Home Weather Station

By:
Jarrett Corr

Senior Project
Computer Engineering Department

California Polytechnic State University
San Luis Obispo

June 2018
# Table of Contents

*Section* | *Page*
--- | ---
Abstract | 4
Acknowledgements | 5
Section 1 – Introduction | 6
Section 2 – Engineering Requirements | 7
Section 3 – Design | 8
  Subsection 1 – Conceptual Design | 8
  Subsection 2 – Indoor Station Hardware | 8
  Subsection 3 – Outdoor Node Hardware | 10
  Subsection 4 – Indoor Station Software | 12
  Subsection 5 – Outdoor Node Software | 14
Section 4 – Testing | 15
Section 5 – Conclusion and Future Work | 18
References | 19
Appendix A - ABET Senior Project Analysis | 21
Appendix B – Electrical Schematics | 23
Appendix C – Indoor Station CAD Files | 29
Appendix D – Outdoor Node CAD Files | 30
Appendix E – Software Flowchart | 31
Appendix F – Indoor Station Code | 33
Appendix G – Outdoor Node Code | 48
List of Tables and Figures

Tables
2.1 - Engineering Requirements ........................................ 7
4.1 - Test Results ................................................................ 15

Figures
3.1.1 - Indoor and Outdoor Cases (Respectively) .............. 8
3.2.2 - Adafruit RFM95 Breakout Board With Antennae ...... 9
3.2.3 - Top View of the Indoor Station ...................... 10
3.3.1 - Adafruit Feather M0 With RFM95 Transceiver and BME280 11
3.3.2 – Top View of the Outdoor Case ...................... 12
3.4.2 – Time Settings Menu ........................................... 13
3.4.2 – Example of Color Shifted Text ...................... 13
4.1 - 1.2Ω \( V_{\text{sense}} \) Resistor For Current Measurement .... 16
4.2 - Wake-Up Voltage Over Current Sense Resistor .... 17
4.3 – Radio Transmission Voltage Over Current Sense Resistor 18
B.1 - Indoor Station Schematic ..................................... 23
B.2 - Outdoor Node Schematic ...................................... 24
B.3 – Feather M0 With RMF95 Transceiver Schematic .... 25
B.4 – Adafruit BME 280 Breakout Schematic ............... 26
B.5 – H8357D Touchscreen Adafruit Breakout Schematic .... 27
B.6 – RFM95 Adafruit breakout schematic .................. 28
C.1 – Indoor Station CAD Image .................................. 29
D.2 – Outdoor Node CAD Image .................................. 30
E.1 – Outdoor Node Program Flow ................................ 31
E.2 – Indoor Station Program Flow .............................. 32
Abstract

This senior project implements a mobile weather station that can relay weather data to an indoor module with an LCD display. The outdoor module utilizes a temperature, humidity, and pressure sensor to collect atmospheric weather data and a radio transceiver to transmit the data to a display station via radio frequency communications. The indoor station displays the temperature, humidity and time with an LCD display. The indoor station has a settings menu which allows the user to configure the unit of measurement for the temperature and time where the device is located.
Acknowledgements

I’d like to thank Dr. Bridget Benson for advising my senior project and being lenient on my procrastination. Next, I’d like to thank my parents, Paul and Jamie Corr, for helping me throughout my college career and believing in the fact that I could see the light at the end of the tunnel when I couldn’t. Also, I’d like to especially thank my father helping me navigate the senior project’s milestones and deadline as it approached the final deadline.
Section 1 – Introduction

This project aims to bring together a product development cycle from the technical knowledge of all the classes taken in the last four years of university. The product is a home weather station whose objective is to provide accurate weather data directly at your house. The data aims to be highly accurate to the conditions at an individual’s house and readily available. The small package of the indoor station allows the user to mount the indoor node in a convenient location to inform them of the current weather conditions.

There are already weather station devices on the market that utilize an outdoor node to transmit environmental information to a base station. I currently own a C85845 weather station from La Crosse Technologies. A large portion of this project is inspired by that weather station. The C85845 provides temperature (both units of measure), humidity, outdoor node battery level, and some weather predictions through an internal algorithm seeded by the current and previous weather conditions [1]. I aim to make a simpler more portable version of the C85845 weather station with a simplistic user interface for a cheaper marketplace for quality-built weather stations providing accurate weather information.

The rest of the paper details the requirements, final build, testing of the device. The requirements are organized in a table with reasoning for each requirement above the table. The next section provides a description of the device’s three main systems: Electronic hardware, software, and physical hardware. Each component subsection each contain a separate description for the indoor station and the outdoor node. Lastly, the report details the testing to analyze the overall success of the design by adhering to the engineering requirements described in Section 2.
Section 2 – Engineering Requirements

The engineering requirements originate from the need to make an accurate, user-friendly, and small weather station. The small package would allow the user to place the station anywhere they want to capture environmental data. The outdoor node needs to maintain a connection with the indoor station at ranges of 50 feet to allow the user to place the outdoor node at any corner of their house and maintain communications. Another major concern is the battery life of the outdoor node which provides a peace of mind to the user about the battery life of the outdoor node. Also, the outdoor node needs to transmit its battery level and the indoor station needs to display the battery level to provide the user ample time to change the outdoor node’s battery and charge the depleted battery without substantial system downtime.

