23/12/2018

Berfin Kavşut
21602459
EE102-3

Final Project Report
Robotic Arm

# FINAL PROJECT REPORT

**Youtube Video:** https://www.youtube.com/watch?v=KF2uJbMepaY

## Abstract

The purpose of this project was to implement a simple robotic arm and let the user to use the robotic arm by arranging the position of the arm to carry objects. The robotic arm can do these motions: turning the arm to left and right, arranging the hand up and down, grabbing and dropping objects. The implementation was planned to be bigger. As the capacity of the servo motor to carry weights is limited, the robotic arm got smaller during the implementation procedure.

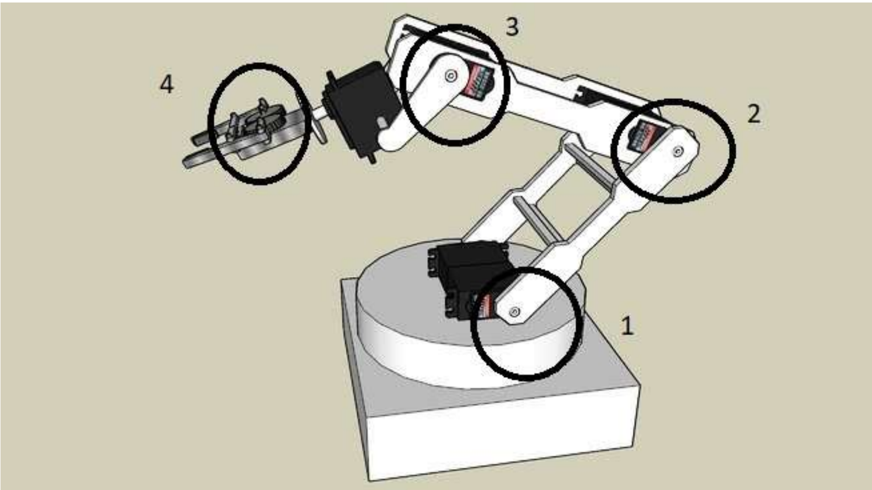



Figure 1. Sample Robotic Arm

In Figure 1, the model of the robotic arm and the positions for the servo motors are demonstrated. The design specification plan and the design methodology were studied according to this model.

23/12/2018

Berfin Kavşut
21602459
EE102-3

Final Project Report
Robotic Arm

## The Design Specification Plan


Figure 2. SG90 9G Wire Configuration

Servo motors are working with pulse width modulation. As it it seen in Figure 2, the servo motor has 3 wires. Orange wires is for the pulse width modulation. Pulse width modulation means giving a signal with a specific duty cycle and that duty cycle decides on the position of the servo motor to hold.

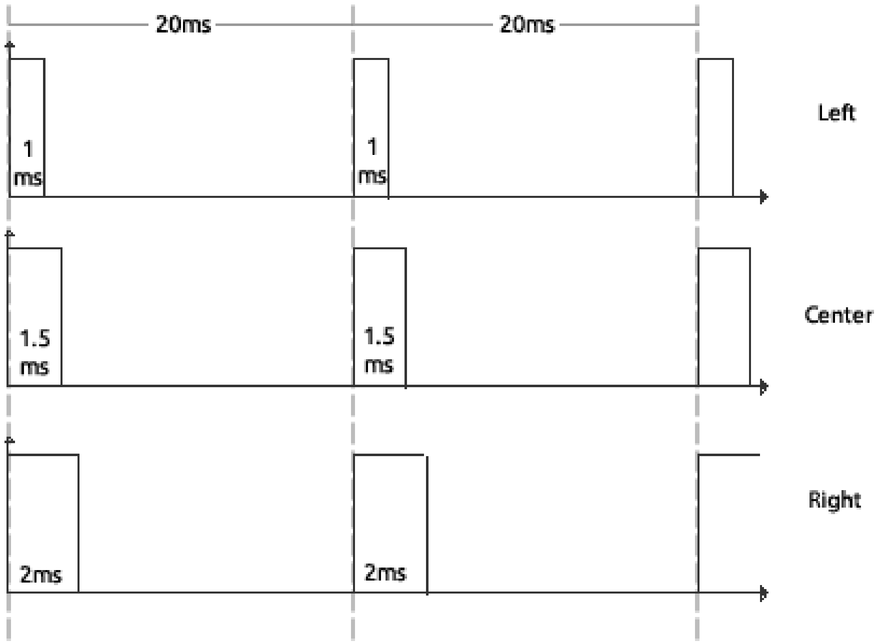



Figure 3. Pulse Width Modulation for Servo Motor SG90

In Figure 3, the pulse width modulation of the SG90 servo motors which are used in this project is explained. The servo motor works with 20 ms pulse width, in other words with 50 Hz frequency. The duty cycle of the pulse decides on the position of the servo motor to be held as it is explained in Figure 3.

The user is controlling the robotic arm by turning the switches on and off. Then, the position is arranged and the duty cycle of the square wave is increasing or decreasing accordingly. As long as the switch is on, the position changes with same time intervals and decreasing or increasing by 1°. PWM is studied in servo_pwm module of the VHDL code. The first position of the servo motors is left and the duty cycle is 5% and increasing to 10% slowly when the switch is on. For the motion in opposite direction of servo motor, the duty cycle is decreasing until 5%. The duty cycle of pulse can be between 5% and 10%.

Each motion is controlled by 2 switches. We have 3 motions, so we will have 6 switches to control the motions. For reset, there is one more switch. In total, 7 switches were used.

**The Design Methodology**

I used 6 control switches and 1 reset switch for the motions of the robotic arm. sw signal is 6-bit signal and reset is 1-bit signal. When the reset is on, all the servo motors go back to their first position. At first, all servo motors are set to be in their minimum position that is -90° left at first.

In Figure 1, the positions of the Servo 1, Servo 2, Servo 3 and Servo 4 are demonstrated. Servo 1 is for the motions turning right and left. Servo 2 and Servo 3 are for the motions going up and down. Servo 4 is for the motions grabbing and dropping the object.

| sw(5) | sw(4) | right | left |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

When left is 1 and right is 0, the position of Servo 1 is increasing each second. When right is 1 and left is 0, its position is decreasing each second. Otherwise, its position does not change.

| sw(3) | sw(2) | up | down |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

When up is 1 and down is 0, the positions of Servo 2 and Servo 3 are increasing. When down is 1 and up is 0, their positions are decreasing each 0.5 second. Otherwise, their positions do not change.

| sw(1) | sw(0) | grab | drop |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

When grab is 1 and drop is 0, the position of Servo 4 is increasing. When drop is 1 and grab is 0, its position is decreasing each 0.5 second. Otherwise, its position does not change.

## The Explanation for the VHDL Code

- **clk180kHz**

Firstly, the clock of Basys3 with 100 MHz frequency was needed to be converted to 180 kHz since the number of positions desired is 180. I implemented a clock divider in clk180kHz module.

