EEE-102 Term Project

-Position Detect System-

Defne Yaz Kılıç
22102167
Section 2

Youtube video: https://youtu.be/82syhwRzdpM

Purpose

This project aims to detect the position of a magnet (pawn) placed on a 2x2 square via magnetic sensors and to place another magnet (pawn) to the exact detected place on another 2x2 square using servo motors. It is a simpler version of a smart chess board.

Methodology

The design consist of two parts:

1. Detecting the place of the pawn / displaying its position via the LEDs

In order to do so, 4 US 1881 active low hall effect sensors are placed under each square (*Fig.1*) . The signals coming from them are used as inputs of the Vhdl design via Pmod ports of Basys3. An encoder is designed to show the position as binary number 1 to 4.

2. Turning the servo motors

Two servo motors are placed on top of each other to achieve a 270 degree turn. The one on the top has a metal rod attached to its swing arm so that it can control the displacement of the second magnet.(*Fig.2,3*)

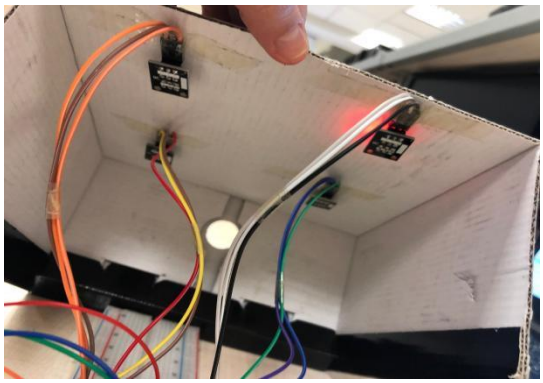


Fig.1 Hall effect sensors

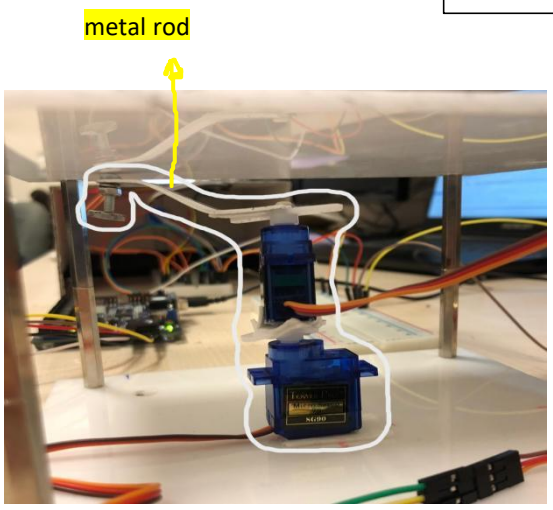


Fig.2 Servo motors with metal rod attched

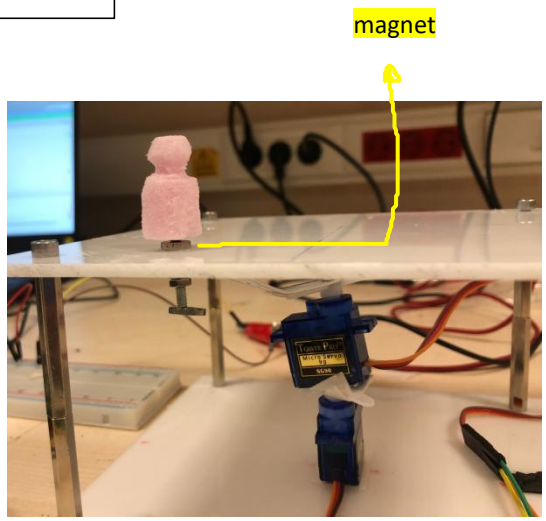


Fig.3 Servo motors controlling the motion of the magnet

Design Specifications

As for the design an FSM is created as main module. It has 7 inputs and 3 outputs:

clk : in STD_LOGIC;
enable: in STD_LOGIC;
reset : in STD_LOGIC;
pin1 : in STD_LOGIC;
pin2 : in STD_LOGIC;
pin3 : in STD_LOGIC;
pin4 : in STD_LOGIC;
- Signals coming from sensors
servo1 : out std_logic;
servo2 : out std_logic;
now : out STD_LOGIC_vector(2 downto 0) - LEDcontrol

The FSM main module consist of an encoder submodule, a 64kHz clk divider and a pulse width modulation submodule for servo control as can be seen below.

