Physics 364, Fall 2012, Lab #10a

(Digital Communication: PWM and Serial Protocals)

Lab for November 12th

In this lab we will examine two common methods used for communicating between components in a circuit. The first will deal with a system for transmitting an analog value called Pulse Width Modulation(PWM). This system relies on measuring the duty cycle of a square wave to store information. The second part of the lab will focus on the serial protocol, which encodes a series of bits on a single pin with a fixed protocol.

PWM signals are commonly used for controlling stepper motors and servo motors. The signal sent can either encode the rate of rotation, or the degree position to rotate to. In this lab we will be using the former system. The Bobot in your kit has two motors on it that we will be learning to control. The wiring for these is straight forward – Red is 5V, Black is GND, and white is the PWM signal. **Be careful not to let your Bobot do something like drive off the table. They can move very quickly if you aren't watching them. Batteries are available if you want them.**

So how do we produce this signal? You could use the square wave code you wrote in the last lab to create a PWM signal, but this has the disadvantage that it takes up the CPU almost entirely. Workarounds for this would result in a timing jitter on the signal as well. Instead the Arduino has conveniently given us ports that output a PWM of a desired duty cycle.

To turn on the PWM pins, find a pin marked PWM on the board and use the analogWrite command:

```
analogWrite( pin, DutyCycle);
```

Where pin is a PWM capable pin in OUTPUT mode, and DutyCycle is a value between 0 and 255. **Be careful: analogWrite does NOT output an analog value in the same sense our DAC did in lab9. The analog nature is PURELY in the dutyCycle, the pin itself is only ever 0 or 5 volts.**

Exercise 1: Hook up a PWM capable pin from the Arduino to the scope and confirm that it is outputting the desired wave. Try varying the DutyCycle value – which corresponds to 5V and which to GND? Is 0.5 at 128? What frequency does the wave correspond to?

Next we are going to work out how to control our Bobot using PWM. Connect the two motors as described above to two different PWM outputs. You should use the

breadboard shield attachment so you can do the wiring on the Bobot itself. We want to now add some functions to the Arduino code that will correspond to controlling the motors. This abstraction makes the code more readable, and it also lets us reuse code rather than rewrite it all the time. Below is an example function called 'turnRight()' which you would call by providing it an integer specifying how fast to turn. For example if you add this code above your setup function:

```
void turnRight(int rate)
{
  // Put your PWM settings here
}
```

You could call it from your main loop with the command:

```
turnRight(10);
```

Exercise 2: Calibrate the Bobot to have stationary wheels at a PWM setting of 128 – this is accomplished by carefully tuning the screws in the back of the motors until they are completely still. How high and low in duty cycle can you make it go before it stops functioning properly? Note that GND and 5V are both stationary because they are really not the right protocol. Make functions for turning right, left, stopping, reversing, and going forward. For the stop function it is probably worth using a GND signal rather than 128, as the motor is less confused by this. You may wish to keep speeds slow enough that you can keep control of the Bobot. Test them out – can you make your Bobot perform a pattern?

We spent last lab working hard to get an ADC working, but it turned out this was mostly to learn how they worked. As a practical matter we would just buy one, and it so happens that there are several built into the Arduino. In order to use them, we use the analogRead(pin) command. Keep in mind this is referring to the pins labelled analog inputs, NOT the digital pins. You do not need to set a pin mode for analog pins as they are all inputs on the this board.

Exercise 3: We would like to make the Bobot a light seeking robot, but this is a pretty big task so we're going to do it in pieces.

First, setup a serial connection as in lab9b and write the results of your ADC(ie, analogRead) conversions back to the computer for monitoring. Try putting 0 and 5V into the input and check that the results are what you expect. How many bits does this ADC have?

Next take a photoresistor from the supply shelf and place it in a voltage divider. Do you want it on the Vcc or the GND side? Monitor the voltage at Vout, and then add

code which will turn on an LED if the light goes below some threshold. This is sortof like an automatic night light or sidewalk lantern.

Take out your LED and wire up a second photoresistor next to the first so that they are pointing like eyes out of the front of the Bobot. Write a program that will compare the light on the two "eyes", and try to balance them to within a window. Once you have a robot which will turn to follow the light, adjust the code so that if it has found the right direction it will move forward. A good feature here would be to have the Bobot remain stationary in the absence of bright light so that the room lights don't trigger it but a flashlight does. Remember to not let your Bobot run off a table!