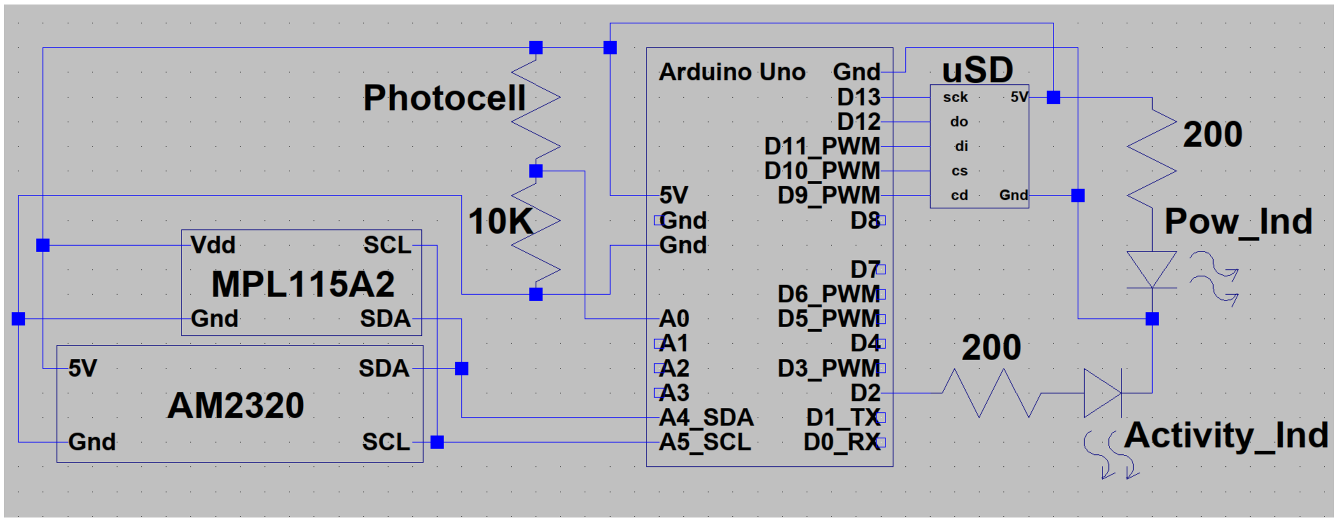
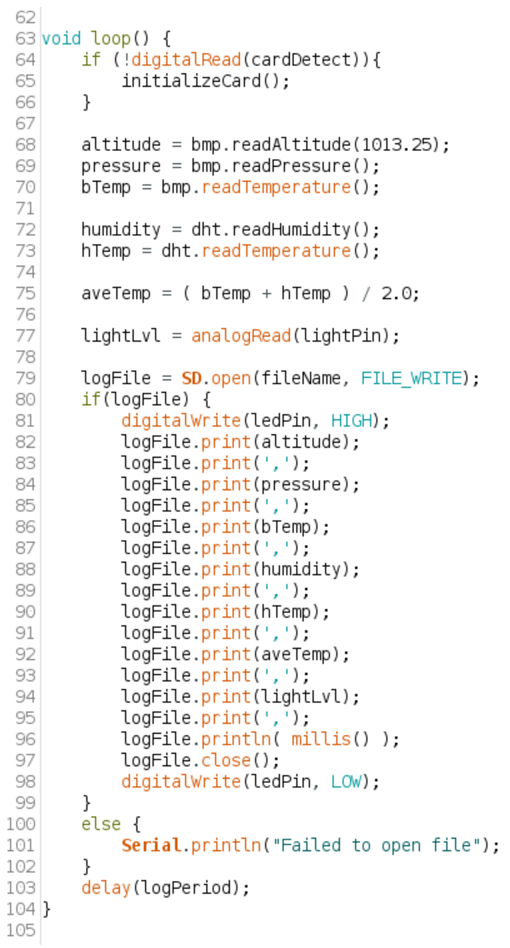
Now that all of the sub circuits have been tested, lets combine them into a full system that can record all temperature, pressure, humidity, and a timestamp so that we have a chronological order to the data. It’s also useful to have a couple indicator leds just for some basic information about the system. Lets add 2 indicator leds, one to show that the system is receiving power, and another to show the arduino is running fine. Construct the following circuit and upload the integration.ino program to the arduino.



