Robotic Hand - FPGA Project

Tan Yumuk

Objective:

This project aims to make a robot hand that can implement some functions through user hand movements. The hand can be controlled if the user enters the correct password. The user can control the robot hand's fingers and the lead in the palm of the robot hand, which is assumed to be a repulser.

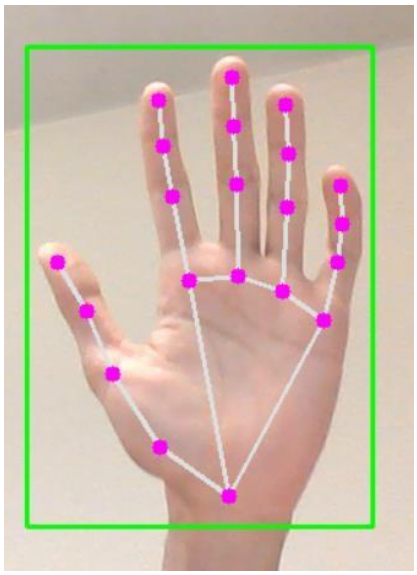
Methodology:

In this project, the inputs from the user are taken through hand gesture visual recognition code in Python. I used the OpenCV library in Python to capture my hand (Figure 1a). Also, I understand whether a specific finger is closed by this library. Then, I generate a list of ones and zeros. Ones represent the finger is open; zeros represent the specific finger is closed. My list has 1 bit for each finger, so I had a 5-bit list at the end of the process. I used Python as the transmitter and Basys 3 as the receiver. I used UART for the transfer. Once the start bit of UART came, I received the 5-bit data bit by bit in the UART receiver. It shiftes bits to prepare next coming bit and stops when stop bit came. I provide this communication at a 9600 baud rate. Then, I assigned those bits to 5 different signals with std_logic, which are going to be used for functions. Firstly, the system is locked and received inputs do not have any impact on the hand, which is demonstrated by the red LED in (figure 1b). There is a finite state machine, a Moore machine with six states. The finite state machine is a digital lock.
There are two different inputs for entering the password. The first input is A and is controlled by the index finger. The second input is B and is controlled by the middle finger. I also used my thumb to control the overloading of inputs because, at the 9600 baud rate, I was receiving many inputs at once. There was no debounce system in visual recognition, so I could not proceed in states. When the correct input is given, it can be the wrong input of the next state's request, so the finite state machine returns to the appropriate state. I wrote the signal with a not gate, representing the thumb, into the process and rising_edge of the finite state machine. Thus, the inputs are valid in every close situation of thumb. It was enough for me to first give the input for the setup time of the flip flop, then close my thumb. Using an input instead of a clock caused an error warning when I tried to implement the code. However, I solved it by writing the code in (figure 1c) to my constraint file. The warning system told me I could use this code if it was necessary to write an input

instead of the clock. My password was ABAAB. Then I entered the password, and when it came to the final state, a signal called “unlock” was assigned high. All functions are valid until the unlock signal is high. When the system is unlocked, the green LED is turned on, and the red LED is turned off (figure 1d). Once the system is unlocked, the user can control the robot’s fingers with finger movements. Each finger is controlled by an MG 90s servo motor, which works at 50 Hz. Hence, I generated 20 ms pwm signals to control those motors. When a finger is close, an approximately 1ms high 19 ms low wave should go to the analog pin of the relevant servo motor. When this finger is open, approximately 2 ms high 18 ms low square wave should be asserted. My system uses Basys 3’s internal 100 MHz clock, so every wave takes ten ns. In every rising edge of the clock, I incremented counters to generate pwms. Hence, I can control the pwms under counters’ value conditions. For instance, if I want to make a signal high for 2 ms, it is enough to increment a counter in each rising edge and make a signal high until this counter reaches 200000. In other conditions, the counter should always turn back to 0 to prevent errors in pwms. I could control all motors simultaneously,
and my system took a maximum of approximately 1 ampere. Also, my system works at 5 Volt. Another feature of the hand is to control the repulser. There is another finite state machine (Moore) with two states to control the repulser. I created a “close” signal, equal to 5 finger signals connected with OR Gates. Thus, the close signal is 0 when all fingers are closed (figure 1e). Then, I incremented a counter on each rising edge of the clock when the close signal was 0. When this counter reaches 300000000 (equals to 3 seconds), an “impulse1” signal is asserted high. When impulse1 is high, the finite state machine changes into another state, and the LED in the palm turns on (figure 1f). After that, I assign low to impulse1 signal. You can do any movement you want without turning off the LED. I implement the same process to return the first state from the second state, but the only difference is that the counter starts to count when I implement the Spiderman move (only the middle and ring finger are close) in (figure 1g). Then impulse2 is asserted high and goes back to the first state. Hence, the LED turns off in (the figure 1h). If I press the button in (the figure I), the digital lock finite state machine resets itself and returns to the first state, so the system is locked.

Inputs: Clk, reset, I_RX_Serial (std_logic for recieving bits)

Outputs: Repulser, PWM(std_logic_vector 5 bit) and also i used many trivial outputs,
assigned to leds on Basys 3 to control project process during making project.



