15.05.2024 Emir Arda Bayer
22201832-4

Secure Keypad Locking System with Multi-Level Access Control

YouTube Link: https://youtu.be/ZUw9XAH6BxY

Purpose

The purpose of my project is to create a locking system that composes of a servo motor, a 4x3 keypad, two LEDs, one buzzer and the LEDs of the BASYS 3. The user has the ability to input an maximum 8 digit passcode via the keypad and lock anything to a desired location, meanwhile observing the state of inputs from the LEDs and the buzzer. There is a resetting capability that lets the user reinput the code, and an admin pass exists for the situations where there are multiple wrong attempts to unlock the servo motor. The system can be implemented to any industry that requires locking systems that require keypad entries.

Components

- BASYS 3
- VHDL on Vivado
- 4x3 Keypad (7 pins)
- Green LED
- Red LED
- Buzzer
- Servo Motor

Design Specifications

There are 2 inputs and 6 outputs in the VHDL code. The inputs are the clock and the 3 bits long vector “cols” that input the states of the 3 columns of the keypad. The 4 bits long “rows” output exists to turn each row of the keypad high separately for 10 ms for further matrix evaluation of the key presses. The 12 bits long “leds” output helps the developer or the user observe if any of the keys were pressed separately on different LEDs of the BASYS 3. There are then 1 bit long “ledg”, “ledr” and “buzzer” that turn on for 0.5 seconds to indicate keypad presses are inputted once. The last output is the 1 bit long “servo”, that sends a pwm signal to later change the degree the servo motor is turned to from zero degrees to ninety according to the state of the system.

There are multiple signals and constants within the architecture. Integer signals “currentrow”, “currentdigit”, “limit”, “wrongattempts” serve to keep certain values within the system. Currentrow indicates which of the rows of the keypad is selected to be high at the moment, currendigit implies which digit of the inputted passcode is expected to be next entered, limit expresses the 8 digit long limit of possible enterable passcodes, and wrongattempts carries the value 3 to later decrease and lock the system when the user enters 3 wrong codes.

Signal “entering” is of type std_logic, and decides if the passcode is enterable at the moment, as the system does not work before the star button is pressed to input codes. The 32 bit long logic vectors “passcode”, “inputpass” and “adminpass” serve the purpose of keeping and comparing inputted passcodes, or activate the admin feature. The signal “state” keeps the general state of the system, whether the system is locked, unlocked or the admin feature is on.

There are three counters of integer type in the code being “counter”, “counterb” and “counterservo”. The last one is for creating the PWM signal for the servo motor’s degree selection. The other two exist separately to serve as a debouncing and error correction system. At each change the digit of the high row, the system waits for the midpoint of the high state of this high row to check whether any of the inputs from the “cols” is high. The different feature of “counterb” is that it checks the “cols” at every half second to provide a basic debouncing system, the combination of the resistors in the analog circuit on the breadboard, the inside pull down resistors in the BASYS 3 and this system provide proper debouncing of inputs and prevent any unintended inputs.

The signals “servostate”, “pwm” exist to change the PWM output when the algorithm decides the degree of the servo should change. Then there are constants for the “counterservo” to change the state of the PWM output, being time limits called “degree90time”, “degree0time” and “oneperiod”.

Methodology and Results

The architecture consists of two clock processes. One process checks the state of “servostate” that can either be ‘0’ or ‘1’ at every rising edge of the clock and changes the degree of the servo motor by changing the structure of the PWM signal. In this process, counterservo increases between 0 and “oneperiod” values and as the algorithm checks whether the value reaches “degree90time” or “degree0time”, the signal “pwm” changes from high to low. At each period the counter resets and forms the 20 ms period and selected high values, 0.5 ms for 0 degrees and 1.5 ms for 90 degrees.

The other process has two inclusive if statements, one of these is for changing the high valued row to the next one. Here, the signal “counter” increments until 10 ms passes and changes the value of “currentrow” to the next one as it also turns one of the rows one, as “0001”, “0010”, “0100” and further. At the end of each 10 ms period the counter resets and the high valued row changes according to the saved “currentrow” value.

The other if statement has multiple if statements in between. Before the activation of any, it checks if the “counterb” has reached 50000000 cycles, meaning 500 ms to provide a way of debouncing and preventation of multiple inputs at one press. This means the user has around half a second to make one input, and the user can hold the button for more to input the same button more than once. Once this if statement states the desired counter value is reached, it resets the “counterb” and checks every single bit of “cols” to detect button presses. Once it detects an input from one of the columns, inner if statements check which of the rows was high at the moment of detection and makes changes in the system according to the signals, especially the “state” and “entering” signals that have to be changed in order to change other signals.

