Precise Weather Rocket

ECE110/120 Fall 2016 Honors Lab Project Report

ECE110 Lucas Cohen (lmcohen2)

ECE120 Zhijie Jin (zhijiej2) / Siye Cen (siyecen2)

I. Introduction

A. Statement of Purpose/Problem Description

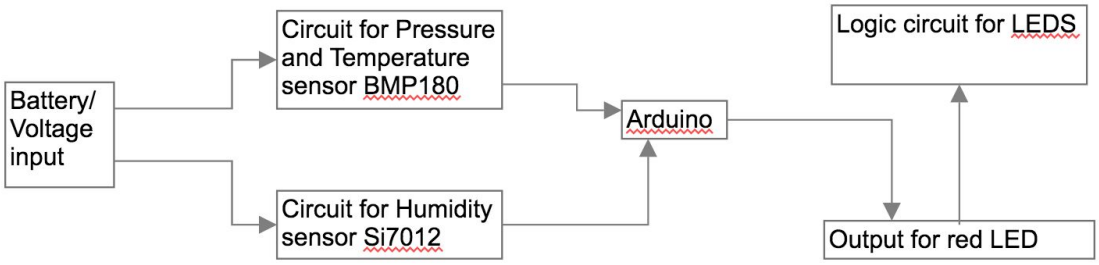
There are many cases in which your phone's weather app and the televised weather report is not so accurate for certain locations. The purpose of this project is to build a small rocket that detects temperature, moisture, air pressure using various sensors to detect possible incoming storms and extreme weather conditions. The rocket will detect conditions much more accurately than normal detectors, and therefore it will be much more localized.

B. Features and benefits

After the system launches, it will detect local weather conditions. Four diodes will light up depending on the conditions, signaling the severity of a storm. This will help detect and track storms and severe weather conditions.

II. Design

A. System Overview



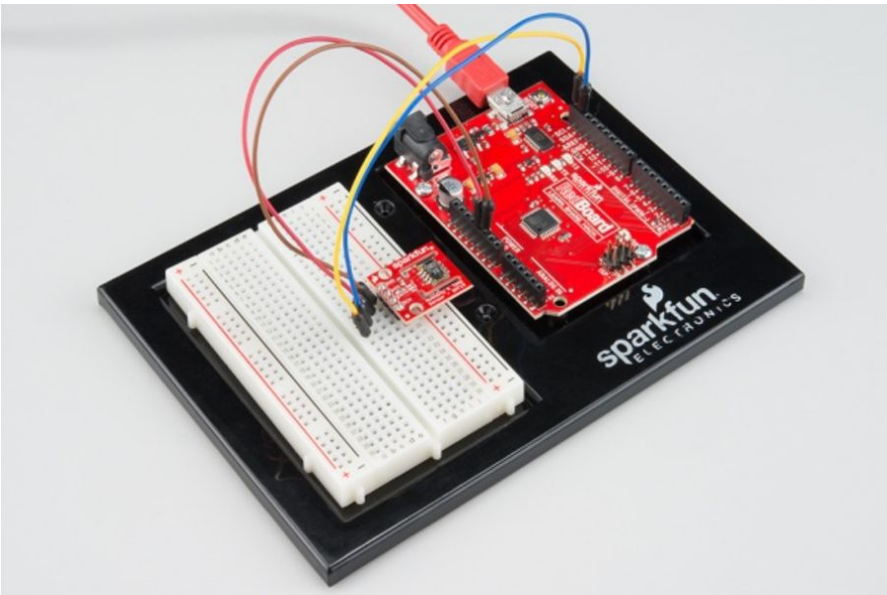
There are two sensors, one that measures humidity of the environment, and the other one measures air pressure and local temperature. These sensors are connected to Arduino using breadboard.

By having the outputs and readings stored in EEPROM in the system, a logic gate circuit will interpret these outputs and tell us whether or not the local weather is more extreme than a certain threshold point. This thresholds are determined by our research on what is typical for severe weather. For the demostration purpose of this course and this lab report, we will set it to be a number we can create in laboratory environment.