Table 1 – Engineering Requirements

<table>
<thead>
<tr>
<th>Spec. Number</th>
<th>Parameter Description</th>
<th>Target</th>
<th>Tolerance</th>
<th>Risk</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indoor Station Volume</td>
<td>3.5” x 2.5” x 1.5”</td>
<td>Min</td>
<td>Low</td>
<td>Inspection</td>
</tr>
<tr>
<td>2</td>
<td>Outdoor Node Volume</td>
<td>3” x 2” x 1”</td>
<td>Min</td>
<td>Low</td>
<td>Inspection</td>
</tr>
<tr>
<td>3</td>
<td>Transceiver Range</td>
<td>50 feet</td>
<td>High</td>
<td>Medium</td>
<td>Testing</td>
</tr>
<tr>
<td>4</td>
<td>Outdoor Node Battery Life</td>
<td>1 year</td>
<td>Med</td>
<td>High</td>
<td>Testing</td>
</tr>
<tr>
<td>5</td>
<td>Low Battery Detection</td>
<td>Display Message</td>
<td>Min</td>
<td>Low</td>
<td>Inspection</td>
</tr>
<tr>
<td>6</td>
<td>Temperature Accuracy</td>
<td>1º F</td>
<td>High</td>
<td>Low</td>
<td>Inspection</td>
</tr>
<tr>
<td>7</td>
<td>Humidity Accuracy</td>
<td>1% RH</td>
<td>High</td>
<td>Low</td>
<td>Inspection</td>
</tr>
<tr>
<td>8</td>
<td>Time Settings</td>
<td>All time zones</td>
<td>Low</td>
<td>Med</td>
<td>Testing</td>
</tr>
<tr>
<td>9</td>
<td>Waterproof Enclosure</td>
<td>No water in container</td>
<td>High</td>
<td>Low</td>
<td>Inspection</td>
</tr>
</tbody>
</table>
Section 3 – Design

Subsection 1 – Conceptional Design
The design implements an extremely small weather monitoring system capable of being placed anywhere in or out of the user’s living space. The design utilizes two different microcontrollers, an Arduino Mega and a Feather M0, to record and communicate the outdoor environmental conditions to the display placed inside the user’s home. Figure 3.1.1 shows the two stations side by side.

Subsection 2 – Indoor Station Hardware
The microcontroller on the indoor station is an Arduino Mega 2560 providing control of the display and receiving transmissions from the outdoor node. The display is connected via an 8-pin parallel connection shown in Figure 3.2.1. Also, the figure shows connections to the radio transceiver that communicates with the outdoor node.

The two peripherals for the indoor station are the HX8357-D LCD display and the RFM95 radio transceiver. The two peripherals were purchased from Adafruit utilizing the given breakout boards to expedite the software as the sensor’s and display’s hardware configuration were already provided.

The display is a 480x320 3.5” LCD screen with touchscreen capabilities. The microcontroller originally connected to the display via an SPI interface but the refresh rate and clearing the display to update the data was visible and was tearing image when updating. The connections were updated to an 8-bit parallel connection to increase the baud rate and minimize the screen’s drawing time. Unfortunately,
there was not a perceivable difference, but the change was implemented and remained in the final design. The touchscreen allows the user to update the settings with a simple, sleek transition to a settings page. The touchscreen utilizes another 4 pins to locate the x, y position of the touch and the pressure of the touch by measuring the voltage gradient of the screen when touched.

The RFM95 radio transceiver provides easy wireless communications via radio frequency broadcasts. The transceiver operates at 915MHz according to ITU Radio Regulations for scientific and industrial radio frequencies [2]. The RFM95 is configured with an SPI connection to communicate to the on-board IC to control of the transceiver. The transceiver then broadcasts the data packet at maximum power with a 3” sold-core wire antennae.

![Figure 3.2.2 – Adafruit RFM-95 breakout board With Antennae](image)

The indoor station is powered by a standard 9V DC power supply. The power supply plugs into the Arduino Mega with a prefabricated socket for the power supply.

The electronic hardware is contained in a 3-D printed rectangular container. There are two pieces, a top with a cut-out for the screen to push flush with top of the case. The screen is fastened to the top with machine screws running through screw holes in the screen’s breakout board. Lastly, the top is finished with a black weather strip to cover the screen’s PCB and make the top perfectly flush. On the base the Arduino Mega is fastened via machine screws with hole pre-drilled into the plastic base.
Subsection 3 – Outdoor Node Hardware

The outdoor node uses a Feather M0 with a LoRa RF transceiver onboard for its microcontroller. The Feather M0 has an extremely small form factor, 2.0” x 0.9” x 0.3”. The small PCB allows for the packaging to be smaller allowing for more flexibility in the node’s placement outdoors. The microprocessor is shipped with an RFM95 transceiver onboard, an identical model to the indoor station. Only the BME 280 needed to be wired to the microprocessor as shown in Figure B.2 in Appendix B.
The outdoor node captures environmental data from the BME280, a three-in-one temperature, humidity, and pressure sensor. The device is accurate to a 0.5°C for temperature and 3% relative humidity as specified by the BME280 datasheet [3]. The temperature sensor is soldered to a PCB and wired to the Feather M0 with solid core wire. The whole device is powered by a 3.7V, 1200mAh Lithium Polymer battery that plugs directly into the Feather M0. The battery is stepped down to 3.3V by an on-board voltage regulator to always provide a safe voltage source for the device.

The outdoor electronics is placed inside a 3D printed case waterproofed with 3M marine epoxy. The epoxy is extremely waterproof and almost weatherproof, a detriment to the product needed more research to be done for the optimal waterproofing material. The epoxy is not a great conductor of heat so the sensor lags behind the current conditions. Also, when in direct sunlight can heat up the case and sensor causing an inaccurate data capture.
Subsection 4 – Indoor Station Software

The indoor station has one purpose, display the information transmitted by the outdoor node. The software achieves this objective by continuously monitoring for any RF communications in the 915MHz band and parsing the data packet sent by the outdoor node. After parsing the communications, the software checks to see if the data is identical to the last transmission by referencing a global variable for the previous packet’s data. If the current temperature is more than 0.5°C different from the previous temperature, the display updates. The whole of the software flow is printed in Appendix C for the indoor station.

The touchscreen runs continuously and captures coordinates from the touchscreen. The coordinates range from (0, 0) to (1000, 1000). If a point is captured in an area of interest, the temperature readings or the time, the code waits for half a second to capture another point.

If the two points match, then an action is performed. When the temperature is touched and held for half a second, the units change between Fahrenheit to Celsius and force the LCD display to rewrite the data so the new values and units display immediately. When the time is held for half a second, the screen displays a new menu change the time and confirm the change.
Also, the text color of the temperature is color shifted by the value of the outdoor temperature. An “ideal” outdoor temperature is decided to be 75°F by an informal survey of people. From the “ideal” outdoor temperature the color of the text is red-shifted the more the temperature rises and blue-shifted the lower the temperature is from the “ideal” temperature.
Subsection 5 – Outdoor Node Software

The outdoor node has two functions record data and broadcast it to the indoor station. The software communicates with the BME280 via SPI and receives temperature, humidity, and pressure data. Then the data is packed into a packet and broadcasted. Once the broadcast is complete the device goes to sleep for 2 minutes and records new data. If the new data is not different than the previous transmission, the node will not transmit the data to save battery power.

