Project 2.5.3 Short ISP: Bike Computer

Purpose

The purpose of my Short ISP, which is a bike computer, is to display the speed and distance of a bicycle on seven-segment displays, based on readings from a reed switch on the fork of the bike

Reference

Project description: http://darcy.rsgc.on.ca/ACES/TEI3M/2324/ISPs.html#logs

Project proposal: http://darcy.rsgc.on.ca/ACES/TEI3M/2324/images/RohanShort.png

Theory

A reed switch is a magnetic field-sensing switch. It operates on the same principles as a normal switch (comes in either normally open and normally closed varieties) but instead of being operated by a user, its state depends on the presence of a magnetic field. Inside a normally open reed switch (which is made of glass) are two metal contacts, connected to the only two leads of the component (in most cases). These contacts are constructed from a ferromagnetic material, meaning that they are susceptible to being influenced by a magnetic field. In practice, this means that when a magnetic field is present, the two contacts touch, allowing current to flow between the two leads, unimpeded, just like a switch or pushbutton (see Figure 1).

This property allows them to be used in a variety of applications, including the position of circular objects, including wheels. By placing a magnet at a certain position of the wheel, and a reed switch at a fixed point that the magnet will pass, on every revolution, the reed switch will go high. This means that the generated pulses allow the revolutions per minute (RPM), and total revolutions to be calculated with the addition of an MCU.

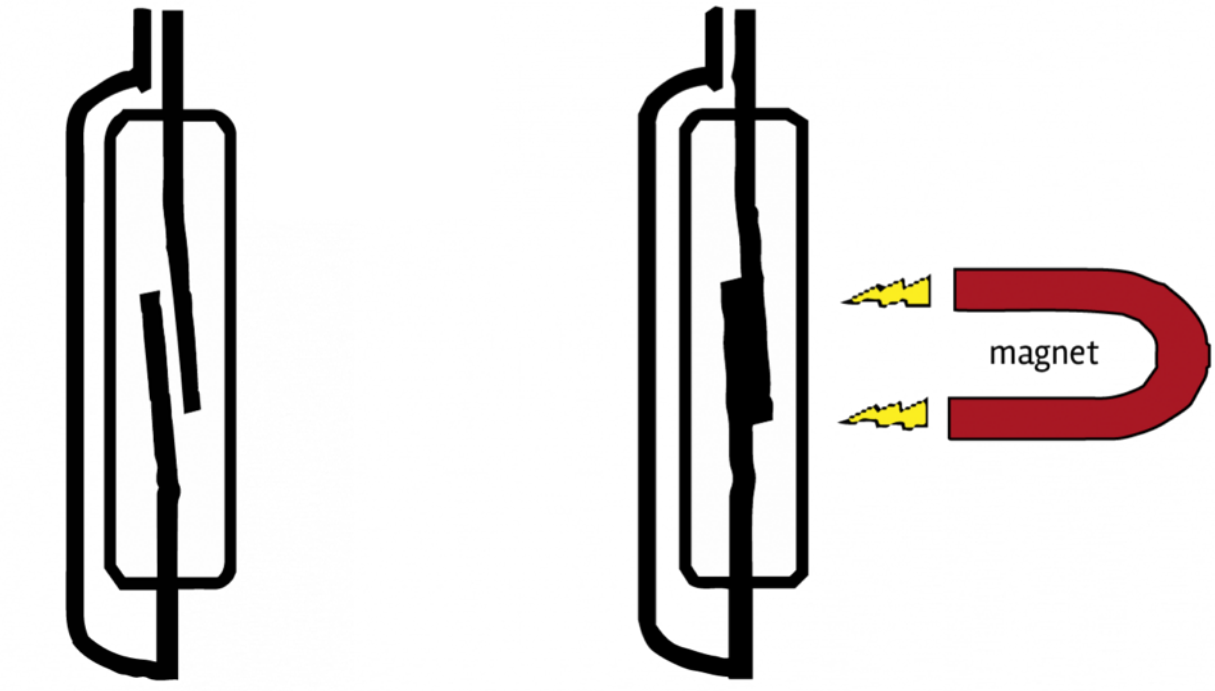


Figure 1. Reed Switch Normally Open with and without Presence of Magnetic Field

Procedure

The bike computer reads out either the speed of the bike, or the distance the bike has travelled since the device has powered on, based on readings from a reed switch (see Theory section). The computer contains a built-in ATmega328P, which takes input from the reed switch, processes it, passes the appropriate binary number to four CD4511 chips. These decode the binary number, into a human-readable output presented on four seven-segment displays (see Figure 1).

The MCU is able to measure the time between the pulses generated by the reed switch, and calculate the speed of the bicycle from its wheel circumference, using the following mathematical relationship: $S = 3.6 \times \frac{C}{\Delta T}$ where S is speed in KPH, C is the circumference of the wheel in mm, and ΔT is the time it takes a revolution to complete in milliseconds. For example, if a revolution took 1 full second to compete, and the wheel had a circumference of 2155 mm, the speed would be: $3.6 \times \frac{2155\ mm}{1000\ ms} \approx 7.8$ km/h. The distance is measured by multiplying the revolutions completed, with the circumference of the wheel, in km.