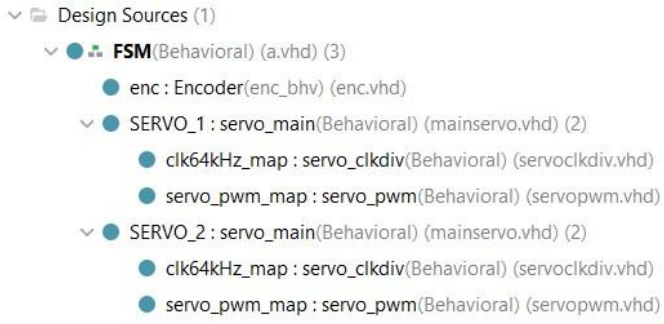


Fig.4 The Hyerarchy

The encoder module takes the active low sensor signals as inputs and gives the binary number corresponding to the place of the magnet through LEDs. Its schematic can be seen below.(Fig. 5)

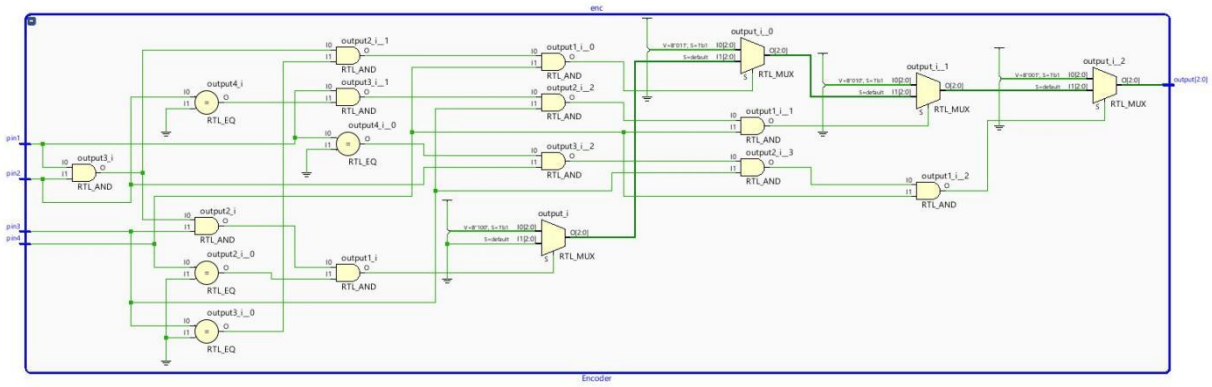


Fig.5 Encoder submodule

For servo control one should understand its working principles. The Sg90 servo motors work with pulse width modulation and it can turn 128 steps in 2 ms therefore it needs a 2ms/128= 64 kHz clock. The clock divider submodule (Fig.6) creates the 64 kHz clock for pwm submodule.