The outdoor node’s software tracks the battery level of the device and sends an alert to the indoor station. The software piggybacks the analog voltage of the battery on the weather data packet to inform the indoor station if it should display an alert to the user. The piggybacking the battery status onto the weather data halves the number of the transmissions which will greatly increase the battery life of the outdoor node.
Section 4- Testing

Table 4.1 – Test Results

<table>
<thead>
<tr>
<th>Spec. Number</th>
<th>Parameter Description</th>
<th>Target</th>
<th>Final Design Spec</th>
<th>Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indoor Station Volume</td>
<td>3.5” x 2.5” x 1.5”</td>
<td>3.5”x 2.5” x 1.75”</td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>Outdoor Node Volume</td>
<td>3” x 2” x 1”</td>
<td>3” x 2.5” x 1.5”</td>
<td>NO</td>
</tr>
<tr>
<td>3</td>
<td>Transceiver Range</td>
<td>50 feet</td>
<td>125 feet</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>Outdoor Node Battery Life</td>
<td>1 year</td>
<td>1.25 years</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>Low Battery Detection</td>
<td>Display Message</td>
<td>Appears next to outdoor data (red when low)</td>
<td>YES</td>
</tr>
<tr>
<td>6</td>
<td>Temperature Accuracy</td>
<td>1° F</td>
<td>Failed in direct sunlight</td>
<td>NO</td>
</tr>
<tr>
<td>7</td>
<td>Humidity Accuracy</td>
<td>1% RH</td>
<td>N/A</td>
<td>YES</td>
</tr>
<tr>
<td>8</td>
<td>Time Settings</td>
<td>All time zones</td>
<td>The time is set manually</td>
<td>YES</td>
</tr>
<tr>
<td>9</td>
<td>Waterproof Enclosure</td>
<td>No water in container</td>
<td>Epoxy coated case</td>
<td>YES</td>
</tr>
</tbody>
</table>

Starting with the first requirement and its result, at the top with the indoor station volume. The final design needed space for the wiring inside the case so an extra 0.25” was added to allow for ease of wiring when constructing the final product. The next iteration would solve this issue by printing a custom PCB with the microcontroller on board and would eliminate all the wiring allowing for a thinner case design.

The outdoor node was larger than expected because during manufacturing the prototype the engineering requirements were used as inner dimensions and the quarter inch thick walls contribute to the extra length and width of the case.

The transceiver range test was conducted through a garage wall and walking down a driveway. Initially the test failed because the original design had the antennae inside the case which could not penetrate the plastic case after 25 feet. Routing the antennae outside the case increased the range by five times. With the antennae outside the case the range when the communications became spotty at 125 feet.

The most important requirement the outdoor node’s battery life. The node does two things take a reading and transmit the data. During the node’s sleep cycle, it wakes up four times as it can only sleep
for 15 seconds at a time and immediately goes to sleep again. The figure below is the device waking up and sleeping.

Figure 4.1 – 1.2Ω Vsense Resistor For Current Measurement

The voltage is over a 1.2Ω sense resistor in between the microcontroller’s ground terminal and the battery’s ground terminal. The red box is the time frame when the node wakes up and goes back to sleep. Using cursors to measure the voltage over the sense resistor is 8mV and the timeframe of 24.3ms. Using Equation 4.1 yields the current and then Equation 4.2 yields the charge consumed from a quick wake up to be 4.50e-5mAh.
Figure 4.2 - Wake-Up Voltage Over Current Sense Resistor

\[ I = \frac{V}{R} \]

Equation 1 – Ohm’s Law

\[ mAh = \frac{I \times T}{36000000} \]

Equation 2 – Charge consumption over time

The on-time charge consumption and temperature reading consumption is in Figure 4.2 and the calculations yield 4.5e-5 mAh. The radio transmission follows the same calculation with Figure 4.2’s measurements, yielding 1.79e-3 mAh. In total each wakeup and transmissions consume 1.84e-3 mAh. Dividing the battery’s capacity of 1200mAh by the wakeup and transmissions consumption there is a total of 652,174 transmissions. Assuming the worst case and a transmission occurs every minute, the battery will last 652,174 minutes which translates to 1.24 years.
Next, the temperature accuracy is accurate so long as the outdoor node is not in direct sunlight. This was demonstrated at the expo when the sensor reported over 100°F when in direct sunlight and 86°F in the shade while the weather station KCASANLU45, an on-campus station, reported 79°F. The issue lies in the epoxy. The epoxy holds a lot of heat, and the black case absorbs a lot heat as well. The compounding issues cause temperature inaccuracy.

The humidity accuracy failed because of the weatherproofing strip between top and bottom of the outdoor case. Once, removed the humidity was accurate again to 1% as compared to the C85845 weather station I have at home.

The waterproof enclosure was tested by pouring water over the top when the case is fully constructed and then swabbed the inside for any water.
Section 5 – Conclusion and Future Work

In conclusion, the device can record and display environmental data from a singular outdoor node using various units of measurement. At the current state of the project, it is not a weather station to take to market to compete in a spot for a basic, low-cost weather station. Some future work needs to be done to put the product at a competitive spot.

One is to implement multiple nodes and allow the user monitor different sections of the house. Any extra nodes would be an increase in the communications software, specifically the packet structure to differentiate between the two nodes and a simple branch in the display logic to update the correct temperature reading on the display.

Also, a smaller form factor could be achieved if a SMT PCB was developed for the indoor station. The singular PCB would allow for a large reduction in the depth of the indoor station allowing it to be flush with many surfaces in a typically household application.

Lastly, the cost could be brought down if the software was overhauled for an ATMega328 (Arduino Uno) rather than an ATMega2560 (Arduino Mega). There are many pins that remain unused with the current design which translates to extraneous component costs.

