EEE102 INTRODUCTION TO DIGITAL DESIGN
TERM PROJECT FINAL REPORT

VHDL-Based Ultrasonic Weapon System

Name: Atalay Üstündağ

Number: 22203554

Section: 003

Instructor: Cem Tekin

1.Purpose of the Project:

The primary objective of the EEE102 term project is to develop and demonstrate vhdl skills in digital design. This project aims to make a vhdl based ultrasonic weapon system. Understanding the mechanism behind servo motors and ultrasonic system was another goal of this project. And after understanding these mechanisms, generating the correct vhdl codes for these devices was one of the biggest challenges in this project.

2. Design Spesifications:

The following components are used in the project:

-Basys3 FPGA Board

-SG90 Servo Motor

-MG 996R Servo Motor

-HCSR04 Ultrasonic Sensor

-Toy Gun

-Laboratory Power Supply

-Logic Level Converter

-Breadboard

-Jumper Cables

In this system ultrasonic sensor detects if there is an object or not in its range. And when there is an object in range, the trigger of the toy gun is pulled with SG90 servo motor. And this range can be arranged between 2 cm to 4 m. And all this weapon system (toy gun and sensor) can turn horizontaly with the help of MG996R servo motor when user pushes movement buttons on FPGA board. The reason why I used two different servo motors is because of the abilities of servo motors. MG996R servo motor is more expensive that SG90 servo motor and it is powerful enough to carry and turn my weapon system. On the other hand, SG90 servo motor is cheaper and powerful enough to pull my toy gun's trigger but not powerful enough to carry the system. And I had to use another voltage source other than the ports of my BASYS3's 3 Volts ports. Because HCSR04 ultrasonic sensor needed 5 volts to be active. Therefore I supplied 5 Volts from laboratory DC power supply. Then I used a logic level converter to connect HCSR04 and Basys3, this component prevented my fpga board from burning.

3. Methodology:

This project has 7 modules:

a) distance:

In this code, I made a clock divider. I did it because my basys3's internal clock is 100MHz. But HCSR04 sensor needs a minimum 10 us trigger impuls duration and a minimum of 60ms measurment cycle. In order to do it, I used a variable constant_trigger, and when this constant_trigger is greater than zero and less than 10000, my system is triggered. This 10000 value is equal to 100us and since it is greater than 10us it is enough. And when is between 10000 and 10000000, trigger is zero. Other than constant_trigger, the code utilizes constant_echo variable , and a signal echo_value to calculate and store the duration of an echo pulse in response to a transmitted ultrasonic signal. The variable determine_high is used as a flag to determine when the echo signal is high. The method regulates the trig signal to produce a pulse that activates the ultrasonic sensor. The echo pulse duration recorded in echo_value is used to calculate the distance information. The data output is assigned the corresponding 3-bit binary code for each of the three distance ranges that are taken into consideration. Distances less than 25,000 units fall into the first range; those between 25,000 (about 4.7 cm) and 280,000 (approximately 52 cm) units fall into the second range; and distances larger than 280,000 units fall into the third range. The distance is efficiently calculated by the code, which also expresses it in binary form. I can adjust my range up to 4 meters by changing these settings.

b) servo_clkdiv:

In this module, the process labeled "freq_divider" is sensitive to changes in the reset and clk signals. When the reset signal is asserted ('1'), the transient signal is set to '0', and the counter is reset to 0. On the rising edge of the input clock (clk), the process checks if the counter has reached 1560. If true, it toggles the transient signal (inverting its current value) and resets the counter to 0. Otherwise, it increments the counter. In essence, this process divides the input clock frequency by 1560, effectively creating a slower clock signal at the output (clk_out). Consequently, the output clock signal (clk_out) is a slower version of the input clock signal (clk), divided by a factor of 1560. The purpose of such a clock divider is to generate a clock signal with 65KHz frequency, because both of my servo motors work with 65KHz frequency.

c) servo_pwm:

