CSCE 236 Embedded Systems, Spring 2012
Lab 3

In Class: Thursday, February 9, 2012

All Checkoffs Due Before Class: Tuesday, February 21, 2012

Names of Group Members:

1 Instructions

This is a group assignment to work on during class. You only need to hand in one copy of this, but make sure that the names of all of your group members are on this sheet to receive credit. Complete all of the sections below and make sure to get the instructor or TA to sign off where required.

2 Dimming LEDs

In this section you will take two different approaches to solving the same problem. The goal is to dim one of the LEDs whenever the button is pressed. You can dim an LED by only turning it on for a very short amount of time and then turning it off for a period of time. If you do this faster than about 30 Hz, you will not notice the flickering of the LED turning on and off and it will appear dimmer than if it is fully on (since the human eye acts as an integrator at this frequency).

2.1 Manual

First, you will achieve this by using manual delays. In your `loop()` write code that turns the LED fully on when the button is not pressed and then dims it when the button is pressed. You can use the function `delay(ms)` to achieve dimming as described above. What delay values work well? If you increase the amount of time the LED is off, at what frequency can you start to see the LED pulsing?

The problem with this approach is that the processor is always busy doing the pulsing of the LED to dim it. If a function is called that prevents this from occurring, the LED will not dim properly. To see this, insert another delay of 100 milliseconds outside of your logic that controls the LED. This simulates the execution of other code that takes a while to complete. What happens when you press the button now?
Checkoff: *Show the LED dimming when the button is pressed and have answers to the above questions.*

2.2 PWM

To overcome the problem that you cannot easily do other actions while dimming the LED, you will now implement LED dimming using PWM. To do this, connect your LED to pin `PB2` (pin 10 on Arduino), which is also `OC1B` (output compare `B` for `timer1`). Now, look at section 16 (Timer/Counter1) of the datasheet (the description of fast PWM mode in 16.9.3 and the register description in 16.11 are especially useful). Configure it such that:

- the timer is in fast PWM mode, with `ICR1` as the top
- OC1B is cleared on compare match and set at bottom.

You will also need to:

- set an appropriate frequency by configuring the clock prescalar and the TOP value (stored in `ICR1`)
- set the duty cycle by writing to the output compare register OCR1B

- set `PB2` as an output pin to enable the PWM output.

Checkoff: *Write the code using the PWM system to dim the LED when the button is pressed. Note how adding code (e.g. delay(100)) in the main loop no longer impacts the dimming.*

3 Servo Control

Servos are also controlled by PWM signals. Most servos enable angular control of the output shaft (e.g. between 0 and 120 degrees). The servos we are using in class are known as continuous rotation servos. Instead of controlling position, we are able to control the forward and backward rotational speed using PWM signals. Most servos use PWM signals with a period of approximately 20 milliseconds. By varying the pulse length between 1ms to 2ms the servo will go from full speed forward to full speed backwards (on a traditional servo this would mean rotating from 0 to 120 degrees).

You have seen how to manually configure the PWM channels to control the frequency and duty cycle. However, instead of manually configuring all the PWM channels, we will instead make use of an Arduino servo control library. Read the information on how to use the servo library here:

`http://arduino.cc/en/Reference/Servo`

To hook up the servos to your breadboard, you will need to first use the 3-pin header to adapt the female servo connector to a male connector. Then, connect the *black wire to ground, red to power, and white to the PWM signal.* Double check these connections before powering your board. Also note that the servos can draw more power than a USB port can supply. Therefore, you should not restrict the output of the servo while using USB power, instead make use of the AC adapter to power the servos whenever AC power is available.

Checkoff: *Demonstrate controlling a servo by making the direction reverse when the button is pressed.*

4 Building Robot and Driving Primitives

This section of the lab requires individual checkoff for each person. I do not expect you to complete this in lab so, please come and get checked off before the start of class on Tuesday, February 21.

Checkoff: *You should now use the supplied parts to build your own robot as demonstrated in Lab. Create driving primitives that enable you to drive in a straight line and also rotate a particular angle (e.g. 90 degrees). Program your robot so that when you press the button it will drive a square.*