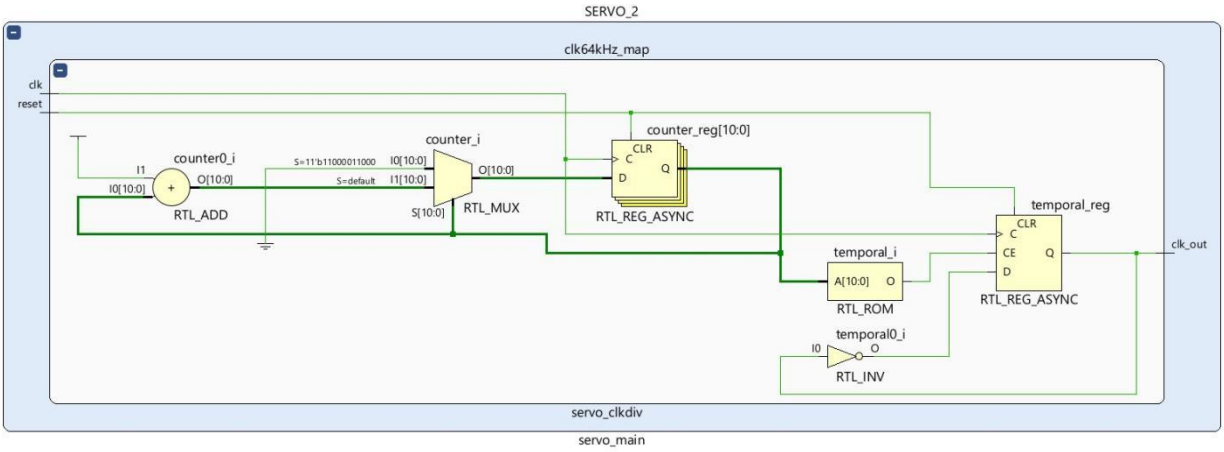


Fig.6 Clock Divider submodule

Servo motor needs 1ms - 2 ms pulses in 20 ms (Fig.7), adding the delay time the pulse range becomes 0.5ms - 2.5 ms. The 0.5 ms long pulse corresponds to a 0 degree turn and 2.5 ms pulse corresponds to a 180 degree turn. The pwm submodule arranges the turns of servos according to given input.(Fig.8)

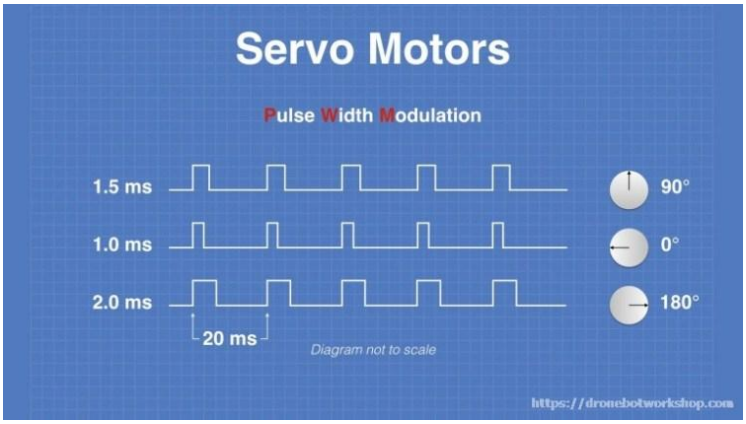


Fig.7 PWM for servo motors

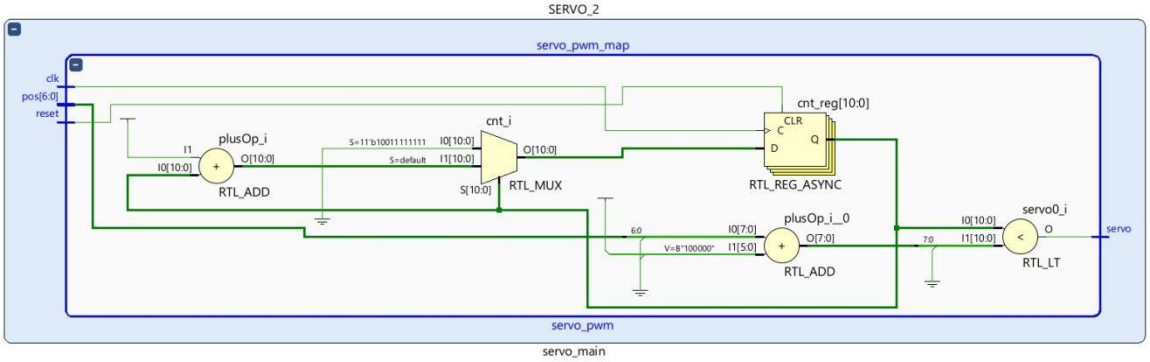


Fig.8 Pwm submodule

For the FSM main module, a moore machine of 4 states (A,B,C,D) is created.(Fig.9,10) Each state corresponds to the magnet being on one square i.e. being in state A means that the magnet is on the first square, being in B means second square etc.For each position the required turn for servo motors are calculated:

