EEE-102 Project: Digital Security and Access Control System

Date: 14.05.2025

Name: Hasan Mert Soycan

Section: 02

ID No: 22403891

1. Purpose

The purpose of this project is to design and implement a functional digital security and access control system using the Basys 3 FPGA board and VHDL. The system simulates a keypad-based digital door lock mechanism that includes authentication, feedback, and alarm functionalities. It demonstrates how finite state machines (FSMs), seven-segment displays, servo motors, buzzer alarms, and peripheral sensors such as motion detectors can be integrated into a coherent digital system.

The experiment aims to enhance understanding of FSM design, input validation, modular hardware description, and output control through VHDL, all while handling real-time interactions with multiple hardware components.

2. Design Specifications

The system is designed to function as a basic access control unit with digital logic components implemented on an FPGA. The key functional specifications and design expectations are the following:

System Goals:

- Accept a 4-digit password input via a matrix keypad.
- Unlock a door mechanism using a servo motor when the correct password is entered.
- Illuminate one of three LEDs after each incorrect attempt.
- Trigger an alarm (buzzer and flashing LEDs) after three consecutive incorrect attempts.
- Display each entered digit on a 7-segment display, blanking out the remaining digits.

- Block any further input while the alarm is active.
- Use a motion sensor (PIR) to control an external light, independent of the main FSM logic.

Input Devices:

- **Keypad (4x4 matrix):** used to enter the password.
- **Verify button (BTN0):** manually confirms each digit entry to avoid ghosting issues from the keypad.
- **PIR sensor (external):** detects motion and triggers a light via transistor.

Output Devices:

- **7-segment display:** visual feedback for entered digits.
- **3 LEDs (progress indicators):** light up after each wrong attempt.
- **Servo motor:** simulates door unlocking.
- **Buzzer:** alarm signal.
- **External LED (motion-triggered):** controlled via transistor by the PIR sensor.

System Logic:

- The system operates through a clearly defined FSM structure with the following states: `IDLE`, `DIGIT1`, `DIGIT2`, `DIGIT3`, `CHECK`, `OPEN_STATE`, `WAIT_10S`, `CLOSE`, `LOCKED_WAIT`, and `LOCKED_ALARM`.
- A correct 4-digit code (default: "1283") causes the servo motor to open the lock and display the full code on the display.
- Three incorrect attempts cause the system to lock and activate the alarm for 30 seconds with a 0.75s on/off buzzer and blinking LEDs.
- During alarm, further input is ignored, and LEDs are synchronized with the buzzer.

3. Methodology

The system was implemented using a modular and hierarchical VHDL design approach. Each subsystem—such as keypad input handling, 7-segment display driving, servo PWM generation, and alarm control—was written as an individual module and integrated into

a centralized `top.vhd` file. The behavioral description style and finite state machine (FSM) architecture were used extensively to coordinate user interaction, input validation, and output control.

Finite State Machine (FSM) Design

The FSM controlled the system through a well-defined sequence of states:

- `IDLE`: Waits for the first digit to be entered and verified.
- `DIGIT1`–`DIGIT3`: Collects the next three digits upon pressing the verify button.
- `CHECK`: Compares the entered 4-digit code with the correct password.
- `OPEN_STATE`: Activates the servo motor and unlocks the system.
- `WAIT_10S`: Holds the door open for 10 seconds.
- `CLOSE`: Closes the door and resets the system.
- `LOCKED_WAIT`: Waits 1 second after the third incorrect entry (3 LEDs on).
- `LOCKED_ALARM`: Activates the buzzer and flashing LEDs for 30 seconds.

Each digit was only registered when the verify button was pressed, preventing ghosting issues caused by the keypad. A `debounce` system ensured only clean edges were detected from the mechanical button.

Modules Developed

- **`keypad.vhd`**
  Scans a 4x4 matrix keypad by cycling through row signals and sampling columns. Includes a basic debounce and key validation mechanism.
- **`display_driver.vhd` + `seg_decoder.vhd`**
  Handles 4-digit 7-segment display through digit multiplexing (using `refresh_phase`) and segment pattern decoding for hexadecimal characters.
- **`clk_divider.vhd`**
  Generates slower timing signals from the 100 MHz system clock for display multiplexing and FSM timing delays.
- **`servo_pwm.vhd`**
  Converts a logic signal (`position`) into a fixed-width PWM pulse to drive a servo motor (1ms or 2ms pulses in a 20ms period).

