An Arduino-Based Weather Station

Introduction

Arduino microcontrollers are inexpensive, widely available, globally supported, open source computing platforms for controlling hardware. There are many environmental monitoring sensors that are easy to interface with Arduino boards. The goal for this project is to construct a basic weather station that:

- measures temperature, relative humidity, and barometric pressure;
- stores data collected at some pre-determined sampling interval, with date and time stamps, for later retrieval;
- uses a solar-charged battery power supply for the station;
- introduces the use of packet radio modules for sending weather station data from one Arduino to another;
- shows how to use an Arduino-compatible high-resolution analog to digital converter for sensors that need resolution better than the on-board 10-bit 0-5 V resolution.

Even if you are used to working with Arduinos, using them for stand-alone outdoor applications poses some additional challenges. I rewrote this document *several* times as I learned more about converting applications that are easy to implement in a familiar indoor breadboard/Arduino UNO environment to systems that minimize power requirements for reliable and sustainable outdoor applications.

This is by no means the only available source of information about Arduino-based weather stations. But, many of the sources I have looked at are short on details, and it is often the smallest details that can facilitate or derail such a project.

The first step is to define the equipment needed to meet the project goals.

Arduino hardware choices

The Arduino UNO is the most widely used board for developing Arduino projects. Although it doesn't require much power to operate, it is not the most energy-efficient Arduino board. Power requirements are important issues for stand-alone applications where you would like your Arduino application to run outdoors away from a plug-in power source, unattended for long periods of time.

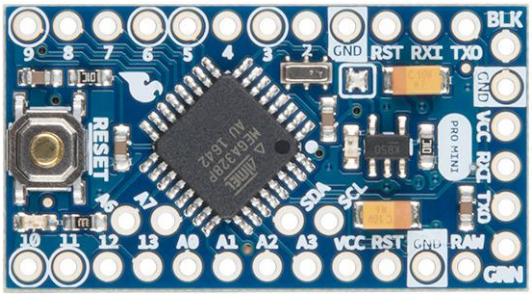
Many Arduino-compatible boards share some on-board hardware:

- microcontroller central processing unit (CPU)
- voltage regulator
- power-on and activity LEDs
- USB interface

Arduino CPUs come in two versions – a 28-pin socket-mounted version (as shown in the UNO board below) and a surface-mount version. Those two versions may have different power requirements. An on-board voltage regulator (not a particularly efficient device) allows power to be supplied from a higher-voltage DC source.[1] 5 V Arduino UNOs and compatibles have a 2.1 mm power-in jack that expects an input voltage in the 7.5-12 V range. A power-on LED serves no essential purpose, but lets you know when the board is under power. Some Arduino boards, including UNOs and compatibles, have an on-board USB interface that allows communication with the Arduino Integrated Development Environment (IDE) when you develop and upload applications. The USB connector can also be used to supply 5 V directly to the board, bypassing the voltage regulator.

Code uploaded to an Arduino (called a "sketch") stays in memory until it is replaced with something else, even when the power is off. Thus, you can develop and upload code when your Arduino board is connected to a computer and then run that code with some other power supply when it is no longer connected to a computer. This is an essential feature for outdoor systems like weather stations.

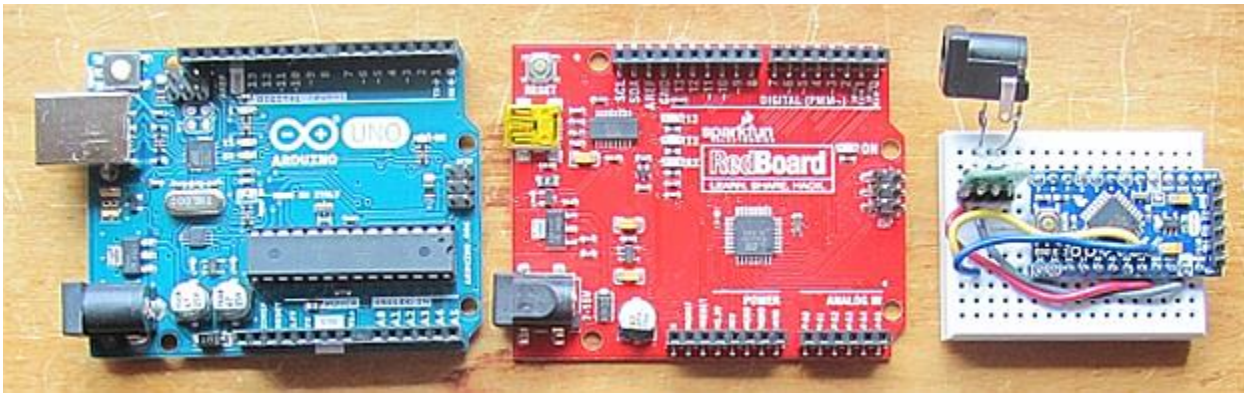
To learn about power requirements, I chose three Arduino boards: an "official" UNO, a UNO-compatible SparkFun RedBoard, and a SparkFun Arduino Pro Mini 328, shown below.[2] My "official" UNO has a socket-mounted processor. The other boards have surface-mount processors. I chose the 5 V version of the Pro Mini, rather than the 3.3 V version, because the Arduino UNO and SparkFun RedBoards are both 5 V boards, because USB cables provide 5 V, and because many sensors require 5 V power or are compatible with 5 V power sources. The Pro Mini does not have an on-board USB communications interface. It requires a separate 6-pin FTDI plug-in board to communicate with a computer. The six connection points for this board are visible at the right end of the Pro Mini board.



The Pro Mini layout is quite different from the other two. Because of its small size, 3.4 cm x 1.9 cm (1.35" x 0.7"), and pin layout, it is not compatible with the many shields that are widely used with Arduino UNOs and their compatibles like the RedBoard. The board comes as shown here. You must solder headers along the top and bottom edges for connecting to a breadboard and a 6-pin header for the FTDI board connection on the right edge. It does not have a power-in jack, but it does include an on-board voltage regulator accessed through the "RAW" connection near the lower right-hand corner of the board

[1] See "Some notes on DC voltage converters (regulators)" below for more about voltage regulators.
[2] Sources for parts mentioned in the text are given in a table later in this document.

The three boards are shown side-by-side here. The blue and yellow wires on the Pro Mini are for the I2C SCL and SDA communication pins, required for many sensors and display devices, which are not in line with the other pins along the edges of the board.



Although you usually power an Arduino UNO or compatible through its 2.1-mm input jack (lower left hand corner) or the USB connector (A to B or mini-B) along the left edge when connected to a computer, it is also possible to connect a ***regulated*** 5 V input directly to the +5 V pin on an UNO or to the VCC pin on a Pro Mini. (Although you would typically use the +5 V pin on a UNO or compatible as a *source* of voltage for powering sensors, you can also at the same time power the board through this pin.) There are two reasons for doing this: (1) to bypass the on-board voltage regulator; (2) to accommodate the Pro Mini board's lack of an on-board USB connection that, if it had one, could be used to power the board.

As noted above, if you don't remove the on-board voltage regulator, you can also power the Pro Mini by applying 7-12 VDC to the RAW pin (*not* the VCC pin!). I didn't do this because the on-board regulator is less efficient than a 5 V "step-down" converter.

The table shows operating current for these three boards. In each case the input voltage to power the board was provided from a plug-in 9 VDC "wall wart" power supply and a Polulu 5 V, 0.3 A D24V3F5 step-down converter. I measured the current by connecting a multimeter between the 5 V input and the +5 V pin on each board. I ran a version of the IDE library's "blink" sketch that turns the on-board pin 13 LED on and off at two-second intervals.

Board	Current, mA LED off/on
Arduino UNO, socketed microcontroller	43/46
Sparkfun RedBoard	22/24
Sparkfun Pro Mini	14/20

For the Pro Mini, it is tempting to disable the power-on LED and voltage regulator; each device requires a little power and serves no essential purpose. In principle, both these devices can be removed from the board. If you don't care about destroying these small surface mount devices,

you can just cut through them with diagonal cutting pliers. I did this with one Pro Mini board (it still worked!), but the difference in operating current wasn't very significant. Considering that it is useful to be able to see the power-on LED, and the possibility of damaging a board by removing components in this way, I continued with an unmodified Pro Mini board, as shown in the breadboard layout below. (The glow from its small power-on LED is visible behind the yellow and blue wires.)

Obviously, there is a *very* clear advantage to using the Pro Mini rather than a UNO board. As will be discussed below, it is possible to further reduce the power requirements through software and hardware.

Sensors for a basic weather station

For this project, I decided to measure three basic weather parameters: temperature, relative humidity, and barometric pressure.

Sensor	Measurement
BME280	temperature (°C or °F); relative humidity, %; absolute pressure, pascals (=100 × millibars)
DHT22	temperature (°C or °F); relative humidity, %

BME280 Specs

DIGITAL HUMIDITY, PRESSURE AND TEMPERATURE SENSOR

Key features

- Package: 2.5 mm x 2.5 mm x 0.93 mm metal lid LGA
- Digital interface: I²C (up to 3.4 MHz) and SPI (3 and 4 wire, up to 10 MHz)
- Supply voltage: V_{DD} main supply voltage range: 1.71 V to 3.6 V; V_{DDIO} interface voltage range: 1.2 V to 3.6 V
- Current consumption: 1.8 µA @ 1 Hz humidity and temperature; 2.8 µA @ 1 Hz pressure and temperature; 3.6 µA @ 1 Hz humidity, pressure and temperature; 0.1 µA in sleep mode
- Operating range: -40…+85 °C, 0…100 % rel. humidity, 300…1100 hPa
- Humidity sensor and pressure sensor can be independently enabled / disabled
- Register and performance compatible to Bosch Sensortec BMP280 digital pressure sensor
- RoHS compliant, halogen-free, MSL1

Key parameters for humidity sensor[1]

- Response time ($\tau_{63\%}$): 1 s
- Accuracy tolerance: ±3 % relative humidity
- Hysteresis: ±1% relative humidity

Key parameters for pressure sensor

- RMS Noise: 0.2 Pa, equiv. to 1.7 cm
- Offset temperature coefficient: ±1.5 Pa/K, equiv. to ±12.6 cm at 1 °C temperature change

DHT22 Specs

Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

Why the redundant temperature and relative humidity measurements? These relatively inexpensive Arduino-compatible sensors are not necessarily high-accuracy devices. Relative humidity, in particular, is difficult to measure accurately. Hence, I wanted to have two different sensors to compare results.

The BME280 board includes a pressure sensor. It measures absolute pressure, sometimes called "station pressure," and not the "weather report" pressure that is always referenced to sea level. Pressure is a function of elevation (altitude). One formula for calculating pressure as a function of elevation, which can be inverted to convert station pressure to weather report (sea level) pressure,

is:

$$P_{station} = P_{sea\ level} \cdot \exp(-0.119h - 0.0013h^2)$$

where pressure P is in millibars and elevation h is in **kilometers**.

This is my own equation based on curve fitting to a table of pressure versus altitude/elevation values. Another calculation is $p_{station}=p_{sea\ level}\cdot(1-2.25577\times10^{-5}h)^{5.25588}$, where h is elevation (altitude) above sea level in **meters**.) If you are looking at weather report data where barometric pressure is given in units of inches of mercury, as is always the case in the U.S., the conversion to mbar is:

$p_{mbar} = p_{"Hg}\cdot1013.25/29.921$

Considerations for outdoor use

An important consideration for an Arduino-based weather station is that Arduinos and compatible sensors are usually not rated for outdoor use where they can be exposed to moisture. Any microcontroller board and its sensors needs to be protected from moisture, whether from direct precipitation or condensation. The former condition is easily handled by keeping electronics in an enclosure shielded from precipitation, but moisture condensation is not as easy to avoid and may be unavoidable under some weather conditions. Relative humidity sensors, in particular, are often not intended for extended use under conditions of 100% relative humidity and may fail or lose calibration if exposed to high RH values over long periods of time. Obviously, it makes no sense to try to "protect" a relative humidity sensor by keeping it in an artificially dry environment as can be done for other sensors. (See **Setting up your station** below.)

Calibration and long-term stability of sensors, inexpensive or not, is always an issue. These issues can be addressed only by trying to protect all electronics from adverse conditions and monitoring performance over time. Ideally, an identical system should be kept indoors as a reference and periodically compared with the system used outdoors.

Software solutions for reducing power consumption

There are two basic software approaches to reducing the power required to run an Arduino board and its application. The first approach involves lowering the speed at which the board's internal clock runs. (It is essentially an elapsed time clock that starts running when you supply power to the board, not a clock that keeps track of hours, minutes, and seconds.) Arduino UNOs and the 5 V Pro Mini board used for this project run at 16 MHz. Through software, it is possible to lower the clock speed in multiples of 2: 8, 4, 2, 1, ... MHz. The lower the clock speed, the less current is required to run the board. Running a board at 1 MHz instead of 16 MHz reduces the current draw by more than a factor of 3.

However, it turns out that lowering the clock speed can create problems with code and devices that make use of the on-board millisecond clock. For example, using `delay(1000)` pauses code execution for 1 s if the board is running at its design speed of 16 MHz, but twice as long if the

speed is cut in half. What I found is that the BME280 sensor is unaffected by running the board at 1 MHz rather than 16 MHz, but the DHT22 sensor no longer works properly. The adafruit.com description for their DHT22 sensor says that it "requires careful timing to grab data." An inquiry to the adafruit.com product forum yielded the additional information that "the DHT22 library uses the microcontroller's `millis()` clock for its timing, and that works by counting ticks of the system clock." In principle, it is possible to write code that tells the board how many ticks there are in a (real) millisecond when the board is running at a slower speed. But, considering that similar problems might arise with other sensors, lowering the board's clock speed doesn't seem like a very good solution.

The second approach to power reduction is to put the Arduino into a "sleep mode" when it's not "doing anything." For this project, with data sampling only every few minutes, the board doesn't have to do anything for most of the time. For code unconcerned with saving power, the `delay()` function is used to pause code execution, But, it is important to understand that the `delay()` function does not put the board in a true sleep mode. It suspends code execution, but doesn't suspend any power-consuming hardware functions. An actual sleep mode would suspend not only code execution but also hardware functions that aren't needed when code isn't being executed. There are libraries available that *will* suspend hardware functions for specified amounts of time. When the board is in sleep mode, the current draw is significantly reduced. When the board is "awake," it runs at the expected clock speed and there are none of the problems that can occur when lowering the clock speed.

