EEE 102 DIGITAL CIRCUIT DESIGN
TERM PROJECT REPORT

Self-Controlled Aircraft

Satılmış Furkan Kahraman
22002075
EEE 102-01

Satılmış Furkan Kahraman EE102 Section: 01 ID: 22002075 22.12.2022

Video link is here https://www.youtube.com/watch?v=oS8ZgEkUb0E&t=105s

Digital Term Project Report : Self-controlled Aircraft

Objective

Today, vehicles with autonomous features come to the fore in every field. That's why I designed an airplane for my digital project to autonomously balance itself in the air. In order to provide this feature, I connected the wing controls to the servo motors on the F-22 model I made. The servos are controlled by the data received from the gyro sensor via the Basys-3. In this way, there will be no need for constant control while the aircraft is moving on a straight route in the air. In case of any angle between the route of the aircraft and the elevator wings, the gyro sensor will detect this change and reset the angle by making the necessary arrangements on the aircraft. I also added some stimulating tools for the pilot in addition to this in my project. For example, if the pilot enters the "stall" position, where the engine power of the aircraft does not provide enough thrust, therefore there is not enough lift on the wings, the VGA is warned with the red color lit on the screen.
Video's link is here:

Methodology

First, I started the design by preparing a schematic plan at Figure 1.

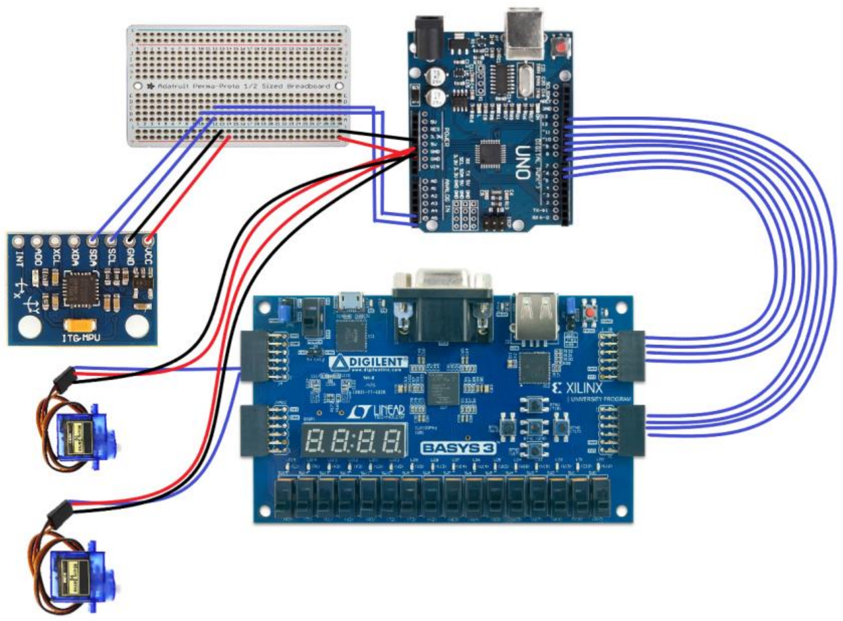

Figure 1 – Plan of the design

After completing the design, the project construction started. In order to present the project, I made a model airplane copying the F22 model for the design. The elevator wings of this model aircraft were suitable to move up and down depending on the servo motors. Since I could only move the elevator wings, the uncontrolled roll movement of the plane could be prevented. Then, the integration of the sub-modules that will form the circuit was started. First, data was received from the gyro sensor and the data was made ready for processing. These received data were then sent to the Basys-3 for processing. According to the processed data result, the PWM signals were sent to the servo motors and the blades were moved to the desired angles. In addition to this system, a VGA module was also written. Thanks to this module, a visual warning was given on the screen in case these movements, other than the roll movement, during autonomous flight, which cannot be controlled by the system, put the flight of the aircraft at risk. In aviation, this risky situation is called "Stall". In the designed circuit, the pilot will be notified by flashing a red color on the screen whether the aircraft is in stall or not. The sub-modules used in the circuit are as follows, in order:

MPU6050 3-Axis Gyro Sensor

In the circuit, MPU 6050 (Figure 2) sensor is used to obtain data that will determine the movements of the aircraft. MPU6050 has 2 data outputs, SDA and SCL. SCL acts as a clock on the sensor and determines the frequency of the transmitted signal. SDA transmits data in packets over the I2C protocol in accordance with the clock frequency. Since I could not write the I2C protocol with VHDL codes during the construction of the circuit, I used Arduino Uno to receive the data with parallel communication instead of serial. After completing the connections of the MPU6050 with the arduino, I analyzed the incoming data. 15 bits of data was coming from the sensor and values ranging from -16384 to 16383 were coming to the monitor. I divided this incoming data into 16 ranges and turned it into 4-bit data.

