

Lab Assignment 8

Controlling Servos

Introduction

The goal of this lab is designing a digital circuit that produces Pulse Width Modulation (PWM) signals to control the servos of the robotic arm. The duty cycle of these signals will be configurable by the user through two push buttons.. When the robotic arm is connected to the ZedBoard, the push buttons can be used to control the current position of a specific joint of the arm. Since we do not have the ZedBoard available, we will only simulate using Simulink.

The following reading list will help you to understand how PWM Signals work and how servo motors in the robotic arm work

Require Reading: PWM Tutorial on Blackboard

★ <https://blackboard.neu.edu/>

Lab 9.1 Generating PWM Signal

We have determined that the ZedBoard works at a frequency of 50MHz. Create a Simulink design that generates a PWM signal. The signal should have a period of 20ms, the period required by a servo motor. To achieve this, create an HDL counter that rolls back after 20ms (show your calculations for the actual number of cycles). For ease of identification later, name this counter *simple_pwm*. Add a Simulink *relational operator* that compares the counter's output against a constant. The goal is we want the comparator to output a "1" while the counter output value is below 1.5ms, and outputs zero otherwise (show your calculations for the number of cycles for 1.5ms). Use that number of cycles in the constant.

Note: use a relational operator that has two inputs, so that the comparison value can be changed at runtime (i.e., do not use the "compare with constant" block, as it has a fixed value at runtime).

Simulate the design for two periods and validate the correctness using the scope block. Report your findings in the Pre-Lab report. Make sure your scope, in 'limit data points', is unchecked.

Lab 9.2

Assuming a counter that rolls over every 20ms when reaching the value calculated in Pre 9.1 above, answer the following:

- What would be the count value at 0.6 ms? (Note this is the minimal duty cycle of an RC servo PWM signal). Show your calculations. We will refer to this value as *minValue*.
- What would be the count value at 1.5 ms? (Note this is the middle duty cycle of an RC servo PWM signal). Show your calculations. We will refer to this value as *midValue*.
- What would be the count value at 2.4 ms? (Note this is the maximal duty cycle of an RC servo PWM signal). Show your calculations. We will refer to this value as *maxValue*.

Show all your calculations.

You will be simulating the control of the Robotic arm using Simulink. Switches select which servo(s) to operate on, and PBNTU/PBNTD change the PWM duty cycle (position) for the selected servo(s).

PBNTU Increase duty cycle
PBNTD Decrease duty cycle

Table 1 Mapping of servos of the robotic arm to the switches on the ZedBoard.


Table 1. Mapping of servos, connections/pins and switches

Servo	Switch
Base	Switch 0
Bicep	Switch 1
Elbow	Switch 2
Wrist	Switch 3
Gripper	Switch 4

Each servo, enabled by its switch, should move in one direction when the button is in the up position (until reaching the upper limit), and move in the opposite direction when the button is in the down position (until reaching the lower limit).

For example: if switch zero is on (i.e., in the up position), then the base servo should move on each push of the button up/down. If two servos are enabled by their switches, both servos will move at the same time.

Simulink Design Settings:

- For each of your Simulink designs, set the solver to a discrete solver. Open the configuration settings in menu entry *Simulation* → *Model Configuration Parameters*. In the dialog, select:
 - Type: Fixed-step
 - Solver: Discrete (no continuous states)
 - Fixed-step size: 1
- On your scopes, click on the *Settings*  icon, click on the *Logging* tab, and ensure that the *Limit data points to last* option is unchecked. You should remove the scopes when downloading into the FPGA.
- Your outermost inputs and constants should be set with the appropriate **data types** (*Boolean*, *int8*, etc) and a **sampling time** of **1**. The rest of the inputs and outputs in your subsystems can either be left to inherit the data type and sampling time, or set to the appropriate data type if necessary. You can also use the *Data Type Conversion* block if needed.

The following steps will guide you through realizing this design.

Opening your required tools

1. Open the “System Generator” from *All Programs* → *Xilinx Design Tools* → *System Generator 2016.2*.
2. With Matlab open, open the Simulink library browser by clicking on its icon or typing simulink in Matlab command line.

