New York City College of Technology

The City University of New York

Fall, 2019

Component and Subsystem Design II

CET 4805 Section **D486**

Meeting Day: **Tuesday**

INSTRUCTOR: Prof. Y. Wang

STUDENT: Gene Nadela, Kevin Munar, Melissa Valle

Final Project
FPGA Servo Control for Robotic Arm

Submission Date: 12/17/2019

Introduction

For the final project, we decided to develop a 3DOF Robotic Arm with Gripper Control. This arm will utilize the Altera DE2 FPGA board to control four servos, necessitating the use of Pulse Width Modulation (PWM) to assign each servo a position. We believe developing this robotic arm will demonstrate much of what we have learned regarding FPGAs and component/subsystem design.

Project Overview

Using an existing design from HowToMechatronics.com, we used a 3D printer to manufacture the arm. The servos are then added, and wired appropriately, with the control wires for each servo separated to allow for easy placement into the GPIO pins on the DE2 board. The servos chosen are widely used servo motors: SG90/MG90 and the MG995.

Materials/Equipment Used:

- Robotic Arm Parts (3D Printed using eSun PLA+)
- 2-SG90 Microservos
- 2 MG995 Servos
- 7.4 Lipo Battery 1800maH
- Altera DE2 FPGA Board

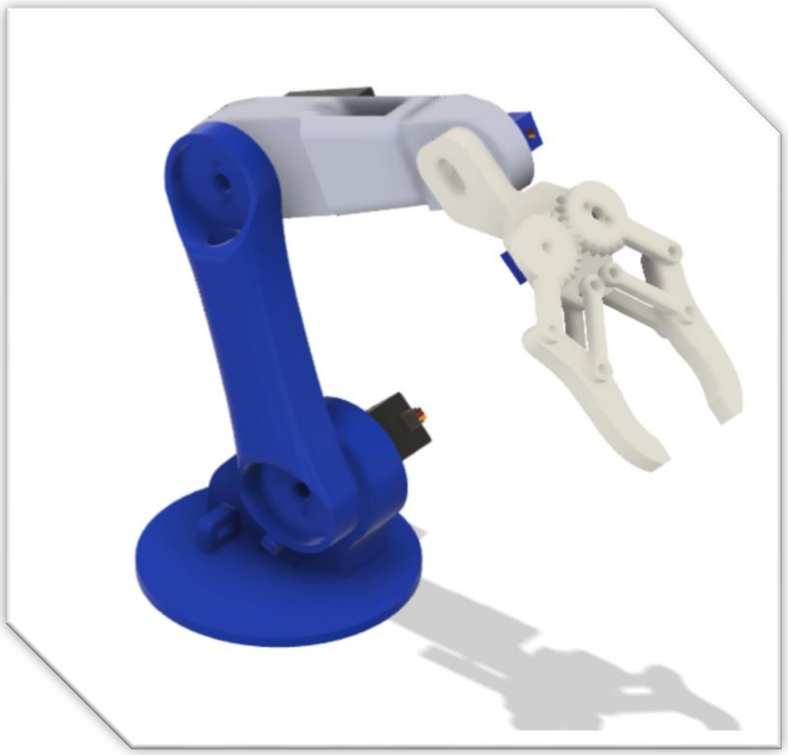

Fig. 1 – Original robotic arm as modeled in Fusion360 CAD software. The final version excludes wrist rotation.