1ms is the difference between minimum and maximum position. The frequency needed is 180 kHz to have 180 positions.

$$f = 180 / 1\,ms = 180\,kHz$$

clk180 is converting 100 MHz clock to 180 kHz with the counter from 0 to 278.

$$T = 556 * (1 / 100\ MHz) = 1/ 180\ kHz$$

The counter is counting until not 556 but 278. Its reason is the counter is needed until the signal is toggled to high and low. It should count until the half of the period and then toggle the output signal. This can be seen in the code below.

```
if (counter = 278) then
    temporal <= not (temporal);
    --The code above toggles the output signal and the counter is counting until not 556 but 278.
    counter  <= 0;
 else
     counter <= counter + 1;
 end if;
```

- **servo_pwm**

In servo_pwm, the pulse width is arranged with the input position. If the position is 0, the pulse width is 1 ms. The following code does it.

```
 -- Minimum value should be 1 ms.
   pwm <= unsigned('0' & pos) + 180;
```

The counter for the pwm module is 3600. 20 ms corresponds to 3600. The range is from 0 to 3600.

$$T = ( 1/ 180\ kHz) * 3600 = 20\ ms$$

The counter is counting from 0 to 3600. The range of input position is from 0 to 180 since the input position can be between 1ms to 2 ms pwm.1ms is corresponds to 180.

- **servo_pwm_clk180kHz**

Input and Output Signals

clk : IN STD_LOGIC;

reset: IN STD_LOGIC;

pos : IN STD_LOGIC_VECTOR(8 downto 0);

servo: OUT STD_LOGIC

clk signal is obtained from the output signal of clk180kHZ port map and this clock is used driected to servo_pwm port map. So that the servo motor can implement the pulse width modulation with the clock of 180 kHz.

- **Top_Module**

Input and Output Signals

clk : IN STD_LOGIC; -- It will be obtained from clk180kHz

reset: IN STD_LOGIC; --reset switch

sw : in STD_LOGIC_VECTOR( 5 downto 0 ); --control switches

servo: OUT STD_LOGIC_VECTOR( 3 downto 0 ) ); --servo pwm outputs

There are 4 servo motors to be controlled, therefore servo output signal is 4-bir signal. 6 switches are used to control these motions: right, left, up, down, grab, drop. Each servo motor works with the same clock. They are altering their positions by 1° in each 1 second or 0.5 second as it is calculated below.

$$T_1 = (1 / 180\ kHz) * 180000 = 1\ s$$

$$T_2 = (1 / 180\ kHz) * 360000 = 0.5\ s$$

Servo 1 is changing its position by 1° with the period of $T_1$. Other servo motors are changing their positions by 1° with the period of $T_2$. They are increasing their positions until they turn 90°. In other words, they are changing their positions from -90° to 0°.