1. Being on the first square
   - -Top servo: 0 degrees turn
   - -Bottom servo: 0 degrees turn

2. Being on the second square
   - -Top servo: 90 degrees turn
   - -Bottom servo: 0 degrees turn

3. Being on the third square
   - -Top servo: 135 degrees turn
   - -Bottom servo: 135 degrees turn

4. Being on the fouth square
   - -Top servo: 90 degrees turn
   - -Bottom servo: 90 degrees turn

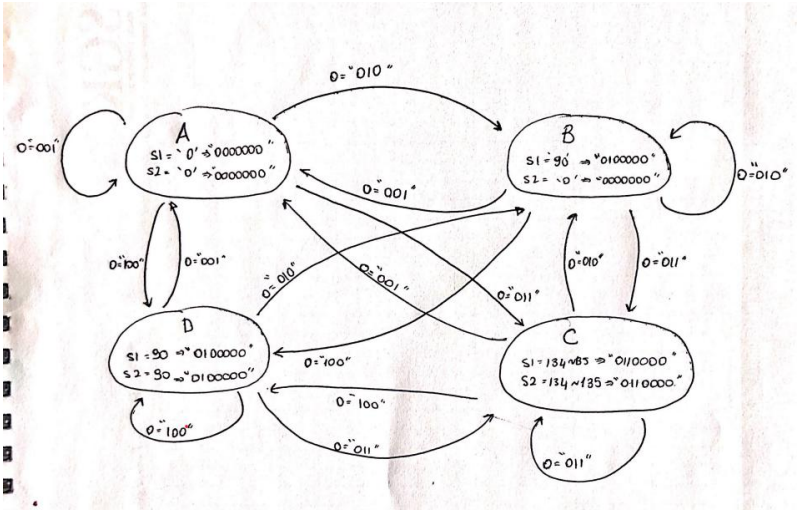


Fig.9 State Transition Diagram

To get rid of the state in which no signals are coming from the sensors that is during the replacement of the magnet, an enable signal is used. When pressed to the enable button, data is sent to receiver. Additionaly a reset signal is also added so that there is an initial position of the magnet, a starting point. It is chosen as the first state A.The setup can be seen in Fig.11.