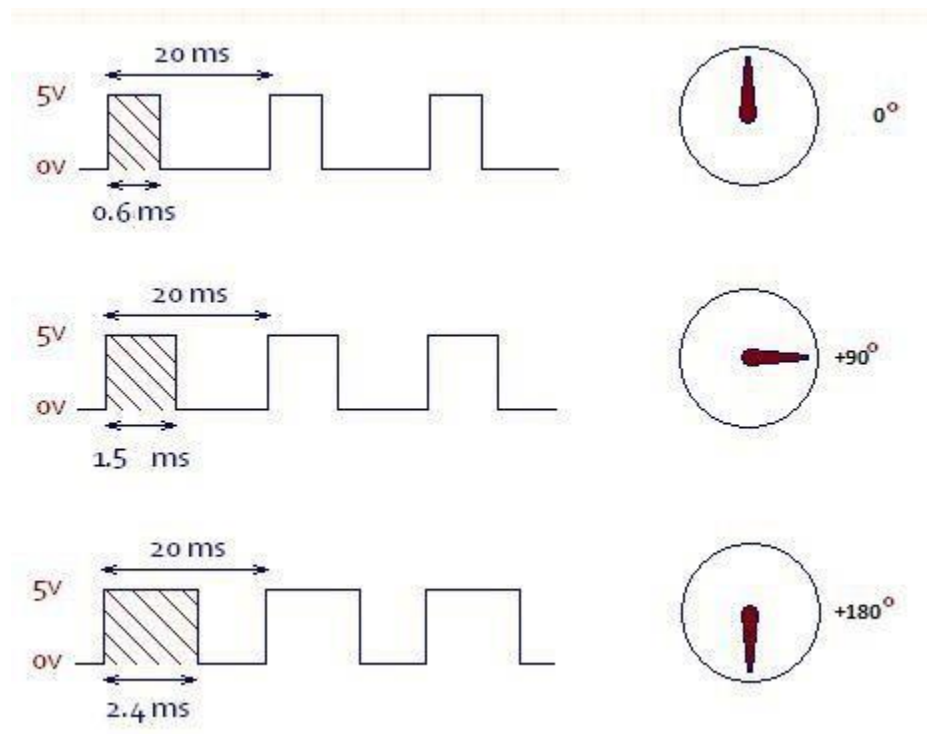
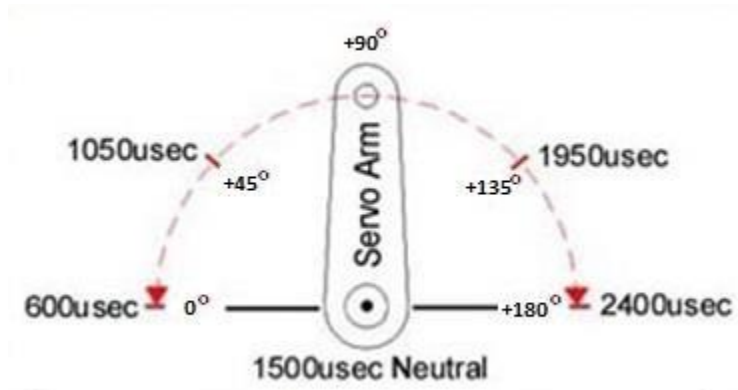


Figure 1. Servo motor operation for different PWM signal duty

9.3 Configurable PWM signal

The Lab 9.1 design produces a PWM signal with a fixed pulse width. Our first improvement on the original design will be making this pulse width configurable, by making it depend on the value stored in a second counter.

1. Make a copy of *simple_pwm.slx* and store it as *configurable_pwm.slx*.
2. Insert a new *HDL Counter* and label it *setCounter* in your design, which will store the current PWM signal width given as a number of simulation cycles.
3. Change the *setCounter* settings to expose an *enable* and a *direction* signal. These signals will allow us to increment and decrement the value stored in the counter.
4. Configure the *setCounter* in such a way that its minimum value is equal to the number of cycles needed to produce a pulse width that takes the servo to a 0° position (cycles for 0.6ms). Configure the maximum value for a 180° position (cycles for 2.4ms). Finally, configure its initial value for a 90° position (cycles for 1.5ms).
5. Configure the *Step Value* of the *setCounter* in such a way that the servo traverses its entire range between 0° and 180° (0.6ms to 2.4ms) in 15 clock cycles, when *enable* is set. Since later we will connect the push button to the *enable* input, this will allow us to move the servos from one end to the other by pushing a button 15 times.
6. Replace the *Constant* component connected to the *Relational Operator* with the output of the *setCounter* such that, you now compare the output from the PWM Counter with the value in the *setCounter*.
7. Modify the new counter design to not further increment but stays there when the *maxValue* has been reached on the *setCounter*, even if the up button is pushed (the down button should still work in this situation). Hint: add a comparator that compares against *maxValue* (this can be compare against constant block).
8. Expand the design additionally so that the *setCounter* does not go below the *minValue* but stays there, even if the down button is pushed (the up button should still work in the situation). You may apply the hint above to this challenge as well.
9. Add the suitable *In1* and *Out1* ports, and create a subsystem named *ConfigurablePWM* from your design. The subsystem should expose inputs *enable* and *direction*, as well as output *pwm* producing the final PWM signal.
10. For simulation purposes, add two *Constant* blocks set to 0 and connect them to the *enable* and *direction* signals.

Assignment 1

Take a screenshot of the design of block *ConfigurablePWM*, and include it in your report, together with a description of it. Before taking the screenshot, double-click on the *setCounter* component and move the pop-up window to a side. Make sure that your screenshot captures your design, as well as the properties you chose for the counter.