**RTL Schematic**

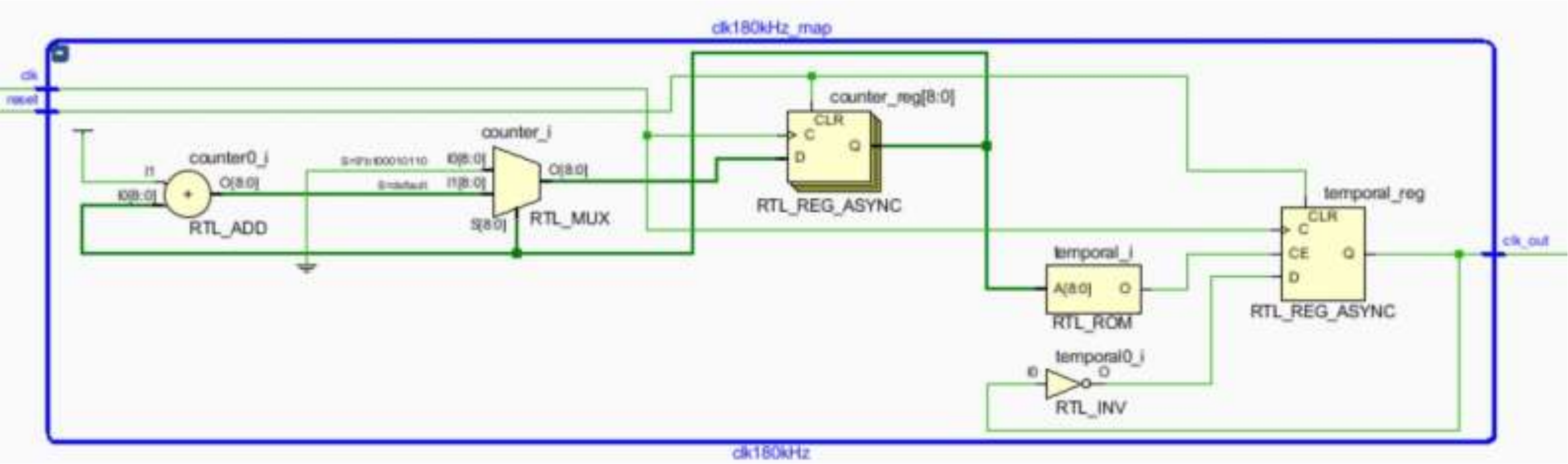



Figure 4. RTL Schematic of clk180kHz Module

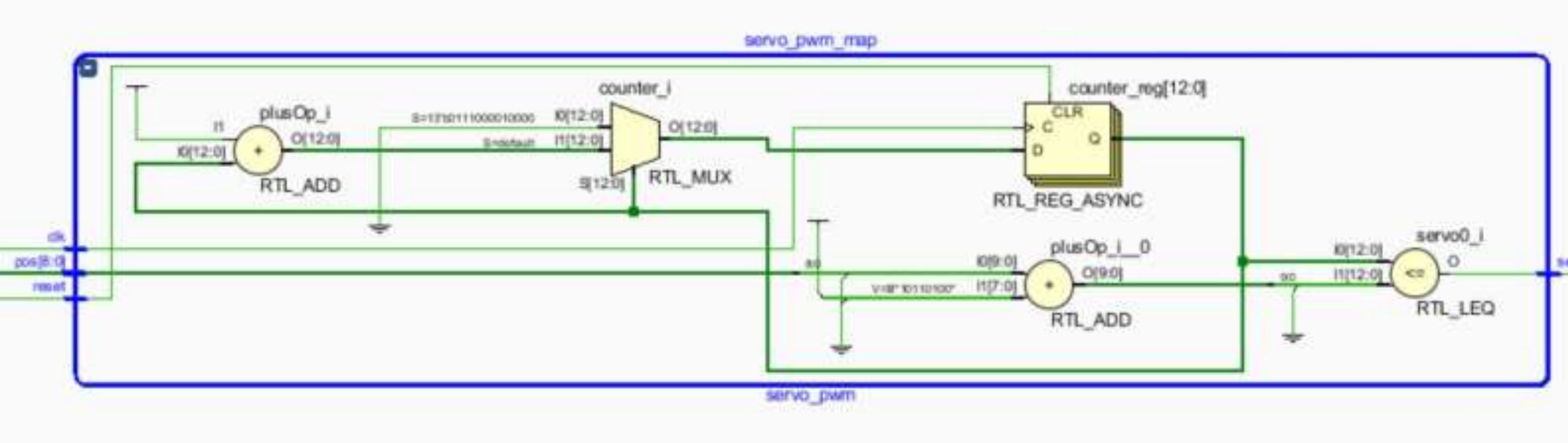



Figure 5. RTL Schematic of servo_pwm Module

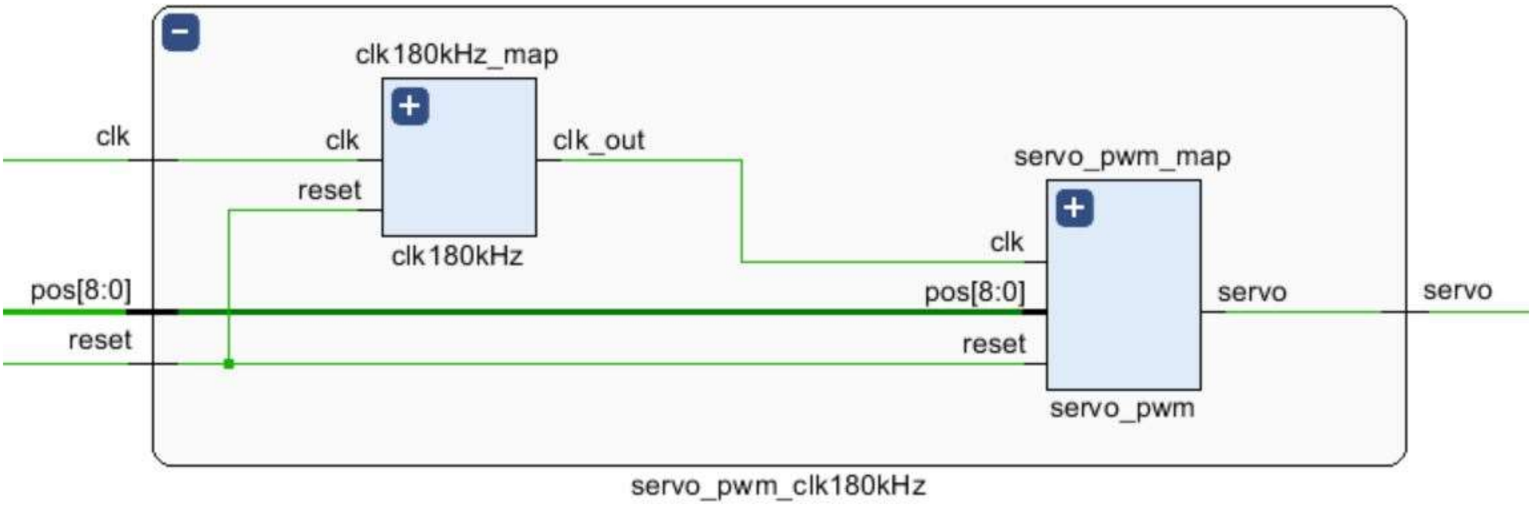



Figure 6 . RTL Schematic of servo_pwm_clk180kHz Module

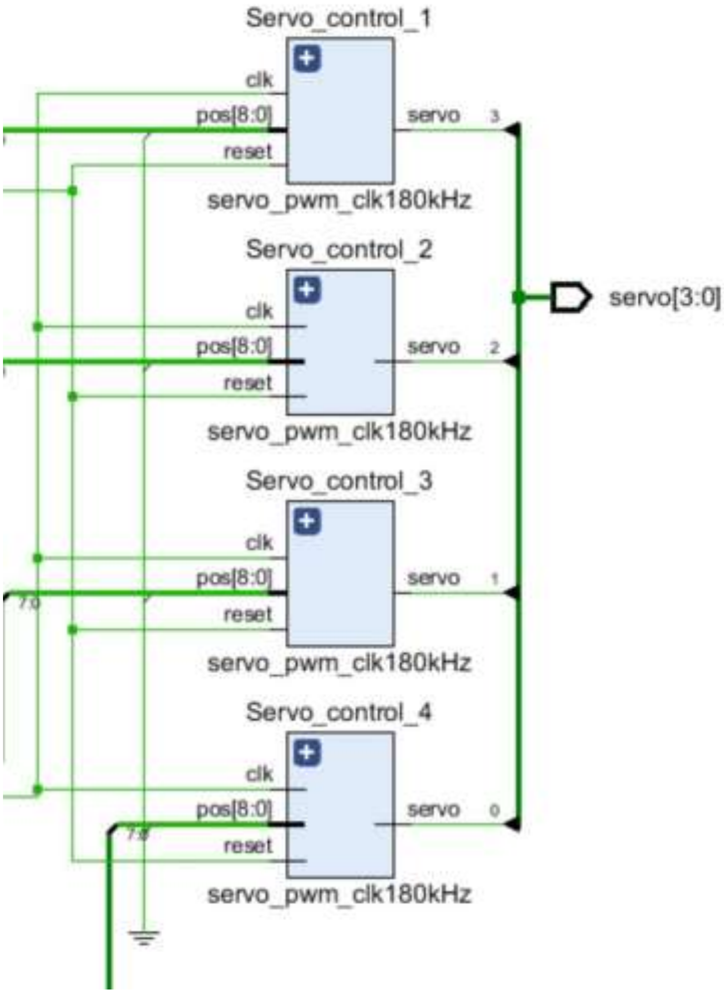



Figure 7. RTL Schematic Illustrating the servo_pwm_clk180kHz Port Maps in Top_Module

23/12/2018

Berfin Kavşut
21602459
EE102-3

Final Project Report
Robotic Arm

The RTL Schematic of Top_Module is included here. However, it cannot be seen clearly since it is very elaborate and large. I explained each function and their corresponding result in RTL schematic one by one.

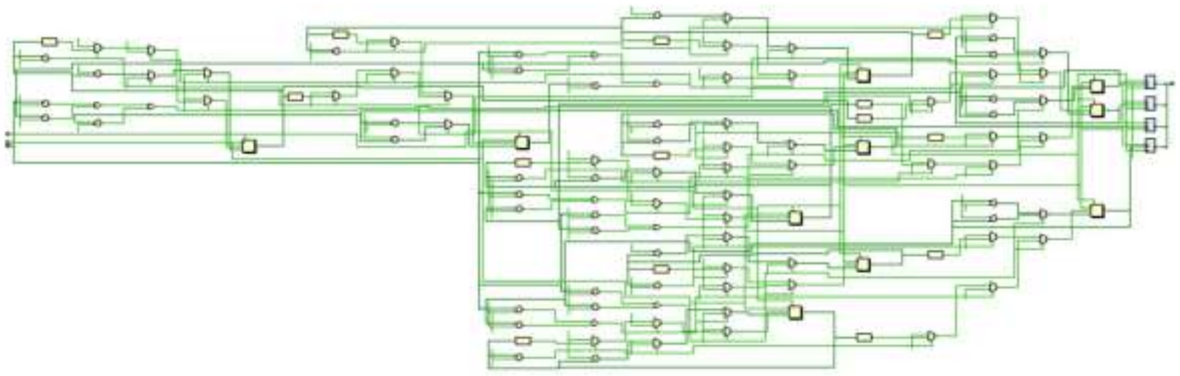


Figure 8. RTL Schematic of Top_Module

- For the logic operations which are addition, subtraction, comparison, VHDL creates a VHDL operator.