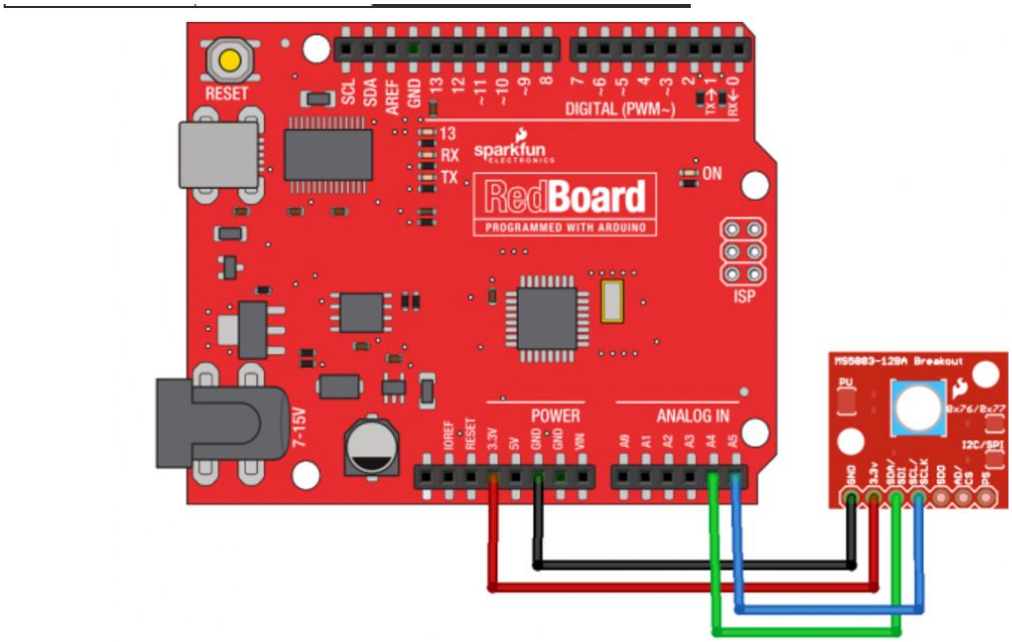
B. Design Details

Each sensor is a Sparkfun sensor, consisting of four ports that need to be connected to our RedBoards. There are also four LEDs connected to digital pins on the RedBoard. Based on the output value from each sensor, a digital pin will either supply power to a LED or it will not. Three of these LEDs (Temperature, Humidity, and Pressure) are connected to a NAND gate that determines whether two of the LEDs have current flowing through them or not. If two of them do have a current flowing through them, the fourth LED will emit light and signal that there is most likely a storm occurring or about to occur.

Temperature/Humidity Sensor Schematic Photo:



Pressure Sensor Schematic:



In the logic part, we have three sensors, giving us three inputs for logic circuit. We want to send out severe weather warning signal when more than one of the data we get is out of threshold value. This comes from our design decision. That is, among temperature, pressure and humidity, when two or more exceeds normal range, we consider it as severe weather. Considering this, we create following truth table and K-map.

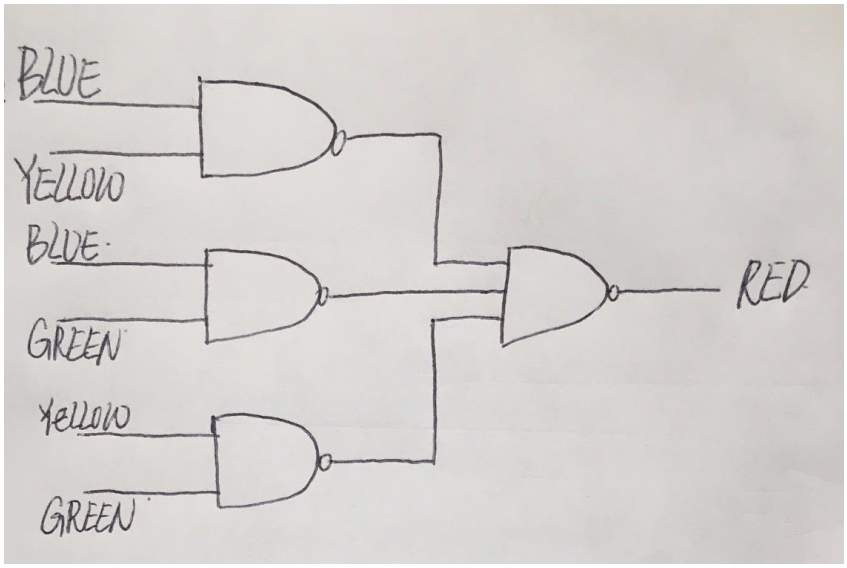
Truth table

A-Temperature	B-Pressure	C-Humidity	Red LED Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

K-Map from the truth table

	BC(00)	BC(01)	BC(11)	BC(10)
A(0)	0	0	1	0
A(1)	0	1	1	1

SOP: BC+AC+AB



In the above Kmap, A represents temperature, which is indicated by Green LED. B represents Pressure, indicated by Blue LED. C represents Humidity sensor output, indicated by Yellow LED. The output of the logic gate connects to red LED light, and will light up when we have severe weather condition.

