

**School of Electronic**

**and Electrical Engineering**

**ELEC5566M: FPGA Design for System on Chip**

ASSIGNMENT 1  
N-Channel 8-bit Servo Controller  
  
  
  
**STUDENT NAME:** SHRAJAN BHANDARY  
**STUDENT ID :** 201289634

**N-CHANNEL 8-BIT SERVO CONTROLLER**

**1. ABSTRACT**

The project describes a procedure to develop an IP core for N-channel 8-bit Servo Controller using Behavioural Verilog HDL. The N-channel 8-bit servo controller was built with a hierarchical approach by instantiating multiple low-level entities as submodules. The top-level parametrised servo controller module was tested and verified using both test-benches and hardware implementation.

**2. INTRODUCTION**

A screenshot of a cell phone

Description generated with very high confidenceServomotors are linear or rotary actuators that render accurate control of linear or angular positions. The motors are embedded with feedback sensors to control the positions. Dedicated controller circuits are required to achieve desired positions [1]. The servomotors used in the project are of the type SG90 and each of them consists of a DC motor operating at 5V. In addition, the shaft of the motor is coupled to a potentiometer to retrieve the position and control the actuation. The inputs to these servomotors are Pulse Width Modulated (PWM) signals and the signals are usually generated by a controller with the aid of a clock signal. SG90 servomotors can actuate to angular positions ranging from -90 degrees (0 degrees) to +90 degrees (180 degrees). In order to control the servomotor, the period of the PWM signals must be a fixed value of 20ms. The minimum ON time should be 0.5ms to actuate to an angular position of -90 degrees and the maximum ON time should be 2.5ms to actuate to an angular position of +90 degrees. The required PWM input signal to the servomotor along with the different ON periods and their corresponding angular positions is depicted in figure 1.

**Figure 1:** Controller output PWM signals – ON Time vs. Angular Position.

To generate the PWM signals and develop the N-channel 8-bit servo controller, the on-board clock of the Cyclone V FPGA system, oscillating at 50 MHz, must be utilized. The ON time can be varied by the manipulating the 8-bit input signals connected to 8 different slide switches of the FPGA board. The positions of the connected servomotors are individually loaded onto their corresponding PWM ports using the address selector and the latch signal which are connected to 2 slide switches and 1 push button of the Cyclone V board, respectively. The N-channel 8-bit servo controller can also be simulated using test benches with a clock frequency varying from 128 kHz to 100 MHz.

**3. THE PROPOSED CONTROLLER**

The N-channel 8-bit Servo Controller was developed by instantiating multiple submodules. These low-level entities perform different functions to produce the desired PWM signals. The submodules used are: N-bit counter, N-bit comparator, single servo controller (with fixed clock frequency) and frequency divider.

**3.1. CALCULATING THE REQUIRED CLOCK CYCLES**

Before delineating about the submodules, it is crucial to generate the exact period of the PWM signal. This can be achieved by implementing the controller circuit with a fixed input clock having a frequency of 128 kHz. The time period of each cycle generated by this clock circuit is 7.8125us. The number of clock cycles required to generate one complete PWM signal of period 20ms is:

= 2560 cycles…………………………………………… (1)

The number of clock cycles required to generate the minimum ON time of 0.5ms is:

= 64 cycles…………………………………………… (2)

The number of clock cycles required to generate the maximum ON time of 2.5ms is:

= 320 cycles…………………………………………… (3)

**3.2. SUBMODULES**

The two major submodules of the N-channel 8-bit Servo Controller are: the module that controls a single servo with a fixed frequency and the module that converts any incoming clock frequency to a fixed clock rate. These modules instantiate submodules of their own to perform the necessary operations.

**3.2.1. Single Servo Controller Submodule**

The module is capable of controlling a single servo using an 8-bit input and a fixed clock frequency of 128kHz. The first task of the module is to count the number of the cycles derived in equations 1, 2 and 3. This is achieved by instantiating a submodule called N-bit counter with a maximum count value of 2560 cycles and supply it with the clock. When the maximum value is reached, the counter value is reset to 0. The second task is to calculate the threshold clock cycle value (shown below in equation 4) where the PWM signal of the servo motor is HIGH. The equation for the threshold is given by:

THRESHOLD = 8-bit INPUT + 64……….………….…………(4)

The final task is to compare the threshold value with the generated clock cycles and then switch the PWM signal accordingly. The reset input actuates the servomotor to its default angular position of 0 degrees. The load input latches the value of the 8-bit duty cycle control onto the PWM output of the module. The implementation of the submodules is described in Appendix A – Section 1, 2 and 3. In order to verify the module, a test bench was developed. The test bench simulates an input clock signal with frequency of 128 kHz, a load signal, a reset signal and an 8-bit duty cycle control that changes its values after repeating for a discrete value of time.

**A picture containing screenshot

Description generated with high confidence**

**Figure 2:** Input and Output Waveforms of submodule single servomotor controller with 128 kHz clock using ModelSim.

It is evident from figure 2 that for different values of the duty cycle control (8-bit input), the PWM signals have different ON-time values with a fixed clock period. This submodule, as a standalone, is incapable of controlling a servomotor with the Cyclone V FPGA board, since the on-board clock has a frequency of 50 MHz.

**3.2.2. Frequency Divider Circuit Submodule**

The limitation of the single servomotor controller is the fact that it can only be implemented on a board with a 128 kHz clock frequency. This drawback can be overcome by designing a circuit that can convert any incoming clock frequency to the required value. The maximum frequency that could be injected into the module is 100 MHz and the minimum frequency is 128 kHz. The frequency divider circuit takes any incoming signal and divides it by the required frequency to obtain a factor called the toggle value. An N-bit counter is instantiated at this stage using a maximum counter limit equal to the toggle value. The output of the counter could then be used to generate an output clock signal that switches (LOW to HIGH or vice-versa) at every instance the maximum counter value is reached. However, the frequency of the resultant signal would be half the value as that of the required clock signal. The reason is that toggle value consists of both the ON cycles and the OFF cycles [2-3]. Therefore, dividing the toggle value by a factor of 2 will produce the desired fixed clock frequency of 128 kHz. The toggle value is given by:

Toggle value = ……………………………… (5)

A screenshot of a video game

Description generated with high confidence

**Figure 3:** Input and Output Waveforms of submodule frequency divider with input clock of 50 MHz clock using ModelSim.

The test bench simulates an input clock signal with frequency of 50 MHz and a reset signal. Figure 3 illustrates the working of the frequency divider with an input clock signal of 50 MHz frequency and the resultant output clock signal of 128 kHz frequency. The submodule is illustrated in Appendix A – Section 4.

