EEE-102 Project Report

EYE PROSTHESIS THAT RESPONDS TO NEAR OBJECTS

Robin Umut Kızıl

Section-1

Youtube link: https://youtu.be/-scGbJCCMjY

1. Introduction

With the latest technological developments, the development of artificial organs and the like is expected. In this project, an attempt was made to prototype a mechanism that could move an eye prosthesis.

2. Aim and Project Description

This project aims to make an eye model that moves and looks according to nearby objects, using a Xilinx Basys 3 FPGA based on a VHDL. The project includes 2 servos, one of which controls the vertical and the other horizontal angle, which is attached vertically to each other as an eye model, and 4 infrared sensors located to the left, right, top and bottom of the eye model, which adjust the servos angles. In this way, the eye model can detect approaching objects with infrared sensors and can create the impression of looking at the object by turning 4 directions and 4 corners.

3. Used Materials

- SG90 RC Mini (9gr) Servo Motor (2 pieces)
- TCRT5000 Infrared Reflection Sensor Module (4 pieces)
- Breadboard
- Necessary materials for the model (cardboard, glue, etc...)
- Basys 3

4. Design Specifications and Methodology

The top module in the design is Top_eye_project and it has 6-input and 6-output. they are all in std_logic format. servo_vertical and servo_horizontal output as PWM signals.

Input ports are;

- **reset:** It resets the clock signals and allows the servos to come to the default position, facing center.
- **clk:** Allows us to import and use the internal 100MHz clock in Basys 3.
- **IR_Left:** receives the left side infrared signal as digital input.
- **IR_Right:** receives the right side infrared signal as digital input.
- **IR_Top:** receives the top side infrared signal as digital input.
- **IR_Bottom:** receives the bottom side infrared signal as digital input.

Output ports are;

- **Led1:** connected to the left infrared sensor, it is used to make sure the sensor is working properly.
- **Led2:** connected to the right infrared sensor, it is used to make sure the sensor is working properly.
- **Led3:** connected to the top infrared sensor, it is used to make sure the sensor is working properly.
- **Led4:** connected to the bottom infrared sensor, it is used to make sure the sensor is working properly.
- **servo_vertical:** connected on the servo controlling the vertical angle. It sends the angle of the servo as a PWM signal at 50Hz (20ms), which is the working clock of the servo.
- **servo_horizontal:** connected on the servo controlling the horizontal angle. It sends the angle of the servo as a PWM signal at 50Hz (20ms), which is the working clock of the servo.

This top module has 2 top_servo submodules and 4 information from infrared go these 2 sub-modules as the angle determined in the top module, and the PWM signal from the modules is exported as output to servo motors.

