AUTOMATIC PASSWORD DOOR

Bilkent University EEE102 Term Project

Emir Batuhan Özgenç
EEE102-2

1 - Video

https://youtu.be/zDVa0HV8Ido

2 - Purpose

In this project, I aim to implement a 4x4 keypad, 2 servo motors and an IR sensor with BASYS 3 to create an automatic door with password and motion input modes.

3 - Methodology

My project has 3 separate components that will be connected to each other.

- At first, I need to make a door frame with 2 servo motors and operate them using the BASYS3. Since the motors can be powered by 3.3V, V_{cc} and ground outputs of the BASYS3 can be used to do so. To move the servo motors, I need to create the waveforms required using the BASYS 3.
- Then I need to implement a 4x4 keypad and make the BASYS 3 record the password entered on the keypad and compare it to the true password, which is coded in the machine.
- After implementing the keypad, I need to connect the IR sensor to the system to operate the motion input mode in which the door will open when there is an obstacle nearby.

To switch between motion and password modes, I need to create a 2 to 1 multiplexer and select the modes using a switch.

4 - Design Specifications

The overall design will have 4 components:

- A clock for the whole system
- A keypad module for password input
- A servo module to control the motors
- An IR sensor for object detection mode
- A multiplexer to select input mode for the door

4.1 - Clock Module

For the clock of the overall system, I used the clocking wizard IP of Vivado and created a 5 MHz clock using the internal 100 MHz clock of the BASYS 3. The clock will be used for timing purposes in the other components like keypad and servo modules.

4.2 – Keypad Module

The keypad module will be used for the password input mode of the door, and it will be connected to the 4x4 Keypad using the PMod connectors of the BASYS 3.

The keypad has a 4x4 matrix connection in order to reduce the required number of ports. It has 8 inputs, 4 for rows and 4 for columns (Figure 4.2.1). When a button is pressed, the corresponding row and columns are connected to each other. We can determine the connection by sending high signals to each row one at a time and detect the high signals from the column connections. When there is one, we can use it and the data for which row was assigned high at that moment to determine the key pressed.

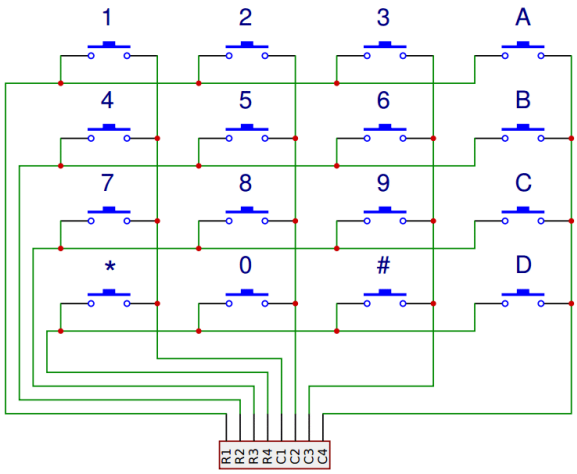

Figure 4.2.1: Connections of a 4x4 keypad

To determine the row output of the BASYS 3, I will use a keypad clock and counter module. (Figure 4.2.4) In the module, I need a 2-bit counter for the keypad which will increment every positive clock edge. Then, this counter will be decoded into one-hot data so that every row is assigned one at a time. However, the clock fort his scenario must be significantly slower than the 5 MHz clock which is the possible minimum of the Clocking Wizard IP. Because of this, I need to set up a counter and create a slower clock (like 50 Hz). Also, the slow clock for this module should be stopped and outputs should be stabilized when a key is pressed. This makes the data input from the keypad much more stable and removes the possibility of reading the same digit more than once on a single press.

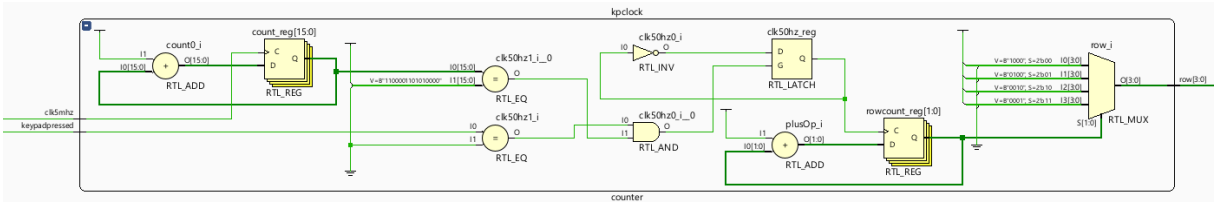

Figure 4.2.4: RTL schematic of the keypad clock

To record the column input on the BASYS 3 and read the digit pressed, I will use an encoder module which will take the current row and column as inputs and give a digit as its output. (Figure 4.2.5) The characters will be encoded as 5-bit data with the most significant bit representing if a key is pressed so that other modules like the password module will be able to determine whether a key is pressed or not and record it accordingly. The remaining 4 bits of the data will represent the data from 0 to 15 where numbers will be represented by their binary values, letters from A to D will be represented as the binary equivalent of their hexadecimal values and the characters # and * will be assumed as hexadecimal E and F respectively.

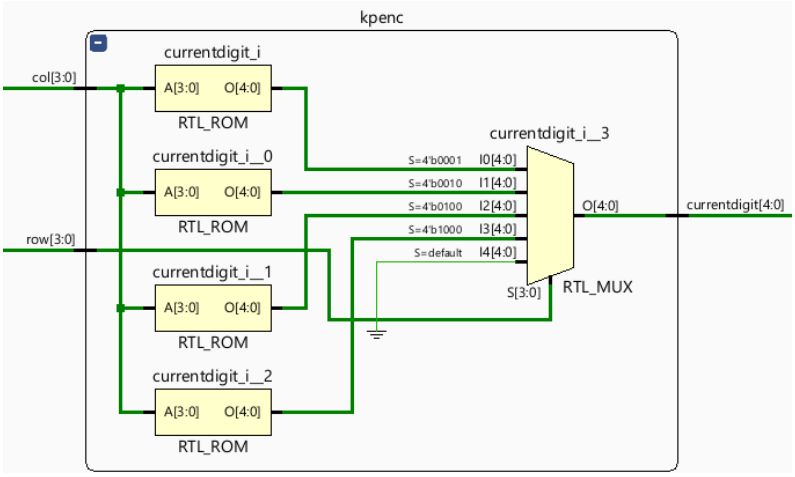

Figure 4.2.5: RTL schematic of the keypad encoder

These designs will work for a keypad but there is a problem of noise in the inputs of the BASYS 3 and may cause unwanted inputs. To minimise the noise and stabilise the data, I need to have a debouncer module which will count the durations of high and low inputs for a digit and decide whether that digit should be assigned high or low by comparing the counters. (Figure 4.2.6) This debouncer module will be implemented on every bit of the digit data. Since digit data is has a length of 5 bits, 5 copies of this module will be needed.