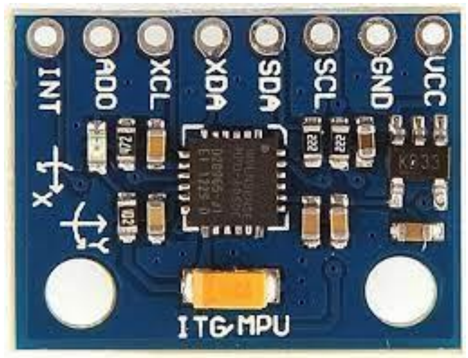

Figure 2 – MPU6050 Gyro Sensor

Arduino UNO Board

I used the data from the Arduino UNO (Figure 3) board sensor because I got an error while writing the serial communication protocol. Instead, I provided parallel communication using Arduino, which is in communication protocols. I sent the data received on the X and Y axis to Basys 3 in sets of 4 bits but here I got an error in this step. Since the Arduino UNO's digital output pins send a 5V signal, I had to reduce it to around 3.3V using a voltage divider. I reduced each incoming signal to 3.3V using 10KΩ and 22KΩ resistors.

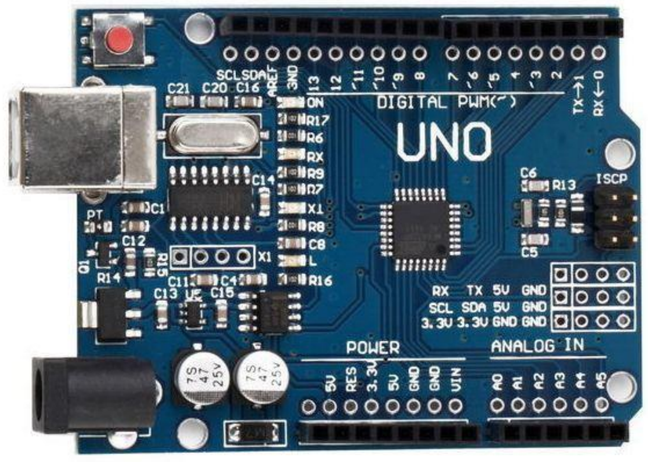

Figure 3 – Arduino UNO

BASYS 3

The Basys 3 FPGA card (Figure 4) is used to implement the logic circuit that will process the data received in the circuit. The results obtained according to the processed data are used to properly operate the modules connected to the card. In the circuit, 4-bit data from Arduino was received as input via PMOD. It processes the incoming data in a few submodules I wrote with VHDL in Vivado and sends it to different components at different outputs. When the data coming from the Y axis is processed and it is determined that certain value ranges are exceeded, a red light is lit on the VGA screen. The data coming from the X axis is processed and the servo motors change their positions at every angle value. In addition, the 4 LEDs on the Basys-3 light up and dim to represent the binary number system in order to see the size of the incoming data.

SG90 Servo Motor

2 SG90 servo motors (Figure 4) will be used on the model. These servos move opposite each other at the same angles. For this, I coded two different modules that provide movement in the opposite direction within the VHDL code. The 5V required for the servo motors to work is provided via the 5V output of the Arduino. The 3.3 V signal from Basys3 was enough for the servo motor to move. PWM signal was used to control the servos. Servo motors work with a frequency of 50 Hertz. Within a 20-millisecond cycle, it turns 0 degrees when 3.3V comes in 1 millisecond and 180 degrees when 3.3V comes in 2 milliseconds. All angles in between can be obtained by varying the positive voltage incoming time. I used 16 different angles on the servos since basys-3 gets four bits of data from the gyro. I determined the times for each angle and implied it in the VHDL code.

Figure 4 – SG90 Servo Motor

VGA Screen

In the event that the slope data obtained from VGA (Figure 5)goes out of the specified ranges, it will be used as a warning element for the pilot and passengers. The only intended application on VGA is to glow red when entering "stall" mode. I considered the refresh rate screen size while writing the VGA module.

Figure 5 – VGA Screen

Results

I used 3 different submodules to design this system. One of these submodules, the VGA module, has 4 different submodules in itself. In Figure 6, the complicated RTL schematics is shown.