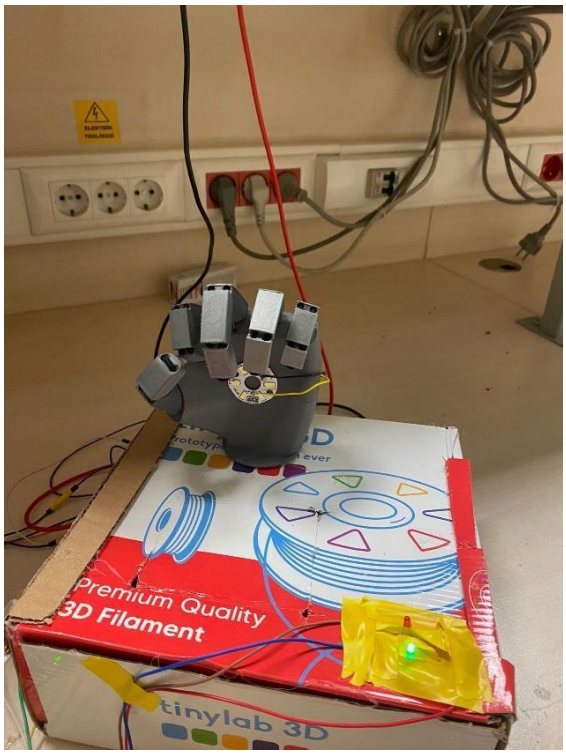
(1a)

(1b)

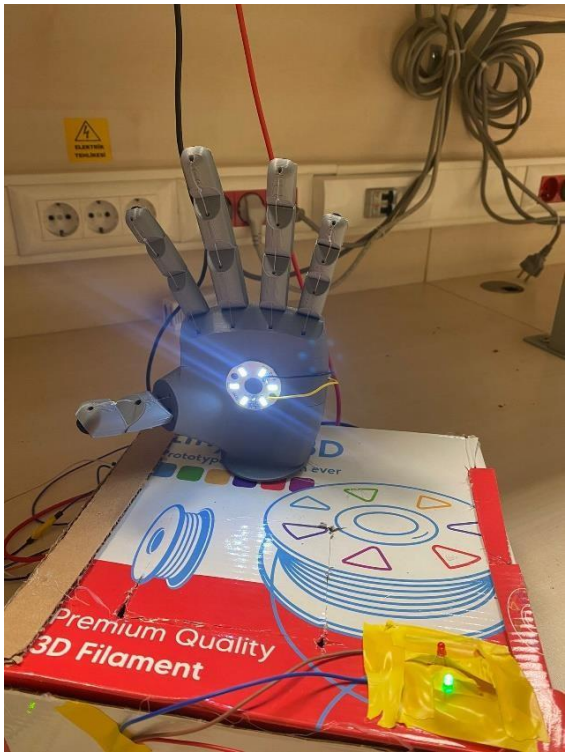
```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets in1_IBUF]
```

(1c)

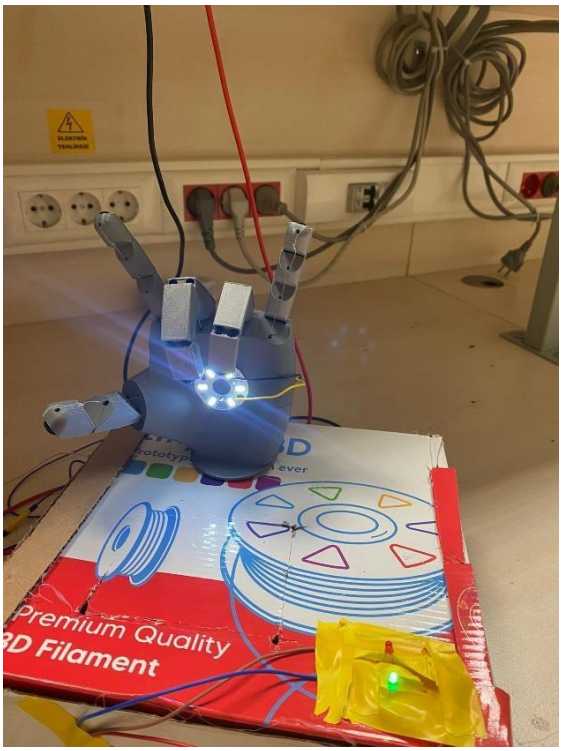
(1d)


Premium Quality
3D Filament
tinylab 3D

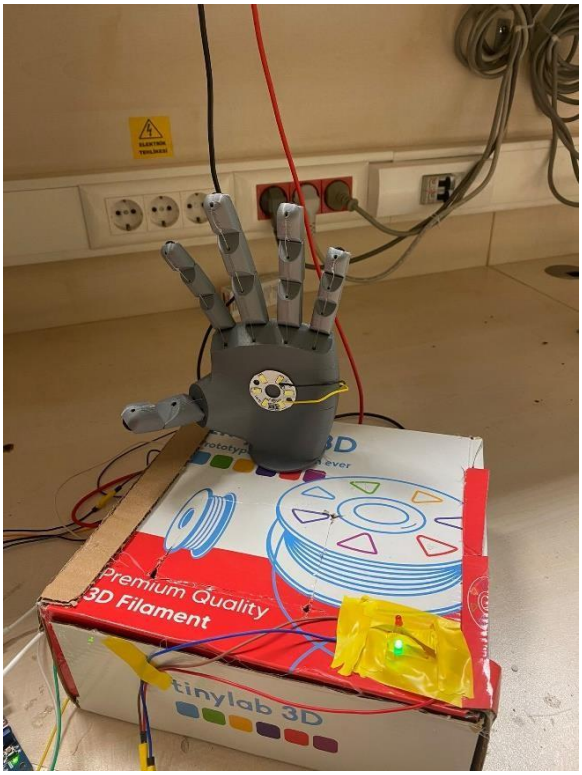
(1e)