**3.3. SERVO CONTROLLER WITH 50 MHz CLOCK**

The single servo controller with a parameterised clock frequency of 50 MHz was developed to test the hardware using the SG90 motor and Cyclone V FPGA board. In addition, this module was also tested using a test bench. The top-level entity is parameterised such that the input clock frequency can be changed. This module then instantiates the two submodules: servo controller with fixed clock frequency of 128 kHz and the frequency divider circuit module. The input clock signal is connected to the frequency divider module that outputs a fixed clock rate. This output clock signal is then supplied to the servo controller module that generates the PWM signal. The module is illustrated in Appendix A – Section 5. This module was tested using ModelSim by creating a test bench having a clock frequency of 50 MHz. The input duty cycle control and the load signals were changed after every 20ms to produce different PWM signals as shown in figure 4. The hardware verification was carried out with the input clock of the module connected to the 50 MHz of the FPGA board. The input duty cycle control was connected to the slide switches. The load signal and the reset signal were connected to two separate push buttons. However, the push buttons are active low and hence the signals were inverted in the module. The Verilog module is illustrated in Appendix A – Section 5.

A picture containing screenshot

Description generated with very high confidence

**Figure 4:** Input and Output Waveforms of the single servo motor using a frequency divider circuit.

**3.4. N-CHANNEL 8-BIT SERVO CONTROLLER**

The single servo test module implemented using 50 MHz input clock rate is hardwired to drive only one servomotor. However, the objective of the project is to develop an IP core that can render and utilize N independent channels. Therefore, the top-level module of the N-channel controller instantiates a parameterised number of servo addresses. This module has a parameterised clock input, a single 8-bit input to set the duty cycle, a load signal to latch the duty cycle value, an address input to update individual servos at a given instance and an N-bit PWM output. Since the number of channels is arbitrary, the servo selector address input must not have a fixed width. This is executed using a pre-processor macro - ceil(log2(N)) as described in Appendix A – section 6. Similar to the module described in section 3.3, the N-channel servo controller module also instantiates the frequency divider circuit. The next task is to initialize N number of servo controller module (section 3.2.1). This is accomplished by a ‘for-loop’ used within an advanced parametrisation block called ‘generate’. The for-loop iterates for values ranging from 0 to (N-1) and instantiates N number of servo controller modules. Furthermore, each instantiated module declared within the for-loop will have the same ports connected to them except for the PWM and load signals. The individual load signals, which control the PWM signals, are determined by the combination of values of the address input and latch signal as shown in figure 5.

A screenshot of a cell phone

Description generated with very high confidence

**Figure 5:** Instantiating ‘N’ modules and selecting the servo to be updated (Appendix A -Section 6).

A test bench was created for this module, similar to the one outlined in section 3.3, that simulates 4 different servo addresses, a 1-bit load signal and an 8-bit duty cycle control with an input clock signal of 50 MHz. The output waveforms depict 4 different values of PWM, each having different ON times with a fixed clock period as shown in Figure 6. This module was also tested using hardware verification by employing 3 servomotors as shown in figure 7. The Verilog module is described in Appendix A – Section 6.

A screenshot of a computer

Description generated with high confidence

**Figure 6:** Input and Output Waveforms of the N-channel 8-bit servo controller using ModelSim.

A circuit board

Description generated with very high confidence

**Figure 6:** Hardware implantation of N-channel 8-bit servo controller.

**4. CONCLUSIONS**

The project successfully establishes an N-channel 8-bit Servo Controller that can work with clock frequencies from 128kHz to 100MHz. The hierarchical structure that represents the complex module of the controller permits easy debugging of errors. Furthermore, simulation using test benches helps in understanding both the flow of the Verilog description and also any shortcomings that exist in the module.

**5. REFERNCES**

1. Wikipedia. Servomotor. [Online]. 2019. [Accessed 10 March 2019]. Available from: https://en.wikipedia.org/wiki/Servomotor
2. Ma, S., Guan, B., and Hou, L. Design of high-accuracy decimal frequency divider with Verilog-HDL. In: International Power, Electronics and Materials Engineering Conference, 2015, China. doi: 10.2991/ipemec-15.2015.16
3. Zhang, S. W., and Zhao, C. Design for realizing arbitrary fractional divider based on FPGA which duty cycle is up to 50%. Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation. Singapore. pp. 1653-1657, 2013.

**APPENDIX – A**

**Appendix - A consists of all the Verilog module files used in the project.**

|  |
| --- |
| **SECTION 1:** N-bit counter module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Verilog Module Name : N\_Bit\_Counter    Code Author : Shrajan Bhandary  Date Created : 26/02/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to count values ranging from 0 to required maximum value.      Once the maximum value is reached the counter is reset to 0. The counter      increments at every positive edge of clock signal and can be reset at every      positive edge of the reset signal i.e., when the reset button is pressed.      For this project the maximum value corresponds to the clock period (20 ms)      of the PWM Signal of the Servo Motor.    \*/  ////////////////////////////////////////////////////////////////////////////////  module N\_Bit\_Counter #(                                                         // Start of the module.        /\* Parameter List of the N\_Bit\_Counter \*/      parameter                COUNTER\_VALUE\_WIDTH = 12,           // The default width is 12 bits.  parameter COUNTER\_MAX\_VALUE = (2\*\*COUNTER\_VALUE\_WIDTH)-1,           // The maximum value that corresponds to (2^width - 1).  parameter                 COUNTER\_INCREMENT = 1             // The counter should increment by 1 every clock cycle.    )(  /\* Port List of the N\_Bit\_Counter \*/  input                COUNTER\_CLOCK ,                  // Counter increments according to the clock.  input                COUNTER\_RESET ,                 // Counter resets to 0 when reset becomes HIGH.  input                COUNTER\_ENABLE ,                 // Counter increments only if enable is HIGH.  output reg [(COUNTER\_VALUE\_WIDTH-1):0] COUNTER\_VALUE = 0                   // The final count value of the counter.  );      /\* Local Parameter List of the N\_Bit\_Counter \*/      localparam ZERO = {(COUNTER\_VALUE\_WIDTH){1'b0}};                            // Local parameter with value 0 having default width of 12 bits.      always @ ( posedge COUNTER\_CLOCK or posedge COUNTER\_RESET )                 // Always statement such that the counter value changes when either          begin                                                                   // reset or clock change from LOW to HIGH.                if ( COUNTER\_RESET )                                                // Check whether reset is HIGH.                  begin                      COUNTER\_VALUE <= ZERO;                                      // Set counter value to 0 if reset is HIGH.                  end                else if ( COUNTER\_ENABLE )                                          // Check if enable is HIGH                  begin                        if ( COUNTER\_VALUE >= COUNTER\_MAX\_VALUE )                   // Check if counter value has surpassed maximum value.                          begin                              COUNTER\_VALUE <= ZERO;                              // Set counter value to 0 if counter value has surpassed maximum value.                          end                        else                          begin                              COUNTER\_VALUE <= COUNTER\_VALUE + COUNTER\_INCREMENT; // If none of the above conditions satisfy, then increment the counter value.                          end                  end          end  endmodule                                                                       // End of the module. |

