanding the UART module and how it operates was particularly difficult at first. Also, I made a lot of researches to learn how the external components function. Sometimes, I modified the existing code structures to properly fit the requirements of my project and cited all the sources I used. For future developments, a wrong password counter can be implemented to temporarily lock the system after a certain number of incorrect attempts. This project and its hardware implementation required a significant amount of time. However, through this process, I learned how to implement everything I learned in this course to a fully working system. In addition, developing a real-world applicable project increased the reliability and practicality of the overall door security design.

References

Digilent. (n.d.). *Basys 3 FPGA board reference manual*.

https://digilent.com/reference/programmable-logic/basys-3/reference-manual

Jojic, B. (2012, February 20). *Servomotor control with PWM and VHDL*. CodeProject.

https://www.codeproject.com/articles/Servomotor-Control-with-PWM-and-VHDL

Larson, S. (2021, March 12). *UART (VHDL)*. DigiKey TechForum.

https://forum.digikey.com/t/uart-vhdl/12670

Madison, D. (2017). *LED-Segment-ASCII* [Computer software]. GitHub.

https://github.com/dmadison/LED-Segment-ASCII

Merrick, R. (n.d.). *UART Serial Port Module*. Nandland.

https://nandland.com/uart-serial-port-module/

Morgan, E. J. (2014, November 16). *HC-SR04 Ultrasonic Sensor* [Datasheet]. AllDatasheet.

https://www.alldatasheet.com/datasheet-pdf/view/1132204/ETC2/HCSR04.html

Appendices

top_module.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity System_Top is
    Port (
        clk          : in STD_LOGIC;
        Rx_in        : in STD_LOGIC;
        btnC         : in STD_LOGIC;
        seg          : out STD_LOGIC_VECTOR (6 downto 0);
        an           : out STD_LOGIC_VECTOR (3 downto 0);
        ja1_servo    : out STD_LOGIC;
        ja2_servo    : out STD_LOGIC;
        ja3_trig     : out STD_LOGIC;
        ja4_echo     : in  STD_LOGIC;
        jb1_buzzer   : out STD_LOGIC;
        jb2_led_green : out STD_LOGIC;
        jb3_led_red  : out STD_LOGIC
    );
end System_Top;
architecture Behavioral of System_Top is
    signal w_rx_data : std_logic_vector(7 downto 0);
    signal w_rx_done : std_logic;
    signal w_d1, w_d2, w_d3, w_d4 : std_logic_vector(3 downto 0);
    signal w_mux_data : std_logic_vector(3 downto 0);
    signal w_servo_enable : std_logic;
    signal w_green_en    : std_logic;
    signal w_alarm_en    : std_logic;
    signal w_red_blink   : std_logic;
```

```
    signal w_buzzer_tone  : std_logic;
    signal shared_pwm     : std_logic;
    signal w_sens_det     : std_logic;
begin
    UART_INST : entity work.uart_rx
    port map (
        clk         => clk,
        rx_pin_in   => Rx_in,
        rx_data_out  => w_rx_data,
        rx_done_tick => w_rx_done
    );
    CTRL_INST : entity work.Main_Controller
    port map (
        clk           => clk,
        rst           => btnC,
        rx_data_in    => w_rx_data,
        rx_done_in    => w_rx_done,
        sensor_trig_in => w_sens_det,
        o_digit1      => w_d1,
        o_digit2      => w_d2,
        o_digit3      => w_d3,
        o_digit4      => w_d4,
        o_servo_start  => w_servo_enable,
        o_green_led_en => w_green_en,
        o_alarm_en    => w_alarm_en
    );
    ALARM_INST : entity work.Alarm_Driver
    port map (
        clk       => clk,
        enable    => w_alarm_en,
```

```
    led_out    => w_red_blink,
    buzzer_out => w_buzzer_tone
);
SCAN_INST : entity work.Seven_Segment_Scanner
port map (
    clk        => clk,
    reset      => btnC,
    in_d1      => w_d1,
    in_d2      => w_d2,
    in_d3      => w_d3,
    in_d4      => w_d4,
    o_mux_data => w_mux_data,
    o_anode    => an
);
DEC_INST : entity work.Seven_Segment_Decoder
port map (
    i_hex => w_mux_data,
    o_seg => seg
);
SERVO_INST : entity work.Servo_Driver
port map (
    clk       => clk,
    servo_en  => w_servo_enable,
    servo_pwm => shared_pwm
);
SENS_INST : entity work.Proximity_Sensor
port map (
    clk     => clk,
    trig    => ja3_trig,
    echo    => ja4_echo,
```

