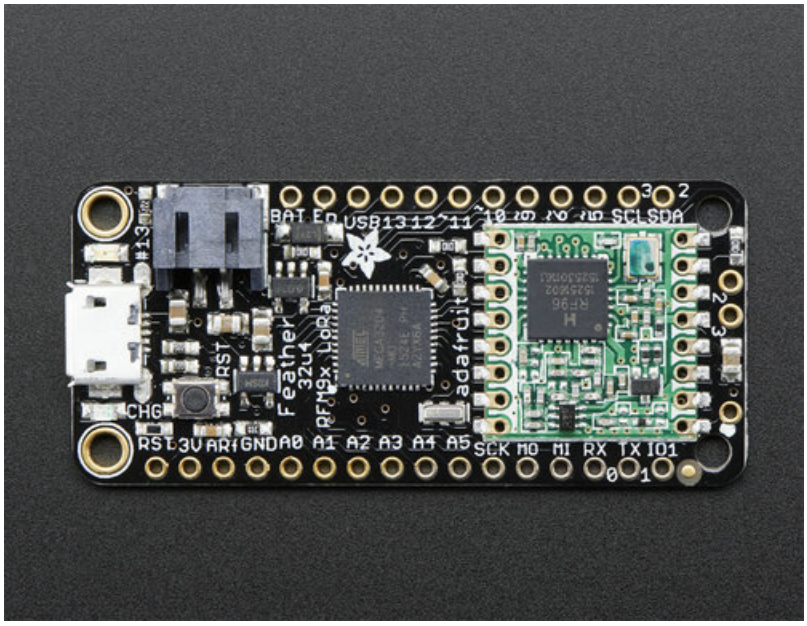
adafruit learning system

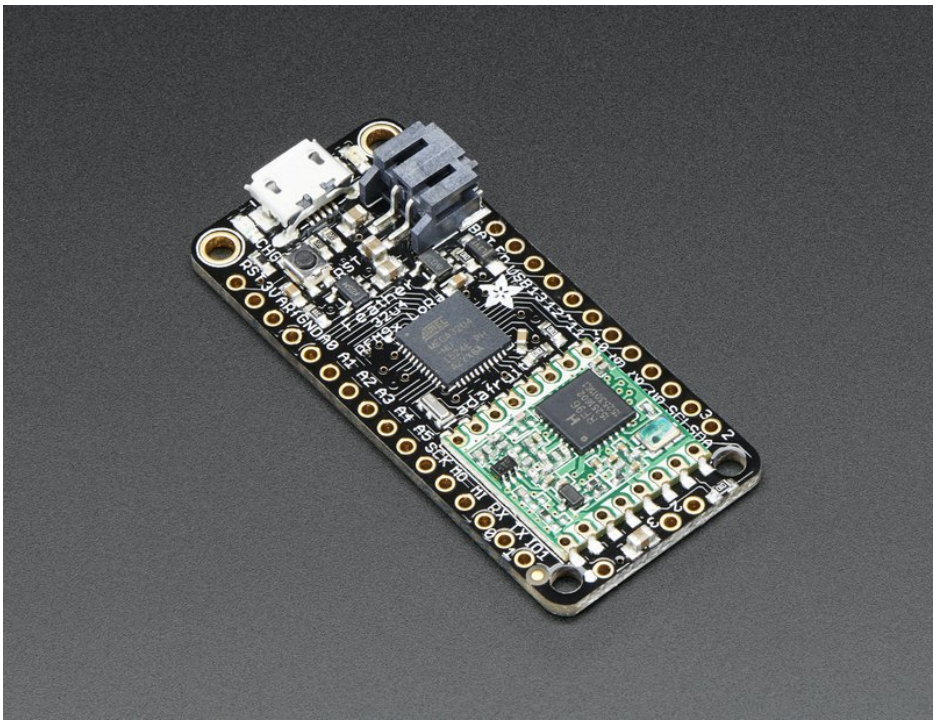
Adafruit Feather 32u4 with LoRa Radio Module