- **`top.vhd`**
  Integrates all modules and coordinates FSM transitions, password checking, fail counter, servo control, display state management, and alarm sequencing.

Transistor-Based Peripheral Control

Due to the limited current-driving capacity of FPGA GPIO pins, transistors were used in two parts of the system:

- **Buzzer:** A transistor circuit allowed the FPGA's PWM signal (`buzzer_on`) to control the buzzer at full current without damaging the GPIO pin.
- **PIR-controlled LED:** The PIR motion sensor triggered an NPN transistor through Basys 3 that powered an external LED.

4. Results

The completed system was successfully implemented on the Basys 3 FPGA development board. All functionalities—including password input, display handling, LED progression, servo control, and alarm triggering—were tested on hardware and seen to work as intended.

System Behavior Observations

- The keypad correctly registered each digit after pressing the verify button.
- Entered digits appeared on the 7-segment display in real time, with only the entered digits visible.
- One LED lit up after each incorrect entry. After three incorrect attempts, the system locked and triggered the alarm.
- During the alarm phase, the buzzer toggled on and off at 0.75-second intervals, and the LEDs blinked synchronously with the buzzer.
- After 30 seconds, the alarm stopped, and the system reset.
- The correct password ("1283") activated the servo motor, rotating it 90° to simulate an unlocking mechanism.
- The PIR sensor correctly activated an external LED when motion was detected. The LED was driven through a transistor for reliable operation.

Photographic Results

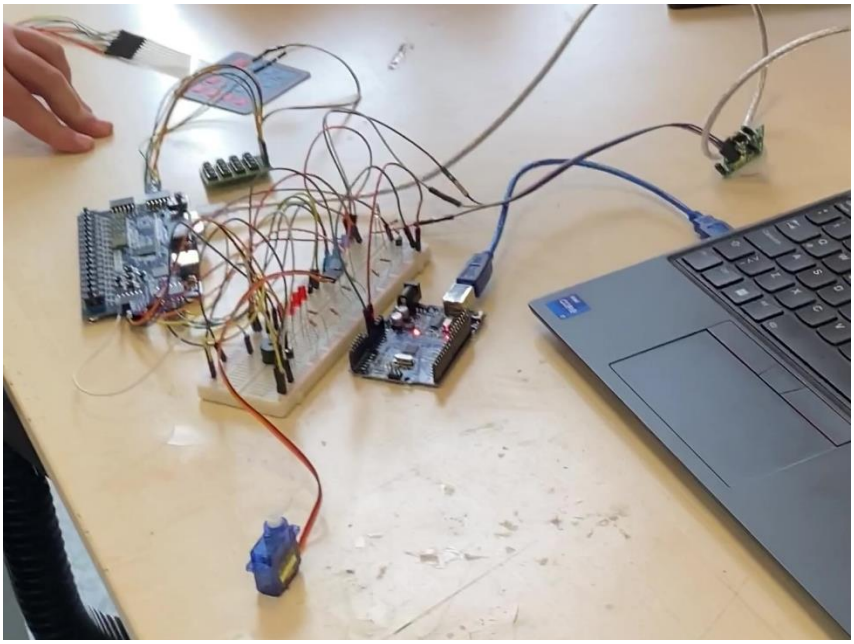

Figure 1: Overall view of the Basys 3 board with all peripheral connections.

