NAU Weather Station User Manual

Domenico Galati, djg323@nau.edu
Faris Alshammari, fa388@nau.edu
Mohammed Alhajri, mna238@nau.edu
Enad Alharbi, eaa286@nau.edu

GTA: Liming Zheng, lz239@nau.edu

Client: Dr. Winfree

NAU Capstone

Intro

This user manual will cover how to use and operate the NAU Weather Stations Arduino based weather station model and explain how to keep the device running for years.

Installation

Starting with the barebones unit, each part will need to be soldered to the PCB to create a functional unit. Each footprint is specific to the components needed so no instructions are included on what goes where. When assembling the shield, the only other things needed to do is to connect two wires with a jumper wire that we were unable to connect with the current design. You can see the pieces on **Figure 1**, The two connectors labeled "JP1" and "REF**" need to be connected to their corresponding pin labelled the same. If these are not connected, the RTC and the SD Card reader will not function at all.

Configuration and use

To begin using the weather station as it stands, you must first ensure the correct software is uploaded to the arduino microcontroller. Please ensure the computer you are using has the arduino compiler installed, if it is not, please go to the following links to either install the software or use the online editor. Once the editor is open, follow the steps below:

1. Plug the arduino into the computer WITHOUT any shields attached
2. Open the **WSC.ino** file attached
3. On the top toolbar, hover over tools and ensure the following settings match:
    a. Board: Arduino/Genuino Uno
    b. Port: This usually only contains 1 choice but if there are multiple, unplug the arduino and once you plug it back in, the COM## device that appears should be the proper connection to select
4. ONLY if your education platform requires modification to the code and the system's functionality, STOP HERE and make those changes now.
5. If no changes are made or the changes have been completed, you can now compile the code by clicking the check mark button(✓). If all goes well, it should say "Done Compiling"
6. Once the code is compiled, press the arrow button(➔) to upload the code to the arduino. Once the upload is complete, it should say "Upload Complete"
7. Unplug the arduino and add the shield attachments as needed in the correct orientation and location as noted in **Figures 3 & 4** on the Appendix.
8. Once the device is assembled, plug the device back into the computer and open the serial monitor by clicking the 🔍 button in the top right corner of the arduino program or using any serial monitor program (e.g Putty)

9. Make sure the Baud rate of the serial monitor matches the baud rate written in the code, program default is 9600 but our code uses 115200 by default

At this point, the arduino will output the current code print statements continuously until the unit is powered down. This is as far as we will take the educational platforms. The units should read out all necessary weather data such as temperature, humidity, pressure, UV index, etc. At this point, each institution is allowed to alter and modify their code as they please to further the educational program for their students

Maintenance

With the way we designed our product, there should be very little to no maintenance required. A weekly check of the unit should allow for more than necessary care to keep the unit functional, the biggest hazard to the computers is water and condensation. If the unit is kept out of the rain and there is relatively little to no other issues. The unit will remain functional for quite some time.

Trouble-shooting

Most all issues ran into will be a computer related issue which can be resolved by checking out the configuration to make sure its properly set up but if it is not. It could be a broken sensor or bad solder job. If a specific sensor is not reading, try swapping it with a known working sensor. If these breakout sensors are placed in the wrong orientation or slot, it could easily break the hardware rendering the sensors useless. ALWAYS ensure the boards are correctly placed BEFORE powering up the unit. We cannot stress this enough.

Conclusion

All in all, if you follow these steps, the device should work as expected without any issues.

Appendix

Schematics, Picture, Code, Etc.

Figure 1. Expansion Board (Bottom View)