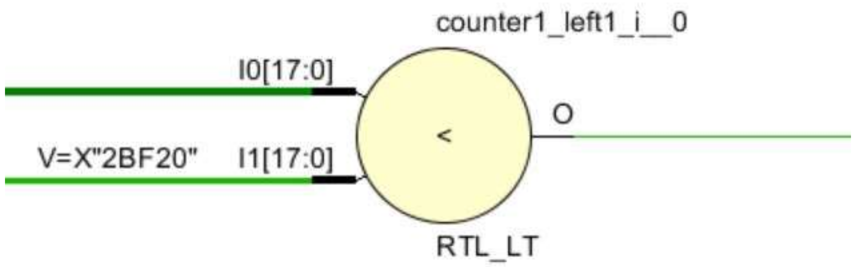



Figure 9. Comparison Operator

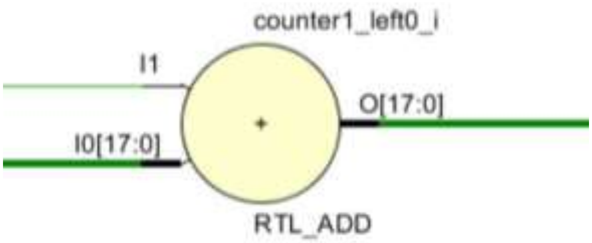



Figure 10. Addition Operator

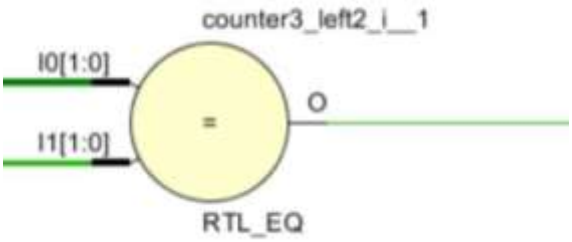



Figure 11. Equality Operator

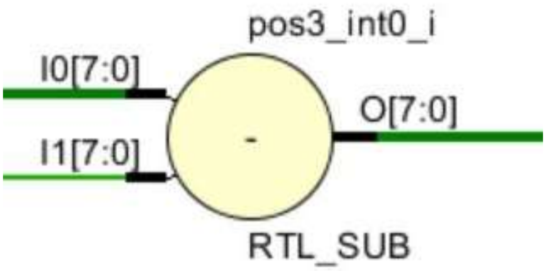



Figure 12. Subtraction Operator

- For all the counters, ROMs are created in VHDL. When the counter is increasing with the rising edge of clock, it is increasing by 1. Every integer number is one address for the corresponding binary number. Later, these data read from the ROMs are directed to registers.

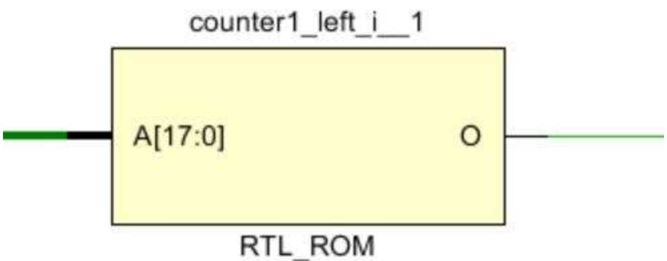



Figure 13. counter1_left ROM

- For pos1_int, pos2_int, pos3_int and pos_int4, ROMs are created in VHDL code. Its implementation in VHDL has the same logic with the counters.

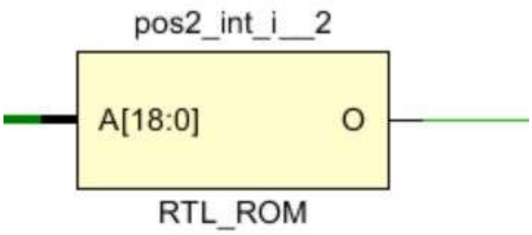



Figure 14. pos2_int ROM

- After the data is read from these ROMs, they are directed to position and counter registers since they should be kept in memory when they are not changed.

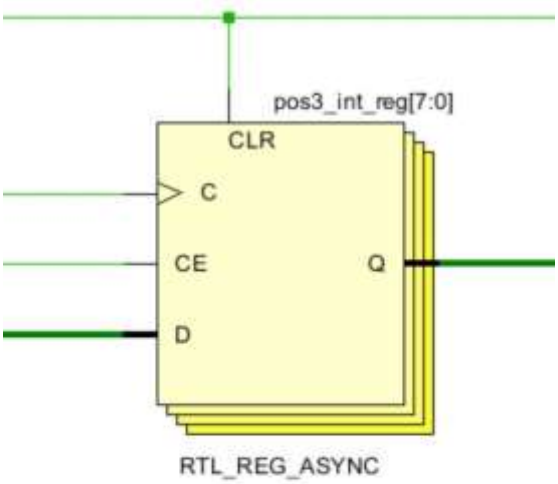



Figure 15. Position Register

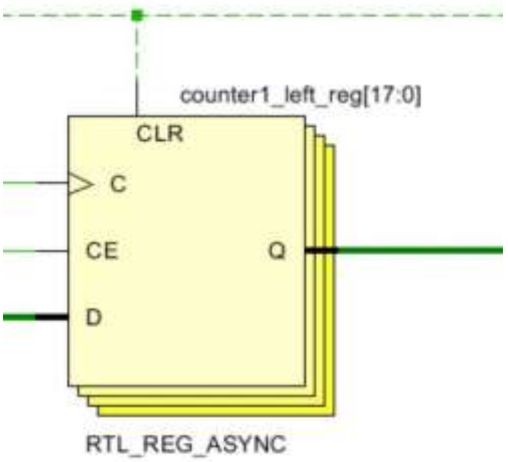



Figure 16. Counter Register

- If statements are implemented by multiplexers.