```
        obj_det => w_sens_det
    );
    ja1_servo <= shared_pwm;
    ja2_servo <= shared_pwm;
    jb2_led_green <= w_green_en;
    jb3_led_red   <= w_red_blink;
    jb1_buzzer    <= w_buzzer_tone;
end Behavioral;
```

uart.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity uart_rx is
    Port (
        clk          : in  STD_LOGIC;
        rx_pin_in    : in  STD_LOGIC;
        rx_data_out  : out STD_LOGIC_VECTOR (7 downto 0);
        rx_done_tick : out STD_LOGIC
    );
end uart_rx;
architecture Behavioral of uart_rx is
    type t_state is (IDLE, START, DATA, STOP, DONE);
    signal state : t_state := IDLE;
    -- 100 MHz  9600 baud: for each bit 10416 clock
    constant BIT_TIMER_LIMIT : integer := 10416;
    signal timer_counter     : integer range 0 to BIT_TIMER_LIMIT := 0;
    signal bit_index         : integer range 0 to 7 := 0;
    signal rx_buffer         : std_logic_vector(7 downto 0) := (others => '0');
```

```
begin
    process(clk)
    begin
        if rising_edge(clk) then
            case state is
                when IDLE =>
                    rx_done_tick <= '0';
                    timer_counter <= 0;
                    bit_index <= 0;
                    if rx_pin_in = '0' then
                        state <= START;
                    end if;
                when START =>
                    if timer_counter = (BIT_TIMER_LIMIT / 2) then
                        if rx_pin_in = '0' then
                            timer_counter <= 0;
                            state <= DATA;
                        else
                            state <= IDLE;
                        end if;
                    else
                        timer_counter <= timer_counter + 1;
                    end if;
                when DATA =>
                    if timer_counter = BIT_TIMER_LIMIT - 1 then
                        timer_counter <= 0;
                        rx_buffer(bit_index) <= rx_pin_in;

                        if bit_index < 7 then
                            bit_index <= bit_index + 1;
```

```
                else
                    bit_index <= 0;
                    state <= STOP;
                end if;
            else
                timer_counter <= timer_counter + 1;
            end if;
        when STOP =>
            if timer_counter = BIT_TIMER_LIMIT - 1 then
                state <= DONE;
                timer_counter <= 0;
            else
                timer_counter <= timer_counter + 1;
            end if;
        when DONE =>
            rx_done_tick <= '1';
            rx_data_out <= rx_buffer;
            state <= IDLE;
        when others =>
            state <= IDLE;
    end case;
  end if;
 end process;
end Behavioral;
```

main_controller.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Main_Controller is
```

```
    Port (
        clk           : in  STD_LOGIC;
        rst           : in  STD_LOGIC;
        rx_data_in    : in  STD_LOGIC_VECTOR (7 downto 0);
        rx_done_in    : in  STD_LOGIC;
        o_digit1      : out STD_LOGIC_VECTOR (3 downto 0);
        o_digit2      : out STD_LOGIC_VECTOR (3 downto 0);
        o_digit3      : out STD_LOGIC_VECTOR (3 downto 0);
        o_digit4      : out STD_LOGIC_VECTOR (3 downto 0);
        o_servo_start  : out STD_LOGIC;
        o_green_led_en : out STD_LOGIC;
        o_alarm_en    : out STD_LOGIC;
        sensor_trig_in : in  STD_LOGIC
    );
end Main_Controller;
architecture Behavioral of Main_Controller is
    constant PASS_1 : std_logic_vector(7 downto 0) := x"31";
    constant PASS_2 : std_logic_vector(7 downto 0) := x"32";
    constant PASS_3 : std_logic_vector(7 downto 0) := x"33";
    constant PASS_4 : std_logic_vector(7 downto 0) := x"34";
    signal r_d1, r_d2, r_d3, r_d4 : std_logic_vector(3 downto 0) := "1010";
    signal digit_counter : integer range 0 to 4 := 0;
    signal p1_ok, p2_ok, p3_ok : std_logic := '0';
    signal door_latch  : std_logic := '0';
    signal alarm_latch : std_logic := '0';
    constant STABLE_LIMIT : integer := 4000000;
    signal stable_counter : integer range 0 to STABLE_LIMIT := 0;
begin
    process(clk, rst)
    begin
```