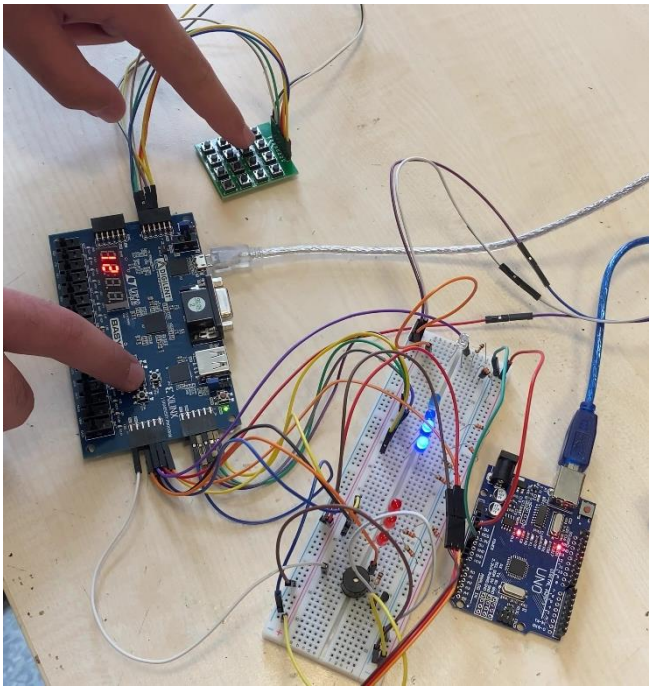

Figure 2: 7-segment display showing entered digits in progression.

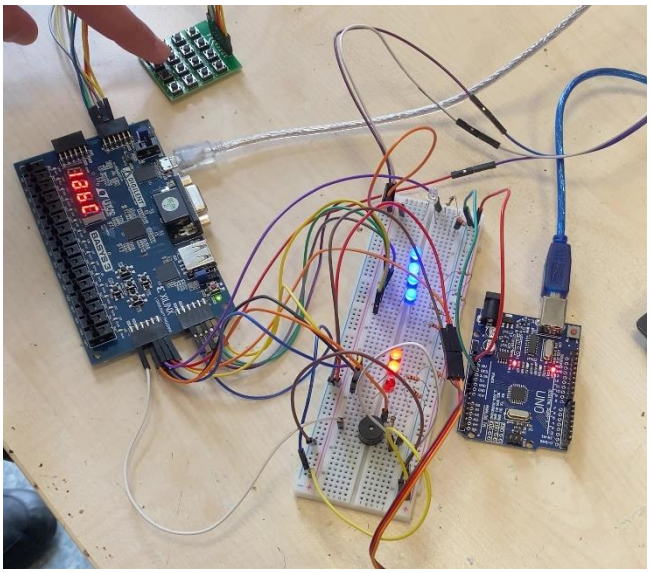

Figure 3: LED indicators lighting up after incorrect password entries.

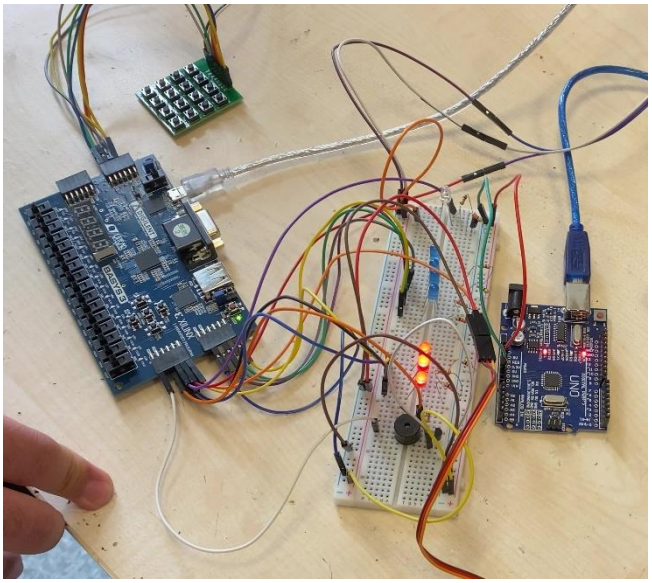

Figure 4: Alarm activation: buzzer on, LEDs blinking.

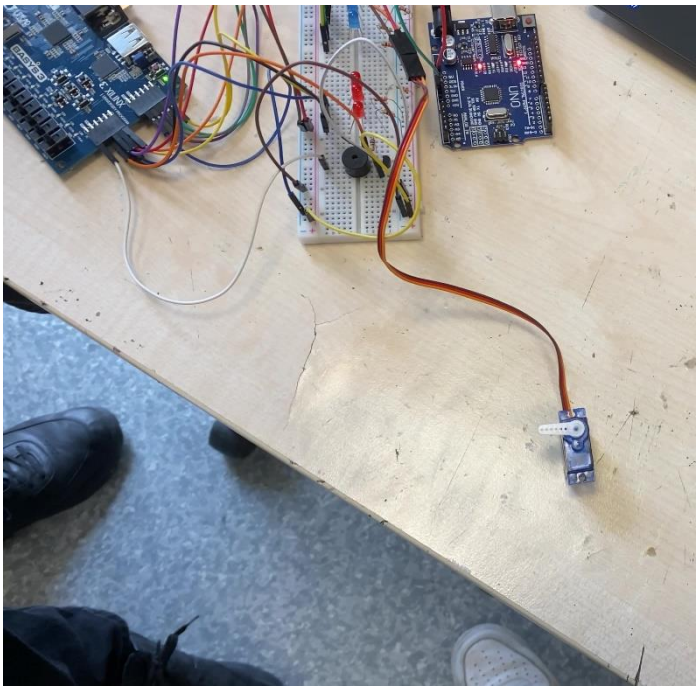

Figure 5: Servo motor unlocked position after successful entry.

