Lab 8 - Microcontrollers
PHGN 317 - Digital Electronics

Galen Vincent

July 18, 2019

Abstract

The goal of this project was to gain experience with micro controllers and their functions by using an Arduino to design different circuits for an application. The project that I built is a basic wheel speed control, where the speed of a spinning wheel can be controlled by the user, measured, and displayed. Through the process of building and testing this design, I learned about the different abilities of the Arduino, and how its functionalities make designing complicated systems more simple.

1 Introduction

The main objective of this project was to learn about the properties and functionality of the Arduino micro controller. The lab was largely left up to us to decide what sensors and circuits to use, but the big goals were to practice using two of the Arduino's main functions: analog output using pulse width modulation (PWM), and digital input & output (DIO). I decided to explore the Arduino's functionality by building a wheel speed controller. This controller uses many different circuits, which all talk to each other, in order to set, control, measure, and display the speed of a spinning wheel.

This application of speed control is one that is seen in every day life quite often. For example, a car speedometer and cruise control needs to be able to measure the speed of a spinning wheel, process that information, and accelerate or brake the car accordingly. Some of my inspiration for how to measure the speed of a spinning wheel came from a bike speedometer, which uses a magnet placed on the spokes of a bike passing by a stationary sensor in order to calculate the speed of the bike [1].

The reason that I pursued this project is that I wanted to gain some insight into how some professionally engineered products work on the inside. The wheel speed control seemed like a project where I could do this with an Arduino, as most bike and car speedometers use a very simple mechanism to measure speed.

2 Technical Overview / Summary

Before I began the design of the project, I gained some basic information about the Arduino. I learned that the Arduino uses the ATmega328P micro controller for its processing. It has an operating voltage of 5V, and a recommended input voltage of 7-12V, with lower and upper input voltage limits of 6V and 20V, respectively. There are 14 DIO, 6 of which have PWM capabilities, and 6 analog input pins. The input pins have a resolution of 10 bits (1024 possible values), or 0.00488V for a 5V max input. It can supply 20 mA of current per I/O pin. It has 31.5 KB of available flash memory, and has a clock speed of 16 MHz [2].

My wheel speed controller consists of a DC motor attached to a wheel so that it can control the speed of the wheel's rotation. On this wheel, there is a small magnet placed near the rim, which passes by a reed switch circuit once per revolution. When the magnet passes by the reed switch, the switch is closed for a short period of time. My Arduino code measures the time in between these passes and uses this information to calculate the measured RPM of the wheel. The speed of the DC motor connected to the wheel is controlled by the user with a potentiometer. The output voltage of this potentiometer is converted to a certain input value for the DC motor, which corresponds to a certain known RPM value. Both the RPM set by the user with the potentiometer, and the RPM being measured with the reed switch circuit are then displayed on an LCD display screen, so that the user can see if there is a difference between the RPM that they set, and the one that the wheel is actually spinning with. Figure 1 shows a simple block diagram of the different systems of the project, and how they pass information to one another. The details of each individual circuit will be addressed in the following sections of this report.

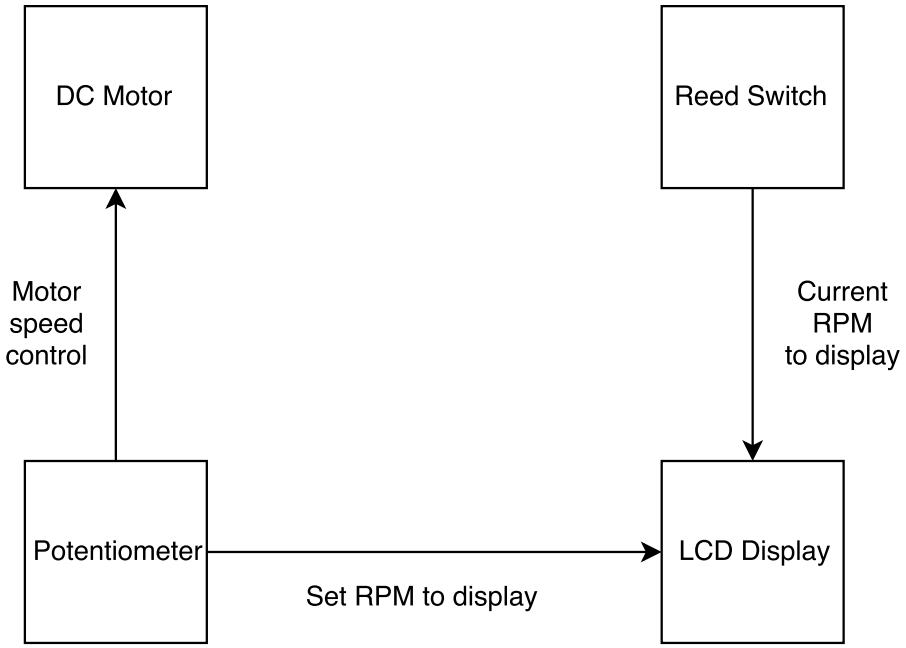

Figure 1: Simple block diagram that helps to visualize how the different systems within the project feed information and interact with each other.

3 Design Documentation

Figure 2 shows what the completed project looked like, with each of the individual sections labeled. These different circuits will be detailed in the following sections. A PDF of all the commented Arduino code used to run and control this project can be found in the appendix. The details of the snippets of the code relating to each circuit will be explained in the appropriate section of the report below.

Note that for all of the circuit diagrams to follow, pin names prefaced by "Ard" are pins that connect to the Arduino, and pin names prefaces by "PS" are pins that are connected to an external power supply.