With the device powered and viewing the serial monitor to see activity from the arduino. The power led should immediately turn on with the activity led inactive until the sd card is inserted. With the sd card inserted, the activity will light up when the logfile is opened, and turn off when the file is closed. At first this may seem like this activity would happen rather slowly, but with the sd card inserted and observed, the led should blink rather quickly. The following is the setup function contents.

```

void setup() {
    Serial.begin(9600);

    pinMode(lightPin, INPUT);
    pinMode(cardDetect, INPUT);
    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, LOW);

    dht.begin();

    if (!bmp.begin()) {
        Serial.println("bmp err");
        while (1);
    }

    // Default settings from datasheet.
    bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
                    Adafruit_BMP280::SAMPLING_X2,
                    Adafruit_BMP280::SAMPLING_X16,
                    Adafruit_BMP280::FILTER_X16,
                    Adafruit_BMP280::STANDBY_MS_500);


    initializeCard();
    logFile = SD.open(fileName, FILE_WRITE);
    if(logFile) {
        logFile.println("altitude,pressure,bTemp,humidity,hTemp,aveTemp,lightLvl,milliseconds");
        logFile.close();
    }
    else {
        Serial.println("Failed to open file");
    }
}

```

We can see the setup functions for all of the individual sensors are here. The photocell analog pin and pin for detecting the sd card are both set to inputs, while the led pin is set to an output. The humidity and barometric pressure sensors startup functions are used to initialize connection to the sensors, and the first line of the logfile is saved for indicating which comma position belongs to what data. Note that our actual logged data should follow this same order defined in the header otherwise our data will be mislabeled when graphing.



```

void loop() {
    if (!digitalRead(cardDetect)){
        initializeCard();
    }

    altitude = bmp.readAltitude(1013.25);
    pressure = bmp.readPressure();
    bTemp = bmp.readTemperature();

    humidity = dht.readHumidity();
    hTemp = dht.readTemperature();

    aveTemp = ( bTemp + hTemp ) / 2.0;

    lightLvl = analogRead(lightPin);

    logFile = SD.open(fileName, FILE_WRITE);
    if(logFile) {
        digitalWrite(ledPin, HIGH);
        logFile.print(altitude);
        logFile.print(',');
        logFile.print(pressure);
        logFile.print(',');
        logFile.print(bTemp);
        logFile.print(',');
        logFile.print(humidity);
        logFile.print(',');
        logFile.print(hTemp);
        logFile.print(',');
        logFile.print(aveTemp);
        logFile.print(',');
        logFile.print(lightLvl);
        logFile.print(',');
        logFile.println( millis() );
        logFile.close();
        digitalWrite(ledPin, LOW);
    }
    else {
        Serial.println("Failed to open file");
    }
    delay(logPeriod);
}

```

Here is the contents of our loop function. Reading through it, we can see the card detect pin is checked to make sure the card is still inserted, and if it isn’t, the card is re-initialized to make sure our file exists. After the system makes sure the card is still inserted, the system starts to poll sensor data. First pressure and temperature is pulled from the pressure sensor, then humidity and temperature is pulled from the humidity sensor. Those temperatures are averaged, and then the light value is read from the photocell. After this, the logfile is opened using the SD library, and if the logfile was opened correctly, the led pin is turned on. With the logfile opened, all of the saved data is printed to the logfile with comma separators (delimiters), and at the end println is used to print the last bit of data with a newline after it. The trailing data is the millis timestamp which is how many milliseconds the current program has been running. After the file is then closed, the activity led is turned off, and an artificial delay is inserted just to allow the system to settle before looping again. The delay isn’t always necessary, and I usually like to just run the system as fast as possible assuming that testing hasn’t shown the system to have bad behavior at those speeds from ripples in the signal corrupting data quality.

Run the system for a short while, observing the behavior of the activity light for a few cycles, and remove the sd card and check the contents to make sure the system indeed worked. Copy the logfile to your computer and we will look at about graphing shortly. First, lets assess the composition of the payload so far. We have a product, that will record the parameters that we desired, and used the barometer to approximate our altitude, and we have a marker to know when each datapoint happens relative to startup time T=0. This timing is using the arduino’s internal clock source to keep time, but its internal time assumes its clock source is a perfect electric metronome at a certain frequency. However in reality every device has its imperfections and clock crystals can be affected by system temperature and still drift over time at a steady temperature. Very accurate time keeping is done with an RTC (Real Time Clock) which uses extremely high frequency and low drift crystal oscillators to keep accurate time for years as long as they aren’t powered down, and they often have small button cell backup batteries so that when a host system is powered off the RTC never fully shuts down the time keeping piece, ensuring time is never lost. Real time clocks are often included with GPS devices which can be used for detecting altitude to a higher altitude than many low cost barometers. Some GPS sensors come with a limiter that makes them stop reporting data before hitting our desired altitude of 100 kft, so make sure that any GPS sensor used in ballooning is unlocked or can be unlocked through commands. If a GPS is used, make sure that the antenna can point upward unobstructed. Many have connectors for external weather proof antenna for mounting a GPS sensor inside a housing without sacrificing reception.

Our test conditions uses the USB as a power supply, and depending on the source, a significant amount of current may not be within the host USB power capabilities, and systems much larger and more power hungry may experience frequent system restarts or the system may not fully turn on, or a dim power led and no activity light flashing. Normally simple low power sensors like barometers or resistive elements don’t consume much power, but transmitting and receiving wireless data can require more power than a laptop USB supply can handle. External power should then be used. In our case, we’re only using very low power sensors, and our data shows up fine on the sd card, so no problems should be apparent so far.