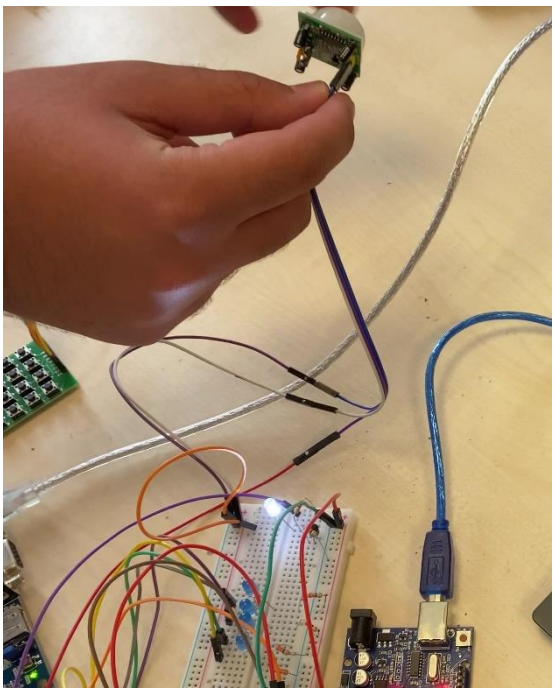

Figure 6: Motion sensor triggering external LED.

5. Conclusion

The objective of this project was to implement a secure and functional digital access control system on the Basys 3 FPGA platform. The final system met all of its intended design specifications, enabling four-digit password input via keypad, real-time display feedback,

error tracking with LEDs, servo-controlled unlocking, and an automatic alarm response after multiple failed attempts. A well-structured finite state machine (FSM) governed the system's behavior, ensuring clean transitions and consistent logic at every interaction stage. Modular VHDL design allowed each component to be developed, tested, and integrated systematically.

During development, one of the most critical technical challenges encountered was related to the keypad. The keypad occasionally produced ghosting, where multiple keys appeared pressed simultaneously, especially when pressing vertically positioned buttons. This led to incorrect digits being entered into the password, which undermined the security logic and produced misleading rejections. Initially, a software-based timed validation system—where key input would only be accepted after a delay—was considered to overcome this issue. However, the most effective and hardware-safe solution was ultimately to require the user to manually confirm each digit using the Basys 3's onboard button (BTN0). This approach guaranteed input reliability by separating key detection from confirmation.

Another critical issue involved output drive limitations. The buzzer, for instance, could not be driven directly by the FPGA due to current constraints. To overcome this, a transistor-based driver circuit was introduced, allowing safe delivery of PWM signals to the buzzer. Likewise, to integrate the motion detection module (PIR sensor) into the system without risking voltage mismatch or GPIO overload, its output was routed through a discrete NPN transistor to control an external LED, ensuring proper isolation and responsiveness.

In conclusion, the project not only achieved its intended functionality but also served as a valuable exercise in solving real-world implementation problems—ranging from input signal integrity to output interfacing. The result is a fully functional, stable, and extensible security control system with both digital logic rigor and practical hardware awareness.