Simulate your design for 2 PWM cycles, and add a screenshot that shows the state of the output of the scope at the end of the simulation. Describe this output.

9.4 Push button pulses

In order to read accurate information from the push buttons, we need to account for the signal bouncing effects, as studied in a previous lab assignment. The goal of this section is creating a logic device that produces a pulse only after the push button input has stabilized for 250ms, with a repeat rate of 1Hz. Notice that we have worked on this component in the previous lab. Use your *debounce* design from that lab.

Now we will combine the PWM generator with the push button controllers in a design that generates a PWM signal with a pulse width that can be adjusted using the push buttons.

1. Using the *ConfigurablePWM* design from above, connect the *debounce* circuit to two inputs that will be linked to two pushbuttons.
2. The two *debounce* blocks' outputs will control the *ConfigurablePWM* block. Create the connections between them in such a way that a pulse coming from the first *debounce* block causes an expansion of the pulse width, while a pulse coming from the second *debounce* block causes its contraction. You may need some additional logic components in your design.
3. If your implementation of the *ConfigurablePWM* block is correct, receiving 15 pulses from either of these two pushbutton blocks should cause the pulse width to take all its possible values from the smallest to the largest.
4. Add a *Scope* component with three inputs. Two of them should be connected to the outputs of the *debounce* components, and the third should be connected to the output of the *ConfigurablePWM* block.

Assignment 2

For simulation purposes, we will assume that the ZedBoard works at a frequency of 50KHz (as opposed to the actual 50MHz). This means that 1 simulation cycle is equivalent to 0.02ms of virtual time. Convert to this scale by setting the appropriate counters to this frequency (This means scaling down the counter values by 3 zeros). When you later synthesize your designs into the ZedBoard, you will need to multiply all counter and constant values by 1000.

- a) Connect two inputs of your circuit to two *Constant* blocks set to 0. Take a screenshot of your design and add it to your report, together with a description of it.
- b) Simulate your design for 1s of virtual time, and take a screenshot of the output provided by the *Scope* component. Explain your results.
- c) Replace the first *Constant* block by a *Step* block that produces a signal set to 1 for the first 0.5s of the simulation. Observe how the first *ButtonToPulse* block generates the pulse at 250ms. In the *Scope* pop-up window, zoom into the transition 0→1 of the PWM pulse right before 250ms and record the exact cycle when it happens. Record the 1→0 transition for the same pulse. Then do the same for the 0→1 and 1→0 transitions for the PWM pulse right after 250ms. Based on the four values you recorded, what is the width of the first and the second recorded pulses?
- d) Now replace the second *Constant* block by a *Step* block of the same kind, and keep the first *Constant* block set to 0. Simulate the design and record the exact widths of the PWM pulses right before and after 250ms. Justify your results.

Lab 9.5 Control the Servos

When controlling multiple servos, each servo will have an own PWM duty cycle (position). This means, each servo channel will have to have an own threshold set by a *setCounter*. However, if we were to simply replicate *setCounter* and connect each to the same up/down buttons, they would all change at the same time simultaneously (end have the same value). In order only update a specific *setCounter*, you will need to control if a counter can be updated or not. This will allow you to select the *setCounter* being manipulated. In this step, we will expand the *setCounter* design to only change its value if enabled by a switch. Lastly, this section will replicate the updated set counter to keep track of the threshold values for all 5 servos (but not generate the PWM yet).

1. Design combinational logic that allows the enable and direction signals to pass to the counter only if enabled by when the switch is in the on position. If switch 0 is off, the counter value should not change, even though up/down button is pushed.
2. Validate your design with simulation.
3. Create a new subsystem out of your design above, naming it *SetCounterGated*. This design should have one additional input (*changeEnable*) as compared to the previous subsystem *SetCounter*. Note that *SetCounterGated* will later be used multiple times, so do not include the switch itself in the subsystem.
4. Replicate the subsystem *SetCounterGated* five times. Each *SetCounterGated* is still driven by the same up/down buttons. Connect each the *SetCounterGated* instance input *changeEnable* to one of the switches (0 through 4).
5. Simulate the design, and validate that only the counters (with *changeEnable* = 1, ie. selected by a switch) change their values.
6. Report your findings, and submit the design as part of your report.

The overall result should be that all 5 servos produce valid signals for each RC servo (continuously). Each servo that is enabled for changing (selected by switch), will increment by pushing up, decrement by pushing down button. If no servo is selected by the switches, no servo position should be changing.

Laboratory Report

You should follow the lab report outline provided on Blackboard. Your report should be developed on a word processor (e.g., OpenOffice or LaTeX), and should include graphics when trying to present a large amount of data. Upload the lab report on blackboard. Make sure all Simulink design files and screenshots of your simulation are included in the lab report.