-

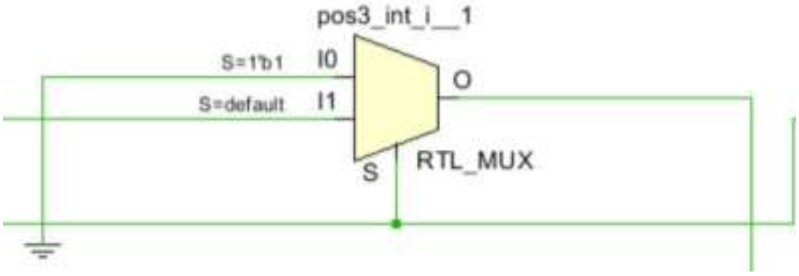



Figure 17. Multiplexer for Positions

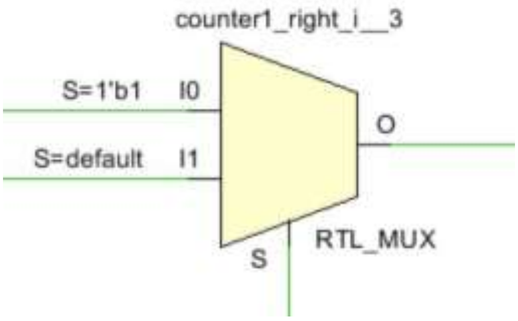



Figure 18. Multiplexer for Counters

23/12/2018

Berfin Kavşut
21602459
EE102-3

Final Project Report
Robotic Arm

**Results**

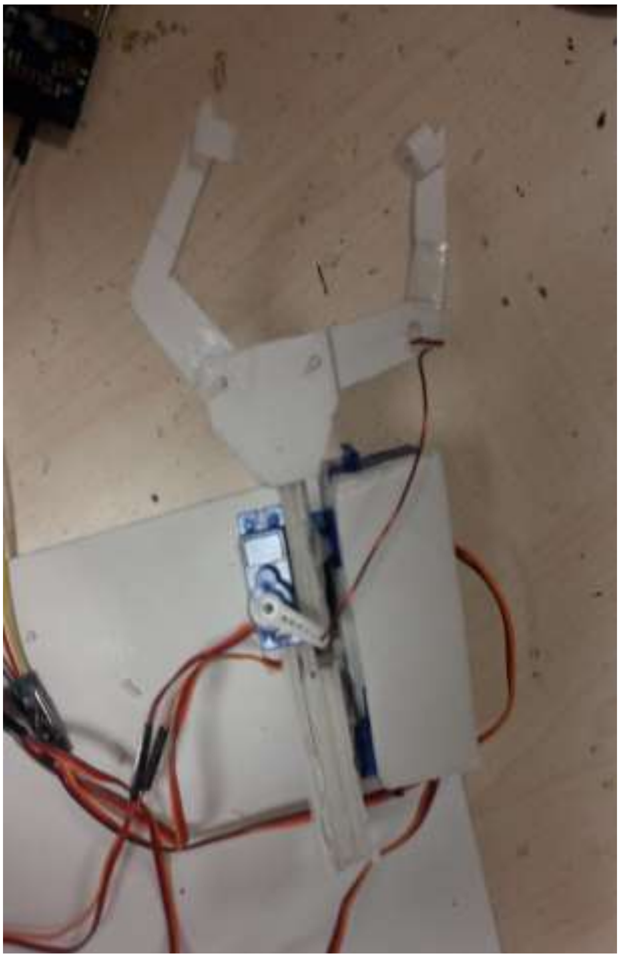


Figure 19. Dropping Motion

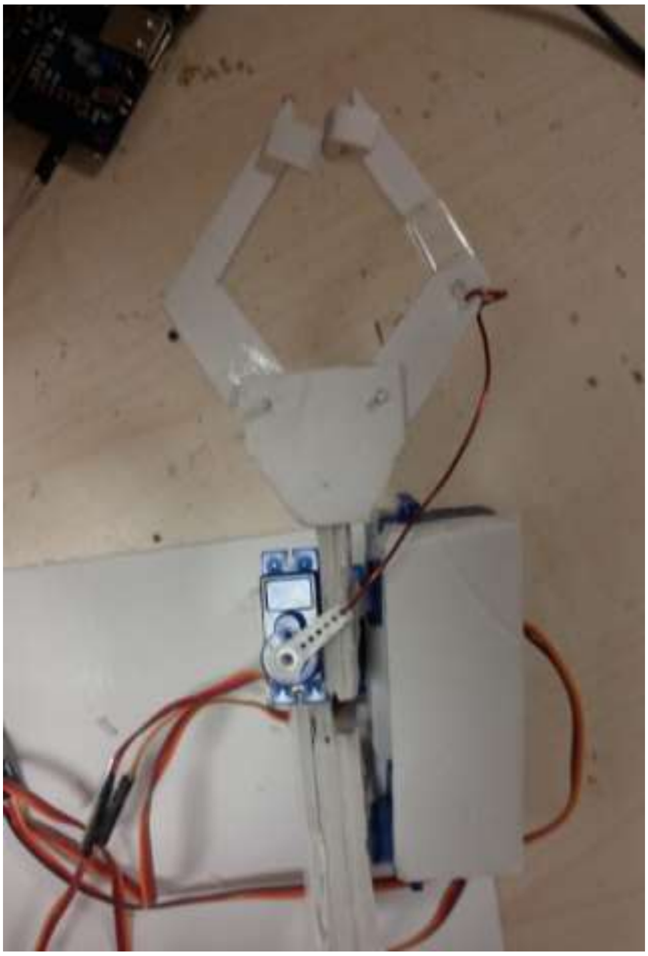


Figure 20. Grabbing Motion

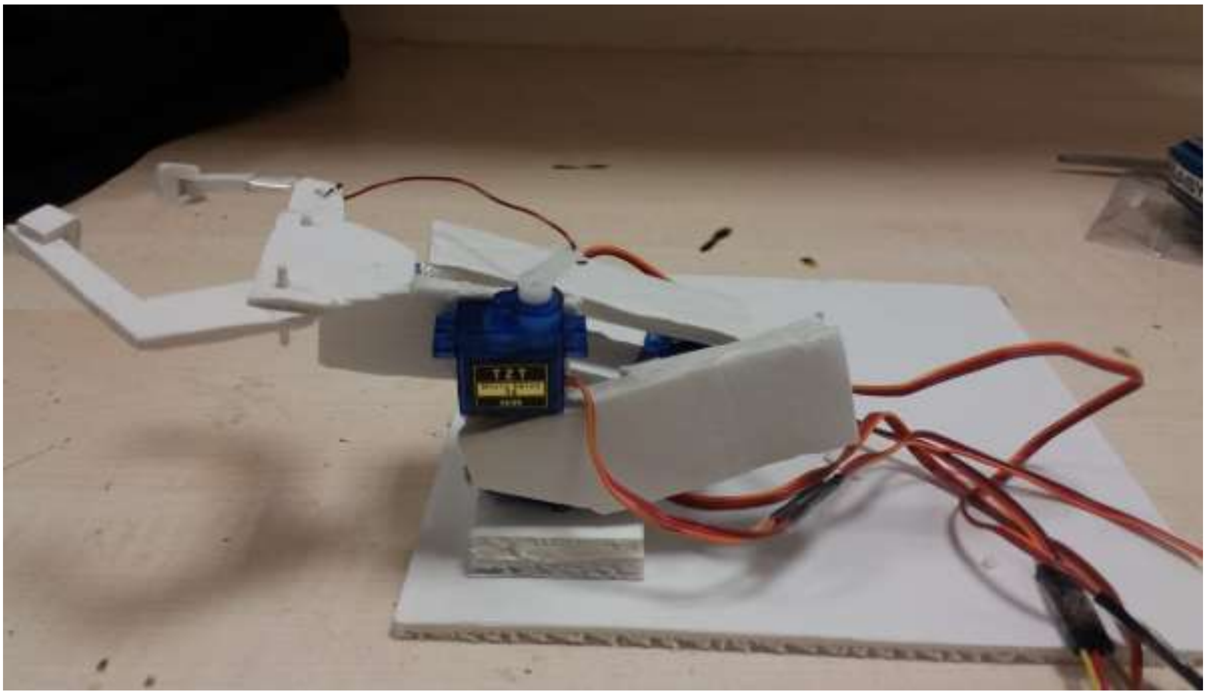


