UNIVERSITY OF LEEDS

ELEC5566M

FPGA Design for System-on-Chip

N-channel 8-bit Servo Controller

Phaneeshwar Mallakkagari

SID: 201387234

M.Phaneeshwar
SID: 201387234

Contents

Page No.

1. Abstract

This report covers the design, implementation, and verification of an N-channel 8-bit Servo-Controller programmed using Verilog in FPGA. The model is highly parameterized with a range of different parameters used in both top level entity and other submodules and allows for a wide range of clock frequencies. The implemented design was verified using a simulation tool and Hardware FPGA Chip.

2. Introduction

The design aims to control multiple Digital Servo Motors using Behavioral Verilog programming language. The Servo Motor used is TowerPro's SG90 Digital Servo Motor. A single PWM channel entirely controls this servo motor that has a fixed period along with a duty cycle in the range of 0.5ms to 2.5ms. With this varying duty cycle, the output of the PWM channel controls the position of the servos. The duty cycle of the signal can be changed using the 8-bit input variable where 0 represents 0.5ms, and 255 denotes a 2.5ms pulse. This task was demonstrated and controlled the servo motors by utilizing the switches and push buttons that are embedded on the De1-SoC development board. The constructed design code is feasible for the users to control various parameters like the number of channels, clock frequency, clock divided frequency, minimum, and maximum duty cycle.

3. Overall Module Design

The Circuit design I have built can be used for any digital servo motors. Although there are few constraints in the design, it is feasible for normal operations. The user has the flexibility to vary different parameters according to the requirements to obtain desired results at the output. This is one of the advantages of this circuit as it is not meant for only one set of values. The constructed design has various functional modules because it would be more comfortable for the implementation and testing of the circuit. Hence, I have divided different functional units and interconnected them in the main module.

4. Code Design

4.1 Top Level Entity

The Top-level entity schematic is designed to visualize the submodules and establish interconnections between them. The design would utilize the PWM channel along with a frequency divider, Address decoder, and clog2 (N) header. A wide range of input clock frequencies can be used, and the frequency divider supports the clock to PWM channel with a fixed rate. The user can set various parameters like minimum duty cycle, maximum duty cycle, the period of the duty cycle, number of channels, and clock enable signal required for the PWM. This design also instantiates N- number of PWM channels. The clog2 header used in the design is to convert N number of PWM channels into address bits. Furthermore, the address decoder used is to access different servo controller channels individually. Figure 2 (Appendix) shows the schematic design of the servo controller.

4.2 Frequency Divider

This module is to design a clock frequency convertor, to obtain the desired output clock for the given input clock frequency. The required output frequency and the input clock frequency are declared as parameters in this module. As per the requirements, the duty cycle should range from 0.5ms to 2.5ms for a resolution of the 8-bit duty cycle. The Minimum clock frequency is calculated to meet the requirements. It is given as follows,

Min required frequency = 256/ (0.0025-0.0005) = = 128Khz

Hence in my original design, I have set the default value for the clock enable signal as 128Khz, and the default value for the input clock is 50Mhz. The user can change these parameter values according to his requirements. The design of this model achieves clock enable frequency or required frequency at the output for the given input clock by determining the number of single clock cycles. For 50Mhz frequency, we would obtain 50 million single clock cycles in one second, to obtain 128 thousand cycles in one second from 50Mhz frequency is given as follows,

Number_of_clk_cycles = 50000000/128000 = 390.6

When we consider the floor of the value, we obtain 390

The value obtained determines the period of one cycle of 128Khz clock frequency. The calculated value is divided into half of the original value, as the first half of the count is to set logic one at the output, and the remaining half of the count is set to logic zero. Varying a 50% duty cycle of the output is achieved by using a counter. This design supports for any values of frequencies that are greater than twice the clock enable signal. This is one of the major constraints of the design. If the clock enable signal is 128Khz, then the input clock should have a frequency higher or equal to 256Khz. I have not solved this constraint as it requires more RTL logic resources, and another limitation of the frequency divider module is that the calculated count value is always rounded off to an integer. This would lead to timing errors at the output, and these errors can be analyzed and viewed using the simulation tool.

4.3 Address Decoder

This modular design is used to decode the given address into the required PWM channel. For this design, I have constructed an N-channel Demultiplexer circuit for data distribution since they transmit the same data to different channels. Here the Latch is the input for this D-mux. When the latch and input address is given to the module, it directs the value of the latch for the given channel address. The latch is essential to design requirements for the servo controller as it is used to store the duty cycle in the channel and maintain the phase difference between them. The output phase of the servo channel can be varied by latching different duty cycles in different at different time intervals. Address input is given as a select line for the Demultiplexer, which connects the input to the required PWM channel. I have made the input pin of this module as active low because

it is helpful while testing it on the hardware as the push buttons on the FPGA board are active low.