Premium Quality
3D Filament

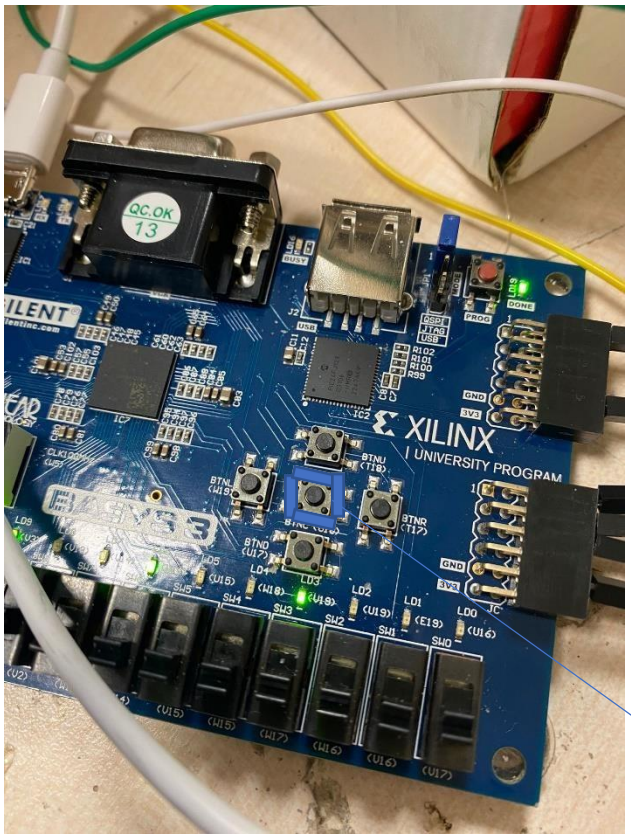
(1f)


Premium Quality
3D Filament

(1g)


Premium Quality
3D Filament
tinylab 3D

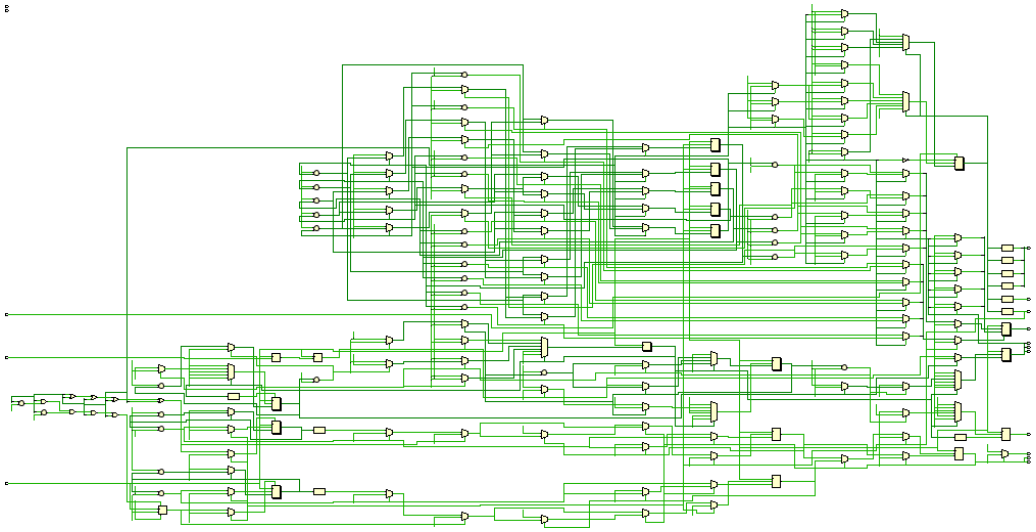
(1h)



(1I)

Design:

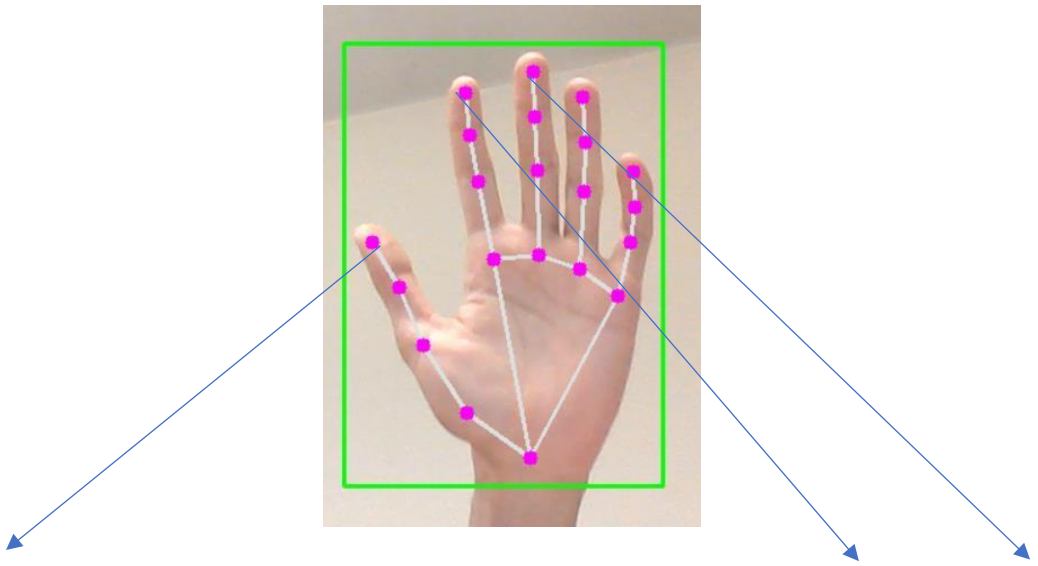
In (the figure 2a), there is the elaborated design of vhdl code.