The freeware Narcoleptic library's `Narcoleptic.delay()` (search online for "download Narcoleptic library") looks just like the `delay()` function, but it suspends not only code execution but also almost all processor functions for the specified length of time. Here are some results obtained by switching back and forth between the built-in `delay()` function and the `Narcoleptic.delay()` function:

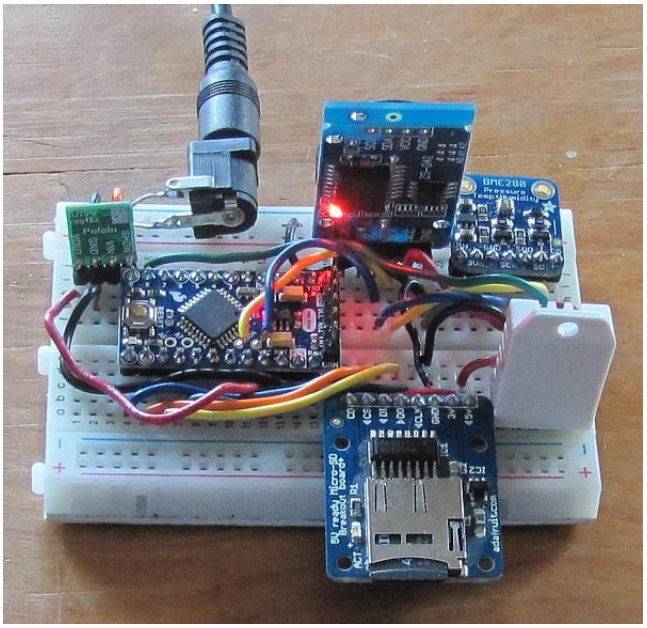
Board	**Current, mA `delay()`/ `Narcoleptic.delay()`**
Sparkfun Pro Mini	16/0.4
Sparkfun RedBoard	23/8

Clearly, when your application isn't "doing anything," with no external devices attached to your board, there is a very significant advantage to using the Narcoleptic library, especially with the Pro Mini. Remember, though, that sensors and other devices connected to the Pro Mini will continue to draw current even when operating under a Narcoleptic software delay.

Recording and logging data

There is little point to measuring weather parameters unless you can record them along with a date and time stamp. With an Arduino UNO or compatible, the easiest way to do this is with a data logging shield like the one from adafruit.com (ID 1141). This device includes both a calendar clock whose date and time is set by your computer clock and an SD card interface for storing data.

Because the Pro Mini doesn't support Arduino shields (it is too small and the pin layouts are different), it is necessary to use separate components for the clock and (micro)SD card rather than a data logging shield that includes both these components and attaches directly to an Arduino UNO or compatible. The separate components require a lot of manual connections that would otherwise be taken care of with a shield. Here is the breadboard layout, with an external 9 VDC power supply and a 5 V step-down converter. (The step-down converter requires an input voltage of at least 6-7 V.)



The clock module is a generic version I happened to have on hand. It is compatible with, but a little larger than, the clock module available from adafruit.com, for example, and it has a relatively large power-on LED that is not necessary. Later, I cut through this power-on LED, which saved a couple of mA. Here are the current draws with the DHT22 and BME280 sensors, real time clock, and micro SD card module:

Board configuration	Current, mA, "sleep"/write
No Narcoleptic library sleep	23/24
Narcoleptic sleep between writes	8/24
Narcoleptic sleep, no clock module power-on LED	6/22

Considering that the board is "sleeping" for most of the time, simply using the Narcoleptic sleep library reduces the power requirements by around 70% or so, depending on the interval between read/write cycles. (The code I wrote collects and writes data roughly every two minutes)

A sharp eye might notice that the right hand pin on the four-pin Polulu step-down converter appears to be tied to the upper left hand pin on the Pro Mini. Because this setup doesn't need that connection, I cut it off so everything would fit on the breadboard.

I also collected some data to determine the current draw for each of the four add-on boards used for this project. I loaded a blank sketch (without putting the board in sleep mode, which didn't matter for this test), monitored the current, and removed the boards one at a time. As long as they are active, the boards will draw current even when the Arduino board is in sleep mode. Of the four boards, The DHT22, BME280, and RTC boards draw 100 µA or less. The SD card draws the most, roughly 1.5 mA. In principle, the DHT22, BME280, and RTC boards can be powered down in between data collection intervals (using a digital pin set HIGH or LOW instead of powering them directly from the Arduino's +5 V pin), to save at most a few hundred µA.

Board configuration	Current, mA
Complete setup	16.6
no DHT22	16.6
no DHT22, SD	15.0
no DHT22, SD, BME280	14.9
no DHT22, SD, BME280, RTC	14.8

Pin connections for the DHT22 are: power, ground, and output to digital pin 8 (for my code). This device needs a 10k pull-up resistor connected between the power and output pins.

For the clock, ground and 5 V, SCL to Arduino SCL, SDA to Arduino SDA. For the BME280, ground and 5 V, SCK to Arduino SCL, SDI to Arduino SDA. Throughout this document, for every circuit that uses an I2C interface, I have always used a yellow wire for the SDA connection and a blue wire for the SCL connection – just a personal choice. Red is always used for power connections and black or gray for ground connections. Colors for other connections may vary from circuit to circuit.

For the micro SD card board:

```
microSD --> Arduino
5V      --> 5V
GND     --> GND
CLK     --> 13
DO      --> 12
DI      --> 11
CS      --> 10
```

Running with the Narcoleptic library, the current draw is about 25 mA when data are being read from the sensors and written to the SD card. In between these times, when `Narcoleptic.delay()` is in effect, the current draw falls to about 10 mA – a very significant power saving for long-term operation!

When I log data, I like to collect measurements at even time intervals. I do this by continuously monitoring the real time clock on an adafruit.com data logging shield. For example, for data sampled at exactly five minute intervals, the code looks for minutes that are evenly divisible by 5 and the second is 0. With Narcoleptic sleep mode, the steps are: sleep for a time a little less than

the desired sampling interval, leave sleep mode, watch the clock until the even five-minute interval arrives, sample and record data, and then sleep again.

If exactly even on-the-minute sampling isn't really important, then the clock-watching step can be eliminated: sleep for a prescribed amount of time, wake up, immediately read the clock, sample and record data, and then go back into sleep mode. This will minimize power consumption over time. Here are some sample data, collected roughly every 2 minutes. I have included in the code a conversion from hours, minutes, and seconds to a fractional day, to provide x-axis values for graphing, and the fact that the data aren't collected at exactly even minute/second intervals doesn't matter. The BME280 and DHT22 temperature and relative humidity values are not exactly the same, as expected. The temperature values usually are close to each other, but the relative humidity values, especially outdoors, can differ by several percent (in relative humidity units rather than as a percentage of value."

	A	B	C	D	E	F	G	H	I	J	K
1	yr	mon	day	hr	min	sec	day_frac	DHT_T(*C)	DHT_RH(%)	BME280_T(*C)	BME280_RH(%)
2	2017	6	29	14	57	30	29.62326	28	52.8	30.45	54.27
3	2017	6	29	14	59	19	29.62453	28	52.8	29.67	49.62
4	2017	6	29	15	1	8	29.62579	28	52.8	29.44	49.99
5	2017	6	29	15	2	57	29.62705	28	52.8	29.31	50.49
6	2017	6	29	15	4	46	29.62831	28	52.8	29.23	50.65
7	2017	6	29	15	6	35	29.62957	28	52.8	29.16	51.17
8	2017	6	29	15	8	23	29.63082	28	52.8	29.11	51.54
9	2017	6	29	15	10	12	29.63208	28	52.8	29.08	51.73
10	2017	6	29	15	12	1	29.63334	28	52.8	29.07	52.04

It is worth noting here that additional power savings can be achieved by powering down the entire Arduino system between data collection and data storage events. This is discussed below, in the section dealing with using packet radio modules to send and receive sensor data. If you do that, the code that will be written for this system, which runs continuously in the `loop()` function once it is powered up, would have to be modified. In power-up/power-down operation the `loop()` function will be empty and all the code to collect and record data will be in the `setup()` function, executed just once every time power is applied to the board.

Solar power for a weather station

A "permanent" outdoor Arduino application requires a reliable self-sustaining power source. One choice is to use solar power with a rechargeable battery. For many years, lead acid batteries were used for such systems. Now, however, lithium-ion polymer (LiPo) batteries are preferred because of their much higher power storage density and other performance characteristics. The drawback for LiPo batteries is that they are usually 3.7 V batteries, so a step-up converter is required to use them as a power source for a 5 V Arduino board.

A solar-based power system requires a solar panel, a battery, and a charge controller. The purpose of the charge controller is to allocate solar resources to power a device and charge a battery – at

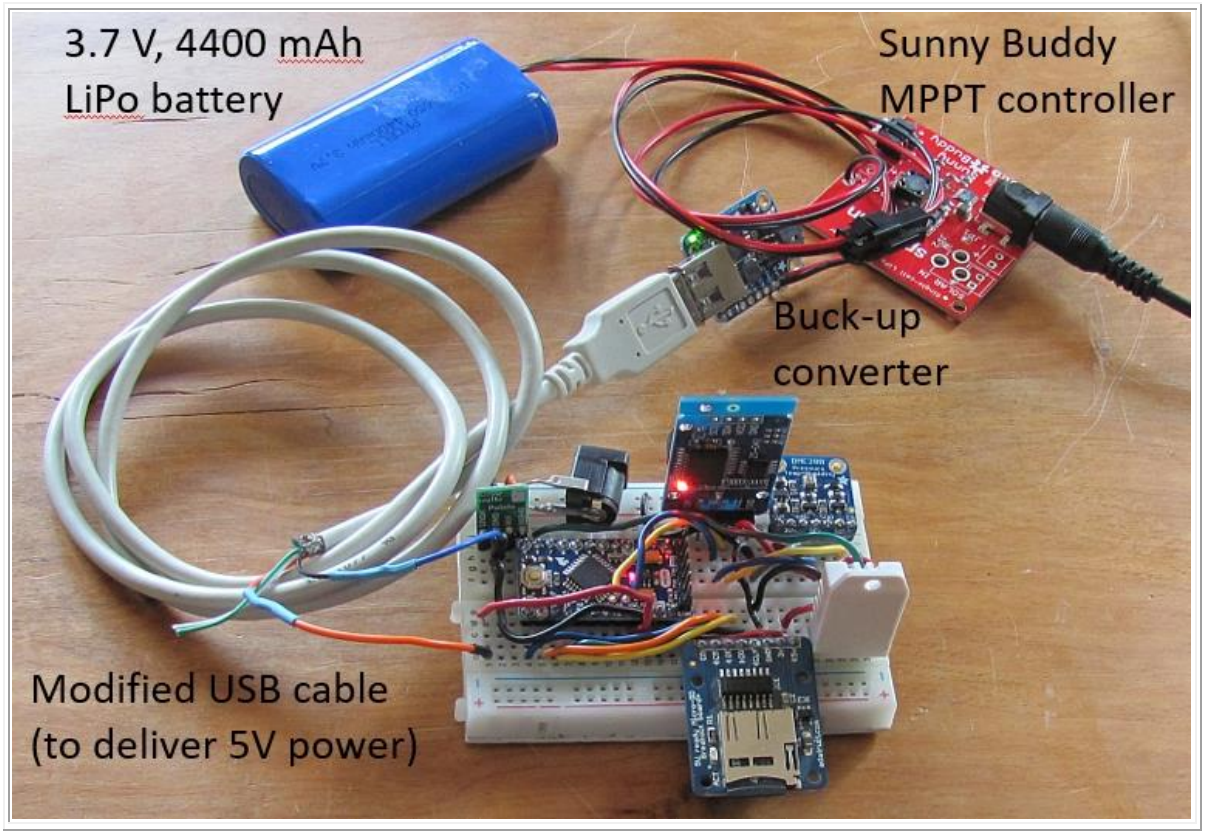
the same time during daylight hours. It is generally believed that the best charge controllers are maximum power point transfer (MPPT) devices that optimize how resources are allocated. (See THIS SITE for a description of MPPT controllers.) During daylight hours, the controller provides power needed for a load and directs the remaining power available from a solar panel to battery charging. At night, the battery completely takes over the task of powering the load. MPPT controllers suitable for Arduino applications are available from www.sparkfun.com. A similar controller available from www.adafruit.com is not a true MPPT device, but Adafruit claims that its design is a better choice for low-power systems.

The success of a solar-powered system requires a battery large enough to power a load continuously based on the ability of the solar panel input to recharge the battery during daylight hours. If the battery capacity is too small, or the solar panel will not recharge the battery with the available sunlight, then the entire system will eventually shut down. The amount of sunlight available varies considerably as a function of weather conditions and, in temperate latitudes, seasonally. A successful system must continue to function even during cloudy winter days and this may pose a significant challenge!

The image below shows the complete weather station layout powered by a sparkfun.com 3.5-W 6 V solar panel and Sunny Buddy solar controller, a 4400 mAh LiPo battery, and an adafruit.com 5 V step-up converter. (The battery may be oversized, or not. Time will tell…) Be sure to read the online material about using the Sunny Buddy, as some initial adjustments need to be made to get the most efficient operation. This photo was taken before I removed the clock module's power-on LED.

For powering the Pro Mini, I already had on hand a USB cable cut open to expose the red and black 5 V and ground wires, to which I soldered jumper wires for connecting to the breadboard. The step-up converter board also has through-holes for soldering wires for the 5 V and ground connections, so you don't have to use the USB jack if it's not convenient.

I left the 2.1-mm power input jack and Polulu step-down converter in place on the board for indoor testing. Another solution for powering the board is to use a step-up/step-down converter that converts input voltage below and above 5 V to 5 V. Then you could power the setup directly from a 3.7 V LiPo battery, several AAA or larger batteries in series, or any DC supply that provides a voltage in a range supported by the converter.



I am aware of the fact that some advanced Arduino users will criticize this setup for not doing all that could be done to minimize the station power requirements. But, from my perspective, all that really matters is that the entire system will continue to operate across a wide range of weather conditions (available sunlight, most importantly). I developed this project during the summer. Performance during winter months will be a challenge both because of reduced sunlight and the decreasing efficiency of LiPo batteries when it is cold. We will see…! And, see below in the section about using packet radios for a much more drastic means of reducing power consumption.