For the outdoor node, if the indoor station was recoded to transmit an acknowledgement to the outdoor node when it receives its packet, a dynamic broadcasting power could be implemented. The dynamic power would minimize battery consumption for any arbitrary range. The basic concept would be: if the outdoor node did not receive a response from the indoor station it would boost power to the radio. If it received a packet it would reduce power to the next broadcast. Finally, there would be a switching limit where once it has moved the power up and down a unit multiple times it would lock in the average while still boosting the power if by chance the packet didn’t get through to the indoor station.
References


8. Olikraus. “Olikraus/U8g2_for_Adafruit_GFX.” GitHub, github.com/olikraus/U8g2_for_Adafruit_GFX.
Appendix A – Senior Project Analysis

Summary of Functional Requirements:
There are three main goals with the project: record data from a microcontroller, transmit the data to another microcontroller, and the second microcontroller displays the information. The project utilizes the dual microcontroller configuration to relay the environmental conditions to another and the user to remotely view the data. The small form factor of the indoor station allows the user to place the display anywhere.

Primary Constraints:
The primary constraint is the battery life of the outdoor node. The battery life of the outdoor node will directly affect the usability of the project. If the battery needs to be recharged every month the device is failing to have little to no upkeep. Another primary constraint was the indoor station packaging. A small package is important to ensure that the user has the freedom to put the station anywhere they would find most convenient.

Economic:
The original cost of the device was approximately $150. This included all the components and pin headers. A major change was the change of microcontrollers from MSP432 to an Arduino Mega and a Feather M0. Also, the price was sourced from Digikey, however the components were all bought on Amazon. Another big change was the inclusion of a package which added approximately $10. The package pricing is related to the cost of printable plastic used on the indoor and outdoor package. The cost does not include the capital of the printer itself. Another cost change is the battery which has replaced the solar cell on the bill of materials, but the change was cost neutral.

The equipment necessary for the project was an oscilloscope, digital multimeter, and some USB-to-microUSB cables to communicate with the Feather M0. The measurement equipment was all contained in an Analog Discovery 2 which is $180 for students to purchase on Digilent’s website.

Manufacturing:
In order to take the project to large scale production four things need to happen: a custom PCB, a new case, a product website, and selective advertising. First a custom PCB allows for sustainably lower per-unit costs as the manufacturing would move from development board assembly to automated pick-and-place machines and reflow soldering. A new plastic case would be designed to be manufacturable and use minimal material while providing a strong case. These two changes aim to decrease the per-unit manufacturing cost.

Because a new business is forming around the product a website and selective advertising would allow individuals to view and purchase the product. This aims to create the revenue to pay for the manufacturing and R&D for future designs.

Environmental:
The environmental impact of the product is considerable. The outdoor node because of its small packaging could be forgotten over the course of time and left outdoors forever. The plastic and silicon
used to manufacture the outdoor node would not discompose. There is no foreseeable health risk to wildlife or plants as the outdoor node is purely contained and is completely solid.

**Ethical:**
If the project was produced on massive scale, there would be some intellectual property copyrights and licenses that would need to be honored. All Adafruit libraries are allowed for redistribution (in text or binary) as long the distributor credits Adafruit with the library functionality in the text file and binary file.

**Health and Safety:**
The outdoor node could be a choking hazard for a small child if ingested. A warning would need to be provided in the user manual, instructing the user to keep the product away from small children. Another potential hazard would be a static shock hazard. Both nodes operate on less than 5V with minimal current, but there would be a chance when opening the outdoor node and replacing the battery the user could touch two exposed pins and shorting them and shocking the user. Other than the choking hazard and potential minor electrical shock there are no other health or safety hazards.

**Social and Political:**
There is no social or political motivating force that produced this product or that the product would produce.

**Development:**
I learned how to implement a graphical display and touchscreen which was an awesome experience. The display required some trial and error. I critical flaw I discovered is that the display does not auto-refresh on a write to it. Writing on top of the display will cause a mess as the previous screen will remain. A project development element I learned was how to slim the product to the requirements specified in the engineering requirements. A slimmer product allows a cheaper product in terms of cost and price, allowing a more competitive product.
Appendix B – Electrical Schematics

Figure B.1 – Indoor Station Schematic

NOTE: Unused Pins Not Shown
Figure B.2 – Outdoor Node Schematic

NOTE: Unused Pins Not Shown
Figure B.3 – Feather M0 With RMF95 Transceiver Schematic
Figure B.4 – Adafruit BME 280 Breakout Schematic
Figure B.5 – H8357D Touchscreen Adafruit Breakout Schematic
Figure B.6 – RFM95 Adafruit breakout schematic
Appendix C – Indoor Case CAD

Figure C.1 – Indoor Station CAD Image
Appendix D – Outdoor Node Case CAD

Figure D.2 – Outdoor Node CAD Image
Appendix E – Software Flowchart

Figure E.1 – Outdoor Node Program Flow
Figure E.2 – Indoor Station Program Flow
Appendix F – Indoor Station Code

#include <Wire.h>
#include <Adafruit_BME280.h>
#include <TouchScreen.h>
#include <Adafruit_SPI TFT_Macros.h>
#include <Adafruit_SPI TFT.h>
#include <registers.h>
#include <pin_magic.h>
#include <Adafruit_TFTLCD.h>
#include <Adafruit_GFX.h>
#include <TimeLib.h>
#include <Time.h>
#include <U8g2_for_Adafruit_GFX.h>
#include <u8g2_fonts.h>
#include <gfxfont.h>
#include <stdlib.h>
#include <SPI.h>
#include <RH_RF95.h>

// LCD display pins
#define LCD_CS 32
#define LCD_CD 33
#define LCD_W 34
#define LCD_R 35
#define LCD_RST 36

// touchscreen pins
#define XP A2   // can be a digital pin
#define YP A3  // must be an analog pin, use "An" notation!
#define XM A0  // must be an analog pin, use "An" notation!
#define YM A1   // can be a digital pin
#define X_RESIST 300 // resistance between X+ and X-
#define TS_MINX 0
#define TS_MINY 0
#define TS_MAXX 320
#define TS_MAXY 480
#define MINPRESSURE 10
#define MAXPRESSURE 1000