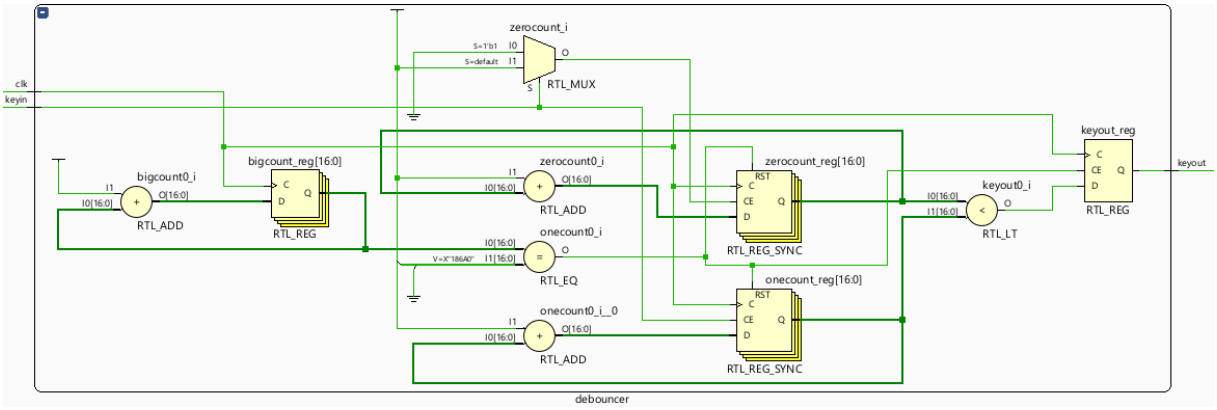

Figure 4.2.6: RTL schematic of the debouncer

After having a stable data input from the keypad, I need to create a password module such that it will compare whether the sequence entered on the keypad is the same as the correct password, which is coded into the machine as “021A”. (Figure 4.2.7) The correct password will be stored in a 20-bit constant which is the combination of the 4 characters in the password in the same format as the data is encoded by the keypad encoder module.

The password entered by the user will be recorded as such: When the first bit of the encoded character goes high, the system will record it into the leas significant 5 bits of a 20-bit sequence representing the entered password. However, if a digit is already entered, (which will be determined by the 4th bit) the data will be stored in the next 5 bits of the same 20-bit sequence.

If all 20 bits have a data input (bits 4, 9, 14 and 19 are high) the password will be compared with the correct password constant and reset. If passwords match, an output will be assigned high. To make it low again, a new 4-digit sequence must be entered. However, the last digit entered in the password (A) will be stored first in the sequence since the module assigns the

least significant bits first. Because of this, the correct password must be stored in the reverse format as “A120” still using the same data format used by the keypad encoder.

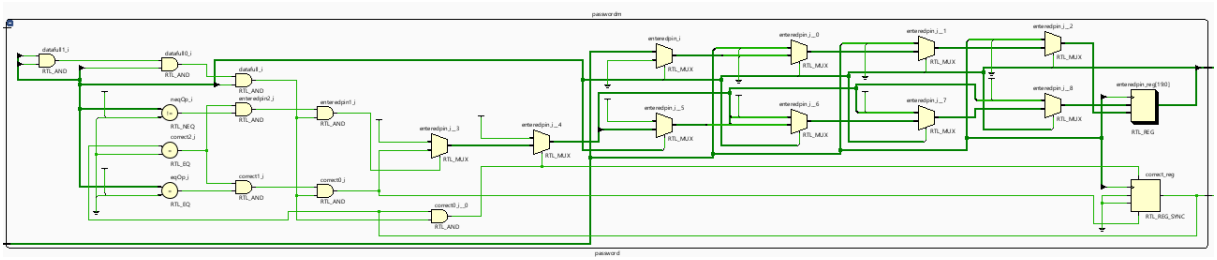

Figure 4.2.7: RTL schematic of the password module

When combining the components described above, the keypad module is completed. This module takes the 5 MHz clock and column data as inputs and gives a correct password signal and row connections of the keypad as outputs. For debugging purposes, the encoded digit entered at the moment and data for which characters of the 4-digit password sequence are full (bits 4, 9, 14 and 19 of the current password signal) will be displayed at LEDs of the BASYS 3.

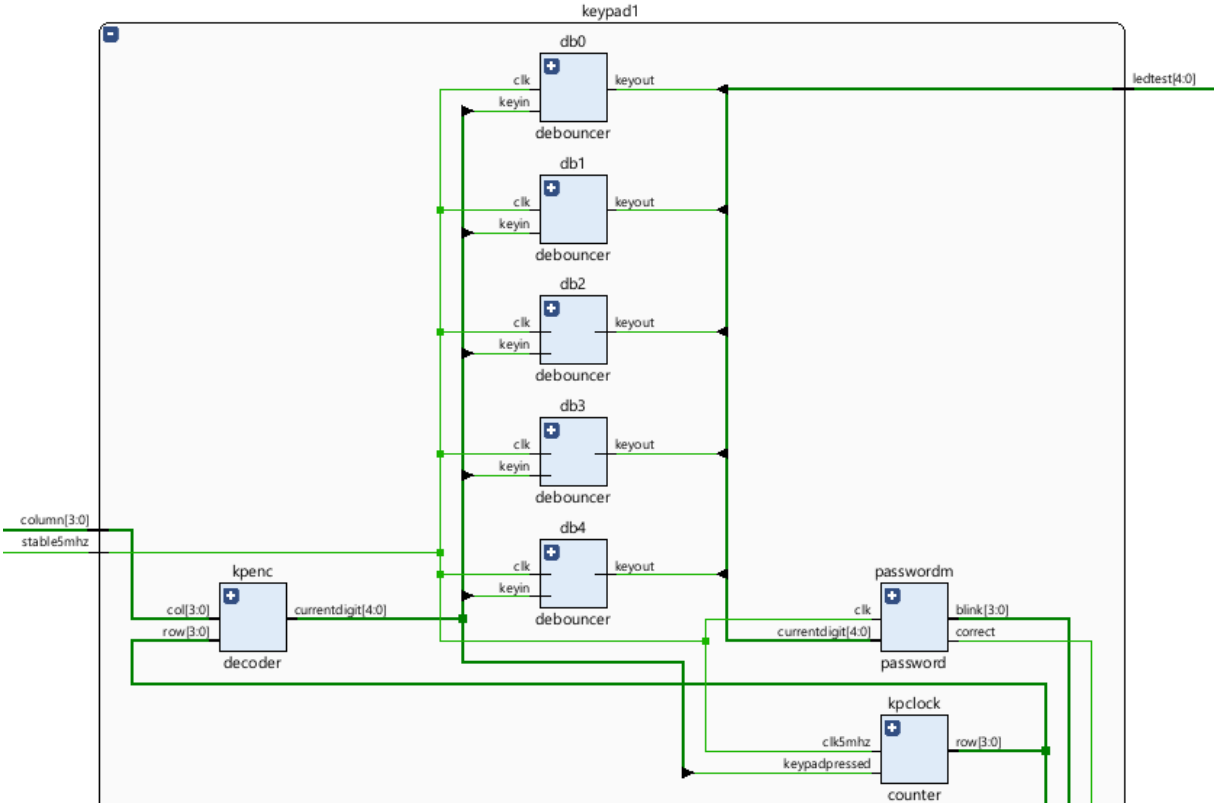

Figure 4.2.8: RTL schematic of the keypad module

4.3 – Servo Controller Module

To control the servo motors of the door, I will implement a servo controller module. The servo motors run using a 20 ms PWM cycle with the first 1-2 ms high (Figure 4.3.1). The duration of the high signal determines the position of the servo. Since I only need 2 modes for servos, I will use the 1 ms high and 2 ms high options for both servos controlling the left and right sides of the door.