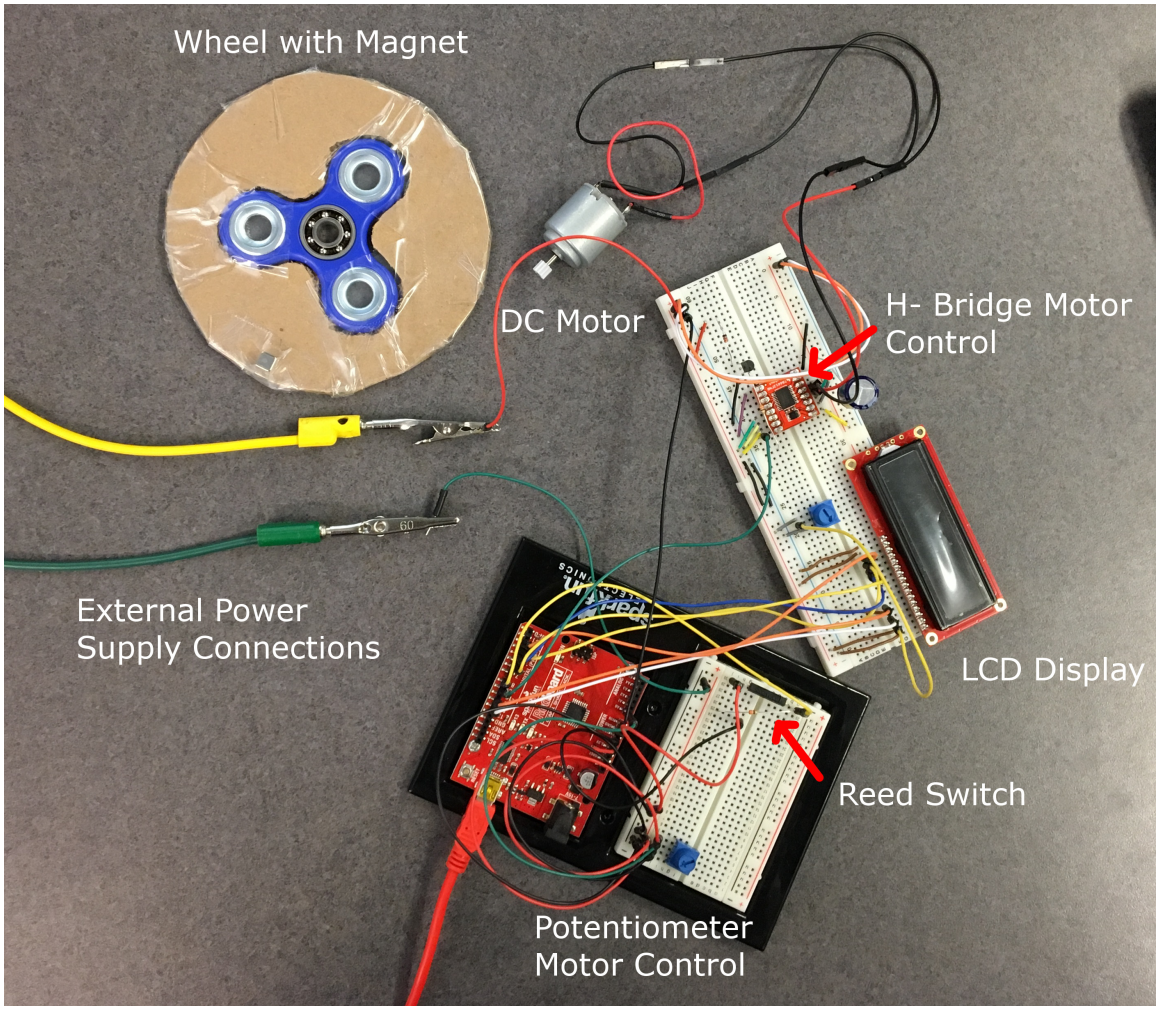

Figure 2: Completed project with the different parts & systems labeled.

3.1 Subsystem A: Potentiometer Speed Setting

The potentiometer circuit is how the user can set the speed of the spinning wheel. The circuit diagram is shown in Figure 3.

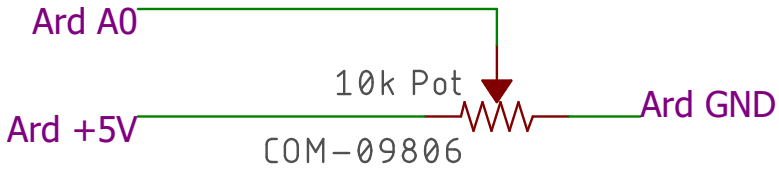

Figure 3: Circuit diagram for the potentiometer circuit.

The potentiometer is a basic voltage divider, and the voltage read into analog pin A0 of the Arduino is based on the position of the knob of the device. The voltage value read into A0, which can be a value between 0 and 5V, is read as a value from 0 to 1023 (10 bits). My code then maps, or linearly scales, this value, to a new value between 0 and the maximum

motor input value. This maximum motor input is defined by the user, and can take on a value between 0 and 255 (8 bits), as these are the range of values that can be used to control the speed of the motor. This new value is then passed to the DC motor control code to set the speed of the wheel rotation.

The code in Figure 4 shows where the potentiometer position is read into the Arduino in the loop() section of the program using a function which I defined. This function keeps track of the current and previous state of the potentiometer, maps the current value onto a value between 0 and the maximum motor speed indicated by the user, and stores this value in a global variable which can then be passed into the motor control circuit and the LCD display elsewhere in the code.

```
void loop() {
  // put your main code here, to run repeatedly:

  // Read the potentiometer to determine desired RPM
  readPot();

 // Function to read the potentiometer value
 void readPot(){
   // Set the previous pot value to the current pot value
   prevPotValDC = potValDC;
   // Read the analog input pin
   potValDC = analogRead(potPin);
   // Map the value to the correct value to pass to the motor
   potValDC = map(potValDC,0,1023,0,motorMax);
   return;
 }
```

Figure 4: Code to read and correctly map the values of the current potentiometer position.

Figure 5 shows an oscilloscope measurement of the voltage on the output pin of the potentiometer while the dial is being swung from one extreme to the other, and back. This shows the full range of the potentiometer output value, which is about 0V to 5V.