This module creates an object named "servo_pwm" that generates a pulse-width modulated (PWM) signal, which is then used to operate my servo motors. The entity consists of four ports: servo is the output PWM signal that drives the servo; reset is the input reset signal; clk is the input clock signal; and seven_bit_position is the input signal that identifies the intended position of the servo motor. The two signals defined by the architectural block are signal_pwmm, which has an 8-bit length, and eleven_bit_cnt, which has an 11-bit length (ranging from 0 to 1279). The signal_pwmm signal is obtained by concatenating a '0' with the seven_bit_position input vector plus an additional offset of 32. Changes in the reset and clk signals stimulate the "contador" mechanism. When the reset signal ('1') is asserted, the eleven_bit_cnt signal is reset to zero. On each rising edge of the clock, the method checks to see if the input clock (eleven_bit_cnt) has reached 1279. If false, it raises eleven_bit_cnt; if true, it resets it to zero. This process essentially creates a counter with a range of 0 to 1279.The final line of the code sets the output port servo's value to '1' when eleven_bit_cnt is less than the signal_pwmm threshold and '0' otherwise. This determines the duty cycle of the conditionally generated PWM signal by comparing the counter value (eleven_bit_cnt) to the PWM threshold (signal_pwmm). My two servo motors' positions are controlled by the generated PWM signal by means of the proper position input (seven_bit_position). The angular position of the servo motor is determined by the duty cycle of the PWM signal.

d) servo_binary:

I decleared a number of signals in this code, including counter, which is an 11-bit counter, interim_servo_store, which is for interim storage, and current_input, which holds the current input data. Constants like DEGREE_180, DEGREE_134, DEGREE_90, DEGREE_45, and DEGREE_0 are defined for particular servo motor locations. Furthermore, a default position is represented by the constant reset_position . Based on the input current_input, the primary process "servo_proc" determines the output servo1 using a case statement. It designates the right constant value according to the given angular position. The servo is set to DEGREE_0 if the input is "000," DEGREE_45 if it is "001," and so on. When none of the conditions listed above apply, DEGREE_180 is the default position. In a nutshell, this code implements a simple servo motor controller that takes a 3-bit input representing different angular positions and outputs a 7-bit control signal for the servo motor based on the specified positions.

e) servo_button:

This module implements a digital circuit with three buttons on my basys3 for incrementing ,decrementing and resetting a binary counter to turn my servo motor at desired angles. My circuit has a reset condition and is clock edge sensitive. The binary counter is increased or decreased in accordance with the pressing of a button. A 3-bit vector receives the binary count that is currently displayed. When performing increment and decrement operations, the circuit makes sure the counter stays inside predetermined bounds. To avoid counting more than one button press, the button states are saved.

f) servo_main:

Two components are used in this module to construct a servo motor control system. A clock divider, which is the first part, converts the input clock into a lower-frequency clock signal which is 65KHz. The second part is a pulse-width modulator (PWM), which accepts a reset signal, a lower-frequency clock, and a 7-bit vector that indicates the desired position of the servo motor.

g) design_main:

Design_main module makes use of servo motor control, binary counting, button inputs, and distance measuring. Data is captured by the distance measuring module using ultrasonic sensor HCSR04. One servo motor (MG 996R) is controlled by button inputs, while the other one (SG90) is triggered for a predetermined amount of time when the distance measurement module produces a predetermined value. All in all, the code combines these elements to produce a system that can regulate the positions of two servos in response to button presses and distance measurements.

4. Youtube Video Link:

https://youtu.be/RSuo42x8pmE

5. Conclusion:

Finally, the project's goal was to build a vhdl based ultrasonic weapon system by using three main components which are a ultrasonic sensor and two servo motors. The hardest part in my project was activating the sensor. Because even when I was sure my codes were correct, I could not send trigger and take an echo signal from the sensor. Then I started the debugging process and it took a lot of time. At the end, I figured out that the problem was not about my vhdl codes, it was logic level converter. Even though I connected it with the right ports of Basys3 and right pins of sensor, it failed to establish connection between sensor and Basys3. Then I soldered the pins on converter and it worked. After I solved this problem, I managed to finish my project.