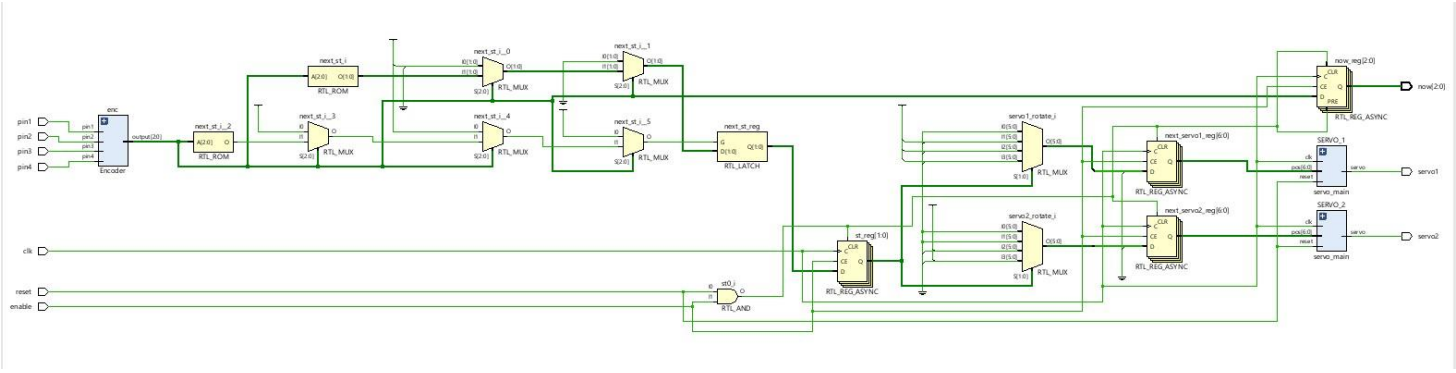


Fig.10 Top Module (FSM) schematic

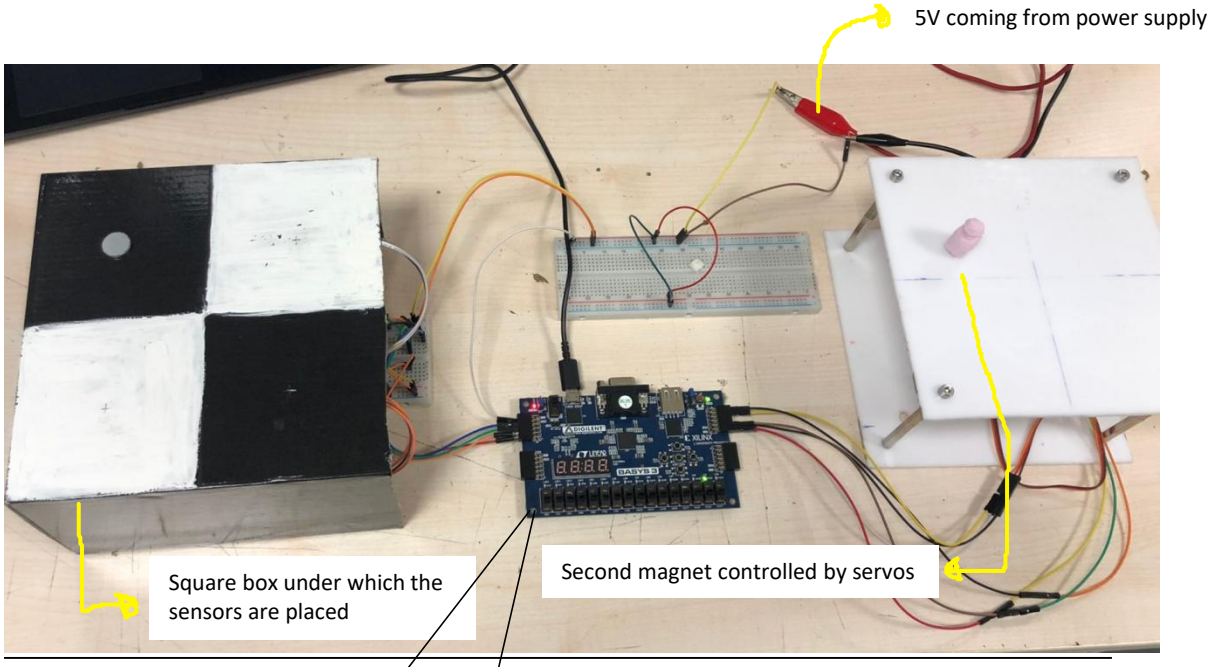
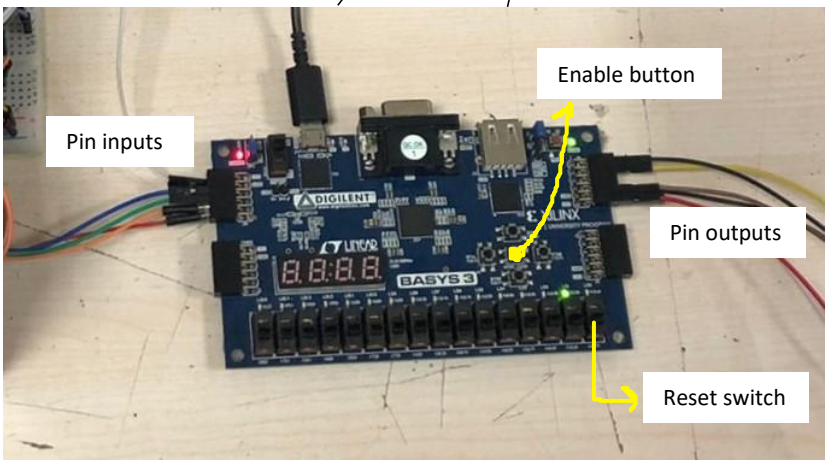