4.4 PWM Channel

This is the major module that controls the phase and power supply to the servomotor. In this module structure, the period of PWM is calculated using the counter. If the input clock frequency of the PWM channel is 128Khz and required period of the output signal is 20ms, then the maximum value of the counter is given by,

Pwm_clk_period = 128Khz * 20ms = 2560

So, the counter must count upto Pwm_clk_period value to complete one cycle period at the output. Here one cycle lasts for 20ms. According to the given requirements, the minimum duty cycle needed to be maintained high at the output is 0.5ms, and the maximum duty cycle is 2.5ms. Hence, I have calculated the delay to maintain a minimum and a maximum width of the duty cycle at the output is given as follows,

Min_duty_width = 128Khz * 0.5ms = 64
Max_duty_width = 128Khz * 2.5ms = 320
Range_duty_width = 128Khz *(2.5ms-0.5ms) = 256
And this value equals to an 8-bit duty cycle input

This module has 8-bit duty cycle input; therefore, for every increment in the input data, there will be a rise in the width of the duty cycle at the output of the channel. The output of the circuit is always high upto 64 clock cycles and always end before 320 clock cycles.

Clocks_output_high = 64 + input_dutycycle

Latch input is used to load the value of the 8-bit duty cycle, which helps to maintain the phase of different channels. And the phase difference between the channels can be varied by latching different duty cycle values at different clock intervals.

4.5 Clog2(N) header file

I have defined the header file, which is used to calculate the width of an address signal by considering the number of channels as input. It finds the result by taking ceil (log2 (*N*)). And I have called this header in the main file.

```
`include "clog2.v"

module servomotor8bit #(
```

Figure 1: Header definition in the top-level model

5. Verification

5.1 Simulation

This part of the task is very important in analyzing any Verilog design. The simulation tool used is very useful in performing timing analysis on the constructed circuit, and to understand the behavioral of the channel output for the given input signals. The testbench of the module is made and verified for different clock frequencies and obtained satisfactory results. I have tested for different values of duty cycles and also changed the minimum and maximum width of the duty cycle at the testbench, For Example, instead of setting the range of duty cycle from 0.5ms to 2.5ms I had changed it to 5ms to 7ms, and expected results are obtained. In a similar fashion, this design was tested for different input clock frequencies and clock enable signals.

5.2 Hardware Testing

5.2.1 Waveform Testing

The 4-PWM channels were implemented on board. Furthermore, it was compiled and tested using the Cyclone V FPGA. The output pins on the board are connected to the oscilloscope and the waveforms were verified for input clock frequency of 50Mhz. Expected waveforms were obtained on the oscilloscope. As it is known that the oscilloscope has only two probes, so we can only view two PWM channels at a time. All the channels are verified by considering two channels at a time. Oscilloscope waveforms are shown in the figure

5.2.2 Servo Testing

The PWM channel outputs were tested using TowerPro's SG90 Digital Servo Motor. The switches control the 8-bit duty cycle, and the first eight switches on the board are assigned to the input duty cycle, and the last two switches are assigned to the 2-bit address input. The latch signal is given to the push button, which is used to control the servo motor. The hardware verification was achieved by instantiating the servo motor with 4 PWM channels along with two address bits, and the clock frequency is set to 50 Mhz. The output pins are connected to the GPIO0 port on the board. After the pin assignment and uploading the design into the chip, the servo motor was tested for different values of duty cycle and address bits.

6. Conclusion

In conclusion, the code is designed to run for different digital servo motors and achieved desired results at the output by testing both in hardware and using the simulation tool. Some of the challenges were faced while instantiating the submodules and interconnecting the different functional modules in the top-level entity. The highly parameterized modules were designed to give flexibility for the code to compile for different requirements.