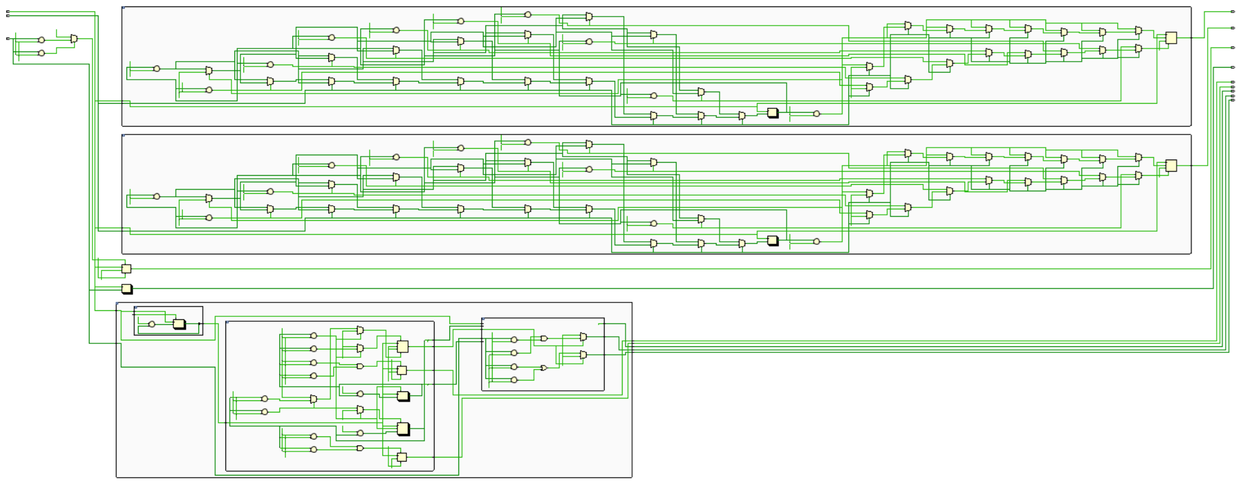

Figure 6 – Complicated RTL Schematics

In Figure 8, simplified RTL schematic is shown. In Figure 7, inputs and output can be clearly seen. There are position input which is equal to x-axis angle and gyroY which is equal to y-axis angle. Also there are servo_l and servo_r output which operate the angle of servos. stall_out is used to light a LED to understand if the aircraft is in stall position or not. Led outputs are used to show the angle of aircraft on the x-axis so we can easily understand that the incline value. VGA outputs are used to display red color on the VGA screen.

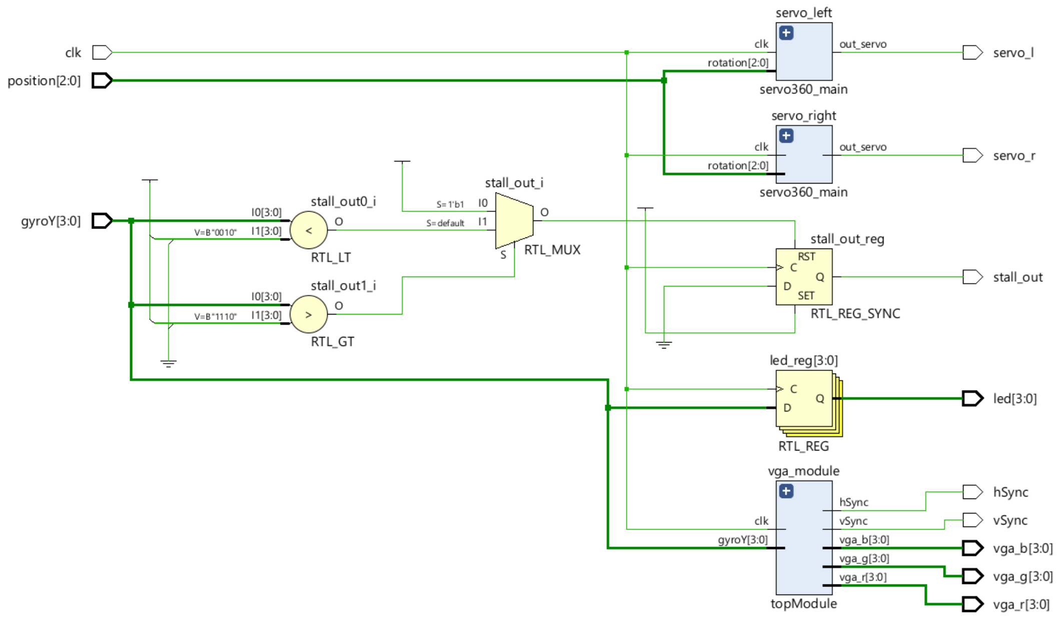

Figure 7 – Simplified RTL Schematics

Conclusion

In this project, I understood better how modules and tools such as gyro sensor, servo motors and VGA Screen are used together in a project. Thanks to the VHDL language, I coded the logic gates of all these components on Basys-3 and made them work as expected. While trying to solve the errors I encountered in this process, I became more inclined to examine datasheets, use oscilloscopes and multimeters. I have experience in creating a finite state machine on VHDL. I've become more familiar with some of the uses within VHDL. In the following, there are some photos of my projects.