The steps that follow presses of numerical digits with star (*) and hashtag (#) buttons are different. The system turns on with the “entering”, “state”, “inputpass”, “limit”, “wrongattempts” as ‘0’. The system waits for activation by the star button and the initial input of it changes the “entering” signal to ‘1’, letting the numerical buttons to function to input passcode values.

When the “state” signal is ‘0’, it awaits an input passcode to be saved and for locking the servo motor. When the code is entered, the press of the star button activates the servo motor and changes the state to ‘1’, implying it is locked. It also changes “entering” to ‘0’, closing the system for numerical inputs and locking the system for opening later. It also resets every signal for step functions being “limit” and “currentdigit” to ‘0’, if it is not resetting the whole system at the moment of the opening of the lock and the reset of every changed signal. The input of star button blinks the green external LED.

The hashtag button serves the purpose of resetting input codes. For example, if the user enters a wrong code while trying to open the lock and gets interrupted with the idea that the was indeed wrong, they have the opportunity to use this button to reset the “inputpass” by not incrementing the “wrongattempts” signal. The input of it resets “limit” and “currentdigit” signals. Additionally, the input of it blinks the red external LED.

The numerical buttons light up the previously selected LED on the servo for the inputted button, light up the green LED and make changes to the input passcode according to the “limit”, “state” and “entering” signals. If “entering” is ‘1’, the system checks the value of the state and changes the value of “passcode” if the phase is the initial unlocked state, and increments “currentdigit” by 4 at every input since every numerical input takes 4 bits in the 32 bit password signals. If it is the locked state, then “inputpass” is changed. For example, if the digit 9 is pressed and this is the 3rd digit of the input code, then the bits 11 to 8 become “1001”, “currentdigit” becomes 15, “limit” becomes 3, meanwhile the green LED and the ninth LED on BASYS 3 blinks with the buzzer.

If the “wrongattempts” become 3, then “passcode” is assigned to become “adminpass” that is selected within the code and cannot be observed in physical uses. The system changes the state to ‘3’ and expects the “inputpass” to be the same as “passcode”, deleting the previous inputted and saved code.

The outputs of the system, the states of the servo motor and the physical design are as in the following figures.