M.Phaneeshwar
SID: 201387234

7. Appendix

7.1 Top Level Schematic Design

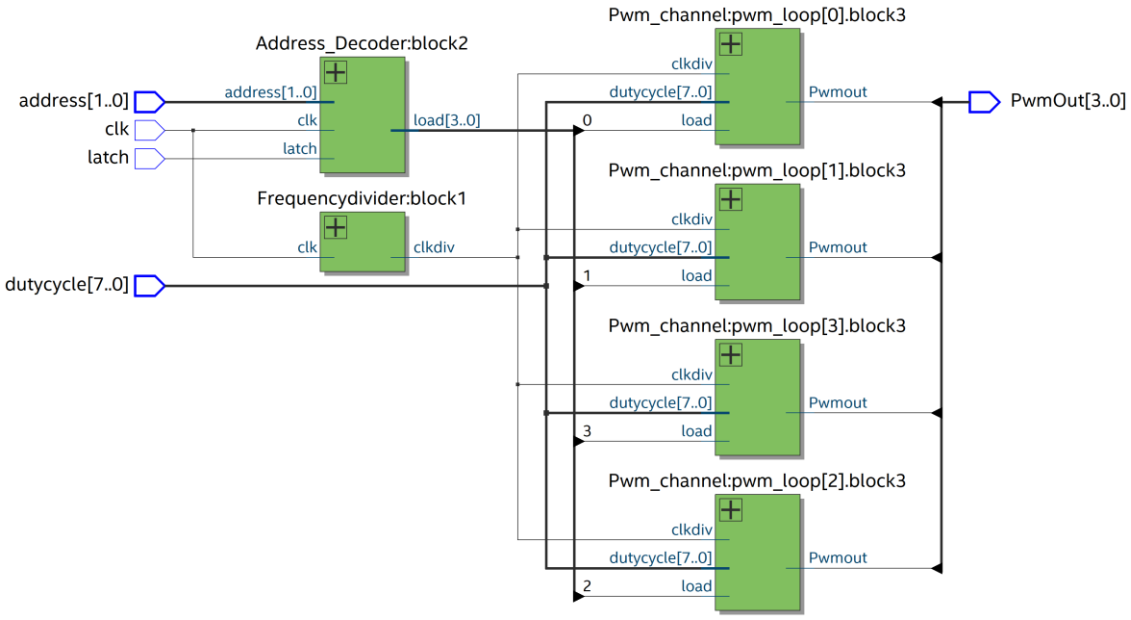

Figure 2: Top level design schematic

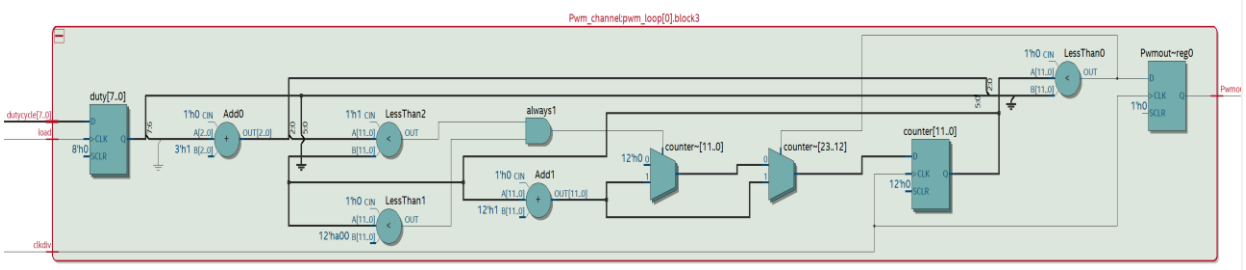

Figure 3: PWM_channel Design

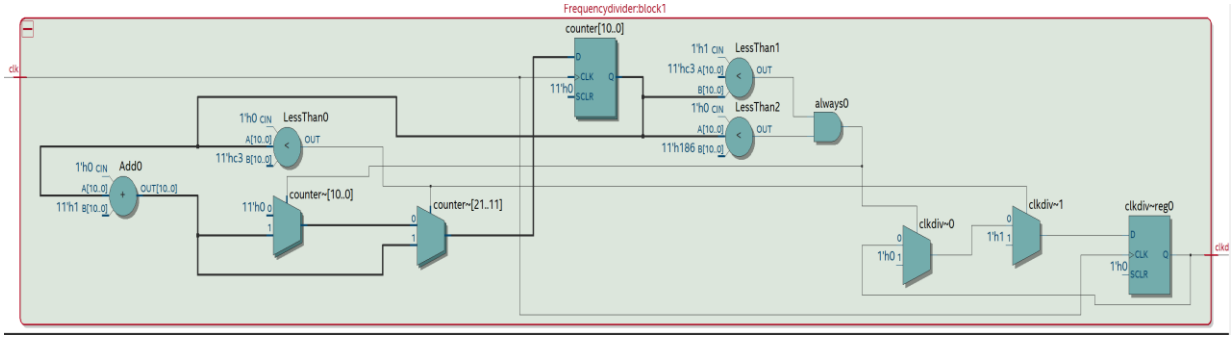

Figure 4: Frequency_Divider

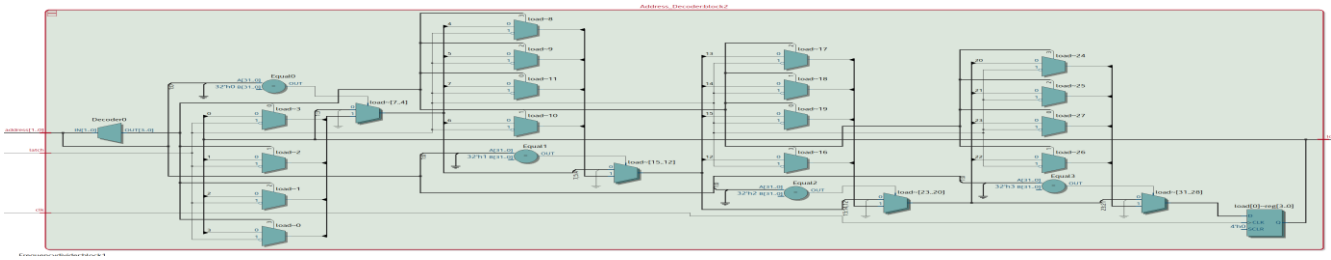

Figure 5: Address Decoder

7.2 Waveforms Using Oscilloscope

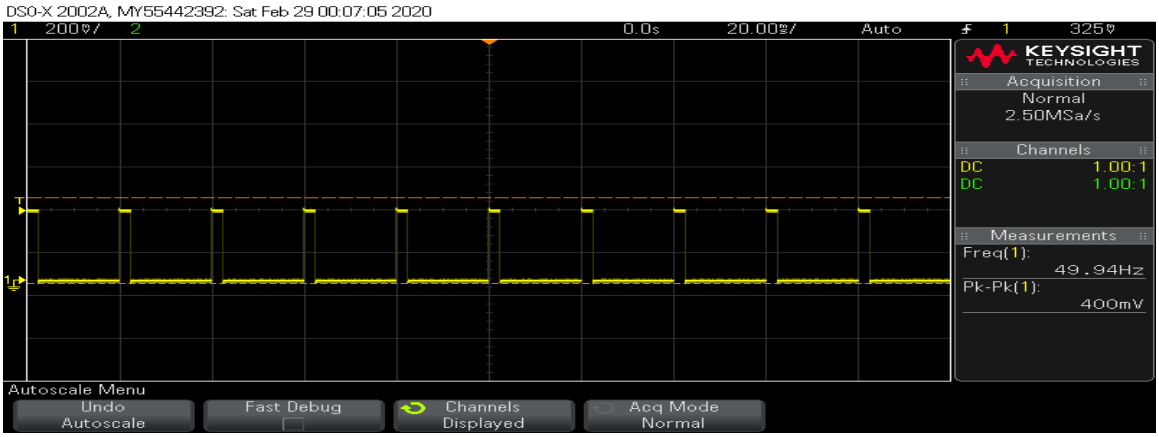

Figure 6: waveform showing for 4 channel Duty cycle = 255, address = 2'b00

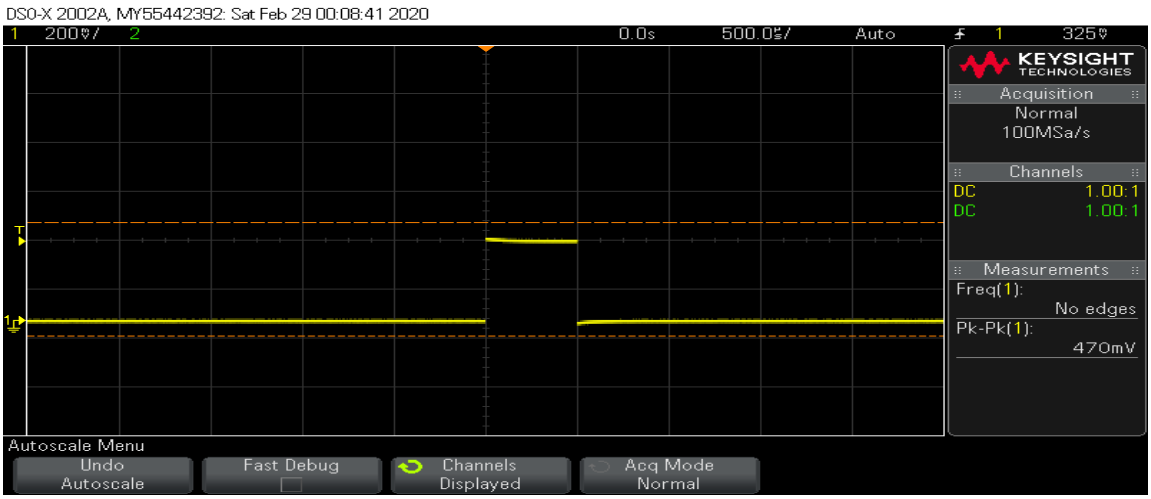

Figure 7: waveform showing for 4 channel Duty cycle = 0, address = 2'b01

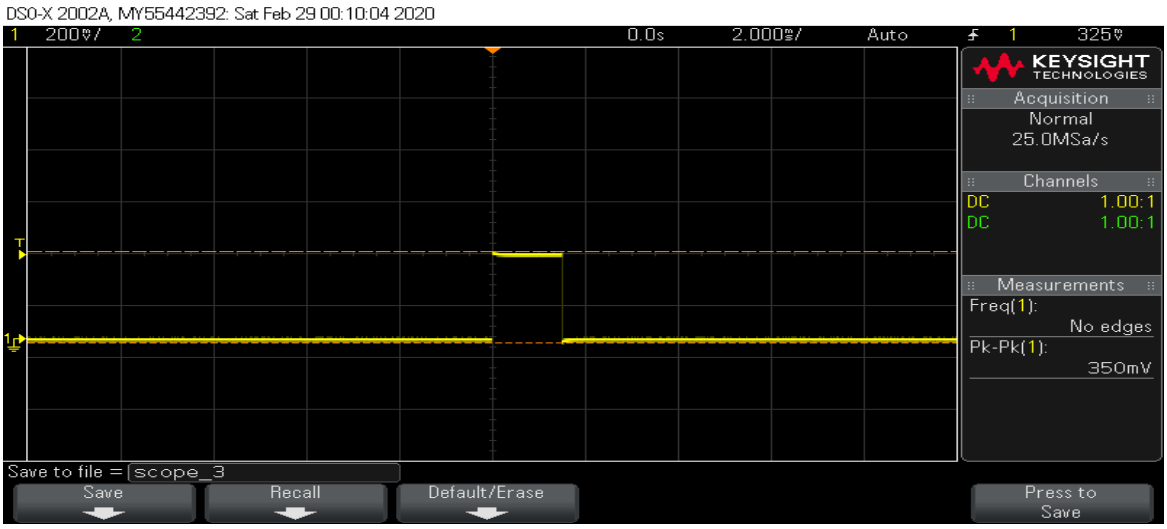

