Lab Assignment 9
Generating PWM Signals in FPGA to Control RC Servos

The goal of this lab is to design digital logic to control the RC servos of the Robotic arm. The digital logic will be implemented in the FPGA on the ZedBoard. The FPGA will generate Pulse Width Modulation (PWM) signals, which will control the RC servos in the robotic arm.

Similar to Lab 7, we will generate a PWM signal with 20ms period and a duty cycle between 600 μsec to 2400 μsec. However in this iteration, the FPGA generates the PWM, and the duty cycle is controlled by pushbuttons such that PBTNU increases the duty cycle and PBTND decreases a duty cycle. Switches select which servo(s) to move (more than one servo can move at the same time).

Instructions

- Each lab assignment consists of a set of Pre-Lab questions, the actual time in the lab, and a lab report.
- The Pre-Lab assignments prepare you for the challenges you will be facing in the actual lab. So that each lab member gets the same benefits, Pre-Lab assignments have to be solved individually. Feel free to discuss problems with class and lab mates, but complete the assignment on your own.

Reading List

The following reading list will help you to complete the Pre-Lab assignment. The readings will also help you in subsequent lab assignments. Please complete the readings that are marked ***[Required]*** before attempting the Pre-Lab assignments.

[1] Review general materials on PWM signal in blackboard
  - https://blackboard.neu.edu/

Pre-Lab Assignment

We have only very limited time for the lab available. To make efficient use of the lab time, you will need to prepare for it. First, go through the assigned reading above. Second, approach the questions below to apply your knowledge. Please submit the solution to the Pre-Lab assignments in PDF format via blackboard.

Pre 9.1) Generating a PWM signal in hardware

Create a Simulink design that generates a PWM signal. The signal should have a period of 2000 simulation steps (for simplicity we focus on simulation steps instead of time). For this, create an HDL counter that rolls back when 2000 is reached. For ease of identification later, name this counter *PWM_counter*. 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 the value of 150, and outputs zero otherwise.
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 and validate the correctness using the scope block. Report your findings in the Pre-Lab report.

Pre 9.2) Increment/Decrement Counter with Limits

The design in Pre-Lab 9.1 has only a constant duty cycle for generating the PWM. This Pre-Lab assignment explores how the duty cycle can be generated using a counter (and how it can be changed at runtime). We will reuse the increment/decrement counter design from Pre-Lab 8.1. We will enhance the counter so that it will stay within limits, which ultimately will yield valid PWM signals for a RC servo (duty cycle > 0.6ms and < 2.4ms duty cycle, as shown in **Figure 1** below).

Assuming a counter that rolls over every 20ms when reaching the value 2000 (similar to the one in Pre-Lab 8.1), answer the following:

a) 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*.
b) 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*.
c) 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*.
d) Modify the increment/decrement counter from Pre-Lab 8-1 to have a default value of *midValue*. Extend the counter design to not further increment when the *maxValue* has been reached, 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).
e) Expand the design additionally so that the counter does not go below the *minValue*, 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.

Make a subsystem out of the counter with max and min limiting elements and call it *SetCounter*. The subsystem should have two inputs (enable, direction), and one output (value).

Simulate your design with the Simulink simulator, validate functional correctness, and report your findings. Submit your design as part of the Pre-Lab.

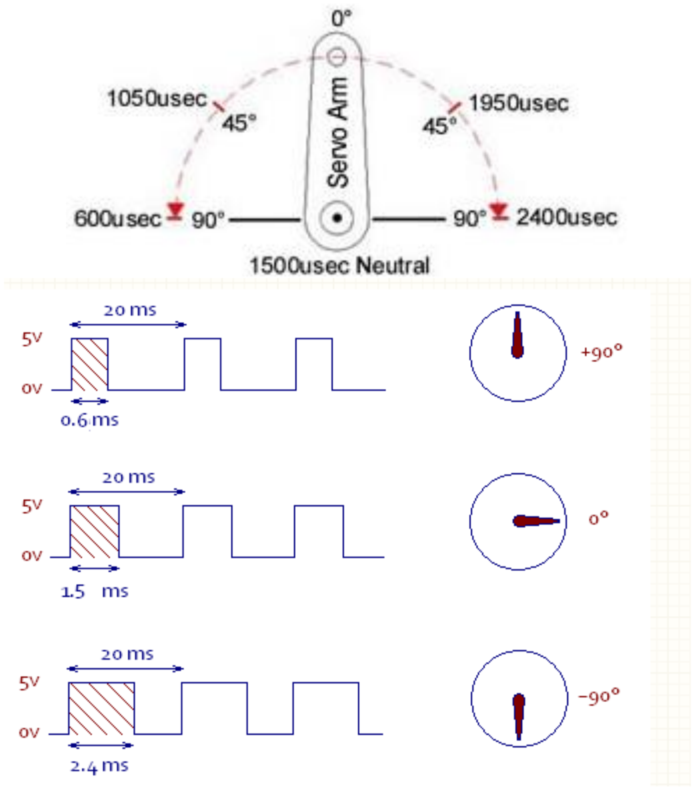

Figure 1. Servo motor operation for different PWM signal duty

Pre 9.3) Clock Divider Design

In this step, we will design a clock divider which later will drive the PWM generator from Pre-Lab 9.1. Assume an input clock of 50 MHz (this is the frequency of the clock on the ZedBoard). Design a clock divider that generates 2000 count pulses (these will be input to the *PWM_counter)* within 20ms. See **Figure 2** (below), it illustrates the input clock, and the output of the clock divider. Please show your calculations. Submit the Simulink design as part of your Pre-Lab.
Hint: See the cascaded counter from Pre-Lab 6.3 and counter driving LEDs in Lab 6.2 as examples.