(2a)

Finite State Machines:

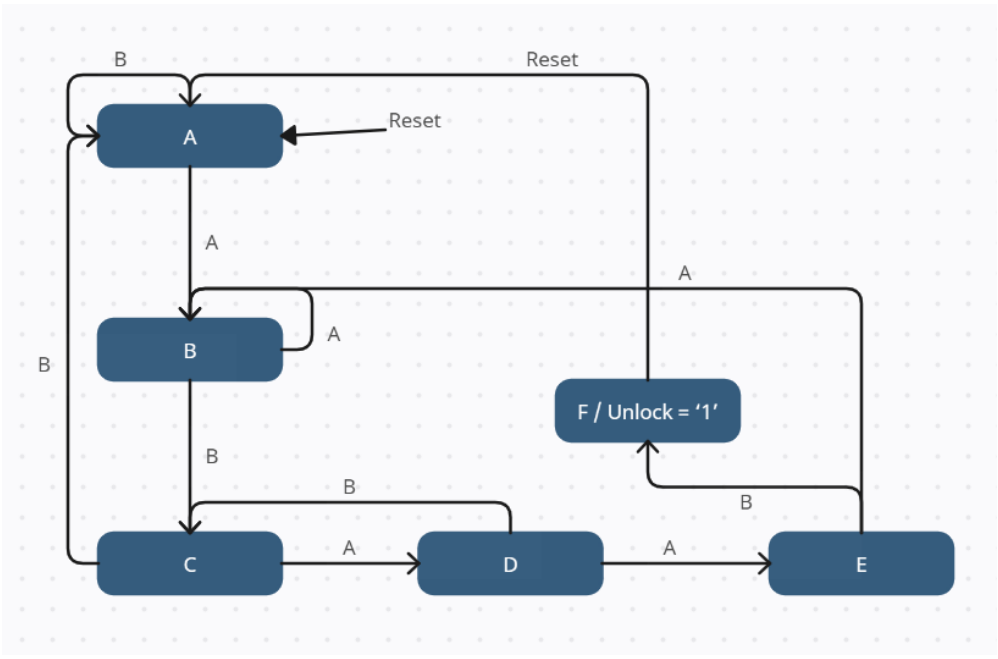
In the figure 3a, it shows correspondings of the inputs A and B with the control signal.



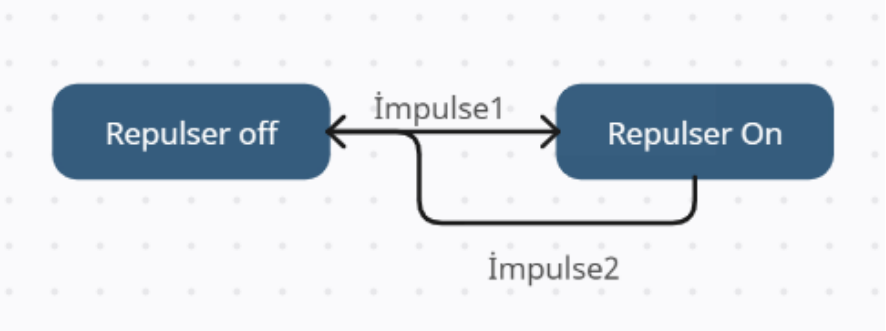
Control signal **(3a)** A input B input

State Diagrams:

Digital lock's finite state machine is demonstrated in (the figure 3b). LED in the palm of hand's state diagram is shown in (the figure 3c). Impulse1 and impulse2 are outputs of combinational circuit, explained in the Methodology.



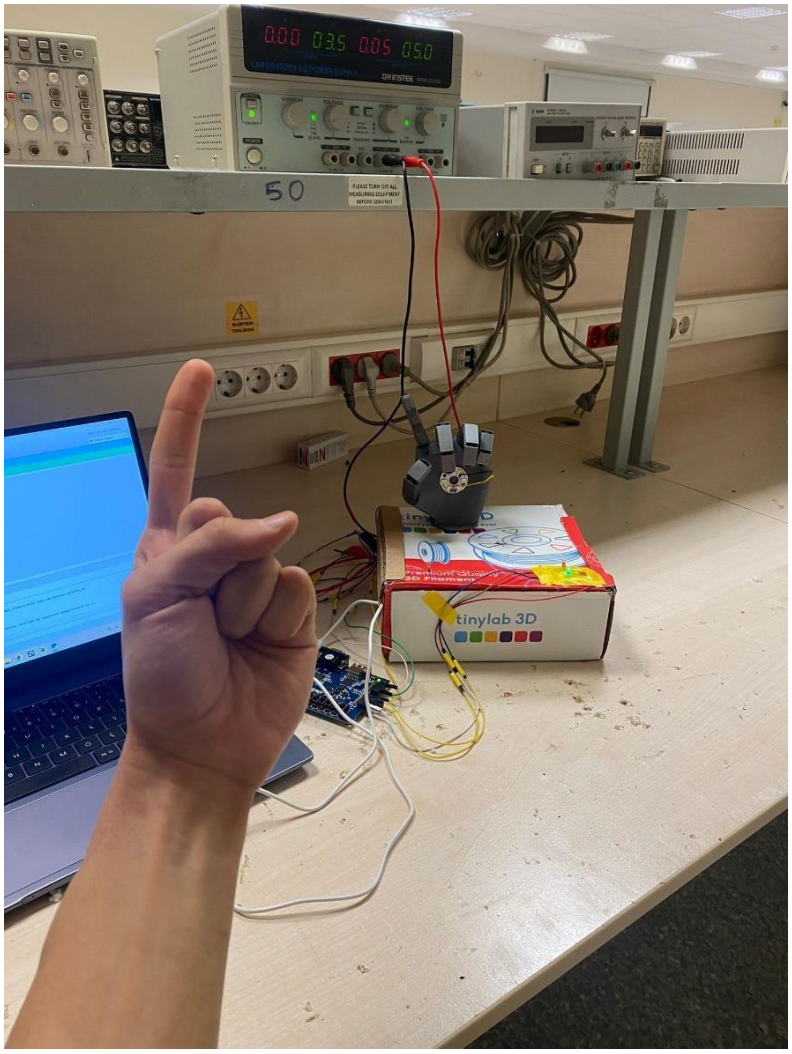
(3b)