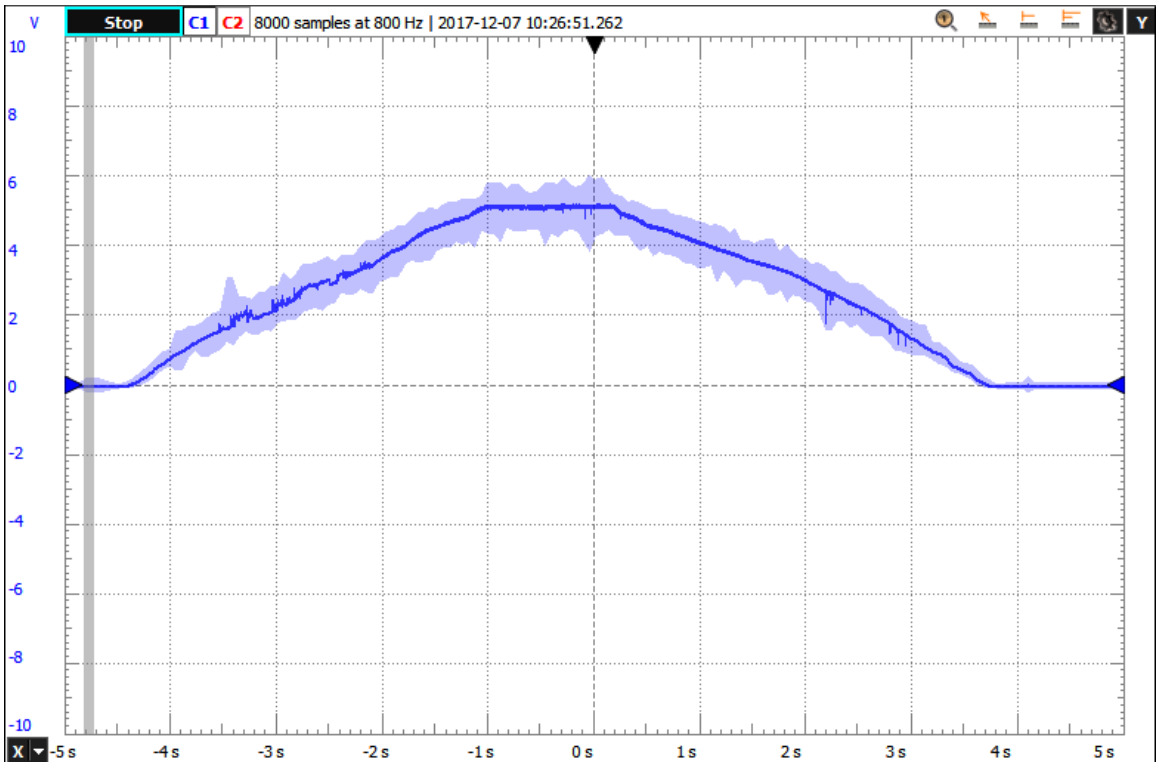

Figure 5: Signal measured from the potentiometer as its control wiper is swung around from one maximum to the other and back. It is evident from the figure that the output voltage range is between 0V and 5V.

3.2 Subsystem B: Initial DC Motor Control

The control circuit for the DC motor went through two iterations before it was fully working. This is the first of those two designs. The circuit schematic shown in Figure 6 was inspired by the SparkFun SIK Experimentation Guide [3]. The DIO9 pin inputs a PWM signal into the transistor base, which regulates the amount of current that flows through the motor. Because the speed of the DC motor is controlled by how much current flows through it, this allows the PWM signal on DIO9 to control the speed of the motor, and therefore the speed of the wheel. Another important note about the DC motor circuit used in this project is that the current used to drive the motor is taken from an external power supply, as can be seen in Figure 6. The Arduino's current output capabilities are not meant to be able to drive current hungry things like motors, so the external supply is needed to make the motor work.

The diode in the circuit is to protect the circuit from the back EMF produced when the motor stops quickly. The code used to control this circuit can be seen in Figure 7. This code shows how a PWM value between 0 and 255 is written to DIO9 of the Arduino to control the speed of the motor. In this code, the position of the potentiometer is controlling the speed of the circuit.

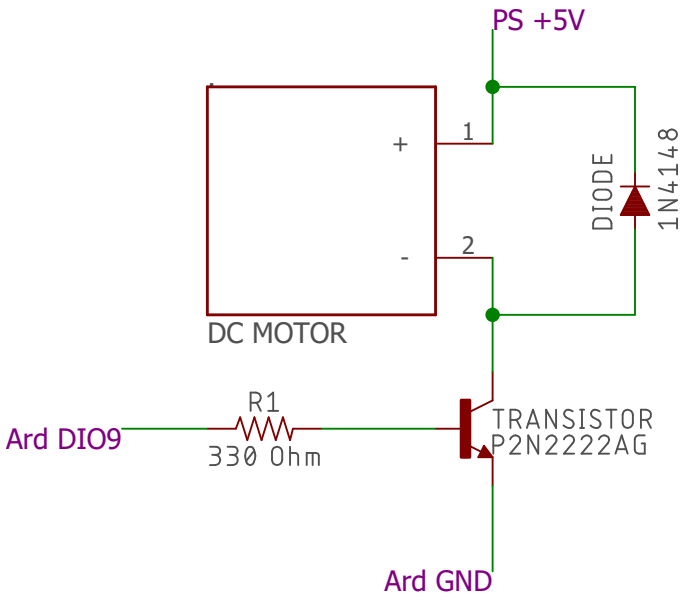

Figure 6: Circuit diagram for the initial DC motor control circuit.

```
const int motorPin = 6;
int val;

void setup()
{
  // Set up the motor pin to be an output:
  pinMode(motorPin, OUTPUT);
}


void loop()
{
  val = analogRead(0);
  val = map(val,0,1023,0,255);
  analogWrite(motorPin,val);
  delay(100);
}
```

Figure 7: Code for the initial DC motor control.

This circuit was scrapped from the project because of the large amount of noise that it produced across the Arduino +5 to GND power buses. This noise was heavily impacting the other circuits of the project, and made controlling any other circuits while the motor was running impossible. I think that this noise was a result of the motor not being far enough removed from the Arduino. The noise can be seen on an oscilloscope capture in Figure 8.

I attempted to reduce this noise by putting a 1000 μF capacitor across the power and ground buses of the Arduino to create a low pass filter, but the noise was still large enough to impact the rest of the circuit. This attempt at reduced noise can be seen in Figure 9.