// radio transciever pins
#define RFM95_RST     3
#define RFM95_CS      53
#define RFM95_INT     2

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// bme pins
#define BME_SCK 45
#define BME_MISO 44
#define BME_MOSI 43
#define BME_CS 42

// LCD Pixel Dimensions
#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 480
#define PRIME_MERIDIAN 160
#define EQUATOR 240

// Color definitions
#define BLACK 0x0000
#define BLUE 0x001F
#define GREEN 0x07E0
#define CYAN 0x07FF
#define RED 0xF800
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE 0xFFFF
#define ORANGE 0xFC00
#define YELLOW_GREEN 0xD7A1

#define INDOOR 1
#define OUTDOOR 2

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);
Adafruit_TFTLCD display = Adafruit_TFTLCD(LCD_CS, LCD_CD, LCD_W, LCD_R, LCD_RST);
U8G2_FOR_ADAFRUIT_GFX text;
TouchScreen ts = TouchScreen(XP, YP, XM, YM, X_RESIST);
Adafruit_BME280 bme;

String recvPacket;
bool farheinheit = true;

void radioInit();
void lcdInit();
void bmeInit();
void readDataPacket(char *recvBuf, float *temperature, float *humidity, float *pressure, float *voltage);
void takeMeasurement(float *temperature, float *humidity, float *pressure);
void printMeasurement(float temperature, float humidity, float pressure, float voltage);
void writeOutdoorTemperature(float temperature, float humidity, bool forced);
void writeIndoorTemperature(float temperature, float humidity, bool forced);
void updateSign(unsigned int x, unsigned int y, float temperature);
void updateOnes(unsigned int x, unsigned int y, float temperature);
void updateTens(unsigned int x, unsigned int y, float temperature);
void updateHundreds(unsigned int x, unsigned int y, float temperature);
void writeLabel(int inOut);
void writeTime(bool forced);
void writeBattery(float voltage);
void checkTouch(float outdoorTemperature, float outdoorHumidity, float indoorTemperature, float indoorHumidity);
void customSetTime();
void printIncrementors();

void setup()
{  
    Serial.begin(9600);
    delay(100);

    pinMode(XM, OUTPUT);
    pinMode(YP, OUTPUT);

    setTime(12, 00, 00, 1, 1, 2018);
    lcdInit();
    radioInit();

    Serial.println("leaving setup");
}

void loop()
{
    static float outdoorTemperature, outdoorHumidity, outdoorPressure, voltage;
    static float indoorTemperature = 21;
    static float indoorHumidity = 47;
    static float indoorPressure;
    static int num = 1;

    writeTime(false);

    if(rf95.available())
    {
        // Should be a message for us now
        uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
        uint8_t len = sizeof(buf);

        if(rf95.recv((uint8_t *)recvPacket.c_str(), &len))
        {
            readDataPacket((char *)recvPacket.c_str(), &outdoorTemperature, &outdoorHumidity,
                          &outdoorPressure, &voltage);
            writeOutdoorTemperature(outdoorTemperature, outdoorHumidity, false);
            num++;
        }
    }

    writeIndoorTemperature(indoorTemperature, indoorHumidity, false);
    checkTouch(outdoorTemperature, outdoorHumidity, indoorTemperature, indoorHumidity);
    writeBattery(voltage);
}

void radioInit()
{
    pinMode(RFM95_RST, OUTPUT);
    digitalWrite(RFM95_RST, HIGH);
    delay(100);

    Serial.println("Feather LoRa TX Test!");
}
// manual reset
digitalWrite(RFM95_RST, LOW);
delay(10);
digitalWrite(RFM95_RST, HIGH);
delay(10);

while(!rf95.init()) {
    Serial.println("LoRa radio init failed");
    while(1);
}
Serial.println("LoRa radio init OK!");

// Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dBm
if(!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while(1);
}
Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on
// The default transmitter power is 13dBm, using PA_BOOST.
// If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
// you can set transmitter powers from 5 to 23 dBm:
rf95.setTxPower(23, false);

void lcdInit() {

display.reset();
uint16_t identifier = display.readID();

if(identifier == 0x8357) {
    Serial.println("Found HX8357D LCD driver");
}
else {
    Serial.println("Chip not found");
    return;
}

display.begin(identifier);
display.setRotation(2);
display.fillScreen(BLACK);
text.begin(display);
}

void bmeInit() {
    Serial.println("before bme");
    if(!bme.begin()) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
    }
while(1);
}

void writeOutdoorTemperature(float temperature, float humidity, bool forced)
{
    static float lastTemp = -1;
    static float lastHumidity = -1;
    unsigned int startX = 125;
    unsigned int startY = 125;

    writeLabel(OUTDOOR);

    setTempColor(temperature); // this sets the font color to color shift based on temperature
    /* convert to farheinheit if desired*/
    if(farheinheit)
        temperature = (temperature * 9 / 5) + 32;

    /* center the temperature */
    temperature = round(temperature);
    /* round the humidity */
    humidity = round(humidity);
    if(temperature > 10)
        startX = 20;
    else if(temperature > 100)
        startX = 0;
    else
        startX = 40;

    /* set the fonts */
    text.setFontMode(1);
    text.setFontDirection(0);
    text.setCursor(startX, startY);
    text.setFontMode(1); // use u8g2 transparent mode (this is default)
    text.setFont(u8g2_font_logisoso92_tn);

    if(temperature != lastTemp || forced)
    {
        if(temperature < 0)
            updateSign(startX, startY, temperature);
        if(temperature > 100)
            updateHundreds(startX, startY, temperature);
        if(temperature > 10)
            updateTens(startX, startY, temperature);
        updateOnes(startX, startY, temperature);

        if(farheinheit)
            text.print("F");
        else
            text.print("C");
    }

    if(humidity > 10)
startX = 200;
else
    startX = 240;
text.setCursor(startX, startY);

text.setForegroundColor(WHITE);
text.setFont(u8g2_font_logisoso58_tn);

if(humidity != lastHumidity || forced)
{
    if(humidity > 10)
        updateTens(startX, startY, humidity);
    updateOnes(startX, startY, humidity);
    text.print("%");
}

startX = text.getCursorX();
startY = text.getCursorY();
display.fillRect(20, startY + 20, 280, 10, WHITE); // make a dividing line

lastTemp = temperature;
lastHumidity = humidity;