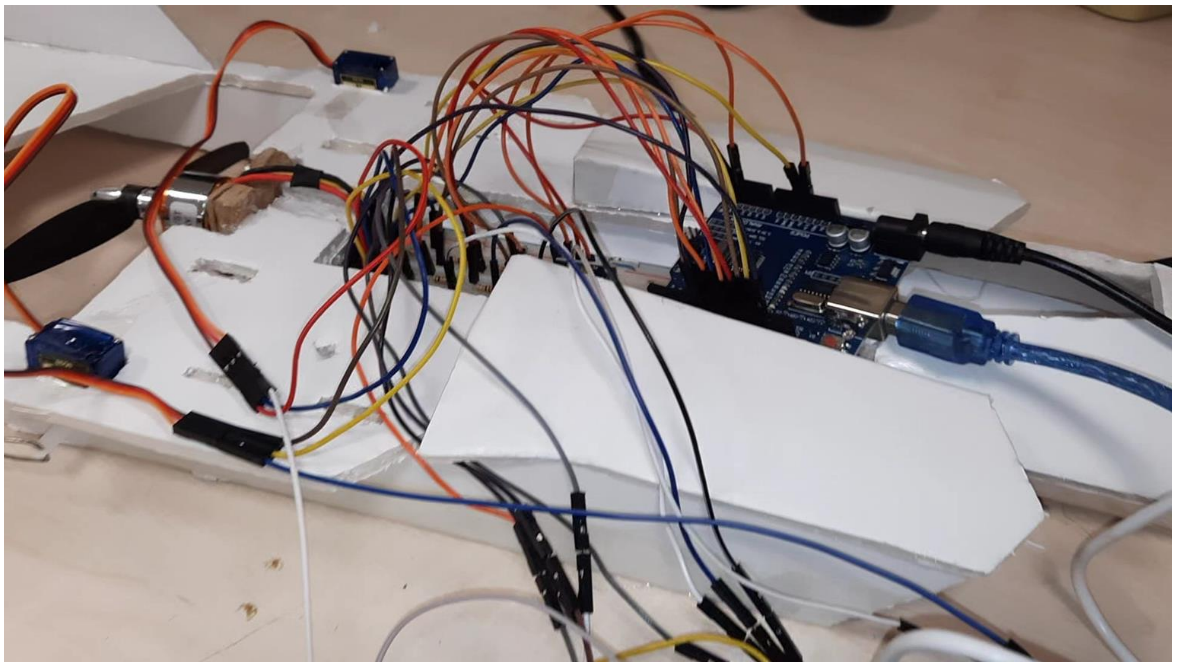
Figure 8 – Arduino and gyro connection

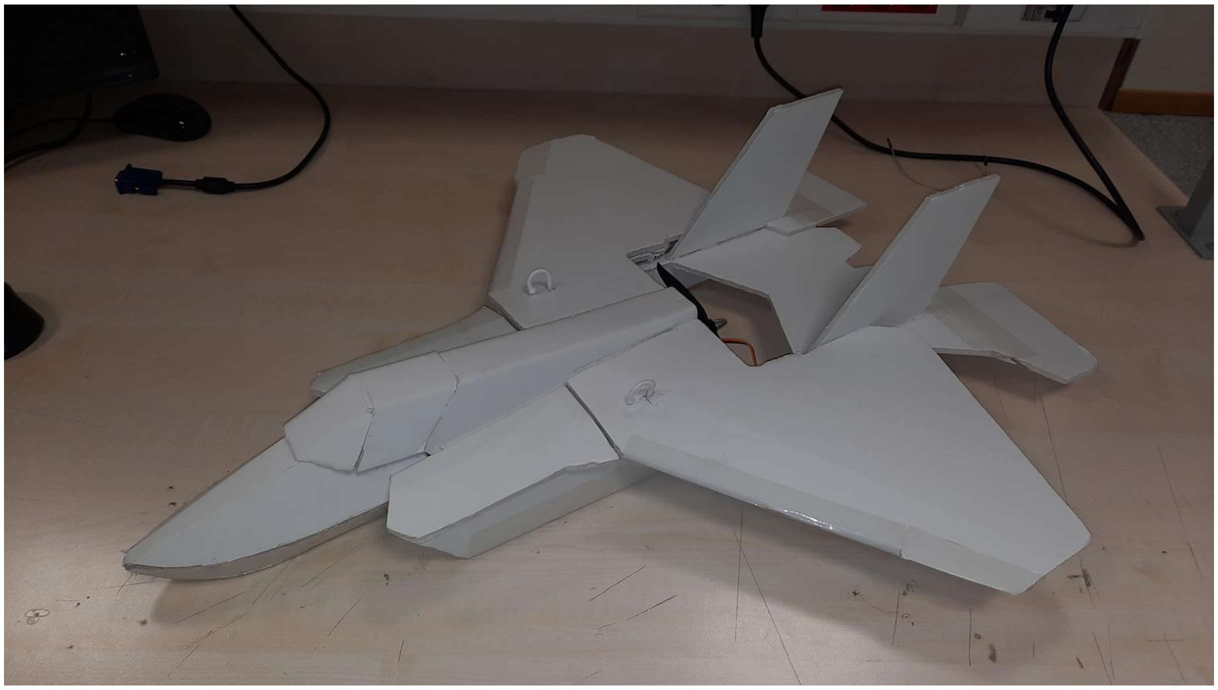
Figure 9 – Model Aircraft

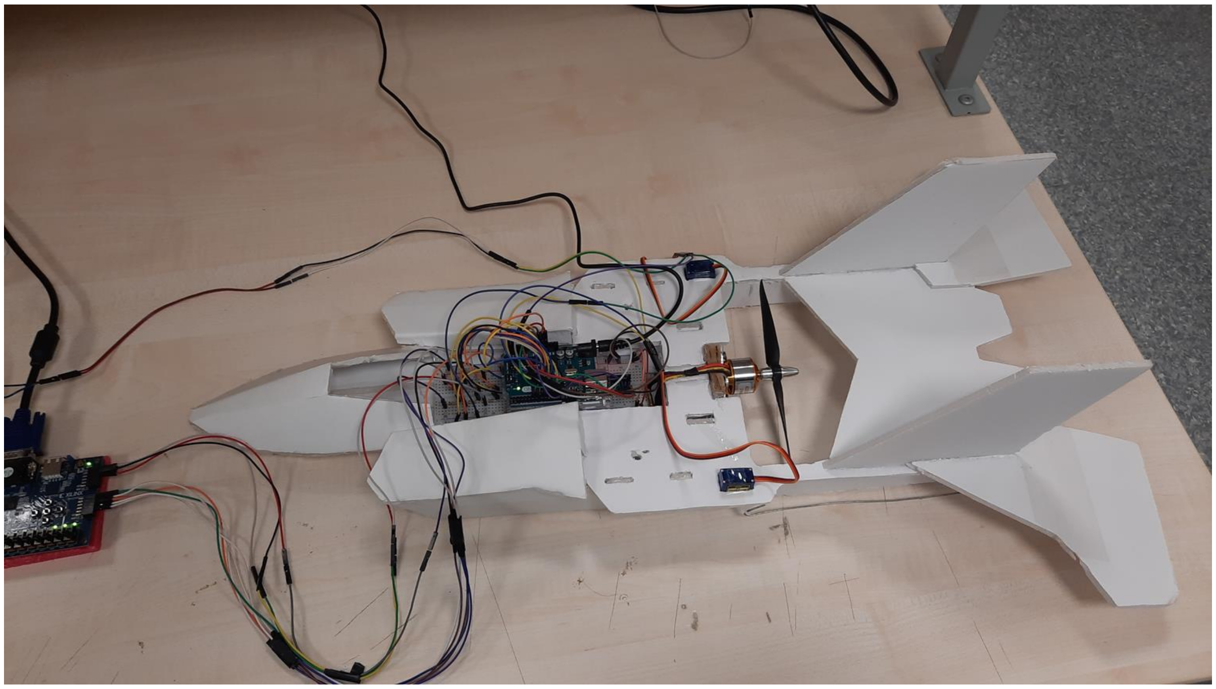
Figure 10 – Connection on the aircraft

Appendices

VHDL Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

-- define inputs and outputs
--position values are equal to x-axis values coming from gyro sensor

entity top is
    Port(clk : in std_logic;
         position : in std_logic_vector(2 downto 0);
         gyroY : in std_logic_vector(3 downto 0);
         stall_out : out std_logic;
         servo_r : out std_logic;
         servo_l : out std_logic;
         hSync, vSync: out std_logic;
         vga_r, vga_g, vga_b : out std_logic_vector(3 downto 0);
         led : out std_logic_vector(3 downto 0)
    );
end top;

architecture Behavioral of top is

--VGAtop component is VGA module's top module

component VGAtop is
Port ( clk : in STD_LOGIC;
       gyroY : in std_logic_vector(3 downto 0);
       hSync : out STD_LOGIC;
       vSync : out STD_LOGIC;
       vga_red : out STD_LOGIC_vector(3 downto 0);
       vga_green : out STD_LOGIC_vector(3 downto 0);
       vga_blue : out STD_LOGIC_vector(3 downto 0));
end component;

--right and left servo motors are connected different module because their direction of rotations are
--different
--servo360_main is for left servo motor
component servo360_main is
Port (clk: in std_logic;
      rotation: in std_logic_vector(2 downto 0);
      out_servo: out std_logic);
end component;

--servo360_main_opposite is for right servo motor

component servo360_main_opposite is
Port (clk: in std_logic;
      rotation: in std_logic_vector(2 downto 0);
      out_servo: out std_logic);
end component;


begin

servo_left  : servo360_main Port Map(clk => clk, rotation => position, out_servo => servo_l);
servo_right : servo360_main_opposite Port Map(clk => clk, rotation => position, out_servo => servo_r);
vga_module  : topModule Port Map(clk => clk, gyroY => gyroY , vsync => vSync, hsync => hSync,
vga_red=> vga_red,  vga_green => vga_green,  vga_blue => vga_blue );

--The following code indicates whether the airplane is in stall or not.

process(clk) begin
    if rising_edge(clk) then
        if (gyroY > "1110") then
            stall_out <= '0';