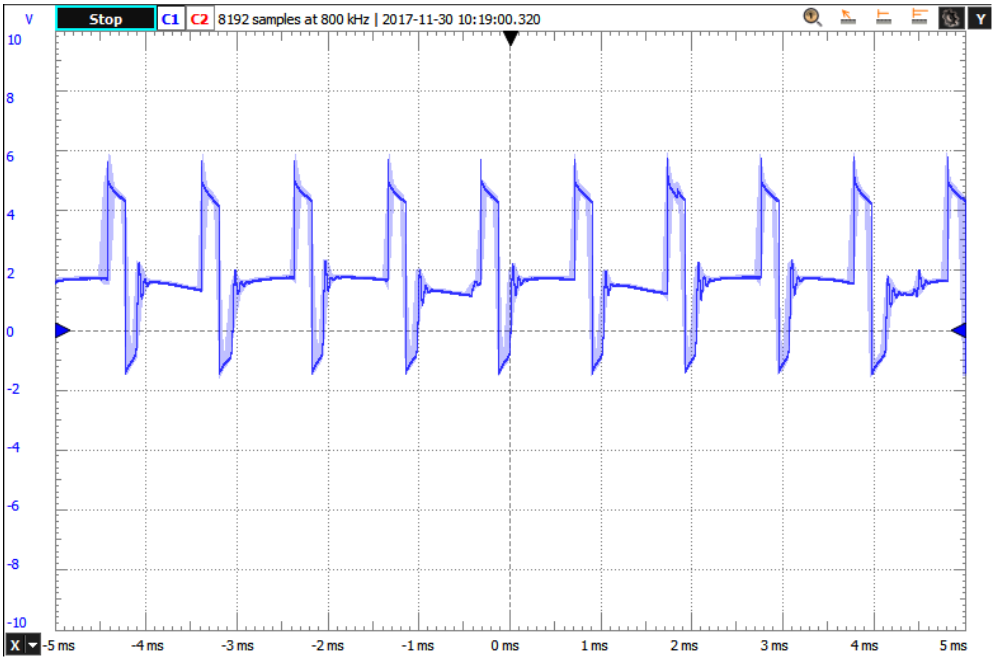

Figure 8: Noise measured across Arduino +5V to GND caused by the DC motor. Measurements were made using the Analog Discovery 2.

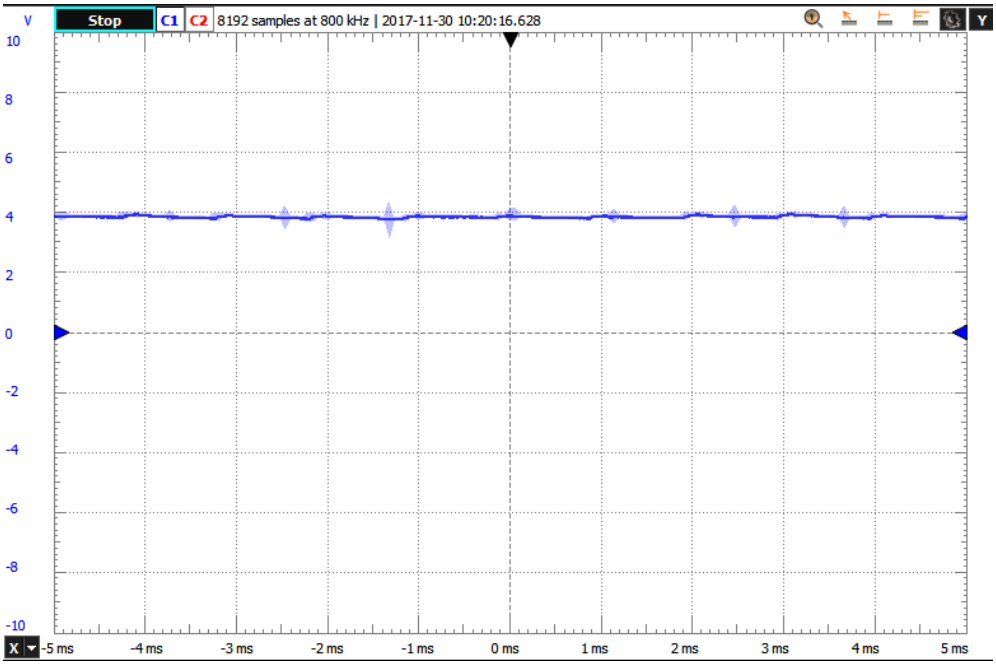

Figure 9: Noise measured across Arduino +5V to GND caused by the DC motor after a 1000 μF capacitor was placed between the power and ground buses in order to create a low pass filter. Measurements were made using the Analog Discovery 2.

After contemplating this problem for a while, I moved on to the second DC motor circuit, which is described below.

3.3 Subsystem C: Final DC Motor Control

In order to reduce the noise from the DC motor circuit that was encountered above, I tried to isolate the motor from the Arduino as much as possible. To do this, I used an h-bridge motor controller IC. The connections between the Arduino, the chip, and the DC motor can be seen in Figure 10.

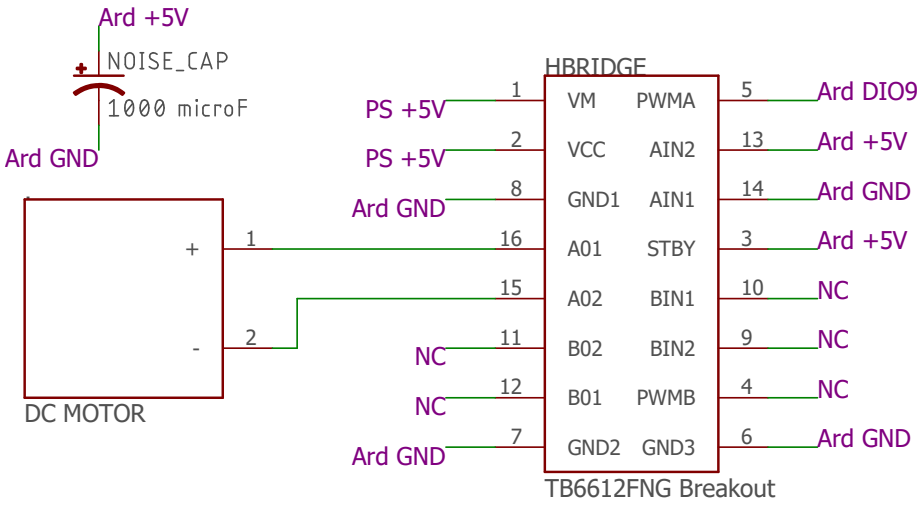

Figure 10: Circuit diagram for the final DC motor control circuit.

An h-bridge is an IC that is designed to control two motors at the same time, and increase functionality so that the user can control the direction and speed of each. For my application, I only needed to use one of the motor outputs, and I hard-wired all of the pins that control direction (AIN1 & AIN2) so that the motor would always spin in the same direction. If I wanted to, I could have implemented the ability to change the direction of the spinning wheel by connecting these control pins to the Arduino and writing some basic code to do so. In order to control the speed of the motor in this circuit, a PWM signal is passed from the Arduino into pin 5 of the h-bridge.

The code to control the motor with this circuit is shown in Figure 11 below. This code is essentially the same as the code from the previous section. The first block defines the maximum motor speed global variable. The second block of code is in the setup() function of the program, and sets the correct pin number as the motor output. The third block of code writes the value read and mapped by the potentiometer function to the motor. This way, the motor speed, and therefore the wheel speed, is controlled by the potentiometer position.

```
// Motor speed calibration - Motor PMW value that gives about 200 RPM
const int motorMax = 100;

// Set up DC Motor pin as an output
pinMode(motorPin, OUTPUT);

 // Write the value 0 - maxMotor (calibrated by the user) to the DC motor, only
 // if the value from the pot has changed (just so it isn't constantly re-writing)
 if(prevPotValDC != potValDC){
   analogWrite(motorPin, potValDC);
   // Serial.println(potValDC);
 }
```

Figure 11: Code to control the final DC motor circuit.