Setting up your station

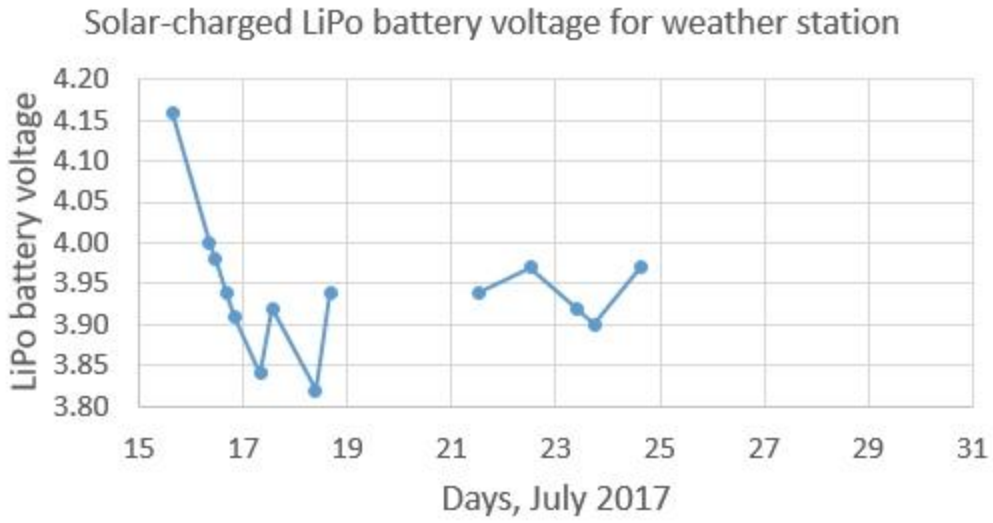
The first priority for an outdoor weather station is protecting the electronics and sensors from moisture – direct precipitation and condensation. The best solution is a ventilated enclosure specifically designed for weather stations. These are referred to as Stevenson screens or enclosures. Unfortunately, Stevenson screens are expensive. A basic white-painted wood shelter from Carolina Biological Supply (www.carolina.com), Item # 745571, costs $177 plus shipping. (Fortunately, I already had one!) The basic design is a rectangular box with louvered sides, a solid back and floor, and a louvered hinged front door. The back can also be louvered. The top slopes so water runs off. In some designs, there is a double roof, separated with spacers by an inch or so, to keep direct solar radiation from impinging directly on the instrument box. Inside the box, there are mounting blocks or standoffs so that sensors aren't in direct contact with any box surface.

Although it is not a particularly simple project, if you have access to woodworking tools you can make a Stevenson screen yourself. Use 3/4" pine or exterior plywood for the frame, roof, and floor, and white commercial louvered shutters for the front, sides and back. You need to be sure that the louvers are angled in a way that allows easy air flow while keeping rain out. Wood is the best material, but vinyl shutters will be much less expensive. If you use commercial shutters, then the size of your box (which is not critical) will be determined by available shutter sizes. Paint everything, inside and out, white.

An alternative to a Stevenson Screen is to mount the Arduino hardware in a moisture-protected enclosure and run cables to sensors in an “inverted pie plate” radiation shield. I do not recommend this because these shields will not protect sensors from wind-blown rain. Neither sensor, with their exposed components and circuitry, should ever be exposed to direct contact with moisture.

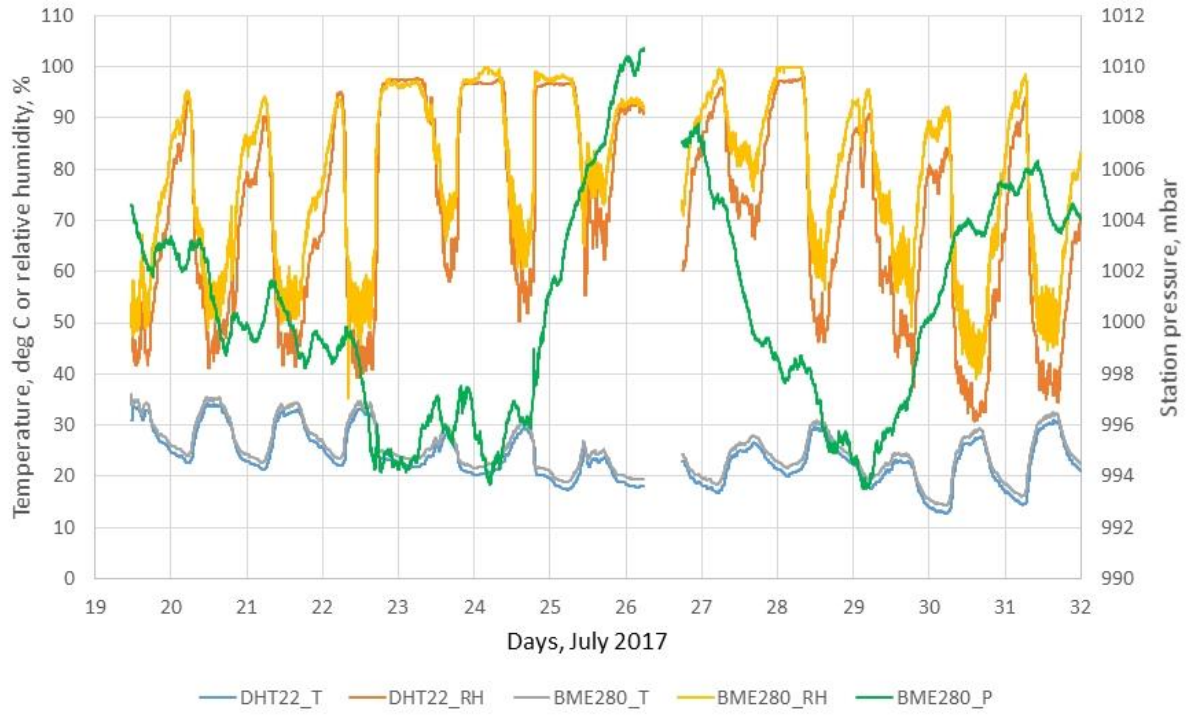
The images below show the weather station as installed. The 3.5-W solar panel is visible partially hidden behind the large homemade pie plate radiation shield (which isn’t used with this station). In the right-hand photo, the red/blue twisted wires are for monitoring the battery voltage.

The graph below shows LiPo battery voltage as a function of time for the station as installed. LiPo batteries are described as 3.7 V batteries. However, the actual starting no-load voltage for this 4400 mAh battery, fully charged indoors with a LiPo charger, turns out to be about 4.16 V. Under load, powering the weather station continuously, the voltage varies between about 3.8 V and 4.0 V during the day. That is, the system as currently configured is self-sustaining at least under these summer sun conditions even when there were several cloudy and overcast days. (The two missing partial day occurred when I took the Pro Mini board setup inside for some additional testing.)



Here are some data collected during July, 2017. The missing data on July 26 were caused not by any equipment failure, but because I took the setup inside for some testing. These data show that the DHT22 and BME280 sensors produce comparable results, but the temperature values are a little different and the relative humidity values (difficult to measure accurately) differ by as much as about 10% (in relative humidity units). The causes of the differences are unknown but, as previously noted, the differences should not be unexpected for these inexpensive sensors.

Remember that the BME280 sensor records actual barometric pressure (station pressure) and not the "weather report" pressure that is referenced to sea level. This site is at an elevation of about 131 m, at which the station pressure is about 15.6 millibars lower than the sea level pressure. Or, to put is another way, a standard sea level atmospheric pressure of 1013.25 millibars is about 997.6 millibars at this elevation.



Arduino code for a basic weather station

This document does not provide an introduction to Arduino programming. For that, you will have to look elsewhere in the literally hundreds of books and online tutorials. I assume you already have access to the freeware Arduino Integrated Development Environment (IDE) on a computer and know how to use it. The real-time clock and SD card boards, and DHT22 and BME280 sensors, all require their own software libraries. If you don't already know how to download and build libraries containing the required `.cpp` and `.h` files, you can get help online. All that is required is to download (free) and copy the required files into appropriately named folders in the Arduino libraries folder when there are no Arduino sketches open. Then, when you open a sketch that requires these libraries, they will automatically be recognized when you include the `.h` file names in your sketch.

The DHT22 sensor is a "one wire" device and the BME280 T/RH/pressure sensor is an I2C device. For the DHT22, "one wire" is in quote marks because this device requires only one data pin for communication with an Arduino board, but it doesn't conform to the widely used Dallas one-wire communication standard. Like All I2C devices, the BME280 uses the SCL and SDA pins for communication. I2C (Inter-Integrated Circuit) is a widely used "two-wire" protocol for communicating with low-speed devices. Arduinos support the I2C protocol and many sensors are designed to use I2C. The details of communicating with an I2C device are contained in the device libraries and are almost never of any concern for an applications-oriented Arduino programmer.

The code I have written uses pre-compile directives to turn output to the serial port or SD card on or off. The serial port option is used for testing the code and, as necessary, to display hardware error messages. When your system works properly indoors and is installed outdoors away from a

computer, turn the serial port option off (`ECHO_TO_SERIAL 0`) and turn the file write option on (`ECHO_TO_FILE 1`).

Here is a code outline:

At top of code:
1. Include all the required libraries.
2. Assign data and communication pins.
3. Initiate the clock and sensor objects.[3]
4. Turn serial and file write compile directives on/off.
5. Define variables for temperature, relative humidity, and pressure from both sensors.
6. Define a log file handle.[4]
7. Define a millisecond delay time.

In `setup`():
1. Begin serial port, clock, DHT, and BME280 communications (I2C devices also require Wire library), with error messages.
2. If `ECHO_TO_FILE` is turned on:
    a. Open SD card communications and assign file name, starting with "`LOGGER00.CSV`" and creating consecutively numbered file names as needed.
    b. Write file header text.
3. If `ECHO_TO_SERIAL` is turned on, write header text to serial port.

In `loop`():
1. Get date/time.
2. Read DHT temperature and relative humidity.
3. Read BME280 temperature, relative humidity, pressure (convert to millibars).
4. If `ECHO_TO_FILE` is turned on:
    a. Write data to SD card.
    b. Delay for specified time – may need multiple calls to a delay function for more than about 30 seconds (2 minutes?).
5. If `ECHO_TO_SERIAL` is turned on:
    a. Write data to serial port.
    b. Delay for specified time.

Before the `setup()` function, all the libraries are included and names are given for accessing the methods needed for the sensors and clock. I have used the Adafruit_BME280 library with the adafruit.com BME280 sensor. The same library also works for the sparkfun.com version of this sensor, the only difference being that the sparkfun version requires +3.3 V power rather than +5 V

[3] This is called "instantiation" in a language that supports objects and methods, like the Arduino language.
[4] A "handle" is the name by which a file will be referred to in code, not the name of the file itself.

pin. (The Pro Mini board doesn't have a 3.3 V output.) Finally, all variables are defined, including a logging file.

In `setup()`, communications with the sensors are initialized. Line 40, which is commented out, can be used to set the RTC clock; this shouldn't be needed once a clock has been set once. If the precompile directives are set for writing to a file, an 8-character file name (the maximum allowed number of characters) is defined, with a `.csv` extension. The name is initially set to `LOGGER00.CSV` and a new consecutively named file is created if files are already stored on the SD card. Note that these Arduino-created files do not contain any of the information you would normally expect about when they were created.

In the `loop()` function, the date and time are read from the clock and the DHT22 and BME280 values read. Output depends on how the precompile directives are set. For writing to a file, the day, hour, minute, and second are converted to a decimal day.

For writing to a file, the `Narcoleptic.delay()` function is used to sleep the Arduino board for a little less than two minutes. Note that 32767 is the largest delay time that can be specified, so longer sleep times require multiple calls to the delay function. At the beginning of `loop()` a short `delay()` function is inserted to make sure that everything is "awake" following the Narcoleptic sleep mode.

As it is, the code shown here creates a new consecutively numbered file every time the system is powered up. You wouldn't want this to happen if you are powering down and restarting the system every couple of minutes because you would be creating hundreds of separate files. In power-up/power-down operation, you could choose a fixed file name. This works when the system is powered down and then powered up again because opening an existing Arduino file in "write" mode appends new data to the end of that existing file. (It is not obvious that file writing will work this way, rather than overwriting an existing new file as happens with other languages such as PHP that have separate file "write" and "append" modes.)

Here is the Arduino code for reporting data from the DHT22 and BME280 sensors. As shown, the precompile directives are set to write data to a file. A text file containing all the code in this document is available at www.instesre.org/ArduinoWeatherStationCode.txt.