6. Appendices

top.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;
```

```
entity top is
    Port (
        clk : in STD_LOGIC;
        verify_btn : in STD_LOGIC;
        rows : out STD_LOGIC_VECTOR(3 downto 0);
        cols : in STD_LOGIC_VECTOR(3 downto 0);
        pwm_out : out STD_LOGIC;
        display_anodes : out STD_LOGIC_VECTOR(3 downto 0);
        display_segments : out STD_LOGIC_VECTOR(6 downto 0);
        progress_leds : out STD_LOGIC_VECTOR(3 downto 0);
        buzzer : out STD_LOGIC;
        led0 : out STD_LOGIC;
        led1 : out STD_LOGIC;
        led2 : out STD_LOGIC;
        pir_in : in STD_LOGIC;
        pir_out : out STD_LOGIC
    );
end top;

architecture Behavioral of top is

    type state_type is (
        IDLE, DIGIT1, DIGIT2, DIGIT3,
        CHECK, OPEN_STATE, WAIT_10S, CLOSE,
```

```
    LOCKED_WAIT, LOCKED_ALARM
);

signal state : state_type := IDLE;
signal input_code : STD_LOGIC_VECTOR(15 downto 0);
signal correct_code : STD_LOGIC_VECTOR(15 downto 0) := "0001001010000011";
signal key : STD_LOGIC_VECTOR(3 downto 0);
signal key_valid : STD_LOGIC;
signal position : STD_LOGIC := '0';
signal timer : INTEGER := 0;

signal alarm_timer : INTEGER := 0;
signal alarm_cycle : INTEGER range 0 to 80 := 0;
signal buzzer_on : STD_LOGIC := '0';

signal fail_count : INTEGER range 0 to 3 := 0;

signal progress : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');
signal entered_digits : integer range 0 to 4 := 0;

signal verify_btn_prev : STD_LOGIC := '0';
signal verify_edge : STD_LOGIC := '0';
signal debounce_count : INTEGER := 0;
constant debounce_max : INTEGER := 50000000;
```

```
constant check_delay : INTEGER := 100000000;
constant locked_wait_time : INTEGER := 100000000;
constant half_cycle_time : INTEGER := 37500000;

signal refresh_phase : STD_LOGIC_VECTOR(1 downto 0);

component keypad
    Port (
        clk : in STD_LOGIC;
        rows : out STD_LOGIC_VECTOR(3 downto 0);
        cols : in STD_LOGIC_VECTOR(3 downto 0);
        key : out STD_LOGIC_VECTOR(3 downto 0);
        key_valid : out STD_LOGIC
    );
end component;

component servo_pwm
    Port (
        clk : in STD_LOGIC;
        position : in STD_LOGIC;
        pwm_out : out STD_LOGIC
    );
end component;

component clk_divider
```

```
        Port (
            clk : in STD_LOGIC;
            refresh_phase : out STD_LOGIC_VECTOR(1 downto 0)
        );
    end component;

    component display_driver
        Port (
            input_code : in STD_LOGIC_VECTOR(15 downto 0);
            entered_digits : in integer range 0 to 4;
            refresh_phase : in STD_LOGIC_VECTOR(1 downto 0);
            display_anodes : out STD_LOGIC_VECTOR(3 downto 0);
            display_segments : out STD_LOGIC_VECTOR(6 downto 0)
        );
    end component;

begin

    keypad_inst: keypad port map(
        clk => clk,
        rows => rows,
        cols => cols,
        key => key,
        key_valid => key_valid
    );
```

```
servo_inst: servo_pwm port map(
    clk => clk,
    position => position,
    pwm_out => pwm_out
);

clk_div_inst: clk_divider port map(
    clk => clk,
    refresh_phase => refresh_phase
);

display_inst: display_driver port map(
    input_code => input_code,
    entered_digits => entered_digits,
    refresh_phase => refresh_phase,
    display_anodes => display_anodes,
    display_segments => display_segments
);

process(clk)
begin
    if rising_edge(clk) then
        verify_edge <= '0';
        if debounce_count = 0 then
```

```
            if verify_btn = '1' and verify_btn_prev = '0' then
                verify_edge <= '1';
                debounce_count <= debounce_count + 1;
            end if;
        else
            if debounce_count < debounce_max then
                debounce_count <= debounce_count + 1;
            else
                debounce_count <= 0;
            end if;
        end if;
        verify_btn_prev <= verify_btn;
    end if;
end process;

process(clk)
begin
    if rising_edge(clk) then
        case state is
            when IDLE =>
                buzzer_on <= '0';
                if fail_count = 3 then
                    timer <= 0;
                    state <= LOCKED_WAIT;
                elsif verify_edge = '1' and key_valid = '1' then
```