This circuit has an advantage over the previous circuit because the h-bridge isolates the DC motor from the Arduino, which reduces the noise that it sends back into the system by a significant amount. However, without a capacitor across the Arduino +5V and GND buses, the noise was still large enough to mess with the rest of the circuitry of the project. This noise is shown in an oscilloscope capture in Figure 12.

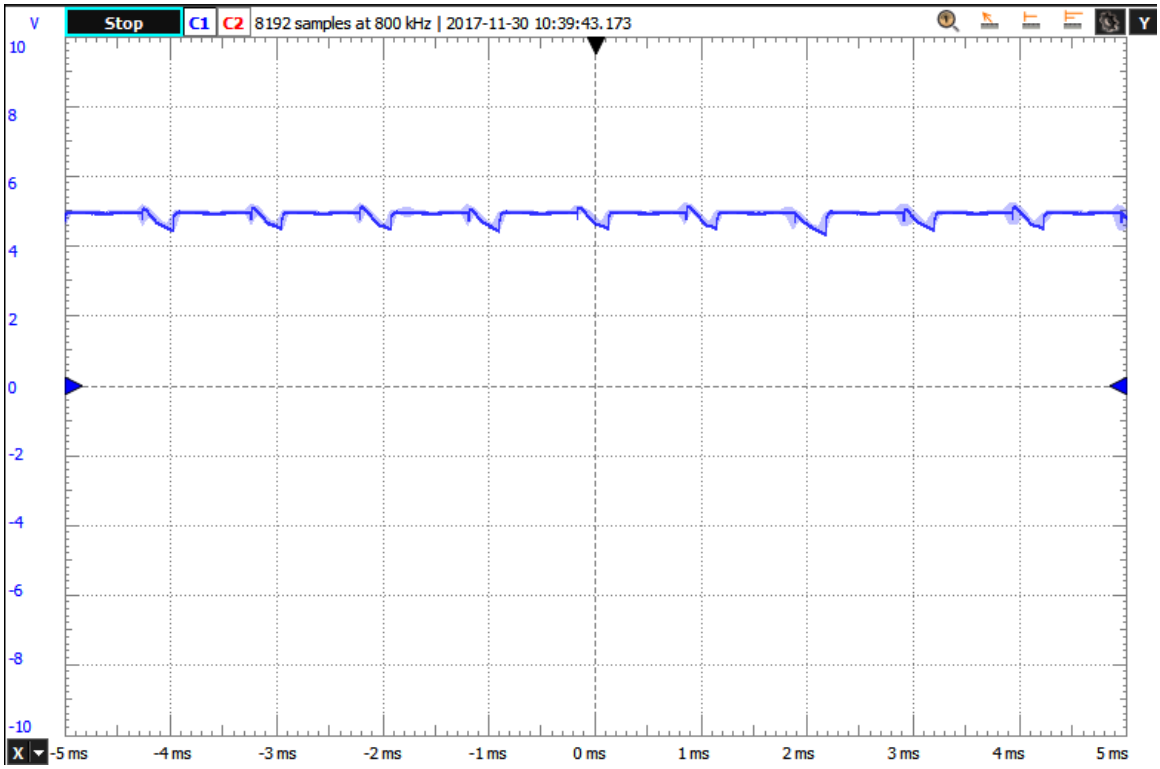

Figure 12: Noise measured across Arduino +5V to GND caused by the DC motor after the switch to using an h-bridge to for motor control. Measurements made using the Analog Discovery 2.

With the addition of a 1000 μF capacitor across the Arduino +5V and GND buses, the noise caused by the motor was finally reduced to a low enough level to not impact any of the other circuits controlled by the Arduino. An image of this noise can be seen in Figure 13.

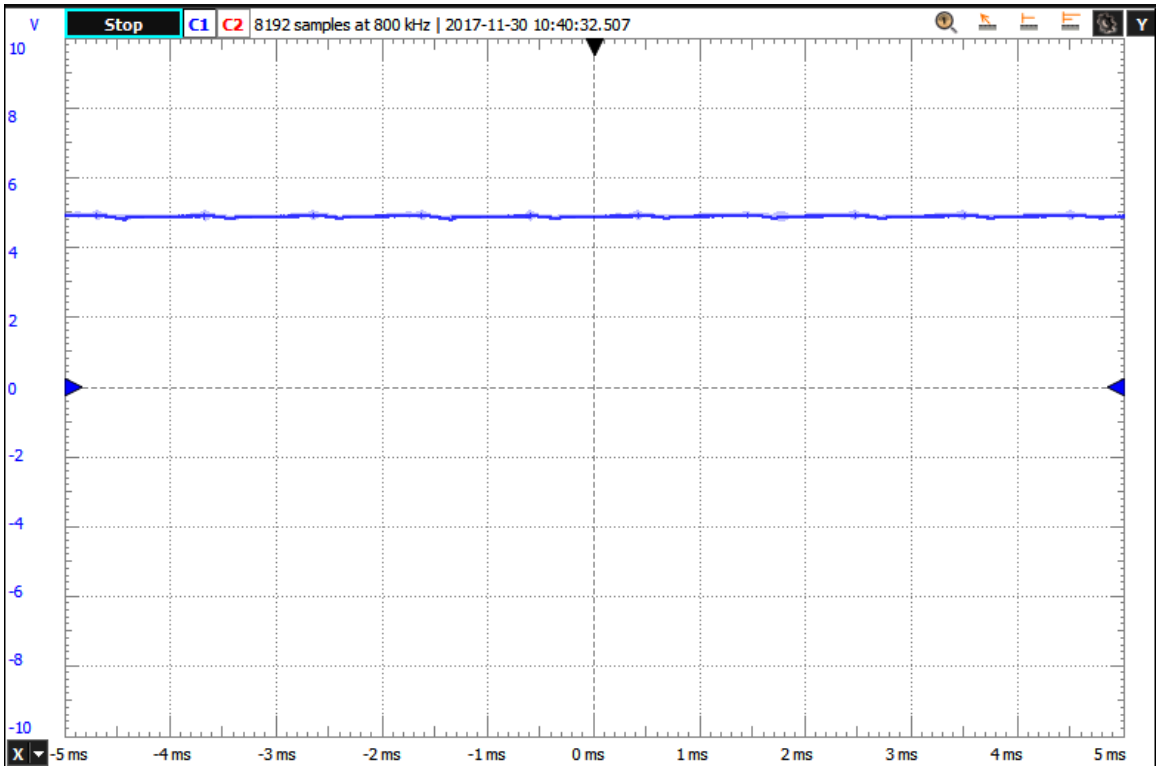

Figure 13: Noise measured across Arduino +5V to GND caused by the DC motor after the switch to using an h-bridge to for motor control, and the addition of a 1000 μF capacitor across the power and ground buses. Measurements made using the Analog Discovery 2.