DHT22_BME280_2 §

```
/* DHT22_BME280_2.ino, David R. Brooks, June 2017
  See DHT22_BME280.ino for code that logging at even minute/second
    intervals. This code increases sleep time by logging at approximate
    intervals rather than watching the clock for even intervals.
  This code assumes use of separate clock and SD card modules, as
    required for use with a SparkFun Pro Mini board (shields don't fit).
  For the DHT22 sensor:
    pin 1 (on the left) to +5V, pin 2 to DHTPIN (digital pin 8 in this case),
    pin 4 (on the right) to GND, 10K resistor from pin 2 (data) to pin 1
      (power) of the sensor
  For clock and BME280, SCL and SDA.
  For SD module:
    microSD --> Arduino
    CLK     --> 13
    DO      --> 12
    DI      --> 11
    CS      --> 10
*/
#include <Wire.h>
#include <SD.h>
#include <SPI.h>
#include <RTClib.h>
#include <Narcoleptic.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <DHT.h> // for DHT11 or DHT22 sensors
#define ECHO_TO_SERIAL 0
#define ECHO_TO_FILE 1
#define DHTPIN 8      // digital pin for reading DHT22
#define DHTTYPE DHT22  // change to DHT11 with DHT11 sensor
#define SDpin 10       // internally connected pin for SD card
RTC_DS1307 rtc;        // real time clock
Adafruit_BME280 bme;   // I2C device
DHT dht(DHTPIN,DHTTYPE);
float T,RH,P,day_frac,DHT_T,DHT_RH;
File logfile;
const int  msec=29000;
void setup() {
  Serial.begin(9600); Wire.begin(); rtc.begin();
  //rtc.adjust(DateTime(__DATE__, __TIME__));
  if (!rtc.isrunning()) {
    Serial.println("RTC not running."); exit;
  }
  Serial.println("DHT22-BME280 data"); dht.begin(); bme.begin();
  #if ECHO_TO_FILE
    Serial.print("Initializing SD card...");
    pinMode(10,OUTPUT);
    if (!SD.begin(SDpin)) {Serial.println("Card failed.");
      delay(50); exit(0);}
    Serial.println("card initialized.");
    char filename[] = "LOGGER00.CSV";
```

```
    for (uint8_t i = 0; i < 100; i++) {
      filename[6] = i/10 + '0'; filename[7] = i%10 + '0';
      // Create a new file LOGGERxx.CSV, xx= 01 to 99, only if it doesn't already exist.
      if (! SD.exists(filename)) {
        logfile = SD.open(filename, FILE_WRITE);
        if (!logfile) {Serial.println("Couldn't create file."); delay(50); exit(0);}
        break;  // Leave the loop once a file is created.
      }
    }
    logfile=SD.open(filename,FILE_WRITE); Serial.print("Logging to: "); Serial.println(filename);
    logfile.println("yr,mon,day,hr,min,sec,day_frac,DHT_T(*C),DHT_RH(%),BME280_T(*C),BME280_RH(%),BME280_P(mbar)");
    logfile.flush();
  #endif // ECHO_TO_FILE
  #if ECHO_TO_SERIAL
    Serial.println("yr,mon,day,hr,min,sec,DHT_T(*C),DHT_RH(%),BME280_T(*C),BME280_RH(%),BME280_P(mbar)");
  #endif // ECHO_TO_SERIAL
}
void loop() {
  delay(100); // Wait a little after end of Narcoleptic sleep...
  DateTime now=rtc.now();
  DHT_T = dht.readTemperature(); DHT_RH = dht.readHumidity();
  T=bme.readTemperature(); RH=bme.readHumidity(); P=bme.readPressure();
  #if ECHO_TO_SERIAL
    Serial.print(now.year()); Serial.print('/'); Serial.print(now.month()); Serial.print('/');
    Serial.print(now.day()); Serial.print(", "); Serial.print(now.hour()); Serial.print(':');
    Serial.print(now.minute()); Serial.print(':'); Serial.print(now.second());
    Serial.print(", "); Serial.print(DHT_T,1); Serial.print(", "); Serial.print(DHT_RH,1);
    Serial.print(", "); Serial.print(T,1); Serial.print(", "); Serial.print(RH,1);
    Serial.print(", "); Serial.print(P/100,1); Serial.println();
    delay(100); // Give prints time to finish before Narcoleptic sleep.
    Narcoleptic.delay(msec);
  #endif // ECHO_TO_SERIAL
  #if ECHO_TO_FILE
    //V_batt=5.*analogRead(batt_pin)/1023;
    logfile.print(now.year()); logfile.print(','); logfile.print(now.month()); logfile.print(',');
    logfile.print(now.day()); logfile.print(','); logfile.print(now.hour()); logfile.print(',');
    logfile.print(now.minute()); logfile.print(','); logfile.print(now.second()); logfile.print(',');
    delay(10);
    day_frac=now.day()+now.hour()/24.+now.minute()/1440.+now.second()/86400.;
    logfile.print(day_frac,6); logfile.print(','); logfile.print(DHT_T,1);logfile.print(',');
    logfile.print(DHT_RH,1); logfile.print(','); logfile.print(T,2); logfile.print(',');
    logfile.print(RH,2); logfile.print(','); logfile.print(P/100,2); logfile.println(); logfile.flush();
    delay(100); // Give prints time to finish before Narcoleptic sleep.
    Narcoleptic.delay(msec); Narcoleptic.delay(msec);
    Narcoleptic.delay(msec); Narcoleptic.delay(msec); // Sleep for a little while.
  #endif // ECHO_TO_FILE
}
```

Creating a wireless link with packet radio modules

The weather station as implemented above uses a micro SD card to store data in a system that is powered entirely in place, using the solar/battery supply described above. In place of, or in addition to, this approach, it is also possible to establish a wireless link between an outdoor weather station and a receiver located indoors in some convenient location. Both wired and wireless Ethernet boards are available for Arduinos. Wired connections might be convenient in some locations, but outdoor stations typically aren't, and shouldn't be, particularly close to a building. The range of Arduino-based wireless Ethernet communications may be very limited.

For a weather station, where small amounts of data (on the order of 100 bytes or so) need to be transmitted one-way to a receiver at time intervals on the order of minutes apart, high speed wireless connections are not required. The solution for this situation is to use two Arduinos and two identical Arduino-compatible radios – one configured to transmit and the other configured to

receive. For this project, I used two 900 MHz packet radio transceiver modules, ID 3070, from adafruit.com. A pair of these devices costs only $20. An advantage to using packet radios is that the transmitter unit's power supply needs to support only the sensors and radio module; the data storage and real time clock modules are indoors with the receiver rather than requiring power at the outdoor location.

These packet radio modules have transmit/receive ranges measured in the hundreds of meters even with very simple wire antennas (3" (7.6 cm) for 900 MHz operation) and a few intervening walls and even farther with "real" antennas like the ID 3340 from adafruit.com. (Note that this antenna and its required connector for the packet radio board cost more than the board itself.)

Shown below is a breadboard layout for a transmitter using a Pro Mini board with a DHT22 sensor. It is certainly possible to add additional sensors such as the BME280 R/RH/pressure sensor to the transmitter board and include their data in the packet string. (See Upgrading the DHT22 battery-powered low power packet radio board.) The pin connections, which are the same for the transmitter and receiver modules, are:

Radio	→	Arduino
VIN	→	+5V
GND	→	GND
EN	➔	(no connection)
G0	→	D3
SCK	→	D13
MISO	→	D12
MOSI	→	D11
CS	→	D4
RST	→	D2

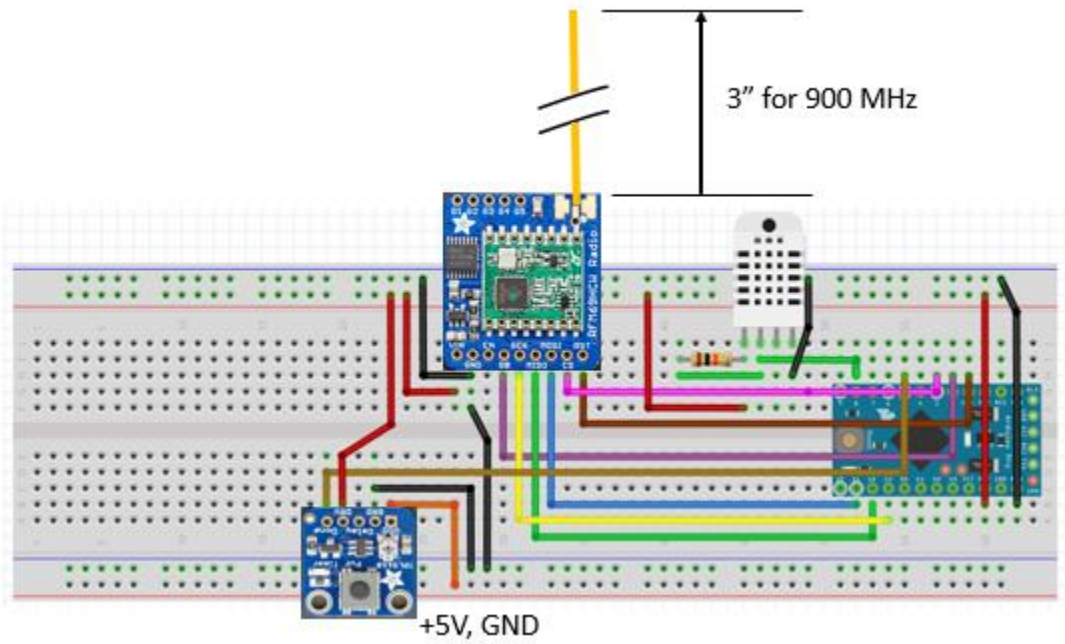
The transmitter board setup requires about 17 mA, most of which is for the radio module. Very significant power savings can be achieved if the Pro Mini board and other hardware connected to it could just be shut down between sampling intervals. As it turns out, this is not difficult to do, using a TPL5110 Low Power Timeout breakout board from adafruit.com (ID 3435). This board, at the bottom left hand corner of the breadboard layout, is inserted between a 5 V power supply and an Arduino board. It powers up the Arduino board with its sensors and radio module at an (approximate) interval set by adjusting an on-board potentiometer; that is, it "passes through" the +5 V input after this interval. (The tiny potentiometer on the board is 200 kΩ, so it is not possible to adjust it precisely to values across that range.) Then, upon receipt of a signal from a digital pin set HIGH on the Arduino board, indicating that data have been collected and the packet has been sent, it shuts down the Arduino board, waits for the prescribed interval, and then powers up again.

Bear in mind that this is not a sleep mode, but an actual cutting of power to the entire Arduino sensor/transmitter setup. In between sampling intervals, the Power Timer breakout board draws less than 100 ***μA***. So, when sampling intervals are measured in minutes and the data collection and packet transmission requires less than a few seconds at most, the power savings are HUGE. It is possible to power such a system for many weeks with a replaceable or rechargeable battery pack rather than using a solar-powered system.

Partial Table of resistance to set TPL5110 sampling interval

(See adafruit.com documentation for more values.)

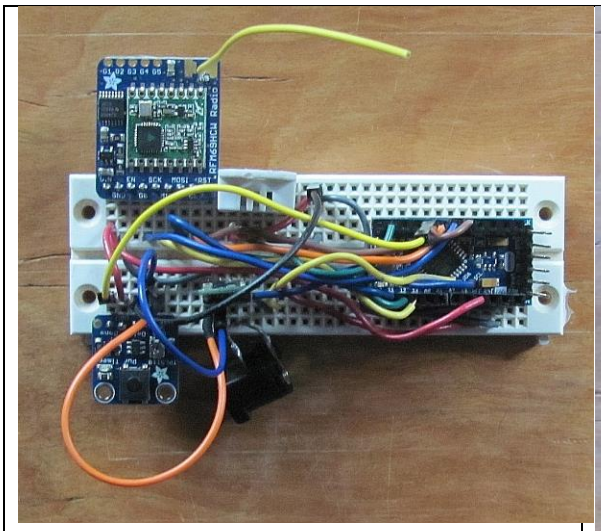
10 Seconds	11.2 kΩ
20 Seconds	14.41 kΩ
40 Seconds	18.75 kΩ
1 Minute	22.0 kΩ
2 Minutes	29.35 kΩ
4 Minutes	39.11 kΩ
6 Minutes	46.29 kΩ
8 Minutes	52.24 kΩ
10 Minutes	57.44 kΩ

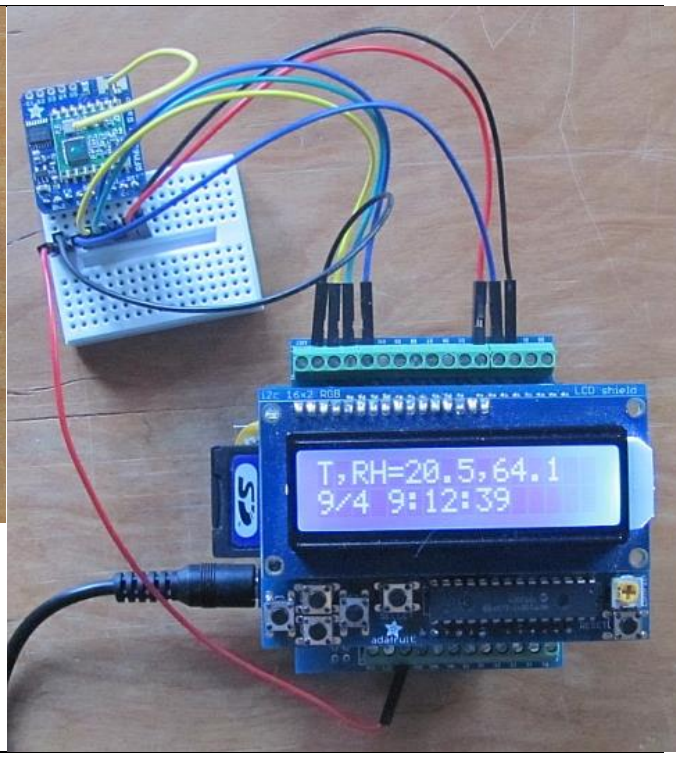


Shown below are assembled breadboards with a transmitter and an UNO-based receiver that includes a screw terminal shield, a liquid crystal display (LCD) from adafruit.com to display temperature and relative humidity values,[5] and an adafruit.com data logging shield. The SD card, which is visible just above the 9 V power in jack, captures data along with a date/time stamp. On both boards, the 3" (7.6cm) unconnected yellow wire is the antenna.)

[5] The adafruit.com I2C LCD module shown here includes some programmable buttons that are not required for this task. Later, I replaced this LCD with a simpler and less expensive ($10) I2C LCD from SunFounder (see "An inexpensive I2C LCD module for displaying data," below)

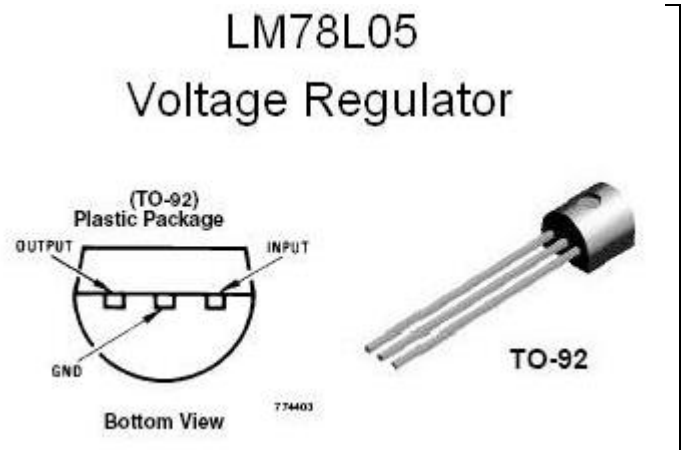
): https://www.sunfounder.com/learn/RFID_kit_V1_for_Arduino/lesson-4-i2c-lcd1602-rfid-v1-for-arduino.html. This device works like the adafruit.com LCD, but it requires a different (downloadable) library.





Just above the 2.1 mm input jack on the transmitter is a 5 V 0.3 A Polulu step-down converter. This converter requires an input voltage of at least 6 V to maintain a 5 V output. I learned from some unfortunate experiences with destroying two TPL5110 boards that the input voltage provided to the TPL5110 should NOT exceed 5 V. Originally, I used a 5 V step-up/step-down converter from Adafruit. As it turned out, this device actually produces a voltage of about 5.2 V. For powering USB devices this is a reasonable choice, to allow for some voltage drop along a long power line. The USB standard specifies 5±0.25 V for powering 5 V devices, so there is no problem with, for example, using such a converter to power a 5 V Arduino board like the Pro Mini. However, this apparently will NOT work with the TPL5110. The Polulu step-down converter shown on the breadboard produces an output voltage of 4.95 V.

An alternative to a step-down converter is the venerable LM78L05 IC linear voltage regulator. These devices, which cost less than $2, are a less efficient than step-down converters. The output voltage can vary a little from source to source, but ones I had on hand produce an output of exactly 5.00 V.