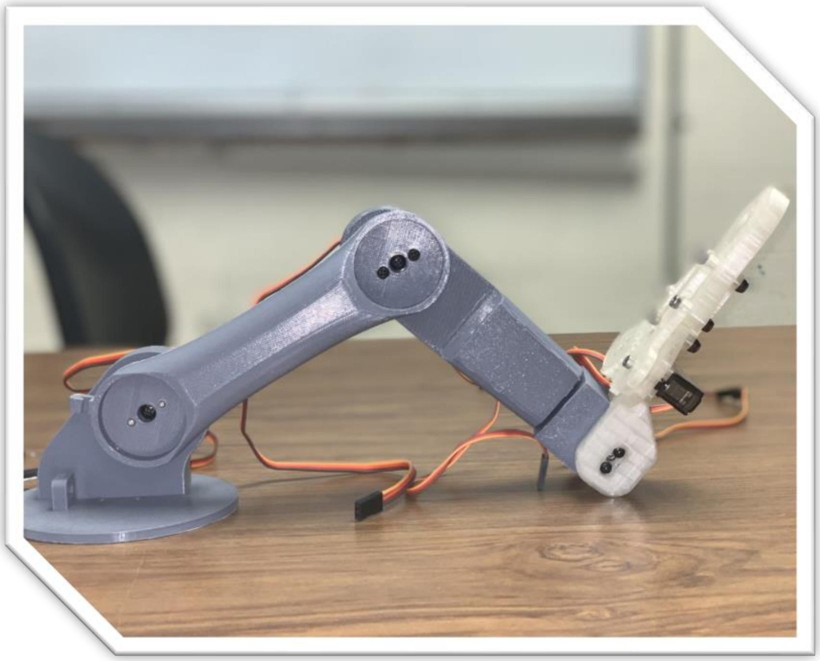

Fig. 2 – Initial robot arm design, with servo motors attached at each point where the limbs are connected, and a single servo for the gripper mechanism.

Servo Analysis, Calculations for PWM and Frequency Divider

Both the SG90 and the MG995 use PWM to control servo position. As such, we were careful to select servos that had the same Duty Cycle, PWM Period and Operating Voltage. The servos operate at a minimum pulse width of 1 ms and a maximum of 2 ms, with a neutral position of 1.5 ms. A 20ms PWM period results in a 50 Hz frequency.

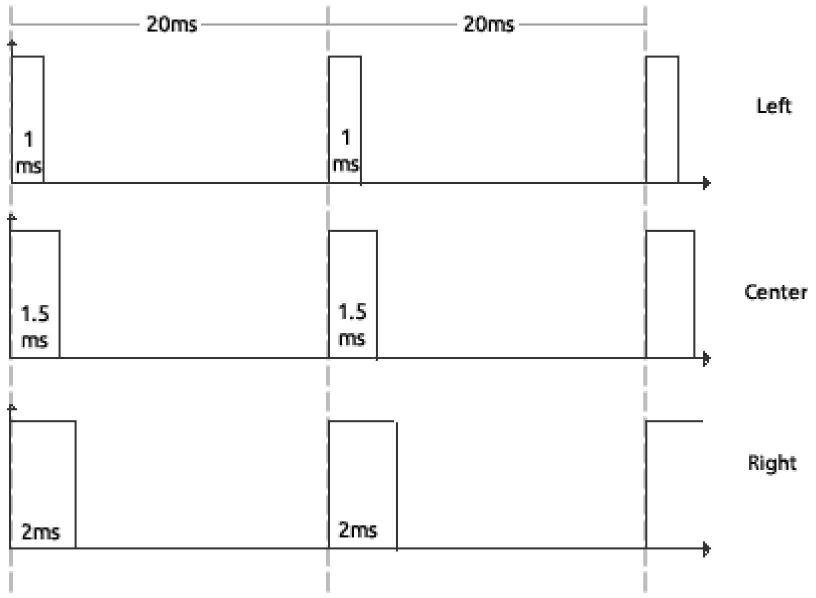

Fig 3 – Diagram of Pulse Width Servo Control (ref: *https://www.codeproject.com /Articles/513169/Servomotor-Control-with-PWM-and-VHDL*).

These values become important as we develop the code to generate a pulse width signal that is recognized by the servo motors.

We first determine the range of operation of the servo motors. This refers to the minimum and maximum pulse width.

$$Range = PulseWidth_{maximum} - PulseWidth_{minimum} = 2ms - 1ms = \mathbf{1ms}$$

We now need to determine the frequency required to set a certain quantity of positions. For this project, we selected 128 as the number of possible positions for the servo motor, which is also known as the resolution of the servo. To determine the frequency needed:

$$Frequency_{128ServoPositions} = \left(\frac{range}{resolution}\right)^{-1} = \left(\frac{1ms}{128}\right) = \mathbf{128\ kHz}$$

To generate this frequency, we will need to create a frequency divider that uses the 50Mhz internal clock of the Altera DE2 Board to output the 128 kHz clock. We will need to determine the scaling factor:

$$ScalingFactor = \frac{f_{in}}{f_{out}} = \frac{50MHz}{128kHz} = \mathbf{390.625}$$

The frequency divider code uses a counter to generate the appropriate square wave from the input frequency. In this case, we want to generate a square wave that is HIGH for half the 390.625 count. This means that the square wave will be HIGH for 194 counts, and LOW for 194 counts.