This was the final DC control circuit used in the project, and it is how the wheel speed is able to be controlled by the user, using the position of the potentiometer.

3.4 Subsystem D: Magnetic Field Sensor & RPM Calculator

At this point, I have described a way for the user to set the position of a potentiometer, and a way to transform that set value to a motor speed. Now, we need a way to measure the speed (in RPM) of the wheel. To do this, I used a magnet attached to the rim of the wheel, which passes by a reed switch once per revolution. A reed switch is a switch that closes when put in close proximity to a magnetic field.

To determine when the magnet made a pass by the reed switch, I created a basic voltage divider out of the switch and a 330 Ω resistor, as shown in Figure 14. When the magnet is near by, the switch will close, and the Arduino pin DIO8 will read high. Otherwise, DIO8 will read low.

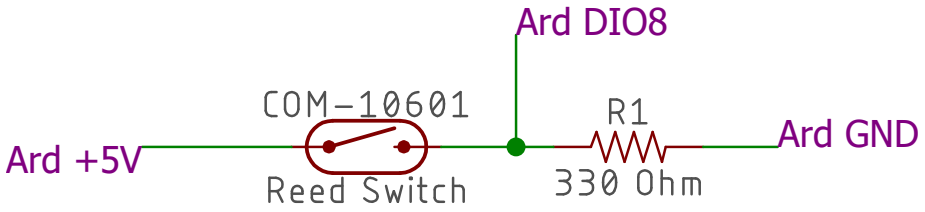

Figure 14: Circuit diagram for the reed switch circuit.

Figure 15 shows the code that I used to calculate RPM based on reading the values from the Arduino DIO8. This is the most complicated code in the entire project, as it requires keeping track of time, and also some processes to only count each pass of the magnet by the reed switch once.

```
// Read the magnet & calcuate RPM
readMag();

// Read Magnet
void readMag(){
  // ***Read the magnet and calculate the rmp***
  // See if the magnet is present
  magVal = digitalRead(magPin);
  // If the magnet is present, and it has been more than .25 seconds since the last
  // detection, to not double count, then calculate the RPM
  if ((magVal == 1) && ((millis() - t) > 250)) {

    // Find the time since the last magnet pass
    tprev = t;
    t = millis();
    deltat = t-tprev;
    // Calculate that time in minutes
    tmin = deltat*1.66667e-5;
    // Calculate the RPM from that time difference
    RPM = 1/tmin;
  }
  // Delay for a quick moment to give the system time to do things like display
  delay(5);
  return;
}
```

Figure 15: Code to read and process data from the reed switch.

The code works in the following way; First, it is checked if the magnet is in the proximity of the reed switch. If it is, *and* there has been over 250 ms since the last detection of such an event, then the RPM is calculated. This 250 ms condition comes into play so that the magnet being in front of the reed switch for multiple readings on the same passing doesn't interfere with the calculations, and only one pass of the magnet is counted per revolution. To calculate the RPM, when a magnet pass is detected, the time since the previous pass is

found, converted to minutes, and then inverted. This gives the current speed of the wheel in RPM. After being calculated, this value is then stored in a global variable, which is eventually passed into the LCD display.

For an example of how the magnet passing by the reed switch can cause some problems with double counting, Figure 16 below shows a logic analyzer of the magnet coming in and out of the switch's proximity. It is evident from the figure that the value of DIO8, shown by the blue line, stays high for longer than one reading, which occur once every 5 ms or so. This reading rate is represented by the red signal, where one reading occurs on each rising edge.

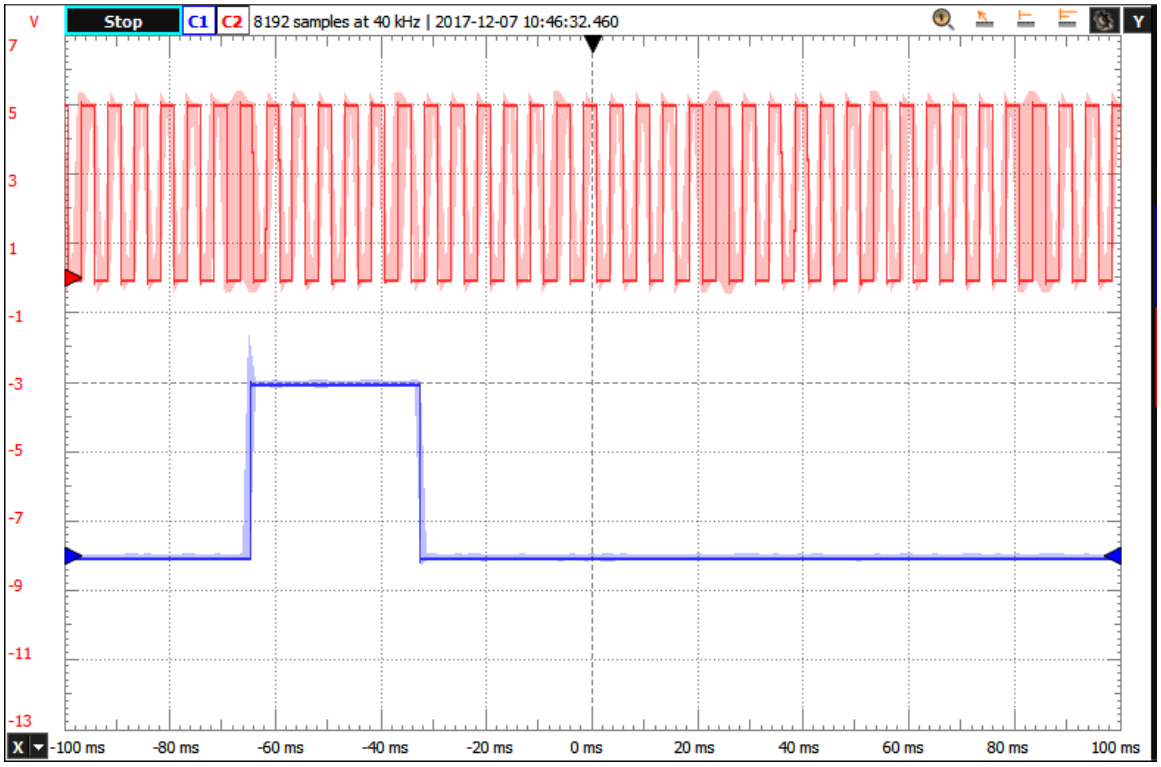

Figure 16: Measurement to show many readings the reed switch stays closed for on each pass of the magnet. It is evident that it stays closed for multiple readings. This is a problem that needed to be addressed in the code.