}

void writeIndoorTemperature(float temperature, float humidity, bool forced)
{
    static float lastTemp = -1;
    static float lastHumidity = -1;
    unsigned int startX = 125;
    unsigned int startY = 300;

    writeLabel(INDOOR);

    setTempColor(temperature); // this sets the font color to color shift based on temperature
    /* convert to farheinheit if desired*/
    if(farheinheit)
        temperature = (temperature * 9 / 5) + 32;

    /* center the temperature */
    temperature = round(temperature);
    /* round the humidity */
    humidity = round(humidity);
    if(temperature > 10)
        startX = 20;
    else if(temperature > 100)
        startX = 0;
    else
        startX = 40;

    /* set the fonts */
    text.setFontMode(1);
    text.setFontDirection(0);
text.setCursor(startX, startY);
    text.setFontMode(1); // use u8g2 transparent mode (this is default)
text.setFont(u8g2_font_logisoso92_tn);

if(temperature != lastTemp || forced)
{
    if(temperature < 0)
        updateSign(startX, startY, temperature);
    if(temperature > 100)
        updateHundreds(startX, startY, temperature);
    if(temperature > 10)
        updateTens(startX, startY, temperature);
    updateOnes(startX, startY, temperature);
    if(farheinheit)
        text.print("F");
    else
        text.print("C");
}

if(humidity > 10)
    startX = 200;
else
    startX = 240;
text.setCursor(startX, startY);

if(humidity != lastHumidity || forced)
{
    if(humidity > 10)
        updateTens(startX, startY, humidity);
    updateOnes(startX, startY, humidity);
    text.print("");
}

lastTemp = temperature;
lastHumidity = humidity;
}

void writeLabel(int inOut)
{
    int startX, startY;

    text.setFontMode(1);
    text.setFontDirection(0);
    text.setFontMode(1); // use u8g2 transparent mode (this is default)
    text.setFont(u8g2_font_logisoso20_tf);
    text.setForegroundColor(WHITE);