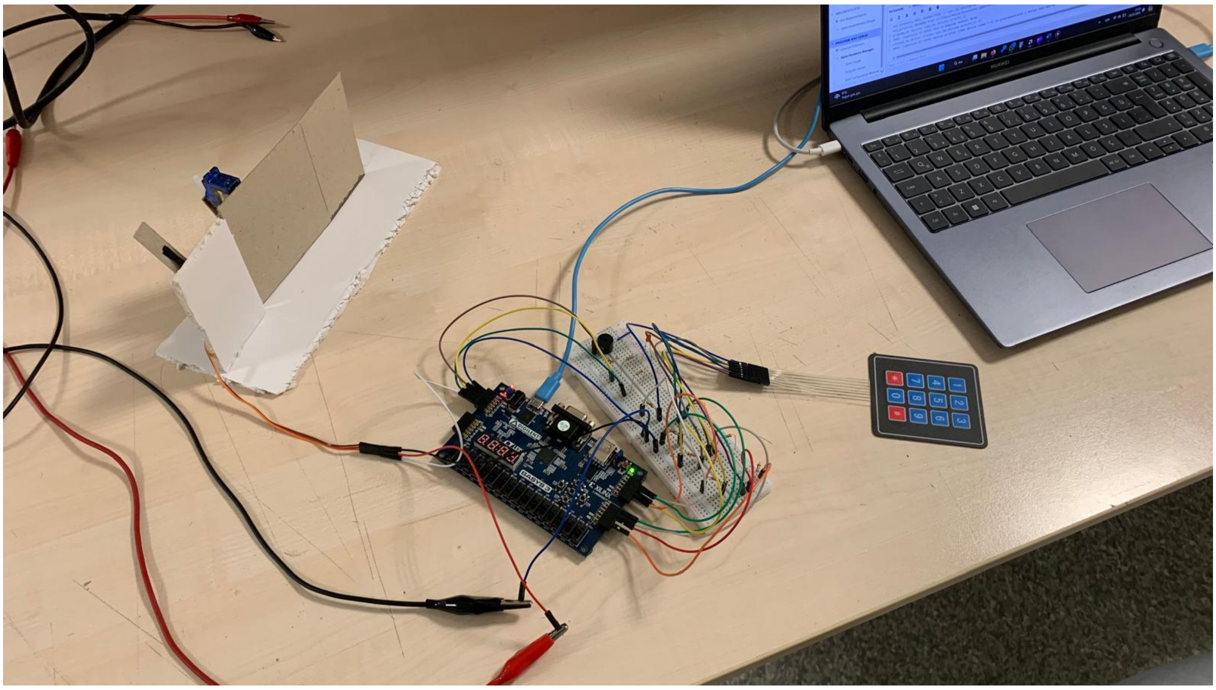

Figure 1: Physical Design of the Project

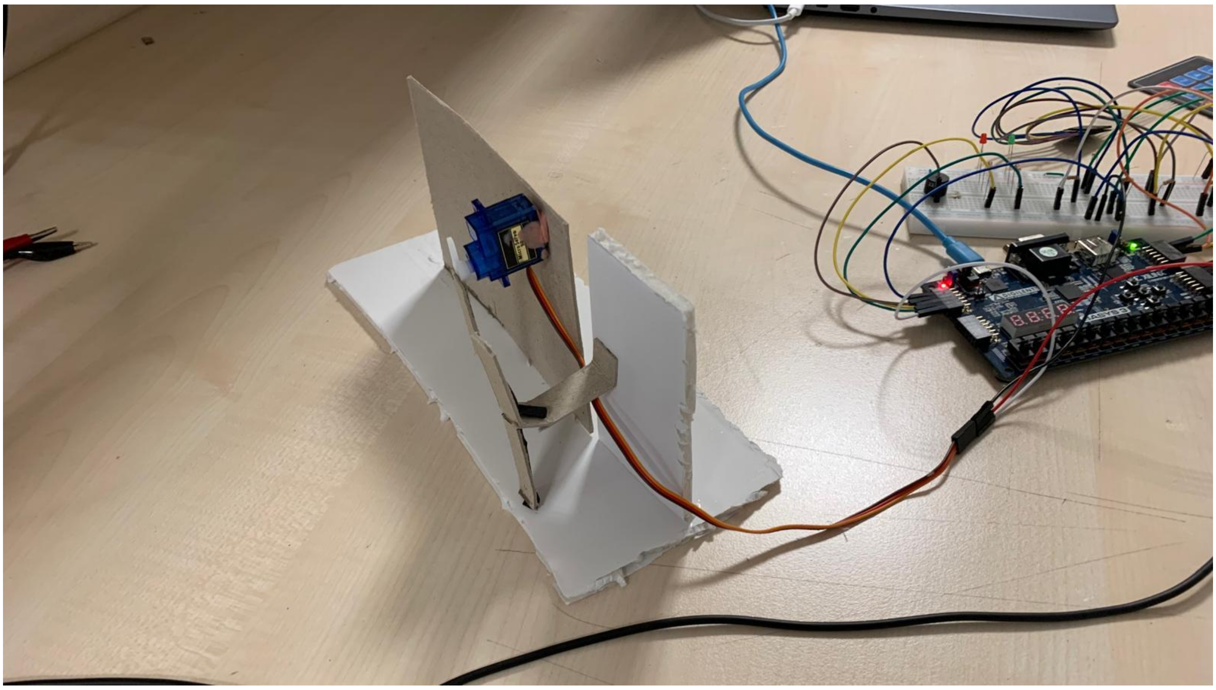

Figure 2: Implementation of the Servo Motor, Unlocked 0 Degrees State