Created by lady ada



Last updated on 2019-04-23 12:56:33 AM UTC

Overview

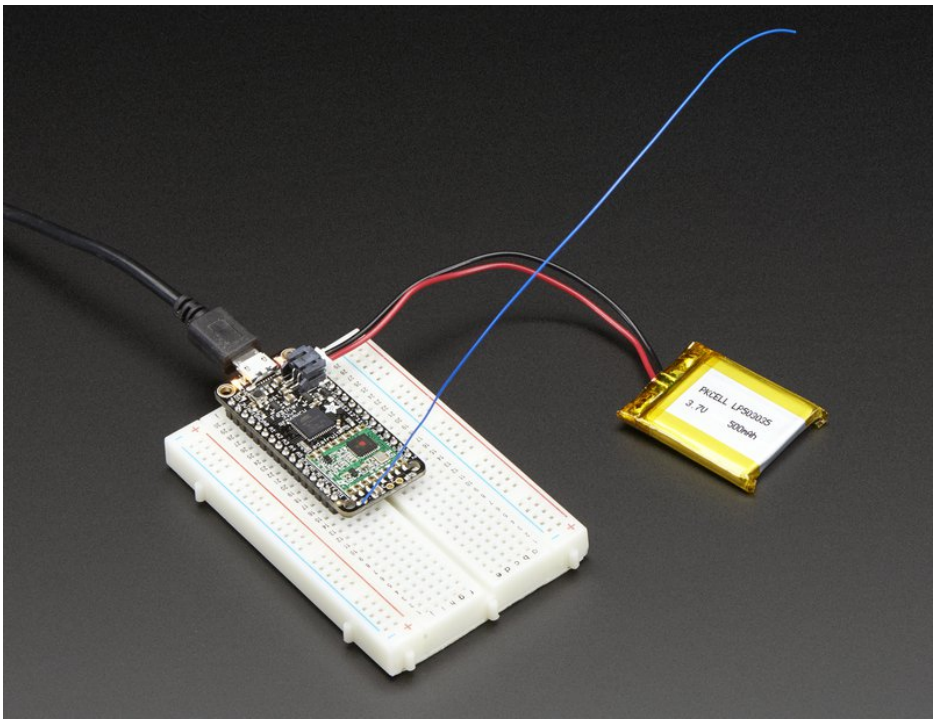


Feather is the new development board from Adafruit, and like its namesake it is thin, light, and lets you fly! We designed Feather to be a new standard for portable microcontroller cores.

This is the **Adafruit Feather 32u4 LoRa Radio (RFM9x)** - our take on an microcontroller with a 'Long Range (LoRa) (https://adafru.it/mFZ)" packet radio transceiver with built in USB and battery charging. Its an Adafruit Feather 32u4 with a 433 or 868/915 MHz radio module cooked in! Great for making wireless networks that can go further than 2.4GHz 802.15.4 and similar, are more flexible than Bluetooth LE and without the high power requirements of WiFi. We have other boards in the Feather family, check'em out here (https://adafru.it/l7B)

At the Feather 32u4's heart is at ATmega32u4 clocked at 8 MHz and at 3.3V logic, a chip setup we've had tons of experience with as it's the same as the Flora (https://adafru.it/dVl). This chip has 32K of flash and 2K of RAM, with built in USB so not only does it have a USB-to-Serial program & debug capability built in with no need for an FTDI-like chip, it can also act like a mouse, keyboard, USB MIDI device, etc.

To make it easy to use for portable projects, we added a connector for any of our 3.7V Lithium polymer batteries and built in battery charging. You don't need a battery, it will run just fine straight from the micro USB connector. But, if you do have a battery, you can take it on the go, then plug in the USB to recharge. The Feather will automatically switch over to USB power when its available. We also tied the battery thru a divider to an analog pin, so you can measure and monitor the battery voltage to detect when you need a recharge.