III. Problem and challenges

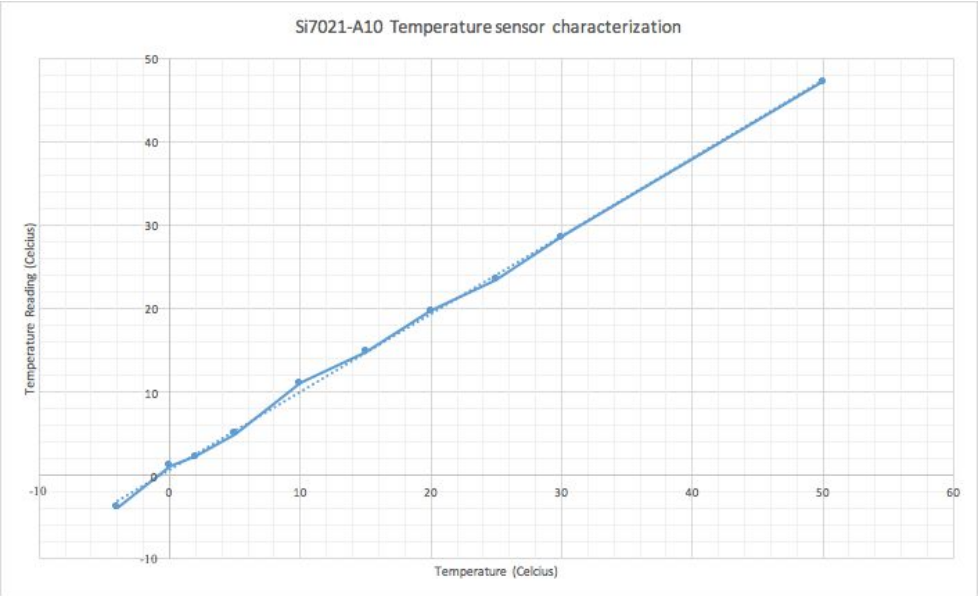
A challenge that we encountered early on was that one of our sensors was much more difficult to operate correctly than we had originally assumed it would be. We ended up using our other sensors to receive the same data that we would have gotten from the more complicated sensor. Another challenge was that we were unable to obtain the model rocket. Thus we cannot truly say how the sensors would fit into the entire rocket + circuit system.

IV. Results

A. Analysis of components

1. Temperature Sensor

Experimental setup: Update Arduino library, connect the sparkfun barometer sensor BMP180 to the arduino and breadboard, modify and upload the code, store data in EEPROM. We used a small foam cup as a container, putting the sensor circuit inside and change the temperature of the environment. Methods include bring it outside during snowing day, and heat it up by bath it in hot water. We used a thermometer as reference, compare our sensor result and the thermometer reading. We got a result very close to a linear graph.



Experimental Data and Plots:

Temperature (Celcius)	Sensor Reading Temperature (Celcius)
-4	-3.84
0	1.12
2	2.27
5	5.01
10	11.02
15	14.79
20	19.67
25	23.44
30	28.54
50	47.17

V. Future plans

Our future plans are mainly to add our combined circuits to the model rocket. Once we've done this, we can use an accelerometer to determine when the sensors should start detecting data in order to receive a more accurate reading. Ideally, it will collect

weather data when the rocket reaches the highest point and when the velocity is zero and gravity is the only accelerating factor. We'd also like to add an antenna that sends information regarding whether weather is severe or not to a local weather station.