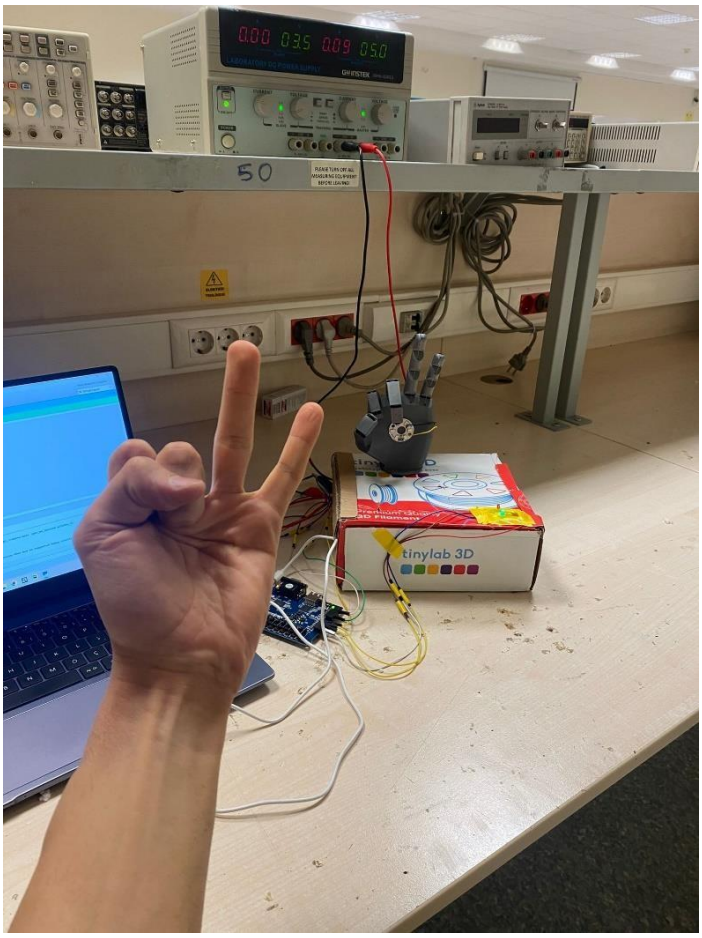
(3c)

Results of Servo Motors:

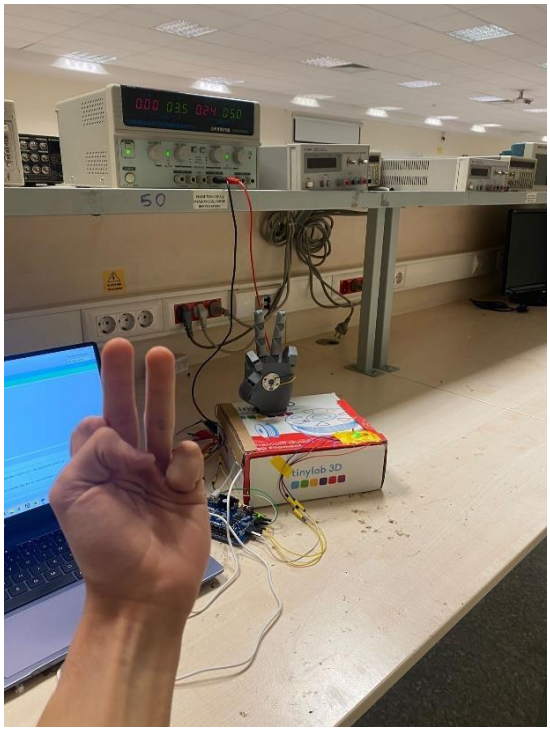
Robot hand simulates the finger movements of user. There are some examples in (the figures 4a, 4b, 4c, 4d, 4e). The working principles of motors are illustrated in Methodology.



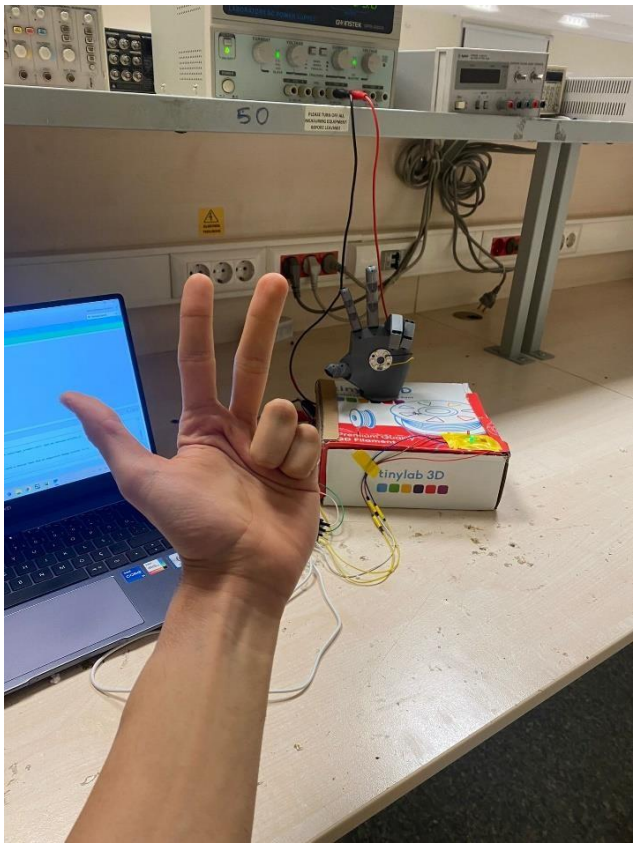
(4a)



