Education Objective

The educational objective of this laboratory is to investigate the creation of custom IP to interface with the NIOS II on the Avalon Switch Fabric. This lab is two parts, combined with Lab 5. Lab 4 focuses on the creation of the core, and Lab 5 on the software interfacing with the component.

Technical Objective

The technical objective of this laboratory is to design an embedded system for the Nios II processor and DE1-SoC development board that will control a servo motor to sweep between two angles.

The embedded system should contain a custom IP that creates the appropriate Pulse-Width Modulation (PWM) signal to sweep the servo-motor's arm. The system also contains 8 switches for entering the start and stop angles, 2 pushbuttons for "locking" in the angles and 5 seven-segment displays for displaying the start and stop angles.

The system operates as follows:

- The user inputs the start or minimum angle (> 45) in binary using the switches and locks it in with KEY3.
- The user inputs the stop or maximum angle (<135) in binary using the switches and locks it in with KEY2.
- The C program passes the angle information to the servo controller.
- The servo controller outputs a waveform to the servo motor that causes it to continuously sweep from the minimum angle to the maximum angle and back again.
- The angles can be changed at any time and in any order.

Servo Motor Background

The text below is taken from ServoCity.com

Servos are controlled by sending them a pulse with a variable width. The control wire of the servo motor accepts this pulse. The parameters for this pulse are that it has a minimum pulse, a maximum pulse, and a constant period. A sample pulse is shown in Figure 1 below.

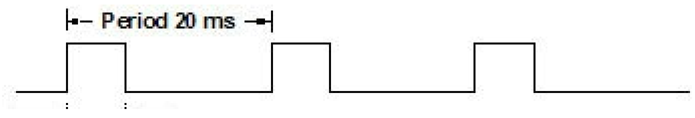


Figure 1: Sample Waveform for Servo Motor

The angle of rotation is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5 ms pulse will make the motor turn to the 90 degree position (neutral position), the minimum width pulse will make the motor turn to the 45 degree position and the maximum width pulse will turn it to the 135 degree position.

When servos are commanded to move they will move to the position and hold that position. If an external force pushes against the servo while the servo is holding a position, the servo will resist moving out of that position. The maximum amount of force the servo can exert is the torque rating of the servo. Servos will not hold their position forever though; the position pulse must be repeated to instruct the servo to stay in position.

When a pulse is sent to a servo that is less than 1.5 ms the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Different brands, and even different servos of the same brand, will have different maximum and minimums.

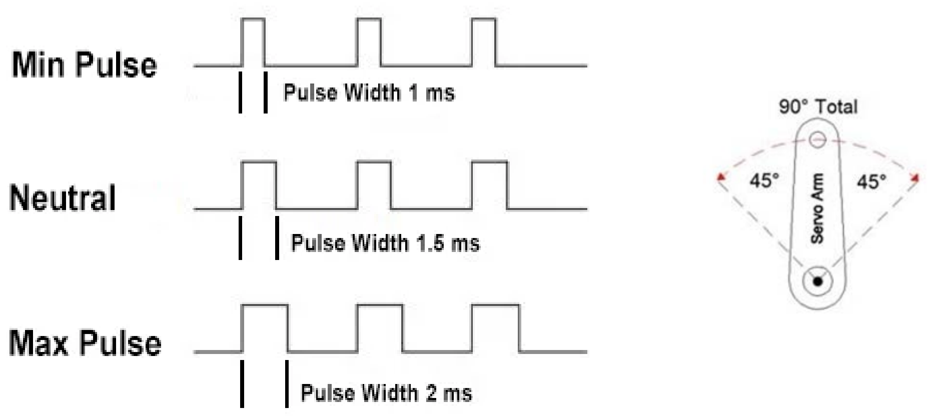
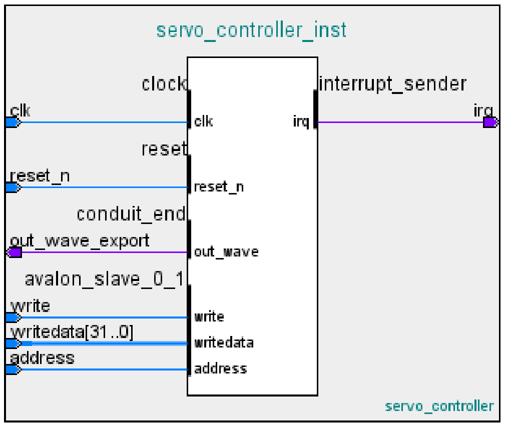


Figure 2: Servo Motor Pulse Width Timing Waveforms

Specifications for the HiTec HS-311 servo motor used in this lab can be found at the web address: http://www.servocity.com/html/hs-311_standard.html.

NOTE To create delays in VHDL, use counters. The clock on the DE1-SoC board is 50 MHz which means it has a period of 20 ns. For a delay of 20 ms, the count would be 20E-3 x 50E6 = 1,000,000. Your C program will read the angles from the switches and will have to determine the correct number of counts to load into the IP for it to create the appropriate pulse width. For example, if the user inputs 90 degrees, the C program will load 1.5E-3 x 50E6 = 75,000 into the IP. Using basic algebra, you can create the equation for a line and use that to determine the counts for any angle.

IP Design



The servo controller IP will have the interfaces illustrated in figure 3. The counduit interface is necessary to send the output PWM signal out on a GPIO pin on the DE1-SoC board. The interrupt_sender interface is necessary for the controller to interrupt the NIOS II processor every time it reaches its min and max angle. The Avalon_slave_interface allows the C program to write the angle information to registers within the controller (see figure 4). Notice that the write signal is active high and the address signal is only 1 bit wide.