Parts Table	
Quantity	**Description**
4	Seven-Segment Display
2	SPDT Slide Switch
29	330 Ω Fixed Resistor
1	10 KΩ Fixed Resistor
2	2 x 3 Male Header Pin
2	1 x 16 Long Male Header Pin
4	1 x 8 Female Header Pin
1	L7805 Linear Voltage Regulator
2	0.1 µF Ceramic Capacitor
4	CMOS CD4511 IC
1	ATmega328P DIP MCU
1	16 MHz Crystal Oscillator
2	22 pF Ceramic Capacitor
2	Terminal Block
6	16-Pin DIP Chipseats
1	9 V Battery
2	Perfboard 25 x 24
1	DC-In Barrel Jack
1	Battery Snap
~	Wires
~	Solder

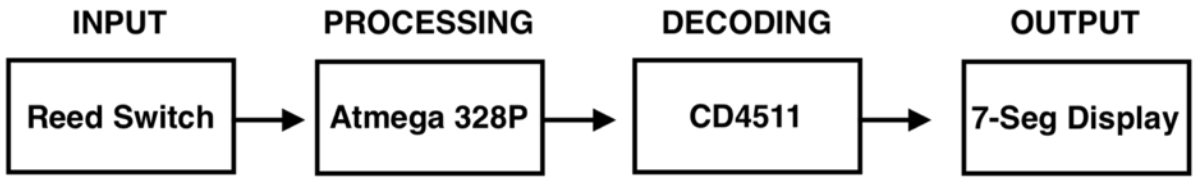


Figure 1. Inputs and Outputs

The device's I/O pins are occupied by a reed switch for input from the bicycle, a user-operated SPDT slide switch to toggle between speed and distance, and 16 pins are occupied by CD4511 chips. The ATmega328P outputs its final number as 4 BCD digits, with the total value multiplied by 10. This is because the second rightmost digit has its decimal place permanently active, so the software must always correct to 1 decimal place. Due to the large footprint of the device and the low amount of space on bike handlebars, the CD4511s, ATmega328P with its supporting hardware, and the 9 V battery are integrated inside a 3D-printed case, connected via header pins to a separate board which contains only the human interface devices (HIDs).

The ATmega328P is the same MCU used in many Arduino microcontrollers, including the Nano and Uno series. When the standard DIP package is used without the addition of the Arduino board, there are a few supporting components that need to be added.

One of the most crucial components is the 16 MHz crystal, with its dual 22 pF "shoulder" capacitors (see Figure 2). By connecting it to pins 9 and 10 (see Figure 3), with the two capacitors between each lead and ground, it produces a square wave at 16 MHz, with a tolerance of ±0.003%, compared to the built-in 8 MHz clock signal's tolerance of ±10%, allowing for increased timing precision.

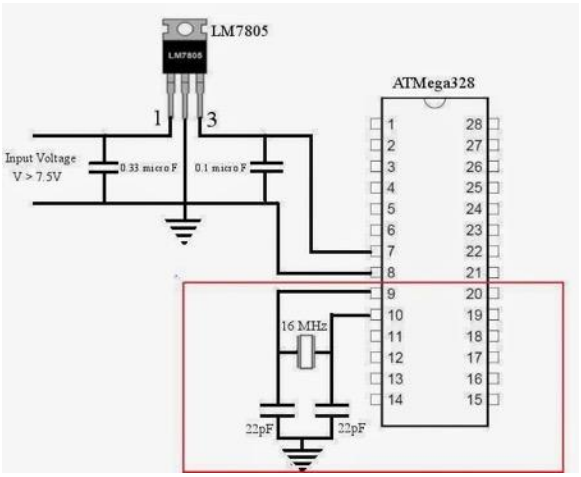


Figure 2. ATmega328P Pinout

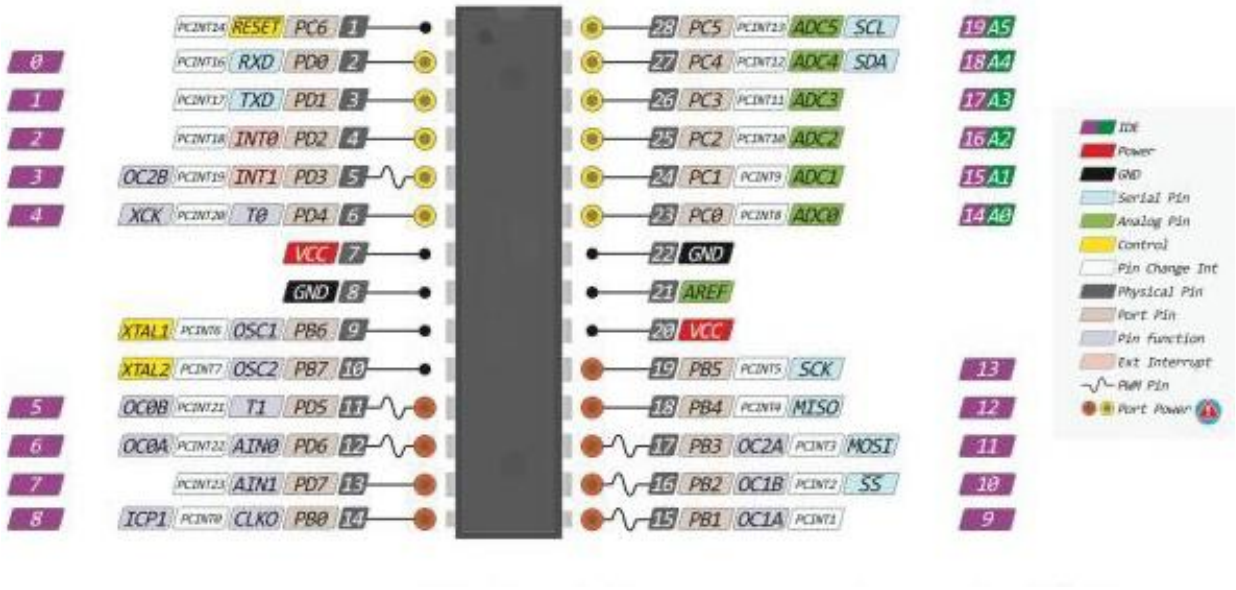


Figure 3. ATmega328P Pinout with Arduino Functions

Figure 2 depicts the other additional component when using an offboard ATmega328P: a voltage regulator. The Arduino contains an onboard voltage regulator, which steps the voltage down to 5 V to prevent damage to the integrated ATmega328P. The output from the voltage regulator is connected to pin 7 of the MCU, and the common ground signal is connected to pin 8 (see Figure 3).