Figure 8: waveform showing for 4 channel Duty cycle = 127, address = 2'b10

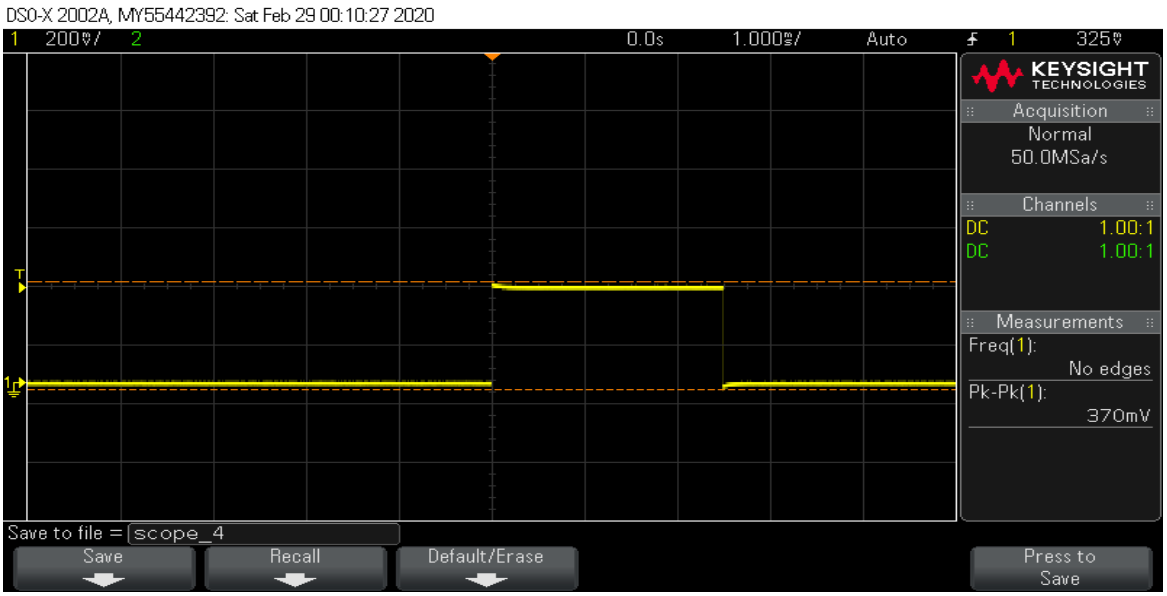

Figure 9: waveform showing for 4 channel Duty cycle = 255, address = 2'b11

7.3 Simulation

7.3.1

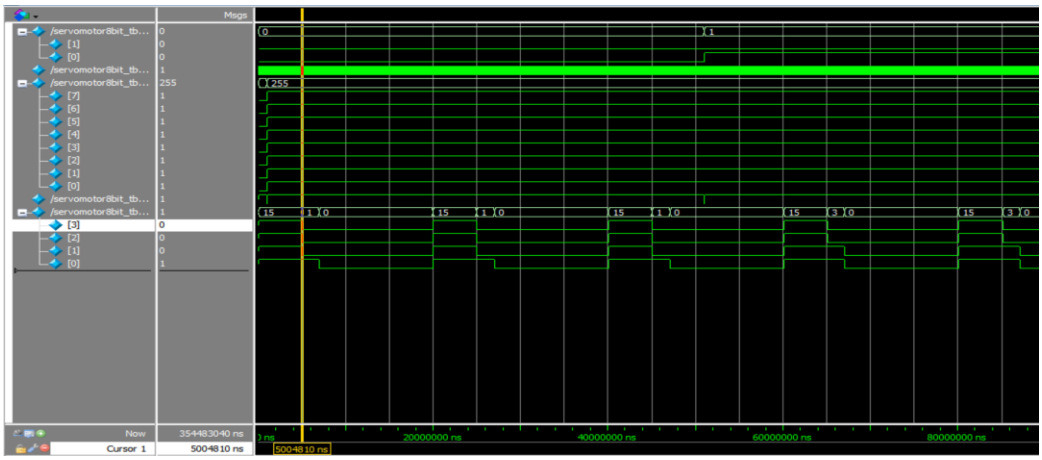

Figure 10: Simulation Output for servomotor8bit_tb

```
initial begin                                                    /

          clk = 1'b0;                                            //
          latch = 1;                                             //

           #1000000;                                             //
           address = 2'b00;                                      //
           dutycycle = 8'hFF;                                    //
           latch = 0;                                            //
           @(posedge clk);
           latch = 1;

           #50000000;
           address = 2'b01;
           dutycycle = 8'hFF;
           latch = 0;
           @(posedge clk);
           latch = 1;

           /*#50000000;
           address = 2'b10;
           dutycycle = 8'hF0;
           latch = 0;
           @(posedge clk);
           latch = 1;

           #50000000;
           address = 2'b11;
           dutycycle = 8'hFF;
           latch = 0;
           @(posedge clk);
           latch = 1;
           */
           $display("%b ns\tSimulation Finished",$time);

   end
```