Figure3: Servo controller interface diagram

The controller should have 2 internal registers to store the angle information. These registers are 32 bits wide. Since there are only 2 registers, only 1 address bit is needed to choose between the two. The writedata is written to the addressed register when the write signal is active. Although it is customary to clear registers to 0 upon reset, in this case the registers should be initialized to the counts for a 45 to 135 degree sweep (X"0000C350" and X"000186A0", respectively).

Offset	31……………………………………………………………………………0
0	Count for min angle pulsewidth
1	Count for max angle pulsewidth

Figure 4: Register map for servo controller

To create the output wave, a counter is needed to keep track of the 20ms period. While the current count is less than the angle count, the waveform is driven high. For the remainder of the period it is driven low. To enable a smooth sweep of the servo motor, the pulse width should start at the min angle width and gradually increment until it reaches the max angle width. Once it reaches the max angle width, it should gradually decrement until it reaches the min angle width. For simulation, the increment and decrement amount should be 1. For real time operation, you can experiment with that number to change the speed and “smoothness” of the sweep.

There is more than one way to write the code. You can keep signals that indicate the state of the system or use a finite state machine. Both have their advantages and disadvantages, but the advantage of using FSM is that the IRQ output is dependent only on the state. Below is a suggested state transition diagram. Please note that you are not required to use a state machine, nor are you required to follow the suggested state transition diagram. The design is completely up to you.

Notice that writing to one of the internal registers is the transition condition out of the interrupt states and thus the action necessary for clearing the IRQ signal.

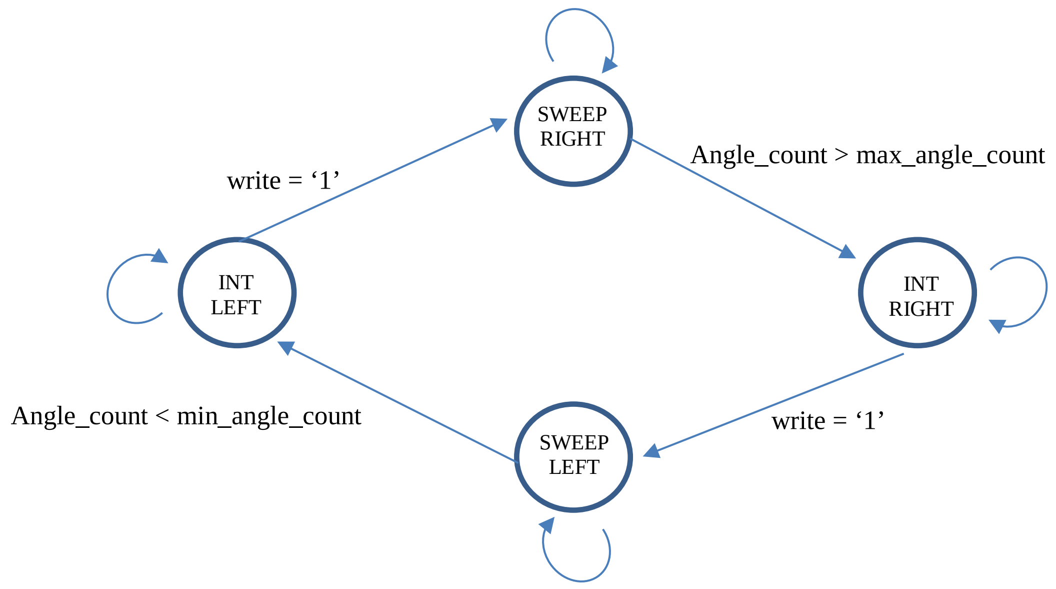


Figure 5: Possible State Transition Diagram for the servo controller

Software Design
The C code for this embedded system if very straight forward. Both the pushbutton and servo controller interrupts need to be registered and the pushbutton interrupts need to be enabled. There should also be an initial write to the servo controller setting its min angle to 45 degrees and its max angle to 135 degrees.

There will be two interrupt service routines. The ISR for the pushbuttons should change the variable for the min or max angle depending on which pushbutton was pushed. The ISR for the servo controller writes to both of the servo controller’s registers. Remember you will be loading the registers with the number of counts necessary to create the pulse width for the specified angle.

The main loop of the program should display the min angle (in decimal) on HEX6 and HEX5 and display the max angle (in decimal) on HEX2-HEX0. If the angles are less than the number of digits allotted, just blank the display or front fill with 0. The displays should be updated every time the min and/or max angle changes. Remember that ISR needs to be in and out quickly. As such, do not update the displays in the ISR.

Demonstration Procedure
Part 1 – Hardware

1. Write the VHDL file to implement the servo motor controller.
2. Write a test bench to test the servo motor controller. Remember to assign constants that make sense for simulation. For example, set the period count to 20 and the angle counts to something like 5 and 15. Verify that your controller operates correctly, increasing its pulsewidth until it reaches the max and then decreasing until it reaches the min. Verify that it raises the IRQ signal when the min and max angle counts are reached and that writing to the registers clears the interrupt. You can add the following statements if you don’t want to figure

out how much time you need to wait for the interrupt to occur.

```
:
Wait for irq = ‘1’;
Address <= …
Writedata <= ….
Write <= ‘1’;
:
```

3. Open Quartus II and create a new project named servo_lab. Add your servo_controller.vhd file to the project.

Demo: Submission or demonstration of lab should include your design and verification VHDL, along with a working simulation. If submitting to the dropbox, include screen shots of important details, along with a written explanation of each capture.