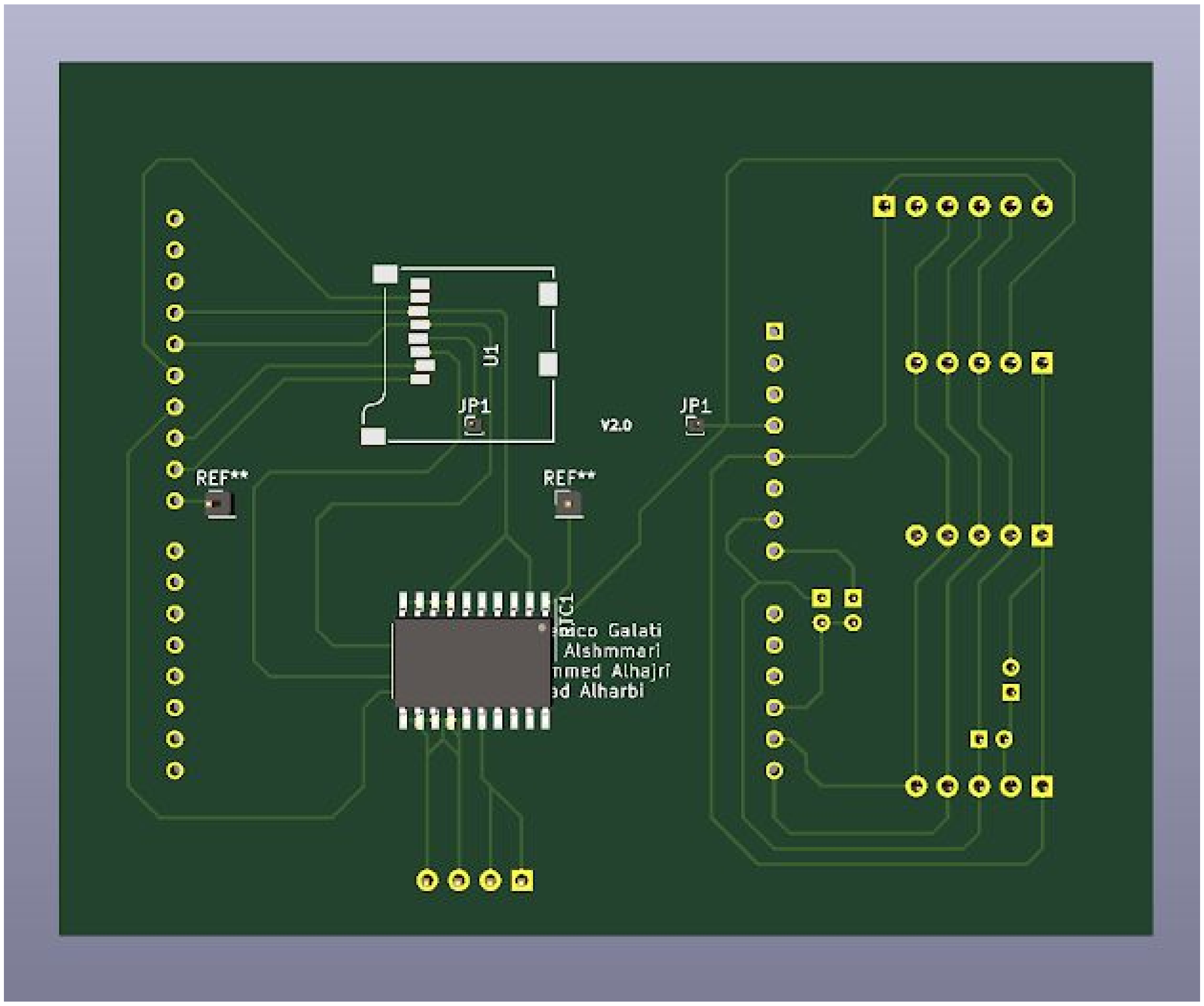


Figure 2. Expansion Board (Bottom View)