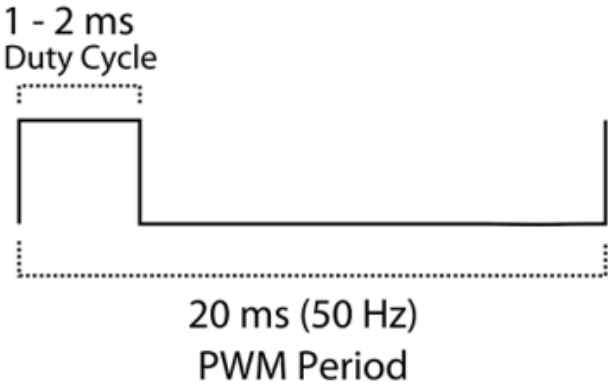

Figure 4.3.1: PWM period of a servo motor

For timing purposes, I will create a counter which increments every clock cycle and assign the outputs controlling servos according to the counter and dooropen signal. Since the servos are located at left and right sides of the door, I need to assign 2 signals so that they can turn in opposite directions to open and close the door. When door is opened or closed one must turn clockwise and the other must turn counterclockwise for the hinges of the turn to be on the same side when door is opened.

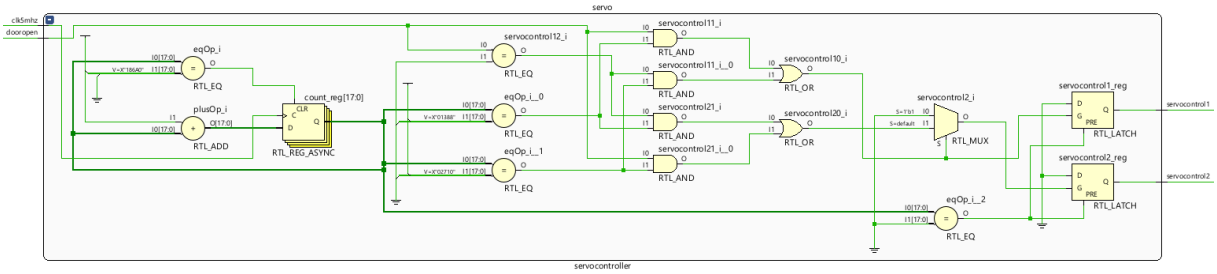

Figure 4.3.2: Servo controller module

4.4 – IR Sensor

The IR sensor provides low voltage when there is an obstacle nearby and high voltage when there is not. Its distance sensitivity can be set using the potentiometer on the device. However, there is a problem with the sensor: voltage provided is relatively low (approximately 2V) compared to the 3.3 V of the BASYS 3. To boost the signal as well as negating it, I will use a MOSFET transistor whose gate input will be connected to the data coming from IR sensor, drain input to the multiplexer and source input to the power source.

4.5 – Multiplexer

To connect two signals (coming from the keypad module and the IR sensor) to the door, I need to have a 2 to 1 multiplexer to determine which signal will be assigned to the door (input of servo controller module). To determine the input mode, I will use a swich. When the switch is low, the servo controller will be connected to the keypad and when the swich is high, it will be connected to the IR sensor. For this project, instead of implementing the multiplexer on the BASYS 3, I implemented is using an integrated circuit. This is because of the fragility of the other modules (when the multiplexer is added to the circuit, the keypad module malfunctions even if there is no change about it). The multiplexer I used is a

DM74LS153 model “dual 1-of-4 Line Data Selectors/Multiplexers” (Figure 4.5.1). I converted it to a single 2 to 1 multiplexer by applying low to strobe 2 (to only activate 2nd multiplexer) and applying high to select B (to select between data inputs 2 and 3). The data input 2 will be connected to the output of the keypad module and data input 3 will be connected to the output of the MOSFET connected to the IR sensor. The output of the multiplexer will be sent to BASYS 3 again in order to drive the servo motors controlling the door.

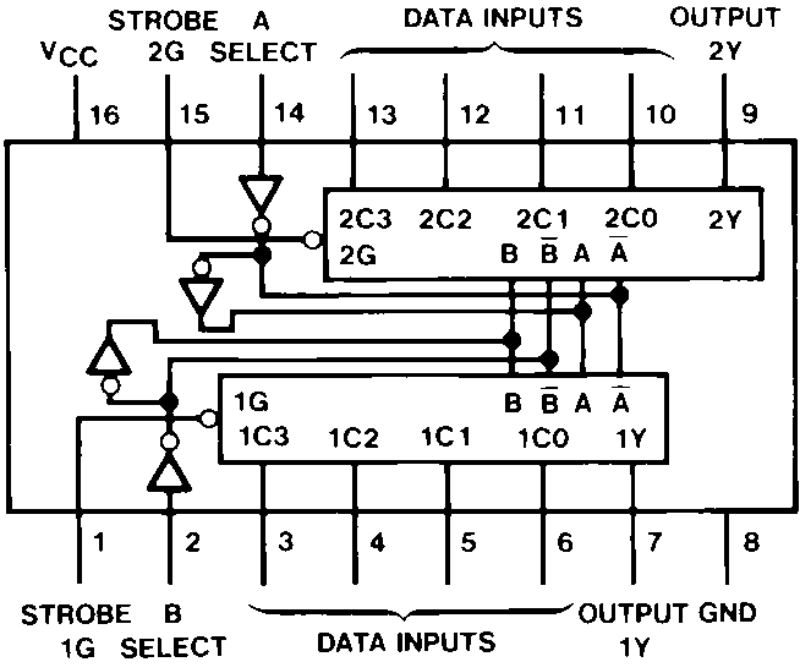

Figure 4.5.1: Connections of the multiplexer

4.6 – The whole circuit

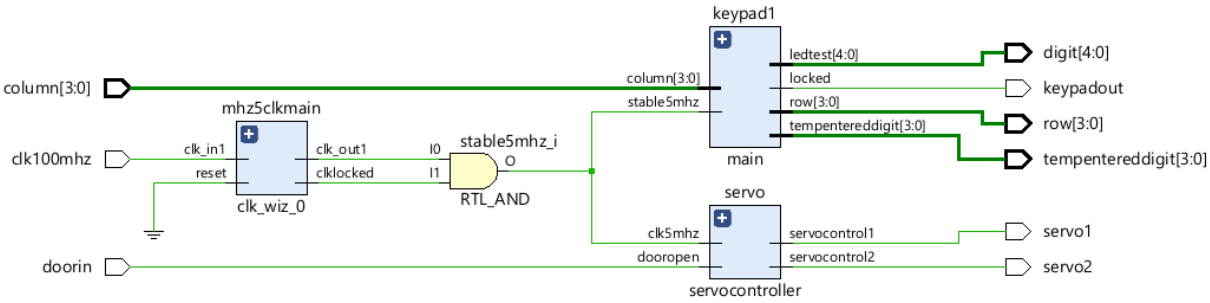

Figure 4.6.1: RTL Schematic of the circuit (modules applied in BASYS 3)

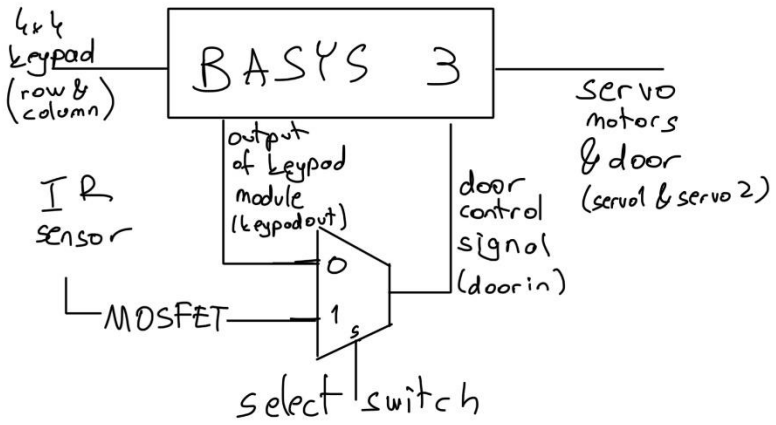

Figure 4.6.2: Schematic of the external circuit

The whole circuit is a combination of logic networks coded in BASYS 3 and external components like IR sensor, servo motors, 4x4 keypad, a MOSFET and a multiplexer. The outputs on the BASYS 3 “tempentereddigit” and “digit” show the password input back and serve for debugging purposes.