(You should check the voltage with a multimeter before you use one.) The image shows the pin-out configuration for a LM78L05 regulator in a standard TO-92 housing. It works perfectly well in place of a step-down converter for this low-power data logging circuit, although its operating current of about 5 mA is a little higher than the current required by the step-down converter. It requires an input voltage of at least 7 VDC and no more than about 35 VDC.

Step-up/step-down converters *should* be a good idea for battery powered systems because they can work over wide range of voltages. A Polulu 5 V step-up/-step down converter I already had on hand produces an output of 5.15 V, so I suspect it should not be used to power the TPL5110 board. The Polulu S7V8A step-up/step-down converter has an adjustable output, so it would be a good choice.

The packet radio modules are not difficult to use even for a casual programmer once you have installed the required libraries, which hide all the details of the communication protocols. Sources like adafruit.com provide documentation and sample code to walk you through all the setup details.

Code for the packet radio TX and RX modules is shown below. The text for the code is available **HERE**. This code is a stripped down version of the sample code provided for these devices. Basically, the code assumes that the two modules will always connect with each other and that the transmissions will always be received. Most of the trouble-shooting messages that can be sent to the serial port to monitor the performance of the modules have been eliminated because they are of no use once this system is in place and operational.

The packet radio module doesn't send numerical data values directly from sensors. Rather, it sends a string of characters (the "packet"). Line 55 takes the two output values from the DHT22 sensor and converts them to a string using the `dtostrf()` function. Then lines 56-63 build a packet string in which the temperature and relative humidity values are separated by a comma. Line 64 sends the packet. On the receiving end, this string can then be manipulated to display and store the results.

Transmit code for packet radio module.

RadioHead69_DHT22_LowPower

```
/* RadioHead69_DHT22_LowPower, D. Brooks, September 2017
  Puts all the TX code in the setup() function. Turns system on to collect data
  and send packet, then turns system off upon receipt of "done" signal from
  Pro Mini Board, using the TPL5110 low power module from adafruit.com.
*/
#include <SPI.h>
#include <RH_RF69.h>
#include <DHT.h>
/************ Radio Setup ***************/
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 900.0
// for UNO
#define RFM69_INT     3  //
#define RFM69_CS      4  //
#define RFM69_RST     2  // "A"
#define DHTPIN 8
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
RH_RF69 rf69(RFM69_CS, RFM69_INT); // Instantiate radio driver.
float T, RH;
int i;
const int donePIN=5;
char TString[6],RHString[6],RadioPacket[13],S1[6]="T(C)=";char S2[8]=" RH(%)=";
// The encryption key has to be the same as the one in the receiver.
uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08
                };
void setup()
{
  pinMode(donePIN,OUTPUT); digitalWrite(donePIN,LOW);
  Serial.begin(9600); dht.begin();
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, LOW);
  // manual reset
  digitalWrite(RFM69_RST, HIGH);
  delay(10);
  digitalWrite(RFM69_RST, LOW);
  delay(10);
  if (!rf69.init()) {
    Serial.println("RFM69 radio init failed");
    while (1);
  }
```

```
  Serial.println("RFM69 radio init OK!");
  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM (for low power module)
  // No encryption
  if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("setFrequency failed");
  }
  rf69.setTxPower(20, true);  // range from 14-20 for power, 2nd arg must be true for 69HCW
  rf69.setEncryptionKey(key);
  Serial.print("RFM69 radio @");  Serial.print((int)RF69_FREQ);  Serial.println(" MHz");
  // Build text string from T and RH values, comma separated.
  delay(20); T = dht.readTemperature();
  delay(10); RH = dht.readHumidity();
  dtostrf(RH, 5, 2, RHString); dtostrf(T, 5, 2, TString);
  for (i = 0; i <= 4; i++) {
    RadioPacket[i] = TString[i];
  }
  RadioPacket[5] = ','; RadioPacket[6] = ' ';
  for (i = 7; i <= 11; i++) {
    RadioPacket[i] = RHString[i - 7];
  }
  RadioPacket[12] = '\0';
  rf69.send((uint8_t *)RadioPacket, strlen(RadioPacket));
  rf69.waitPacketSent();
  // Now wait for a reply
  uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
  if (rf69.waitAvailableTimeout(500))  {
    // Should be a reply message for us now
    if (rf69.recv(buf, &len)) {
      Serial.print("Got a reply: "); Serial.println((char*)buf);
    } else {
      Serial.println("Receive failed");
    }
  } else {
    Serial.println("No reply, is another RFM69 listening?");
  }
  delay(50); digitalWrite(donePIN,HIGH); // turn off power
}
void loop() {
}
```

Receive code for packet radio module.

RadioHead69_ReceiveTRH_LCD_SD §

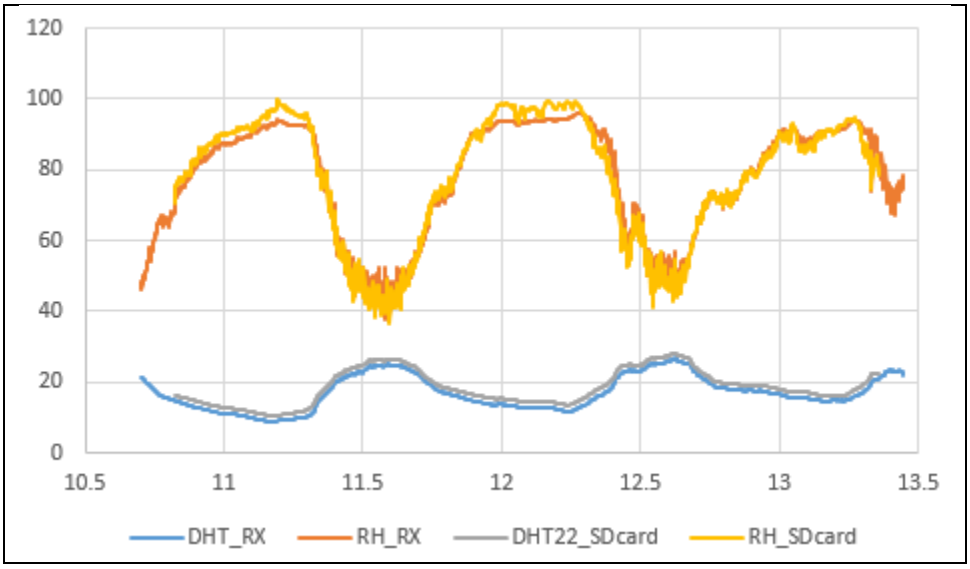
```
/*
  RadioHead69_ReceiveTRH_LCD_SD.ino, D. Brooks, September 2017
  A modification of the example RX code for the RH-RF69 packet radio module.
  Receives a text string from the TX module (see RadioHead69_DHT22_LowPower.ino)
  and formats it for display on a 2-line LCD and saving on an SD card with
  date/time stamp using datalogging shield, both from adafruit.com.
*/
#include <SPI.h>
#include <Wire.h>
#include <SD.h>
#include <RTClib.h>
#include <Adafruit_RGBLCDShield.h>
#include <RH_RF69.h>
#define RF69_FREQ 900.0
#define ON 0x1
#define SDpin 10
#define RFM69_INT     3
#define RFM69_CS      4
#define RFM69_RST     2
#define LED           13
RTC_DS1307 rtc;
Adafruit_RGBLCDShield lcd=Adafruit_RGBLCDShield();
RH_RF69 rf69(RFM69_CS, RFM69_INT);
char L1_text[6]="T,RH=";
int i;
File logfile;
char filename[]="LOG00000.CSV";
void setup()
{
  Serial.begin(9600); SD.begin(); Wire.begin(); rtc.begin();
  logfile=SD.open(filename,FILE_WRITE);
  lcd.begin(16,2); lcd.setBacklight(ON);
  pinMode(LED, OUTPUT);
  pinMode(RFM69_RST, OUTPUT); digitalWrite(RFM69_RST, LOW);
  // manual reset
  digitalWrite(RFM69_RST, HIGH); delay(10);
  digitalWrite(RFM69_RST, LOW); delay(10);
  if (!rf69.init()) {
    Serial.println("RFM69 radio init failed");
    while (1);
  }
  Serial.println("RFM69 radio init OK!");
  if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("setFrequency failed");
  }
  rf69.setTxPower(20, true);  // range from 14-20 for power,
  // The encryption key has to be the same as the one in the TX.
  uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
  rf69.setEncryptionKey(key);
```

```
  pinMode(LED, OUTPUT);
  Serial.print("RFM69 radio @"); Serial.print((int)RF69_FREQ);
  Serial.println(" MHz");
}
void loop() {
 if (rf69.available()) {
    // Should be a message for us now
    uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (rf69.recv(buf, &len)) {
      if (!len) return;
      buf[len] = 0;
      Serial.println(); Serial.print(L1_text);
      lcd.clear(); lcd.print(L1_text);
      for (i=0; i<4; i++) {
        Serial.print((char)buf[i]);
        lcd.print((char)buf[i]);
        logfile.print((char)buf[i]);
      }
      Serial.println(); lcd.print(',');
      //lcd.setCursor(0,1); lcd.print(L2_text); Serial.print(L2_text);
      logfile.print(',');
      for (i=7; i<=10; i++) {
        Serial.print((char)buf[i]);
        lcd.print((char)buf[i]);
        logfile.print((char)buf[i]);
      }
      Serial.println(); logfile.print(',');
      DateTime now=rtc.now();
      logfile.print(now.year()); logfile.print(',');
      logfile.print(now.month()); logfile.print(',');
      logfile.print(now.day()); logfile.print(',');
      logfile.print(now.hour()); logfile.print(',');
      logfile.print(now.minute()); logfile.print(',');
      logfile.print(now.second()); logfile.print(',');
      logfile.println(); logfile.flush(); lcd.setCursor(0,1);
      lcd.print(now.month()); lcd.print('/');
      lcd.print(now.day()); lcd.print(' ');
      lcd.print(now.hour()); lcd.print(':');
      lcd.print(now.minute()); lcd.print(':');
      lcd.print(now.second());
    } else {
      lcd.print("Receive failed");
    }
  }
}
```

Shown below is a graph of September, 2017, temperature and relative humidity data from the DHT22 sensor on the self-contained weather station data recording system described earlier in this document and data sent from a DHT22 sensor on the packet radio board to an indoor receiver. Both setups are inside a Stevenson screen, about 25 m from my second floor office. The data are transmitted through the wall of the wooden enclosure and then through three stud walls to my office. The outer house wall is stucco installed over 6” stud framing and an open mesh metal lath on exterior plywood. I haven’t tested the limits over which data can be transmitted and received reliably with these devices. In any event, the limits will always depend not just on straight-line but

also on what kinds of structural barriers are in place between the transmitter and receiver to block the line of sight.

The point of showing these data side-by-side is to see whether there was any difference in the DHT22 values when they are powered continuously on the self-contained solar/battery weather station board with SD card storage or powered up intermittently roughly every two minutes on the packet radio transmitter board. The data shown here are all consistent with the variability expected among DHT22 sensors, confirming that there is no problem with intermittently turning this device on and off.



Some notes on DC voltage converters (regulators)

For all the circuits described in this document, and for all Arduino-based applications, the voltage sources used to power these devices are very important components. All the hardware used in this document work on 5 VDC power. Some of the sensors may require 3.3 V instead of 5 V, but when the sensors are mounted on "plug in" boards, as adafruit.com does, those boards may include voltage shifter circuits to convert 5 V input to the required 3.3 V. Note that these voltage requirements apply most often to digital sensors. Analog devices like TMPxx temperature sensors will often operate correctly with a range of input voltages – see the spec sheet for the device.

3.3 V and 5 V devices are NOT interchangeable unless specifically allowed. For example, the adafruit.com BME280 sensor board was used with the 5 V Pro Mini board rather than the sparkfun.com BME280 sensor board. Why? Because of this very clear and specific warning on the website for this device:

The BME280 is a 3.3V device! Supplying voltages greater than ~3.6V can permanently damage the IC. As long as your Arduino has a 3.3V supply output, and you're OK with using I^2C, you shouldn't need any extra level shifting. But if you want to use SPI, you may need a Logic Level Converter.

If a data collection system were using an Arduino UNO or compatible, the sparkfun.com BME280 board could be used because UNOs have a 3.3 V output pin. The Pro Mini does not.

There are four basic power supply options for Arduinos and their attached accessories:

1. USB cable when connected to a computer;
2. 9 V "wall wart" power supply connected through the 2.1mm power-in jack on UNO and compatible boards;
3. battery supply through the 2.1mm power-in jack;
4. "wall wart" or battery supply through a DC-to-DC voltage converter.

Obviously, options 1 and 2 are available only for indoor use or outdoor locations with access to a 120VAC (in the U.S.) outlet.

Arduino UNOs and many other Arduino boards have on-board an integrated circuit (IC) voltage regulator to convert voltages greater than about 7 V and no more than 12 V (or maybe a little higher) down to the 5 V required to operate the boards. These devices, like the venerable 78L05 IC, are "linear voltage regulators." They work by using a self-adjusting voltage divider to maintain a constant output voltage. They are very reliable and very inexpensive and they have been used in circuits for *many* years. They are also not very efficient because they dissipate the difference between the input and output voltages as heat. The more current drawn by the circuit being powered, the more power – P=IV – must be dissipated, where V is the difference between the input and output voltages.

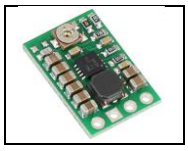
The inefficiency of linear regulators is typically not much of a problem for "indoor" devices, but they are more problematic for outdoor systems away from AC outlets and USB connections. The alternative is to use "buck" DC-to-DC voltage converters, which have efficiencies in the 90+% range. For an explanation of how they work, see, for example, https://en.wikipedia.org/wiki/Buck_converter. Buck converters come in step-up, step-down, or step-up/step-down versions. Usually the output voltages are fixed, but versions are available with adjustable output voltage. For the projects described in this document, I have at various times used buck converters from www.adafruit.com and www.polulu.com.You can find the input voltage range requirements for these devices on the supplier website.