Fig.11 Setup



Results

After completing the setup, the first magnet is put to different places and the second magnet repeated the motion. Some of the examples can be seen below:

- When the magnet is on the first square:
  pin1= "0", pin2 = "1", pin3 = "1", pin4 = "1 (active low)
  LED output = "001"
  Servo1<= 0 degrees turn
  Servo2 <= 0 degrees turn
  State: A

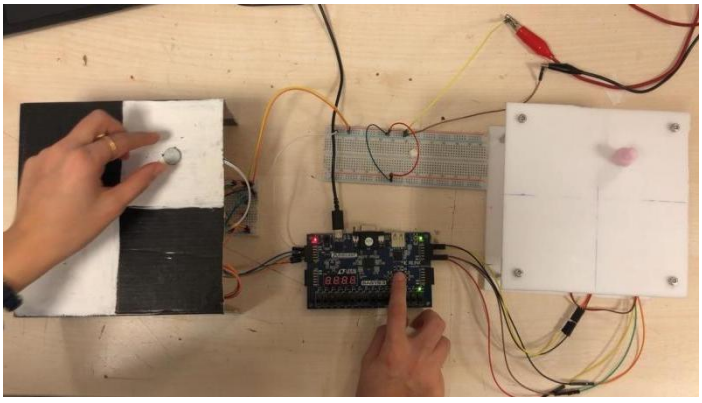


Fig.12 State A case

- When the magnet is on the second square:
  State: B
  pin1= “1”, pin2 = “0”, pin3 = “1”, pin4 = “1
  LED output = “010”
  Servo1<= 90 degrees turn
  Servo2 <= 0 degrees turn

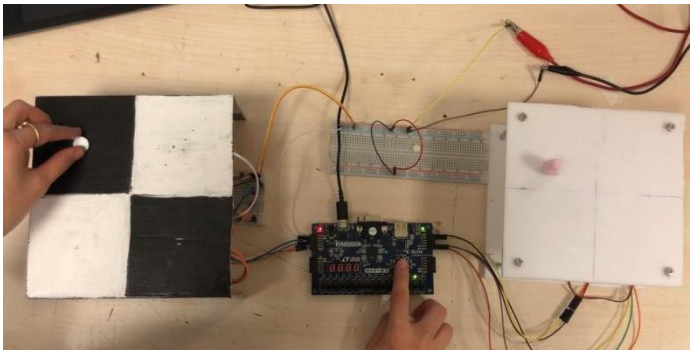


Fig.13 State B case

- When the magnet is on the third square:
  State: C
  pin1= “1”, pin2 = “1”, pin3 = “0”, pin4 = “1
  LED output = “011”
  Servo1<= 135 degrees turn
  Servo2 <= 135 degrees turn

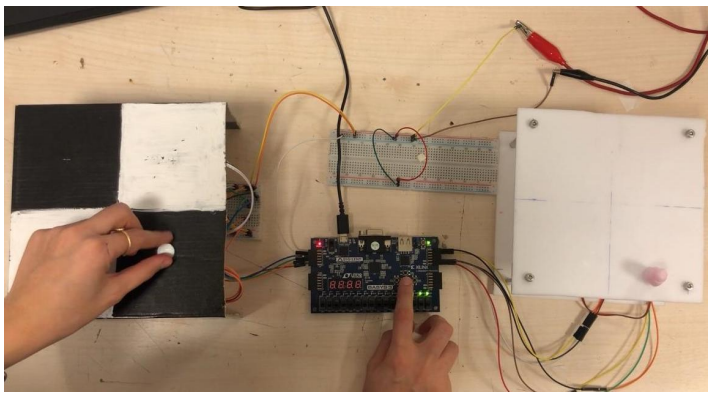


Fig.14 State C case

- When the magnet is on the fourth square:
  State: D
  pin1= “1”, pin2 = “1”, pin3 = “1”, pin4 = “0”
  LED output = “100”
  Servo1<= 90 degrees turn
  Servo2 <= 90 degrees turn