5 - Results

5.1 – Simulation

After designing the whole circuit, I tested it using a simulation (Figure 5.1.1). As expected, the row outputs switch from 0b”1000” to 0b”0100”, 0b”0010” and “0b0001” successfully. When relevant column input has provided by the simulator, I observed that digits were stored in the system as the output on *tempentereddigit* change from 0b”0000” to “0b1000” 0b”1100” 0b”1110” 0b”1111” and back to 0b”0000” when there is a column input different than 0b“0000”. (Figure 5.1.2) Furthermore, the LEDs also show the digit pressed successfully. Finally, when the whole sequence is entered correctly, the output *keypadout* went high as expected. (Figure 5.1.3)

Since the servos have a different input, I needed to stimulate that input in another process during the simulation. When I did so, I observed that the waveforms for servo motors run correctly as cycles of 20 ms with 2 or 1 ms high periods. When the input for the servos is toggled, the high periods of the output are exchanged as well, enabling the door to open and close.

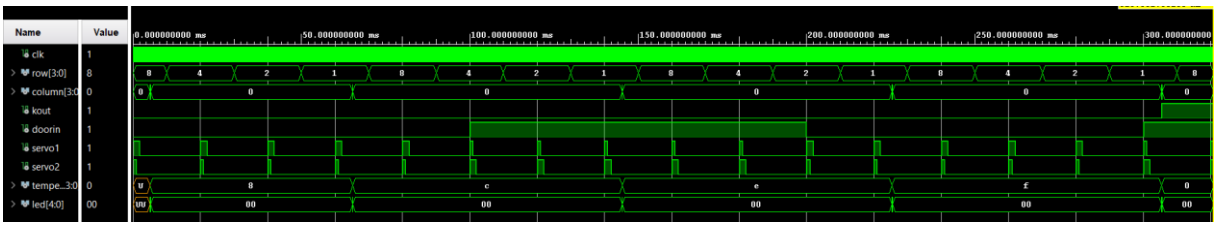

Figure 5.1.1: The simulation

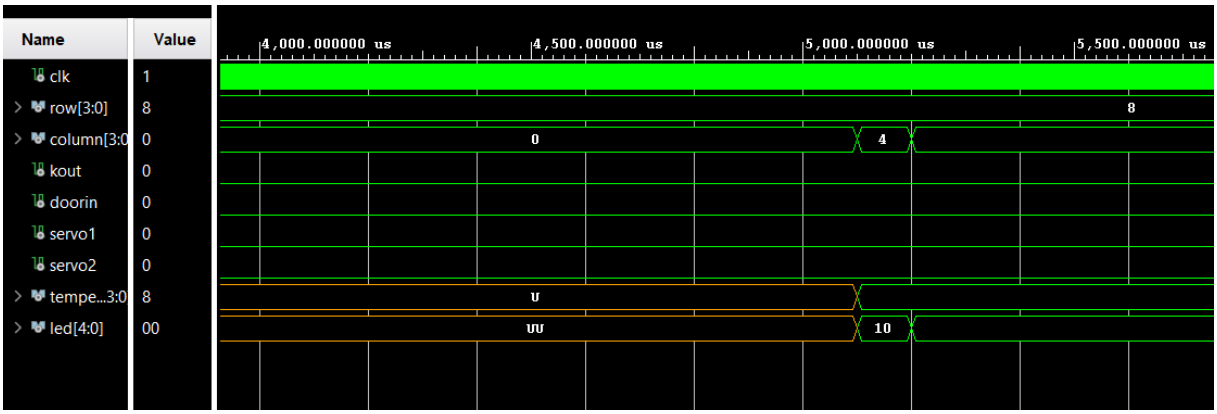

Figure 5.1.2: The simulation (beginning zoomed in) (0 pressed on keypad)

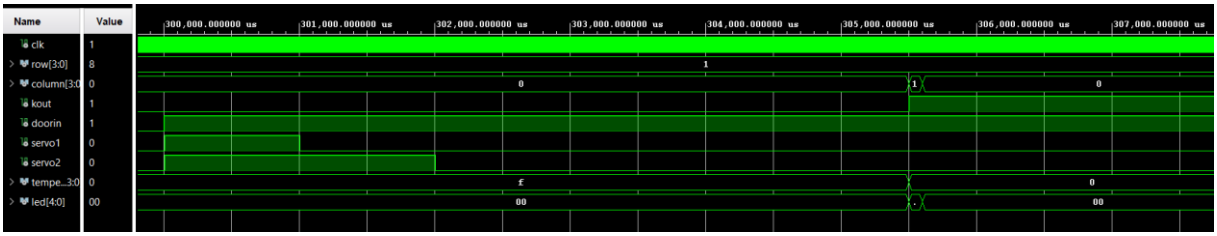

Figure 5.1.3: The simulation (end zoomed in) (kout goes high)

5.2 – Implementation on BASYS 3 and external components

While implementing the project, I used a breadboard to connect the MOSFET, multiplexer, swich and IR sensor to each other and BASYS 3. (Figure 5.2.1) The breadboard (Figure 5.2.2), keypad and servo motors (Figure 5.2.3) are directly connected to the PMod connectors of the BASYS 3. (Figure 5.2.4) However, there was a problem with the *keypadout* output that when I assigned it to the PMod connectors, it would not provide enough signal. Because of this, I assigned it to an LED and connected a wire to the anode of the LED. Other end of that wire is going to one of the data inputs of the multiplexer, with the other data input connected to the MOSFET receiving data from IR sensor and negating the data as well as boosting the signal.

On the breadboard, I used a dip switch which is connected to the select signal of the multiplexer in order to change between motion input and password input mode. Also, the 3 LEDs on the breadboard represent whether the door is open, whether there is an input from the keypad module and whether there is an input from the IR sensor respectively (top to bottom).