Here's some handy specs! Like all Feather 32u4's you get:

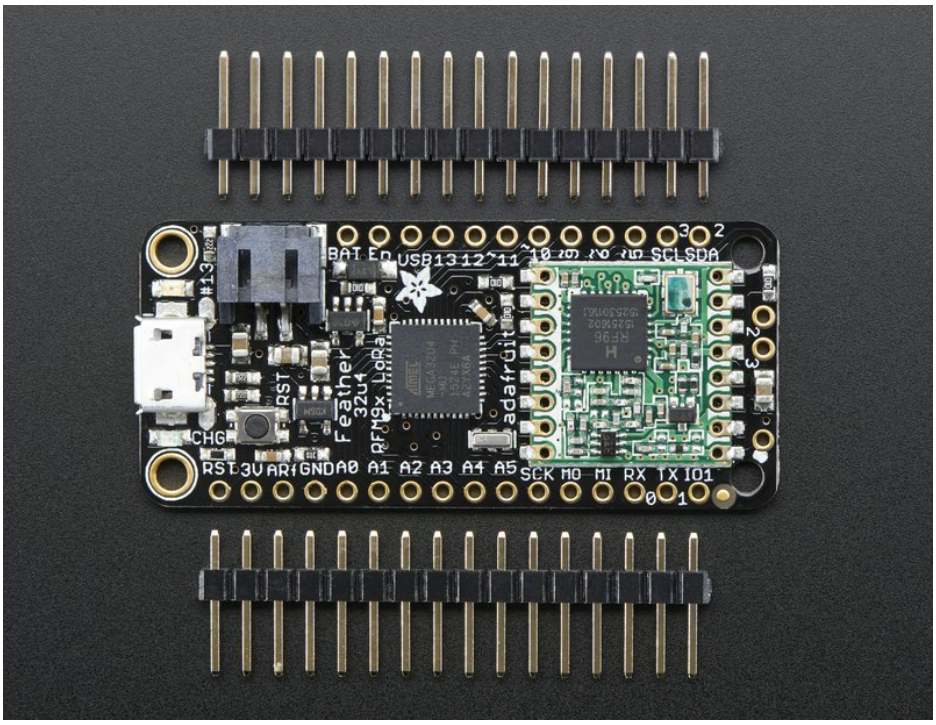
- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.5 grams
- ATmega32u4 @ 8MHz with 3.3V logic/power
- 3.3V regulator with 500mA peak current output
- USB native support, comes with USB bootloader and serial port debugging
- You also get tons of pins - 20 GPIO pins
- Hardware Serial, hardware I2C, hardware SPI support
- 7 x PWM pins
- 10 x analog inputs
- Built in 100mA lipoly charger with charging status indicator LED
- Pin #13 red LED for general purpose blinking
- Power/enable pin
- 4 mounting holes
- Reset button

The **Feather 32u4 Radio** uses the extra space left over to add an RFM9x LoRa 433 or 868/915 MHz radio module. These radios are not good for transmitting audio or video, but they do work quite well for small data packet transmission when you need more range than 2.4 GHz (BT, BLE, WiFi, ZigBee)

- SX1272 LoRa® based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the amateur or license-free ISM bands (https://adafru.it/mOE): 433MHz is ITU "Europe" license-free ISM or ITU "American" amateur with limitations. 900MHz is license free ISM for ITU "Americas"
- +5 to +20 dBm - up to 100 mW Power Output Capability (power output selectable in software)
- ~300uA during full sleep, ~120mA peak during +20dBm transmit, ~40mA during active radio listening.
- Simple wire antenna or spot for uFL connector

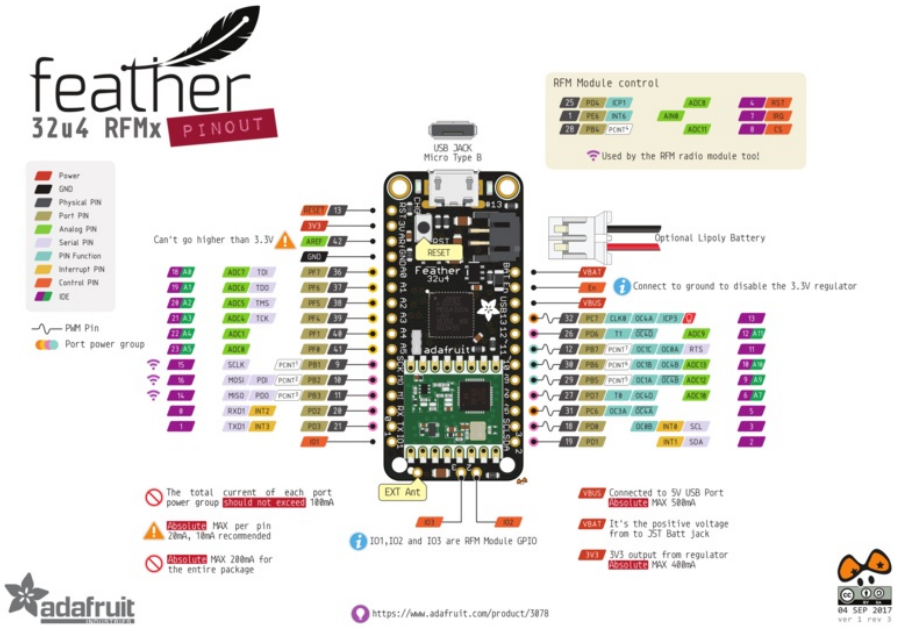
Our initial tests with default library settings: over 1.2mi/2Km line-of-sight with wire quarter-wave antennas. (With setting