        elsif (gyroY < "0010") then
            stall_out <= '0';
        else
            stall_out <= '1';
        end if;
    end if;
end process;





--The following code shows the gyroY values with LEDs

process(clk) begin
    if rising_edge(clk) then
```

```
        if gyroY(0) = '1' then
            led(0) <= '1';
        else
            led(0) <= '0';
        end if;
        if gyroY(1) = '1' then
            led(1) <= '1';
        else
            led(1) <= '0';
        end if;
        if gyroY(2) = '1' then
            led(2) <= '1';
        else
            led(2) <= '0';
        end if;
        if gyroY(3) = '1' then
            led(3) <= '1';
        else
            led(3) <= '0';
        end if;
     end if;
 end process;

end Behavioral;
```

Servo360_main

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
```

```
entity servo360_main is
Port (clk: in std_logic;
      rotation: in std_logic_vector(2 downto 0);
      out_servo: out std_logic);
end servo360_main;

architecture Behavioral of servo360_main is

--define the pwm signal

signal pwmcounter: integer:=0;

begin

process(rotation,clk,pwmcounter) begin
if rising_edge(clk) then
pwmcounter<=pwmcounter+1;

--every angle value has a pwm value in the folloing code

    if rotation="000" then

        if pwmcounter<100000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="001" then

        if pwmcounter<115000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="010" then
        if pwmcounter<130000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="011" then

        if pwmcounter<145000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="100" then

        if pwmcounter<160000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="101" then

        if pwmcounter<175000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="110" then

        if pwmcounter<190000 then
```

```
          out_servo<='1';
       elsif pwmcounter<2000000 then
          out_servo<='0';
       else
          pwmcounter<=0;
       end if;

   elsif  rotation ="111" then

       if pwmcounter<200000 then
          out_servo<='1';
       elsif pwmcounter<2000000 then
          out_servo<='0';
       else
          pwmcounter<=0;
       end if;
   end if;

end if;
end process;

end Behavioral;
```