Figure 11: Testbench Input

7.3.2

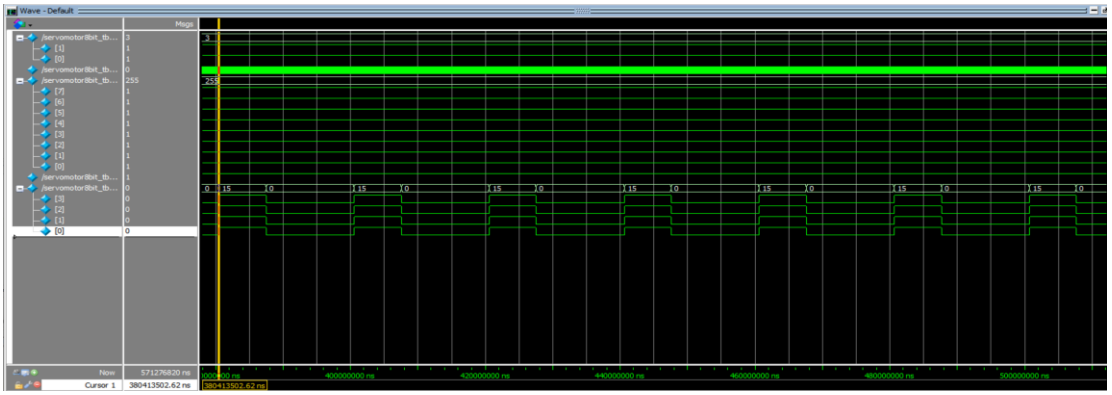

Figure 12: Simulation Output for servomotor8bit_tb

```
initial begin

        clk = 1'b0;
        latch = 1;

         #1000000;
         address = 2'b00;
         dutycycle = 8'hFF;
         latch = 0;
         @(posedge clk);
         latch = 1;

         #50000000;
         address = 2'b01;
         dutycycle = 8'hFF;
         latch = 0;
         @(posedge clk);
         latch = 1;

         #50000000;
         address = 2'b10;
         dutycycle = 8'hFF;
         latch = 0;
         @(posedge clk);
         latch = 1;

         #50000000;
         address = 2'b11;
         dutycycle = 8'hFF;
         latch = 0;
         @(posedge clk);
         latch = 1;

         $display("%b ns\tSimulation Finished",$time);

    end
endmodule
```