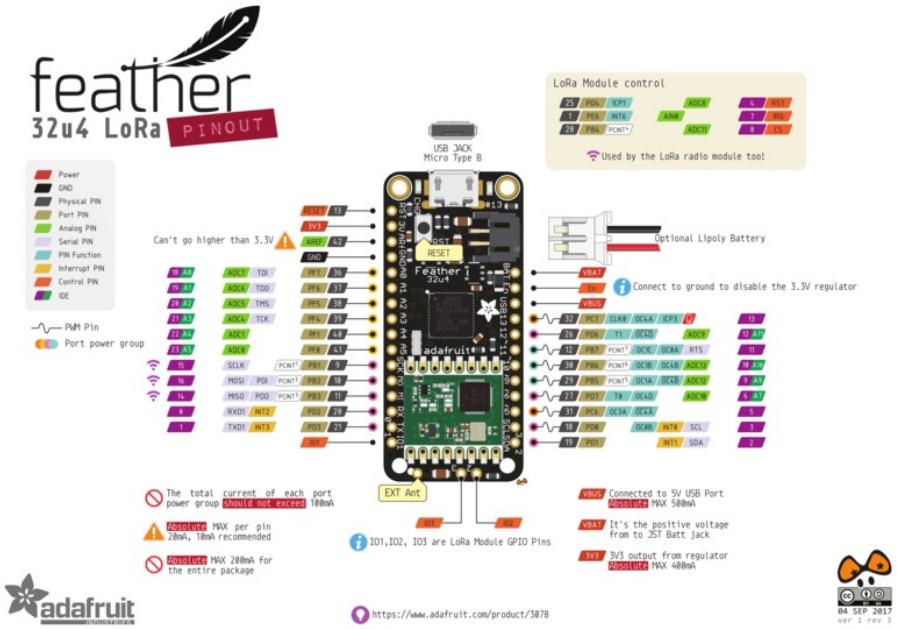
tweaking and directional antennas, 20Km is possible (https://adafru.it/mGa))



Comes fully assembled and tested, with a USB bootloader that lets you quickly use it with the Arduino IDE. We also toss in some header so you can solder it in and plug into a solderless breadboard. You will need to cut and solder on a small piece of wire (any solid or stranded core is fine) in order to create your antenna. **Lipoly battery and USB cable not included** but we do have lots of options in the shop if you'd like!