Control Scheme using a Finite State Machine

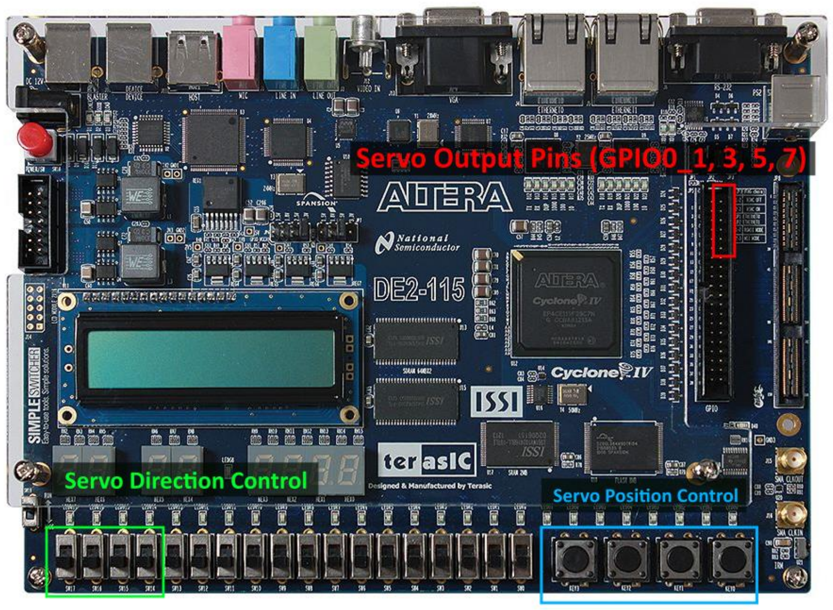

Fig 4 – Input and Output planned arrangement on DE2 board.

We want to be able to control all four servos simultaneously, choosing the direction of the position using the onboard switches and incrementing the movement using onboard pushbutton control.

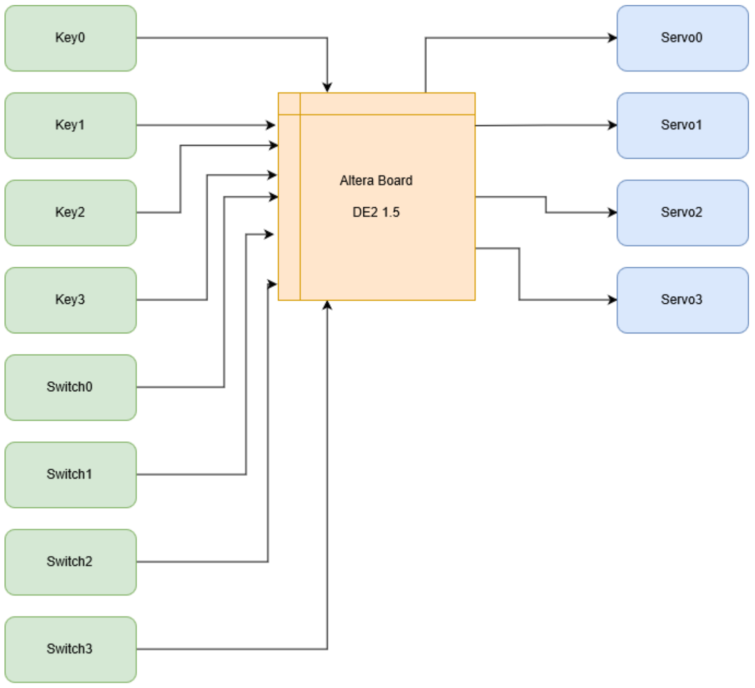

Fig. 5 – General scheme for switch and pushbutton control of robotic arm servos.