Polulu D24V3F5	5 V, 300 mA step-down
Polulu D24V5F5	5 V, 500 mA step-down (replacement for D24V3F5 with better performance specs)
Polulu U1V10F5	5 V, 200 mA step-up
Polulu S10V4F5	5 V, 400 mA step-up/step-down
Polulu S7V8A	adjustable output voltage, 1A step-up/step-down
Adafruit VERTER	5 V, 500 mA step-up/step-down
Adafruit PowerBoost	5 V, 500 mA step-up

An important caveat for using these devices is that the output voltages for the "fixed" output devices may not be exactly 5 V, for example. The output voltage may depend somewhat on the

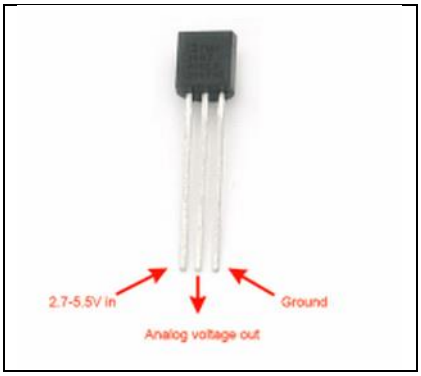
current drawn by the load. This *may* cause problems. As noted above, the adafruit.com low power timer appeared to fail for any input voltage above 5 V.

Step-up/step-down devices are especially useful for battery-powered systems because they provide a constant output voltage over a wide range of input voltages above and below the output voltage. The Adafruit VERTER "5 V" step-up/step-down regulator actually produces an output of 5.2 V. This is OK for 5 V USB devices like Arduino UNOs, for which the specification is 5±0.25 V, but it is *not* OK for the low power timer board!



The solar/LiPo battery supply Pro Mini-based weather station module uses an Adafruit PowerBoost regulator to boost the <4 V LiPo voltage up to the required 5 V because it's what I had on hand at the time. But now, my device of choice for all the projects described in this document is the Polulu S7V8A adjustable output voltage regulator, shown here. You need a digital multimeter and a small screwdriver to adjust the output voltage with the small potentiometer at the top of the board. Once you have set the output, it is a good idea to cover the top of the potentiometer with a small blob of bathtub caulk to keep it from moving.

A 16-bit A/D board for analog sensors



Neither of the two sensors used in the above weather station (DHT22 and BME280) are straightforward analog sensors. Analog sensors, like the well-known TMP35 and TMP36 temperature sensors, require a power and ground connection and produce an analog voltage as output. For connection to an Arduino analog pin, this output voltage needs to be between 0-5 VDC.

There are many sensors that work this way, but whose outputs (including TMP35/36 sensors) are well under 5 V. Because the Arduino on-board A/D resolution is only 10 bits, this may limit the usefulness of such sensors. Consider a pyranometer[6] that produces an output of about 250 mV in full summer sunlight – an insolation of about 1000 W/m^2 – or about 250 µV/(W/m^2). The 0-250 mV output voltage range is found in pyranometers available from the Institute for Earth Science Research and Education and Apogee Instruments, for example. The output from the widely used Kipp & Zonen SP-Lite pyranometer is even smaller, around 70 µV/(W/m^2).

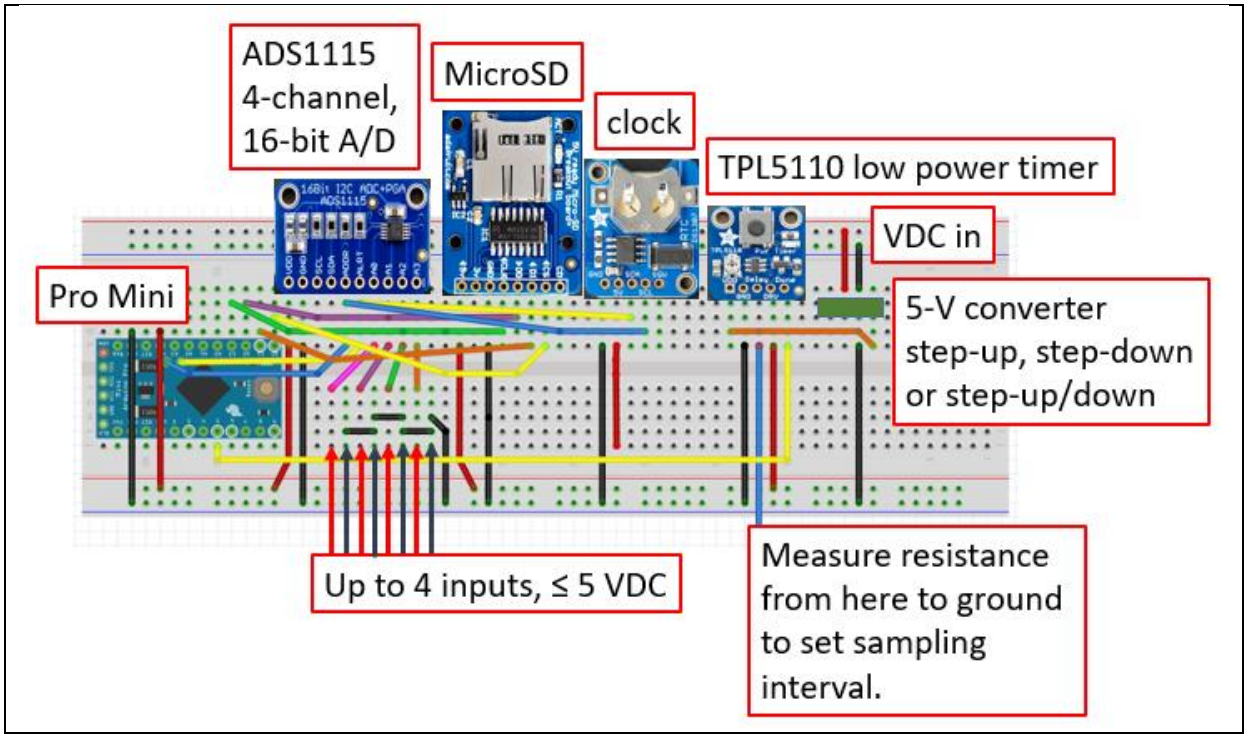
Arduino's 10-bit A/D resolution divides 0-5 V into 1024 integer values from 0 to 1023, so the resolution is 5/1023≈5 mV. That resolution divides the 0-250 mV, 0-1000 W/m^2 pyranometer output range into 50 (roughly) 20-W/m^2 intervals. This is *not* sufficient resolution for this measurement. (This resolution is OK, but not great, for TMP35/36 sensors.) A solution to the Arduino's sometimes too-coarse on-board A/D resolution is to use a high-resolution Arduino-compatible add-on board like the ADS1115 from adafruit.com. This I2C device provides 4

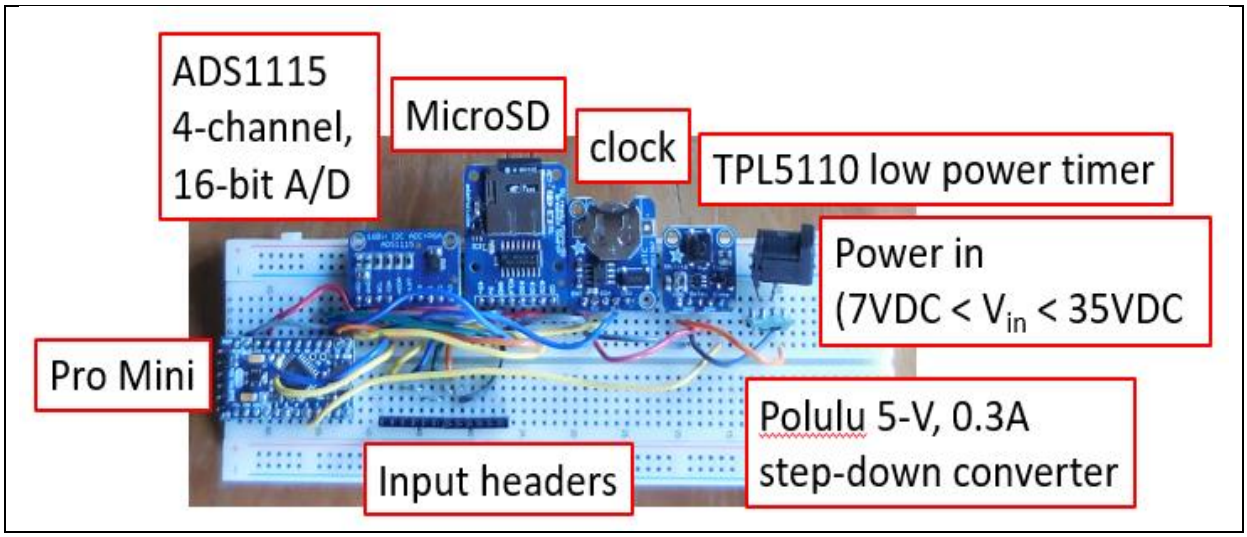
[6] A pyranometer is an instrument that measures insolation, the total incoming solar radiation on a horizontal surface, with metric units of W/m^2. Silicon photodiode-based instruments are widely used but are more accurately described as "surrogate" pyranometers because they do not respond uniformly across the entire solar energy spectrum.

channels of 16-bit A/D conversion, with programmable gain. (All four channels have the same programmed gain.)

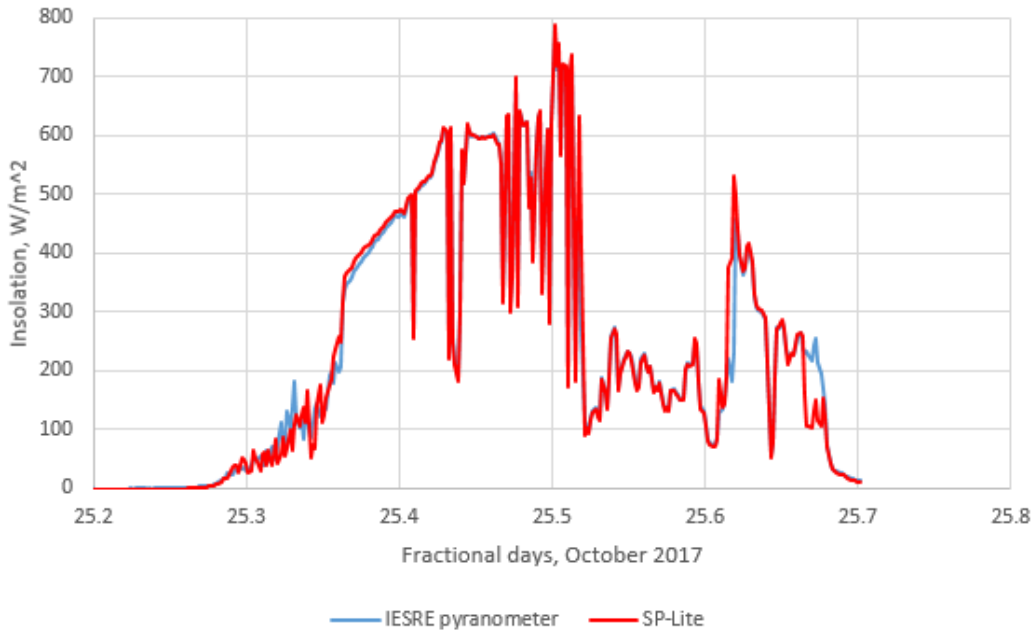
The image below shows a Pro Mini-based breadboard layout using an ADS1115. The top image shows a breadboard schematic and the lower image shows the actual breadboard, on which the wiring I used is functdionally equivalent to, but different from, the schematic. I used an 840 contact breadboard, PB-850, from allelectronics.com. Rather than relying on solar/battery power for outdoor use, this system uses the adafruit.com low power timer board powered by a Polulu 5 V step-down converter, collecting data every couple of minutes. Indoors, the power can come from a 9 VDC "wall wart." Outdoors, as noted above, the system will run for a long time on a battery pack (providing no less than 6 V) because it is powered on for only a couple of seconds between much longer sampling intervals. The pyranometers mentioned above are passive devices using silicon photodiode detectors. They do not require any exsternal power source or "warm up" time before collecting data.

The whole system, which requires a lot of wiring on a breadboard, would be much simpler with an Arduino UNO and a data logging shield from adafruit.com. The ADS1115, step-down converter, and low power timer boards would still need their own breadboard, although there might be enough room on the adafruit data logging shield's prototyping space to mount all the hardware. As noted above, you could replace the step-down converter with a step-up/step-down converter





Here are some data collected using channels 0 and 1 on the ADS1115 board. The graph compares instantaneous insolation values (total incoming solar radiation) collected at two-minute intervals from a commercial Kipp & Zonen SP-Lite pyranometer (67.5 µV/(W/m^2)) and an inexpensive pyranometer developed by the Institute for Earth Science Research and Education (~260 µV/(W/m^2)) (http://www.instesre.org/Solar/Solar_home_page.htm). After a relatively clear sky in the morning (there is some shadowing from trees early in the morning), clouds started to move in later in the morning, with overcast skies after noon. At this location, (40N, -75W), the clear sky insolation at solar noon on October 25 is about 625 W/m^2. Reflections from the sides of clouds can produce instantaneous insolation values well above the clear sky value. The maximum output voltage from the SP-Lite during this test was only about 50 mV and the maximum output from the IESRE pyranometer was just under 200 mV.



Here is the code for the ADS1115 data logger. The `Serial.print()` statements used for testing are commented out, as is the code required to read ADS1115 channels 2 and 3.