|  |
| --- |
| **SECTION 2:** N-bit comparator module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : N\_Bit\_Comparator  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 26/02/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used compare the value of two numbers. The output of the      module is HIGH when the first number is greater than or equal to the second      number. The output of the module is LOW when the first number is smaller      than the second number. The time period for which the output of the      comparator is HIGH determines the ON\_PERIOD of the PWM Signal of the Servo      Motor.    \*/  ////////////////////////////////////////////////////////////////////////////////  module N\_Bit\_Comparator #(                                                     // Start of the module.  /\* Parameter List of the N\_Bit\_Comparator \*/  parameter                      NUMBER\_WIDTH = 12                           // The default width is 12 bits to match value from the counter.    )(      /\* Port List of the N\_Bit\_Comparator \*/  input   [(NUMBER\_WIDTH-1):0] FIRST\_NUMBER ,                         // The first number of the comparator.  input [(NUMBER\_WIDTH-1):0] SECOND\_NUMBER ,                         // The second number of the comparator.  output reg                   FN\_GREATER\_THAN\_SN                          // The output of the comparator.  );      always @ ( FIRST\_NUMBER , SECOND\_NUMBER )          begin                if ( FIRST\_NUMBER >= SECOND\_NUMBER )                                // LOW if second number is greater than first number, else HIGH.                  begin                      FN\_GREATER\_THAN\_SN <= 1'b1;                  end                else                  begin                      FN\_GREATER\_THAN\_SN <= 1'b0;                  end          end  endmodule                                                                       // End of the module. |

|  |
| --- |
| **SECTION 3:** Servo Motor Controller Module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Servo\_Motor\_Controller  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to control a single SG90 Servo Motor. The servo motor      requires an input PWM signal to drive to the required angular position. The      PWM signal for the SG90 motor should have a clock period of 20 ms. The      desired angular position can be obtained by controlling the ON period (duty      cycle) of the PWM signal. The duty cycle is can be manually varied by      changing the 8 bit inputs connected to 8 different slide switches. The value      of the duty cycle is latched to the output by using a load signal connected      to a push button. The servo motor can be initialised to a default position      by pressing the reset push button. The default frequency of the clock is      128 kHz.    \*/  ////////////////////////////////////////////////////////////////////////////////  module Servo\_Motor\_Controller # (                                               // Start of the module.        /\* Parameter List of the Servo\_Motor\_Controller \*/      parameter                       DUTY\_CYCLE\_WIDTH = 8 ,                    // The default width is 8-bits ( 0 to 255 ) corresponding to ( -90 to +90 ).      parameter                       PWM\_SIGNAL\_WIDTH = 12 ,                    // The required width is 12 bits to enclose maximum value of 2560 (20ms) with a clock of 128 kHz frequency.      parameter                       MAX\_CLOCK\_CYCLES = 2560                     // With a clock of 128 kHz frequency, 2560 clock cycles are required to generate 20 ms.    )(      /\* Port List of the Servo\_Motor\_Controller \*/      input                           CLOCK\_SIGNAL ,                         // Connects to the hardware clock to generate the required PWM signal.      input                           CLOCK\_ENABLE ,                         // Servo motor turns only if enable is HIGH.      input                           SERVO\_RESET ,                         // Servo resets to default when reset becomes HIGH.      input                           LOAD\_SIGNAL ,                         // Latch the duty cycle of the PWM signal.      input [(DUTY\_CYCLE\_WIDTH-1):0] DUTY\_CYCLE\_CONTROL,                         // Controls the duty cycle of the PWM signal, thus controlling the angle.      output                          PWM\_SIGNAL                                  // The output of the module that is the input to the Servo motor.  );      /\* Register List of the Servo\_Motor\_Controller \*/      wire [(PWM\_SIGNAL\_WIDTH-1):0] CURRENT\_CLOCK\_CYCLES;                         // This holds the number of clock cycles generated by the clock with 128 kHz frequency.      reg [(PWM\_SIGNAL\_WIDTH-1):0] REQUIRED\_ON\_CYCLES ;                         // The ON period (duty cycle) is determined by the 8-bit input.      wire DIVIDED\_CLOCK\_SIGNAL;                                                  // Output of the frequency divider circuit that converts current clock frequency to the default value.        /\* Local Parameter List of the Servo\_Motor\_Controller \*/      localparam DEFAULT\_ON\_CYCLES = {(PWM\_SIGNAL\_WIDTH){12'd192}};               // To set the servo to angle 0, 1.5ms is required which corresponds to 192 clock cycles.      localparam OFFSET\_ON\_CYCLES = {(PWM\_SIGNAL\_WIDTH){12'd64}} ;               // The on cycles vary from 64(0.5ms) to 320(2.5ms) to turn angles from -90 to 90.        /\* Check status of load signal and rest and then take necessary actions \*/      always @ ( posedge LOAD\_SIGNAL or posedge SERVO\_RESET )                     // Always statement is used to set the required on cycle depending on the value of the 8-bit input.          begin                                                                   // The sensitivity list contains the latch signal and reset signal.                if ( SERVO\_RESET )                                                  // Check if reset is HIGH.                  begin                      REQUIRED\_ON\_CYCLES <= DEFAULT\_ON\_CYCLES;                    // Set the servo to its default position if the reset if HIGH.                  end                else if ( CLOCK\_ENABLE )                                            // Check if clock enable is HIGH.                  begin                      REQUIRED\_ON\_CYCLES <= DUTY\_CYCLE\_CONTROL + OFFSET\_ON\_CYCLES;//  Assigning value to the on cycle based upon the duty cycle and the offset.                  end          end        /\* Instantiating N Bit Counter to generate the required clock cycles for the servo motor. \*/      N\_Bit\_Counter # (          . COUNTER\_VALUE\_WIDTH ( PWM\_SIGNAL\_WIDTH ),                         // Setting the parameter values.          . COUNTER\_MAX\_VALUE ( MAX\_CLOCK\_CYCLES )        ) Cycle\_Generator (          . COUNTER\_CLOCK ( CLOCK\_SIGNAL         ),                         // Setting the connections to their corresponding ports.          . COUNTER\_RESET ( SERVO\_RESET ),          . COUNTER\_ENABLE ( CLOCK\_ENABLE ),          . COUNTER\_VALUE      ( CURRENT\_CLOCK\_CYCLES )      );        /\* Instantiating N Bit Comparator to generate the HIGH cycles of the PWM signal. \*/      N\_Bit\_Comparator #    (          . NUMBER\_WIDTH ( PWM\_SIGNAL\_WIDTH )                          // Setting the parameter values.        ) On\_Period\_Generator (          . FIRST\_NUMBER ( REQUIRED\_ON\_CYCLES ),                          // Setting the connections to their corresponding ports.          . SECOND\_NUMBER ( CURRENT\_CLOCK\_CYCLES ),          . FN\_GREATER\_THAN\_SN ( PWM\_SIGNAL )      );    endmodule                                                                       // End of the module. |