Figure 21. Going Down Motion

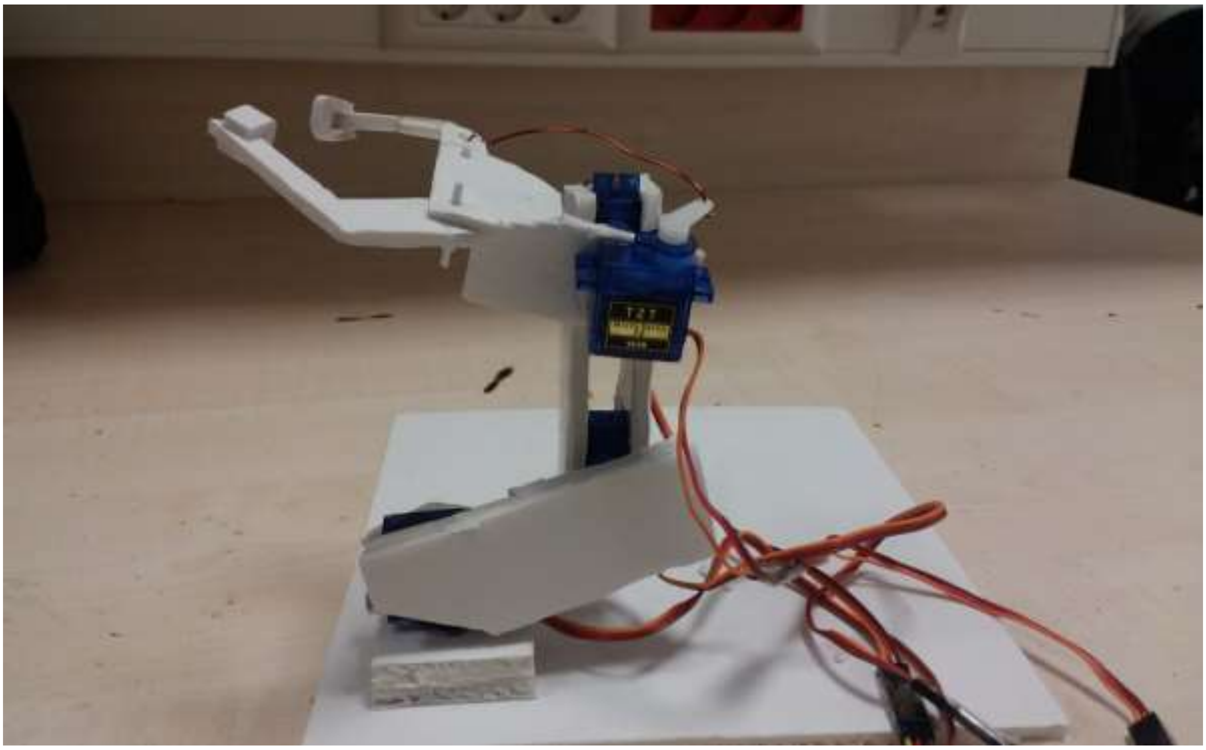


Figure 22. Going Up Motion

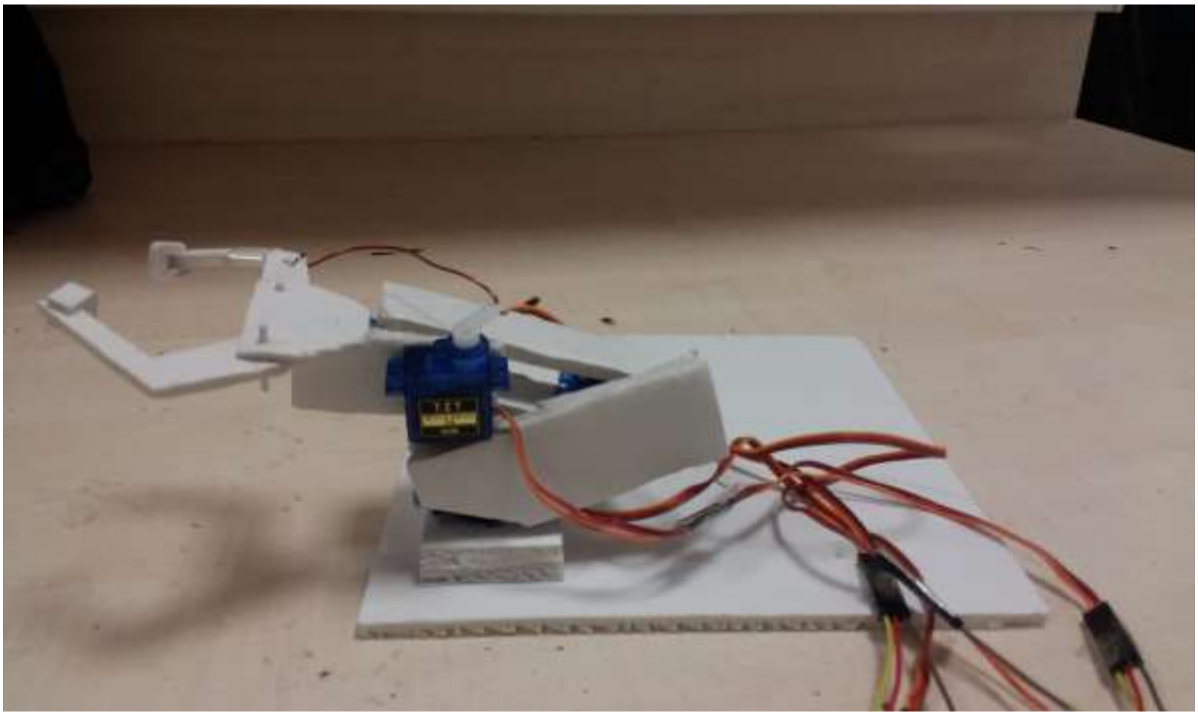


Figure 23. Turning Right Motion

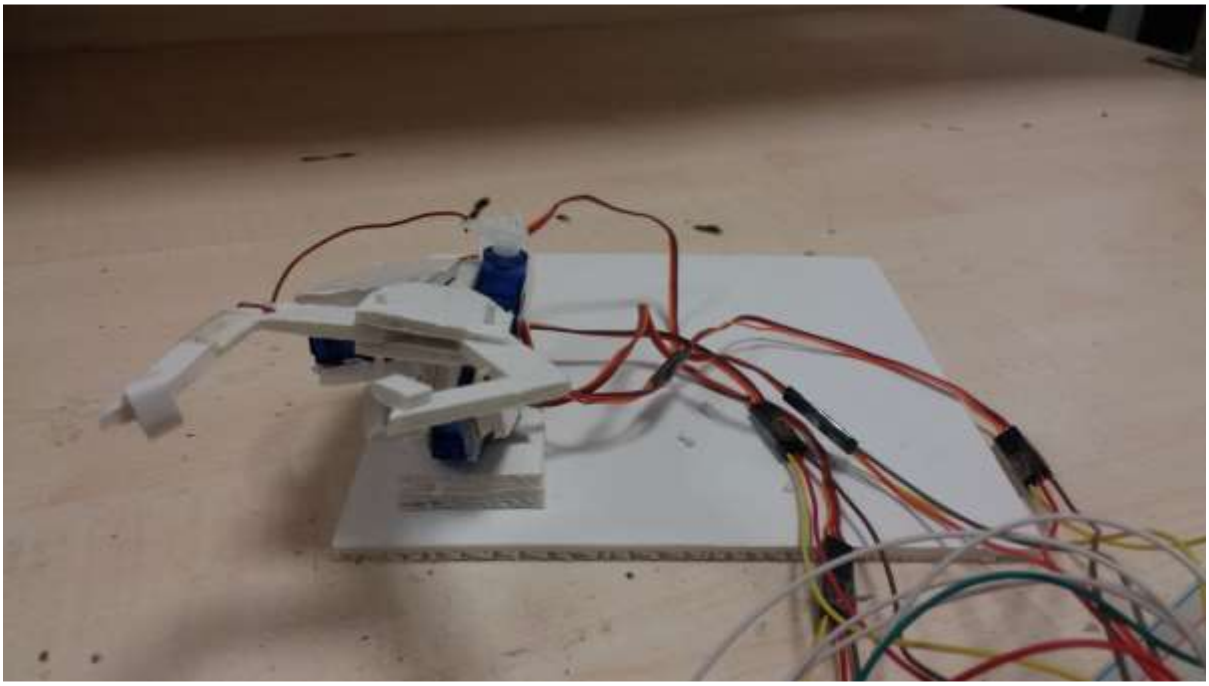


Figure 24. Turning Left Motion

The robotic arm and its motions can be seen in Figure 19 – 24. The servo motors are moving until 90° and it is seen in the figures as it was planned and calculated.