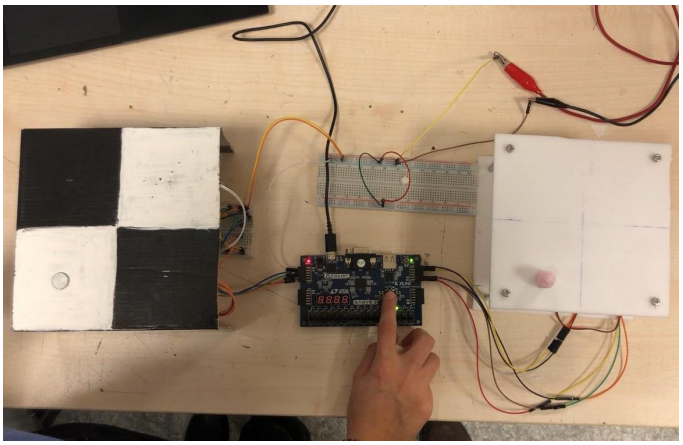


Fig.15 State D case

Conclusion

In this project signals coming from the sensors are processed via VHDL design. The enable button allowed the communication between the transmitter part and the reciever part. That way, another state E (the state of no signal) is erased from the design. To further develop this project the position detect part can be applied both sides so that a two sided communication can be achieved which will create a playable smart chess board but with 4 squares. Also the communication process can be done wirelessly.

References

Ramos, C. A. (2012, December 20). *Servomotor control with PWM and VHDL*. CodeProject. Retrieved December 24, 2022, from https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL

https://www.google.com/search?q=servo+motor+cntrol&oq=servo+motor+cntrol&aqs=chrome..69i57j0i13i512l3j0i13i30l6.6866j0j4&sourceid=chrome&ie=UTF-8

Appendix

Encoder.vhd

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

entity Encoder is
   Port (
        pin1 : in STD_LOGIC;
        pin2 : in STD_LOGIC;
        pin3 : in STD_LOGIC;
        pin4 : in STD_LOGIC;
        output : out std_logic_vector(2 downto 0));
end Encoder;

architecture enc_bhv of Encoder is
begin
process ( pin1,pin2,pin3,pin4)
begin
IF pin1 = '0' and pin2= '1'and pin3= '1'and pin4= '1' then
output <= "001";
elsif  pin1 = '1' and pin2= '0'and pin3= '1'and pin4= '1' then
output <= "010";
ELSIF pin1 = '1' and pin2= '1'and pin3= '0'and pin4= '1' then
output <= "011";
elsif pin1 = '1' and pin2= '1'and pin3= '1'and pin4= '0' then
output <= "100";
else
output <= "000";




end if;
end process;
```

```
end enc_bhv;
```

ClockDivider.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------------
entity servo_clkdiv is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end servo_clkdiv;
----------------------------------------------------------------------
architecture Behavioral of servo_clkdiv is
    signal temporal: STD_LOGIC;
    signal counter : integer range 0 to 1560 := 0;
begin
    ----------------------------------------------------------------------
    freq_divider: process (reset, clk) begin
        if (reset = '1') then
            temporal <= '0';
            counter <= 0;
        elsif rising_edge(clk) then
            if (counter = 1560) then
                temporal <= NOT(temporal);
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
    ----------------------------------------------------------------------
    clk_out <= temporal;
    ----------------------------------------------------------------------
end Behavioral;
```