|  |
| --- |
| **SECTION 4:** Frequency divider circuit module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Frequency\_Divider  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to reduce the incoming clock rate to a fixed value.      The fixed value is 128 kHz and the incoming clock rate can vary from 128 kHz      to 100 MHz.    \*/  ////////////////////////////////////////////////////////////////////////////////  module Frequency\_Divider #(                                                     // Start of the module.  /\* Parameter List of the Frequency\_Divider \*/  parameter   INCOMING\_CLOCK\_FREQUENCY = 50000000,                           // The minimum operable frequency is 128 kHz and the maximum operable frequency is 100 MHz.      parameter   FIXED\_CLOCK\_FREQUENCY = 128000,                             // All the servo default parameters are calculated at this frequency.      parameter   TOGGLE = (INCOMING\_CLOCK\_FREQUENCY / FIXED\_CLOCK\_FREQUENCY)/2, // The toggle value is used to change the state of the output. The maximum value is 781.      parameter   MAXIMUM\_WIDTH = 10 ,                             // 10 bits to encompass all the possible values of toggle.      parameter INCREMENT                = 1 // The value by which the clock counter should increase.    )(      /\* Port List of the Servo\_Motor\_Controller \*/  input   FD\_CLOCK\_IN,                                                    // The incoming clock is connected to this port.  input FD\_RESET,                                                       // The reset pin is connected to this port.  output reg  FD\_CLOCK\_OUT                                                    // This provides the fixed clock rate depending upon the divider value.  );      wire [(MAXIMUM\_WIDTH-1):0] CURRENT\_COUNTER ;                                // This maximum value of the counter will be equal to the toggle which needs 10 bits.        localparam LOW = 0;                                                        // 1 bit Local parameter with value 0      localparam HIGH = 1;                                                        // 1 bit Local parameter with value 1      /\* Instantiating the N Bit counter to count values up to Toggle \*/      N\_Bit\_Counter #          (          . COUNTER\_VALUE\_WIDTH ( MAXIMUM\_WIDTH ),          . COUNTER\_MAX\_VALUE  ( TOGGLE       ),          . COUNTER\_INCREMENT  ( INCREMENT    )        ) FD\_Toggler             (          . COUNTER\_CLOCK      ( FD\_CLOCK\_IN  ),          . COUNTER\_RESET      ( FD\_RESET     ),          . COUNTER\_ENABLE ( HIGH            ),          . COUNTER\_VALUE  ( CURRENT\_COUNTER )      );        always @ ( posedge FD\_CLOCK\_IN or posedge FD\_RESET )                        // Always statement such that the counter value changes when either          begin                                                                   // reset or clock change from LOW to HIGH.                    if ( FD\_RESET )                                                 // Check whether reset is HIGH.                      begin                          FD\_CLOCK\_OUT <= LOW;                                    // Reset the initialize the clock.                      end                    else if ( CURRENT\_COUNTER == TOGGLE )                           // Check if the toggle value has been reached.                      begin                          FD\_CLOCK\_OUT <= ~ FD\_CLOCK\_OUT;                         // Switch (LOW to HIGH or HIGH to LOW ) the output clock when the current value has reached the toggle value.                      end          end    endmodule                                                                       // End of the module |

