Lab Report: Particle Argon Peripherals

Lue Xiong

March 4, 2020

1 Introduction

The focus of the lab is to gain understanding of circuits and how current flow sthroughout a given breadboard schematic with different electrical components such as sensors and peripherals. The lab also shows how these electrical components might be used in isolation as well as in combination with each other; denoting some sense of real-world application. Secondary to this is how a breadboard interacts with the given electrical components, and how to compose them together.

2 Problem Statement

The lab can be summarized down to five main different parts, which include:

- RGB LED – Utilize Particle Argon's pulse-width modulation capability on pins *D2*, *D3*, and *D4*
- Piezzo Buzzer – Modify the code to play the same sequence as the "two-bits.mp4"
- Switches – Explain the difference between **mom = 0** and **mom = 1** in relation to the momentary switch being depressed as well as the minimum voltage required for **mom = 1**
- Servo – Modify the code to reset when the servo rotation passes 180 degrees, rotating at 20 degrees increment when the momentary switch is depressed. Then publish each incremental reset to an IoT cloud platform
- Analog Temperature Sensor – Convert the ADC values to celsius and fahrenheit values and publish them as separate events to an IoT cloud platform.

3 Process

3.1 Part One: RGB LED

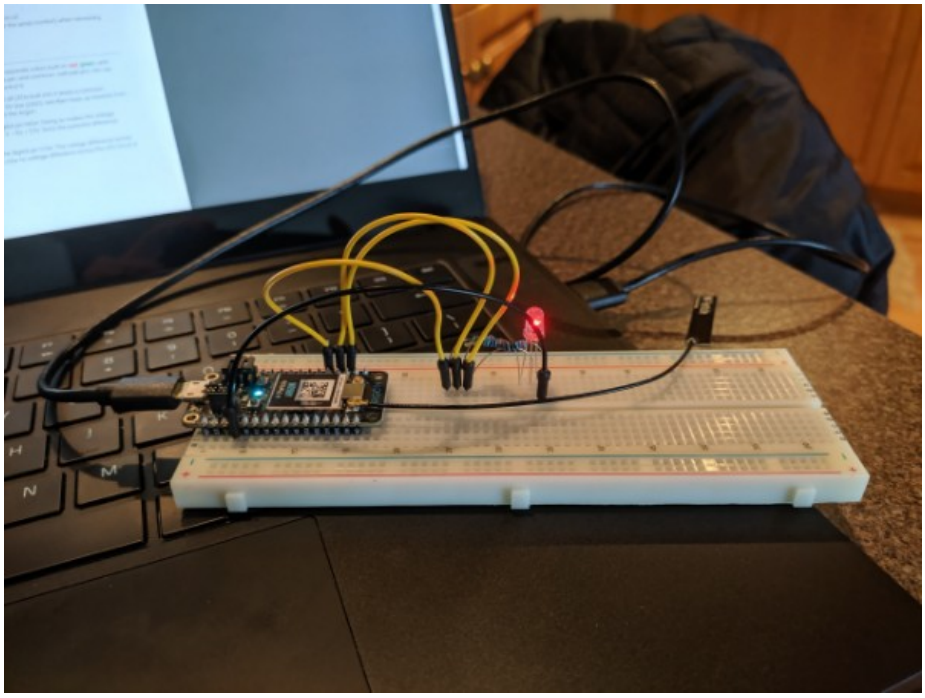

Figure 1: Particle Argon RGB LED

The code below shows how to create the fade effect on the RGB LED by using a function called **fade()**. The function starts with a loop that increments *brightness* by 1 from 0 to 255 with 10 millisecond delays in between. Upon reaching *brightness* value 255, another loop will run and decrement by 1 from 255 to 0 with 10 millisecond delays in between. It is in practice, a rather slow fade but it works nonetheless. Figure **??** is a photo of the components composition on the breadboard.

```
int bPin = D2;
int gPin = D3;
int rPin = D4;

void setup() {
  pinMode(bPin, OUTPUT);
  pinMode(gPin, OUTPUT);
  pinMode(rPin, OUTPUT);
}

void loop() {
  int delay_in_milliseconds = 10;

  // blink(bPin, delay_in_milliseconds);
  // blink(gPin, delay_in_milliseconds);
  // blink(rPin, delay_in_milliseconds);
  fade(bPin, delay_in_milliseconds);
  fade(gPin, delay_in_milliseconds);
  fade(rPin, delay_in_milliseconds);

  delay(1000);
}

void blink(int pin, int delay_in_milliseconds) {
  digitalWrite(pin, HIGH);
  delay(delay_in_milliseconds);
  digitalWrite(pin, LOW);
  delay(delay_in_milliseconds);
}

void fade(int pin, int delay_in_milliseconds) {
  for(int brightness = 0; brightness < 256; brightness++) {
    analogWrite(pin, brightness);
    delay(delay_in_milliseconds);
  }

  for(int brightness = 255; brightness >= 0; brightness--) {
    analogWrite(pin, brightness);
    delay(delay_in_milliseconds);
  }
}
```