```
            input_code(15 downto 12) <= key;
            state <= DIGIT1;
        end if;

    when DIGIT1 =>
        if verify_edge = '1' and key_valid = '1' then
            input_code(11 downto 8) <= key;
            state <= DIGIT2;
        end if;

    when DIGIT2 =>
        if verify_edge = '1' and key_valid = '1' then
            input_code(7 downto 4) <= key;
            state <= DIGIT3;
        end if;

    when DIGIT3 =>
        if verify_edge = '1' and key_valid = '1' then
            input_code(3 downto 0) <= key;
            timer <= 0;
            state <= CHECK;
        end if;

    when CHECK =>
        if timer < check_delay then
```

```
            timer <= timer + 1;
        else
            timer <= 0;
            if input_code = correct_code then
                fail_count <= 0;
                state <= OPEN_STATE;
            else
                position <= '0';
                if fail_count < 3 then
                    fail_count <= fail_count + 1;
                end if;
                state <= IDLE;
            end if;
        end if;

    when OPEN_STATE =>
        position <= '1';
        timer <= 0;
        state <= WAIT_10S;

    when WAIT_10S =>
        if timer < 1000000000 then
            timer <= timer + 1;
        else
            state <= CLOSE;
```

```
    end if;

when CLOSE =>
    position <= '0';
    state <= IDLE;

when LOCKED_WAIT =>
    buzzer_on <= '0';
    if timer < locked_wait_time then
        timer <= timer + 1;
    else
        timer <= 0;
        alarm_cycle <= 0;
        alarm_timer <= 0;
        state <= LOCKED_ALARM;
    end if;

when LOCKED_ALARM =>
    if alarm_timer < half_cycle_time then
        alarm_timer <= alarm_timer + 1;
    else
        alarm_timer <= 0;
        alarm_cycle <= alarm_cycle + 1;
        buzzer_on <= not buzzer_on;
    end if;
```

```
                if alarm_cycle = 80 then
                    alarm_cycle <= 0;
                    buzzer_on <= '0';
                    fail_count <= 0;
                    state <= IDLE;
                end if;

            when others =>
                state <= IDLE;
        end case;
    end if;
end process;

process(state)
begin
    case state is
        when IDLE       => entered_digits <= 0; progress <= "0000";
        when DIGIT1     => entered_digits <= 1; progress <= "0001";
        when DIGIT2     => entered_digits <= 2; progress <= "0011";
        when DIGIT3     => entered_digits <= 3; progress <= "0111";
        when CHECK      => entered_digits <= 4; progress <= "1111";
        when others     => entered_digits <= 0; progress <= "0000";
    end case;
end process;
```

```
    progress_leds <= progress;

    led0 <= buzzer_on when state = LOCKED_ALARM else
            '1' when fail_count >= 1 else '0';

    led1 <= buzzer_on when state = LOCKED_ALARM else
            '1' when fail_count >= 2 else '0';

    led2 <= buzzer_on when state = LOCKED_ALARM else
            '1' when fail_count = 3 else '0';

    buzzer <= buzzer_on;

    pir_out <= '1' when pir_in = '1' else '0';

end Behavioral;
```

keypad.vhd

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

entity keypad is
    Port (
        clk : in STD_LOGIC;
        rows : out STD_LOGIC_VECTOR(3 downto 0);
        cols : in STD_LOGIC_VECTOR(3 downto 0);
        key : out STD_LOGIC_VECTOR(3 downto 0);
        key_valid : out STD_LOGIC
    );
end keypad;

architecture Behavioral of keypad is
    signal row_sel : STD_LOGIC_VECTOR(1 downto 0) := "00";
    signal clk_count : INTEGER := 0;
    signal debounce : INTEGER := 0;
    signal key_temp : STD_LOGIC_VECTOR(3 downto 0) := "0000";
    signal valid : STD_LOGIC := '0';
    constant debounce_limit : INTEGER := 200000;
begin
    process(clk)
    begin
```

```
if rising_edge(clk) then
    clk_count <= clk_count + 1;
    if clk_count = 50000 then
        clk_count <= 0;
        row_sel <= std_logic_vector(unsigned(row_sel) + 1);
    end if;

    case row_sel is
        when "00" => rows <= "1110";
        when "01" => rows <= "1101";
        when "10" => rows <= "1011";
        when others => rows <= "0111";
    end case;

    if debounce = 0 then
        if cols = "1110" or cols = "1101" or cols = "1011" or cols = "0111" then
            valid <= '1';
            case row_sel is
                when "00" =>
                    case cols is
                        when "1110" => key_temp <= "0001";
                        when "1101" => key_temp <= "0010";
                        when "1011" => key_temp <= "0011";
                        when "0111" => key_temp <= "1010";
                        when others => valid <= '0';
```

```
    end case;
when "01" =>
    case cols is
        when "1110" => key_temp <= "0100";
        when "1101" => key_temp <= "0101";
        when "1011" => key_temp <= "0110";
        when "0111" => key_temp <= "1011";
        when others => valid <= '0';
    end case;
when "10" =>
    case cols is
        when "1110" => key_temp <= "0111";
        when "1101" => key_temp <= "1000";
        when "1011" => key_temp <= "1001";
        when "0111" => key_temp <= "1100";
        when others => valid <= '0';
    end case;
when others =>
    case cols is
        when "1110" => key_temp <= "1110";
        when "1101" => key_temp <= "0000";
        when "1011" => key_temp <= "1111";
        when "0111" => key_temp <= "1101";
        when others => valid <= '0';
    end case;
```

```
                end case;
                debounce <= debounce + 1;
            else
                valid <= '0';
            end if;
        else
            debounce <= debounce + 1;
            if debounce > debounce_limit then
                debounce <= 0;
                valid <= '0';
            end if;
        end if;
    end if;
end process;

key <= key_temp;
key_valid <= valid;
end Behavioral;
```