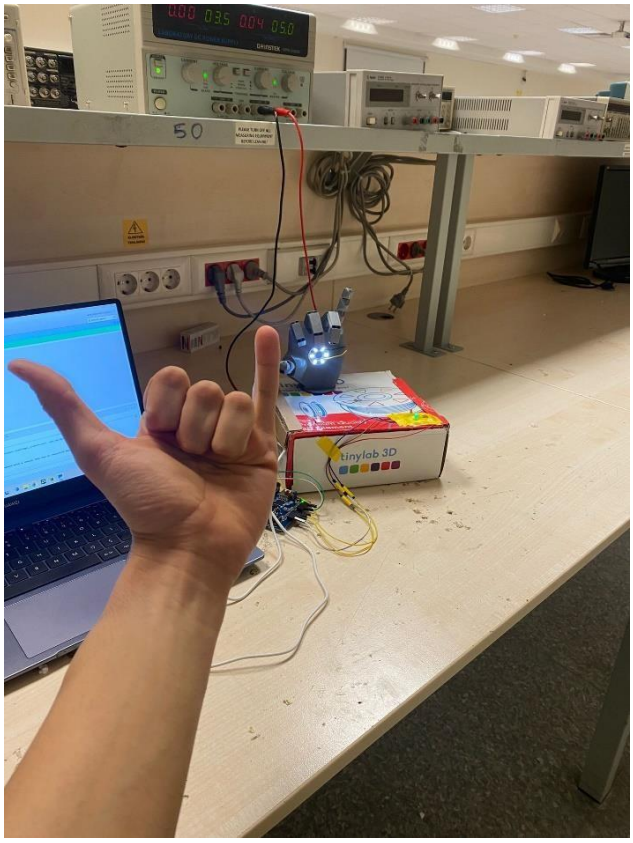
(4b)



(4c)



(4d)



(4e)

Conclusion:

In this project, I made a robot hand with a digital lock that can control an LED in the palm. Also, the user can control their fingers. All inputs are given visually. I encountered some problems in this project. First, when I tried to enter the password by giving visual inputs, there was no debouncing system. I could decrease the clock, but it would undermine my servo control process by making it very slow and adding a delay to it. So, I solved it by adding a control signal (thumb). Second, sometimes, the user can enter the correct password by luck. Even when the wrong input is entered, the machine returns to the appropriate state; my friend gave two inputs and unlocked the system. It could be because I left the system in the fourth state, and my friend entered two correct inputs by luck. I could solve it by adding one more input and assigning it to the ring finger. Hence, it would avoid the risk. Lastly, sometimes, if I remove my hand from the vision of my camera, my robot hand closes all its fingers. Actually, it happens by approximately a 10% chance, but in my video, it happened too by a low chance, and I wanted to mention it. Also, the LED turns on if the user does not recognize the hand closed all fingers for 3 seconds. I am happy not to use a real repulser instead of an LED since I figured out this bug.

Project Video link:

https://www.youtube.com/watch?v=_u1SvZUec80

References:

UART receiver video => https://www.youtube.com/watch?v=XpfEHPg5AxU

3D Printed Hand files => https://www.viralsciencecreativity.com/post/arduino-flex-sensor-controlled-robot-hand

Appendix:

Pyhton:

```
import serial

import cvzone

import cv2

ser = serial.Serial('COM5', 9600)

cap = cv2.VideoCapture(0)

detector = cvzone.HandDetector(maxHands=1, detectionCon=1)


while True:
    success, img = cap.read()
    img = detector.findHands(img)
    lmList, bbox = detector.findPosition(img)
    if lmList:
        fingers = detector.fingersUp()
        fingers.append(0)
        fingers.append(0)
        fingers.append(0)
        binary_byte = int(''.join(map(str, fingers)), 2)
        print(fingers)
        ser.write(bytes([binary_byte]))
    cv2.imshow("Image", img)
    cv2.waitKey(1)
ser.close()
```

VHDL:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 08.04.2024 18:35:21
-- Design Name:
-- Module Name: servo - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity servo is
    generic (
    g_CLOCKS_PER_BIT : integer := 10416
    );
    Port (clk: In std_logic;
          reset: in std_logic;
          i_RX_Serial : in  std_logic;
          trivial: out std_logic;
          pwm: out std_logic_vector(7 downto 3);
          o_RX_Byte   : out std_logic_vector(7 downto 0);
          repulser : OUT std_logic;
          P, Q     :   IN STD_LOGIC;
          c1, c2, c3, c4 : OUT std_logic;
          lon, loff: out std_logic;
```

```
        del: out std_logic_vector(3 downto 0));
end servo;

 architecture Behavioral of servo is

signal counter1 : natural range 0 to 2000000:= 0;
signal counter2 : natural range 0 to 2000000:= 0;
signal counter3 : natural range 0 to 2000000:= 0;
signal counter4 : natural range 0 to 2000000:= 0;
signal counter5 : natural range 0 to 2000000:= 0;
signal counterr : natural range 0 to 300000000:= 0;
signal counterr2 : natural range 0 to 300000000:= 0;

signal fingers: std_logic_vector (7 downto 0);
type t_Main_State is (Idle, RX_Start_Bit, RX_Data_Bits,
                      RX_Stop_Bit, Cleanup);
  signal r_Main_State : t_Main_State := Idle;

  signal r_Data_R : std_logic := '0';
  signal r_Data   : std_logic := '0';

  signal r_Clock_Count : integer range 0 to g_CLOCKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;
  signal r_Received_Byte   : std_logic_vector(7 downto 0) := (others => '0');
```

```
 signal r_Received_Valid      : std_logic := '0';


TYPE State_type IS (A, B, C, D, E, F);
   signal State : State_Type;


   signal button1: std_ulogic;
   signal button2: std_ulogic;
   signal unlock: std_logic;
   signal s1 : std_logic;



TYPE State_type2 IS (Roff, Ron);
   signal State2 : State_Type2;
   signal close: std_logic ;
   signal spider: std_logic;
   signal impulse1 : std_logic;
   signal impulse2 : std_logic;


begin


  p_Sample : process (clk)
  begin
    if rising_edge(clk) then
      r_Data_R <= i_RX_Serial;
```

```
    r_Data  <= r_Data_R;
  end if;
end process p_Sample;



p_UART_RX : process (clk)
begin
  if rising_edge(clk) then
    case r_Main_State is
    when Idle =>
       r_Received_Valid      <= '0';
       r_Clock_Count <= 0;
       r_Bit_Index <= 0;

       if r_Data = '0' then
         r_Main_State <= RX_Start_Bit;
       else
         r_Main_State <= Idle;
       end if;


     when RX_Start_Bit =>
       if r_Clock_Count = (g_CLOCKS_PER_BIT-1)/2 then
         if r_Data = '0' then
```