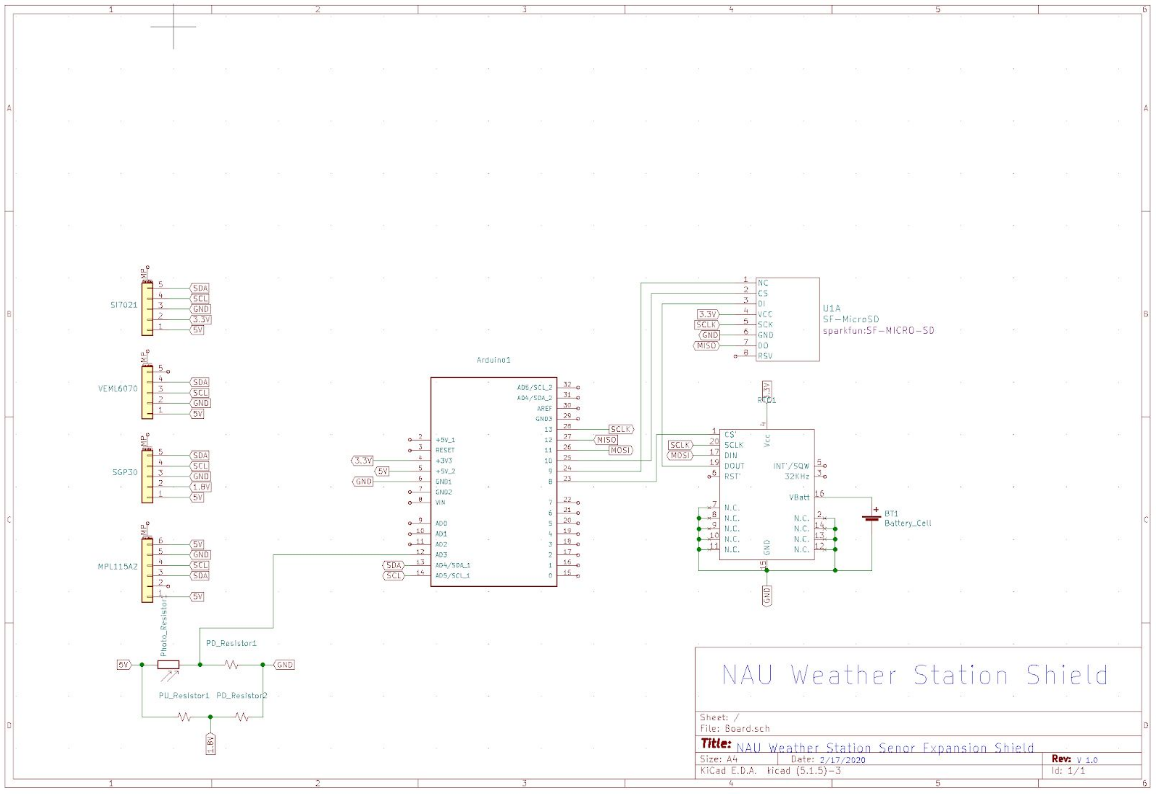


Figure 3. Sensor Layout (Bottom View)