3.2 Part Two: Piezzo Buzzer

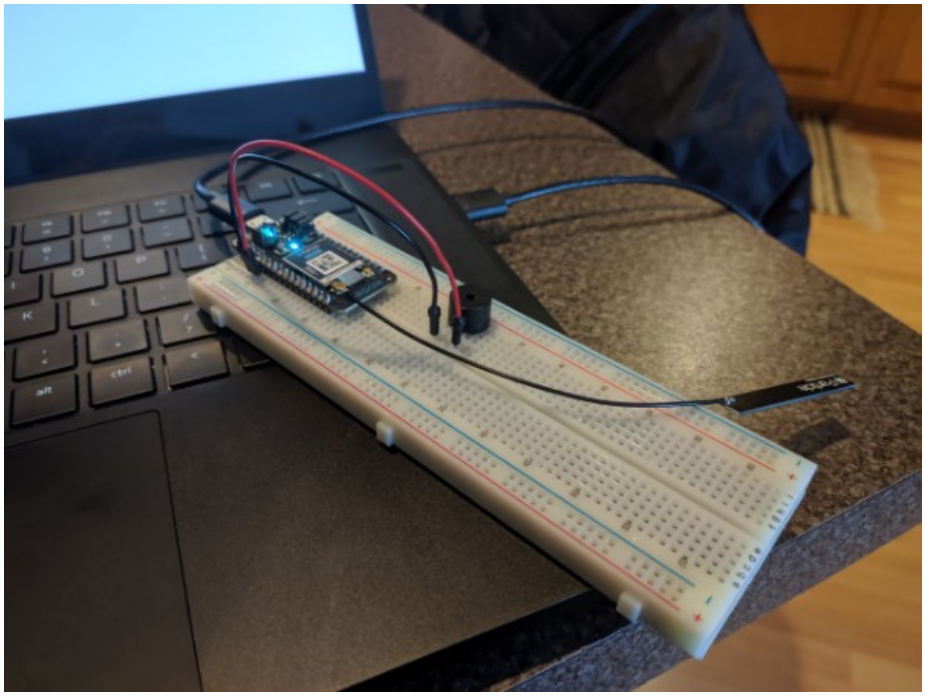

Figure 2: Particle Argon Piezzo Buzzer

The code below shows how to create the two-bits sound on the piezzo buzzer by using a function called **twoBitsBuzz()**. The function sends currents of multiple levels to the piezzo buzzer at arbitrary times in attempt to mimic the "two-bits.mp4" tune. Figure **??** is a photo of the components composition on the breadboard.

```
int buzzPin = A0;

void setup() {
  pinMode(buzzPin, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  Serial.println("Buzzing...");

  twoBitsBuzz(buzzPin);
}

void buzzEffectOne(int buzzPin, int delayInMilliseconds) {
  analogWrite(buzzPin, 128);
  delay(delayInMilliseconds);
  analogWrite(buzzPin, 0);
  delay(delayInMilliseconds);
}

void buzzEffectTwo(int buzzPin, int delayInMilliseconds) {
  for(int i = 0; i < 256; i++) {
    analogWrite(buzzPin, i);
    delay(delayInMilliseconds * 0.2);
    analogWrite(buzzPin, 0);
    delay(delayInMilliseconds * 0.2);
  }
}

void twoBitsBuzz(int buzzPin) {