```
    r_Clock_Count <= 0;
   r_Main_State  <= RX_Data_Bits;
   else
    r_Main_State  <= Idle;
   end if;
  else
   r_Clock_Count <= r_Clock_Count + 1;
   r_Main_State  <= RX_Start_Bit;
  end if;



when RX_Data_Bits =>
  if r_Clock_Count < g_CLOCKS_PER_BIT-1 then
   r_Clock_Count <= r_Clock_Count + 1;
   r_Main_State  <= RX_Data_Bits;
  else
   r_Clock_Count          <= 0;
   r_Received_Byte(r_Bit_Index) <= r_Data;



   if r_Bit_Index < 7 then
   r_Bit_Index <= r_Bit_Index + 1;
   r_Main_State  <= RX_Data_Bits;
   else
```

```
      r_Bit_Index <= 0;
      r_Main_State  <= RX_Stop_Bit;
    end if;
  end if;


when RX_Stop_Bit =>

  if r_Clock_Count < g_CLOCKS_PER_BIT-1 then
    r_Clock_Count <= r_Clock_Count + 1;
    r_Main_State  <= RX_Stop_Bit;
  else
    r_Received_Valid      <= '1';
    r_Clock_Count <= 0;
  r_Main_State  <= Cleanup;
  end if;


when Cleanup =>
r_Main_State <= Idle;
r_Received_Valid  <= '0';
when others =>
r_Main_State <= Idle;
```

```
      end case;
    end if;
  end process p_UART_RX;

  trivial  <= r_Received_Valid;
  fingers <= r_Received_Byte;
  close <= fingers(3)OR fingers(4)OR fingers(5)OR fingers(6)OR fingers(7);

process(clk)
 begin
 if rising_edge (clk) then
   if fingers(3) = '1' and fingers(4)= '0' and fingers(5) = '0' and fingers(6) = '1' and  fingers(7) = '1' then
    spider <= '0';
   else
   spider <= '1';
   end if;
end if;

end process;

button1 <= fingers(6);
button2 <= fingers(5);
```

```
s1 <= not(fingers(7));


PROCESS (s1, reset)
  BEGIN
    IF reset = '1' THEN
        State <= A;


    ELSIF rising_edge(s1) THEN


        CASE State IS
            WHEN A =>
                IF button1='0' THEN
                    State <= B;
                END IF;
            WHEN B =>
                IF button1='0' THEN
                    State <= B;
                elsif button2 = '0' then
                    State <= C;
                END IF;
            WHEN C =>
                IF button1='0' THEN
                    State <= D;
```

```
            elsif button2 = '0' then
               State <= A;
            END IF;
         WHEN D =>
            IF button1='0' THEN
               State <= E;
            elsif button2 = '0' then
               State <= C;
            end if;
         WHEN E =>
            IF button1='0' THEN
               State <= B;
            elsif button2 = '0' then
               State <= F;
            end if;
         WHEN F=>
            IF reset='1' THEN
               State <= A;
            END IF;
      END CASE;
   END IF;
 END PROCESS;
del(0) <= '1' when state = A else '0';
del(1) <= '1' when state = B else '0';
```

```
del(2) <= '1' when state = C else '0';
del(3) <= '1' when state = D else '0';

c3 <= button1;
c4 <= button2;
unlock <= '1' WHEN State=F ELSE '0';
lon <= '1' WHEN State=F ELSE '0';
loff <= '0' WHEN State=F ELSE '1';

process (clk)
   begin
      if rising_edge (clk) then
        if unlock = '1' then
          if close = '0' then
             if counterr < 300000000 then
                counterr <= counterr + 1;
                if counterr = 299999999 then
                   counterr <= 0;
                   impulse1 <= '1';
                end if;
             end if;
          else
             counterr <= 0;
          end if;
```

```
 if spider = '0' then
    if counterr2 < 300000000 then
        counterr2 <= counterr2 + 1;
        if counterr2 = 299999999 then
            counterr2 <= 0;
            impulse2 <= '1';
        end if;
    end if;
else
    counterr2 <= 0;
end if;



Case State2 is
    when Roff =>
        if impulse1 = '1' then
            State2 <= Ron;
            impulse1 <= '0';
        end if;
    when Ron =>
        if impulse2 = '1' then
            State2 <= Roff;
            impulse2 <= '0';
```

```
                end if;
            END CASE;
          end if;
        end if;
END PROCESS;

repulser <= '1' WHEN State2 = Ron ELSE '0';
c1 <= '1' WHEN State2 = Ron ELSE '0';
c2 <= '1' WHEN State2 = Roff ELSE '0';

process (clk)
begin
if rising_edge (clk) then

  if unlock = '1' then
   if fingers(7) = '1' then

        if counter1 < 250000 then
        pwm(7) <= '1';
        counter1 <= counter1 + 1;
        elsif counter1 < 2000000 then
        pwm(7) <= '0';
        counter1 <= counter1 + 1;
            if counter1 = 1999999 then
```

```
            counter1 <= 0;
            end if;
        else
            counter1 <= 0;
        end if;

elsif fingers(7) = '0' then

        if counter1 < 50000 then
        pwm(7) <= '1';
        counter1 <= counter1 + 1;
        elsif counter1 < 2000000 then
        pwm(7) <= '0';
        counter1 <= counter1 + 1;
            if counter1 = 1999999 then
            counter1 <= 0;
            end if;
        else
            counter1 <= 0;
        end if;
end if;

if fingers(6) = '1' then
```

```
    if counter2 < 250000 then
    pwm(6) <= '1';
    counter2 <= counter2 + 1;
    elsif counter2 < 2000000 then
    pwm(6) <= '0';
    counter2 <= counter2 + 1;
        if counter2 = 1999999 then
        counter2 <= 0;
        end if;
    else
        counter2 <= 0;
    end if;

elsif fingers(6) = '0' then

    if counter2 < 50000 then
    pwm(6) <= '1';
    counter2 <= counter2 + 1;
    elsif counter2 < 2000000 then
    pwm(6) <= '0';
    counter2 <= counter2 + 1;
        if counter2 = 1999999 then
        counter2 <= 0;
        end if;
```

