Lab 4
Embedded Systems
Fall 2015

Pulse Width Modulation for Steering and Speed Control

Physical Objective:

Use four switches to represent 16 different values for the steering angle and motor speed.

Learning Objective:

Understand how to look at and embedded systems parts and design a hardware unit to control it. Understand what pulse width modulation (PWM) is and how to create one.

What you have to implement:

- Steering PWM hardware control module
- Speed PWM hardware control module
- Top level Steering and Speed control unit
- Testbench for steering and speed pwm modules

Breakdown:

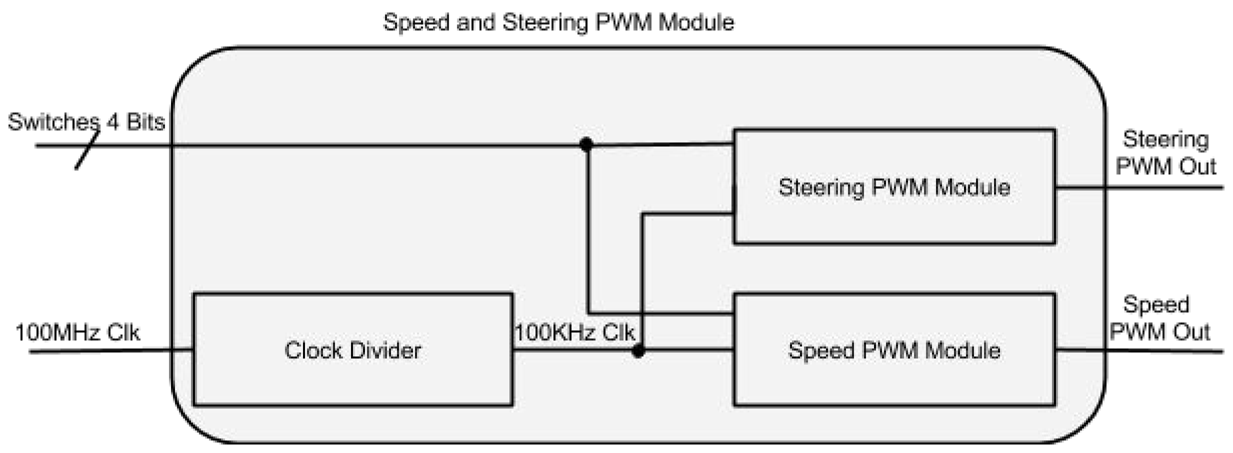
- Create project
- Create Steering PWM hardware control module
- Create Speed PWM hardware control module
- Create Top Level Steering and Speed module
- Write Testbench and simulate design (Get checked off)
- Add Processing System
- Set Pinout
- Synthesize, Implement, Create bitstream
- Test on car (Get checked off)

Introduction:

In this lab you will implement a Pulse Width Modulator (PWM) to control the Steering Servo and the Electronic Speed Control (ESC) system of the RazorCar. Input to the modulator will be provided by the switches of the extension board, and the generated pulses will feed the servo and the motor of the car to give both directions (left and right) and speed (forward, stand or backward). You will use the Clock divider module to produce a frequency of 100KHz from a 100MHz input clock that will be used by the PWM modules.

The Architecture:

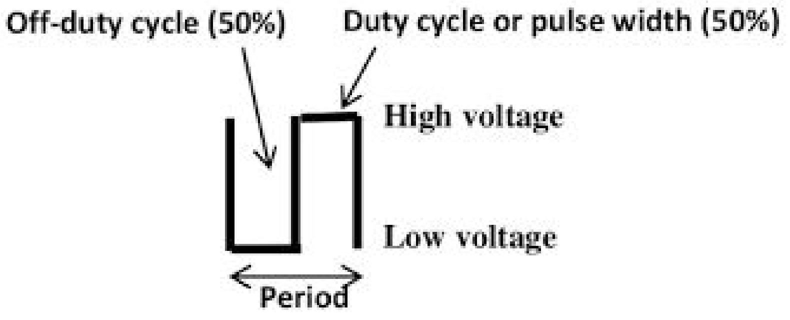
The architecture that will be designing in this lab is shown below.



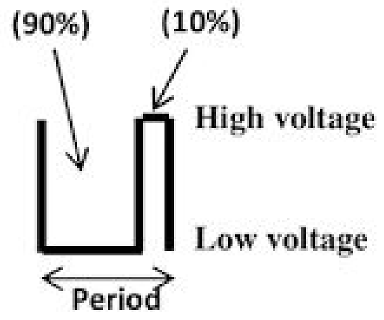
The system takes as input four bits from the switches and a 100MHz clock, and outputs two modulated clock signals that will feed the steering servo system and motor electronic speed control (ESC).