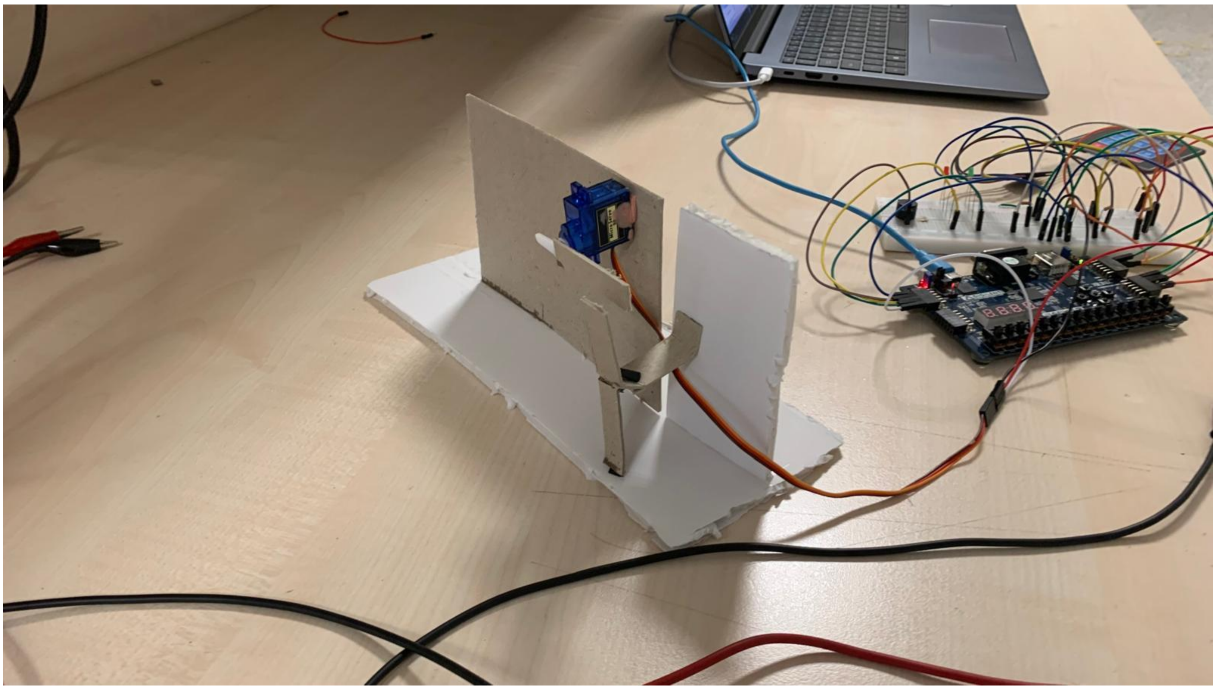

Figure 3: Servo Motor’s Locked State, 90 Degrees

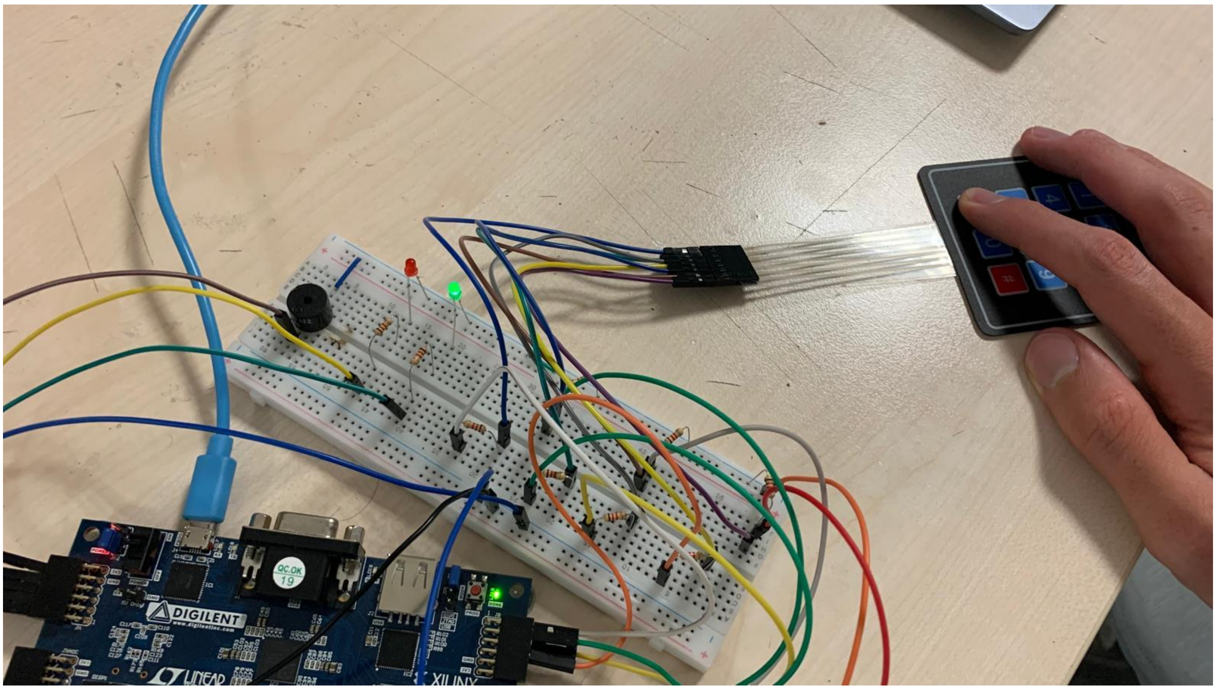

Figure 4: Star (*) Button’s Change of Digital Signals Indicated by Input Declaration on the Green LED