PWM.vhd

```
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;
        pos : IN STD_LOGIC_VECTOR(6 downto 0);
        servo : OUT STD_LOGIC
    );
end servo_pwm;
----------------------------------------------------------------------
architecture Behavioral of servo_pwm is
```

```
    signal cnt : unsigned(10 downto 0);
    signal pwmi: unsigned(7 downto 0);

begin
    ----------------------------------------------------------------------
    pwmi <= unsigned('0' & pos) + 32;
    contador: process (reset, clk) begin
        if (reset = '1') then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            if (cnt = 1279) then
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;
    ----------------------------------------------------------------------
    servo <= '1' when (cnt < pwmi) else '0';
    ----------------------------------------------------------------------
end Behavioral;
```

FSM.vhd

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity FSM is
    Port (clk : in STD_LOGIC;
          enable: in STD_LOGIC;
          reset : in STD_LOGIC;
          pin1 : in STD_LOGIC;
          pin2 : in STD_LOGIC;
          pin3 : in STD_LOGIC;
          pin4 : in STD_LOGIC;
          servo1 : out std_logic;
          servo2 : out std_logic;
          now : out STD_LOGIC_vector(2 downto 0)
         );
end FSM;

architecture Behavioral of FSM is
    TYPE State_type IS (A,B,C,D);
    SIGNAL st,next_st : State_type;
    signal servo1_rotate,servo2_rotate : std_logic_vector( 6 downto 0);
    signal next_servo1, next_servo2: std_logic_vector( 6 downto 0);
    signal o : std_logic_vector(2 downto 0);


begin
```

```
enc: entity work.encoder(enc_bhv)
   port map(pin1 => pin1, pin2=> pin2, pin3 => pin3, pin4 => pin4, output => o);

SERVO_1: entity work.servo_main(Behavioral)
   port map(clk => clk,
            reset => reset,
            pos => next_servo1,
            servo => servo1);

SERVO_2: entity work.servo_main(Behavioral)
   port map(clk => clk,
            reset => reset,
            pos => next_servo2,
            servo => servo2);


process(clk,reset)
begin
   IF (reset = '1' and enable = '1') THEN
      st <= A;
      next_servo1 <= "0000000";
      next_servo2 <= "0000000";
      now <= "001";


   ELSIF enable = '1' and(Clk = '1' AND Clk'EVENT) then
      st <= Next_st;
      now <= o;
      next_servo1 <= servo1_rotate;
      next_servo2 <= servo2_rotate;

   END IF;
END PROCESS;


process (st,o)
begin
   Case st Is

      when A =>
         servo1_rotate <= "0000000"; --0--
         servo2_rotate <= "0000000"; --0--

         if o = "001" then --1--
            next_st<= A;



         elsif o = "010" then --2 --
            next_st<= B;



         elsif o = "011"  then --3--
```

```
        next_st<= C;


    elsif o = "100"  then --4--
        next_st<= D;



    end if;

when B =>
    servo1_rotate <= "0100000"; --90--
    servo2_rotate <= "0000000";--0--

    if o = "001"  then
        next_st<= A;


    elsif o = "010"  then
        next_st<= B;


    elsif o = "011"  then
        next_st<= C;


    elsif o = "100"  then
        next_st<= D;


    end if;
when c =>
    servo1_rotate <= "0110000"; --134--
    servo2_rotate <= "0110000"; --134--
    if o = "001"  then
        next_st<= A;

    elsif o = "010"  then
        next_st<= B;


    elsif o = "011"  then
        next_st<= C;


    elsif o = "100"  then
        next_st<= D;


    end if;

when D =>
    servo1_rotate <= "0100000"; --90--
    servo2_rotate <= "0100000"; --90--
    if o = "001"  then
```

```
                next_st<= A;


            elsif o = "010"  then
                next_st<= B;



            elsif o = "011"  then
                next_st<= C;



            elsif o = "100"  then
                next_st<= D;




            end if;
        end case;
    end process;


end Behavioral;
```