Pinouts

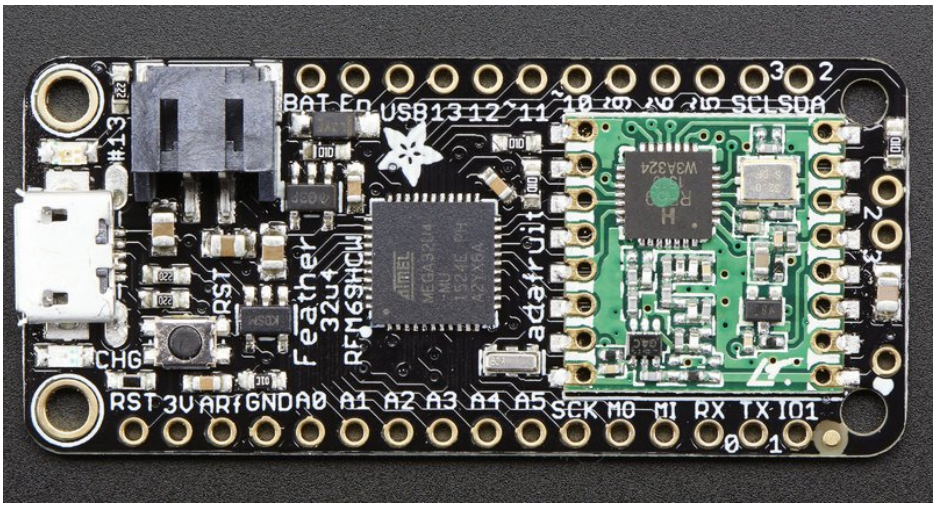




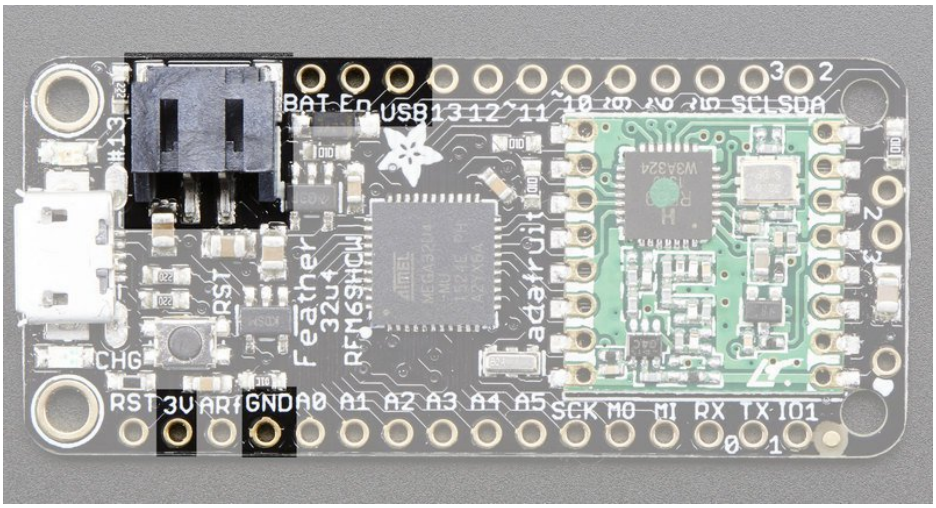
The Feather 32u4 Radio is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!

Note that the pinouts are identical for both the Feather 32u4 RFM69 and LoRa radios - you can look at the silkscreen of the Feather to see it says "RFM69" or "LoRa"

Pinouts are also the same for both 433MHz and 900Mhz. You can tell the difference by looking for a colored dot on the chip or crystal of the radio, green/blue is 900MHz & red is 433MHz

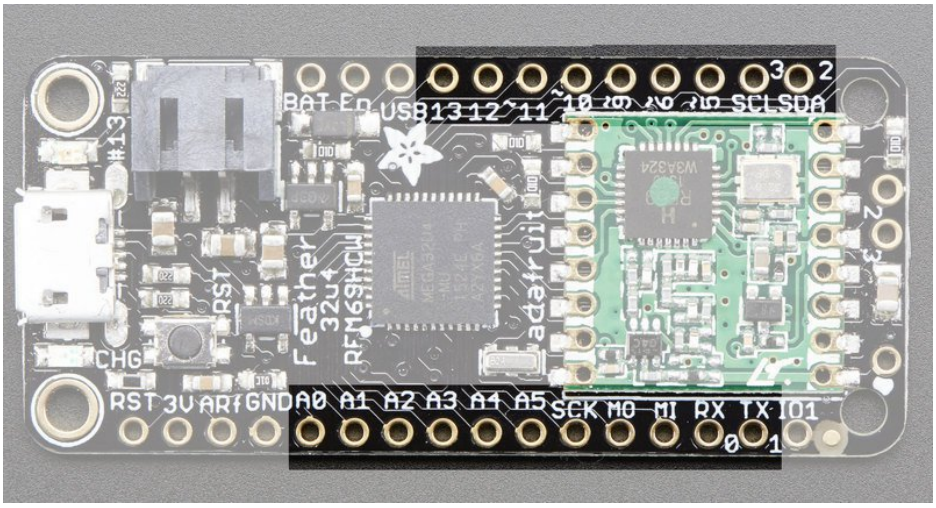


Power Pins



- **GND** - this is the common ground for all power and logic
- **BAT** - this is the positive voltage to/from the JST jack for the optional Lipoly battery
- **USB** - this is the positive voltage to/from the micro USB jack if connected
- **EN** - this is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator
- **3V** - this is the output from the 3.3V regulator, it can supply 500mA peak