An FSM becomes an important tool in being able to input increments that cycle, with directional control. It is implemented in the following manner. We divide each position into seven states, each corresponding to a position.

State	Value	Value$_{\text{Binary}}$	Position
0	0	0000000	Leftmost
1	21	0010101	
2	42	0101010	
3	63	0111111	Center/Neutral
4	84	1010100	
5	105	1101001	
6	127	1111111	Rightmost

Results

Oscilloscope Measurements

Channel 1 (Yellow) is a reading from GPIO0_1 (servo0), while channel 2 is a reading from GPIO0_3 (servo1). We increment servo0 using the pushbutton, thus increasing the pulse width with each push.

Channel 2 is used for comparison. The output frequency of each wave is about 62.5Hz, which is not 50Hz, but appears to be well within the operational frequency of each servo motor.

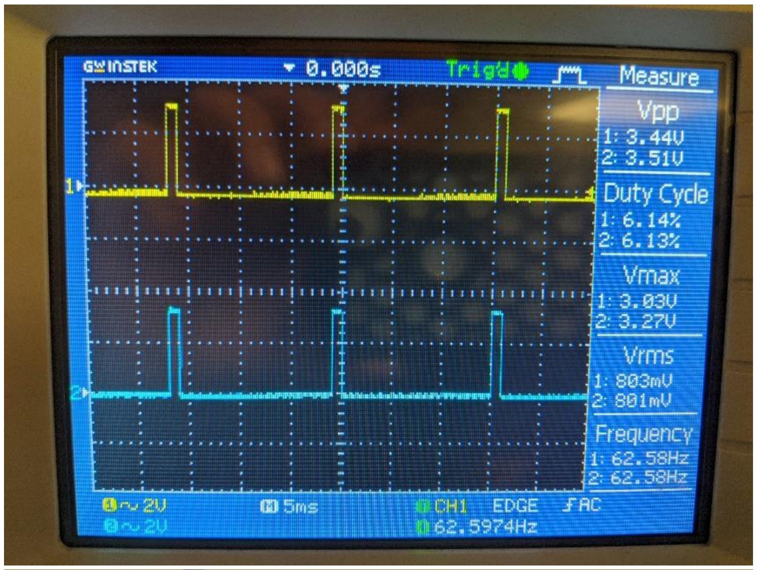

Fig. 6a – State 0, Minimum pulse width of approximately 1ms.

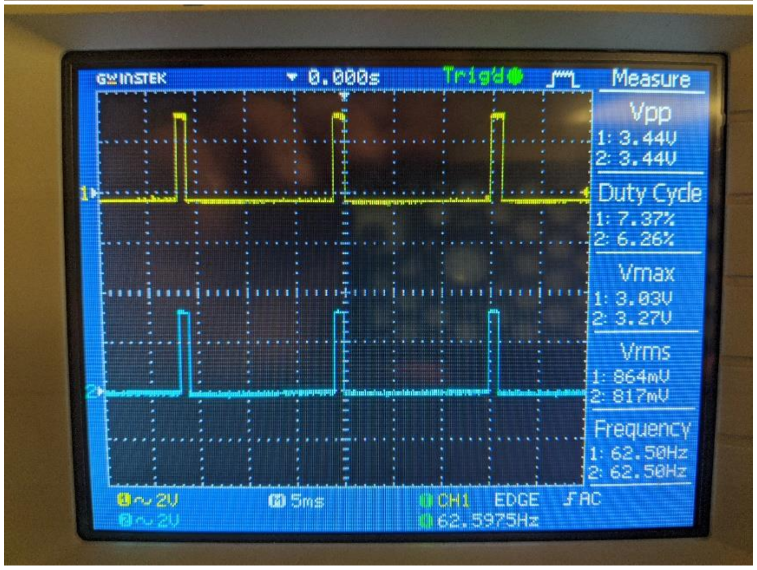

Fig. 6b – State 1, when pushbutton is pressed once, pulse width increases.