The bike speedometer uses an LM7805 voltage regulator (see Figure 2). This type of regulator is called a *linear voltage regulator*. A linear voltage regulator is a device that maintains a steady output voltage. While they are extremely cheap and versatile, one of their major downsides is their inefficiency. Since they use a negative feedback loop combined with a metal-oxide semiconductor field effect transistor (MOSFET), they are extremely inefficient, effectively dissipating excess voltage as heat. The LM7805 is capable of converting any voltage in excess of 7.5 V to a constant 5 V. This allows the device to be powered by a 9 V battery, despite the fact that it operates at 5 V.

Finally, a 6-pin in-system programmer (ISP) header (see Figure 4) must be added to the circuit in order to program the ATmega328P. On an Arduino board, there is an MCU that converts USB signals to ISP signals. For a standalone ATmega328P, a dedicated in-system programmer must be used. The in-system programmer connects to both power rails, as well as MISO (D12), SCK (D13), reset, and MOSI (D11) (see Figure 3). With the addition of these three components, the standalone ATmega328P has similar functionality to an Arduino board, and can be interfaced with the Arduino IDE.

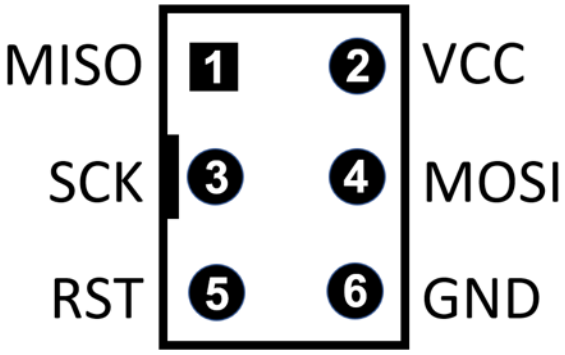


Figure 4. 6-Pin ISP Header Pinout

The code of the bike computer contains 5 variables (see Code section). There are 4 data types used (see Figure 5). A float is used only for kph, the variable that stores the speed of the bike, as a decimal. 64-bit unsigned integers are used for variables that must store the value of `millis()`, a function that returns the running time in milliseconds.

Figure 5. Arduino Data Types

Type	Bytes	Max	Precision
`Float`	4	$\sim 2^{128}$	Decimal
`Uint64_t`	8	2^{64}-1	Integer
`Uint32_t`	4	2^{32}-1	Integer
`Uint16_t`	2	2^{16}-1	Integer

At the start of the `loop()` function is a while loop, used for timing. The part of the code that doesn't interface with the seven-segment displays is written in the loop. The condition of the loop is that `millis()` is less than `tLoop + 1000`, where `tLoop` was set to `millis()` at the start. This causes the loop to run until more than a second has elapsed, at which point it will execute the code below, which is the reading out of the speed or distance (unless the speed is 0, in which case it updates right away).

Inside the timing loop, the program waits for the magnet to pass the reed switch (see Figure 6), then starts a timer by setting `sTime` to `millis()`. When the wheel comes around again, the program adds one to the total revolutions, and calculates the speed with the equation $S = 3.6 \times \frac{C}{\Delta T}$ where S is speed in KPH, C is the circumference of the wheel in mm, and ΔT is the difference between `sTime` and `millis()`; if, however, the wheel took more than 2 seconds to complete a revolution, the speed is automatically overridden to 0, since the actual speed is very close to 0, and within the override, the speed is read out; the override overrides both the speed, and the timing loop. If the override does not trip, and the timing loop expires (1 second has gone by), the speed or distance is read out.

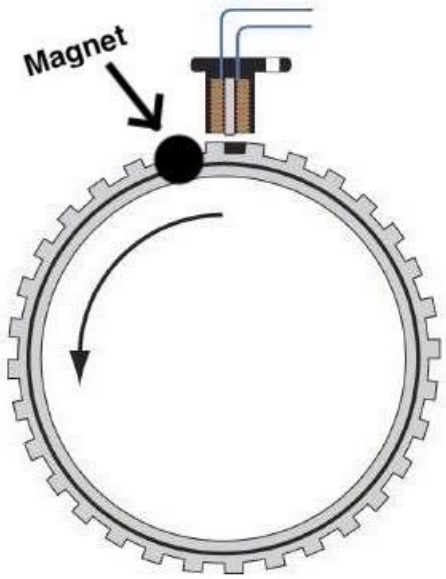


Figure 6. When Rev Timer Starts

There are two additional functions within the code: `lightDisplay()`, and `readOut()`. The latter builds upon the former. Both functions are used to simplify the software interfacing with the seven-segment displays (see Figure 7) via CD4511s.The `lightDisplay()` function has two parameters both of which are 8-bit unsigned integers: digit, which can be from 1-4, and number, which is a single number from 0-9. The purpose of the function is to light the display specified by the digit parameter, with the number specified by the number parameter. For example, `lightDisplay(4, 8)` would show the number 8 on display number 4, which is the leftmost display. This is accomplished by storing the CD4511 pin assignments in a 2D array, and writing to them with 4 `digitalRead()` calls. Whether the bit is written high or low is determined by passing the "number" parameter through the built-in `bitRead()` function with the appropriate bit. For example, to write to bit 3 of digit 2 when the number 7 is passed to `lightDisplay()`, the following line would be used (note that arrays start at 0, not 1):

```
digitalWrite(CD4511_Pins[1][2], bitRead(7, 2));
```