Servo360_main_opposite

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity servo360_main_opposite is
Port (clk: in std_logic;
      rotation: in std_logic_vector(2 downto 0);
      out_servo: out std_logic);
end servo360_main_opposite;

architecture Behavioral of servo360_main_opposite is

--define the pwm signal

signal pwmcounter: integer:=0;

begin

process(rotation,clk,pwmcounter) begin
if rising_edge(clk) then
pwmcounter<=pwmcounter+1;

    if rotation="111" then

        if pwmcounter<100000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="110" then

        if pwmcounter<115000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="101" then

        if pwmcounter<130000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="100" then

        if pwmcounter<145000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="011" then

        if pwmcounter<160000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="010" then

        if pwmcounter<175000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif rotation ="001" then

        if pwmcounter<190000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    elsif  rotation ="000" then

        if pwmcounter<200000 then
            out_servo<='1';
        elsif pwmcounter<2000000 then
            out_servo<='0';
        else
            pwmcounter<=0;
        end if;

    end if;

end if;
end process;

end Behavioral;
```

VGA_top

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

entity VGA_top is

Port ( clk : in STD_LOGIC;
       stall : in std_logic;
       gyroY : in std_logic_vector(3 downto 0);
       hSync : out STD_LOGIC;
       vSync : out STD_LOGIC;
       vga_red : out STD_LOGIC_vector(3 downto 0);
       vga_green : out STD_LOGIC_vector(3 downto 0);
       vga_blue : out STD_LOGIC_vector(3 downto 0));

end VGA_top;

architecture Behavioral of VGA_top is

component VGA
   Port(   clk : in std_logic;
            hSync : out std_logic;
            vSync : out std_logic;
            displayEn : out std_logic;
            row: out integer;
            column: out integer);
end component;


component clock_divider
   Port (  clk : in std_logic;
           reset : in std_logic;
           clockOut : out std_logic);
end component;


component color_generator
     Port(  clk : in std_logic;
            gyroY : in std_logic_vector(3 downto 0);
            disp_ena : in std_logic;
             row : in integer ;
             column : in integer;
              VGA_R : out std_logic_vector(3 downto 0) := (others => '0');
              VGA_G : out std_logic_vector(3 downto 0) := ( others => '0');
              VGA_B : out std_logic_vector(3 downto 0) := ( others => '0'));
end component;

signal sig_clock : std_logic;
signal row: integer;
signal reset: std_logic;
signal display_enable : std_logic;
signal column: integer;
```

```
begin


module1 : clock_divider port map (clk => clk , reset => reset , sig_clock => sig_clock);
module2 : VGA port map( sig_clock => clk , hSync => hSync , vSync => vSync, display_enable =>
display_enable , row => row, column=> column);
module3 : color_generator port map (clk => clk, gyroY=> position, display_enable =>disp_enable , row
=> row, column => column , VGA_R => vga_red , VGA_G => vga_green, VGA_B => vga_blue);

end Behavioral;
```

clock_divider

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

entity clock_divider is

Port  ( clk : in std_logic;
        reset : in std_logic;
        clock_out : out std_logic);
end clock_divider;
```

```
architecture Behavioral of clock_divider is

signal clockout : std_logic_vector(1 downto 0);

begin

process(clk)

begin

        if(reset = '1') then
                clockout <= (others => '0');

        elsif(rising_edge(clk)) then
                clockout <= clockout + 1;

        end if;

end process;

clockout <= clock_out(0);

end Behavioral;
```

VGA

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity VGA is

Port (  clk : in std_logic;
      hSync : out  std_logic;
      vSync : out  std_logic;
      display_enable : out  std_logic;
      row : out integer;
```

```
    column : out integer);
end VGA;

architecture Behavioral of VGA is

signal Hrfrsh: std_logic_vector(9 downto 0):="0000000000";
signal Vrfrsh: std_logic_vector(9 downto 0):="1000001000";

constant HM: integer :=800;   --horizontal
constant HD: integer :=640;
constant HF: integer :=16;  -- horizontal wait time
constant HB: integer :=48;  -- horizontal wait time
constant HR: integer :=96;  -- horizontal wait time

constant VM: integer :=540;    --vertical
constant VD: integer :=480;
constant VF: integer :=10;  -- vertical wait time
constant VB: integer :=33;  -- vertical wait time
constant VR: integer :=2;   -- vertical wait time
begin

process(clk)

begin

if (clk'event and clk='1') then
  if (Hrfrsh = HM-1) then
    Hrfrsh <= "0000000000";

    if (Vrfrsh= VM-1) then
      Vrfrsh <= "0000000000";
      display_enable <= '1';
    else
      if Vrfrsh < VD-1 then
        display_enable <= '1';
      end if;
      Vrfrsh <= Vrfrsh+1;
    end if;
    else
      if Hrfrsh = HD-1 then
      displayEn <= '0';
      end if;
```

```
            Hrfrsh <= Hrfrsh + 1;
        end if;
    end if;
end process;


process(clk)
begin
    if (clk'event and clk='1') then
        if ( Hrfrsh >= (HD+HF) and Hrfrsh <= (HD+HF+HR-1)) then
            hSync <= '0';
        else
            hSync <= '1';
        end if;
    end if;
end process;

process(clk)
begin
if (clk'event and clk='1') then
if ( Vrfrsh >= (VD+VF) and Vcnt <= (VD+VF+VR-1)) then --- Vrfrsh >= 799 and vcnt<= 539
vSync <= '0';
else

vSync <= '1';


end if;
end if;
end process;
row <= conv_integer( unsigned(Hrfrsh));
column <= conv_integer( unsigned(Vrfrsh));
end Behavioral;
```