Pulse Width Modulation (PWM):

PWM is a technique for controlling analog devices via a digital signal. It works by simply adjusting or modulating the width of the on-duty cycle or pulse width (signal = 1) during a time period. The PWM mechanism is used for any device that needs an adjustable square wave length to run. For a normal square wave, the on-duty cycle is equal to 50% of the time period (or half of the period). We used so far such waves to generate specific frequencies to slow-down the speed of the Lights counter and to trigger the Buzzer system. In PWM, this proportion (length of the on-duty cycles) will be constantly adjusted to provide commands to a given analog device.



a) Normal Square Wave



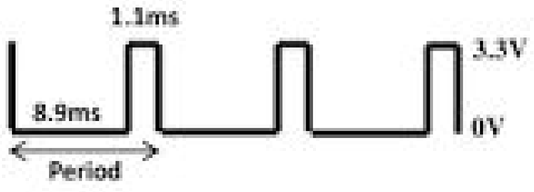
b) Modulated Square Wave

Technical properties of the Steering & Speed Controller on the RazorCar

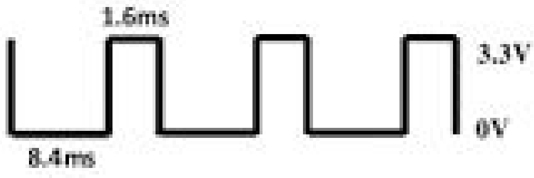
The RazorCar has the following technical properties:

a – Steering

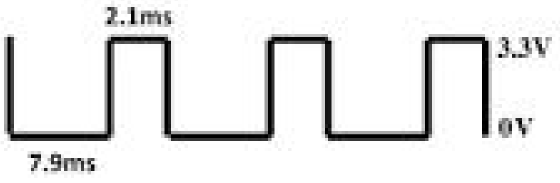
1) Full left-turn (pulse width – 13%)



2) Go Ahead (pulse width – 16%)

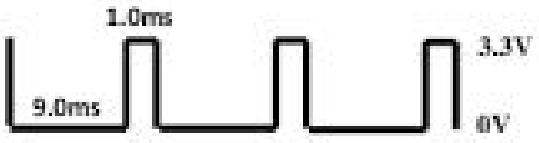


3) Full right-turn (pulse width – 19%)

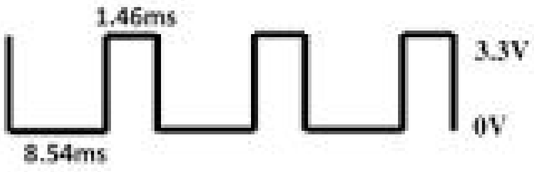


b – Speed

1) Back-up (pulse width – 10%)



2) Stand (pulse width – 16%)



3) Full Speed-up (pulse width – 19%)

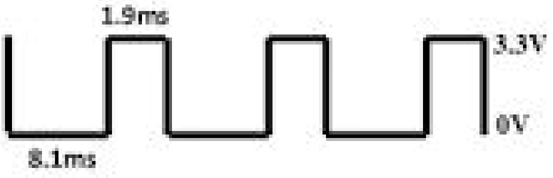


Fig.3 – Technical Properties of the RazorCar

- Period of a cycle for the servo and the speed control system: 10ms
- Traversal Range to operate the steering servo: [1.1ms – 2.1ms]
- Traversal Range to operate the speed control: [1.0ms – 1.9ms]

In addition to the above information you must make sure that the PWM signal coming out of the Speed PWM module starts up at the standing state otherwise the ESC will not work.

Create project:

- Open Vivado
- Create a new project
- Add your clock divider

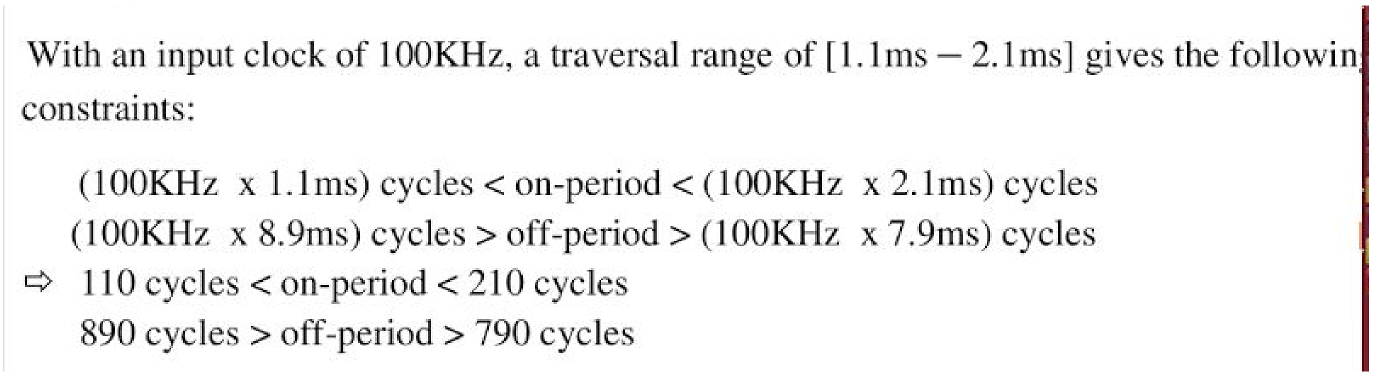
  Right click on “Design Sources” and select “Add Sources…” Make sure that “Add or Create Design Sources” is marked and select next. Add your file and finish.