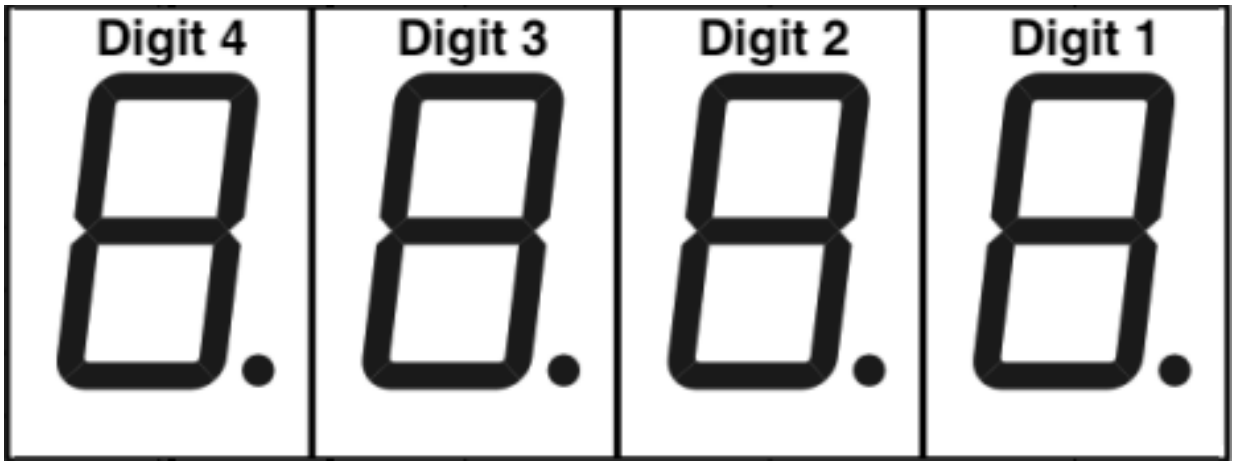


Figure 7. lightDisplay Digit Assignments

The second additional function, `readOut()`, had only one parameter: disp, a 16-bit unsigned integer. The purpose of the `readOut()` function is to display the value of disp (any number between 0 and 9999) on the set of four seven-segment displays (see Figure 7). It works by storing the given number into its 4 digits, each in a variable. It does this using the division and modulo operators. Then, it sets each digit to its respective variable using the `lightDisplay()` function (see Figure 8).

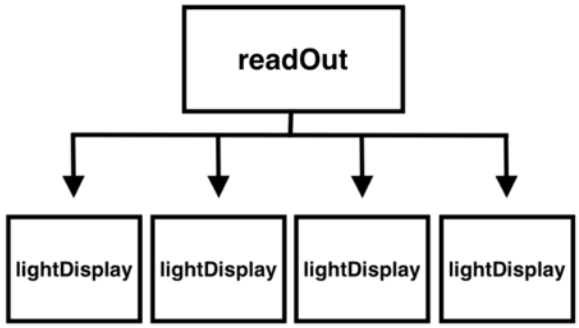


Figure 8. readOut to lightDisplay

The side of the bike computer's 3D printed case contains both a power switch, and a power barrel connector. The barrel connector is actually used as an info jack for the reed switch. The male connector is connected to the leads of the reed switch, while the connector has its wires going to the MCU (see Figure 9); this removably passes the reed switch to the MCU.

Figure 9. Info Jack

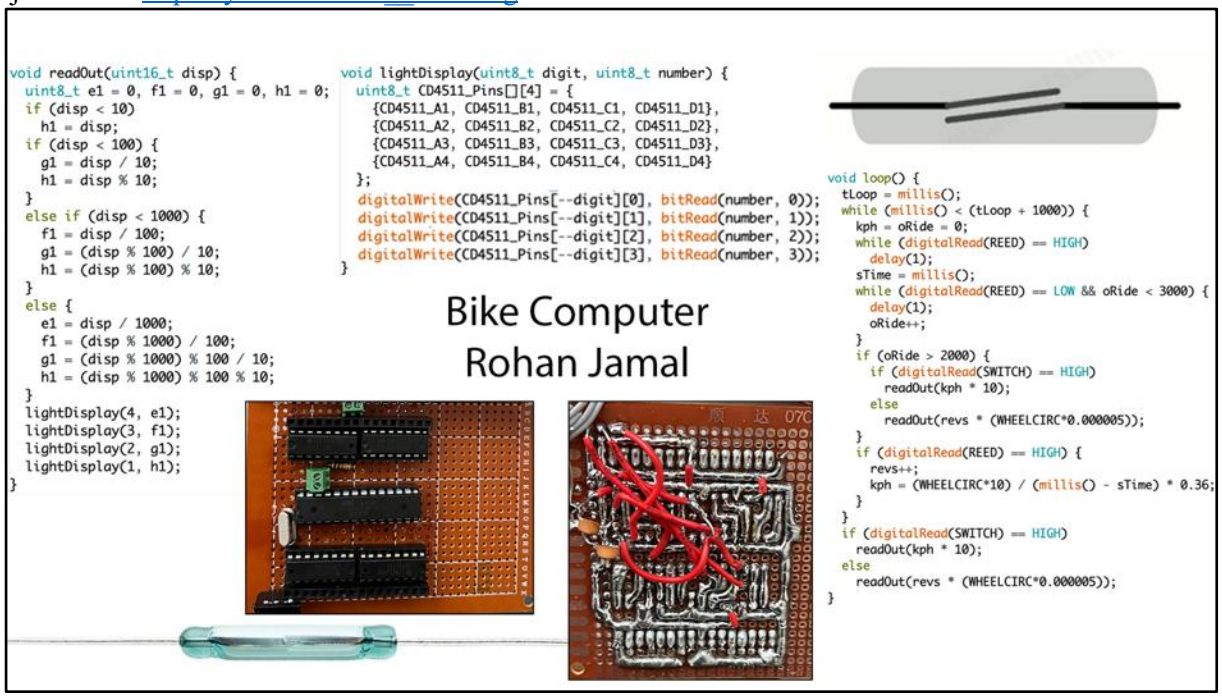
Code

```
// PROJECT   : Bike speedometer
// AUTHOR    : R. Jamal
// PURPOSE   : To display the speed and distance of a bike on seven-segment displays
// COURSE    : ICS3U-E
// DATE      : 05 10 2023
// MCU       : 328P (DIP)
// STATUS    : Working
// REFERENCE : http://darcy.rsgc.on.ca/ACES/ISPs/Hardware.html
// NOTES     : Written in IDE 1.8.19. Uses reed switch for input to the bike
#define WHEELCIRC 2155                //Wheel circumference (in mm)
#define CD4511_A1   5
#define CD4511_B1   8
#define CD4511_C1   7
#define CD4511_D1   6
#define CD4511_A2   9
#define CD4511_B2   13
#define CD4511_C2   12
#define CD4511_D2   10
#define CD4511_A3   1
#define CD4511_B3   11
#define CD4511_C3   4
#define CD4511_D3   2
#define CD4511_A4   14
#define CD4511_B4   19
#define CD4511_C4   18
#define CD4511_D4   15
#define REED        17                //Pin for reed input from bike
#define SWITCH      16                //Pin for distance/speed switch
float kph;                            //Decimal-precision variable for KPH
uint64_t sTime, tLoop;                //Unsigned 64 bit integer: for millis()
uint32_t revs;                        //Unsigned 32 bit integer for revolutions
uint16_t oRide;                       //Unsigned 16 bit integer for override
void setup() {                        //Runs once, when powered on
  for (uint8_t i = 2; i <= 15; i++)   //Declare pins 2-15 as outputs
    pinMode(i, OUTPUT);
  pinMode(18, OUTPUT);                //Declare pin 18 as output
  pinMode(19, OUTPUT);                //Declare pin 19 as output
}
void loop() {                         //Runs on repeat after void setup()
  tLoop = millis();                   //Starts timer between readouts
  while (millis() < (tLoop + 1000)) { //Creates loop to enforce timer
    kph = oRide = 0;                  //Initializes necessary variables
    while (digitalRead(REED) == HIGH) //Wait for REED to go high then low
      delay(1);                       //Delays while REED is high
    sTime = millis();                 //Starts timer between revs
    while (digitalRead(REED) == LOW && oRide < 3000) {
      delay(1);                       //Wait 1 millisecond
      oRide++;                        //Count the ms the switch stays low
    }
    if (oRide > 2000) {               //Code if override has tripped
      if (digitalRead(SWITCH) == HIGH)
        readOut(kph * 10);            //Readout the stored speed, with decimal
      else                            //Reads out distance if in that mode
        readOut(revs * (WHEELCIRC*0.000005));     //Readout revs times circ
    }
    if (digitalRead(REED) == HIGH) {  //Executes when wheel has gone around
      revs++;
      kph = (WHEELCIRC*10) / (millis() - sTime) * 0.36;
    }
  }
  if (digitalRead(SWITCH) == HIGH)    //If in speed mode, readout speed
```