```
        else
            counter2 <= 0;
        end if;
    end if;

    if fingers(5) = '1' then

        if counter3 < 250000 then
        pwm(5) <= '1';
        counter3 <= counter3 + 1;
        elsif counter3 < 2000000 then
        pwm(5) <= '0';
        counter3 <= counter3 + 1;
            if counter3 = 1999999 then
            counter3 <= 0;
            end if;
        else
            counter3 <= 0;
        end if;

     elsif fingers(5) = '0' then

        if counter3 < 50000 then
        pwm(5) <= '1';
```

```
        counter3 <= counter3 + 1;
        elsif counter3 < 2000000 then
        pwm(5) <= '0';
        counter3 <= counter3 + 1;
            if counter3 = 1999999 then
            counter3 <= 0;
            end if;
        else
            counter3 <= 0;
        end if;
    end if;

    if fingers(4) = '1' then

        if counter4 < 250000 then
        pwm(4) <= '1';
        counter4 <= counter4 + 1;
        elsif counter4 < 2000000 then
        pwm(4) <= '0';
        counter4 <= counter4 + 1;
            if counter4 = 1999999 then
            counter4 <= 0;
            end if;
        else
```

```
            counter4 <= 0;
        end if;

    elsif fingers(4) = '0' then

        if counter4 < 50000 then
        pwm(4)<= '1';
        counter4 <= counter4 + 1;
        elsif counter4 < 2000000 then
        pwm(4) <= '0';
        counter4 <= counter4 + 1;
            if counter4 = 1999999 then
            counter4 <= 0;
            end if;
        else
            counter4 <= 0;
        end if;
    end if;

    if fingers(3) = '1' then

        if counter5 < 250000 then
        pwm(3) <= '1';
        counter5 <= counter5 + 1;
```

```
    elsif counter5 < 2000000 then
    pwm(3) <= '0';
    counter5 <= counter5 + 1;
        if counter5 = 1999999 then
        counter5 <= 0;
        end if;
    else
        counter5 <= 0;
    end if;

elsif fingers(3) = '0' then

    if counter5 < 50000 then
    pwm(3) <= '1';
    counter5 <= counter5 + 1;
    elsif counter5 < 2000000 then
    pwm(3) <= '0';
    counter5 <= counter5 + 1;
        if counter5 = 1999999 then
        counter5 <= 0;
        end if;
    else
        counter5 <= 0;
    end if;
```

```
      end if;
   end if;
   end if;
end process;
o_RX_Byte <= fingers;
end Behavioral;
```

Constraint:

```
set_property PACKAGE_PIN W5 [get_ports clk]

                        set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN U18 [get_ports reset]

                        set_property IOSTANDARD LVCMOS33 [get_ports reset]


set_property PACKAGE_PIN M18 [get_ports {lon}]
                        set_property IOSTANDARD LVCMOS33 [get_ports {lon}]
##Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports {loff}]
                        set_property IOSTANDARD LVCMOS33 [get_ports {loff}]


set_property PACKAGE_PIN J1 [get_ports {pwm[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {pwm[7]}]
##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {pwm[6]}]
```

```
                    set_property IOSTANDARD LVCMOS33 [get_ports {pwm[6]}]
##Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {pwm[5]}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {pwm[5]}]
##Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {pwm[4]}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {pwm[4]}]

set_property PACKAGE_PIN A14 [get_ports {pwm[3]}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {pwm[3]}]

set_property PACKAGE_PIN B18 [get_ports i_RX_Serial]
    set_property IOSTANDARD LVCMOS33 [get_ports i_RX_Serial]

set_property PACKAGE_PIN U16 [get_ports {o_RX_Byte[0]}]

                    set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[0]}]
set_property PACKAGE_PIN E19 [get_ports {o_RX_Byte[1]}]

                    set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[1]}]
set_property PACKAGE_PIN U19 [get_ports {o_RX_Byte[2]}]

                    set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[2]}]

set_property PACKAGE_PIN V19 [get_ports {o_RX_Byte[3]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[3]}]

set_property PACKAGE_PIN W18 [get_ports {o_RX_Byte[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[4]}]

set_property PACKAGE_PIN U15 [get_ports {o_RX_Byte[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[5]}]

set_property PACKAGE_PIN U14 [get_ports {o_RX_Byte[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[6]}]

set_property PACKAGE_PIN V14 [get_ports {o_RX_Byte[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_RX_Byte[7]}]

set_property PACKAGE_PIN V13 [get_ports {del[0]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {del[0]}]
set_property PACKAGE_PIN V3 [get_ports {del[1]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {del[1]}]
set_property PACKAGE_PIN W3 [get_ports {del[2]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {del[2]}]
set_property PACKAGE_PIN U3 [get_ports {del[3]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {del[3]}]
```

```
set_property PACKAGE_PIN P1 [get_ports {c1}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {c1}]
set_property PACKAGE_PIN L1 [get_ports {c2}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {c2}]
set_property PACKAGE_PIN P3 [get_ports {c4}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {c4}]
set_property PACKAGE_PIN N3 [get_ports {c3}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {c3}]

#jb4
set_property PACKAGE_PIN B16 [get_ports {trivial}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {trivial}]

#Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {repulser}]
                    set_property IOSTANDARD LVCMOS33 [get_ports {repulser}]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets in1_IBUF]
```