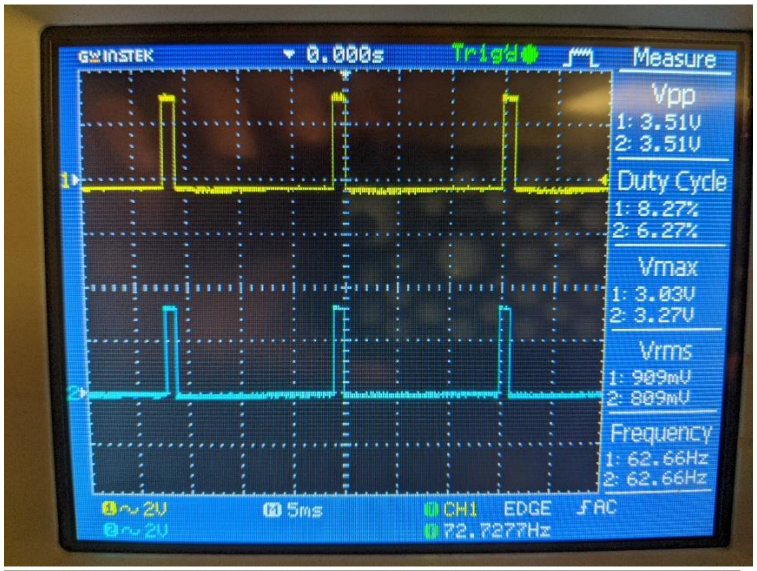

Fig. 6c – State 2, when pushbutton is pressed once, pulse width increases.

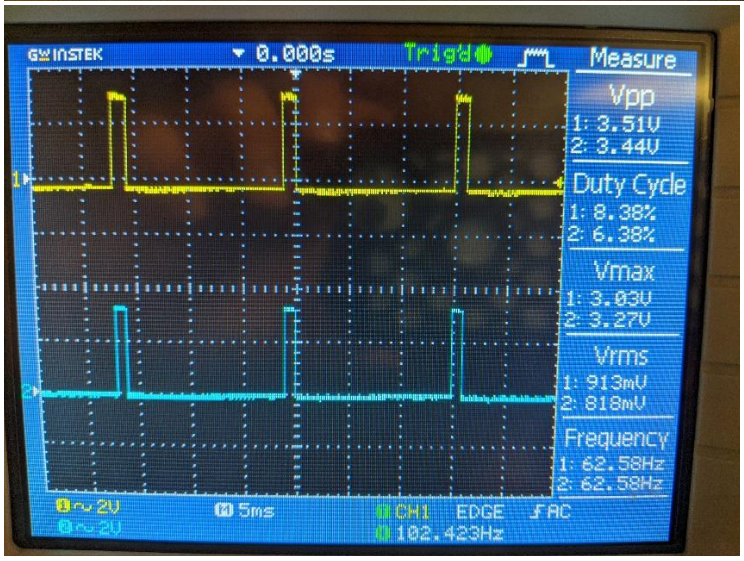

Fig. 6d – State 3, when pushbutton is pressed once, pulse width increases.

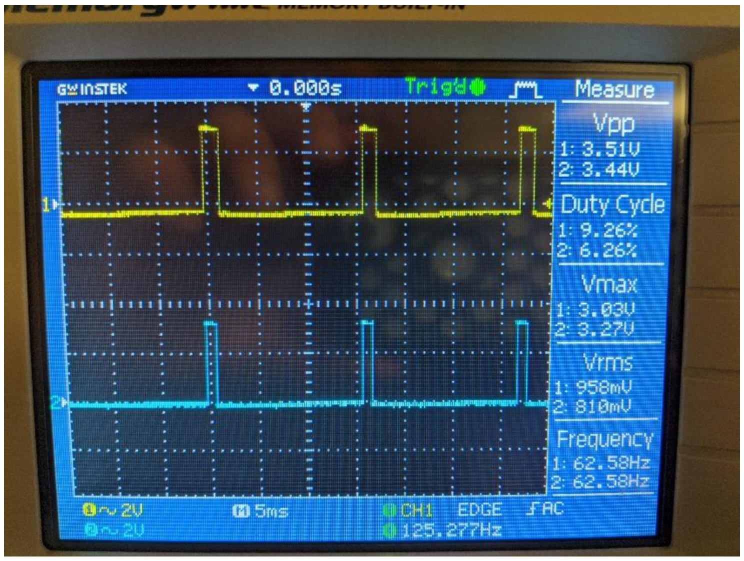

Fig. 6e – State 4, when pushbutton is pressed once, pulse width increases.