Create Steering PWM hardware control module:

- Create Steering Module

  Right click on “Design Sources” and select “Add Sources…” Make sure that “Add or Create Design Sources” is marked and select next. Create your Steering module naming it what you wish. For the “Define Module” dialog box you need to determine what inputs and outputs you need as well as their bit length. If you scroll up to the Architecture section you can see the Steering PWM modules inputs and outputs as well as the bit lengths. Use this as a reference when feeling out the Define Module Dialog box and click OK to complete the modules creation
- Code logic for steering PWM module

  If we look at the technical properties of the steering control on the RazorCar from above we can see that the off period goes from 8.9ms to 7.9ms and the on period goes from 1.1ms to 2.1ms . This means that we have 1ms that we need to divide up with our 16 possible switch combinations. With this division part will go to the off period and part will go the on period (duty cycle). If we take 1ms/16 we get 0.0625. We can round this to 0.06ms. The following shows why.



With an input clock of 100KHz, a traversal range of [1.1ms – 2.1ms] gives the followin
constraints:

(100KHz x 1.1ms) cycles < on-period < (100KHz x 2.1ms) cycles
(100KHz x 8.9ms) cycles > off-period > (100KHz x 7.9ms) cycles
⇨ 110 cycles < on-period < 210 cycles
890 cycles > off-period > 790 cycles

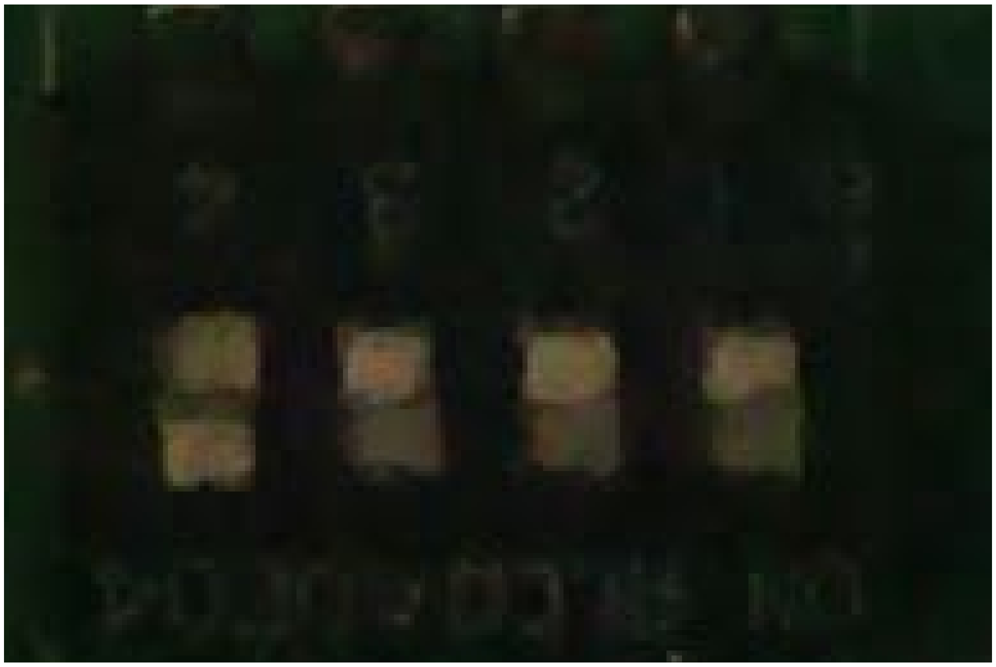
Therefore: on-period = 110 + (steeringBitIn * 6) and off-period = 890 - (steeringBitIn * 6).

Create Speed PWM hardware control module:

- Create a new VHDL file like you did above when creating the steering module. Use the architecture section to show you what inputs and outputs you need.