    if(inOut == OUTDOOR)
    {
        startX = 25;
        startY = 25;
    }
```cpp
    text.setCursor(startX, startY);
    text.print("outdoor");

    } else if(inOut == INDOOR)
    {
      startX = 25;
      startY = 200;
      text.setCursor(startX, startY);
      text.print("indoor");
    }
    }

void writeBattery(float voltage)
{
  int startX = 115;
  int startY = 9;

  unsigned int color = WHITE;

  if(voltage < 3.4)
  {
    color = RED;
    display.fillRect(startX + 4, startY + 3, 10, 10, RED);
  }

  if(voltage > 3.4)
    display.fillRect(startX + 4, startY + 3, 10, 10, WHITE);
  if(voltage > 3.6)
    display.fillRect(startX + 16, startY + 3, 10, 10, WHITE);
  if(voltage > 3.8)
    display.fillRect(startX + 29, startY + 3, 10, 10, WHITE);

  display.drawRect(startX, startY, 44, 17, color);
  display.fillRect(startX + 44, startY + 5, 5, 8, color);
}

void setTempColor(float temperature)
{
  if(temperature < 0)
    text.setForegroundColor(BLUE);
  else if(temperature > 0 && temperature <= 15)
    text.setForegroundColor(CYAN);
  else if(temperature > 15 && temperature <= 24)
    text.setForegroundColor(GREEN);
  else if(temperature > 24 && temperature <= 33)
    text.setForegroundColor(YELLOW);
  else if(temperature > 33)
    text.setForegroundColor(RED);
  else
    text.setForegroundColor(WHITE);
}
void updateSign(unsigned int x, unsigned int y, float temperature) {
    if(temperature > 0) {
        text.print("+");
    } else {
        text.print("-");
    }
}

void updateOnes(unsigned int x, unsigned int y, float temperature) {
    x = text.getCursorX();
    y = text.getCursorY();
    display.fillRect(x, y - 95, 75, 110, BLACK);
    text.setCursor(x, y);
    text.print((int)temperature % 10);
    text.setFont(u8g2_font_logisoso58_tf);
    x = text.getCursorX();
    y = text.getCursorY();
    display.fillRect(x, y - 95, 50, 110, BLACK);
    return;
}

void updateTens(unsigned int x, unsigned int y, float temperature) {
    int digit = (int)temperature / 10;
    x = text.getCursorX();
    y = text.getCursorY();
    display.fillRect(x, y - 95, 75, 110, BLACK);
    text.setCursor(x, y);
    text.print(digit);
    return;
}

void updateHundreds(unsigned int x, unsigned int y, float temperature) {
    int digit = (int)temperature / 100;
    x = text.getCursorX();
    y = text.getCursorY();
    display.fillRect(x, y - 95, 75, 110, BLACK);
    text.setCursor(x, y);
    text.print(digit);
    return;
}

void writeTime(bool forced) {
    int currHour, currMinute, currSecond, startX, startY;
    static int lastHour = 100;
    static int lastMinute = 100;
    static int lastSecond = 100;
    currHour = hourFormat12();
    currMinute = minute();
    currSecond = second();
}
startX = 30;
startY = 425;
text.setCursor(startX, startY);

if(currHour != lastHour || currMinute != lastMinute || forced)
{
    text.setForegroundColor(WHITE);
    text.setFont(u8g2_font_logisoso54_tf);
    text.setCursor(30, 450);
    display.fillRect(text.getCursorX(), text.getCursorY() - 75, 70, 80, BLACK); /* clear hours */
    if(currHour < 10)
        text.print(" ");
    text.print(currHour); /* print new hours*/
    display.fillRect(text.getCursorX(), text.getCursorY() - 75, 70, 80, BLACK); /* clear minutes */
    if(currMinute < 10)
        text.print("0");
    text.print(currMinute); /* print new minutes */
    display.fillRect(text.getCursorX(), text.getCursorY() - 75, 100, 80, BLACK); /*clears AM/PM */
    if(isAM())
        text.print("AM");
    else
        text.print("PM");
}

/*if(currSecond != lastSecond)
{
    text.setForegroundColor(WHITE); // this sets the font color to color shift based on temperature
    text.setFont(u8g2_font_logisoso54_tf);
    text.setCursor(30, 450);
    if(currHour < 10)
        text.print(" ");
    text.print(currHour);
    display.fillRect(text.getCursorX(), text.getCursorY() - 75, 20, 80, BLACK);
    _delay_ms(250);
    text.print(":");
}
*/

lastHour = currHour;
lastMinute = currMinute;
lastSecond = currSecond;

}

void readDataPacket(char *recvBuf, float *temperature, float *humidity, float *pressure, float *voltage)
{
    char *token;
    token = strtok(recvBuf, " ");
    *temperature = atof(token);
    token = strtok(NULL, " ");
    *humidity = atof(token);
    token = strtok(NULL, " ");
    *voltage = atof(token);
token = strtok(NULL, " ");
*pressure = atof(token);

void takeMeasurement(float *temperature, float *humidity, float *pressure)
{
    *temperature = bme.readTemperature();
    *humidity = bme.readHumidity();
    *pressure = bme.readPressure();
}

void printMeasurement(float temperature, float humidity, float pressure, float voltage)
{
    Serial.println();
    Serial.print("The temperature is:");
    Serial.println(temperature);
    Serial.print("The humidity is:");
    Serial.println(humidity);
    Serial.print("The pressure is:");
    Serial.println(pressure);
    Serial.print("The voltage is:");
    Serial.println(voltage);
    Serial.println();
}

void checkTouch(float outdoorTemperature, float outdoorHumidity, float indoorTemperature, float indoorHumidity)
{
    static TSPoint point;
    point = ts.getPoint();

    if(point.x > 200 && point.x < 550 && point.y > 150 && point.y < 300)
    {
        delay(500);
        point = ts.getPoint();
        if(point.x > 200 && point.x < 550 && point.y > 150 && point.y < 300)
        {
            if(farheinheit == false)
                farheinheit = true;
            else
                farheinheit = false;
            writeOutdoorTemperature(outdoorTemperature, outdoorHumidity, true);
            writeIndoorTemperature(indoorTemperature, indoorHumidity, true);
        }
    }
    else if(point.x > 250 && point.x < 800 && point.y > 800 && point.y < 950)
    {
        delay(500);
        point = ts.getPoint();
        if(point.x > 250 && point.x < 800 && point.y > 800 && point.y < 950)
        {
            customSetTime();
        }
    }
void customSetTime()
{
    int hourX = 110;
    int hourY = 75;
    int minuteX = 110;
    int minuteY = 225;
    int amX = 25;
    int amY = 350;
    int pmX = 210;
    int pmY = 350;
    int confirmX = 250;
    int confirmY = 440;
    int cancelX = 12;
    int cancelY = 440;
    int lastHour = -1;
    int lastMinute = -1;
    int newHour = hourFormat12(); // get the current hour
    int newMinute = minute(); // get the current minute
    bool confirm = false;
    bool cancel = false;
    bool pm = isPM();
    static TSPoint point;

display.fillScreen(BLACK); // reset the screen
text.setFontMode(1);
text.setFontDirection(0);
text.setFont(u8g2_font_logisoso58_tf);

text.setForegroundColor(WHITE); // set text to white
printIncrementors(); // print the + and - signs
text.setCursor(amX, amY);
if(!isPM())
    display.drawRect(amX - 7, amY - 75, 90, 90, WHITE);
text.print("AM");
text.print(" ");
if(isPM())
    display.drawRect(pmX - 7, pmY - 75, 90, 90, WHITE);
text.print("PM");
text.setFont(u8g2_font_logisoso34_tf);
text.setForegroundColor(RED); // set the text to red
text.setCursor(cancelX, cancelY); // position "cancel"
text.print("cancel ");
text.setForegroundColor(GREEN); // set the text to green
text.print("confirm");

text.setForegroundColor(WHITE);
text.setFont(u8g2_font_logisoso58_tf);

do
{
    if(lastHour != newHour)
    {
        display.fillRect(hourX, hourY - 125, 75, 135, BLACK);
        text.setCursor(hourX, hourY);