Figure 13: Testbench Input

7.4 Hardware Implementation

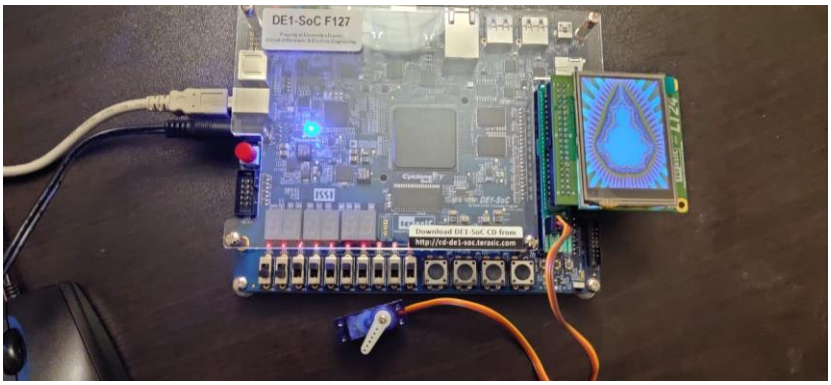

Figure 14: Hardware, dutycycle = 255, address = 2b00, Angle = 180

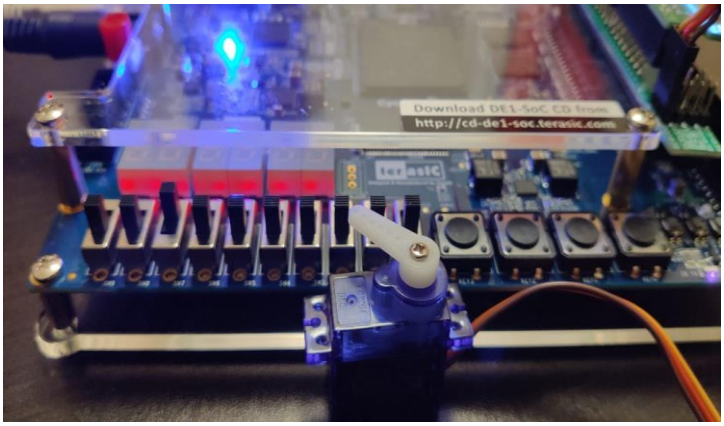

Figure 15: Hardware, dutycycle = 127, address = 2b00, Angle = 90

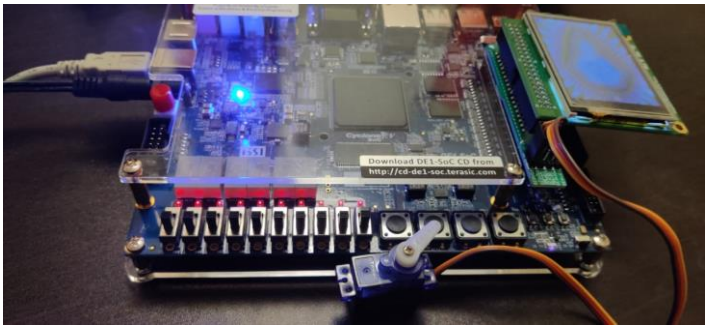

Figure 16: Hardware, dutycycle = 0, address = 2b00, Angle = 0

7.5 Code

7.5.1 servomotor8bit.v

```
`include "clog2.v"

module servomotor8bit #(          // All the parameters mentioned below can be varied by the user as
//per his requirements

parameter min_duty_cycle = 0.0005,//min duty cycle required for the zeroth bit,

parameter timeperiod = 0.02,     // to set timeperiod of the output signal

 parameter required_clk_frequency = 128000, // clock enable signal used as input clock for the PWM

parameter n_channels = 4,            // To set Number of PWM channels

        parameter clk_frequency = 50000000,   // to set the input clock
        parameter nbits = `clog2(n_channels) //  to determine the number of address bits based on
number of channels
        )(
        input [nbits-1:0] address,     // input the address line
        input latch,                // latch signal to load the duty cycle value
        input clk,                  // input clock
        input [7:0] dutycycle,        // 8-bit duty cycle
        output [n_channels-1:0] PwmOut  // N-Pwm channel
        );

wire clkdiv;             // used to connect frequency divider model and pwm channel
wire [n_channels-1:0] load;  // it is used to connect address decoder output to the pwm channel

Frequencydivider # (                        // Instantiating frequency divider module
    .required_clk_frequency (required_clk_frequency ),
    .clk_frequency (clk_frequency )
     ) block1 (
                .clk (clk ),
                .clkdiv (clkdiv )
         );
```

```
    Address_Decoder # (                    // Instantiating Address Decoder
        .channels (n_channels),
                    .nbits    (nbits    )
                    ) block2 (
                    .clk      (clk      ),
                    .address (address    ),
                    .latch    (latch     ),
                    .load     (load      )
                    );

    genvar i;  // create a variable, used in the generate block

    generate // generate block helps us to instantiate N number of channels.
        for (i=0 ; i < n_channels ; i = i + 1) begin: pwm_loop

                Pwm_channel #(
                                        .required_clk_frequency (required_clk_frequency ),   // give required
    parameters to the model
                                .timeperiod (timeperiod),
                                 .min_duty_cycle (min_duty_cycle )
                                ) block3 (
                                .clkdiv  (clkdiv ),
                                .load    (load[i] ),
                                .dutycycle (dutycycle ),
                                .Pwmout (PwmOut[i] )
                                        );
    end
    endgenerate
    endmodule
```

7.5.2 Frequencydivider.v

```
module Frequencydivider #(

    parameter required_clk_frequency = 128000, //default frequency value obtained at the output of the
//frequency divider circuit,

parameter clk_frequency = 500000000    // default input clock frequency.

                    ) (                        // Parameter values can be changed according to the user
//requirements

                     input clk,

                     output reg clkdiv

                     );

//*************************************************************************************************
***

//if we required ceil of the calculated count then we can use the below logic.

//*************************************************************************************************
***
```

```
//*************************************************************************************************
***

// ceil value

//localparam integer clkcycle = ((clk_frequency + required_clk_frequency - 1) / required_clk_frequency);

//*************************************************************************************************
***



//floor value

localparam integer clkcycle = clk_frequency / required_clk_frequency; // to count the number of single
cycles

reg[10:0] counter = 0; // initialize the counter value to zero



always @(posedge clk) begin                    //for every positive edge of the clock

  if (counter < (clkcycle / 2)) begin

            counter <= counter + 1;              // Increment the counter value

                clkdiv <= 1;                      // Set the output clk to high upto half of the claculated
clkcycle value

                end

          else if (counter >= (clkcycle / 2) && counter < clkcycle ) begin  // to check if it is less than
clkcycle and greater than half of clkcycle

            counter <= counter + 1;              // Increment the counter value

                        clkdiv <= 0;                    // Set output to zero for next half of the cycle

                        end

                else begin

                  counter <= 0;

                end

end

endmodule
```

7.5.3 Address_Decoder.v

```
module Address_Decoder # (    // this module behave like a dmux

   parameter channels = 4, // number of channels are parametrized and 4 is set as default value

                parameter nbits = 2 // width of address bits and 2 is set as default value

                )(
```

```
                input clk,                  // input clock
                input [nbits-1:0] address,      // input address line
                input latch,                // latch signal
                output reg [channels-1:0] load   // output of the Address decoder, the width of the
output is equal to number of channels
                );
integer i;    // this variable is used in the for loop
always @(posedge clk) begin          // check for every positive edge of the clock
for (i = 0; i < channels; i = i + 1) begin   // this loop is used to connect appropriate output line to the input
//latch
  if (address == i) begin
 load[address] <= ~latch;          // latch value is negated and given to the addressed channel, as it would
//be useful when testing the hardware
        end
        else begin
        load[i] <= 0;         // set bit zero for remaining output lines
        end
        end
end
endmodule
```