|  |
| --- |
| **SECTION 5:** Test single servo motor module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Verilog Module Name : Test\_Single\_Servo\_Motor    Code Author : Shrajan Bhandary  Date Created : 03/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to control a single SG90 Servo Motor. The servo motor      requires an input PWM signal to drive to the required angular position. The      PWM signal for the SG90 motor should have a clock period of 20 ms. The      desired angular position can be obtained by controlling the ON period (duty      cycle) of the PWM signal. The duty cycle is can be manually varied by      changing the 8 bit inputs connected to 8 different slide switches. The value      of the duty cycle is latched to the output by using a load signal connected      to a push button. The servo motor can be initialised to a default position      by pressing the reset push button.    \*/  ////////////////////////////////////////////////////////////////////////////////  module Test\_Single\_Servo\_Motor #(                                               // Start of the module.  /\* Parameter List of the Test\_Single\_Servo\_Motor \*/  parameter   CLOCK\_FREQUENCY = 50000000,                                    // The minimum operable frequency is 128 kHz and the maximum operable frequency is 100 MHz. ( In Hz).      parameter   DUTY\_CYCLE\_WIDTH = 8    ,                                    // The number of bits for the duty cycle control.      parameter   INVERTED\_INPUT = 1                                      // Parameter to select between active LOW ( Invert = 1 ) inputs and active HIGH ( Invert = 0 )inputs.                                                                                  // Inverted input should be 0 and 1 for test bench verification and hardware verification respectively.  )(      /\* Port List of the Test\_Single\_Servo\_Motor \*/  input                       CLOCK ,                         // The incoming clock is connected to this port.  input                     RESET ,                         // The reset pin is connected to this port.  input                           LOAD\_SIGNAL ,                         // Latch the duty cycle of the PWM signal.      input [(DUTY\_CYCLE\_WIDTH-1):0] DUTY\_CYCLE\_CONTROL,                         // Controls the duty cycle of the PWM signal, thus controlling the angle.  //  output                          DIVIDED\_CLOCK    ,                         // This signal is used only in test bench verification to check the divided clock.                                                                                  // Comment lines 40 and 49 to perform hardware verification.      output                          PWM\_SIGNAL                                  // The output of the module that is the input to the Servo motor.  );      wire CLOCK\_CONNECTOR;                                                       // Net to connect output of the frequency divider to the servo motor controller.      reg INVERT\_RESET;                                                           // Inverts the input reset signal for LOW active low inputs.      reg INVERT\_LOAD\_SIGNAL;                                                     // Inverts the input load signal for LOW active low inputs.      localparam HIGH\_VALUE = 1;                                                  // Local parameter with value 1.    //  assign DIVIDED\_CLOCK = CLOCK\_CONNECTOR;                                     // This signal is used only in test bench verification to check the divided clock. Comment this for hardware verification.        /\* To check whether the inputs are active LOW or not and then take necessary actions \*/      always @ ( RESET , LOAD\_SIGNAL ) begin                                      // Always statement that changes with respect to reset and load signals.            if ( INVERTED\_INPUT )                                                   // Check if the inputs are active LOW or active HIGH.              begin                                                               // If yes, invert the inputs.                  INVERT\_RESET <= ~ RESET;                  INVERT\_LOAD\_SIGNAL <= ~ LOAD\_SIGNAL;              end            else              begin                                                               // If no, assign the same inputs.                  INVERT\_RESET <= RESET;                  INVERT\_LOAD\_SIGNAL <= LOAD\_SIGNAL;              end      end        /\* Instantiating the frequency divider to convert incoming clock frequency to fixed frequency. \*/      Frequency\_Divider # (          . INCOMING\_CLOCK\_FREQUENCY  ( CLOCK\_FREQUENCY )        ) Clock\_Enable\_Generator (          . FD\_CLOCK\_IN               ( CLOCK          ),          . FD\_RESET                  ( INVERT\_RESET   ),          . FD\_CLOCK\_OUT              ( CLOCK\_CONNECTOR )        );      // Instantiating N number of servomotors with separate PWM signal and load signal.      Servo\_Motor\_Controller Servo\_Motor\_Driver\_Block (          . CLOCK\_SIGNAL       ( CLOCK\_CONNECTOR    ),          . CLOCK\_ENABLE          ( HIGH\_VALUE     ),          . SERVO\_RESET           ( INVERT\_RESET       ),          . LOAD\_SIGNAL           ( INVERT\_LOAD\_SIGNAL ),          . DUTY\_CYCLE\_CONTROL    ( DUTY\_CYCLE\_CONTROL ),          . PWM\_SIGNAL            ( PWM\_SIGNAL         )      );  endmodule |
| **SECTION 6:** N-channel 8-bit servo controller |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : N\_Channel\_Servo\_Controller  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to instantiate a parametrized number of servo controllers      to allow the IP core to have N independent channels. This module has a clock      input, an N-bit PWM output, a single 8-bit input to set the duty cycle, a      1-bit load signal to latch the duty cycle value, and an address input to      control which servo is being updated.    \*/  ////////////////////////////////////////////////////////////////////////////////  /\* ceil(log2(N)) Preprocessor Macro \*/  `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)  module N\_Channel\_Servo\_Controller #(                                            // Start of the module.  /\* Parameter List of the N\_Channel\_Servo\_Controller \*/      parameter   NO\_OF\_CHANNEL    = 4                    ,                       // The number of servos to be controlled corresponding to number of PWM signals.      parameter   ADDRESS\_WIDTH    = `clog2(NO\_OF\_CHANNEL),                       // Determine the required width to of the servo selector.  parameter   CLOCK\_FREQUENCY = 50000000             ,                       // The minimum operable frequency is 128 kHz and the maximum operable frequency is 100 MHz. ( In Hz).      parameter   DUTY\_CYCLE\_WIDTH = 8                   ,                       // The number of bits for the duty cycle control.      parameter   INVERTED\_INPUT = 1                                     // Parameter to select between active LOW ( Invert = 1 ) inputs and active HIGH ( Invert = 0 )inputs.                                                                                  // Inverted input should be 0 and 1 for test bench verification and hardware verification respectively.  )(      /\* Port List of the N\_Channel\_Servo\_Controller\*/  input                       CLOCK ,                         // The incoming clock is connected to this port.  input                     RESET ,                         // The reset pin is connected to this port.  input                           LOAD\_SIGNAL ,                         // Latch the duty cycle of the PWM signal.      input [(DUTY\_CYCLE\_WIDTH-1):0] DUTY\_CYCLE\_CONTROL,                         // Controls the duty cycle of the PWM signal, thus controlling the angle.      input [(ADDRESS\_WIDTH-1):0]    SERVO\_SELECTOR   ,                         // Determines the servo that has to be actuated.      output [(NO\_OF\_CHANNEL-1):0]    PWM\_SIGNALS                                 // The output of the module that is the input to the Servo motor.  );      wire CLOCK\_CONNECTOR;                                                       // Net to connect output of the frequency divider to the servo motor controller.      reg INVERT\_RESET;                                                           // Inverts the input reset signal for LOW active low inputs.      reg INVERT\_LOAD\_SIGNAL;                                                     // Inverts the input load signal for LOW active low inputs.      localparam HIGH\_VALUE = 1;                                                  // Local parameter with value 1.        reg [(NO\_OF\_CHANNEL-1):0]INDIVIDUAL\_LOAD ;                                  // Each channel has its individual load register.        /\* To check whether the inputs are active LOW or not and then take necessary actions \*/      always @ ( RESET , LOAD\_SIGNAL ) begin                                      // Always statement that changes with respect to reset and load signals.            if ( INVERTED\_INPUT )                                                   // Check if the inputs are active LOW or active HIGH.              begin                                                               // If yes, invert the inputs.                  INVERT\_RESET <= ~ RESET;                  INVERT\_LOAD\_SIGNAL <= ~ LOAD\_SIGNAL;              end            else              begin                                                               // If no, assign the same inputs.                  INVERT\_RESET <= RESET;                  INVERT\_LOAD\_SIGNAL <= LOAD\_SIGNAL;              end      end      /\* Instantiating the frequency divider to convert incoming clock frequency to fixed frequency. \*/      Frequency\_Divider # (          . INCOMING\_CLOCK\_FREQUENCY  ( CLOCK\_FREQUENCY )        ) Clock\_Enable\_Generator (          . FD\_CLOCK\_IN               ( CLOCK          ),          . FD\_RESET                  ( INVERT\_RESET   ),          . FD\_CLOCK\_OUT              ( CLOCK\_CONNECTOR )        );        genvar COUNT;                                                               // Creating a general variable to implement a for loop.        generate      for ( COUNT = 0 ; COUNT < NO\_OF\_CHANNEL ; COUNT = (COUNT + 1) )             // Creating a for loop to instantiate N number of servomotors.          begin : Servo\_Motor\_Driver\_Block\_loop                localparam NUMBER = COUNT ;                                         // A local parameter to hold the number of possible servo combinations.                /\* To check the value of the load signal and then take necessary actions. \*/              always @ ( INVERT\_LOAD\_SIGNAL )                  begin                        if ( INVERT\_LOAD\_SIGNAL )                                   // Check whether the load signal button has been pressed.                          begin                                if ( SERVO\_SELECTOR == NUMBER )                     // Check if the input selector is same as that of the current servomotor ID.                                  begin                                      INDIVIDUAL\_LOAD[COUNT] <= 1'b1;             // Set the load of that particular servomotor to HIGH. (This signal should not remain always HIGH).                                  end                              else                                  begin                                      INDIVIDUAL\_LOAD[COUNT] <= 1'b0;             // If the input selector does not match to the current servomotor ID, then set the load of the                                  end                                             // that particular servomotor to HIGH.                          end                        else                          begin                              INDIVIDUAL\_LOAD[COUNT] <= 1'b0;                     // Once the load signal button is released set all the load signals to LOW.                          end                  end                Servo\_Motor\_Controller Servo\_Motor\_Driver\_Block (                   // Instantiating N number of servomotors with separate PWM signal and load signal.                    . CLOCK\_SIGNAL       ( CLOCK\_CONNECTOR        ),                  . CLOCK\_ENABLE          ( HIGH\_VALUE         ),                  . SERVO\_RESET           ( INVERT\_RESET           ),                  . LOAD\_SIGNAL           ( INDIVIDUAL\_LOAD[COUNT] ),                  . DUTY\_CYCLE\_CONTROL    ( DUTY\_CYCLE\_CONTROL     ),                  . PWM\_SIGNAL            ( PWM\_SIGNALS [COUNT]    )              );          end      endgenerate    endmodule                                                                       // End of the module. |