```
if rst = '1' then
    digit_counter <= 0;
    r_d1 <= "1010"; r_d2 <= "1010"; r_d3 <= "1010"; r_d4 <= "1010";
    p1_ok <= '0'; p2_ok <= '0'; p3_ok <= '0';
    door_latch  <= '0';
    alarm_latch <= '0';
    stable_counter <= 0;
elsif rising_edge(clk) then
    if sensor_trig_in = '1' then
        if stable_counter < STABLE_LIMIT then
            stable_counter <= stable_counter + 1;
        else
            door_latch <= '1';
            alarm_latch <= '0';
        end if;
    else
        stable_counter <= 0;
    end if;
    if rx_done_in = '1' then
        case digit_counter is
            when 0 =>
                r_d1 <= rx_data_in(3 downto 0);
                if rx_data_in = PASS_1 then p1_ok <= '1'; else p1_ok <= '0'; end if;
                digit_counter <= 1;
            when 1 =>
                r_d2 <= rx_data_in(3 downto 0);
                if rx_data_in = PASS_2 then p2_ok <= '1'; else p2_ok <= '0'; end if;
                digit_counter <= 2;
            when 2 =>
                r_d3 <= rx_data_in(3 downto 0);
```

```
                if rx_data_in = PASS_3 then p3_ok <= '1'; else p3_ok <= '0'; end if;
                digit_counter <= 3;
            when 3 =>
                r_d4 <= rx_data_in(3 downto 0);
                if (p1_ok='1' and p2_ok='1' and p3_ok='1' and rx_data_in=PASS_4) then
                    door_latch  <= '1';
                    alarm_latch <= '0';
                else
                    if door_latch = '0' then
                        alarm_latch <= '1';
                    end if;
                end if;
                digit_counter <= 4;
            when others =>
                digit_counter <= 0;
                r_d1 <= "1010"; r_d2 <= "1010"; r_d3 <= "1010"; r_d4 <= "1010";
                door_latch  <= '0';
                alarm_latch <= '0';
                p1_ok <= '0'; p2_ok <= '0'; p3_ok <= '0';
        end case;
    end if;
  end if;
end process;
o_servo_start  <= door_latch;
o_green_led_en <= door_latch;
o_alarm_en     <= alarm_latch;
o_digit1 <= r_d1;
o_digit2 <= r_d2;
o_digit3 <= r_d3;
o_digit4 <= r_d4;
```

```
end Behavioral;
```

alarm.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Alarm_Driver is
    Port (
        clk       : in  STD_LOGIC;
        enable    : in  STD_LOGIC;
        led_out   : out STD_LOGIC;
        buzzer_out: out STD_LOGIC
    );
end Alarm_Driver;
architecture Behavioral of Alarm_Driver is
    constant C_BLINK_CYCLES : integer := 25000000;
    signal blink_counter    : integer range 0 to C_BLINK_CYCLES := 0;
    signal blink_state      : std_logic := '0';
    constant C_TONE_CYCLES  : integer := 25000;
    signal tone_counter     : integer range 0 to C_TONE_CYCLES := 0;
    signal tone_state       : std_logic := '0';
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if enable = '1' then
                if blink_counter < C_BLINK_CYCLES - 1 then
                    blink_counter <= blink_counter + 1;
                else
                    blink_counter <= 0;
                    blink_state   <= not blink_state;
```

```
            end if;
            if tone_counter < C_TONE_CYCLES - 1 then
               tone_counter <= tone_counter + 1;
            else
               tone_counter <= 0;
               tone_state   <= not tone_state;
            end if;
         else
            blink_counter <= 0;
            blink_state   <= '0';
            tone_counter  <= 0;
            tone_state    <= '0';
         end if;
      end if;
   end process;
   led_out <= blink_state when enable = '1' else '0';
   buzzer_out <= tone_state when (enable = '1' and blink_state = '1') else '0';
end Behavioral;
```