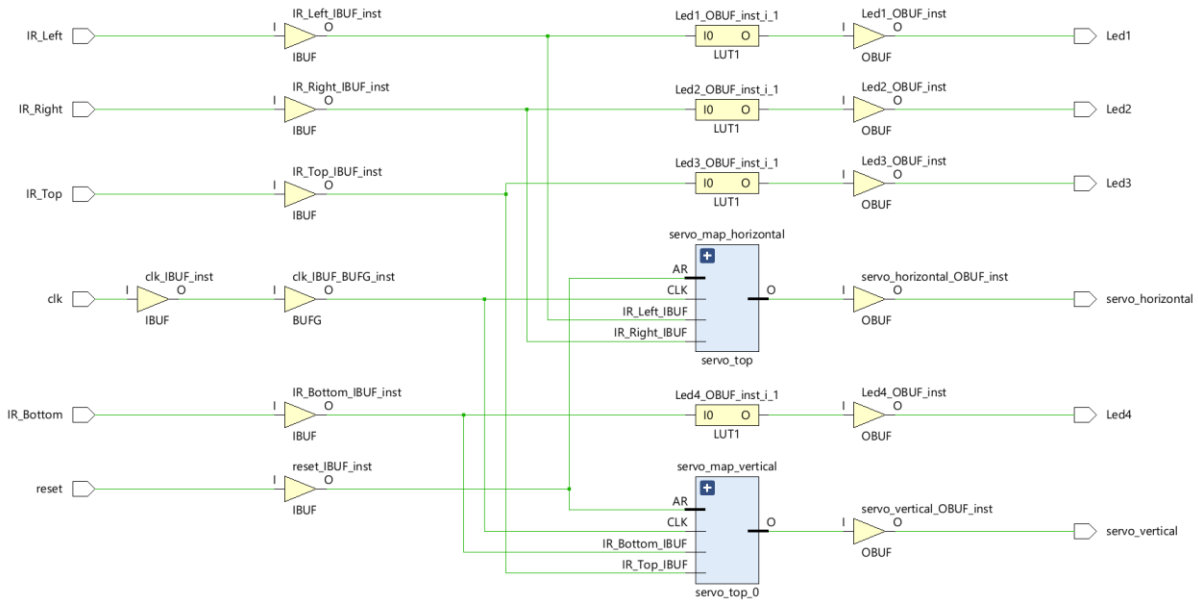

Figure 1: Top_eye_project Schematic

4.1 servo_top Submodule

The aim of this module is to keep the modules critical for the servo together and connect them with the top module. This module includes servo_clk and servo_pwm modules as Submodules. It sends the clk signal coming from the top module, to the servo_clk Submodule, allowing us to lower our clock to the 50Hz (20ms) range and divide it so that we can control it at 128 different angles. then it sends the signal and position signal that comes from the top module, there to the servo_pwm Submodule and sends the PWM signal coming from there back to the Top module.

4.2 servo_clk Submodule

The aim of this module is to divide the clk signal coming from the upper module and create a clock signal suitable for the pwm signal that can control the servo. The basic internal clock, 100MHz, is divided by the appropriate integer (1560 used in the project). The clock frequency found should be internal clock divided by a multiple of the estimated number of angles for the servo to rotate and the servo's operating frequency. This new clock frequency is then transferred to the top module to be sent to the PWM signal.

4.3 servo_pwm Submodule

This module is there to generate the PWM signal that will move the servo relative to the position vector. Top takes the position vector and the new clock from the module, using these two to generate the PWM signal that operates at the operating frequency of the servo and returns it to the desired position.

5. Results and Conclusion

The aim of the project was to make an eye model that reacts to objects coming near and gives the impression of looking at them. 2 servos, one horizontal and one vertical, were used to rotate the eye model, and 4 infrared sensors was used to detect the object so that the eye model can rotate in 4 directions and 4 corners. As a mistake, the PWM code was not accurate enough because the servo was cheap. I refactored the code for this. Also, the 0.5kg torque of the servo did not support a round eye weight. so the eye was made as small as possible (and rectangular in accordance with the shape of the servo). In this way, an eye model was made that appears to be responding to the near object, so the Project was carried out successfully and many of what was taught in the class were used.

SOURCE;

- Jensen J. J.. 21 Jul 2012. *"RC SERVO CONTROLLER USING PWM FROM AN FPGA PIN"* 13 May 2023. https://vhdlwhiz.com/rc-servo-controller-using-pwm/
- Memeşil A.. 19 July 2017. *"FPGA İle PWM RC Servo Motor"*. 13 May 2023. https://roboturka.com/fpga/fpga-ve-pwm-rc-servo-motor/
- Ramos C.A.. 20 Dec 2012. "*Servomotor Control with PWM and VHDL"*. 15 May 2023. https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL

DATASHEETS;

- http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
- https://pdf.direnc.net/upload/tcrt5000-sensor.pdf

Appendix A:

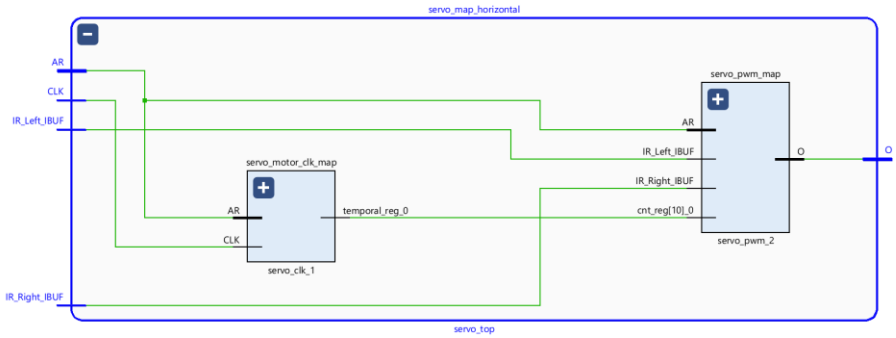

Figure 2: servo_top schematic

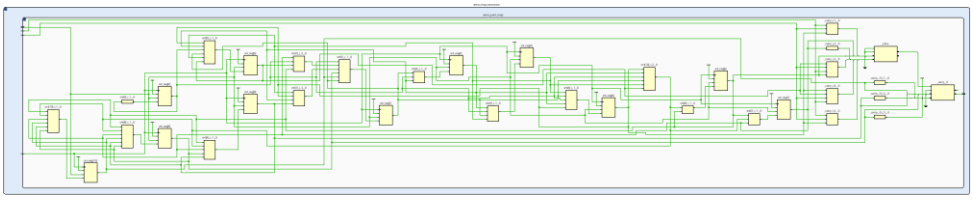

Figure 3: Servo_pwm schematic

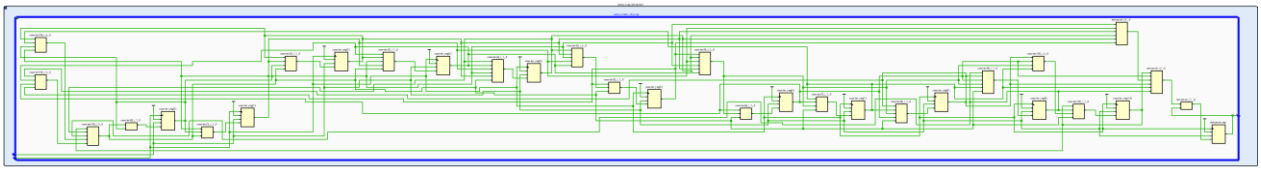

Figure 4: servo_clk schematic

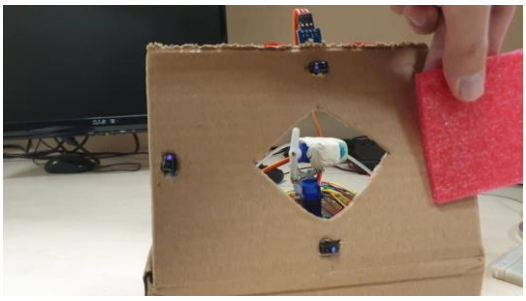

Figure 5: right facing position

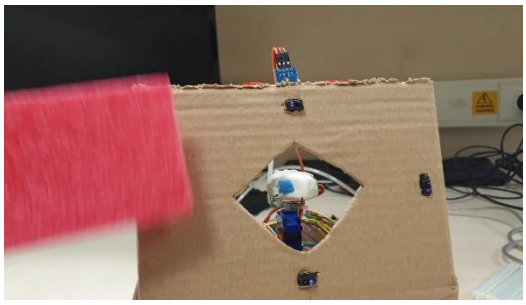

Figure 6: left facing position

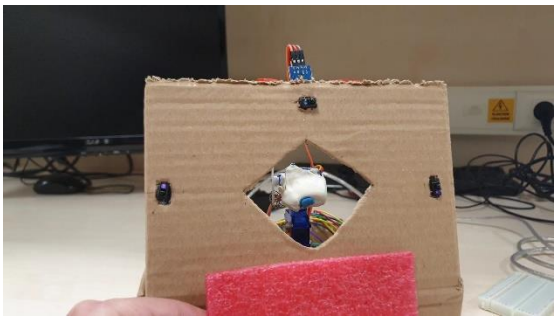

Figure 7: bottom facing position

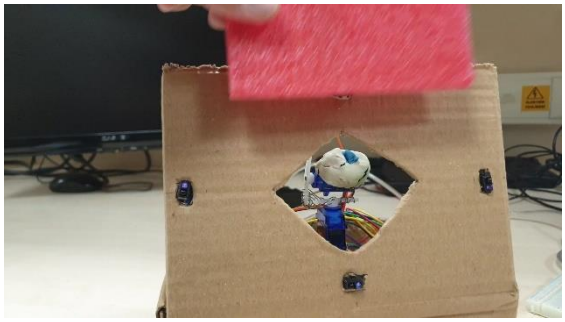

Figure 8: top facing position