**APPENDIX – B**

**Appendix B consists of all Verilog test bench files used in the project.**

|  |
| --- |
| **SECTION 1:** Servo Motor Controller test bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Servo\_Motor\_Controller\_tb  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to control a single SG90 Servo Motor. The servo motor      requires an input PWM signal to drive to the required angular position. The      PWM signal for the SG90 motor should have a clock period of 20 ms. The      desired angular position can be obtained by controlling the ON period (duty      cycle) of the PWM signal. The duty cycle is can be manually varied by      changing the 8 bit inputs connected to 8 different slide switches. The value      of the duty cycle is latched to the output by using a load signal connected      to a push button. The servo motor can be initialised to a default position      by pressing the reset push button.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns /100 ps  module Servo\_Motor\_Controller\_tb;  /\* Parameter List of the Servo\_Motor\_Controller\_tb \*/  localparam NUM\_CYCLES = 50000;     // Simulate this many clock cycles. Maximum value is 1 billion.  localparam CLOCK\_FREQ = 128000;          // Current Clock frequency (in Hz).  localparam RST\_CYCLES = 2;        // Number of cycles of reset at beginning.  localparam REPEAT\_DUTY\_CYCLES = 25600;                                          // Repeat values for different values of the input.  localparam WAIT\_PERIOD = 2;                                                     // Creating a small delay.  /\* Test Bench Generated Signals of the Servo\_Motor\_Controller\_tb \*/  reg TB\_CLOCK ;                                                                 // Connects to the clock of the servo motor controller.  reg TB\_ENABLE ;                                                                 // Connects to the enable of the servo motor controller.  reg TB\_RESET ;                                                                 // Connects to the reset of the servo motor controller.  reg TB\_LOAD\_SIGNAL ;                                                            // Connects to the load signal of the servo motor controller.  reg [7:0] TB\_DUTY\_CYCLE\_CONTROL ;                                               // Connects to the 8-bit duty cycle control of the servo motor controller.  /\* Device Under Test (DUT) Output Signals of the Servo\_Motor\_Controller\_tb \*/  wire TB\_PWM\_SIGNAL;  /\* Device Under Test (DUT) of the Servo\_Motor\_Controller\_tb \*/  Servo\_Motor\_Controller Servo\_Motor\_Controller\_DUT (                             // Setting the connections to their corresponding ports.  .CLOCK\_SIGNAL ( TB\_CLOCK ),  .CLOCK\_ENABLE ( TB\_ENABLE ),  .SERVO\_RESET ( TB\_RESET ),  .LOAD\_SIGNAL ( TB\_LOAD\_SIGNAL ),  .DUTY\_CYCLE\_CONTROL ( TB\_DUTY\_CYCLE\_CONTROL ),  .PWM\_SIGNAL ( TB\_PWM\_SIGNAL )  );      /\* Reset the entire control system so that the servo initializes to the default value. \*/      initial begin          TB\_RESET = 1'b1;                   // Set the reset signal to HIGH.          TB\_LOAD\_SIGNAL = 1'b1;                                                  // Set the load signal to HIGH.          repeat( RST\_CYCLES ) @ ( posedge TB\_CLOCK );              // Wait for a couple of clocks.          TB\_RESET = 1'b0;                   // Set the reset signal to LOW.          TB\_LOAD\_SIGNAL = 1'b0;                                                  // Set the load signal to LOW.      end      /\* Clock generator and simulation time limit. \*/      initial begin          TB\_CLOCK = 1'b0;                                                        // Initialise the clock to zero.      end      real HALF\_CLOCK\_PERIOD = (1000000000.0 / $itor(CLOCK\_FREQ)) / 2.0; // Calculating the time delay for each half of the clock cycle and storing it in a variable.      integer half\_cycles = 0;                                                    // Variable to count the elapsed number of half cycles.      always begin            TB\_ENABLE = 1'b1;                                                       // Set the enable signal to HIGH.            /\* Duty Cycle Control = 0 \*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd0;                                   // Assign a value of 0 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            /\* Duty Cycle Control = 128 \*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd128;                                 // Assign a value of 128 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            /\* Duty Cycle Control = 255 \*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd255;                                 // Assign a value of 255 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            $stop;                                               // Break the simulation        end  endmodule |

|  |
| --- |
| **SECTION 2:** Frequency divider test bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Frequency\_Divider\_tb  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to reduce the incoming clock rate to a fixed value.      The fixed value is 128 kHz and the incoming clock rate can vary from 128 kHz      to 100 MHz.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns /100 ps  module Frequency\_Divider\_tb;  /\* Parameter List of the Frequency\_Divider\_tb \*/  localparam NUM\_CYCLES = 50000000;      // Simulate this many clock cycles. Maximum value is 1 billion.  localparam CLOCK\_FREQ = 50000000;        // Current Clock frequency (in Hz).  localparam RST\_CYCLES = 2;        // Number of cycles of reset at beginning.  /\* Test Bench Generated Signals of the Frequency\_Divider\_tb \*/  reg TB\_CLOCK\_IN ;                                                            // Connects to the clock of the frequency divider circuit.  reg TB\_RESET ;                                                            // Connects to the reset of the frequency divider circuit.    /\* Device Under Test (DUT) Output Signals of the Frequency\_Divider\_tb \*/  wire TB\_CLOCK\_OUT;  /\* Device Under Test (DUT) of the Frequency\_Divider\_tb \*/  Frequency\_Divider Frequency\_Divider\_DUT (                                       // Setting the connections to their corresponding ports.  .FD\_CLOCK\_IN          ( TB\_CLOCK\_IN ),  .FD\_RESET             ( TB\_RESET ),  .FD\_CLOCK\_OUT     ( TB\_CLOCK\_OUT )  );      /\* Reset the entire control system so that the servo initializes to the default value. \*/      initial begin          TB\_RESET = 1'b1;                   // Set the reset signal to HIGH.          repeat( RST\_CYCLES ) @ ( posedge TB\_CLOCK\_IN );           // Wait for a couple of clocks.          TB\_RESET = 1'b0;                   // Set the reset signal to LOW.      end      /\* Clock generator and simulation time limit. \*/      initial begin          TB\_CLOCK\_IN = 1'b0;                                                     // Initialise the clock to zero.      end        real HALF\_CLOCK\_PERIOD = (1000000000.0 / $itor(CLOCK\_FREQ)) / 2.0; // Calculating the time delay for each half of the clock cycle and storing it in a variable.      integer half\_cycles = 0;                                                    // Variable to count the elapsed number of half cycles.        always begin          /\* Generating individual half cycles of clock. \*/          #(HALF\_CLOCK\_PERIOD);                                          // Delay for half a clock period.          TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                               // Toggle the clock signal.          half\_cycles = half\_cycles + 1;                                          // Increment the count of number of half cycles.              /\* Check if we have simulated enough half clock cycles. \*/          if (half\_cycles == (2\*NUM\_CYCLES))          begin              half\_cycles = 0;                                                  // Reset half cycles, so if we resume running with "run -all", we perform another chunk.              $stop;                                          // Break the simulation          end  end  endmodule |