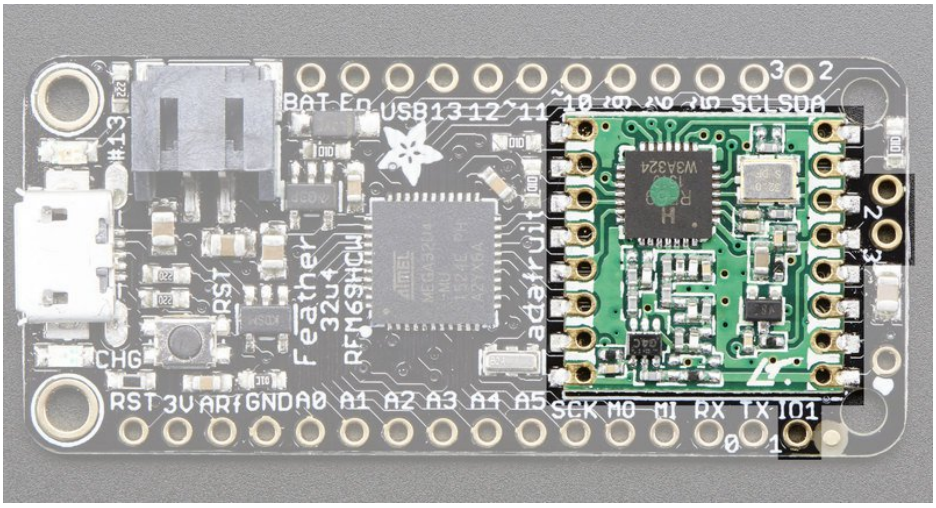
Logic pins



This is the general purpose I/O pin set for the microcontroller. All logic is 3.3V

- **#0 / RX** - GPIO #0, also receive (input) pin for **Serial1** and Interrupt #2
- **#1 / TX** - GPIO #1, also transmit (output) pin for **Serial1** and Interrupt #3
- **#2 / SDA** - GPIO #2, also the I2C (Wire) data pin. There's no pull up on this pin by default so when using with I2C, you may need a 2.2K-10K pullup. Also Interrupt #1
- **#3 / SCL** - GPIO #3, also the I2C (Wire) clock pin. There's no pull up on this pin by default so when using with I2C, you may need a 2.2K-10K pullup. Can also do PWM output and act as Interrupt #0.
- **#5** - GPIO #5, can also do PWM output
- **#6** - GPIO #6, can also do PWM output and analog input **A7**
- **#9** - GPIO #9, also analog input **A9** and can do PWM output. This analog input is connected to a voltage divider for the lipoly battery so be aware that this pin naturally 'sits' at around 2VDC due to the resistor divider
- **#10** - GPIO #10, also analog input **A10** and can do PWM output.
- **#11** - GPIO #11, can do PWM output.
- **#12** - GPIO #12, also analog input **A11** and can do PWM output.
- **#13** - GPIO #13, can do PWM output and is connected to the**red LED** next to the USB jack
- **A0 thru A5** - These are each analog input as well as digital I/O pins.
- **SCK/MOSI/MISO** - These are the hardware SPI pins, **used by the RFM radio module too!** You can use them as everyday GPIO pins if you don't activate the radio and keep the RFM CS pin high. However, we really recommend keeping them free as they should be kept available for the radio module. If they are used, make sure its with a device that will kindly share the SPI bus! Also used to reprogram the chip with an AVR programmer if you need.