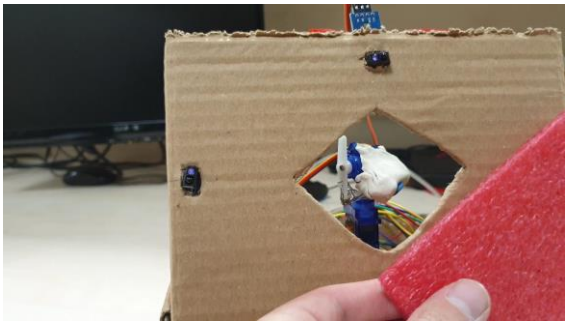

Figure 9: bottom-right facing position

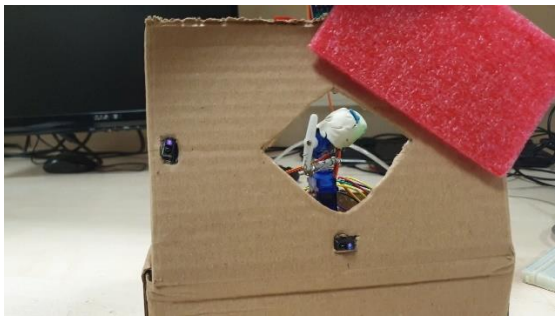

Figure 10: top-right facing position

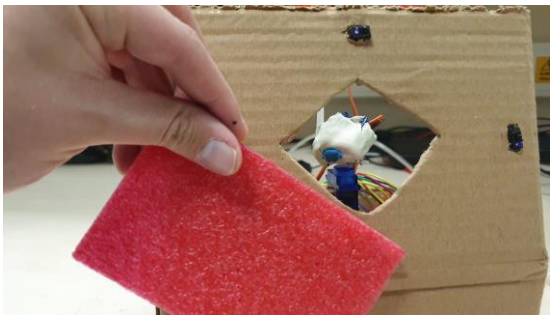

Figure 11: bottom-left facing position

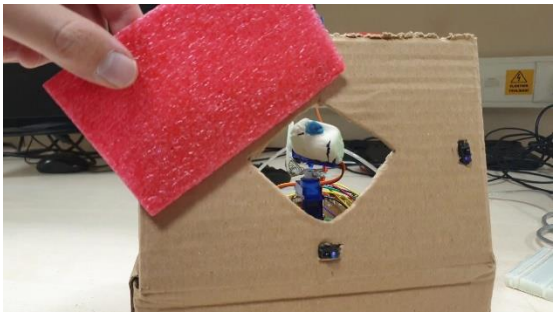

Figure 12: top-left facing position

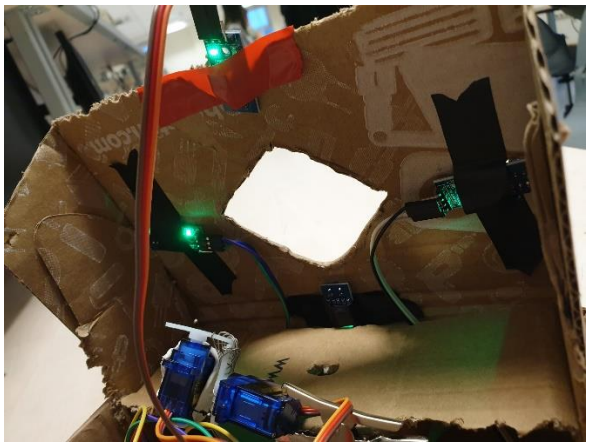

Figure 13: placement of the electronic pieces 1

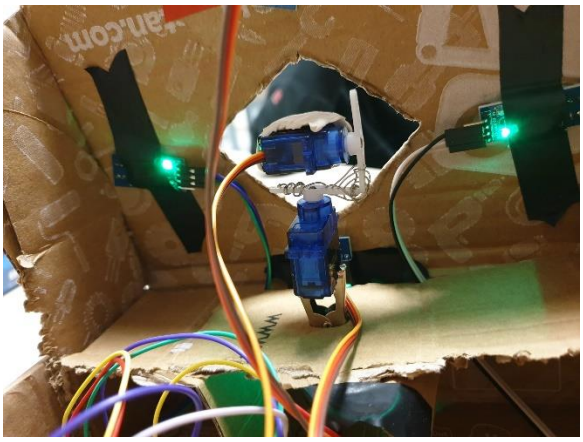

Figure 14: placement of the electronic pieces 2

Appendix B :