As expected, the door opens automatically when the correct password has entered on the keypad for password input mode (Figure 5.2.5) and when there is an object nearby for the object detection input mode. (Figure 5.2.6)

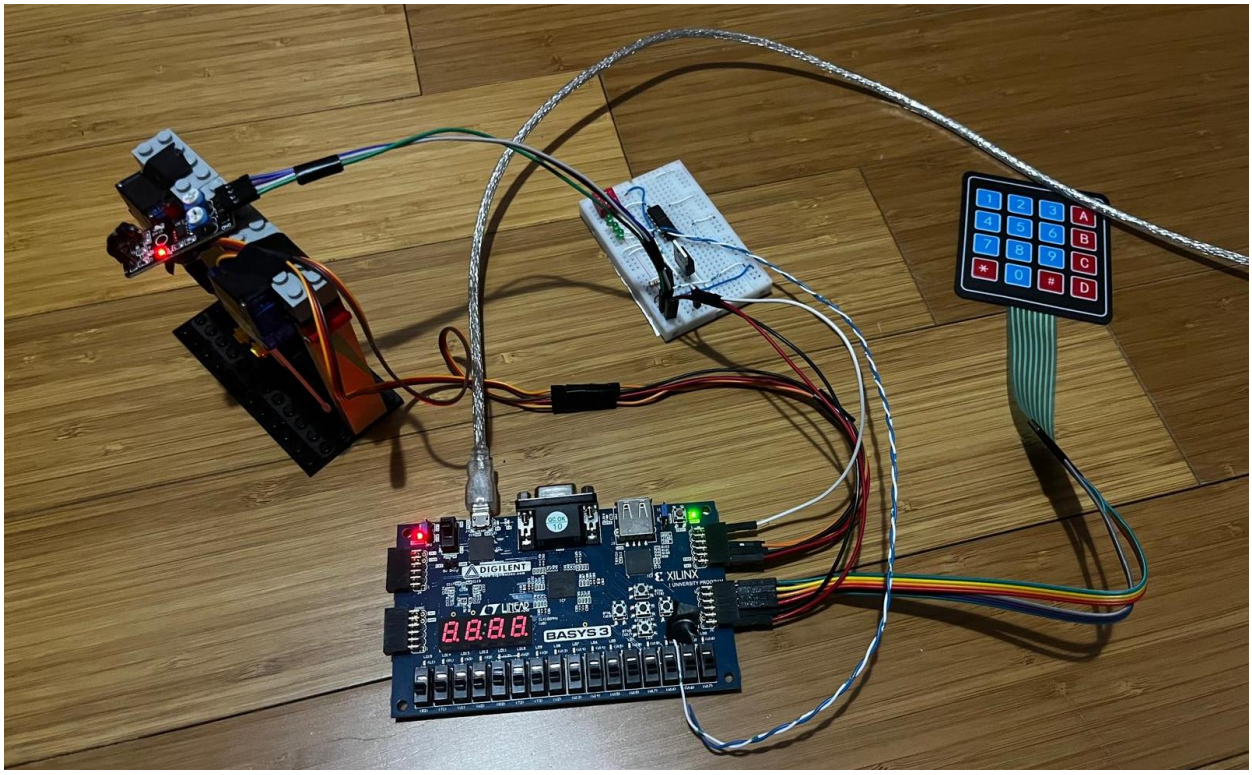

Figure 5.2.1: The whole design implemented

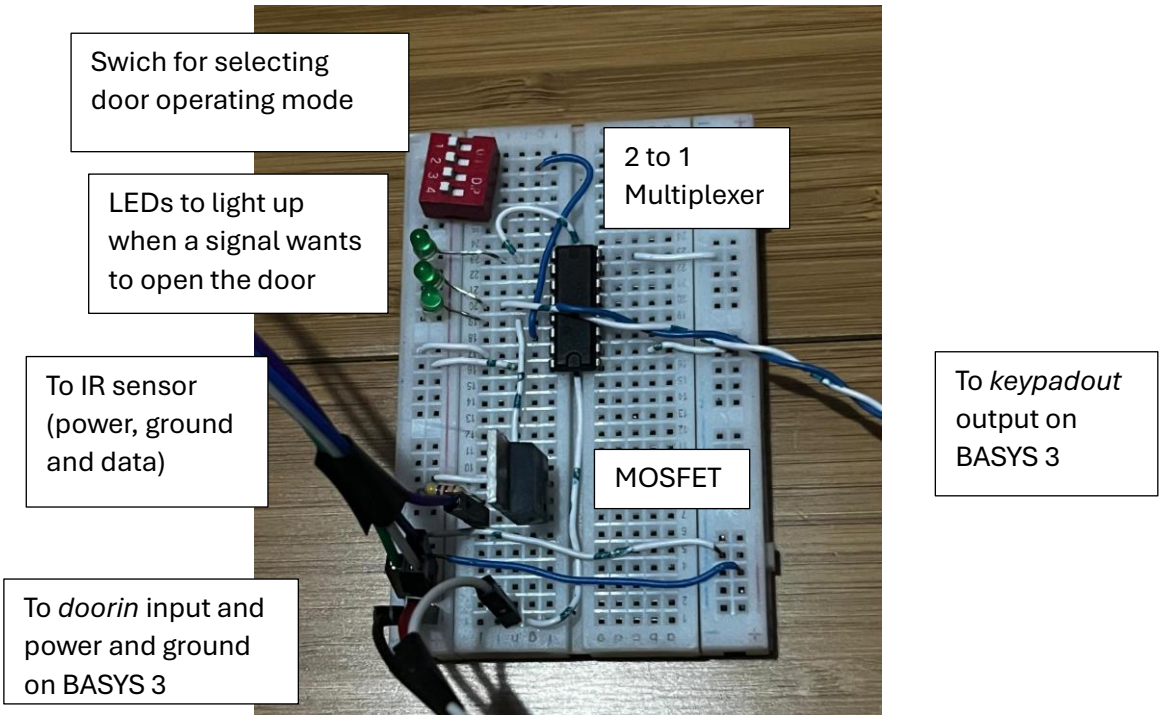

Figure 5.2.2: The breadboard

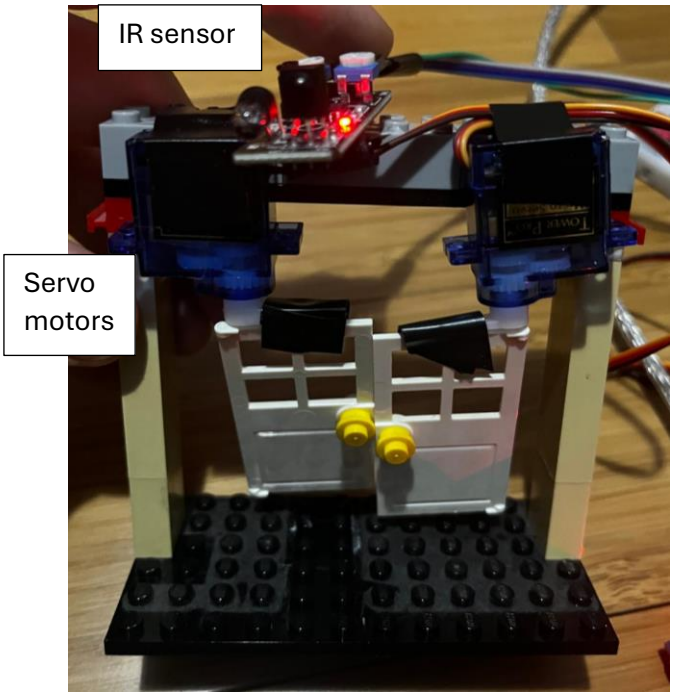

Figure 5.2.3: The door and IR sensor

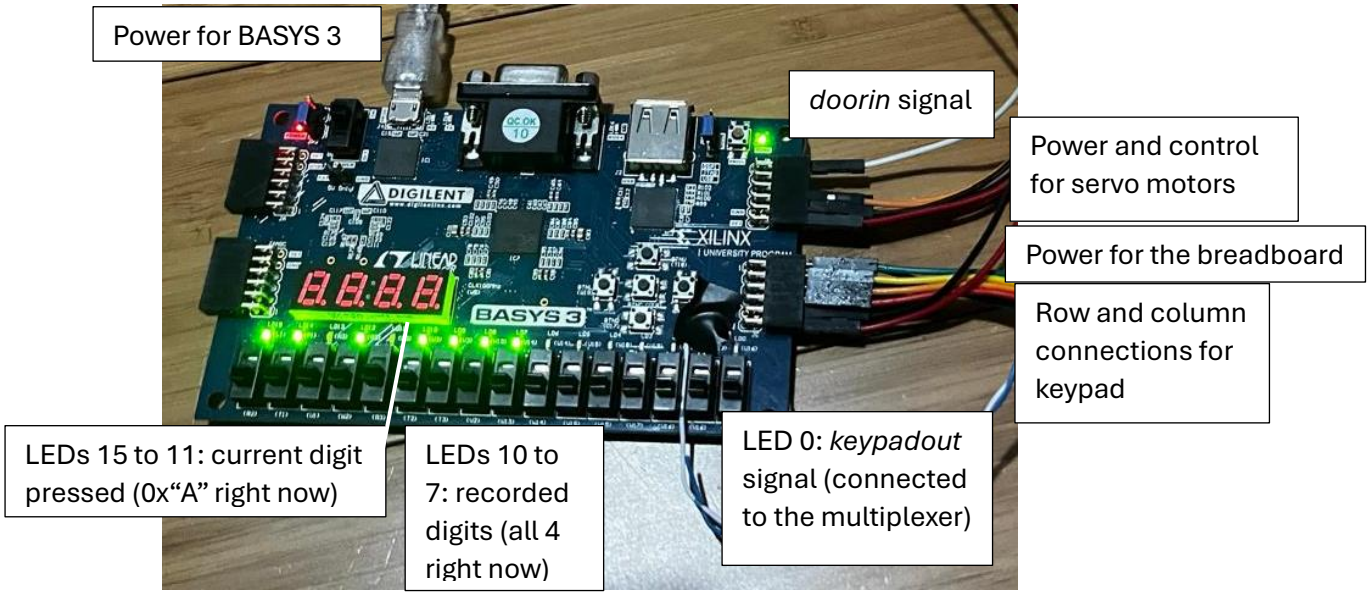

Figure 5.2.4: The BASYS 3 and connections

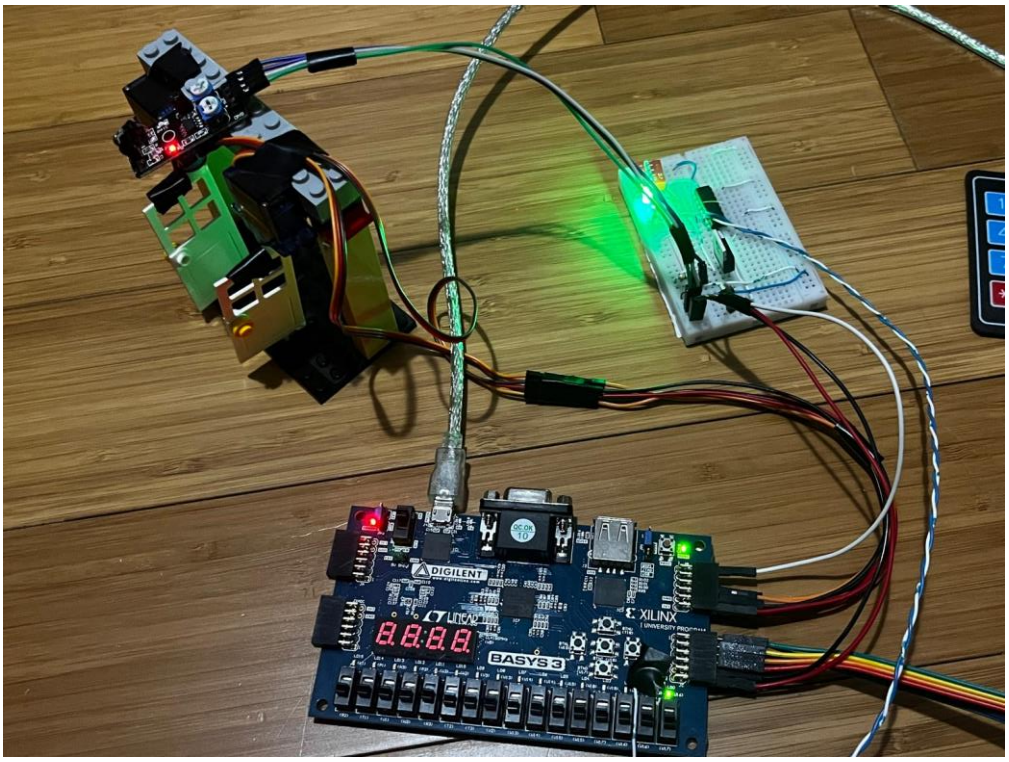

Figure 5.2.5: The door opened (password input mode)

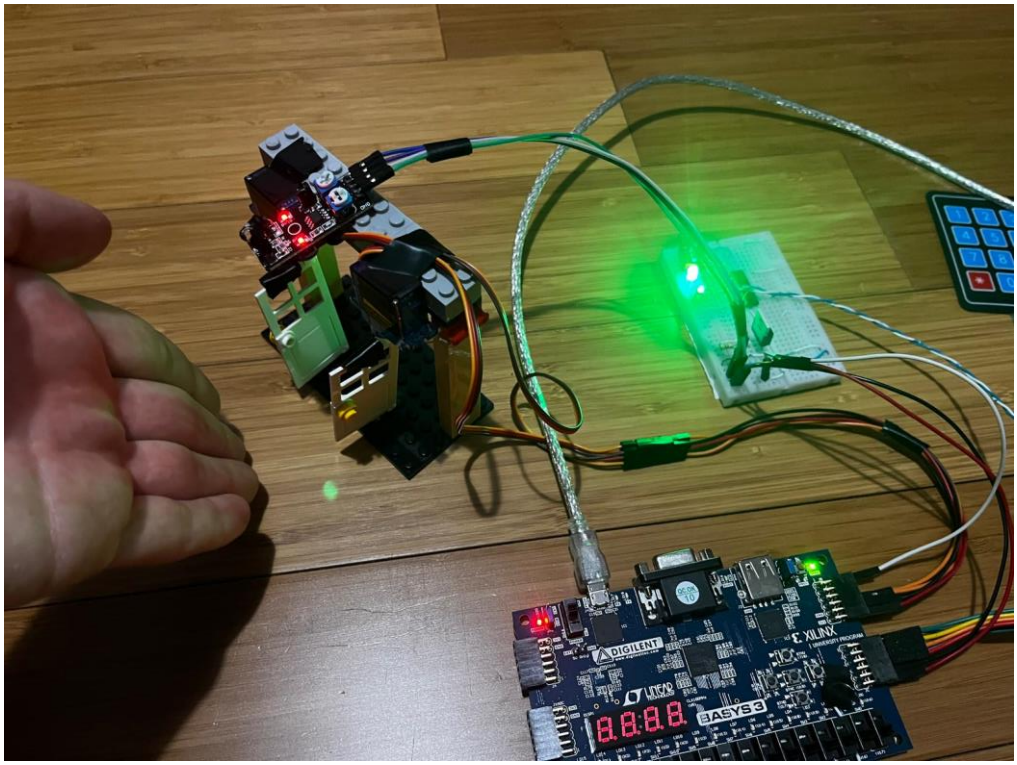

Figure 5.2.6: The door opened (motion input mode)

6 - Conclusion

In this project, I implemented a 4x4 keypad, 2 servo motors and an IR sensor successfully. The most challenging part was the keypad in my opinion since it requires so much detail and different components all connected. Beside that, noise and interference in the electronic circuits had an important effect on how this circuit was designed. For example, when I tried to implement the multiplexer inside the BASYS 3 instead of an external component, some other modules like password module failed to function properly which may be caused by electronic interference within the device.