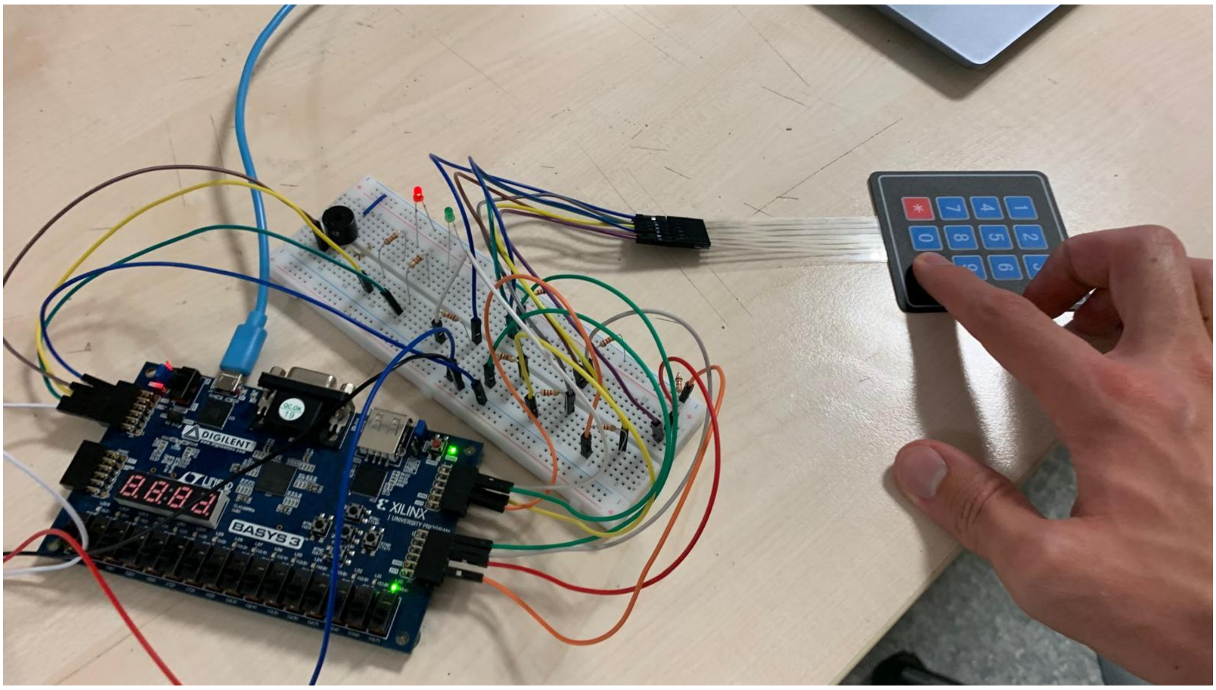

Figure 5: Blink of the Red LED due to an Input from Hashtag (#) Button, Indicating Change of Signals