Top_eye_project

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



entity Top_eye_project is
    Port ( reset : in std_logic;
           clk : in std_logic;
           IR_Left : in std_logic;
           IR_Right : in std_logic;
           IR_Top : in std_logic;
           IR_Bottom : in std_logic;
           Led1: out std_logic;
           Led2 : out std_logic;
           Led3 : out std_logic;
           Led4 : out std_logic;
           servo_vertical : out std_logic;
           servo_horizontal : out std_logic);
end Top_eye_project;
```

```
architecture Behavioral of Top_eye_project is

component servo_top is
 PORT(
 clk : IN STD_LOGIC;
 reset: IN STD_LOGIC;
 position : IN STD_LOGIC_VECTOR(6 downto 0);
 servo: OUT STD_LOGIC);
end component;

signal position_vertical : std_logic_vector(6 downto 0);
signal position_horizontal : std_logic_vector(6 downto 0);

begin

servo_map_vertical : servo_top port map(clk=>clk,reset
=>reset,position=>position_vertical,servo=>servo_vertical);
servo_map_horizontal : servo_top port map(clk=>clk,reset
=>reset,position=>position_horizontal,servo=>servo_horizontal);

Led1 <= not IR_Left ;
Led2 <= not IR_Right;
```

```
Led3 <= not IR_Top ;

Led4 <= not IR_Bottom;


process(IR_Left,IR_Right,clk)

begin


if IR_Left = '0' then

position_horizontal <= "0000000";

elsif IR_Right = '0' then

position_horizontal <= "0100000";

else

position_horizontal <= "0010000";

end if;


end process;



process(IR_Top,IR_Bottom,clk)

begin


if IR_Top = '0' then

position_vertical <= "0000000";

elsif IR_Bottom = '0' then
```

```
position_vertical <= "0100000";
else
position_vertical <= "0010000";
end if;

end process;



end Behavioral;
```

servo_top

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity servo_top is
 PORT(
 clk : IN STD_LOGIC;
 reset: IN STD_LOGIC;
 position : IN STD_LOGIC_VECTOR(6 downto 0);
 servo: OUT STD_LOGIC);
end servo_top;

architecture Behavioral of servo_top is
```

```
COMPONENT servo_clk
PORT(
clk : in STD_LOGIC;
reset : in STD_LOGIC;
clk_out: out STD_LOGIC );
END COMPONENT;

COMPONENT servo_pwm
PORT (
clk : IN STD_LOGIC;
reset : IN STD_LOGIC;
position : IN STD_LOGIC_VECTOR(6 downto 0);
servo : OUT STD_LOGIC );
END COMPONENT;

signal clk_out : STD_LOGIC := '0';

begin

servo_motor_clk_map: servo_clk PORT MAP(clk=>clk, reset=>reset, clk_out=>clk_out );
```

```
 servo_pwm_map: servo_pwm PORT MAP(clk=>clk_out, reset=>reset, position=>position, servo=>
servo);


end Behavioral;
```

servo_clk

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity servo_clk is
 Port (

 clk : in STD_LOGIC;
 reset : in STD_LOGIC;
 clk_out: out STD_LOGIC);

end servo_clk;



architecture Behavioral of servo_clk is

 signal new_clock_time: STD_LOGIC;
```

```
signal counter : integer range 0 to 1560 := 0;

begin

frequencyD: process (reset, clk) begin
if (reset = '1') then
new_clock_time <= '0';
counter <= 0;

elsif rising_edge(clk) then
if (counter = 1560) then
new_clock_time <= NOT(new_clock_time);
counter <= 0;
else

counter <= counter + 1;
end if;
end if;
```

```
 end process;




 clk_out <= new_clock_time;
end Behavioral;
```

servo_pwm

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

entity servo_pwm is



 PORT (
 clk : IN STD_LOGIC;
 reset : IN STD_LOGIC;
 position : IN STD_LOGIC_VECTOR(6 downto 0);
 servo : OUT STD_LOGIC );
end servo_pwm;
```

```
architecture Behavioral of servo_pwm is

 signal count : unsigned(10 downto 0);
 signal pwm_signal: unsigned(7 downto 0);
begin

 pwm_signal <= unsigned('0' & position) + 32;

 counter: process (reset, clk) begin
 if (reset = '1') then
 count <= (others => '0');

 elsif rising_edge(clk) then
 if (count = 1279) then
 count <= (others => '0');

 else
 count <= count + 1;

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

```
 servo <= '1' when (count < pwm_signal) else '0';

end Behavioral;
```