servo_pwm.vhd

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

entity servo_pwm is
    Port (
        clk : in STD_LOGIC;
        position : in STD_LOGIC;
        pwm_out : out STD_LOGIC
    );
end servo_pwm;

architecture Behavioral of servo_pwm is
    signal counter : INTEGER := 0;
    signal pulse_width : INTEGER := 0;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if counter < 2000000 then
                counter <= counter + 1;
            else
                counter <= 0;
            end if;
```

```
            if position = '1' then
                pulse_width <= 200000;
            else
                pulse_width <= 100000;
            end if;

            if counter < pulse_width then
                pwm_out <= '1';
            else
                pwm_out <= '0';
            end if;
        end if;
    end process;
end Behavioral;
```

clk_divider.vhd

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

entity clk_divider is
    Port (
        clk : in STD_LOGIC;
        refresh_phase : out STD_LOGIC_VECTOR(1 downto 0)
    );
end clk_divider;

architecture Behavioral of clk_divider is
    signal count : unsigned(15 downto 0) := (others => '0');
begin
    process(clk)
    begin
        if rising_edge(clk) then
            count <= count + 1;
        end if;
    end process;

    refresh_phase <= std_logic_vector(count(15 downto 14));

end Behavioral;
```

display_driver.vhd

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

entity display_driver is
    Port (
        input_code : in STD_LOGIC_VECTOR(15 downto 0);
        entered_digits : in integer range 0 to 4;
        refresh_phase : in STD_LOGIC_VECTOR(1 downto 0);
        display_anodes : out STD_LOGIC_VECTOR(3 downto 0);
        display_segments : out STD_LOGIC_VECTOR(6 downto 0)
    );
end display_driver;

architecture Behavioral of display_driver is
    signal current_digit : STD_LOGIC_VECTOR(3 downto 0);
    signal active_digit_index : integer range 0 to 3;
    signal show_blank : boolean;
    signal segments_raw : STD_LOGIC_VECTOR(6 downto 0);

    component seg_decoder
        Port (
            digit_value : in STD_LOGIC_VECTOR(3 downto 0);
            segments : out STD_LOGIC_VECTOR(6 downto 0)
        );
```

```
    end component;
begin

    process(refresh_phase, input_code, entered_digits)
    begin
        show_blank <= false;

        case refresh_phase is
            when "00" =>
                current_digit <= input_code(15 downto 12);
                display_anodes <= "1110";
                active_digit_index <= 0;
            when "01" =>
                current_digit <= input_code(11 downto 8);
                display_anodes <= "1101";
                active_digit_index <= 1;
            when "10" =>
                current_digit <= input_code(7 downto 4);
                display_anodes <= "1011";
                active_digit_index <= 2;
            when others =>
                current_digit <= input_code(3 downto 0);
                display_anodes <= "0111";
                active_digit_index <= 3;
        end case;
```