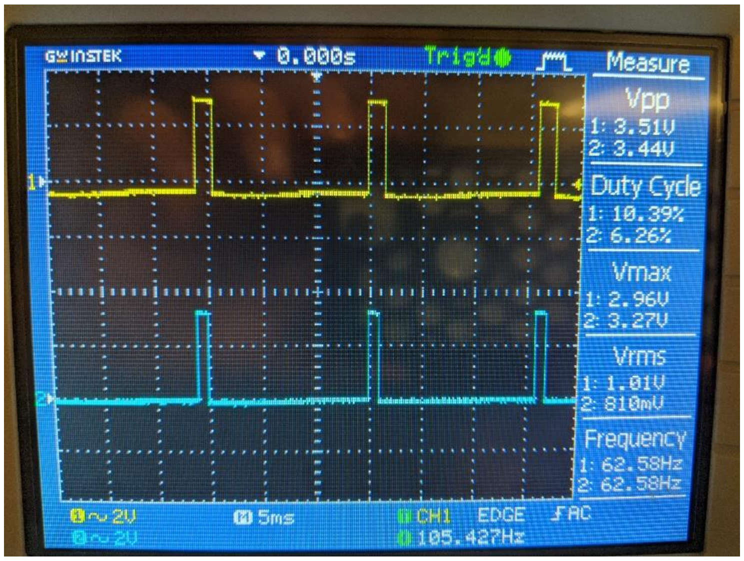

Fig. 6f – State5, when pushbutton is pressed once, pulse width increases.

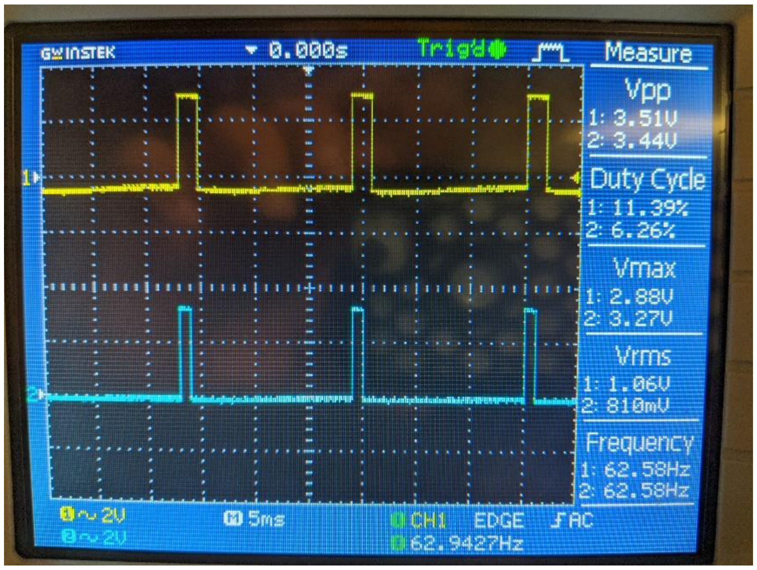

Fig. 6g – State6, when pushbutton is pressed once, pulse width increases.

Code

Frequency Counter 50Mhz input to 128 kHz

```
library IEEE; --Frequency Counter - Uses 50Mhz internal clock to generate 
128kHz frequency
use IEEE.STD_LOGIC_1164.ALL;

entity clk128kHz is
    Port (
        clk     : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end clk128kHz;
```

```
architecture Behavioral of clk128kHz is
    signal temporal: STD_LOGIC;
    signal counter : integer range 0 to 194 := 0;
begin
    freq_divider: process (clk) begin

        if rising_edge(clk) then
            if (counter = 194) then
                temporal <= NOT(temporal);
                counter  <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    clk_out <= temporal;
end Behavioral;
```