  analogWrite(buzzPin, 208);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(300);
  analogWrite(buzzPin, 160);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(50);
  analogWrite(buzzPin, 160);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(50);
  analogWrite(buzzPin, 160);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(75);
  analogWrite(buzzPin, 160);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(500);
  analogWrite(buzzPin, 208);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(300);
  analogWrite(buzzPin, 240);
  delay(100);
  analogWrite(buzzPin, 0);
  delay(3000);
}
```

3.3 Part Three: Switches

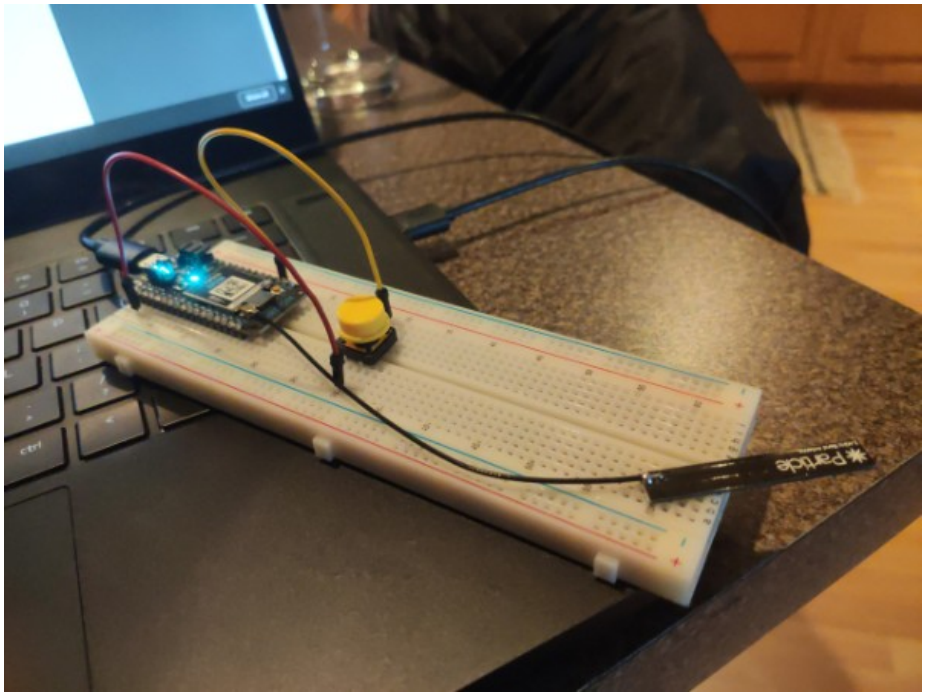

Figure 3: Particle Argon Switches

When the momentary switch is depressed, it engages metal to metal contact which creates a circuit for currents to run through. When it is released from being depressed, the circuit is broken. In this context, assuming the momentary switch is depressed, 3.3 volts will flow from the **3V3** pin to the **D2** pin and keep looping as such. Essentially this is what happens when **mom = 1**.