seven_seg_driver.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Seven_Segment_Scanner is
   Port (
      clk       : in  STD_LOGIC;
      reset     : in  STD_LOGIC;
      in_d1     : in  STD_LOGIC_VECTOR (3 downto 0);
      in_d2     : in  STD_LOGIC_VECTOR (3 downto 0);
      in_d3     : in  STD_LOGIC_VECTOR (3 downto 0);
      in_d4     : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
        o_mux_data : out STD_LOGIC_VECTOR (3 downto 0);
        o_anode    : out STD_LOGIC_VECTOR (3 downto 0)
    );
end Seven_Segment_Scanner;
architecture Behavioral of Seven_Segment_Scanner is
    signal refresh_counter : std_logic_vector(19 downto 0) := (others => '0');
    signal sel : std_logic_vector(1 downto 0);
begin
    process(clk, reset)
    begin
        if(reset='1') then
            refresh_counter <= (others => '0');
        elsif(rising_edge(clk)) then
            refresh_counter <= refresh_counter + 1;
        end if;
    end process;
    sel <= refresh_counter(19 downto 18);
    process(sel, in_d1, in_d2, in_d3, in_d4)
    begin
        case sel is
            when "00" =>
                o_anode    <= "0111";
                o_mux_data <= in_d1;
            when "01" =>
                o_anode    <= "1011";
                o_mux_data <= in_d2;
            when "10" =>
                o_anode    <= "1101";
                o_mux_data <= in_d3;
            when "11" =>
```

```
            o_anode    <= "1110";
            o_mux_data <= in_d4;
        when others =>
            o_anode    <= "1111";
            o_mux_data <= "0000";
    end case;
  end process;
end Behavioral;
```

seven_seg_decoder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Seven_Segment_Decoder is
  Port (
    i_hex : in  STD_LOGIC_VECTOR (3 downto 0);
    o_seg : out STD_LOGIC_VECTOR (6 downto 0)
  );
end Seven_Segment_Decoder;
architecture Behavioral of Seven_Segment_Decoder is
begin
  process(i_hex)
  begin
    case i_hex is
      when "0001" => o_seg <= "1001111"; -- 1
      when "0010" => o_seg <= "0010010"; -- 2
      when "0011" => o_seg <= "0000110"; -- 3
      when "0100" => o_seg <= "1001100"; -- 4
      when "0101" => o_seg <= "0100100"; -- 5
      -- (-) (Input 10: "1010" )
      when "1010" => o_seg <= "1111110";
      when others => o_seg <= "1111111";
```

```
        end case;
    end process;
end Behavioral;
```

servo.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Servo_Driver is
    Port (
        clk       : in  STD_LOGIC;
        servo_en  : in  STD_LOGIC;
        servo_pwm : out STD_LOGIC
    );
end Servo_Driver;
architecture Behavioral of Servo_Driver is
    -- 20ms period (50Hz) -> 2,000,000
    constant C_PWM_PERIOD : integer := 2000000;
    -- 0.5 ms * 100 MHz = 50,000 (Closed)
    constant C_SERVO_CLOSED : integer := 50000;
    -- 2.5 ms * 100 MHz = 250,000 (Open)
    constant C_SERVO_OPEN : integer := 250000;
    signal pwm_counter : integer range 0 to C_PWM_PERIOD := 0;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if pwm_counter < C_PWM_PERIOD - 1 then
                pwm_counter <= pwm_counter + 1;
            else
                pwm_counter <= 0;
```

```
            end if;
            if servo_en = '1' then
                if pwm_counter < C_SERVO_OPEN then
                    servo_pwm <= '1';
                else
                    servo_pwm <= '0';
                end if;
            else
                if pwm_counter < C_SERVO_CLOSED then
                    servo_pwm <= '1';
                else
                    servo_pwm <= '0';
                end if;
            end if;
        end if;
    end process;
end Behavioral;
```

proximity_sensor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Proximity_Sensor is
    Port (
        clk       : in  STD_LOGIC;
        trig      : out STD_LOGIC;
        echo      : in  STD_LOGIC;
        obj_det   : out STD_LOGIC
    );
end Proximity_Sensor;
architecture Behavioral of Proximity_Sensor is
```

```
    signal count_timer : integer range 0 to 10000000 := 0;
    signal count_echo  : integer range 0 to 2000000  := 0;
    signal echo_last   : std_logic := '0';
    signal waiting     : std_logic := '0';
    constant THRESHOLD : integer := 88000;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if count_timer < 10000000 then count_timer <= count_timer + 1;
            else count_timer <= 0; end if;
            if count_timer < 1000 then
                trig <= '1';
                waiting <= '1';
                count_echo <= 0;
            else
                trig <= '0';
            end if;
            if waiting = '1' then
                if echo = '1' then
                    count_echo <= count_echo + 1;
                elsif echo_last = '1' and echo = '0' then
                    waiting <= '0';
                end if;
            end if;
            echo_last <= echo;
            if (count_echo > 15000) and (count_echo < THRESHOLD) then
                obj_det <= '1';
            else
                obj_det <= '0';
```

```
            end if;
        end if;
    end process;
end Behavioral;
```