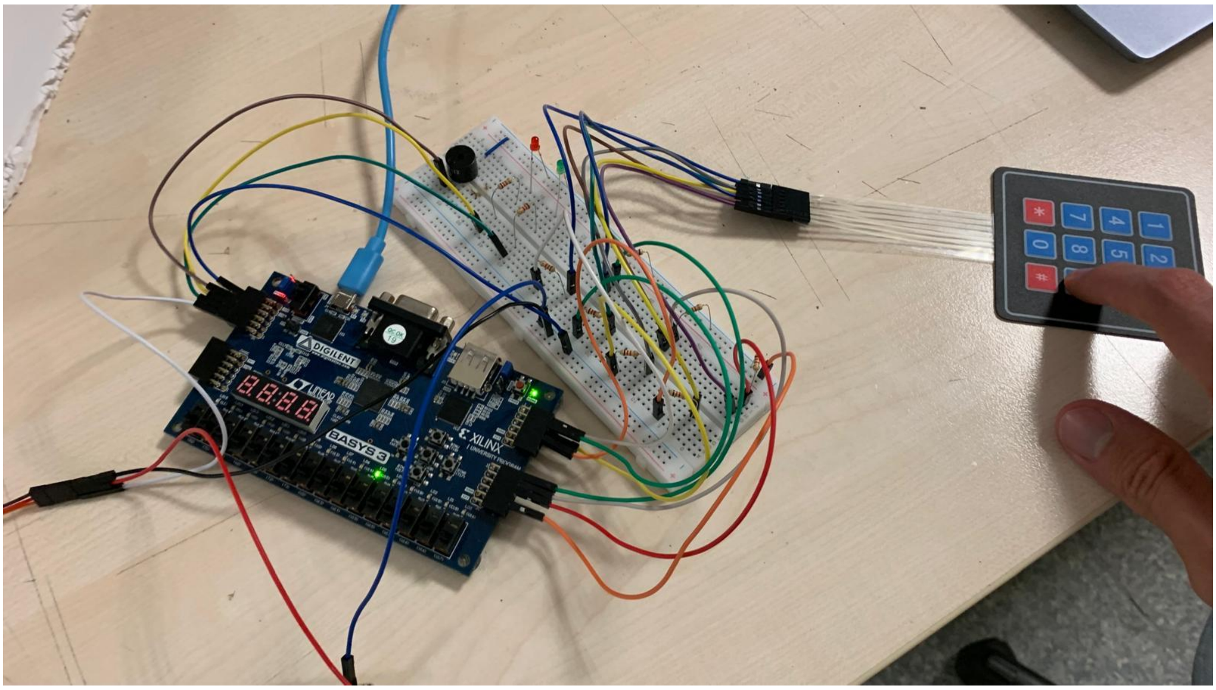

Figure 6: Input of Numerical Button does not Affect Inner Signals, Green LED off, 9th BASYS 3 LED is on

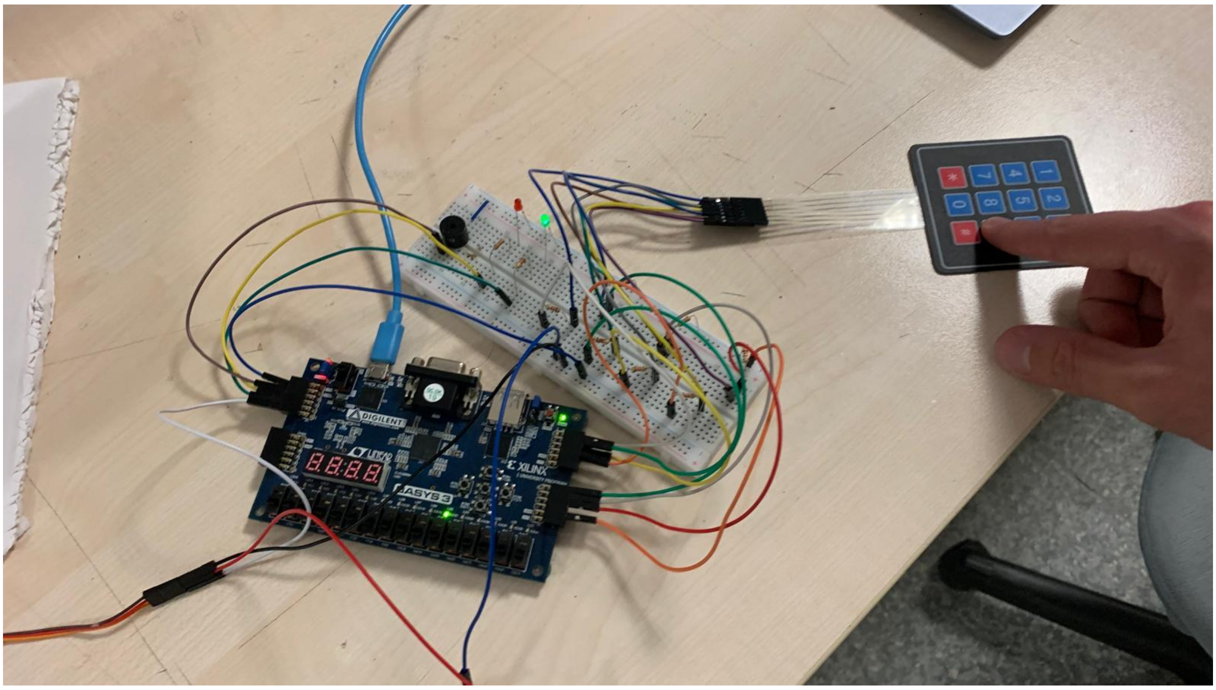

Figure 7: Entering Mode is On, Numerical Input Blinks Green LED and Indication of Change of Inner Signals