Parameter	Symbol	Min	Typ	Max	Unit
LiPo Battery Voltage	V_{LiPo}	+3.3		+4.4	V
Supply Input Voltage	V_{3V3}	+3.0	+3.3	+3.6	V
Supply Output Voltage	V_{3V3}		+3.3		V
Operating Current (uC on, Radio ON)	$I_{Li+\ avg}$		8	240	mA
Operating Current (EN pin = LOW)	$I_{disable}$		20	30	uA
Operating Current (uC on, Radio OFF)	$I_{Li+\ avg}$		TBD	TBD	mA
Sleep Current (4.2V LiPo, Radio OFF)	I_{Qs}		TBD	TBD	mA
Deep Sleep Current (4.2V LiPo, Radio OFF)	I_{Qds}		TBD	TBD	uA
Operating Temperature	T_{op}	-20		+60	°C
Humidity Range Non condensing, relative humidity				95	%

Figure 4: Particle Argon Operating Conditions

As shown on Figure **??** – more specifically the **Supply Input Voltage** parameter – the minimum voltage stated is 3.0 volts for inputs like momentary switches. Interestingly enough, the maximum voltage is 3.6 volts, which means that there is a +/- 0.3 volt difference from the typical 3.3 volt.

```
int momPin = D2;

void setup() {
  pinMode(momPin, INPUT_PULLDOWN);
  Serial.begin(115200);
}

void loop() {
  int momState = 0;
  momState = digitalRead(momPin);

  Serial.println("mom:␣");
  Serial.println(momState);

  delay(200);
}
```

The code above produces the outputs on the serial monitor in Figure **??**. The **mom = 0** state means that there is no physical connection between the **3V3** and **D2** pin. The **mom = 1** means that the momentary switch is depressed to create a physical connection between **3V3** and **D2** pin.

Figure 5: Particle Argon Switches

3.4 Part Four: Servo

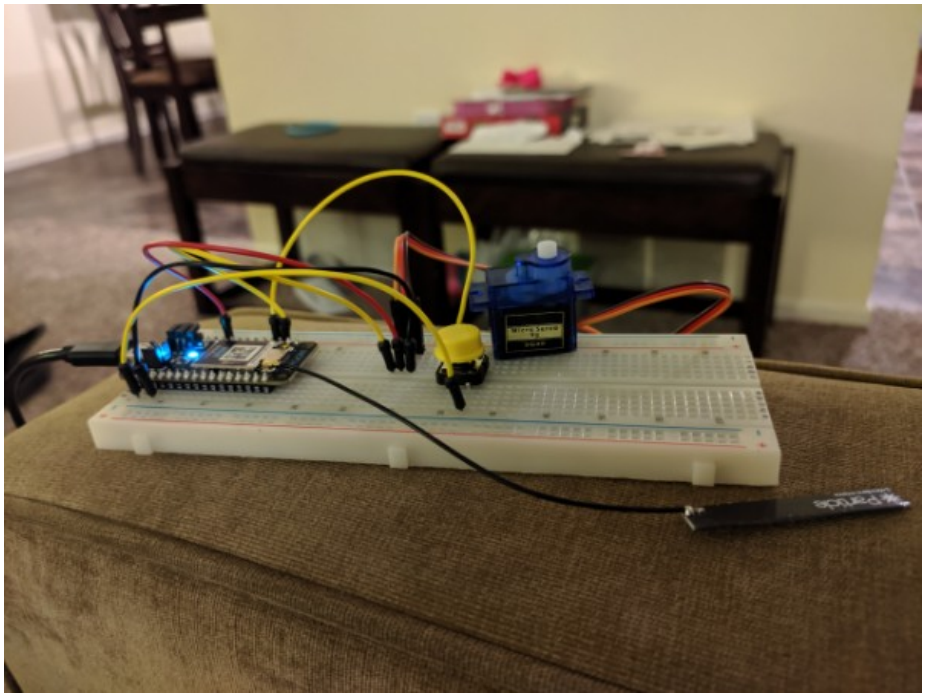

Figure 6: Particle Argon Servo

The code below shows how to rotate the servo from 0 to 181 degrees with 20 degrees rotation per button depression. Upon reaching any value greater than 180, the servo will reset itself to 0 degrees and publishes a reset counter to Particle Cloud. The reset counter increments for every time the servo resets to 0 degrees.

```
Servo servo;

int momPin = D3;
int position = 0;
int resetCounter = 0;

void setup() {
  pinMode(momPin, INPUT_PULLDOWN);

  servo.attach(D2);
  servo.write(0);

  delay(500);

  Serial.begin(115200);
}

void loop() {
  int momState = 0;
  momState = digitalRead(momPin);

  delay(150);

  if(momState == 1) {
    if(position < 181) {
      servo.write(position += 20);
    }

    if(position > 180) {
      servo.write(position -= 180);
      Particle.publish("resetCounter", String(resetCounter += 1));
    }

    Serial.println("Position:␣");
    Serial.println(String(position));
  }
}
```