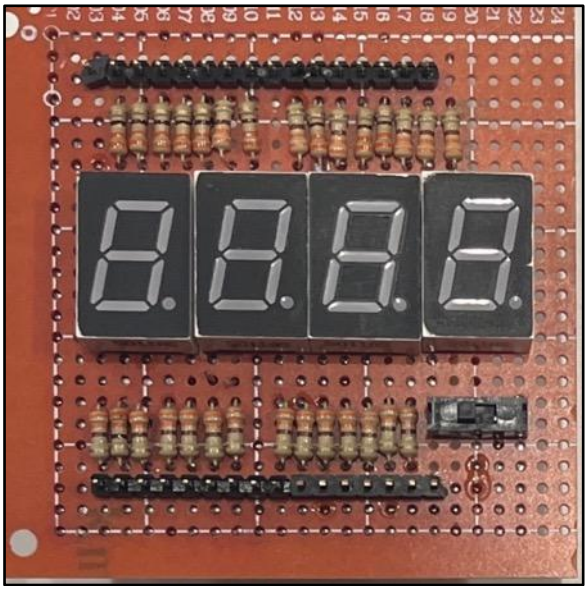
```
    readOut(kph * 10);                //Readout the stored speed, with decimal
  else                                //Reads out distance if in that mode
    readOut(revs * (WHEELCIRC*0.000005));   //Show revs times circ (distance)
}
void lightDisplay(uint8_t digit, uint8_t number) {
  uint8_t CD4511_Pins[][4] = {        //Defines 2D array for 4511 pins
    {CD4511_A1, CD4511_B1, CD4511_C1, CD4511_D1},
    {CD4511_A2, CD4511_B2, CD4511_C2, CD4511_D2},
    {CD4511_A3, CD4511_B3, CD4511_C3, CD4511_D3},
    {CD4511_A4, CD4511_B4, CD4511_C4, CD4511_D4}
  };
  digitalWrite(CD4511_Pins[digit - 1][0], bitRead(number, 0));  //LSB
  digitalWrite(CD4511_Pins[digit - 1][1], bitRead(number, 1));  //2nd LSB
  digitalWrite(CD4511_Pins[digit - 1][2], bitRead(number, 2));  //2nd MSB
  digitalWrite(CD4511_Pins[digit - 1][3], bitRead(number, 3));  //MSB
}
void readOut(uint16_t disp) {         //Reads out number in brackets
  uint8_t e1 = 0, f1 = 0, g1 = 0, h1 = 0;
  if (disp < 10)
    h1 = disp;                        //Gets number < 10 into h1
  if (disp < 100) {                   //Split number < 100 into digits
    g1 = disp / 10;                   //Separates high digit
    h1 = disp % 10;                   //Separates low digit
  }
  else if (disp < 1000) {             //Split number < 1000 into digits
    f1 = disp / 100;                  //Separates high digit
    g1 = (disp % 100) / 10;           //Separates middle digit
    h1 = (disp % 100) % 10;           //Separates low digit
  }
  else {                              //Split number > 1000 into digits
    e1 = disp / 1000;                 //Separates high digit
    f1 = (disp % 1000) / 100;         //Separates third digit
    g1 = (disp % 1000) % 100 / 10;    //Separates second digit
    h1 = (disp % 1000) % 100 % 10;    //Separates low digit
  }
  lightDisplay(4, e1);                //Lights up display with e1
  lightDisplay(3, f1);                //Lights up display with f1
  lightDisplay(2, g1);                //Lights up display with g1
  lightDisplay(1, h1);                //Lights up display with h1
}
```