  You will create the Speed PWM module similar to what you did with the Steering PWM module. **<u>The major difference is that at the midway setting of the switches, 8, “1000”, we need to make sure to set the on period for 1.46ms and off period for 8.54ms.</u>**

  <u>When we upload our bit stream we need to set the switches to output 8, “1000” as pictured below otherwise the ESC will not calibrate correctly at startup.</u>



Create Top Level Steering and Speed module:

- Create a new VHDL source file for your top level file

  It should have a clock and 4 bits from the switches for input, and the steering and speed PWM module signals for output. You will need to instantiate your steering module, speed module, and clock divider.

Write Testbench and simulate design:

- Create a testbench to set your two new PWM modules.

  Right click on “Simulation Sources” and add a testbench to your project. Go through all of the 16 possible switch settings and check that your input and outputs are correct. Don’t ask me to check you off if you don’t know what the waveform should look like. You should know what to expect from you waveform. If you do not know what you should be seeing then you should ask before running the waveform. You will have to do some light math to be able to determine what is expected from your PWM units.

***** 3 screenshots of this waveform must be in your lab report. Front, last, and middle of the testbench. *****

***** Get Checked Off *****

Add Processing System:

- Add the processing system as we have in previous labs.
  Run Block automation, Create the clock port, change the clock to 100MHz, and connect FCLK_CLK0 to M_AXI_GPIO_ACLK. Save your file and exit (upper right X) Create the wrapper file for the .bd file.
- Delete the clock from the top level entity. Create your own internal signal for the clock. Instantiate the PS wrapper and connect the processing system to the clock divider.

Set Pinout:

- Using the pinout from moodle set the pins.
  Assign pins for the switches, the speed and steering PWM signals. Save after creating pin assignments. Save as filename whatever you like.

Synthesize, Implement, Create bitstream:

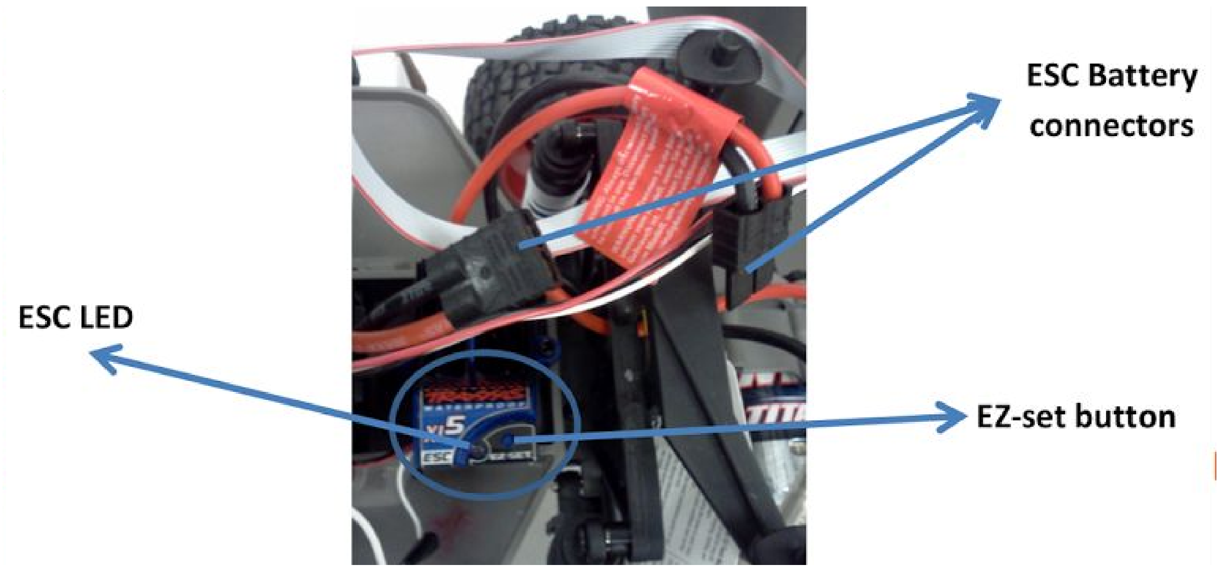
- Run Synthesis, Implementation, and Generate Bitstream.
  Correct any errors that occur along the way.

Upload to car:

- Upload to car and test for correct action.

<u>MAKE SURE YOU HAVE THE SWITCHES SET TO 8, “1000”. If you don’t the speed controller will not work</u>.

<u>**Before using the speed control system, MAKE SURE THAT THE TIRES OF THE CAR ARE NOT TOUCHING THE GROUND OR ANY FLAT SURFACE!! ****</u>** Then, plug the battery to the ESC Speed control box and press (and hold) the EZ-set button for one second.If the board is already programed, the LED on the ESC should turn red.



Make sure all switch settings work.

*****Get Checked Off*****