Figure **??** shows reset counters published to Particle Cloud. Figure **??** is a follow-up with a webhook to capture the value to the Losant IoT platform.

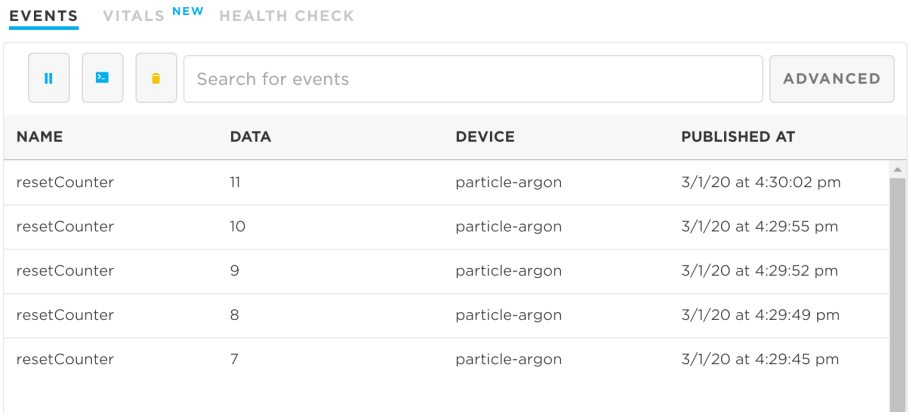

Figure 7: Particle Argon Cloud Events

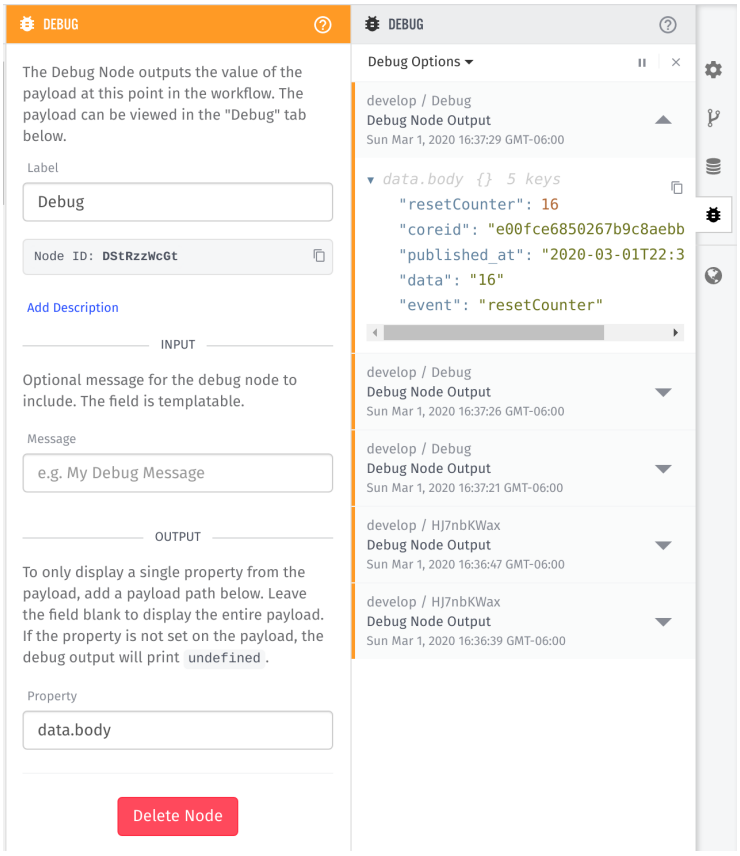

Figure 8: Particle Argon Reset Counter

3.5 Part Five: Thermistor

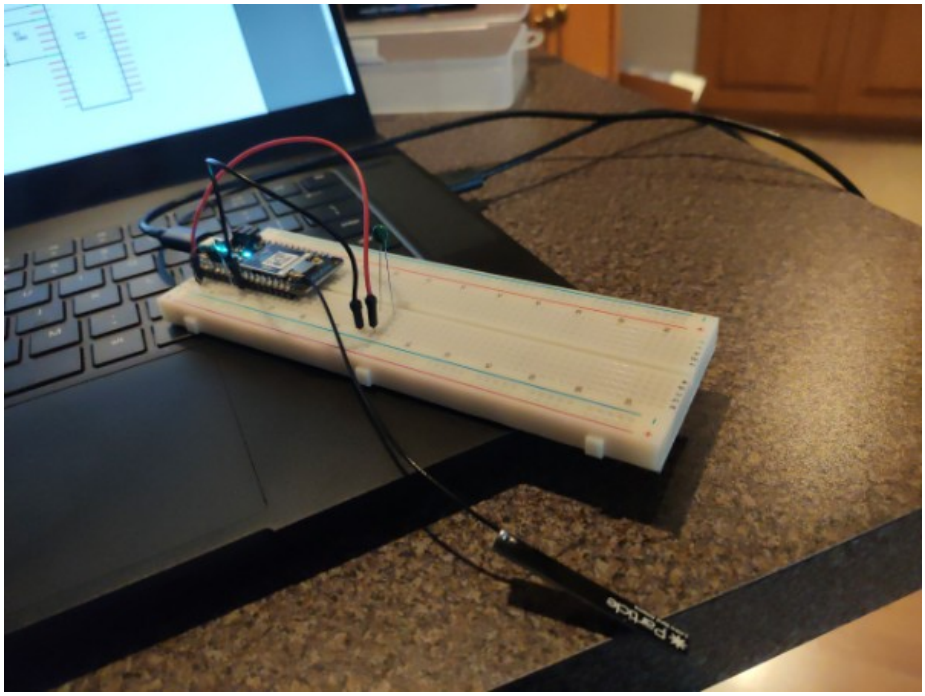

Figure 9: Particle Argon Thermistor

The code below calulates ADC values into celsius and fahrenheit values. The main equation used was the Steinhart and Hart equation for NTC thermistors, which is mathematically represented as such:

$$T = \frac{1}{A + B\ln(R) + C[\ln(R)]^3}$$

T in the equation represents temperature in celsius degrees. In order to convert from celsius to fahrenheit degrees, the following mathematical forumla was used:

$$T_{(^\circ\mathrm{F})} = T_{(^\circ\mathrm{C})} \times 9/5 + 32$$

```
#include <math.h>

int temperaturePin = A4;

void setup() {
  Serial.begin(115200);
}

void loop() {
  int adcValue = analogRead(temperaturePin);
  double output_voltage = ( (adcValue * 3.3) / 4095.0 );
  double thermistor_resistance_kilo_ohms = ( ( 3.3 * ( 10.0 /
      output_voltage ) ) - 10 );
  double thermistor_resistance_ohms =
      thermistor_resistance_kilo_ohms * 1000 ;
  double thermistor_resistance_log = log(
      thermistor_resistance_ohms);
  double kelvin = ( 1 / ( 0.001129148 + ( 0.000234125 *
      thermistor_resistance_log ) +
                  ( 0.0000000876741 * thermistor_resistance_log *
                      thermistor_resistance_log *
                      thermistor_resistance_log ) ) );
  double celsius = kelvin - 273.15;
  double fahrenheit = ( celsius * 9 / 5 ) + 32;

  Serial.println("Temperature␣in␣Celsius:␣");
  Serial.println(String(celsius));

  Serial.println("Temperature␣in␣Fahrenheit:␣");
  Serial.println(String(fahrenheit));

  Particle.publish("Celsius:␣", String(celsius));
  Particle.publish("Fahrenheit:␣", String(fahrenheit));

  delay(1000);
}
```

Both of these formulas are represented in C code. After the calculations are made, they are both published as events shown in Figure **??** and then separated to two different channels to be displayed on graphs. The graphs can be found in Figure **??**.