Media

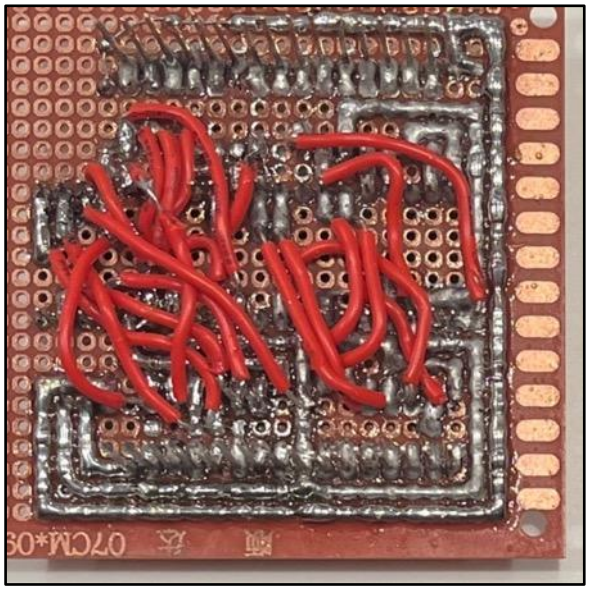
Project video: https://youtu.be/SSE__S8BEXg