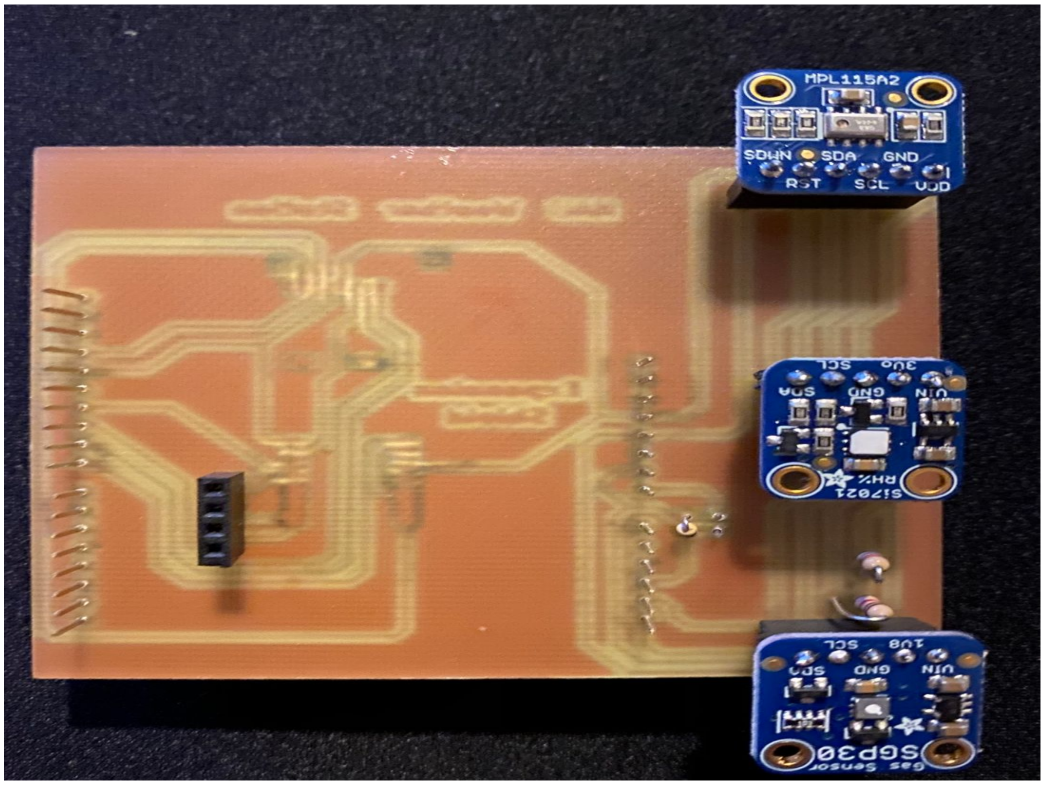
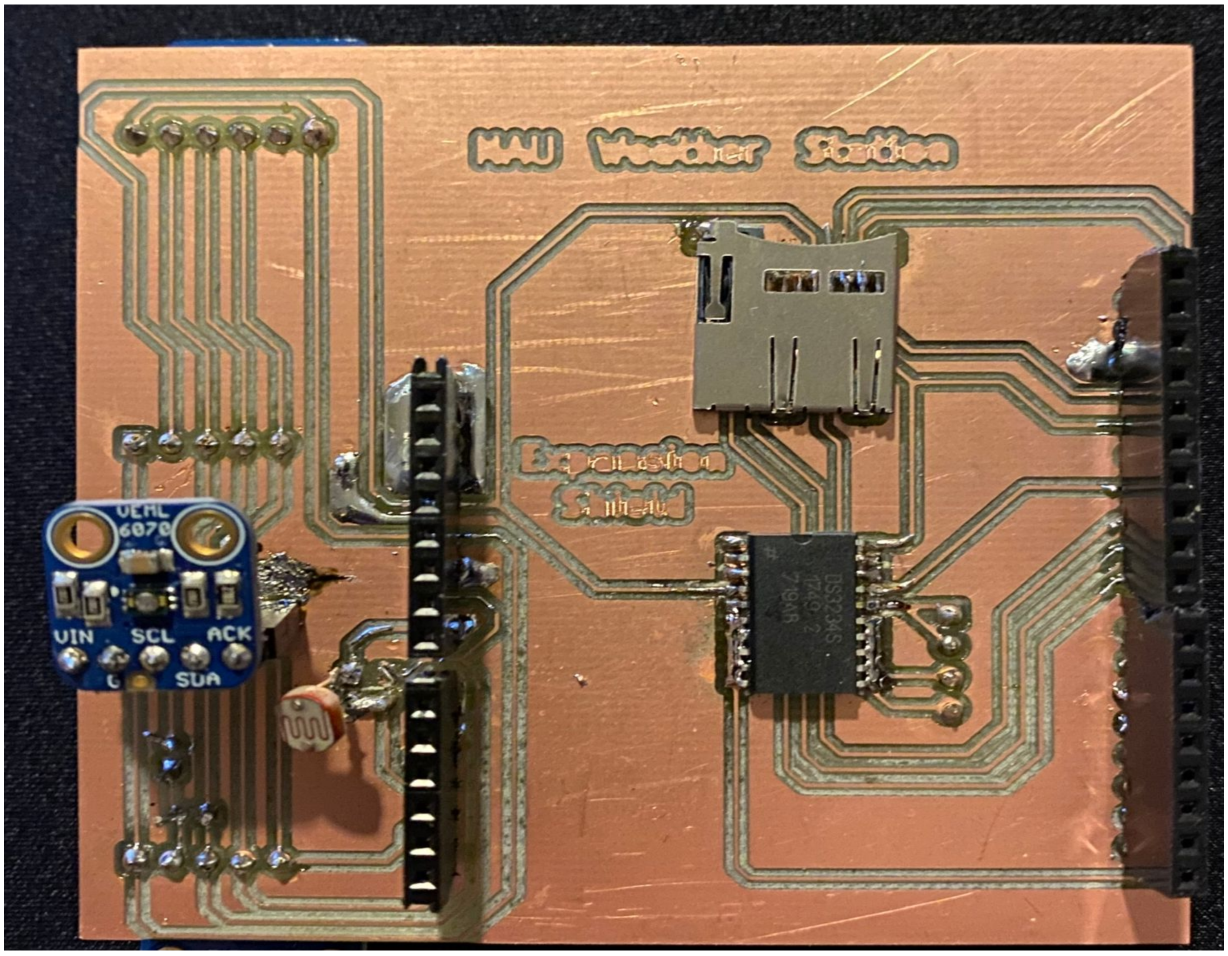


Figure 4. Sensor Layout (Top View)