|  |
| --- |
| **SECTION 3:** Test single servomotor test bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Test\_Single\_Servo\_Motor\_tb  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to control a single SG90 Servo Motor. The servo motor      requires an input PWM signal to drive to the required angular position. The      PWM signal for the SG90 motor should have a clock period of 20 ms. The      desired angular position can be obtained by controlling the ON period (duty      cycle) of the PWM signal. The duty cycle is can be manually varied by      changing the 8 bit inputs connected to 8 different slide switches. The value      of the duty cycle is latched to the output by using a load signal connected      to a push button. The servo motor can be initialised to a default position      by pressing the reset push button.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns /100 ps  module Test\_Single\_Servo\_Motor\_tb;  /\* Parameter List of the Test\_Single\_Servo\_Motor\_tb \*/  localparam CLOCK\_FREQ = 50000000;        // Current Clock frequency (in Hz).  localparam RST\_CYCLES = 1;        // Number of cycles of reset at beginning.  localparam REPEAT\_DUTY\_CYCLES = 2000000;                                        // Repeat values for different values of the input.  localparam WAIT\_PERIOD = 2;                                                     // Creating a repeat block to make the reset the load signal.  /\* Test Bench Generated Signals of the Test\_Single\_Servo\_Motor\_tb \*/  reg TB\_CLOCK\_IN ;                                                              // Connects to the clock of the servo motor controller.  reg TB\_RESET ;                                                                 // Connects to the reset of the servo motor controller.  reg TB\_LOAD\_SIGNAL ;                                                            // Connects to the load signal of the servo motor controller.  reg [7:0] TB\_DUTY\_CYCLE\_CONTROL ;                                               // Connects to the 8-bit duty cycle control of the servo motor controller.  /\* Device Under Test (DUT) Output Signals of the Test\_Single\_Servo\_Motor\_tb \*/  wire TB\_PWM\_SIGNAL;                                                             // Connects to the PWM signal of the servomotor.  wire TB\_DIVIDED\_CLOCK;                                                          // This signal is used only in test bench verification.  /\* Device Under Test (DUT) of the Test\_Single\_Servo\_Motor\_tb \*/  Test\_Single\_Servo\_Motor Test\_Single\_Servo\_Motor\_DUT (                         // Setting the connections to their corresponding ports.  .CLOCK        ( TB\_CLOCK\_IN ),  .RESET        ( TB\_RESET ),  .LOAD\_SIGNAL ( TB\_LOAD\_SIGNAL ),  .DUTY\_CYCLE\_CONTROL ( TB\_DUTY\_CYCLE\_CONTROL ),  .DIVIDED\_CLOCK       ( TB\_DIVIDED\_CLOCK      ),                              // This signal is used only in test bench verification.  .PWM\_SIGNAL ( TB\_PWM\_SIGNAL )  );      /\* Reset the entire control system so that the servo initializes to the default value. \*/      initial begin          TB\_RESET = 1'b1;                   // Set the reset signal to HIGH.          TB\_LOAD\_SIGNAL = 1'b1;                                                  // Set the load signal to HIGH.          repeat( RST\_CYCLES ) @ ( posedge TB\_CLOCK\_IN );              // Wait for a couple of clocks.          TB\_RESET = 1'b0;                   // Set the reset signal to LOW.          TB\_LOAD\_SIGNAL = 1'b0;                                                  // Set the load signal to LOW.      end      /\* Clock generator and simulation time limit. \*/      initial begin          TB\_CLOCK\_IN = 1'b0;                                                     // Initialise the clock to zero.      end      real HALF\_CLOCK\_PERIOD = (1000000000.0 / $itor(CLOCK\_FREQ)) / 2.0; // Calculating the time delay for each half of the clock cycle and storing it in a variable.      integer half\_cycles = 0;                                                    // Variable to count the elapsed number of half cycles.      always begin            /\* Duty Cycle Control = 0 \*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd0;                                   // Assign a value of 0 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                          // Toggle the clock signal.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                          // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            ////////////////////////////////////////////////////////////////////////            /\* Duty Cycle Control = 128 \*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd128;                                 // Assign a value of 0 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                          // Toggle the clock signal.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                          // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            ////////////////////////////////////////////////////////////////////////            /\* Duty Cycle Control = 255 \*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd255;                                 // Assign a value of 0 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                          // Toggle the clock signal.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK\_IN = ~ TB\_CLOCK\_IN;                          // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end              $stop;                                               // Break the simulation        end  endmodule |