Up to this point in the project description, we have the calculated value for speed of the wheel in RPM, along with the RPM value set by the user, which was transfered into a speed for the DC motor turning the wheel. All that is left is to display to the user both the measured and the set RPM for the wheel.

3.5 Subsystem E: LCD Display

The LCD display is a circuit that required a lot of wiring, but was quite simple to code. The connections for all of the wires to the display used is shown in Figure 17.

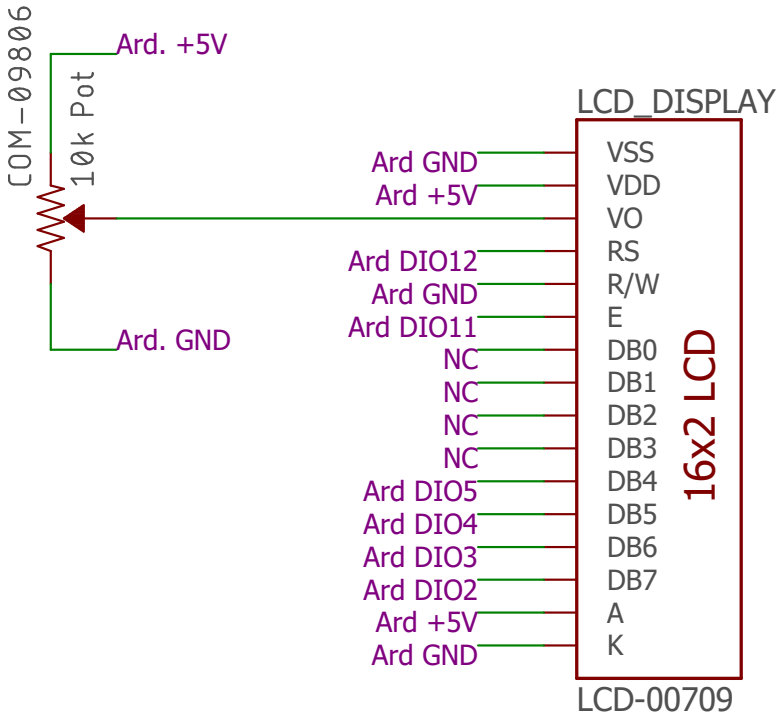

Figure 17: Circuit diagram for the LCD display control.

The potentiometer seen in Figure 17 controls the contrast of the LCD screen, and needs to be adjusted in order to get a contrast that can be read on the screen. All of these connections were hooked up according to the SparkFun SIK Experimentation Guide [3].

The code to control the LCD screen can be seen in Figure 18. This code requires the inclusion of a special library which helps to make talking to the display much easier. This is what the #include<LiquidCrystal.h> is for at the beginning of the code. Figure 18 shows the whole setup process required for the LCD, and also shows the commands required to display things to the screen.

One tricky part of this code occurs when the screen switches from displaying a value with three digits to a value with two digits. If two digit value were just written on top of the three digit one, the third digit from the previous value would stay on the display, and the value shown would be incorrect. To fix this, whenever a two digit value is displayed, the third digit is cleared, and whenever a one digit value is displayed, the second and third digit are cleared before displaying the new value. This is what the majority of the last code block in Figure 18 is for.

In order to display to the screen, it is first necessary to tell the Arduino the position of the cursor, and then tell it the object to display. This is shown multiple times in Figure 18.

```
//Include Libraries
#include <LiquidCrystal.h>

// Set up LCD Display
LiquidCrystal lcd(12,11,5,4,3,2);

 // Set up LCD Display
 lcd.begin(16, 2);
 lcd.clear();
 lcd.print("RPM Setting: ");
 lcd.setCursor(0,1);
 lcd.print("Current RPM: ");

 // Start serial communication
 Serial.begin(9600);

 // Display the desired RPM and the calculated current RPM
 disp(potValDC,RPM);

 // Display Stuff
 void disp(int toShow1, int toShow2){
   // If statements to clear the output if you go from a 3 digit
   // number to 2 digit (or 2 to 1).
   if(toShow1 < 100){
     lcd.setCursor(15,0);
     lcd.print(" ");
   }
   if (toShow2 < 100){
     lcd.setCursor(15,1);
     lcd.print(" ");
   }
   if(toShow1 < 10){
     lcd.setCursor(14,0);
     lcd.print(" ");
   }
   if (toShow2 < 10){
     lcd.setCursor(14,1);
     lcd.print(" ");
   }
   // Print the passed in values in the correct positions
   lcd.setCursor(13,0);
   lcd.print(toShow1);
   lcd.setCursor(13,1);
   lcd.print(toShow2);
   return;
 }
```

Figure 18: Code to control the LCD display.

The LCD screen takes in the value for the set RPM, from the potentiometer reading, and also takes in the measured RPM value, from the reed switch system, and displays these values to the user. This way, the user can tell if the wheel speed is lagging behind the set RPM. Figure 19 shows an image of what the LCD display looks like in action.