color_generator

```
library ieee;
use ieee.std_logic_1164.all;

entity color_generator is

Generic( y_axis : integer := 800;
              x_axis : integer := 540);

Port(   clk : in std_logic;
        gyroy : in std_logic_vector(3 downto 0);
        disp_enable : in std_logic;
        row : in integer;
        column : in integer;
        vga_red : out std_logic_vector(3 downto 0) := (others => '0');
        vga_green : out std_logic_vector(3 downto 0) := (others => '0');
        vga_blue : out std_logic_vector(3 downto 0) := (others => '0'));
end color_generator;

architecture Behavioral of color_generator is

begin

process(disp_enable, row, column, clk)

begin

if (disp_enable = '1') then

 if (gyroY > "1110" or gyroY < "0010" ) then
        if (row < y_axis and column < x_axis) then
        VGA_R <= (others => '1');

        VGA_G <= (others => '0');

        VGA_B <= (others => '0');
        end if;

 elsif (gyroY > "0001" and gyroY < "1111") then

       if (row < y_axis and column < x_axis) then

       VGA_R <= (others => '0');

       VGA_G <= (others => '1');

       VGA_B <= (others => '0')
       end if;
    end if;
end process;
end Behavioral;
```

Constraint Code

```
set_property PACKAGE_PIN W5 [get_ports clk]
 set_property IOSTANDARD LVCMOS33 [get_ports clk]

set_property PACKAGE_PIN U16 [get_ports {led[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]

set_property PACKAGE_PIN A14 [get_ports {position[0]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {position[0]}]
set_property PACKAGE_PIN A16 [get_ports {position[1]}]
  set_property IOSTANDARD LVCMOS33 [get_ports {position[1]}]
set_property PACKAGE_PIN B15 [get_ports {position[2]}]
```

```
  set_property IOSTANDARD LVCMOS33 [get_ports {position[2]}]

set_property PACKAGE_PIN K17 [get_ports {gyroY[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {gyroY[0]}]
set_property PACKAGE_PIN M18 [get_ports {gyroY[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {gyroY[1]}]
set_property PACKAGE_PIN N17 [get_ports {gyroY[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {gyroY[2]}]
set_property PACKAGE_PIN P18 [get_ports {gyroY[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {gyroY[3]}]


set_property PACKAGE_PIN J1 [get_ports servo_l]
    set_property IOSTANDARD LVCMOS33 [get_ports servo_l]
set_property PACKAGE_PIN L2 [get_ports servo_r]
    set_property IOSTANDARD LVCMOS33 [get_ports servo_r]
set_property PACKAGE_PIN J2 [get_ports stall_out]
    set_property IOSTANDARD LVCMOS33 [get_ports stall_out]

set_property PACKAGE_PIN G19 [get_ports {vga_r[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[0]}]
set_property PACKAGE_PIN H19 [get_ports {vga_r[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[1]}]
set_property PACKAGE_PIN J19 [get_ports {vga_r[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[2]}]
set_property PACKAGE_PIN N19 [get_ports {vga_r[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[3]}]


set_property PACKAGE_PIN N18 [get_ports {vga_b[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[0]}]
set_property PACKAGE_PIN L18 [get_ports {vga_b[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[1]}]
set_property PACKAGE_PIN K18 [get_ports {vga_b[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[2]}]
set_property PACKAGE_PIN J18 [get_ports {vga_b[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[3]}]

set_property PACKAGE_PIN J17 [get_ports {vga_g[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[0]}]
set_property PACKAGE_PIN H17 [get_ports {vga_g[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[1]}]
set_property PACKAGE_PIN G17 [get_ports {vga_g[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[2]}]
set_property PACKAGE_PIN D17 [get_ports {vga_g[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[3]}]


set_property PACKAGE_PIN P19 [get_ports hSync]
    set_property IOSTANDARD LVCMOS33 [get_ports hSync]
set_property PACKAGE_PIN R19 [get_ports vSync]
    set_property IOSTANDARD LVCMOS33 [get_ports vSync]
```