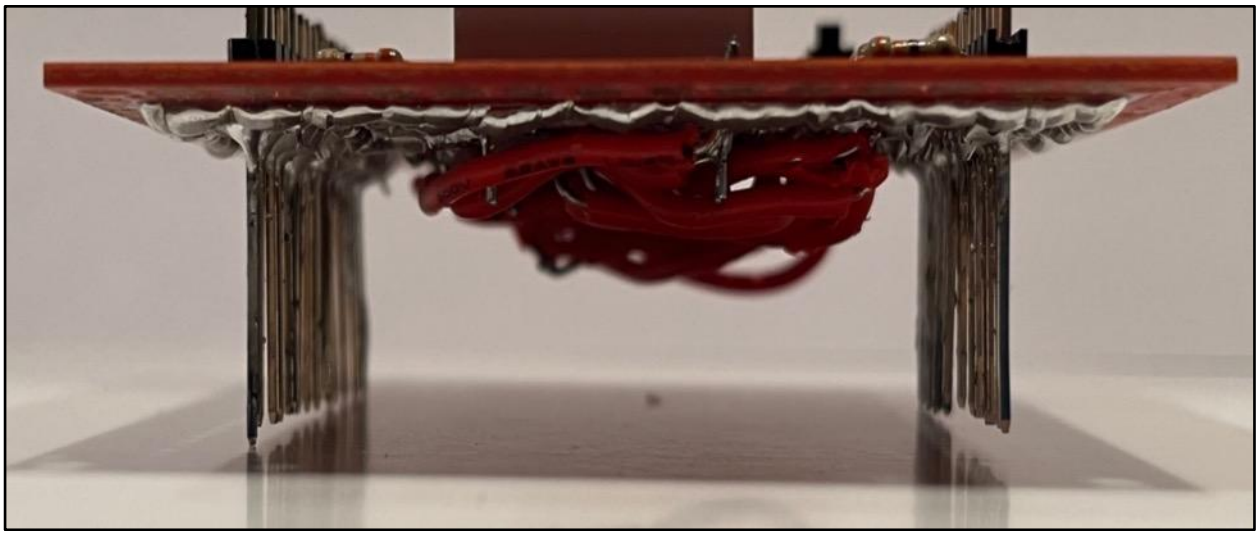
Presentation Background



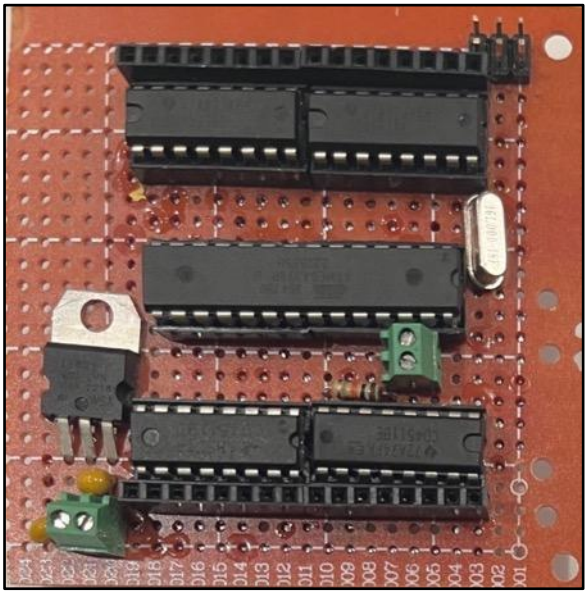
HID Board Top View



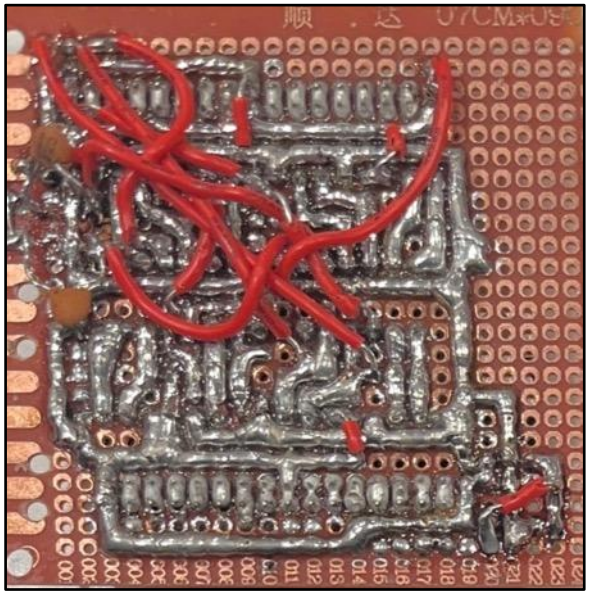
HID Board Bottom View



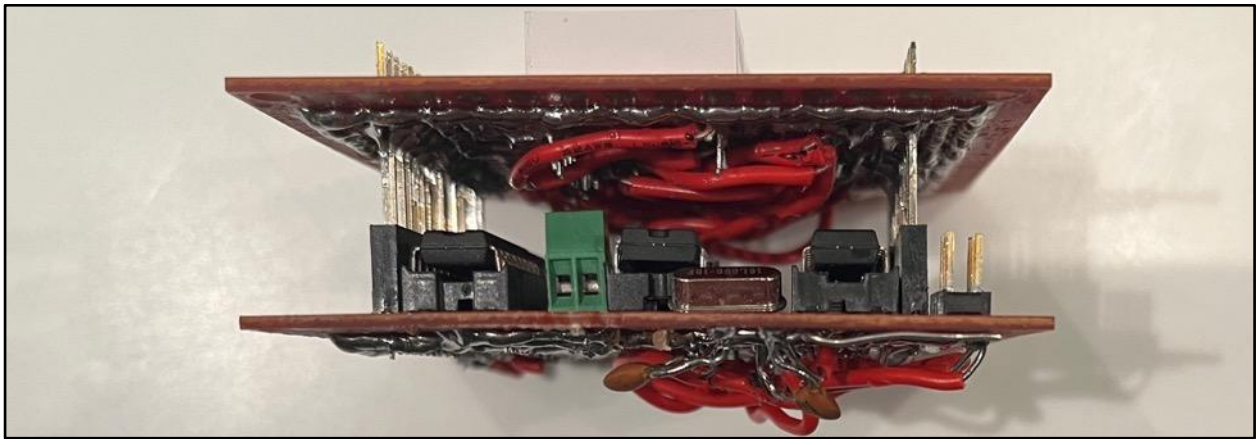
HID Board Side View



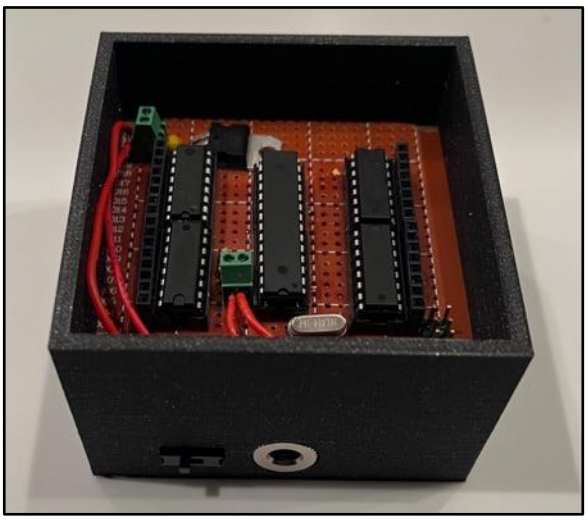
Internals Board Top View



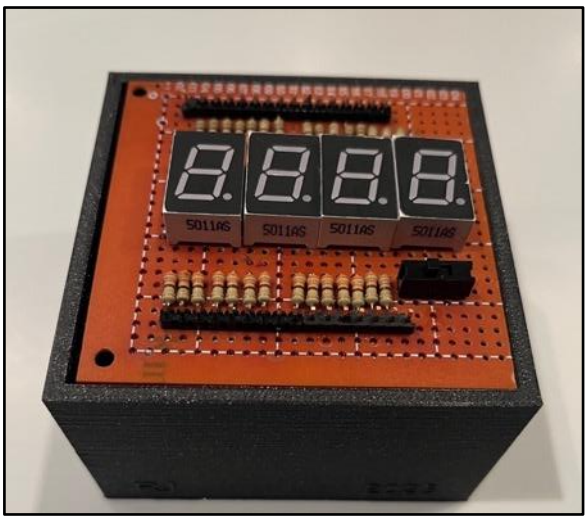
Internals Board Bottom View



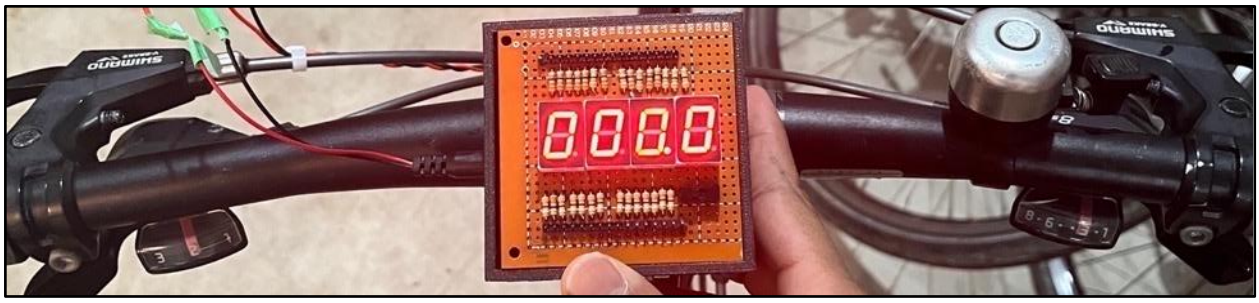
Entire Device Side Profile



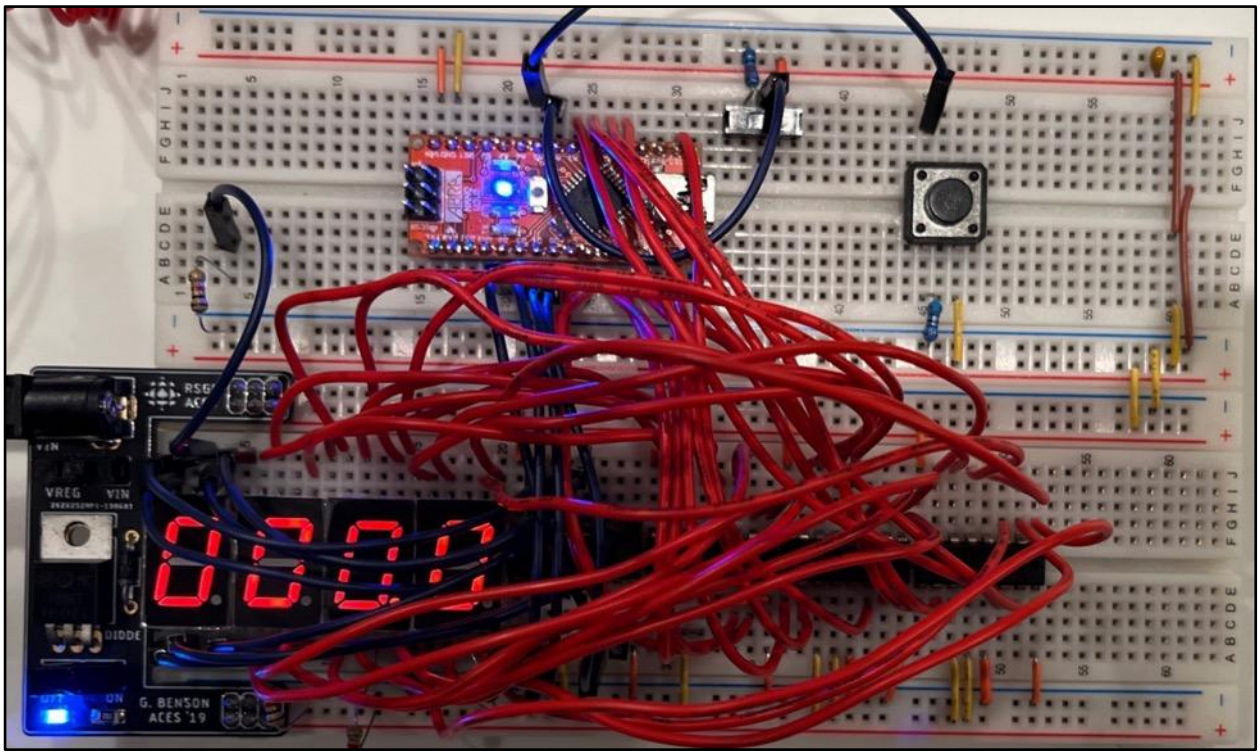
Inside the Case, Power Switch, Info Jack



Fully Assembled Device

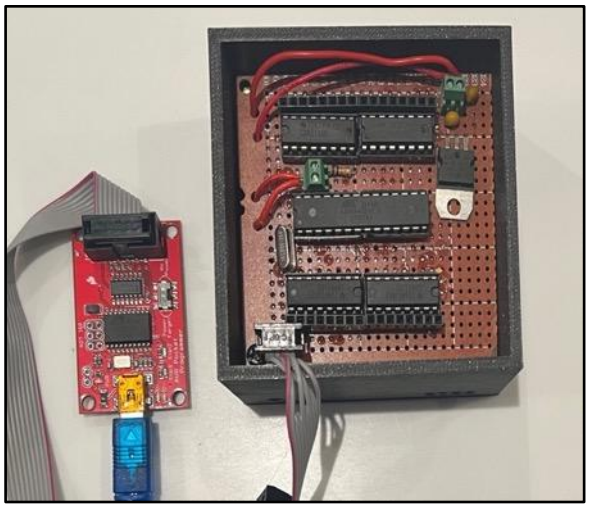


Device Connected to Info Jack on Bicycle



Original Prototype with PBNO in Place of Reed Switch and Arduino Nano

Reed Switch with Magnet



In-System Programmer (ISP) Connected to Header

Reflection

This has definitely been my favourite project this year and last year. This is the first project where I have built something that has a practical function, and I guess it was just really refreshing to have a project in the background for a few months where I could just unplug from everything else and work on it. It's also very satisfying to have a finished project that I made from off-the-shelf components. I think that this project has launched me back into creating things. Before this, I was still creating things, like the required DERs, but it wasn't quite the same. I guess I think there's something to be said about having a project that isn't guaranteed to be possible. Starting with a hair-brained idea, turning it into a concept, and finally a fully working device is so much more interesting to me than simply following instructions to build a device. Don't get me wrong, there is value in both, I just think that every once in a while, it's important to go off the deep end and build something start to finish.

I genuinely think that this might be the first school project I have started more than a week before the due date, at least this year. If I'm being completely honest, I'm kind of surprised that my device even works. I knew that it would all work in theory, but there were just so many variables that could have gone wrong; like one of the million solder joints being compromised, or the last-minute addition of the voltage regulator, or jerry-rigging a power jack to transmit information. I guess an important lesson that I learned is to test each new addition; if you build everything, then power up the device at the end and something doesn't work, it's next to impossible to debug. In addition to starting well in advance, the key to this project's success was to test the circuit after each chip addition, display addition, board addition, or code update; when something went wrong, I knew exactly what part of the circuit to debug, and in most cases, could have the circuit up and running again within the hour.

As for things that did not work at all: about a month ago, when I got my first breadboard prototype working with a reed switch, I tried to change to a solid-state hall effect sensor for improved reliability. Unfortunately, I couldn't get it to work. No matter what I tried, the sensor wouldn't detect anything unless the magnet was almost touching it. I figured that since the reed switch was working, I could just leave it as is for the time being. It's been about a month with the same reed switch and magnet, and so far, it still works extremely accurately, almost never missing a pulse. Overall, this project has probably been the most successful yet, and I hope my next ISP (in the new year) can go as smoothly.