RFM/SemTech Radio Module



Since not all pins can be brought out to breakouts, due to the small size of the Feather, we use these to control the radio module

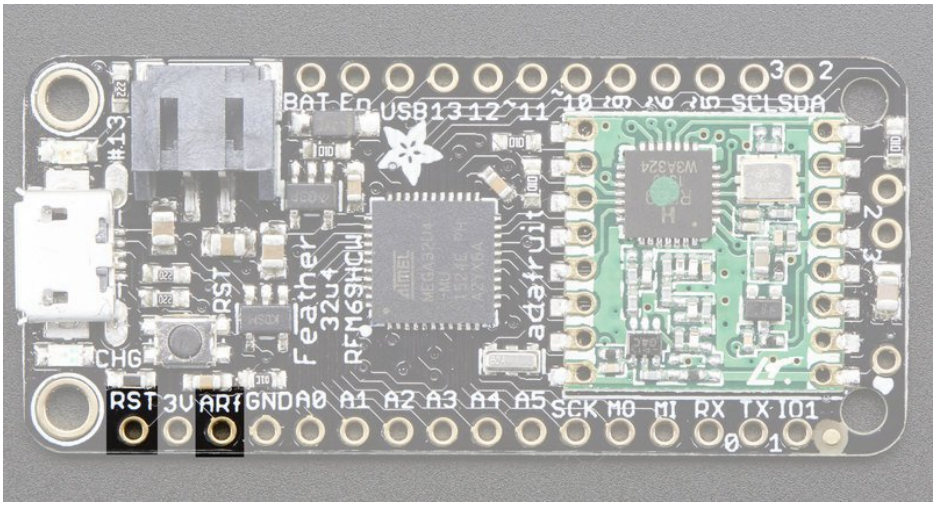
- **#8** - used as the radio **CS** (chip select) pin
- **#7** - used as the radio **GPIO0** / **IRQ** (interrupt request) pin.
- **#4** - used as the radio **Reset** pin

Since these are not brought out there should be no risk of using them by accident!

There are also breakouts for 3 of the RFM's GPIO pins (**IO1, IO2** and **IO3**). You probably wont need these for most uses of the Feather but they are available in case you need 'em!

Other Pins!

- **RST** - this is the Reset pin, tie to ground to manually reset the AVR, as well as launch the bootloader manually
- **ARef** - the analog reference pin. Normally the reference voltage is the same as the chip logic voltage (3.3V) but if you need an alternative analog reference, connect it to this pin and select the external AREF in your firmware. Can't go higher than 3.3V!

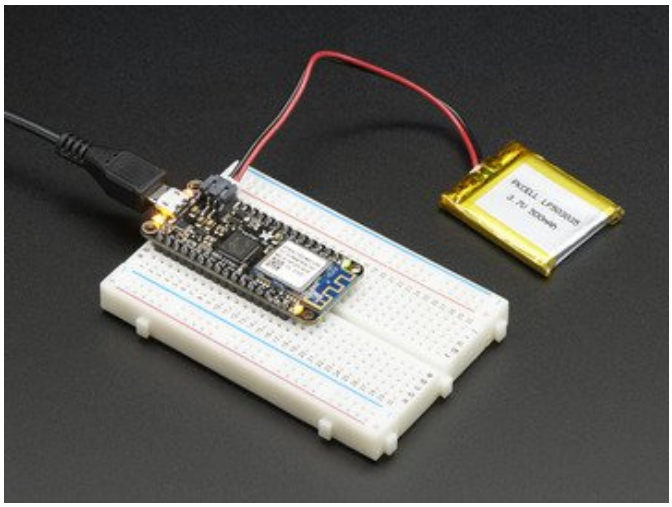


Assembly

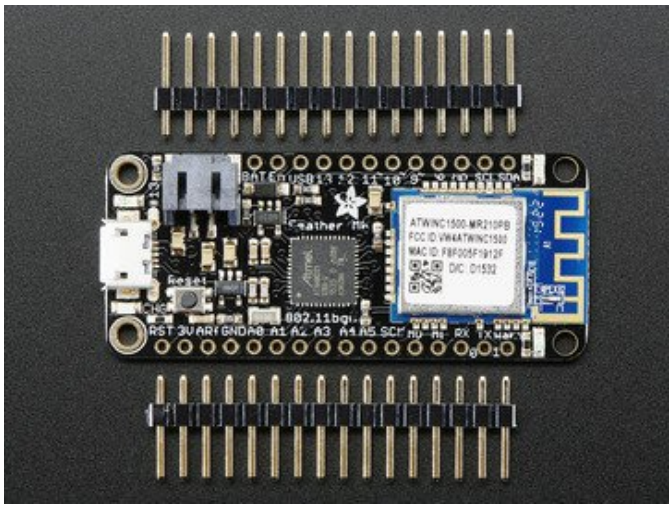
We ship Feathers fully tested but without headers attached - this gives you the most flexibility on choosing how to use and configure your Feather