Conclusion

The purpose of this project was to design a locking system using a servo motor, keypad, LEDs and a buzzer, including features such as an admin access and resetting. Multiple concepts about digital circuit design was made use of while making use of this project, such as counters, debouncing, duty cycles, PWM signals, using keypads and similar circuit element. I had to make lengthy research about the keypad and servo motor, had to critically think about implementing them to my own project and had to cope with errors I faced along the way. I had to understand how I can properly debounce the keypad inputs, make it possible to input only once at every button press, and find ways to implement this to my project that saves data through signals. I had to understand how servo motors work, what PWM signals are, and in what ways I can change the degree of my servo motor. I had to understand how to blink LEDs at desired lengths of time, along with using buzzers. I had to understand how to save steps of the algorithm through the use of several kinds of signals and constants. While preparing this project, I faced multiple problems with the keypad and the servo motor. The keypad was giving undesired inputs, to the extent that the demo Vivado project I created for only the keypad was giving high inputs constantly. I had to find ways to debounce, one being setting counter limits to check the input once every 500 ms, other being placing resistors on the analog circuit on the breadboard. Later, I had to understand how PWM signals work and realized that certain duty cycles exist that choose the degrees of the motor. I made another mistake of connecting the ground of the servo motor directly to the power supply, creating two proposed grounds in the project, causing potential risks. I had to connect the ground of the supply to the ground on the breadboard. This project was a deep and enlightening experience for an electrical and electronics engineering student, as I had to not only learn the concepts of the course and extra details required for my project's components, but also had to think like an engineer, find ways to solve problems I faced along the way of implementing and designing an engineering project.

Appendices

top.vhd

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


entity top is
    Port (
        clk : in std_logic;
        cols : in std_logic_vector(2 downto 0);
        rows : out std_logic_vector(3 downto 0);
        leds : out std_logic_vector(11 downto 0);
        ledg: out std_logic;
        ledr: out std_logic;
        buzzer: out std_logic;
        servo : out STD_LOGIC
    );
end top;

architecture Behavioral of top is

--keypad signals
signal currentrow: integer:=0;
signal counter : integer:=0;
signal counterb : integer:=0;
--

--servo signals
```

```
signal servostate: std_logic := '0';
signal pwm: std_logic := '0';
signal counterservo: integer:= 0;
constant degree90time: natural := 150000;
constant degree0time: natural := 50000;
constant oneperiod: natural := 2000000;
--

--signals for the algorithm
signal entering: std_logic:='0';
signal passcode: std_logic_vector(31 downto 0):="00000000000000000000000000000000";
signal inputpass: std_logic_vector(31 downto 0):="00000000000000000000000000000000";
signal adminpass: std_logic_vector(31 downto 0):="00000000000000000000000000000000";

signal state: std_logic:='0';
signal current_digit: integer:=3;
signal limit: integer:=0;
signal wrongattempts: integer:=0;
--



begin




process (clk) begin
    if rising_edge (clk) then
        if servostate = '1' then
            if counterservo < degree90time then
```

```
          counterservo <= counterservo + 1;
          pwm <= '1';
        elsif counterservo = degree90time then
          pwm <= '0';
          counterservo <= counterservo + 1;
        elsif degree90time < counterservo and counterservo < oneperiod then
          counterservo <= counterservo + 1;
        else
          counterservo <= 0;
        end if;
      else
        if counterservo < degree0time then
          counterservo <= counterservo + 1;
          pwm <= '1';
        elsif counterservo = degree0time then
          pwm <= '0';
          counterservo <= counterservo + 1;
        elsif degree0time < counterservo and counterservo < oneperiod then
          counterservo <= counterservo + 1;
        else
          counterservo <= 0;
        end if;
      end if;
    end if;
end process;
```

```
process (clk) begin
    if rising_edge(clk) then
        if counter = 999999 then
            counter <= 0;
            if currentrow = 0 then
                rows <= "0001";
                currentrow <= 1;
            elsif currentrow = 1 then
                rows <= "0010";
                currentrow <= 2;
            elsif currentrow = 2 then
                rows <= "0100";
                currentrow <= 3;
            elsif currentrow = 3 then
                rows <= "1000";
                currentrow <= 0;
            end if;
        else
            counter <= counter + 1;
        end if;




        if counterb = 50000000 then
            counterb <= 0;
            if cols(0) = '1' then
```

```
if currentrow = 1 then
    leds <= "000000000010";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "0001";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "0001";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;


elsif currentrow = 2 then
    leds <= "000000010000";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "0100";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
```

```
            else
                inputpass(current_digit downto (current_digit-3)) <= "0100";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;


elsif currentrow = 3 then
    leds <= "000010000000";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "0111";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "0111";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;
```

```
elsif currentrow = 0 then -- GREEN BUTTON
    leds <= "010000000000";
    ledg <= '1';
    buzzer <= '1';

    if entering = '0' then
        entering <= '1';

    elsif entering = '1' and limit >= 0 then
        entering <= '0';
        if state = '0' then
            servostate <= '1';
            entering <= '0';
            limit <= 0;
            current_digit <= 3;
            state <= '1';
        else
            if inputpass = passcode then
                servostate <= '0';
                entering <= '0';
                limit <= 0;
                current_digit <= 3;
                state <= '0';
                wrongattempts <= 0;
                inputpass <= "00000000000000000000000000000000";
                passcode <= "00000000000000000000000000000000";
            elsif wrongattempts <= 3 then
                wrongattempts <= wrongattempts + 1;
```

