Pre-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 7.1 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.

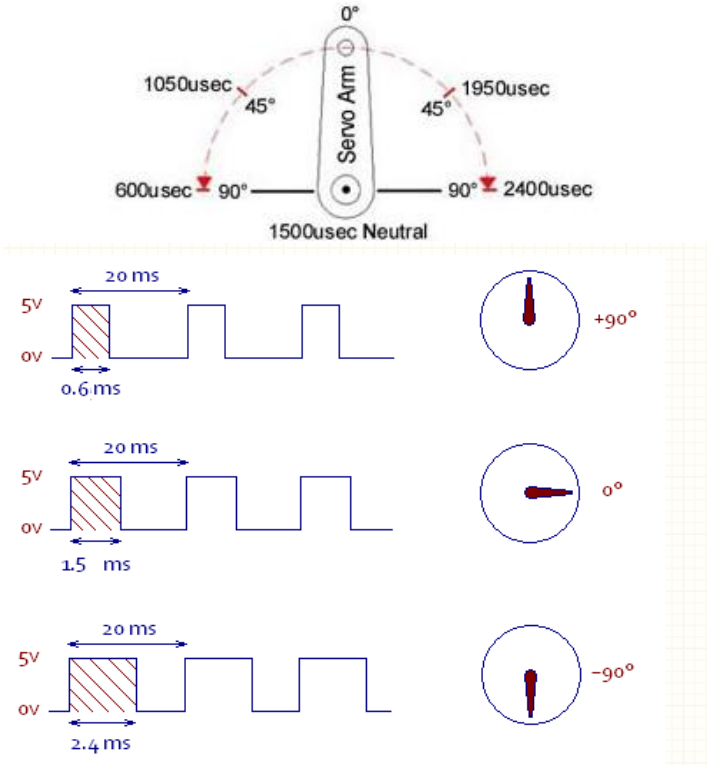

Figure 1. Servo motor operation for different PWM signal duty

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.

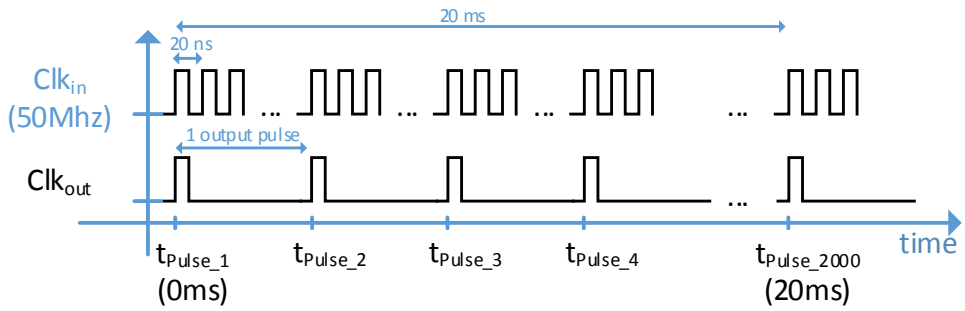

Figure 2. Clock Divider Output