```
        if active_digit_index >= entered_digits then
            show_blank <= true;
        end if;
    end process;

    decoder_inst: seg_decoder port map(
        digit_value => current_digit,
        segments => segments_raw
    );

    display_segments <= segments_raw when not show_blank else "1111111";

end Behavioral;
```

seg_decoder.vhd

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

entity seg_decoder is
    Port (
        digit_value : in  STD_LOGIC_VECTOR (3 downto 0);
        segments : out STD_LOGIC_VECTOR (6 downto 0)
    );
end seg_decoder;

architecture lookup of seg_decoder is
begin
    process(digit_value)
    begin
        case digit_value is
            when "0000" => segments <= "0000001";
            when "0001" => segments <= "1001111";
            when "0010" => segments <= "0010010";
            when "0011" => segments <= "0000110";
            when "0100" => segments <= "1001100";
            when "0101" => segments <= "0100100";
            when "0110" => segments <= "0100000";
            when "0111" => segments <= "0001111";
            when "1000" => segments <= "0000000";
```

```
            when "1001" => segments <= "0000100";
            when "1010" => segments <= "0000010";
            when "1011" => segments <= "1100000";
            when "1100" => segments <= "0110001";
            when "1101" => segments <= "1000010";
            when "1110" => segments <= "0110000";
            when "1111" => segments <= "0111000";
            when others => segments <= "1111111";
        end case;
    end process;
end lookup;
```

constraints.xdc

```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 10.0 [get_ports clk]

set_property PACKAGE_PIN U18 [get_ports verify_btn]
set_property IOSTANDARD LVCMOS33 [get_ports verify_btn]

set_property PACKAGE_PIN J1 [get_ports {rows[0]}]
set_property PACKAGE_PIN L2 [get_ports {rows[1]}]
set_property PACKAGE_PIN J2 [get_ports {rows[2]}]
set_property PACKAGE_PIN G2 [get_ports {rows[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rows[*]}]

set_property PACKAGE_PIN H1 [get_ports {cols[0]}]
set_property PACKAGE_PIN K2 [get_ports {cols[1]}]
set_property PACKAGE_PIN H2 [get_ports {cols[2]}]
set_property PACKAGE_PIN G3 [get_ports {cols[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {cols[*]}]

set_property PACKAGE_PIN K17 [get_ports {pwm_out}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwm_out}]

set_property PACKAGE_PIN A14 [get_ports {progress_leds[0]}]
set_property PACKAGE_PIN A16 [get_ports {progress_leds[1]}]
```

```
set_property PACKAGE_PIN B15 [get_ports {progress_leds[2]}]
set_property PACKAGE_PIN B16 [get_ports {progress_leds[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {progress_leds[*]}]

set_property PACKAGE_PIN W7 [get_ports {display_segments[6]}]
set_property PACKAGE_PIN W6 [get_ports {display_segments[5]}]
set_property PACKAGE_PIN U8 [get_ports {display_segments[4]}]
set_property PACKAGE_PIN V8 [get_ports {display_segments[3]}]
set_property PACKAGE_PIN U5 [get_ports {display_segments[2]}]
set_property PACKAGE_PIN V5 [get_ports {display_segments[1]}]
set_property PACKAGE_PIN U7 [get_ports {display_segments[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display_segments[*]}]

set_property PACKAGE_PIN W4 [get_ports {display_anodes[0]}]
set_property PACKAGE_PIN V4 [get_ports {display_anodes[1]}]
set_property PACKAGE_PIN U4 [get_ports {display_anodes[2]}]
set_property PACKAGE_PIN U2 [get_ports {display_anodes[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display_anodes[*]}]

set_property PACKAGE_PIN L17 [get_ports led0]
set_property PACKAGE_PIN M19 [get_ports led1]
set_property PACKAGE_PIN P17 [get_ports led2]
set_property IOSTANDARD LVCMOS33 [get_ports {led0}]
set_property IOSTANDARD LVCMOS33 [get_ports {led1}]
set_property IOSTANDARD LVCMOS33 [get_ports {led2}]
```

```
set_property PACKAGE_PIN R18 [get_ports buzzer]
set_property IOSTANDARD LVCMOS33 [get_ports buzzer]

set_property PACKAGE_PIN M18 [get_ports pir_in]
set_property IOSTANDARD LVCMOS33 [get_ports pir_in]

set_property PACKAGE_PIN N17 [get_ports pir_out]
set_property IOSTANDARD LVCMOS33 [get_ports pir_out]
```