Also, I needed to connect pulldown resistors in various places in the circuit like the column inputs coming from the 4x4 keypad (using constraints) and drain output of the MOSFET (using a 4.2 kΩ resistor). This is because I needed to maintain that these signals are low (connected to ground) when there is no active signal coming to those components. For the case of 4x4 keypad, when keys are not pressed, there is no signal controlling the column data and for the MOSFET, when the gate input was high, there is no output (which caused some problems with the multiplexer).

Beside those errors, I learned how to design a modular circuit both using FPGA's and integrated circuits by this project. Also, reading and searching for datasheets of various components was a skill I obtained during this project.

7 - Appendices

7.1 – VHDL code for keypad counter

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


entity counter is
    Port (clk5mhz : in std_logic;
          row : out std_logic_vector(3 downto 0);
          keypadpressed : in std_logic);
end counter;


architecture Behavioral of counter is
signal rowcount : std_logic_vector(1 downto 0) := "00";
signal count : integer range 0 to 50000:= 0;
signal clk50hz : std_logic := '0';
begin
```

```
process(clk5mhz)begin
if rising_edge(clk5mhz) then
count <= count + 1;
end if;
if (keypadpressed = '0' and count = 50000) then
clk50hz <= not clk50hz;
end if;
end process;

process(clk50hz)begin
if rising_edge(clk50hz) then
rowcount <= rowcount + 1;
end if;
end process;

process (rowcount) begin
case rowcount is
when "00" => row <= "1000";
when "01" => row <= "0100";
when "10" => row <= "0010";
when "11" => row <= "0001";
when others => row <= "0000";
end case;
end process;

end Behavioral;
```

7.2 – VHDL code for keypad encoder

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

```
entity decoder is
    Port (row : in std_logic_vector(3 downto 0);
          col : in std_logic_vector(3 downto 0);
          currentdigit : out std_logic_vector(4 downto 0));
end decoder;

architecture Behavioral of decoder is
begin

process (row, col) begin
case row is
when "0001" =>
    case col is
    when "1000" => currentdigit <= "10001"; --1
    when "0100" => currentdigit <= "10010"; --2
    when "0010" => currentdigit <= "10011"; --3
    when "0001" => currentdigit <= "11010"; --A
    when others => currentdigit <= "00000"; --no input
    end case;
when "0010" =>
    case col is
    when "1000" => currentdigit <= "10100"; --4
    when "0100" => currentdigit <= "10101"; --5
    when "0010" => currentdigit <= "10110"; --6
    when "0001" => currentdigit <= "11011"; --B
    when others => currentdigit <= "00000"; --no input
    end case;
when "0100" =>
    case col is
    when "1000" => currentdigit <= "10111"; --7
    when "0100" => currentdigit <= "11000"; --8
```

```
    when "0010" => currentdigit <= "11001"; --9
    when "0001" => currentdigit <= "11100"; --C
    when others => currentdigit <= "00000"; --no input
    end case;
when "1000" =>
    case col is
    when "1000" => currentdigit <= "11111"; --* (F)
    when "0100" => currentdigit <= "10000"; --0
    when "0010" => currentdigit <= "11110"; --# (E)
    when "0001" => currentdigit <= "11101"; --D
    when others => currentdigit <= "00000"; --no input
    end case;
when others => currentdigit <= "00000";
end case;
end process;
end Behavioral;
```

7.3 – VHDL code for debouncer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity debouncer is
     Port ( keyin : in std_logic;
     keyout : out std_logic;
     clk : in std_logic);
end debouncer;

architecture Behavioral of debouncer is
signal bigcount : integer range 0 to 100000 := 0;
signal onecount : integer range 0 to 100000 := 0;
signal zerocount : integer range 0 to 100000 := 0;
begin
process(clk) begin
```

```
if rising_edge(clk) then
    bigcount <= bigcount + 1;
    if keyin = '1' then
        onecount <= onecount + 1;
    else
        zerocount <= zerocount + 1;
    end if;
    if bigcount = 100000 then
        if zerocount < onecount then
            keyout <= '1';
        else
            keyout <= '0';
        end if;
        zerocount <= 0;
        onecount <= 0;
     end if;
end if;
end process;
end Behavioral;
```

7.4 – VHDL code for password module

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

entity password is
    Port ( currentdigit : in STD_LOGIC_VECTOR (4 downto 0);
           clk: in STD_LOGIC;
           correct : out STD_LOGIC;
           blink : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end password;

architecture Behavioral of password is
signal enteredpin: STD_LOGIC_VECTOR (19 downto 0) := "00000000000000000000";
constant correctpin: STD_LOGIC_VECTOR (19 downto 0) := '1' & x"A" & '1' & x"1" & '1' & x"2" & '1' & x"0";
signal datafull, corrects: STD_LOGIC;
begin
datafull <= enteredpin(4) and enteredpin(9) and enteredpin(14) and enteredpin(19);

process(currentdigit) begin
    if rising_edge(currentdigit(4)) then
        if corrects = '0' and enteredpin = correctpin and datafull ='1' then
            corrects <= '1';
            enteredpin <= "00000000000000000000";
        end if;
        if corrects = '0' and enteredpin /= correctpin and datafull ='1' then
            enteredpin <= "00000000000000000000";
        end if;
        if corrects = '1' and datafull ='1' then
            corrects <= '0';
            enteredpin <= "00000000000000000000";
        end if;

        if enteredpin(4) = '0' then
            enteredpin(4 downto 0) <= currentdigit;
        elsif enteredpin(9) = '0' then
            enteredpin(9 downto 5) <= currentdigit;
        elsif enteredpin(14) = '0' then
            enteredpin(14 downto 10) <= currentdigit;
        elsif enteredpin(19) = '0' then
            enteredpin(19 downto 15) <= currentdigit;
        end if;
```