WSC.ino Code

```
// include the SD library:
#include <SPI.h>
#include <SD.h>

#include <Wire.h>
#include "Adafruit_VEML6070.h"
#include "Adafruit_Si7021.h"
#include "Adafruit_SGP30.h"
#include <Adafruit_MPL115A2.h>

Adafruit_MPL115A2 mpl115a2;
Adafruit_SGP30 sgp;
Adafruit_Si7021 sensor = Adafruit_Si7021();
Adafruit_VEML6070 uv = Adafruit_VEML6070();
int counter = 0;

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;
const int chipSelect = 10;

void setup() {

//Si7021
  Serial.begin(115200);

  // wait for serial port to open
  while (!Serial) {
    delay(10);
  }

  Serial.println("Si7021 test!");

  if (!sensor.begin()) {
    Serial.println("Did not find Si7021 sensor!");
    while (true)
      ;
  }

  Serial.print("Found model ");
  switch(sensor.getModel()) {
    case SI_Engineering_Samples:
      Serial.print("SI engineering samples"); break;
    case SI_7013:
      Serial.print("Si7013"); break;
    case SI_7020:
      Serial.print("Si7020"); break;
    case SI_7021:
      Serial.print("Si7021"); break;
    case SI_UNKNOWN:
    default:
      Serial.print("Unknown");
  }
  Serial.print(" Rev(");
  Serial.print(sensor.getRevision());
  Serial.print(")");
  Serial.print(" Serial #"); Serial.print(sensor.sernum_a, HEX); Serial.println(sensor.sernum_b, HEX);

//MPL115A2
  Serial.println("MPL115A2 Start up");

  Serial.println("Getting barometric pressure ...");
  mpl115a2.begin();

  //SGP30
  Serial.println("SGP30 test");

  if (! sgp.begin()){
    Serial.println("Sensor not found :(");
    while (1);
  }
  Serial.print("Found SGP30 serial #");
  Serial.print(sgp.serialnumber[0], HEX);
  Serial.print(sgp.serialnumber[1], HEX);
  Serial.println(sgp.serialnumber[2], HEX);

  //VEML6070
  Serial.println("VEML6070 Test");
  uv.begin(VEML6070_1_T);  // pass in the integration time constant
  //SD Card Detect
  Serial.print("\nInitializing SD card...");

  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.println();
  Serial.print("Card type:         ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }
}

void loop() {
Serial.println();
Serial.println();
Serial.println("Reading Sensors");

  //SI7021
  Serial.print("Humidity:    ");
  Serial.print(sensor.readHumidity(), 2);
  Serial.print("\tTemperature: ");
  Serial.println(sensor.readTemperature(), 2);

  //MPL115A2
  float pressureKPA = 0, temperatureC = 0;
  pressureKPA = mpl115a2.getPressure();
  Serial.print("Pressure (kPa): "); Serial.print(pressureKPA, 4); Serial.println(" kPa");
  temperatureC = mpl115a2.getTemperature();
  Serial.print("Temp (*C): "); Serial.print(temperatureC, 1); Serial.println(" *C");
  delay(1000);

  //SGP30
   if (! sgp.IAQmeasure()) {
    Serial.println("Measurement failed");
    return;
  }
  Serial.print("TVOC "); Serial.print(sgp.TVOC); Serial.print(" ppb\t");
  Serial.print("eCO2 "); Serial.print(sgp.eCO2); Serial.println(" ppm");

  if (! sgp.IAQmeasureRaw()) {
    Serial.println("Raw Measurement failed");
    return;
  }
  Serial.print("Raw H2 "); Serial.print(sgp.rawH2); Serial.print(" \t");
  Serial.print("Raw Ethanol "); Serial.print(sgp.rawEthanol); Serial.println("");

  counter++;
  if (counter == 30) {
    counter = 0;

    uint16_t TVOC_base, eCO2_base;
    if (! sgp.getIAQBaseline(&eCO2_base, &TVOC_base)) {
      Serial.println("Failed to get baseline readings");
      return;
    }
    Serial.print("****Baseline values: eCO2: 0x"); Serial.print(eCO2_base, HEX);
    Serial.print(" & TVOC: 0x"); Serial.println(TVOC_base, HEX);
  }


  //VEML6070
  Serial.print("UV light level: ");
  Serial.println(uv.readUV());
  Serial.print("Photo Cell Value: ");
  Serial.println(analogRead(A3));

  delay(2500);
}

uint32_t getAbsoluteHumidity(float temperature, float humidity) {
    // approximation formula from Sensirion SGP30 Driver Integration chapter 3.15
    const float absoluteHumidity = 216.7f * ((humidity / 100.0f) * 6.112f * exp((17.62f * temperature) / (243.12f + temperature)) / (273.15f + temperature)); // [g/m^3]
    const uint32_t absoluteHumidityScaled = static_cast<uint32_t>(1000.0f * absoluteHumidity); // [mg/m^3]
    return absoluteHumidityScaled;
}
```