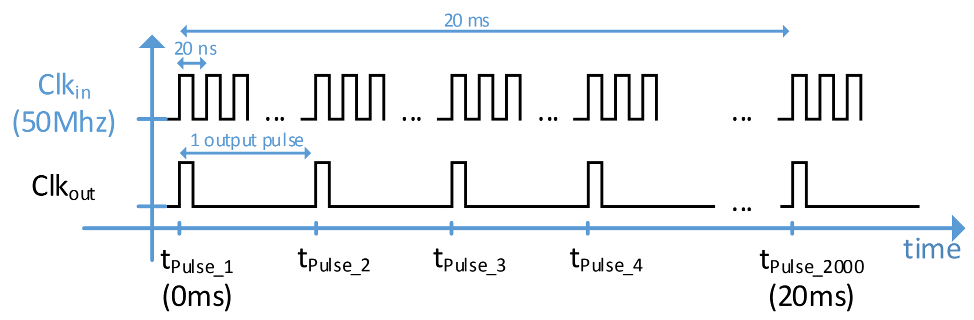

Figure 2. Clock Divider Output

Lab Assignments

In Lab 7, you developed a C program to control the Robotic arm using the Processing System (PS) side of the ZedBoard. This time, you will be controlling the Robotic arm using the Programmable Logic (PL) side of the ZedBoard. 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	Connector/pin	Switch
Base	JA[0]	Switch 0
Bicep	JA[1]	Switch 1
Elbow	JA[2]	Switch 2
Wrist	JA[3]	Switch 3
Gripper	JA[4]	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 enables by their switches, both servos will move at the same time.

The following steps will guide you through realizing this design.

Connecting to the ZedBoard

1. Login into the Windows desktop PC (we will refer to this system as the host) using your myneu credentials.
2. Make sure your Zedboard is powered (plugged in and the power switch on the board is in the ON position), and has been given time to boot up (generally takes a few minutes). Then connect the Zedboard to the PC host via Ethernet to the RJ45 connector.
3. Make an SSH connection to the Zedboard with root account, where the IP address is 192.168.1.10, and port number of 22.
   In this lab, you need to login with root

Lab 9.1 Single PWM signal for one Servo

Control the wrist servo connected to JA[3] using the pushbuttons up / down. For this devise a subsystem that generates a valid PWM signal permanently, and allow changing the duty cycle of the PWM signal by using up/down buttons. The following steps guide you through the process.

1. Instantiate the PWM_counter from Pre-Lab 9.1. Instantiate within the PWM_Counter subsystem a clock divider as configured in Lab 9.3. Now, the PWM_Counter should output a PWM signal with a 20ms period. Instantiate the *SetCounter* designed in Pre-Lab 9.2 as an input for the comparator. At startup, the design should generate a PWM signal of 20ms period with a 1.5ms duty cycle. “up” increases the duty cycle. “down” decrease the duty cycle.
2. A single push of the up/down button will increase/decrease the *SetCounter* by 1. How many steps are needed to pass completely through the whole range from a duty cycle of 0.6ms to 2.4ms? How should the step size be set so that the *SetCounter* can increment from *minVal* to *maxVal*with with about 15 button presses? Show your calculations in the report.
3. Update the *SetCounter*’s step size with the value found above.
4. Validate that your design generates a valid output for the RC Servo using simulation. For this set the “Model Configuration Parameters” to:
   a. Type: Fixed-step
   b. Solver: discrete (no-continues states)
   c. Fixed-step size: 1

   Using these settings the system will simulate individual cycles. We want to simulate the system for 40ms to observe 2 periods. Given that FPGA runs at 50MHz, how many cycles are in 40ms? Show the calculation in the report.
5. Simulate the system for as many cycles as you computed above. Validate, that a valid PWM signal is generated. Report the simulation results in your report.
6. If you have not yet connected the robot, power down the Zedboard and connect the robot. Then power up the Zedboard. Remember, if you boot the Zedboard, the mapping of the switches/button will be reprogrammed during the boot process, so you will need to reprogram the FPGA to reestablish the memory mapping of these inputs.
7. Synthesize your design using the HDL workflow advisor. Make sure to use the correct pin assignment.
8. Do not forget that you need power to the daughter card that interfaces between the ZedBoard and the Robotic arm.
9. Report your results and errors that you may have found.

Lab 9.2 Enable/Disable SetCounter (Validation in Simulation Only)

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 off 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.

Lab 9.3 Control all Servos

Now that we have five threshold (one generated by each *SetCounterGated* instance), we can use this to derive five different PWM signals. Each PWM channel could have an own PWM counter. However, since all PWM outputs will have the same period, the design can be optimized using a single PWM_counter.

To generate the 5 PWM signals with the same period, use the same PWM_counter, but compare it against five different values (as provided for by each *SetCounterGated* instance).

1. For each of the *SetCounterGated* instantiate a remaining relational operators to compare against the output of PWM_counter, and connect the outputs to the servos.
2. Synthesize the design and load it to the FPGA. Make sure that the robot is turned off.
3. When turning on the robot, pay attention as it will rapidly move to the middle position.
4. Play with your robot. Report your findings (including errors you have encountered along the way).
5. Don’t forget to take a video of your working design.

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.

Lab 9.4 Extra credit

1. Modify your design such that it will be able to control the speed. Use left pushbutton to increase the speed and the right pushbutton to decrease the speed.

Laboratory Report

You should follow the lab report outline provided on Blackboard. Your report should be developed on a wordprocessor (e.g., OpenOffice or LaTeX), and should include graphics when trying to present a large amount of data. Upload the lab report on blackboard. Include each program that you wrote in the appendix of the report.