7.5.4 Pwm_channel.v

```
module Pwm_channel # (
    parameter required_clk_frequency = 128000, //input clock frequency for the pwm channel
    parameter timeperiod = 0.02,            // timeperiod of the signal
          parameter min_duty_cycle = 0.0005        // minimum duty cycle
          )(                          // Parameter values can be changed by the user in the top level entity
          input clkdiv,         // clock input, it comes from frequency divider module
          input load,          // to latch the duty cycle
          input [7:0] dutycycle,  // 8-bit dutycycle, it is used to change the width of the duty cycle
          output reg Pwmout      // Pwm channel output is defined
          );


reg [7:0] duty = 0;       // intialize a seperate dummy variable, used to store the value of the duty cycle
reg [11:0] counter = 0;    // counter is used to obtaine required delay
localparam integer clkdiv_period = (required_clk_frequency * timeperiod);     // calculate the time period
```

```
localparam integer mincycle_count = (required_clk_frequency * min_duty_cycle); // to calculate
//mininmum duty cycle to be maintained at the output


always @(posedge load ) begin  // it checks whenever the latch signal is given
    duty <= dutycycle;    // if it detects the latch signal it would store the value
end


always @(posedge clkdiv) begin    // check for every positive edge of clock


      if ( counter < (mincycle_count + duty)) begin  // to maintain min 0.5ms and in addition duty value is
added. so counter is used to set that much
                Pwmout <= 1;                    // delay and the output should be high for that delay
                        counter<= counter + 1;          // increment the counter
                       end


                    else if ((counter < clkdiv_period) && (counter >= (mincycle_count + duty))) begin //
for values greater than duty cycle and lesser than the total period
                      Pwmout <= 0;              // set logic 0 for remaining time of period
                        counter<= counter + 1;      // increment the counter
                       end
                  else begin
                    counter <= 0;       // make it zero for remaing time
                        Pwmout <= 0;
     end
   end
endmodule
```

7.5.5 clog2.v

```
`ifndef _clog2_h_        // I had defined it has header, so that it can be easily accesed in the main model
`define _clog2_h_        // this function is used to convert number of channels into address bits
`define clog2(x) ( \
 ((x) <=    2) ? 1: \
 ((x) <=    4) ? 2: \
 ((x) <=    8) ? 3: \
 ((x) <=   16) ? 4: \
```

```
 ((x) <=   32) ? 5: \
 ((x) <=   64) ? 6: \
 ((x) <=  128) ? 7: \
 ((x) <=  256) ? 8: \
 ((x) <=  512) ? 9: \
 ((x) <= 1024) ? 10: \
 ((x) <= 2048) ? 11: \
 ((x) <= 4096) ? 12: 16)
`endif
```

7.5.6 servomotor8bit_tb.v

```
`timescale 1 ns / 100 ps
module servomotor8bit_tb;          // define the testbench module
                reg [1:0] address = 2'b00;        // create a reg for inputs
                reg clk = 0;
                reg [7:0] dutycycle = 8'h00;
                reg latch = 1'b0;
                wire [3:0] PwmOut;      // wire for the outputs
                localparam min_duty_cycle = 0.005;                  // initialize the parameter mininmum duty cycle to 0.5ms or any other value
                localparam timeperiod = 0.02;                       // initialize the parameter timeperiod, to set the period of the output signal
                localparam n_channels = 4;                          // to define number of channels
                localparam nbits = 2;                               // define number of bits
                localparam required_clk_frequency = 128000;                         // this frequency defined will be input to the pwm channel
                localparam clk_frequency = 50000000;                // to set input clock frequency to 50 Mhz
 servomotor8bit # (                                 // Instantiate the main module
        .min_duty_cycle (min_duty_cycle),
                .timeperiod  (timeperiod),
                .required_clk_frequency (required_clk_frequency),
                .n_channels (n_channels),
                .clk_frequency (clk_frequency),
                .nbits  (nbits)
```

```
    ) dut (
     .address (address),
                .clk    (clk    ),
                .dutycycle   (dutycycle ),
                .latch (latch),
                .PwmOut   (PwmOut    )
                );
 localparam frequency = (1000000000/(2*clk_frequency));    // this will set the pulse width of the clock in nano seconds
 always #frequency clk = ~clk;                    // to toggle the clock input
 initial begin                                  // this is where the execution of the inputs will begin
                clk = 1'b0;                      //set input clock to 0 initially
                latch = 1;                      // latch is set to logic one because it is active low
                        #1000000;                        // delay
                        address = 2'b00;                   //select the pwm channel by giving the address
                        dutycycle = 8'hFF;                  // set the duty cycle
                        latch = 0;                      // give latch 0, as it is active low, output is high
                        @(posedge clk);
                        latch = 1;

                        #50000000;
                        address = 2'b01;
                        dutycycle = 8'hFF;
                        latch = 0;
                        @(posedge clk);
                        latch = 1;
                        #50000000;
                        address = 2'b10;
                        dutycycle = 8'hF0;
                        latch = 0;
                        @(posedge clk);
                        latch = 1;
                        #50000000;
                        address = 2'b11;
```

```
                dutycycle = 8'hFF;
                latch = 0;
                @(posedge clk);
                latch = 1;
                $display("%b ns\tSimulation Finished",$time);
        end
endmodule
```

7.5.7 pwm_tb.v

```
`timescale 1 ns / 100 ps
module pwm_tb;
 reg clkdiv = 0;
 reg load = 1'b0;
 reg [7:0] dutycycle = 8'b00000000;
 wire Pwmout;
 pwm dut (
  .clkdiv (clkdiv ),
        .load   (load   ),
        .dutycycle (dutycycle ),
        .Pwmout (Pwmout )
        );
        localparam frequency = 1;;
        always #frequency clkdiv = ~clkdiv;
        initial begin
                clkdiv = 1'b0;
                load = 0;
                #30000000;
                dutycycle = 8'b10000001;
            load = 1;
                @(posedge clkdiv);
                load = 0;
                $display("%b ns\tSimulation Finished",$time);
 End
 endmodule
```