Finite State Machine – Controller for Servo Position, 7-bit Binary Output

```
LIBRARY ieee; --Servo Position Controller, 7-bit Binary Output
USE ieee.std_logic_1164.ALL;
ENTITY FSMcontroller IS
      PORT ( increment, dir: IN STD_LOGIC;
      Q : OUT STD_LOGIC_VECTOR (6 downto 0));
END FSMcontroller;

ARCHITECTURE arc of FSMcontroller IS
      TYPE state_machine IS (S0, S1, S2, S3, S4, S5, S6);
      SIGNAL state: state_machine;

BEGIN
      PROCESS(increment)
      BEGIN
            IF increment'EVENT and increment ='1' THEN
                  IF dir ='1' THEN
                        CASE state IS
                              WHEN S0 => state <= S1;
                              WHEN S1 => state <= S2;
                              WHEN S2 => state <= S3;
                              WHEN S3 => state <= S4;
                              WHEN S4 => state <= S5;
                              WHEN S5 => state <= S6;
                              WHEN S6 => state <= S0;
                        END CASE;
                  ELSE
                        CASE state IS
                              WHEN S0 => state <= S6;
                              WHEN S1 => state <= S0;
```

```
                                            WHEN S2 => state <= S1;
                                            WHEN S3 => state <= S2;
                                            WHEN S4 => state <= S3;
                                            WHEN S5 => state <= S4;
                                            WHEN S6 => state <= S5;
                                   END CASE;
                          END IF;
                 END IF;
        END PROCESS;

        WITH state SELECT
                 Q <= "0000000" WHEN S0, --0
                         "0010101" WHEN S1, --21
                         "0101010" WHEN S2, --42
                         "0111111" WHEN S3, --63
                         "1010100" WHEN S4, --84
                         "1101001" WHEN S5, --105
                         "1111111" WHEN S6; --127
END arc;
```

PWM Generator for Servo Position

```
        library IEEE; -- PWM Servo Control
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity servo_pwm is
    PORT (
        clk   : IN  STD_LOGIC;
        pos   : IN  STD_LOGIC_VECTOR(6 downto 0);
        servo : OUT STD_LOGIC
    );
end servo_pwm;

architecture Behavioral of servo_pwm is
    -- Counter, from 0 to 2559.
    signal cnt : unsigned(10 downto 0);
    -- Temporal signal used to generate the PWM pulse.
    signal pwmi: unsigned(7 downto 0);
begin
    -- Minimum value should be 1ms.
    pwmi <= unsigned('0' & pos) + 128;
    -- Counter process, from 0 to 2559.
    counter: process (clk) begin
        if rising_edge(clk) then
            if (cnt = 2559) then
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;
    -- Output signal for the servomotor.
    servo <= '1' when (cnt < pwmi) else '0';
end Behavioral;
```