```
    end if;
end process;

blink <= enteredpin(4)&enteredpin(9)&enteredpin(14)&enteredpin(19);
correct <= corrects;
end Behavioral;
```

7.5 – VHDL code for keypad module

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

entity main is
    Port (
        stable5mhz : in std_logic;
        row : out std_logic_vector(3 downto 0);
        column : in std_logic_vector(3 downto 0);
        locked : out std_logic := '0';
        recordeddigits : out std_logic_vector(3 downto 0);
        digitdisplay: out std_logic_vector(4 downto 0)
        );
end main;

architecture Behavioral of main is
signal rows: std_logic_vector (3 downto 0);
component counter is
    Port (  clk5mhz : in std_logic;
            row : out std_logic_vector(3 downto 0);
            keypadpressed : in std_logic);
end component;

component decoder is
    Port (row : in std_logic_vector(3 downto 0);
```

```
        col : in std_logic_vector(3 downto 0);
        currentdigit : out std_logic_vector(4 downto 0));
end component;

component debouncer is
    Port ( keyin : in std_logic;
    keyout : out std_logic;
    clk : in std_logic);
end component;

component password is
   Port ( currentdigit : in STD_LOGIC_VECTOR (4 downto 0);
          clk: in STD_LOGIC;
          correct : out STD_LOGIC;
          blink : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal undebounced, debounced : std_logic_vector(4 downto 0);

begin

kpenc: decoder Port Map (row => rows, col => column, currentdigit => undebounced);
kpclock : counter Port Map (row => rows, clk5mhz => stable5mhz, keypadpressed
=>undebounced(4));
db4 : debouncer Port Map ( keyin => undebounced(4), keyout => debounced(4), clk =>
stable5mhz);
db3 : debouncer Port Map ( keyin => undebounced(3), keyout => debounced(3), clk =>
stable5mhz);
db2 : debouncer Port Map ( keyin => undebounced(2), keyout => debounced(2), clk =>
stable5mhz);
db1 : debouncer Port Map ( keyin => undebounced(1), keyout => debounced(1), clk =>
stable5mhz);
db0 : debouncer Port Map ( keyin => undebounced(0), keyout => debounced(0), clk =>
stable5mhz);
digitdisplay <= debounced;
```

```
passwordm: password Port map (currentdigit => debounced, clk=> stable5mhz , correct =>
locked , blink => recordeddigits);


row <= rows;
end Behavioral;
```

7.6 – VHDL code for servo module

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

entity servocontroller is
    Port ( clk5mhz : in STD_LOGIC;
           dooropen : in STD_LOGIC;
           servocontrol1 : out STD_LOGIC;
           servocontrol2 : out STD_LOGIC);
end servocontroller;

architecture Behavioral of servocontroller is
signal count : std_logic_vector (17 downto 0) := "000000000000000000";

begin
process (clk5mhz) begin
        if count = 100000 then
                count <= "000000000000000000";
        elsif rising_edge(clk5mhz) then
                count <= count + 1;
        end if;
        if count = 0 then
                servocontrol1 <= '1';
                servocontrol2 <= '1';
        elsif (dooropen = '1' and count = 5000) or (dooropen = '0' and count = 10000) then
```

```
                servocontrol1 <= '0' ;
        elsif (dooropen = '0' and count = 5000) or (dooropen = '1' and count = 10000) then
                servocontrol2 <= '0' ;
        end if;
end process;
end Behavioral;
```

7.7 – VHDL code for top module

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

entity realtopmodule is
    Port (
        clk100mhz : in std_logic;
        row : out std_logic_vector(3 downto 0);
        column : in std_logic_vector(3 downto 0);
        keypadout : out std_logic := '0';
        doorin: in std_logic;
        servo1: out std_logic;
        servo2: out std_logic;
        tempentereddigit : out std_logic_vector(3 downto 0);
        digit : out std_logic_vector(4 downto 0)
        );
end realtopmodule;

architecture Behavioral of realtopmodule is

component clk_wiz_0
port
 (clk_out1: out std_logic;
  reset: in std_logic;
  clklocked: out std_logic;
```

```
  clk_in1: in std_logic);
end component;

component servocontroller is
    Port ( clk5mhz : in STD_LOGIC;
           dooropen : in STD_LOGIC;
           servocontrol1 : out STD_LOGIC;
           servocontrol2 : out STD_LOGIC);
end component;

component main is
    Port (
        stable5mhz : in std_logic;
        row : out std_logic_vector(3 downto 0);
        column : in std_logic_vector(3 downto 0);
        locked : out std_logic := '0';
        recordeddigits : out std_logic_vector(3 downto 0);
        digitdisplay : out std_logic_vector(4 downto 0)
        );
end component;

signal clk5mhz, stable5mhz, lockedclock, door, openmydoor: std_logic;
begin

stable5mhz <= clk5mhz and lockedclock;
mhz5clkmain : clk_wiz_0 port map ( clk_out1 => clk5mhz ,reset => '0', clklocked =>
lockedclock, clk_in1 => clk100mhz);
servo: servocontroller Port Map( clk5mhz => stable5mhz, dooropen => openmydoor,
servocontrol1 => servo1, servocontrol2 => servo2);

keypad1: main
    Port Map (
        stable5mhz => stable5mhz,
```

```
        row => row,
        column => column,
        locked => door,
        recordeddigits => tempentereddigit,
        digitdisplay => digit);

keypadout <= door;

openmydoor <= doorin;

end Behavioral;
```

7.8 – VHDL code for testbench

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

entity tb is
end tb;

architecture Behavioral of tb is
component realtopmodule is
    Port (
        clk100mhz : in std_logic;
        row : out std_logic_vector(3 downto 0);
        column : in std_logic_vector(3 downto 0);
        keypadout : out std_logic := '0';
        doorin: in std_logic;
        servo1: out std_logic;
        servo2: out std_logic;
        tempentereddigit : out std_logic_vector(3 downto 0);
```

```
        digit : out std_logic_vector(4 downto 0)
        );
end component;

signal clk :std_logic := '0';
signal row : std_logic_vector(3 downto 0):= "1000";
signal column :  std_logic_vector(3 downto 0) := "0000";
signal kout: std_logic := '0';
signal doorin:  std_logic:= '0';
signal servo1: std_logic;
signal servo2: std_logic;
signal tempentereddigit: std_logic_vector(3 downto 0);
signal led: std_logic_vector(4 downto 0);

begin
top: realtopmodule port map (clk, row, column, kout, doorin, servo1, servo2,
tempentereddigit, led);

process begin
    wait for 5 ns;
    clk <= not clk;
end process;

process begin
    wait for 100 ms;
    doorin <= not doorin;
end process;

process begin
    wait for 5.1 ms;
    column <= "0100";
    wait for 0.1 ms;
    column <= "0000";
```

```
    wait for 60 ms;
    column <= "0100";
    wait for 0.1 ms;
    column <= "0000";
    wait for 80 ms;
    column <= "1000";
    wait for 0.1 ms;
    column <= "0000";
    wait for 80 ms;
    column <= "0001";
    wait for 0.1 ms;
    column <= "0000";
    wait for 80 ms;
    column <= "0001";
    wait for 0.1 ms;
    column <= "0000";
    wait for 1 ms;
    wait;
end process;
end Behavioral;
```

8 - References

Alldatasheet.com. “MG90S Download.” ALLDATASHEET.COM, pdf1.alldatasheet.com/datasheet-pdf/download/1132104/ETC2/MG90S.html. Accessed 15 May 2024.

“Lesson 20 – 4×4 Matrix Keypad.” Lesson 20 – 4×4 Matrix Keypad " Osoyoo.Com, osoyoo.com/2017/09/13/arduino-lesson-4x4-matrix-keypad/. Accessed 15 May 2024.

“MOSFET BASICS: Types, Working, Structure, and Applications.” Electronics For You, 24 Aug. 2023, www.electronicsforu.com/technology-trends/learn-electronics/mosfet-basics-working-applications.

Nawazi, Farwah. “HW201 Infrared (IR) Sensor Module.” Circuits DIY, 14 Sept. 2023, www.circuits-diy.com/hw201-infrared-ir-sensor-module/.

Philip Butler Keypad, philipgbutler.files.wordpress.com/2019/05/philip-butler-keypad-final-report.pdf. Accessed 15 May 2024.