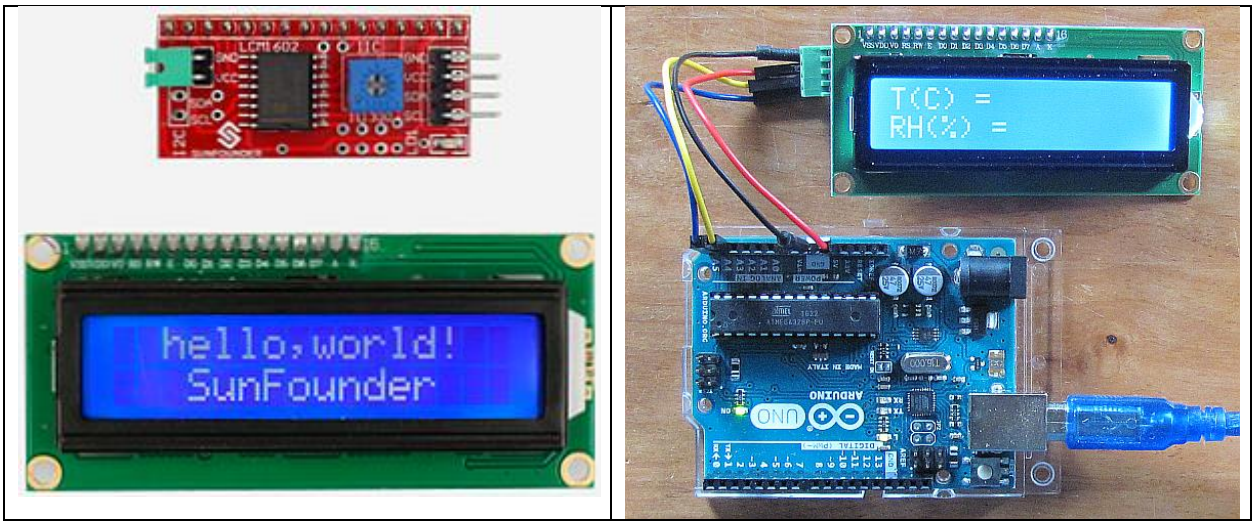
```
/*
  ADC1115_LowPowerTest.ino, D. Brooks, Nov. 2017
  Uses adafruit low power timer board to collect insolation data
  from two pyranometes.
  Connections on Pro Mini:
  "Done" to digital 5
  "DRV" to VCC
*/#include <Wire.h>
#include <SD.h>
#include <SPI.h>
#include <RTClib.h>
#include <Adafruit_ADS1015.h>
Adafruit_ADS1115 ads(0x48); // default address is 0x48
RTC_DS1307 rtc;
float V0,V1;
//float V2,V3;;
//float DtoA=0.015625/1000.;
float DtoA=0.007813/1000;
const int donePin=5,SDpin=10; // SDpin fixed for the microSD board.
File logfile;
char filename[]="LOG00000.CSV";
void setup() {
  //Serial.begin(9600);
  // Power is applied...
  pinMode(donePin,OUTPUT);
  digitalWrite(donePin,LOW); // power on
  //Serial.println("low power pin LOW");
  //Serial.begin(9600);
  Wire.begin(); rtc.begin();
  if (!rtc.isrunning()) {
    //Serial.println("RTC not running.");
    exit;
  }
  //rtc.adjust(DateTime(__DATE__, __TIME__));
  //Serial.print("Initializing SD card...");
  pinMode(10,OUTPUT);
  if (!SD.begin(SDpin)) {
    //Serial.println("Card failed.");
  delay(50);exit(0);}
  //Serial.println("card initialized.");
  ads.begin();
  //ads.setGain(GAIN_EIGHT); // full scale = 0.512 V
  ads.setGain(GAIN_SIXTEEN);
  int16_t adc0,adc1; // ADC reading produces 16-bit integers
  //int16_t adc2,adc3;
  adc0 = ads.readADC_SingleEnded(0);
  adc1 = ads.readADC_SingleEnded(1);
  //adc2 = ads.readADC_SingleEnded(2);
  //adc3 = ads.readADC_SingleEnded(3);
  V0 = adc0 * DtoA;
  V1 = adc1 * DtoA;
  //V2 = adc2 * DtoA;
  //V3 = adc3 * DtoA;
  //Serial.print(adc0); Serial.print(" V0=");
```

```
  //Serial.print(V0, 4); Serial.print(' ');
  //Serial.print(adc1); Serial.print(" V1=");
  //Serial.print(V1, 4); Serial.print(' ');
  //Serial.print(adc2); Serial.print(" V2=");
  //Serial.print(V2, 4); Serial.print(' ');
  //Serial.print(adc3); Serial.print(" V3=");
  //Serial.print(V3, 4); Serial.print(' ');
  //Serial.println();
  DateTime now=rtc.now();
  logfile=SD.open(filename,FILE_WRITE);
  logfile.print(now.year()); logfile.print(',');
  logfile.print(now.month()); logfile.print(',');
  logfile.print(now.day()); logfile.print(',');
  logfile.print(now.hour()); logfile.print(',');
  logfile.print(now.minute()); logfile.print(',');
  logfile.print(now.second()); logfile.print(',');
  logfile.print(V0,5); logfile.print(',');
  logfile.print(V1,5); logfile.print(',');
  //logfile.print(V2,5); logfile.print(',');
  //logfile.print(V3,5);
  logfile.println();
  logfile.flush();
  delay(250); logfile.close(); delay(250);
  digitalWrite(donePin,HIGH); // power off
  //Serial.println("low power pin HIGH");
}
void loop() {}
```

An inexpensive I2C LCD board for displaying data

Here is a 5 V two-line 16 character LCD board, the SunFounder I2C LCD1602 (https://www.sunfounder.com/i2clcd.html). It is a less expensive alternative (US$10) to the adafruit.com I2C LCD used in the packet radio system described above (without the unneeded programmable buttons), and it has the added advantage of not requiring any assembly. The red board, which comes already mounted on the back of the display board, provides the I2C interface. A four-line 20 character version of this board is also available (https://www.sunfounder.com/catalogsearch/result/?q=20x4+lcd, US$15).

The blue potentiometer is for adjusting the backlighting. I soldered a 4-pin screw terminal block to the four-pin connector on the I2C board so I could easily attach jumper or other wires to an Arduino. Because this LCD is an I2C device it requires only two pins for control. In this example, I used A4 and A5 for the SDA and SCL pins.



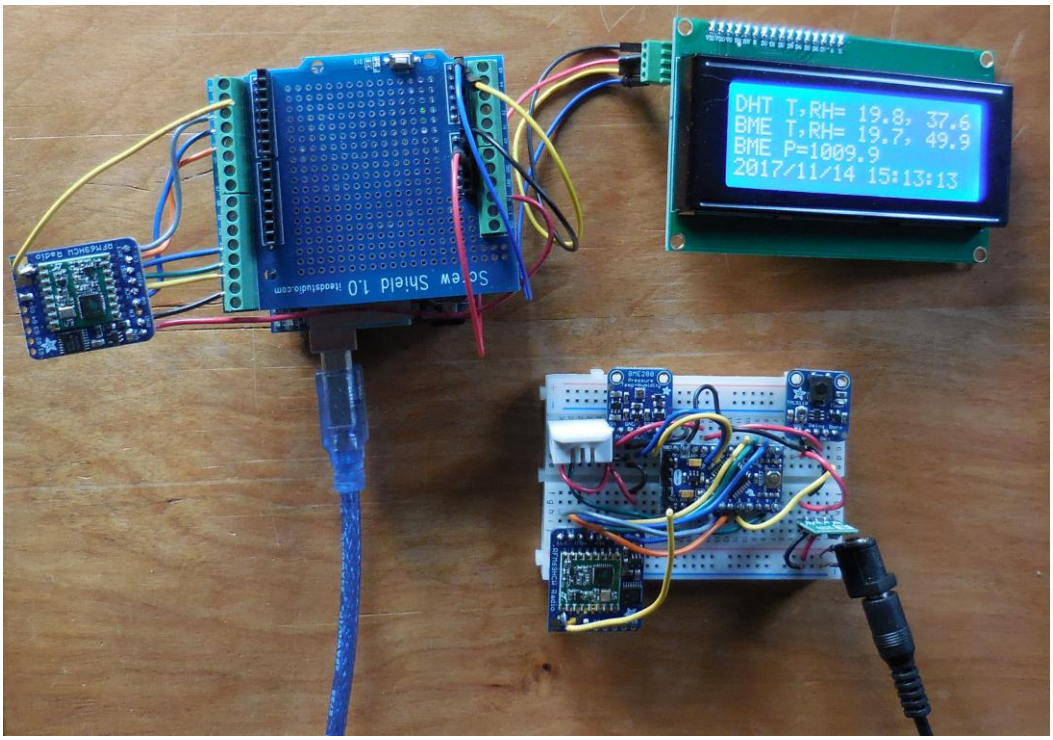
The code below shows how to use the SunFounder device with the `LiquidCrystal_I2C` library downloadable from SunFounder's website.

```
/* LCD1602_I2C_template, D. Brooks, September 2017
  Email:support@sunfounder.com
  Website:www.sunfounder.com
*/
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2); /LCD address is 0x27 for 16 char, 2 line display
char line1[]="T(C) = ";
char line2[]="RH(%) = ";
int col1,col2;
void setup()
{
  Serial.begin(9600);
  lcd.init(); //initialize the lcd
  lcd.backlight(); //turn on the backlight
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(line1);
  lcd.setCursor(0,1);
  lcd.print(line2);
}
void loop() {}
```

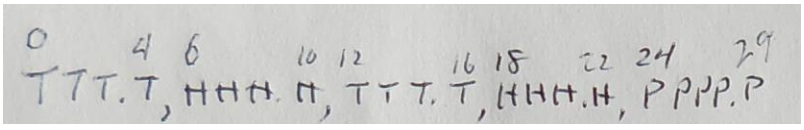
Upgrading the DHT22 battery-powered low power packet radio board

Based on success with the Pro Mini/DHT22 sensor board powered with a TPL1550 low power timer board and 6 D cell battery pack, I replaced that setup with a transmitter/receiver setup that includes output from a BME280 sensor. This has a negligible impact on the power requirements because the sensors draw only a tiny current. Because of the additional sensor information I used the SunFounder 20x4 LCD mentioned above on the receiving end. The only setup change this requires is replacing the `LiquidCrystal_I2C lcd(0x27,16,2);` statement with `LiquidCrystal_I2C lcd(0x27,20,4);`.

Here's the hardware. The transmitter's low power timer board (upper right hand corner of the transmitter breadboard) uses an adjustable-output Polulu step-up/step-down DC voltage converter (lower right hand corner), set for an output just under 5 V. The receiver setup includes an adafruit data logging shield under the screw terminal shield. For the indoor test running when this photo was taken, the data logging code was commented out, so there isn't any SD card in the slot (visible just above the power-in jack). Rather than installing headers and using breadboard space for the packet radio board, I soldered wires and connected them to the screw terminal shield. As is usually the case (as I have learned), the BME280 and DHT22 relative humidity values are significantly different but the temperature values are close.



The code changes involve first laying out the contents of the packet string. I just drew this little sketch. The values are separated by commas. The first two sets of data are DHT22 temperature and relative humidity and the last three sets are temperature, relative humidity, and pressure from the BME280 sensor. The `TTT.T` format allows for negative (Celsius) temperatures and the `HHH.H` format allows for 100% relative humidity. Atmospheric pressures at low elevations are around 1000 millibars. Character 30 is the invisible "end-of-string" marker, `'\0'`.



Here is the transmitter code:

```
/* RadioHead69_DHT22_BME280_TXLowPower, D. Brooks, November 2017
  Puts all the TX code in the setup() function. Turns system on to collect data
  and send packet, then turns system off upon receipt of "done" signal from
```

```
  Pro Mini Board, using the TPL5110 low power module from adafruit.com.
*/
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <RH_RF69.h>
#include <DHT.h>
/************ Radio Setup ***************/
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 900.0
// for UNO
#define RFM69_INT     3  //
#define RFM69_CS      4  //
#define RFM69_RST     2  //
#define DHTPIN 8
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
Adafruit_BME280 bme;
RH_RF69 rf69(RFM69_CS, RFM69_INT); // Instantiate radio driver.
float T, RH,T_BME,RH_BME,P_BME;
int i;
const int donePIN=5;
char RadioPacket[30];
char TString[6],RHString[6]; // DHT22
char T_BMEString[6],RH_BMEString[6],P_BMEString[7]; // BME280
// The encryption key has to be the same as the one in the receiver.
uint8_t key[] = { 0x02, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x02, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08
                };
void setup()
{
  pinMode(donePIN,OUTPUT); digitalWrite(donePIN,LOW);
  Serial.begin(9600); dht.begin();
  bme.begin();
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, LOW);
  // manual reset
  digitalWrite(RFM69_RST, HIGH);
  delay(10);
  digitalWrite(RFM69_RST, LOW);
  delay(10);
  if (!rf69.init()) {
    Serial.println("RFM69 radio init failed");
    while (1);
  }
  Serial.println("RFM69 radio init OK!");
  // Defaults after init are 434.0MHz,
  // modulation GFSK_Rb250Fd250,
  // +13dbM (for low power module)
  // No encryption
  if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("setFrequency failed");
  }
  rf69.setTxPower(20, true);  // range from 14-20 for power
  rf69.setEncryptionKey(key);
  Serial.print("RFM69 radio @");  Serial.print((int)RF69_FREQ);
  Serial.println(" MHz");
  // Build text string from T and RH values, comma separated.
  // read DHT22
  delay(2000); T = dht.readTemperature(); delay(20);
  delay(10); RH = dht.readHumidity(); delay(20); RH-dht.readHumidity();
  dtostrf(RH, 5, 1, RHString); dtostrf(T, 5, 1, TString);
```

```
  // read BME280
  T_BME=bme.readTemperature(); RH_BME=bme.readHumidity();
  P_BME=bme.readPressure();
  // Serial.print(T_BME); Serial.print(' '); Serial.print(RH_BME);
  // Serial.print(' ');
  // Serial.println(P_BME);
  dtostrf(RH_BME,5,1,RH_BMEString);dtostrf(T_BME,5,1,T_BMEString);
  dtostrf(P_BME/100.,6,1,P_BMEString);
  // DHT22 part of packet string
  for (i = 0; i <= 4; i++) {
    RadioPacket[i] = TString[i];
  }
  RadioPacket[5] = ',';
  for (i = 6; i <= 10; i++) {
    RadioPacket[i] = RHString[i - 6];
  }
  // BME280 part of packet string
  RadioPacket[11]=',';
  for (i=12; i<=16; i++) {
    RadioPacket[i] = T_BMEString[i-12];
  }
  RadioPacket[17]=',';
  for (i=18; i<=22; i++) {
    RadioPacket[i] = RH_BMEString[i-18];
  }
  RadioPacket[23] =',';
  for (i=24; i<=29; i++) {
    RadioPacket[i] = P_BMEString[i-24];
  }
  // string terminator
  RadioPacket[30] = '\0';
  // Test contents of RadioPacket string.
  //for (i=0; i<30; i++) {
  //  Serial.print(RadioPacket[i]);
  //}
  //Serial.println();
  rf69.send((uint8_t *)RadioPacket, strlen(RadioPacket));
  rf69.waitPacketSent();
  // Now wait for a reply
  uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
  if (rf69.waitAvailableTimeout(500))  {
    // Should be a reply message for us now
    if (rf69.recv(buf, &len)) {
      Serial.print("Got a reply: "); Serial.println((char*)buf);
    } else {
      Serial.println("Receive failed");
    }
  } else {
    Serial.println("No reply, is another RFM69 listening?");
  }
  delay(50); digitalWrite(donePIN,HIGH); // turn off power
}
void loop() {
}
```