## Conclusion

In this project, I learnt how SG90 servo motors work, how pulse width modulation works and is implemented in VHDL code, how to use clock divider and read RTL schematic in VHDL. While I am searching about servo motor, I saw other areas it is used and had an idea about what else I can use them for. During making the servo motors work, I had some difficulties and learnt from my mistakes how to use them properly. Firstly,I forgot that I should ground all servo motors from the same connection with GND pin of PMOD in Basys3. Secondly, because I used the motors a lot during building the robotic arm, some of them were broken from inside and the servo motors could not change their positions. The power supply gave much more current supply than it should have given since the servo motors were showing resistance due to broken pieces inside. They heated up and I had to change these servo motors. At the end, after fixing these problems, the robotic arm functioned in the way it was designed according to my calculations for positions and time intervals.

## References

"Servo Motor SG-90." Components101. Accessed December 18, 2018. components101.com/servo-motor-basics-pinout-datasheet.

"Servo Motor SG90 Data Sheet."Imperial College London. Accessed December 15, 2018. ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf.

"How Does a Servo Work?" Servo City. Accessed December 17, 2018. servocity.com/how-does-a-servo-work.

"Servomotor Control with PWM and VHDL." Code Project. Accessed December 12, 2018. codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL

**Appendix**

**VHDL Code**

- **constraint**

```
#Clock

# Clock signal

set_property PACKAGE_PIN W5 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]

create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]


#Control Switches

set_property PACKAGE_PIN T2 [get_ports {sw[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]

set_property PACKAGE_PIN R3 [get_ports {sw[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]

set_property PACKAGE_PIN W2 [get_ports {sw[2]}]

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

set_property PACKAGE_PIN U1 [get_ports {sw[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]

set_property PACKAGE_PIN T1 [get_ports {sw[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]

set_property PACKAGE_PIN R2 [get_ports {sw[5]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]


#Reset Switches

set_property PACKAGE_PIN V17 [get_ports {reset}]

set_property IOSTANDARD LVCMOS33 [get_ports {reset}]


#PMOD Header JB

#Sch name = JB1

set_property PACKAGE_PIN A14 [get_ports {servo[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {servo[3]}]

#Sch name = JB7

set_property PACKAGE_PIN A15 [get_ports {servo[2]}]

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


#Pmod Header JC

#Sch name = JC1

set_property PACKAGE_PIN K17 [get_ports {servo[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {servo[1]}]

#Sch name = JC7

set_property PACKAGE_PIN L17 [get_ports {servo[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {servo[0]}]
```

- **clk180kHz**

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity clk180kHz is

    Port ( clk         :  in  STD_LOGIC;

               reset     : in  STD_LOGIC;

               clk_out : out STD_LOGIC);

end clk180kHz;



architecture arch_clk180kHz of clk180kHz is

signal temporal: STD_LOGIC;

signal counter : integer range 0 to 278 := 0;



begin

   clock_divider: process (reset, clk) begin

        if (reset = '1') then

            temporal <= '0';

            counter  <= 0;

        elsif rising_edge(clk) then
```

```
            if (counter = 278) then
               temporal <= not (temporal);
               counter  <= 0;
            else
               counter <= counter + 1;
            end if;
        end if;
    end process;
    clk_out <= temporal;
end arch_clk180kHz;
```

- **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;
            pos    : in  STD_LOGIC_VECTOR (8 downto 0);
            servo : out  STD_LOGIC);
```

```
end servo_pwm;


architecture arch_servo_pwm of servo_pwm is

signal counter : unsigned(12 downto 0);

-- pwm signal is used to generate the output pulse to give to the servo motors.

signal pwm: unsigned(9 downto 0);

begin

    -- 1 ms for minimum position.

    pwm <= unsigned('0' & pos) + 180;

    -- Counter process: It counts from 0 until 3600, which corresponds to 20 ms.

    Counter_process: process (reset, clk) begin

        if (reset = '1') then

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

        elsif rising_edge(clk) then

            if (counter = 3600) then

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

            else

                counter <= counter + 1;

            end if;

        end if;

    end process;
```

```
    -- pulse for the servo motors

    servo <= '1' when (counter <= pwm) else '0';

end arch_servo_pwm;
```

- **servo_pwm_clk180kHz**

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity servo_pwm_clk180kHz is

    port( clk    : in  STD_LOGIC;

          reset : in  STD_LOGIC;

          pos   : in  STD_LOGIC_VECTOR (8 downto 0);

          servo : out STD_LOGIC);

end servo_pwm_clk180kHz;



architecture arch_servo_pwm_clk180kHz of servo_pwm_clk180kHz is

    component clk180kHz

        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;
              pos   : in  STD_LOGIC_VECTOR(8 downto 0);
              servo : out STD_LOGIC);
    end component;
signal clk_out : STD_LOGIC := '0';