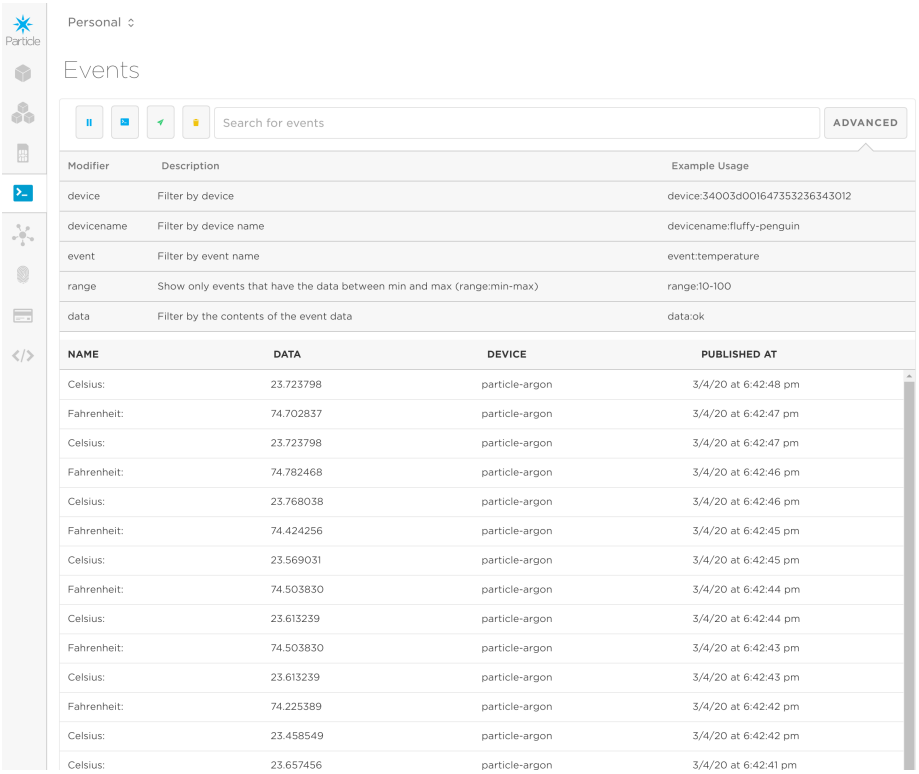

Figure 10: Particle Argon Celsius and Fahrenheit

Figure 11: Particle Argon Celsius and Fahrenheit Graphs

4 Discussion of Results

Part five of the lab concerning temperature were the most difficult because there were many unfamiliar formulas in addition to having to do mathematics in general. The results were however satisfactory as they fit real world numbers that would be measured with either a temperature sensor or thermometer. Part two concerning the piezzo buzzer was the most interesting because all of the buzzes were the same pitch. It was also hard to decipher the time gaps in between each buzz to recreate the "two-bits.mp4" melody. The least interesting was the momentary switch as it was a matter of on or off; though I see can this being very practical to use for everyday convenience, such as a remote control.

5 Conclusion

The lab was a good learning experience being that there were a variety of electrical components to manipulate. Applying how the ground pin relates to the voltage pin with currents circulating to each electrical component also cemented the idea of how eletric works. It is enlightening to experience the practicality and usefulness of the breadboard. Not having to solder metal together and just having things work on the breadboard makes this lab very convenient. I will say however, that my bias favored me torwards the coding portion of the lab, since I am a software engineer at heart.