Appendix References:

https://github.com/sparkfun/Si7021_Breakout/tree/master/Libraries/Arduino/Si7021/src

- https://www.sparkfun.com/products/13763
- https://www.sparkfun.com/products/11824
- https://www.sparkfun.com/products/10167
- https://en.wikipedia.org/wiki/Weather_satellite

CODE FOR PRESSURE AND TEMPERATURE SENSOR:

```
#include <SFE_BMP180.h>
#include <Wire.h>
#include <EEPROM.h>
#include <avr/eeprom.h>

// You will need to create an SFE_BMP180 object, here called "pressure":

SFE_BMP180 pressure;

#define ALTITUDE 222.59 // Altitude of SparkFun's HQ in Boulder, CO. in meters

void setup()
{
  Serial.begin(9600);
  Serial.println("REBOOT");
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);

  // Initialize the sensor (it is important to get calibration values stored on the device).
```

```
  if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    // Oops, something went wrong, this is usually a connection problem,
    // see the comments at the top of this sketch for the proper connections.

    Serial.println("BMP180 init fail\n\n");
    while(1); // Pause forever.
  }
}

int eeAddress = 0;
int eeAddress1 = 1;
int eeAddress2 = 2;
int eeAddress3 = 3;

void loop()
{
  char status;
  double T,P,Ph,p0,a;

  // Loop here getting pressure readings every 10 seconds.

  // If you want sea-level-compensated pressure, as used in weather reports,
  // you will need to know the altitude at which your measurements are taken.
  // We're using a constant called ALTITUDE in this sketch:

  Serial.println();
  Serial.print("provided altitude: ");
  Serial.print(ALTITUDE,0);
  Serial.print(" meters, ");
  Serial.print(ALTITUDE*3.28084,0);
```

```
Serial.println(" feet");

// If you want to measure altitude, and not pressure, you will instead need
// to provide a known baseline pressure. This is shown at the end of the sketch.

// You must first get a temperature measurement to perform a pressure reading.

// Start a temperature measurement:
// If request is successful, the number of ms to wait is returned.
// If request is unsuccessful, 0 is returned.

status = pressure.startTemperature();
if (status != 0)
{
  // Wait for the measurement to complete:
  delay(status);

  // Retrieve the completed temperature measurement:
  // Note that the measurement is stored in the variable T.
  // Function returns 1 if successful, 0 if failure.

  status = pressure.getTemperature(T);
  if (T-30 > 0) {
    digitalWrite(8, HIGH);
  }
  else if (T-30 <= 0) {
    digitalWrite(8, LOW);
  }
  if (status != 0)
  {
    // Print out the measurement:
    Serial.print("temperature: ");
    Serial.print(T,2);
```

```
    Serial.print(" deg C, ");
    Serial.print((9.0/5.0)*T+32.0,2);
    Serial.println(" deg F");
    EEPROM.put(eeAddress, int(T));
    int d1 = (T-int(T))*100;
    EEPROM.put(eeAddress1, d1);



    // Start a pressure measurement:
    // The parameter is the oversampling setting, from 0 to 3 (highest res, longest wait).
    // If request is successful, the number of ms to wait is returned.
    // If request is unsuccessful, 0 is returned.

    status = pressure.startPressure(3);
    if (status != 0)
    {
      // Wait for the measurement to complete:
      delay(status);

      // Retrieve the completed pressure measurement:
      // Note that the measurement is stored in the variable P.
      // Note also that the function requires the previous temperature measurement (T).
      // (If temperature is stable, you can do one temperature measurement for a number of pressure
measurements.)
      // Function returns 1 if successful, 0 if failure.

      status = pressure.getPressure(P,T);
      if (P > 990){
        digitalWrite(9, HIGH);
      }
      else if (P <=990) {
        digitalWrite(9, LOW);
      }
```

```
if (status != 0)
{
  // Print out the measurement:
  Serial.print("absolute pressure: ");
  Serial.print(P,2);
  Serial.print(" mb, ");
  Ph = P*0.0295333727;
  Serial.print(Ph);
  Serial.println(" inHg");
  EEPROM.put(eeAddress2, int(Ph));
  int d2 = (Ph - int(Ph))*100;
  EEPROM.put(eeAddress3, d2);

  // The pressure sensor returns abolute pressure, which varies with altitude.
  // To remove the effects of altitude, use the sealevel function and your current altitude.
  // This number is commonly used in weather reports.
  // Parameters: P = absolute pressure in mb, ALTITUDE = current altitude in m.
  // Result: p0 = sea-level compensated pressure in mb

  p0 = pressure.sealevel(P,ALTITUDE); // we're at 1655 meters (Boulder, CO)
  Serial.print("relative (sea-level) pressure: ");
  Serial.print(p0,2);
  Serial.print(" mb, ");
  Serial.print(p0*0.0295333727,2);
  Serial.println(" inHg");
  //EEPROM.put(eeAddress2, p0);

  // On the other hand, if you want to determine your altitude from the pressure reading,
  // use the altitude function along with a baseline pressure (sea-level or other).
  // Parameters: P = absolute pressure in mb, p0 = baseline pressure in mb.
  // Result: a = altitude in m.

  a = pressure.altitude(P,p0);
```

```
          Serial.print("computed altitude: ");
          Serial.print(a,0);
          Serial.print(" meters, ");
          Serial.print(a*3.28084,0);
          Serial.println(" feet");
          //EEPROM.put(eeAddress3, a);

        }
        else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  delay(5000);  // Pause for 5 seconds.
}
```