begin
    clk180kHz_map: clk180kHz port map( clk, reset, clk_out );
    --PWM is operated with 180 kHz clock.
    servo_pwm_map: servo_pwm port map ( clk_out, reset, pos, servo );
end arch_servo_pwm_clk180kHz;
```

- **Top_Module**

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

-- Arithmetic Functions with Signed or Unsigned Values
use IEEE.NUMERIC_STD.ALL;
```

```
entity Top_Module is

    Port (   clk        : in STD_LOGIC;

             reset  : in  STD_LOGIC;

             sw       : in STD_LOGIC_VECTOR (5 downto 0);

             servo  : out STD_LOGIC_VECTOR (3 downto 0) );

end Top_Module;


architecture arch_Top_Module of Top_Module is

    component servo_pwm_clk180kHz

        port( clk  : in  STD_LOGIC;

              reset: in  STD_LOGIC;

              pos  : in  STD_LOGIC_VECTOR(8 downto 0);

              servo: out STD_LOGIC);

   end component;


--positions

signal pos1_int  : integer range 0 to 180;

signal pos2_int  : integer range 0 to 180;

signal pos3_int  : integer range 0 to 180;

signal pos4_int  : integer range 0 to 180;
```

```
--Counters for increasing and decreasing the positions of the servo motors

signal counter1_right    : integer range 0 to 180000; -- counts until 180000

signal counter1_left     : integer range 0 to 180000; -- counts until 180000

signal counter2_up       : integer range 0 to 360000; -- counts until 360000

signal counter2_down  : integer range 0 to 360000; -- counts until 360000

signal counter3_right    : integer range 0 to 360000; -- counts until 360000

signal counter3_left     : integer range 0 to 360000; -- counts until 360000


-- Switch Controls

signal sw1 : STD_LOGIC_VECTOR( 1 downto 0 );

signal sw2 : STD_LOGIC_VECTOR( 1 downto 0 );

signal sw3 : STD_LOGIC_VECTOR( 1 downto 0 );


-- Positions in std_logic_vector(8 downto 0) types

signal pos1 : STD_LOGIC_VECTOR( 8 downto 0 );

signal pos2 : STD_LOGIC_VECTOR( 8 downto 0 );

signal pos3 : STD_LOGIC_VECTOR( 8 downto 0 );

signal pos4 : STD_LOGIC_VECTOR( 8 downto 0 );


begin
```

```
sw1 <= sw (5 downto 4); -- right and left

sw2 <= sw (3 downto 2); -- up and down

sw3 <= sw (1 downto 0); -- grab and drop



--Switch Controls

Servo_1_process: process (reset,clk) begin

    -- reset

    if reset = '1' then

        pos1_int <= 0;

        counter1_left <= 0;

        counter1_right <= 0;



    -- set

    elsif rising_edge (clk) then



        --right

        if sw1 = "01" AND pos1_int > 0 then

           if counter1_left < 180000  then

                counter1_left <= counter1_left + 1;

           elsif counter1_left = 180000 then

                pos1_int <= pos1_int - 1;
```

```
            counter1_left <= 0;
        end if;


      --left
      elsif sw1 = "10" AND pos1_int < 90 then
        if counter1_right < 180000 then
          counter1_right <= counter1_right + 1;
        elsif counter1_right = 180000 then
          pos1_int <= pos1_int + 1;
          counter1_right <= 0;
        end if;
      --stay
      else
         counter1_right <= 0;
         counter1_left <= 0;
      end if;


   end if;


end process;
```

```
--Switch controls

Servo_2_and_3_process: process (reset,clk) begin

    --reset

    if reset = '1' then

        pos2_int <= 0;

        pos3_int <= 0;

        counter2_up <= 0;

        counter2_down <= 0;

    --set

    elsif rising_edge (clk) then

        --up

        if sw2 = "10" AND ( pos2_int > 0 AND pos3_int > 0 )then

           if counter2_up < 360000 then

                counter2_up <= counter2_up + 1;

           elsif counter2_up = 360000 then

                pos2_int <= pos2_int - 1;

                pos3_int <= pos3_int - 1;

                counter2_up <= 0;

           end if;

        --down

        elsif sw2 = "01" AND ( pos2_int < 90 AND pos3_int < 90 ) then
```

```
            if counter2_down < 360000 then
                counter2_down <= counter2_down + 1;
            elsif counter2_down = 360000 then
                pos2_int <= pos2_int + 1;
                pos3_int <= pos3_int + 1;
                counter2_down <= 0;
             end if;
        --stay
        else
           counter2_up <= 0;
           counter2_down <= 0;
        end if;
    end if;
end process;


--Switch Controls
Servo_4_process: process (reset,clk) begin
    --reset
    if reset = '1' then
        pos4_int <= 0;
        counter3_left <= 0;
```

```
    counter3_right <= 0;


--set
elsif rising_edge (clk) then
    --left
    if sw3 = "10" and pos4_int > 0 then
       if counter3_left < 180000 then
            counter3_left <= counter3_left + 1;
       elsif counter3_left = 180000 then
            pos4_int <= pos4_int - 1;
            counter3_left <= 0;
       end if;


    --right
    elsif sw3 = "01" and pos4_int < 90 then
       if counter3_right < 180000 then
         counter3_right <= counter3_right + 1;
       elsif counter3_right = 180000 then
         pos4_int <= pos4_int + 1;
         counter3_right <= 0;
       end if;
```

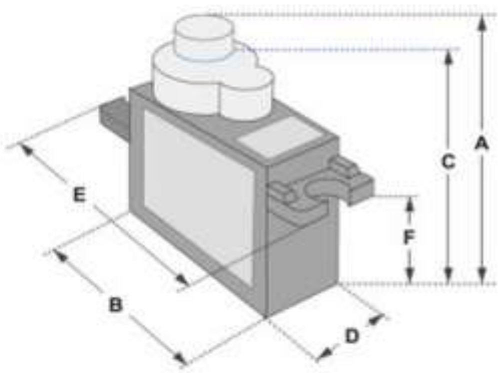
```
        --stay
        else
            counter3_right <= 0;
            counter3_left <= 0;
        end if;
    end if;

end process;

pos1 <= std_logic_vector( to_unsigned( pos1_int, pos1'length ) );
pos2 <= std_logic_vector( to_unsigned( pos2_int, pos2'length ) );
pos3 <= std_logic_vector( to_unsigned( pos3_int, pos3'length ) );
pos4 <= std_logic_vector( to_unsigned( pos4_int, pos4'length ) );

Servo_control_1 : servo_pwm_clk180kHz port map (clk, reset, pos1, servo(3));
Servo_control_2 : servo_pwm_clk180kHz port map (clk, reset, pos2, servo(2));
Servo_control_3 : servo_pwm_clk180kHz port map (clk, reset, pos3, servo(1));
Servo_control_4 : servo_pwm_clk180kHz port map (clk, reset, pos4, servo(0));
end arch_Top_Module;
```

Berfin Kavşut
21602459
EE102-3



**Datasheet of SG90 Servo Motor**


Figure 25. SG90 Servo Motor




| Dimensions & Specifications |
|---|
| A (mm) : 32 |
| B (mm) : 23 |
| C (mm) : 28.5 |
| D (mm) : 12 |
| E (mm) : 32 |
| F (mm) : 19.5 |
| Speed (sec) : 0.1 |
| Torque (kg-cm) : 2.5 |
| Weight (g) : 14.7 |
| Voltage : 4.8 - 6 |

Figure 26. Dimensions & Specifications

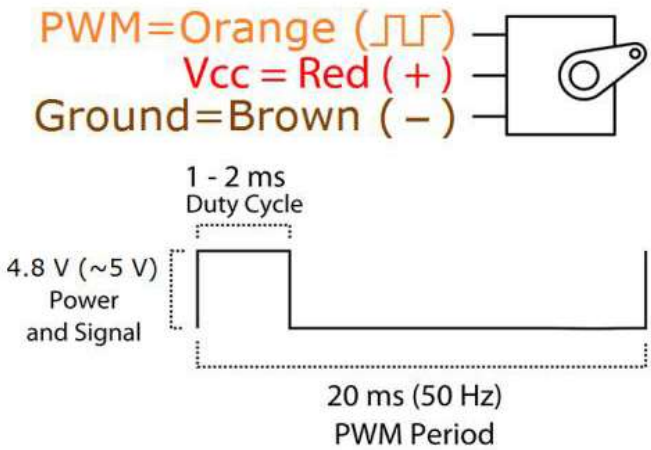



Figure 27. Wire Configuration and Pulse Width Modulation of SG90 Servo Motor