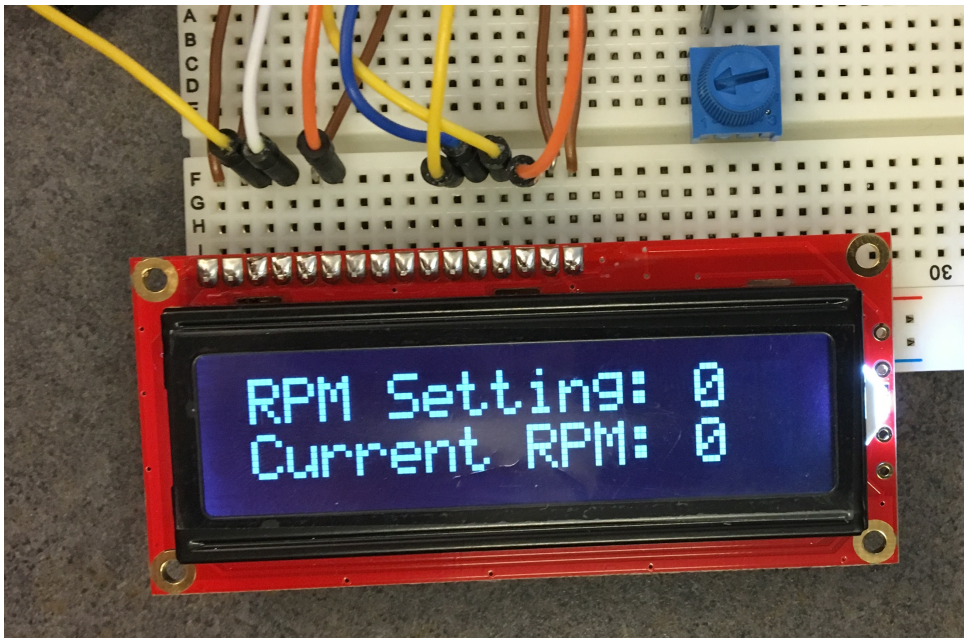

Figure 19: Picture showing what the LCD output looks like to the user. Also seen in the figure is the potentiometer used to control the contrast setting of the display.

The report has now covered each and every system that is needed for the full functionality of the wheel speed monitor.

4 Conclusions

Over the course of this lab, I learned how to use the different functionalities of the Arduino micro controller, including digital IO, analog input, and simulated analog output using PWM. DIO was used in the reed switch system. Analog input was used to read the value of the potentiometer. PWM was used to control the speed of the DC motor. In the end, I created a project which connected together separate circuits built around the Arduino into one functioning product.

The biggest problem that I ran into over the course of this design was that the DC motor produced a very large amount of noise in the circuit, which rendered the rest of the circuits unusable. Originally I attempted to fix this using only a capacitor, but this was not enough. In the end, separating the motor from the Arduino with an h-bridge and using a capacitor was enough to reduce the noise down. In the future, I will know to be wary hooking DC motors directly up to any sort of power rail, as I know how much noise they produce.

References

[1] Amazon, *Bicycle Speedometer and Odometer Wireless Waterproof Cycle Bike Computer with LCD Display & Multi-Functions by YS* (2017). [Online]. Available: `https://www.amazon.com/Speedometer-Odometer-Wireless-Waterproof-Multi-Functions/dp/B01HL0B5AU` [Accessed 23 Nov. 2017].

[2] Arduino, *Arduino Uno Rev 3 Tech Specs* (2017). [Online]. Available: `https://store.arduino.cc/usa/arduino-uno-rev3` [Accessed 5 Dec. 2017].

[3] SparkFun Electronics, *The SIK Guide for the SparkFun Inventor's Kit for the SparkFun RedBoard*. Version 3.0. pp. 64-67, 76-79. San Fransisco, CA: Creative Commons.

A Full Arduino Code

```
//Include Libraries
#include <Servo.h>
#include <LiquidCrystal.h>

// Declare Global Variables
const int potPin = 0;
//int potValServo = 0;
int potValDC = 0;
int prevPotValDC = 0;
//const int servoPin = 1;
const int magPin = 8;
const int motorPin = 9;
int magVal;
int t = 0;
int tprev = 0;
int deltat = 0;
float rad = 0.052; // In meters
float tmin = 0;
float RPM = 0;

// Motor speed calibration - Motor PMW value that gives about 200
RPM
const int motorMax = 100;

// Set up LCD Display
LiquidCrystal lcd(12,11,5,4,3,2);


void setup() {
  // put your setup code here, to run once:

  // Set up magnet input as a digital input
  pinMode(magPin, INPUT);

  // Set up DC Motor pin as an output
  pinMode(motorPin, OUTPUT);
```

```
  // Set up LCD Display
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("RPM Setting: ");
  lcd.setCursor(0,1);
  lcd.print("Current RPM: ");

  // Start serial communication
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:

  // Read the potentiometer to determine desired RPM
  readPot();

  // Write the value 0 - maxMotor (calibrated by the user) to the
DC motor, only
  // if the value from the pot has changed (just so it isn't
constantly re-writing)
  if(prevPotValDC != potValDC){
    analogWrite(motorPin, potValDC);
    // Serial.println(potValDC);
  }

  // Display the desired RPM and the calculated current RPM
  disp(potValDC,RPM);

  // Read the magnet & calcuate RPM
  readMag();
}

// Function Definitions

// Function to read the potentiometer value
void readPot(){
```

```
  // Set the previous pot value to the current pot value
  prevPotValDC = potValDC;
  // Read the analog input pin
  potValDC = analogRead(potPin);
  // Map the value to the correct value to pass to the motor
  potValDC = map(potValDC,0,1023,0,motorMax);
  return;
}

// Read Magnet
void readMag(){
  // Read the magnet and calculate the rmp

  // See if the magnet is present
  magVal = digitalRead(magPin);
  //Serial.println(t);
  //Serial.println(magVal);

  // If the magnet is present, and it has been more than .25
seconds since the last
  // detection, to not double count, then calculate the RPM
  if ((magVal == 1) && ((millis() - t) > 250)) {

    // Find the time since the last magnet pass
    tprev = t;
    t = millis();
    deltat = t-tprev;
    // Calculate that time in minutes
    tmin = deltat*1.66667e-5;
    // Calculate the RPM from that time difference
    RPM = 1/tmin;

    Serial.print(RPM);
    Serial.println(" RPM");
  }
  // Delay for a quick moment to give the system time to do things
like display
```

```
  delay(5);
  return;
}

// Display Stuff
void disp(int toShow1, int toShow2){
  // If statements to clear the output if you go from a 3 digit
  // number to 2 digit (or 2 to 1).
  if(toShow1 < 100){
    lcd.setCursor(15,0);
    lcd.print(" ");
  }
  if (toShow2 < 100){
    lcd.setCursor(15,1);
    lcd.print(" ");
  }
  if(toShow1 < 10){
    lcd.setCursor(14,0);
    lcd.print(" ");
  }
  if (toShow2 < 10){
    lcd.setCursor(14,1);
    lcd.print(" ");
  }
  // Print the passed in values in the correct positions
  lcd.setCursor(13,0);
  lcd.print(toShow1);
  lcd.setCursor(13,1);
  lcd.print(toShow2);
  return;
}
```