ARDUINO Code

```
#include<Wire.h>

#define MPU 0x68

int16_t ivmeY, ivmeX;
int X1, X2, X3, X4;
int Y1, Y2, Y3, Y4;

void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
  pinMode(X1, OUTPUT);
  pinMode(X2, OUTPUT);
  pinMode(X3, OUTPUT);
  pinMode(X4, OUTPUT);
  pinMode(Y1, OUTPUT);
  pinMode(Y2, OUTPUT);
  pinMode(Y3, OUTPUT);
  pinMode(Y4, OUTPUT);

}
void loop(){

   verileriOku();

   if(ivmeX > 15750){
   X1 = 1;
   X2 = 1;
   X3 = 1;
   X4 = 1;
   }
   else if(ivmeX > 13500) {
   X1 = 0;
   X2 = 1;
   X3 = 1;
   X4 = 1;
   }
   else if(ivmeX > 11250) {
   X1 = 1;
   X2 = 0;
   X3 = 1;
   X4 = 1;
   }
   else if(ivmeX > 9000) {
   X1 = 0;
   X2 = 0;
   X3 = 1;
   X4 = 1;
   }
   else if(ivmeX > 6750) {
   X1 = 1;
   X2 = HIGH;
   X3 = LOW;
   X4 = HIGH;
   }
   else if(ivmeX > 4500) {
```

```
X1 = LOW;
X2 = HIGH;
X3 = LOW;
X4 = HIGH;
}
else if(ivmeX > 2250) {
X1 = HIGH;
X2 = LOW;
X3 = LOW;
X4 = HIGH;
}
else if(ivmeX > 0) {
X1 = LOW;
X2 = LOW;
X3 = LOW;
X4 = HIGH;
}
else if(ivmeX > -2250){
X1 = HIGH;
X2 = HIGH;
X3 = HIGH;
X4 = LOW;
}
else if(ivmeX > -4500) {
X1 = LOW;
X2 = HIGH;
X3 = HIGH;
X4 = LOW;
}
else if(ivmeX > -6750) {
X1 = HIGH;
X2 = LOW;
X3 = HIGH;
X4 = LOW;
}
else if(ivmeX > -9000) {
X1 = LOW;
X2 = LOW;
X3 = HIGH;
X4 = LOW;
}
else if(ivmeX > -11250) {
```

```
X1 = HIGH;
X2 = HIGH;
X3 = LOW;
X4 = LOW;
}
else if(ivmeX > -13500) {
X1 = LOW;
X2 = HIGH;
X3 = LOW;
X4 = LOW;
}
else if(ivmeX > -15750) {
X1 = HIGH;
X2 = LOW;
X3 = LOW;
X4 = LOW;
}
else if(ivmeX > -18500) {
X1 = LOW;
X2 = LOW;
X3 = LOW;
X4 = LOW;
}

if(ivmeY > 15750){
Y1 = HIGH;
Y2 = HIGH;
Y3 = HIGH;
Y4 = HIGH;
}
else if(ivmeY > 13500) {
Y1 = LOW;
Y2 = HIGH;
Y3 = HIGH;
Y4 = HIGH;
}
else if(ivmeY > 11250) {
Y1 = HIGH;
Y2 = LOW;
Y3 = HIGH;
Y4 = HIGH;
}
```

```
else if(ivmeY > 9000) {
Y1 = LOW;
Y2 = LOW;
Y3 = HIGH;
Y4 = HIGH;
}
else if(ivmeY > 6750) {
Y1 = HIGH;
Y2 = HIGH;
Y3 = LOW;
Y4 = HIGH;
}
else if(ivmeY > 4500) {
Y1 = LOW;
Y2 = HIGH;
Y3 = LOW;
Y4 = HIGH;
}
else if(ivmeY > 2250) {
Y1 = HIGH;
Y2 = LOW;
Y3 = LOW;
Y4 = HIGH;
}
else if(ivmeY > 0) {
Y1 = LOW;
Y2 = LOW;
Y3 = LOW;
Y4 = HIGH;
}
else if(ivmeY > -2250){
Y1 = HIGH;
Y2 = HIGH;
Y3 = HIGH;
Y4 = LOW;
}
else if(ivmeY > -4500) {
Y1 = LOW;
Y2 = HIGH;
Y3 = HIGH;
Y4 = LOW;
}
```

```
else if(ivmeY > -6750) {
Y1 = HIGH;
Y2 = LOW;
Y3 = HIGH;
Y4 = LOW;
}
else if(ivmeY > -9000) {
Y1 = LOW;
Y2 = LOW;
Y3 = HIGH;
Y4 = LOW;
}
else if(ivmeY > -11250) {
Y1 = HIGH;
Y2 = HIGH;
Y3 = LOW;
Y4 = LOW;
}
else if(ivmeY > -13500) {
Y1 = LOW;
Y2 = HIGH;
Y3 = LOW;
Y4 = LOW;
}
else if(ivmeY > -15750) {
Y1 = HIGH;
Y2 = LOW;
Y3 = LOW;
Y4 = LOW;
}
else if(ivmeY > -18500) {
Y1 = LOW;
Y2 = LOW;
Y3 = LOW;
Y4 = LOW;
}

Serial.print(" X1 = "); Serial.print(X1);
Serial.print(" X2 = "); Serial.print(X2);
Serial.print(" X3 = "); Serial.print(X3);
Serial.print(" X4 = "); Serial.print(X4);
Serial.print(" Y1 = "); Serial.print(Y1);
```

```
   Serial.print(" Y2 = "); Serial.print(Y2);
   Serial.print(" Y3 = "); Serial.print(Y3);
   Serial.print(" Y4 = "); Serial.print(Y4);
   Serial.print("  ivmeX = "); Serial.print(ivmeX);
   Serial.print(" | ivmeY = "); Serial.println(ivmeY);
   delay(100);
}

void verileriOku(){
  Wire.beginTransmission(MPU);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU,14,true);

  ivmeX=Wire.read()<<8|Wire.read()/1024;
  ivmeY=Wire.read()<<8|Wire.read()/1024;
  ivmeX=Wire.read()<<8|Wire.read()/1024;
  digitalWrite(13, X1);
  digitalWrite(12, X2);
  digitalWrite(11, X3);
  digitalWrite(10, X4);
  digitalWrite(9, Y1);
  digitalWrite(8, Y2);
  digitalWrite(7, Y3);
  digitalWrite(6, Y4);
  delay(50);
}
```