6. References:

https://github.com/CankutBoraTuncer/Bilkent-EEE102-Labs/blob/main/Term%20Project/ProjectReportEE102%20(1).pdf

7. Appendix:

Design_main:

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


entity design_main is
  Port (
    fpgaclk: in std_logic;
    reset1: in std_logic;
    echo: in std_logic;
    trig: buffer std_logic;
    left, right: in std_logic;
```

```
    servo11: out std_logic;
    servo22: out std_logic
  );
end design_main;

architecture Behavioral of design_main is
  signal servo_loc1, servo_loc2: std_logic_vector(6 downto 0);
  signal data1, data2: std_logic_vector(2 downto 0);
  signal servo_angle_counter: integer := 0;

begin

  distance_inst: entity work.distance(Behavioral)
    port map(fpgaclk => fpgaclk,
             echo => echo,
             trig => trig,
             data => data1);

  servo_ first_button: entity work.servo_button(Behavioral)
    port map(clk => fpgaclk,
             reset => reset1,
             first_button => right,
             second_button => left,
             position_choice => data2);
```

```
servo_binary1: entity work.servo_binary(Behavioral)
  port map(clk => fpgaclk,
           reset => reset1,
           input_dataa => data2,
           servo1 => servo_loc1);



SERVO_1: entity work.servo_main(Behavioral)
  port map(clk => fpgaclk,
           reset => reset1,
           seven_bit_position => servo_loc1,
           servo => servo11);

SERVO_2: entity work.servo_main(Behavioral)
  port map(clk => fpgaclk,
           reset => reset1,
           seven_bit_position => servo_loc2,
           servo => servo22);


process(fpgaclk)
begin
  if rising_edge(fpgaclk) then
    if data1 = "010" then
      if servo_angle_counter < 1000000 then
        servo_angle_counter <= servo_angle_counter + 1;
```

```
        servo_loc2 <= "0100000" ;
      else
        servo_angle_counter <= 0;
        servo_loc2 <= "0000000" ;
      end if;
    else
      servo_loc2 <= "0000000" ;
      servo_angle_counter <= 0;
    end if;
  end if;
end process;

end Behavioral;
```

distance:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity distance is
  Port (fpgaclk: in std_logic;
        trig: buffer std_logic;
        echo: in std_logic;
        data : buffer std_logic_vector
```

```
        (2 downto 0));
end distance;

architecture Behavioral of distance is

signal echo_value: integer:=0;

begin
process(fpgaclk, echo)
variable constant_trigger, constant_echo:integer:=0;
variable determine_high:std_logic:='0';
begin
if rising_edge(fpgaclk) then
if(constant_trigger =0) then
    trig<='1';
elsif(constant_trigger =10000) then
    trig<='0';
    determine_high:='1';
elsif(constant_trigger =10000000) then
    constant_trigger:=0;
    trig<='1';
end if;
constant_trigger:= constant_trigger +1;

if(echo = '1') then
```

```
    constant_echo:= constant_echo +1;
end if;

if(echo = '0' and determine_high ='1') then
    echo_value <= constant_echo;
    constant_echo:=0;
    determine_high:='0';
end if;



if(echo_value < 25000) then
    data <= "001";
elsif (25000 < echo_value) and (echo_value < 280000)then
    data <= "010";
elsif (280000 < echo_value) then
    data <= "100";
end if;
end if;
end process;
end Behavioral;
```

servo_button:

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

```
entity servo_button is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           first_button , second_button : in STD_LOGIC;
           position_choice : out std_logic_vector(2 downto 0));
end servo_button;

architecture Behavioral of servo_button is
    signal rotation_binary_num : STD_LOGIC_VECTOR(2 downto 0) := "000";
    signal first_button_pressed : std_logic := '0';
    signal second_button_pressed : std_logic := '0';

begin
    process(clk, reset, first_button, second_button)
    begin
        if reset = '1' then
            rotation_binary_num <= "000";
        elsif rising_edge(clk) then
            if first_button = '1' and first_button_pressed = '0' then
                if rotation_binary_num = "100" then
                    rotation_binary_num <= rotation_binary_num;
                else
                    rotation_binary_num <= (unsigned(rotation_binary_num) + 1);
```

```
            end if;
        end if;
        first_button_pressed <= first_button;
        if second_button = '1' and second_button_pressed = '0' then
            if rotation_binary_num = "000" then
                rotation_binary_num <= rotation_binary_num;
            else
                rotation_binary_num <= (unsigned(rotation_binary_num) - 1);

            end if;
        end if;
        second_button_pressed <= second_button;
      end if;
   end process;

   position_choice <= rotation_binary_num;

end Behavioral;
```

servo_binary:

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

```
entity servo_binary is
 Port ( clk: in std_logic;
 reset: in std_logic;
 input_dataa : in std_logic_vector(2 downto 0);
 servo1: out std_logic_vector(6 downto 0));

end servo_binary;

architecture Behavioral of servo_binary is
 signal current_input : std_logic_vector(2 downto 0 ):=(others => '0');
 signal interim_servo_store: std_logic_vector(2 downto 0);
 signal counter: std_logic_vector(10 downto 0);

 constant DEGREE_180: std_logic_vector(6 downto 0):= "0111111";
 constant DEGREE_134: std_logic_vector(6 downto 0):= "0110000";
 constant DEGREE_90: std_logic_vector(6 downto 0):= "0100000";
 constant DEGREE_45: std_logic_vector(6 downto 0):= "0010000";
 constant DEGREE_0: std_logic_vector(6 downto 0):= "0000000";
 constant reset_position: std_logic_vector(2 downto 0):= "111";
begin

servo_proc: process begin
 case current_input is
 when "000" =>
```

```
servo1 <= DEGREE_0;

when "001" =>
servo1 <= DEGREE_45;

when "010" =>
servo1 <= DEGREE_90;

when "011" =>
servo1 <= DEGREE_134;

when "100" =>
servo1 <= DEGREE_134;

when others =>
servo1 <= DEGREE_180;



end case;
end process;
ser_proc: process begin
if reset = '1' then
current_input <= reset_position;
else
if rising_edge(clk) then
```

```
 current_input <= input_dataa;
 end if;
 end if;
 end process;
end Behavioral;
```

servo_main:

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

entity servo_main is
 PORT(
 clk : IN STD_LOGIC;
 reset: IN STD_LOGIC;
 seven_bit_position : IN STD_LOGIC_VECTOR(6 downto 0);
 servo: OUT STD_LOGIC
 );
end servo_main;

architecture Behavioral of servo_main is
 signal clk_out : STD_LOGIC := '0';
begin

 clk64kHz_map: entity work.servo_clkdiv(behavioral) PORT MAP(
 clk, reset, clk_out
```

```
);

servo_pwm_map: entity work.servo_pwm(Behavioral) PORT MAP(
clk_out, reset, seven_bit_position, servo
);

end Behavioral;
```

servo_clkdiv:

```
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 transient: STD_LOGIC;
 signal counter : integer range 0 to 1560 := 0;
begin
freq_divider: process (reset, clk) begin
 if (reset = '1') then
 transient <= '0';
```

```
 counter <= 0;
 elsif rising_edge(clk) then
 if (counter = 1560) then
 transient <= NOT(transient);
 counter <= 0;
 else
 counter <= counter + 1;
 end if;
 end if;
 end process;
clk_out <= transient;
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;
 seven_bit_position : IN STD_LOGIC_VECTOR(6 downto 0);
 servo : OUT STD_LOGIC
 );
end servo_pwm;
```

```
architecture Behavioral of servo_pwm is
 signal eleven_bit_cnt : unsigned(10 downto 0);
 signal signal_pwmm: unsigned(7 downto 0);

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