Here is the receiver code:

```
/*
  RadioHead69_ReceiveTRH_LCD_SD.ino, D. Brooks, November 2017
  A modification of the example RX code for the RH-RF69 packet radio module.
  Receives a text string from the TX module (see RadioHead69_DHT22_LowPower.ino)
```

```
  and formats it for display on a 2-line LCD and saving on an SD card with
  date/time stamp using datalogging shield, both from adafruit.com.
*/
#include <SPI.h>
#include <Wire.h>
#include <SD.h>
#include <RTClib.h>
#include <LiquidCrystal_I2C.h>
#include <RH_RF69.h>
#define RF69_FREQ 900.0

//#define SDpin 10
#define RFM69_INT     3
#define RFM69_CS      4
#define RFM69_RST     2
#define LED           13
RTC_DS1307 rtc;
LiquidCrystal_I2C lcd(0x27,20,4); // set the LCD address to 0x27 and specify 20x4
display
RH_RF69 rf69(RFM69_CS, RFM69_INT);
char BMEL2_text[10]=   "BME T,RH=";
char DHTL1_text[10]="DHT T,RH=";
int i;
//File logfile;
//char filename[]="LOG00000.CSV";
void setup()
{
  lcd.init(); lcd.backlight();
  Serial.begin(9600);
  //SD.begin();
  Wire.begin(); rtc.begin();
  //logfile=SD.open(filename,FILE_WRITE);
  pinMode(LED, OUTPUT);
  pinMode(RFM69_RST, OUTPUT); digitalWrite(RFM69_RST, LOW);
  // manual reset
  digitalWrite(RFM69_RST, HIGH); delay(10);
  digitalWrite(RFM69_RST, LOW); delay(10);
  if (!rf69.init()) {
    Serial.println("RFM69 radio init failed");
    while (1);
  }
  Serial.println("RFM69 radio init OK!");
  if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("setFrequency failed");
  }
  rf69.setTxPower(20, true);  // range from 14-20 for power,
  // The encryption key has to be the same as the one in the TX.
  uint8_t key[] = { 0x02, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                    0x02, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
  rf69.setEncryptionKey(key);
  pinMode(LED, OUTPUT);
  Serial.print("RFM69 radio @"); Serial.print((int)RF69_FREQ);
  Serial.println(" MHz");
}
void loop() {
 if (rf69.available()) {
    // Should be a message for us now
    uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (rf69.recv(buf, &len)) {
      if (!len) return;
      buf[len] = 0;
      for (i=0; i<len; i++) {
```

```
        Serial.print((char)buf[i]);
      }
      Serial.println(); Serial.print(DHTL1_text);
//-----------------------------------------------
      // Display DHT22 T, RH.
      lcd.clear(); lcd.setCursor(0,0);lcd.print(DHTL1_text);
      for(i=0; i<=10; i++) {
        lcd.print((char)buf[i]);
      }
      // Display BME280, T, RH.
      lcd.setCursor(0,1); lcd.print(BMEL2_text);
      for (i=12; i<=22; i++) {
        lcd.print((char)buf[i]);
      }
      // Display BME280 P.
      lcd.setCursor(0,2); lcd.print("BME P=");
      for (i=24; i<=29; i++) {
        //Serial.print((char)buf[i]);
        lcd.print((char)buf[i]);
        // logfile.print((char)buf[i]);
      }
      //Serial.println(); logfile.print(',');
//-------------------------------------------
      // Display date/time.
      DateTime now=rtc.now();
      //logfile.print(now.year()); logfile.print(',');
      //logfile.print(now.month()); logfile.print(',');
      //logfile.print(now.day()); logfile.print(',');
      //logfile.print(now.hour()); logfile.print(',');
      //logfile.print(now.minute()); logfile.print(',');
      //logfile.print(now.second()); logfile.print(',');
      //logfile.println(); logfile.flush();
      lcd.setCursor(0,3);
      lcd.print(now.year()); lcd.print('/');
      lcd.print(now.month()); lcd.print('/');
      lcd.print(now.day()); lcd.print(' ');
      lcd.print(now.hour()); lcd.print(':');
      lcd.print(now.minute()); lcd.print(':');
      lcd.print(now.second());
      Serial.print(now.year()); Serial.print('/');
      Serial.print(now.month()); Serial.print('/');
      Serial.print(now.day()); Serial.print(' ');
      Serial.print(now.hour()); Serial.print(':');
      Serial.print(now.minute()); Serial.print(':');
      Serial.println(now.second()); Serial.println();
    } else {
      lcd.print("Receive failed");
    }
  }
}
```

Equipment list and sources for an Arduino-based weather station

The table below lists the equipment I used for this weather station project. If you are already using Arduinos you may have some of what you need. Otherwise, an investment significantly beyond just the station hardware will be required. Some of the items are available from several different sources. SparkFun and “official” Arduino devices, for example, are widely available online. Because the entire hardware Arduino enterprise is open sourced, as noted above there are many unofficial clones available online, mostly from Chinese sources. Although I have sometimes used

these “no-brand” boards, they can cause compatibility problems and sometimes appear to be of lower quality than hardware from primary sources.

Practical considerations aside, there is also the question of supporting the truly remarkable Arduino enterprise. Innovative companies like SparkFun (www.sparkfun.com) and Adafruit (www.adafruit.com) have made microcontrollers and sensors reliable and inexpensive. In addition to hardware, they have invested heavily in software and project development and support. Adafruit, in particular, provides exceptionally detailed tutorials and software for all their products. Both SparkFun and Adafruit support a wide range of user forums, which I have often used whenever I had hardware or software questions.

My advice, then, is to support reliable developers and vendors and resist the understandable temptation to use very inexpensive Arduino clones. (A SparkFun Pro Mini board costs only $10.) Having said that, I also recognize that the hardware choices I have made for this project are not necessarily the only reasonable choices. This equipment list shows only what I have done (sometimes using a comparable part I already had on hand) and is not intended to exclude other choices.

Some random thoughts about the equipment list:

- It is a good idea to have two LiPo batteries and keep one of them charged indoors to replace the one in your station if needed during weather that prevents the weather station from being fully charged during the day. Cold weather, in particular, is hard on LiPo batteries and they may not charge properly. I use a charger from adafruit.com that connects to a USB source and has a 2-pin JST jack for LiPo batteries (the standard connector for such batteries). For the size batteries required for this project, there is a jumper that should be shorted out to raise the charging current to 500 mA – see the adafruit.com documentation for this device. (Also, see the note about soldering, below.)

- In general, for working with Arduino projects you will need an assortment of USB cables. For example, Arduino UNOs use a type A-to-B cable, SparkFun RedBoards use a mini-B connector and the Adafruit LiPo charger uses a micro-B connector. Type B cables are commonly used for printers. Mini-B cables are used for some digital cameras. Many smartphones and some digital cameras use micro-B cables.

- Even though there is no mention in the equipment list, you *will* need some soldering equipment. For example, the Pro Mini board and many sensors come without header pins. To use any of these devices on a breadboard, or to make other direct connections, soldering is required. Use 1/32" (1 mm) or smaller resin core solder and a soldering iron with a small pointed tip. A small (25-40 watt) iron is OK, but the non-replaceable tips often found on very cheap soldering irons tend to be too big to use for the kind of soldering required for working with Arduinos, sensors, and other accessories. You can use lead-free solder if you want, but it is more expensive than tin/lead solder and more difficult to use because it requires higher temperatures to flow onto a connection.

  Soldering is a necessary electronics skill to develop. It is not difficult to overheat and destroy small sensitive electronics devices with a soldering iron. It takes practice to

minimize the amount of heating time by working quickly and precisely. If you have never soldered anything, get some help from someone who has, before working with any component that's important or expensive!

- Like other Arduino UNO-compatible shields, the screw terminal shields from robotshop.com and other sources will not work with the Pro Mini board. However, it is very useful for trying out sensors with UNO-compatible boards. The headers on such boards will accept 22 AWG solid wire, but not stranded wire or much smaller or larger wires of any kind. To work with such sensors, a screw terminal shield that accepts a range of wire sizes, including stranded wire, and provides access to all the board's input pins is the solution. The robotshop.com screw terminal board also has its own headers, so you can stack Arduino UNO shields on it as needed.

- Currently, multi-GB SD cards have become the standard. However, 2 GB cards are now quite inexpensive from surplus electronics distributors such as allelectronics.com and they will hold a LOT of data from sensors. More expensive large-capacity cards simply aren't required for this project.

- For the PowerBoost step-up converter, you don't have to use the USB connector like I did. You can just solder red (+) and black (-, ground) wires to the board.

- If you don't already have one, you should get a digital multimeter for checking voltages and currents. I use 50LE multimeters from kelvin.com. These are less than $4 each and less than $3 in quantities of 10. They are so inexpensive that I think of them as "throwaway" instruments that are perfectly adequate for these kinds of projects. (They are excellent for classroom use.) It is useful to have two, with the probe tips on one replaced with small alligator clips.

- Although jumper wires in various lengths are commonly used for breadboarding circuits, for semi-permanent breadboard layouts, it is useful to have some rolls of 22 AWG solid copper wire with insulation in a variety of colors. This is the recommended wire size for breadboards and the headers on Arduino boards. (See the photos of the Pro Mini mounted on a breadboard.)

Item	Approximate price
Station Components	
Arduino Integrated Development Environment (IDE)	freeware
SparkFun Pro Mini microcontroller, sparkfun.com DEV-11113 or adafruit.com ID 2378	$10
SparkFun FTDI Basic 5 V Breakout, sparkfun.com DEV-09716 or adafruit.com ID 284	$15
DS1307 real time clock board, adafruit.com ID 3296	$8
microSD card board, adafruit.com ID 254	$8

400 tie point breadboard, robotshop.com RB-Cix-11	$4
1 GB micro SD card with SD adapter, allelectronics.com CAT#s MSD-1, SDR-6	$3.25
BME280 temperature/relative humidity/barometric pressure sensor breakout board, adafruit.com ID 2652, or SparkFun.com SEN-13676	$20
DHT22 temperature/relative humidity sensor with 10kΩ pull-up resistor, adafruit.com ID 385	$9
Solar power supply components	
6 V solar panel, 3.5W, adafruit.com ID 500 or sparkfun.com PRT-13782	$39
Lithium Ion Polymer (LiPo) solar battery charger/controller, sparkfun.com "Sunny Buddy" PRT-12885 ($25) or adafruit.com ID 390 ($18)	$18-$25
3.7 V, 4400 mAh LiPo battery pack, adafruit.com ID 354 or 6600 mAh, adafruit.com ID 353	$20 or $30
PowerBoost 500 Basic DC/DC step-up converter, adafruit.com ID 372, or 5 V step-up/step-down converter, adafruit.com ID 2190	$10
Components for assembly and testing	
Polulu 5 V, 0.5A D24V3F5 step-down converter, polulu.com (This item is being replaced by the D24V5F5 converter, which is more efficient and requires an input voltage only a few hundred millivolts above 5 V.)	$5
2.1 mm barrel jack, robotshop.com RB-Spa-1102	$1
9 V battery to 2.1 mm barrel jack connector, robotshop.com RB-Dfr-100, or 9 V 650 mA wall adapter power supply, robotshop.com RB-Spa-103	$1.50 or $6
assorted M/M jumper wires, robotshop.com RB-Cix-01	$4
USB cable with type A connector, robotshop.com RB-Cyt-172 (optional, modified for powering Pro Mini from step-up converter)	$1
LiPo USB battery charger for indoor use, adafruit.com ID 1904	$7
screw terminal shield for Arduino Uno, robotshop.com RB-lte-127	$4
digital multimeter, kelvin.com	$4
22 AWG solid copper wire, 6 x 25 ft, various colors, e.g., adafruit.com ID 1311	$16

Components for packet radio project	
Receiver: Arduino UNO or SparkFun RedBoard	$20-$25
Transmitter: Pro Mini board (see weather station parts list)	$10
Packet radio transceivers (2), adafruit.com ID 3070	$10 × 2 = $20
16 x 2 I2C LCD shield, adafruit.com ID 772 (unassembled, with considerable soldering required), or	$10-$20

SunFounder 16 x 2 I2C LCD, assembled	
data logging shield, adafruit.com ID 1141, plus CR1220 coin cell battery for clock, ID 380	$15
Polulu D24V3F5 5 V step-down converter (6 V minimum input), or Polulu S7V8A 5 V (adjustable) step-up/step-down converter	$5-$6
400 tie point breadboard, robotshop.com RB-Cix-11 or allelectronics.com PB-850 840 tie point breadboard	$4-$8

Online sources:

As noted previously, there are *many* sources for Arduino-related hardware, thanks to the fact that the entire Arduino "ecosystem," both hardware and software, is open-sourced. But, unless you live in a very large city you will almost certainly be unable to buy what you need locally. These are reliable online sources that I have used. They are not the only sources for much of the hardware used for this project. However, I reiterate the need for caution when using online sources without established product support. Accessible technical support is especially important for sensors and other devices that require software libraries.

Suggested sources
adafruit.com, sensors and other components
allelectronics.com, miscellaneous electronics and hardware
sparkfun.com, sensors, RedBoard, Pro Mini board
robotshop.com, Arduino UNOs, screw terminal shields, sensors, and other components
www.polulu.com ,www.adafruit.com, step-down, step-up, or step-up/step-down converters to provide 5 V for Pro Mini
www.sunfounder.com/i2clcd.html for I2C 16 × 2 and 20 × 4 LCDs
kelvin.com, inexpensive digital multimeter
Narcoleptic library, for shutting down some hardware functions between data sampling (free download)
https://www.arduino.cc/en/Main/Software, Arduino IDE (free download)

Some ongoing implementation notes

As noted above, the Adafruit low power timer board used for the packet radio transmitter is VERY sensitive to its power supply voltage and no more than 5 V should be applied as input. A six D cell battery pack supplying about 8.8 V to a 5 V sun step-down converter worked (the batteries weren't new) and registered only a very small voltage drop of about 0.1 V after a few weeks of operation. The converter supplied an output of 4.95 V and, as described above, that voltage is "passed through" to the Pro Mini board during a power-on cycle.

The solar/battery-powered DHT22/BME280 system suffers occasional dropouts or produces "off the scale" values that can last for several hours. This appears to occur most often during very high relative humidity conditions. It would be possible to provide additional moisture protection for the Pro Mini/data collection board and attach the BME280 and DHT22 sensors to the system through separate cabling that isolates the sensor from the rest of the system. This might or might not fix the problems. In any case, both sensors should remain inside a ventilated enclosure that protects them from direct contact with moisture; as discussed above, "pie plate" radiation shields are not a good choice for these sensors because of possible problems with wind-blown rain. My conclusion from these data problems is that the BME280 is not a completely reliable sensor for gathering outdoor weather data.

Initially, I had concerns about whether the solar panel and charge controller system would keep the LiPo battery fully charged over a range of conditions. On October 27, when day length is decreasing rapidly, a mid-morning test on a sunny day following several partly cloudy days showed a battery voltage of 3.94 V. So far, so good…