Block Diagram

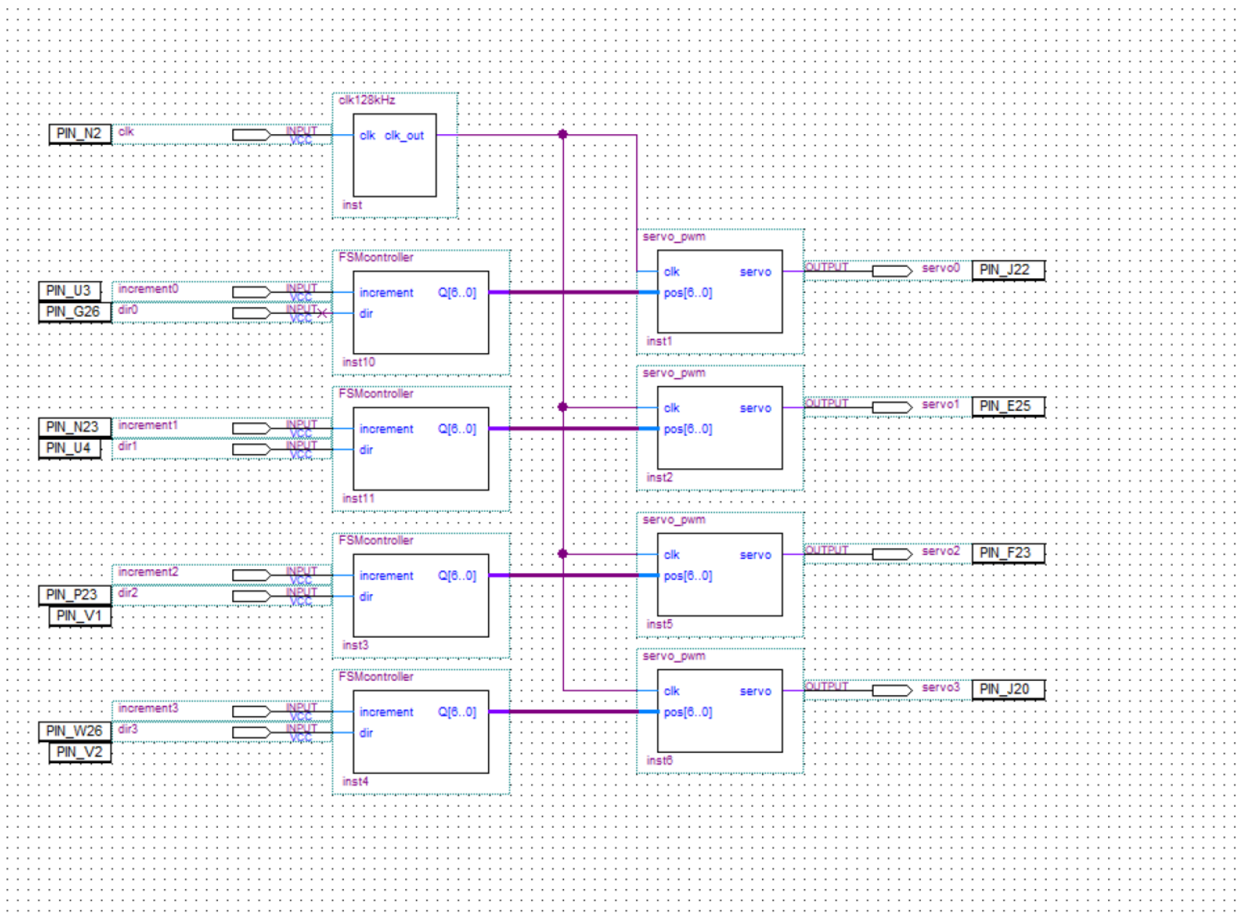

Fig. 7 – Quartus II Block Diagram of System – A single frequency divider component outputs to four separate PWM generators. Each PWM generator component is in turn controlled by a separate Finite State Machine component that applies a specified 7-bit binary output based on input from the user.

Pin Assignments

To	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
clk	Input	PIN_N2	2	B2_N1		
dir0	Input	PIN_U3	1	B1_N0		

dir1	Input	PIN_U4	1	B1_N0		
dir2	Input	PIN_V1	1	B1_N0		
dir3	Input	PIN_V2	1	B1_N0		
increment0	Input	PIN_G26	5	B5_N0		
increment1	Input	PIN_N23	5	B5_N1		
increment2	Input	PIN_P23	6	B6_N0		
increment3	Input	PIN_W26	6	B6_N1		
servo0	Output	PIN_J22	5	B5_N0		
servo1	Output	PIN_E25	5	B5_N0		
servo2	Output	PIN_F23	5	B5_N0		
servo3	Output	PIN_J20	5	B5_N0		

Troubleshooting:

While doing the project we came across one problem, our problem was the proper input frequency was just half and our calculations were a little off. Because of this minor error, our movements with our servos were smaller than how they were supposed to. After redoing the calculations and using an oscilloscope to check the proper input frequency we got it to the full 128 kHz that we needed and that solved the problem with the movements in our servo motor.

Conclusion

The toughest part about the project was researching and understanding how the code worked after looking at many examples of the code we finally understood how it worked. Thereafter the results from our calculations had to be checked with our oscilloscope. After that, the rest of our project went smoothly we learned a lot about PWM and FPGA. We also got a further understanding of the code used for the board.

References

https://electronics.stackexchange.com/questions/234788/micro-standard-servo-resolution
https://www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL
https://www.codeproject.com/Articles/443644/Frequency-Divider-with-VHDL
https://howtomechatronics.com/tutorials/arduino/diy-arduino-robot-arm-with-smartphone-control/