CODE FOR HUMIDITY AND TEMPERATURE SENSOR:

```
#include "SparkFun_Si7021_Breakout_Library.h"
#include <Wire.h>

float humidity = 0;
float tempf = 0;



int power = A3;
int GND = A2;

//Create Instance of HTU21D or SI7021 temp and humidity sensor and MPL3115A2 barrometric sensor
Weather sensor;
```

```
//---------------------------------------------------------------
void setup()
{
    Serial.begin(9600);   // open serial over USB at 9600 baud

    pinMode(power, OUTPUT);
    pinMode(GND, OUTPUT);
    pinMode(9, OUTPUT);
    digitalWrite(power, HIGH);
    digitalWrite(GND, LOW);

    //Initialize the I2C sensors and ping them
    sensor.begin();

}
//---------------------------------------------------------------
void loop()
{
    //Get readings from all sensors
    getWeather();
    printInfo();
  if (humidity > 35){
    digitalWrite(9, HIGH);
  }
  else if (humidity <= 35) {
    digitalWrite(9, LOW);
  }
    delay(1000);

}
//---------------------------------------------------------------
void getWeather()
```

```
{
  // Measure Relative Humidity from the HTU21D or Si7021
  humidity = sensor.getRH();

  // Measure Temperature from the HTU21D or Si7021
  tempf = sensor.getTempF();
  // Temperature is measured every time RH is requested.
  // It is faster, therefore, to read it from previous RH
  // measurement with getTemp() instead with readTemp()
}
//---------------------------------------------------------------
void printInfo()
{
//This function prints the weather data out to the default Serial Port

  Serial.print("Temp:");
  Serial.print(tempf);
  Serial.print("F, ");

  Serial.print("Humidity:");
  Serial.print(humidity);
  Serial.println("%");

}
```

CODE FOR READING TEMPERATURE AND PRESSURE FROM EEPROM:

```
/*
 * EEPROM Read
 *
 * Reads the value of each byte of the EEPROM and prints it
```

```
 * to the computer.
 * This example code is in the public domain.
 */

#include <EEPROM.h>
#include <avr/eeprom.h>

// start reading from the first byte (address 0) of the EEPROM
double value[2];
double Ph;

void setup() {
  // initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
}

int address = 0;
void loop() {
  // read a byte from the current address of the EEPROM
  for (; address < 2; address++) {
      Serial.print(address);
      if (address == 0){
        value[address] = EEPROM.read(0) + double(EEPROM.read(1))/100;
        Serial.print(". Temperature in deg C is:");
      }
      else {
        Ph = EEPROM.read(2) + double(EEPROM.read(3))/100;
        value[address] = Ph/0.0295333727;
        Serial.print(". Absolute pressure in mb is:");
      }
```

```
    Serial.print("\t");
    Serial.print(value[address]);
    Serial.println();
  }
  /***
    Advance to the next address, when at the end restart at the beginning.

    Larger AVR processors have larger EEPROM sizes, E.g:
    - Arduno Duemilanove: 512b EEPROM storage.
    - Arduino Uno:        1kb EEPROM storage.
    - Arduino Mega:       4kb EEPROM storage.

    Rather than hard-coding the length, you should use the pre-provided length function.
    This will make your code portable to all AVR processors.
  ***/
 address = address + 1;
  if (address == EEPROM.length()) {
    address = 0;
  }

  /***
    As the EEPROM sizes are powers of two, wrapping (preventing overflow) of an
    EEPROM address is also doable by a bitwise and of the length - 1.

    ++address &= EEPROM.length() - 1;
  ***/
  delay(500);
}
```