|  |
| --- |
| **SECTION 4:** N-channel 8-bit servo controller test bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : N\_Channel\_Servo\_Controller\_tb  Target Device       : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location            : University of Leeds  Module              : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:      The module is used to instantiate a parametrized number of servo controllers      to allow the IP core to have N independent channels. This module has a clock      input, an N-bit PWM output, a single 8-bit input to set the duty cycle, a      1-bit load signal to latch the duty cycle value, and an address input to      control which servo is being updated.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns /100 ps  module N\_Channel\_Servo\_Controller\_tb;  /\* ceil(log2(N)) Preprocessor Macro \*/  `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)  /\* Parameter List of the N\_Channel\_Servo\_Controller\_tb \*/  localparam CLOCK\_FREQ        = 50000000 ;       // Current Clock frequency (in Hz).  localparam RST\_CYCLES        = 1        ;        // Number of cycles of reset at beginning.  localparam REPEAT\_DUTY\_CYCLES = 2000000 ;                          // Repeat values for different values of the input.  localparam WAIT\_PERIOD = 2        ;                          // Creating a repeat block to make the reset the load signal.  localparam NO\_OF\_CHANNEL = 4                    ;                          // The number of servos to be controlled corresponding to number of PWM signals.  localparam ADDRESS\_WIDTH = `clog2(NO\_OF\_CHANNEL);                          // Determine the required width to of the servo selector.  /\* Test Bench Generated Signals of the N\_Channel\_Servo\_Controller\_tb \*/  reg TB\_CLOCK                                   ;                               // Connects to the clock of the servo motor controller.  reg TB\_RESET                                    ;                               // Connects to the reset of the servo motor controller.  reg TB\_LOAD\_SIGNAL                              ;                               // Connects to the load signal of the servo motor controller.  reg [7:0] TB\_DUTY\_CYCLE\_CONTROL                 ;                               // Connects to the 8-bit duty cycle control of the servo motor controller.  reg [(ADDRESS\_WIDTH-1):0] TB\_SERVO\_SELECTOR     ;                               // Connects to the N-bit servo selector of the servo motor controller.  /\* Device Under Test (DUT) Output Signals of the N\_Channel\_Servo\_Controller\_tb \*/  wire [(NO\_OF\_CHANNEL-1):0] TB\_PWM\_SIGNALS;                                      // Connects to the N-bit PWM signal of the servomotors.  /\* Device Under Test (DUT) of the N\_Channel\_Servo\_Controller\_tb \*/  N\_Channel\_Servo\_Controller N\_Channel\_Servo\_Controller\_DUT (                // Setting the connections to their corresponding ports.  .CLOCK        ( TB\_CLOCK ),  .RESET        ( TB\_RESET ),  .LOAD\_SIGNAL ( TB\_LOAD\_SIGNAL ),  .DUTY\_CYCLE\_CONTROL ( TB\_DUTY\_CYCLE\_CONTROL ),  .SERVO\_SELECTOR     ( TB\_SERVO\_SELECTOR     ),  .PWM\_SIGNALS ( TB\_PWM\_SIGNALS )  );      /\* Reset the entire control system so that the servo initializes to the default value. \*/      initial begin          TB\_RESET = 1'b1;                   // Set the reset signal to HIGH.          TB\_LOAD\_SIGNAL = 1'b1;                                                  // Set the load signal to HIGH.          repeat( RST\_CYCLES ) @ ( posedge TB\_CLOCK );              // Wait for a couple of clocks.          TB\_RESET = 1'b0;                   // Set the reset signal to LOW.          TB\_LOAD\_SIGNAL = 1'b0;                                                  // Set the load signal to LOW.      end      /\* Clock generator and simulation time limit. \*/      initial begin          TB\_CLOCK = 1'b0;                                                        // Initialise the clock to zero.      end      real HALF\_CLOCK\_PERIOD = (1000000000.0 / $itor(CLOCK\_FREQ)) / 2.0; // Calculating the time delay for each half of the clock cycle and storing it in a variable.      integer half\_cycles = 0;                                                    // Variable to count the elapsed number of half cycles.      always begin            /\* Duty Cycle Control = 0 and servo selector = 0\*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd0;                                   // Assign a value of 0 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  TB\_SERVO\_SELECTOR = 2'd0;                                       // Select the first servo.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            ////////////////////////////////////////////////////////////////////////            /\* Duty Cycle Control = 128 and servo selector = 1\*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd128;                                 // Assign a value of 128 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  TB\_SERVO\_SELECTOR = 2'd1;                                       // Select the second servo.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            ////////////////////////////////////////////////////////////////////////            /\* Duty Cycle Control = 255 and servo selector = 2\*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd255;                                 // Assign a value of 255 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  TB\_SERVO\_SELECTOR = 2'd2;                                       // Select the third servo.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            ////////////////////////////////////////////////////////////////////////            /\* Duty Cycle Control = 192 and servo selector = 3\*/          repeat ( REPEAT\_DUTY\_CYCLES )                                           // Repeat the loop for some time.              begin                    TB\_DUTY\_CYCLE\_CONTROL = 8'd192;                                 // Assign a value of 192 for the duty cycle control.                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  TB\_SERVO\_SELECTOR = 2'd3;                                       // Select the fourth servo.                  TB\_LOAD\_SIGNAL = 1'b1;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                end            repeat ( WAIT\_PERIOD )                                                  // Repeat the loop for some time.              begin                    /\* Generating individual half cycles of clock \*/                  #(HALF\_CLOCK\_PERIOD);                                  // Delay for half a clock period.                  TB\_CLOCK = ~ TB\_CLOCK;                           // Toggle the clock signal.                  half\_cycles = half\_cycles + 1;                                  // Increment the count of number of half cycles.                    TB\_LOAD\_SIGNAL = 1'b0;                                          // Set the load signal to HIGH so that the output latches on to the required duty cycle.              end            $stop;                                               // Break the simulation      end  endmodule |

**APPENDIX – C**

**Appendix – C contains the hardware verification images.**

|  |
| --- |
| **SECTION 1:** Single servo motor controller. The servomotor is at its default position of 0 degrees since the value has not been loaded. |
| A circuit board  Description generated with very high confidence |

|  |
| --- |
| **SECTION 2:** Single servo motor controller. The servomotor is at -79.46 degrees since the value of input duty cycle control = 15. |
| A circuit board  Description generated with very high confidence |

|  |
| --- |
| **SECTION 3:** Single servo motor controller. The servomotor is at -87.3 degrees since the value of input duty cycle control = 252. |
| A circuit board  Description generated with very high confidence |

|  |
| --- |
| **SECTION 4:** 3 – Channel servo motor controller. All three servomotors are at their default position of 0 degrees due to input servo selector value = 0 and input duty cycle control = 0. |
| A circuit board  Description generated with very high confidence |

|  |
| --- |
| **SECTION 5:** 3 – Channel servo motor controller. First servo on the left is at 10.54 degrees due to input servo selector value = 0 and input duty cycle control = 15. The remaining servomotors are at their default position of 0 degrees. |
| A circuit board  Description generated with high confidence |

|  |
| --- |
| **SECTION 6:** 3 – Channel servo motor controller. First servo on the right is at 10.54 degrees due to input servo selector value = 1 and input duty cycle control = 15. The servomotor at the centre is its default position of 0 degrees. |
| A circuit board  Description generated with very high confidence |

|  |
| --- |
| **SECTION 7:** 3 – Channel servo motor controller. The servomotor at the centre is at 10.54 degrees due to input servo selector value = 2 and input duty cycle control = 15. |
| A circuit board  Description generated with very high confidence |