        if(newHour < 10)
            text.print("0");
        text.print(newHour);
        lastHour = newHour;
    }
    if(lastMinute != newMinute)
    {
        display.fillRect(minuteX, minuteY - 125, 75, 135, BLACK);
        text.setCursor(minuteX, minuteY);
        if(newMinute < 10)
            text.print("0");
        text.print(newMinute);
        lastMinute = newMinute;
    }
    _delay_ms(100);
    point = ts.getPoint();
    /*while(1)
    {
        point = ts.getPoint();
        Serial.print("x = ");
        Serial.print(point.x);
        Serial.print( "\\\//\// y = ");
        Serial.println(point.y);
        _delay_ms(250);
    }/**/
    // increment the hour
    if(point.x > 750 && point.x < 850 && point.y > 125 && point.y < 225)
    {
        newHour++;
        if(newHour == 13)
            newHour = 1;
    }
    // decrement the hour
    if(point.x > 175 && point.x < 275 && point.y > 125 && point.y < 225)
    {
        newHour--;
        if(newHour == 0)
            newHour = 12;
    }
    // increment the minute
    if(point.x > 750 && point.x < 850 && point.y > 400 && point.y < 500)
    {
        newMinute++;
    }
if(newMinute == 60)
    newMinute = 0;

// decrement the minute
if(point.x > 225 && point.x < 275 && point.y > 400 && point.y < 500)
{
    newMinute--;
    if(newMinute < 0)
        newMinute = 59;
}

if(point.x > 200 && point.x < 400 && point.y > 600 && point.y < 775)
{
    display.drawRect(amX - 7, amY - 75, 90, 90, WHITE);
    display.drawRect(pmX - 7, pmY - 75, 90, 90, BLACK);
    pm = false;
}

if(point.x > 600 && point.x < 850 && point.y > 600 && point.y < 775)
{
    display.drawRect(pmX - 7, pmY - 75, 90, 90, BLACK);
    display.drawRect(pmX - 7, pmY - 75, 90, 90, WHITE);
    pm = true;
}

if(point.x > 550 && point.x < 800 && point.y > 800 && point.y < 950)
    confirm = true;
if(point.x > 250 && point.x < 400 && point.y > 800 && point.y < 950)
    cancel = true;
}
while(!confirm && !cancel);

if(confirm)
{
    Serial.println(newHour);
    if(pm && newHour != 12)
        newHour = newHour + 12;
    else if(!pm && newHour == 12)
        newHour = newHour - 12;
    setTime(newHour, newMinute, 0, 0, 0, 0);
}
display.fillScreen(BLACK);

void printIncrementors()
{
    int startX = 25;
    int startY = 75;
    int right = 225;
    int second = 150;
    text.setFont(u8g2_font_logisoso58_tf);
    display.drawRect(startX, startY-40, 40, 40, WHITE);
    text.setCursor(startX, startY);
    display.drawRect(startX + right, startY-40, 40, 40, WHITE);
    text.print("-");
    text.setCursor(startX + right, startY);
text.print("+");

text.setCursor(startX, startY + second);
display.drawRect(startX, startY + second - 40, 40, 40, WHITE);
text.print("-");
display.drawRect(startX + right, startY + second - 40, 40, 40, WHITE);
text.setCursor(startX + right, startY + second);
text.print("+");
}
Appendix G – Outdoor Node Code

// Feather9x_TX
// -*- mode: C++ -*-
// Example sketch showing how to create a simple messaging client (transmitter)
// with the RH_RF95 class. RH_RF95 class does not provide for addressing or
// reliability, so you should only use RH_RF95 if you do not need the higher
// level messaging abilities.
// It is designed to work with the other example Feather9x_RX

#include <Adafruit_SleepyDog.h>
#include <Adafruit_BME280.h>
#include <SPI.h>
#include <RH_RF95.h>

/* RFM95 Pins */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3

/* BME280 Pins */
#define BME_CS 19
#define BME_SCK 24
#define BME_MISO 23
#define BME_MOSI 22

/* Battery Voltage Pin*/
#define BATVOLT A7

// Change to 434.0 or other frequency, must match RX's freq!
define RF95_FREQ 915.0

/* one minute in milliseconds */
define ONE_MINUTE_MS 50000

// Instance of the RF transceiver
RH_RF95 rf95(RFM95_CS, RFM95_INT);
// Instance of the temperature
Adafruit_BME280 bme;

String sendString;

void radioInit();
void bmeInit();
void takeMeasurement(float *temperature, float *humidity, float *pressure, float *voltage);
bool checkSend(float temperature, float humidity, float lastTemp, float lastHumid);
void printMeasurement(float temperature, float humidity, float pressure, float voltage);
void buildDataPacket(float temperature, float humidity, float pressure, float voltage);
void readDataPacket(char *recvBuf, float *temperature, float *humidity, float *pressure, float *voltage);

void setup()
{
    Serial.begin(9600);
    Watchdog.enable();
    pinMode(LED_BUILTIN, OUTPUT);
radioInit();
bmeInit();

digitalWrite(LED_BUILTIN, HIGH);
delay(200);
digitalWrite(LED_BUILTIN, LOW);
delay(200);
digitalWrite(LED_BUILTIN, HIGH);
delay(200);
digitalWrite(LED_BUILTIN, LOW);
delay(200);
digitalWrite(LED_BUILTIN, HIGH);
delay(200);
digitalWrite(LED_BUILTIN, LOW);

}

void loop()
{
String sendBuf;
static float temperature, humidity, pressure, voltage;
static float lastTemp, lastHumid, lastPress;

takeMeasurement(&temperature, &humidity, &pressure, &voltage);
// printMeasurement(temperature, humidity, pressure, voltage);

// TODO: uncomment this to save power by checking if the measurement is different enough to transmit
if(checkSend(temperature, humidity, lastTemp, lastHumid))
{
    buildDataPacket(temperature, humidity, pressure, voltage);
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
digitalWrite(LED_BUILTIN, LOW);
rf95.send((uint8_t *)sendString.c_str(), sendString.length());
rf95.waitPacketSent();
lastTemp = temperature;
lastHumid = humidity;
lastPress = pressure;
rf95.sleep();
}

for(int i = 0; i < 4; i++)
    Watchdog.sleep(ONE_MINUTE_MS);
}

void radioInit()
{
    pinMode(RFM95_RST, OUTPUT);
digitalWrite(RFM95_RST, HIGH);

delay(100);

    // manual reset
digitalWrite(RFM95_RST, LOW);
delay(10);
digi talWrite(RFM95_RST, HIGH);
delay(10);

while(!rf95.init()) {
    while(1);
}

// Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
if(!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while(1);
}

// you can set transmitter powers from 5 to 23 dBm:
rf95.setTxPower(23, false);
}

void bmeInit()
{
    pinMode(BME_CS, OUTPUT);
    if(!bme.begin()) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while(1);
    }
}

void buildDataPacket(float temperature, float humidity, float pressure, float voltage)
{
    sendString = "";
    sendString.concat(temperature);
    sendString.concat(" ");
    sendString.concat(humidity);
    sendString.concat(" ");
    sendString.concat(voltage);
    sendString.concat(" ");
}

void takeMeasurement(float *temperature, float *humidity, float *pressure, float *voltage)
{
    *temperature = bme.readTemperature();
    *humidity = bme.readHumidity();
    *pressure = bme.readPressure();
    *voltage = 2 * 3.3 * analogRead(BATVOLT) / 1024;
}

bool checkSend(float temperature, float humidity, float lastTemp, float lastHumid)
{
    if(temperature > lastTemp + 0.5 || temperature < lastTemp - 0.5)
        return true;
    else if(humidity > lastHumid + 1 || humidity < lastHumid - 1)
        return true;
    else
        return false;
}

void printMeasurement(float temperature, float humidity, float pressure, float voltage)
{
Serial.println();
Serial.print("The temperature is:");
Serial.println(temperature);
Serial.print("The humidity is:");
Serial.println(humidity);
Serial.print("The pressure is:");
Serial.println(pressure);
Serial.print("The voltage is:");
Serial.println(voltage);
Serial.println();
Serial.println();