```
                inputpass <= "00000000000000000000000000000000";
                limit <= 0;
                current_digit <= 3;
            else
                passcode <= adminpass;
                limit <= 0;
                current_digit <= 3;
            end if;
        end if;
    end if;

end if;







elsif cols(1) = '1' then
    if currentrow = 1 then
        leds <= "000000000100";
        if entering = '1' then
            ledg <= '1';
            buzzer <= '1';
            if limit <= 3 then
                if state = '0' then
                    passcode(current_digit downto (current_digit-3)) <= "0010";
                    current_digit <= current_digit + 4;
```

```
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "0010";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;


elsif currentrow = 2 then
    leds <= "000000100000";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "0101";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "0101";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;
```

```
elsif currentrow = 3 then
    leds <= "000100000000";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "1000";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "1000";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;


elsif currentrow = 0 then
    leds <= "000000000001";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "0000";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
```

```
                else
                    inputpass(current_digit downto (current_digit-3)) <= "0000";
                    current_digit <= current_digit + 4;
                    limit <= limit + 1;
                end if;
            end if;
        end if;
    end if;



elsif cols(2) = '1' then
    if currentrow = 1 then
        leds <= "000000001000";
        if entering = '1' then
            ledg <= '1';
            buzzer <= '1';
            if limit <= 3 then
                if state = '0' then
                    passcode(current_digit downto (current_digit-3)) <= "0011";
                    current_digit <= current_digit + 4;
                    limit <= limit + 1;
                else
                    inputpass(current_digit downto (current_digit-3)) <= "0011";
                    current_digit <= current_digit + 4;
                    limit <= limit + 1;
                end if;
            end if;
        end if;
```

```
elsif currentrow = 2 then
    leds <= "000001000000";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "0110";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "0110";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;


elsif currentrow = 3 then
    leds <= "001000000000";
    if entering = '1' then
        ledg <= '1';
        buzzer <= '1';
        if limit <= 3 then
            if state = '0' then
                passcode(current_digit downto (current_digit-3)) <= "1001";
```

```
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            else
                inputpass(current_digit downto (current_digit-3)) <= "1001";
                current_digit <= current_digit + 4;
                limit <= limit + 1;
            end if;
        end if;
    end if;


elsif currentrow = 0 then -- RED BUTTON
    leds <= "100000000000";
    ledr <= '1';
    buzzer <= '1';

    if state = '0' then
        passcode <= "00000000000000000000000000000000";
        limit <= 0;
        current_digit <= 3;
    else
        inputpass <= "00000000000000000000000000000000";
        limit <= 0;
        current_digit <= 3;
    end if;
end if;
```

```
            else
                leds <= "000000000000";
                ledg <= '0';
                ledr <= '0';
                buzzer <= '0';
            end if;
        else
            counterb <= counterb + 1;
        end if;




    end if;
end process;
servo <= pwm;  -- Assign output signal to servo control
end Behavioral;
```

pins.xdc

set_property PACKAGE_PIN W5 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports cols[0]]

set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports cols[1]]

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports cols[2]]

set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS33 } [get_ports rows[0]]

set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS33 } [get_ports rows[2]]

set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports rows[1]]

set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports rows[3]]

set_property -dict { PACKAGE_PIN U3   IOSTANDARD LVCMOS33 } [get_ports leds[0]]

set_property -dict { PACKAGE_PIN W3   IOSTANDARD LVCMOS33 } [get_ports leds[1]]

set_property -dict { PACKAGE_PIN V3   IOSTANDARD LVCMOS33 } [get_ports leds[2]]

set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports leds[3]]

set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports leds[4]]

set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports leds[5]]

set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports leds[6]]

set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports leds[7]]

set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports leds[8]]

set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports leds[9]]

set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports leds[10]] set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports leds[11]]

set_property -dict { PACKAGE_PIN J2   IOSTANDARD LVCMOS33 } [get_ports ledr]

set_property -dict { PACKAGE_PIN L2   IOSTANDARD LVCMOS33 } [get_ports ledg]

set_property -dict { PACKAGE_PIN G2   IOSTANDARD LVCMOS33 } [get_ports buzzer]

set_property -dict { PACKAGE_PIN J1   IOSTANDARD LVCMOS33 } [get_ports servo]