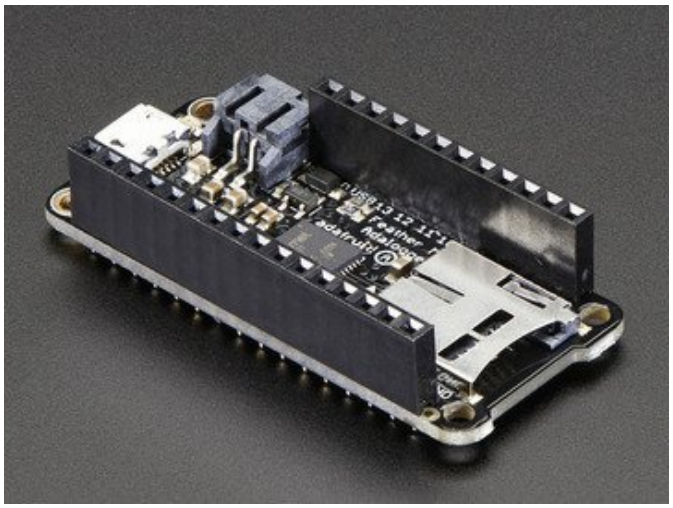
Header Options!

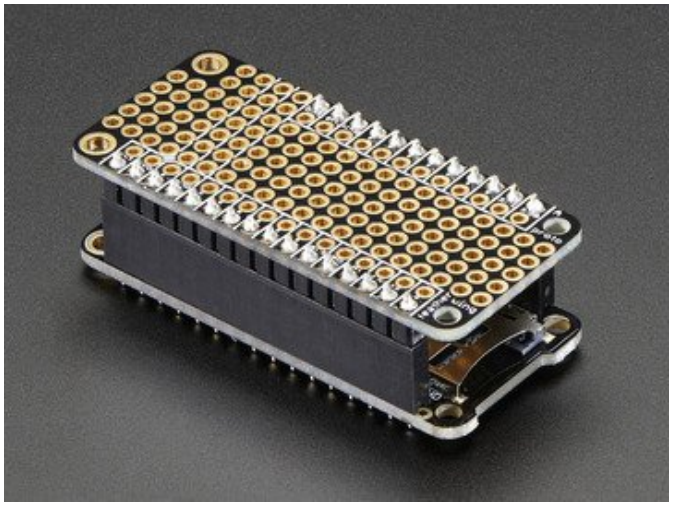
Before you go gung-ho on soldering, there's a few options to consider!



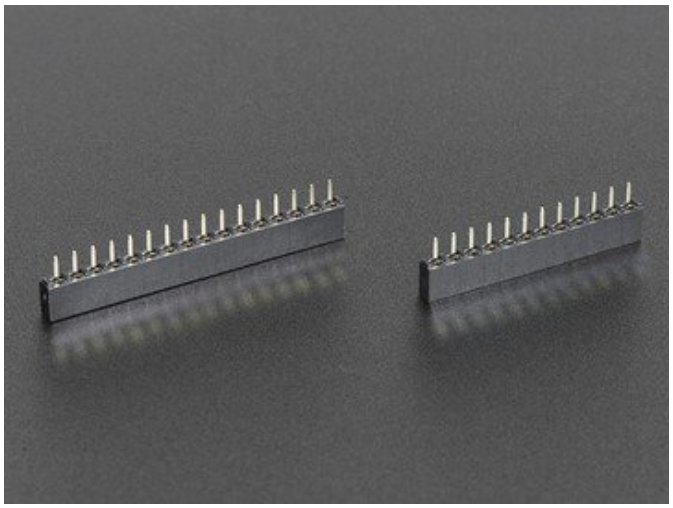
The first option is soldering in plain male headers, this lets you plug in the Feather into a solderless breadboard



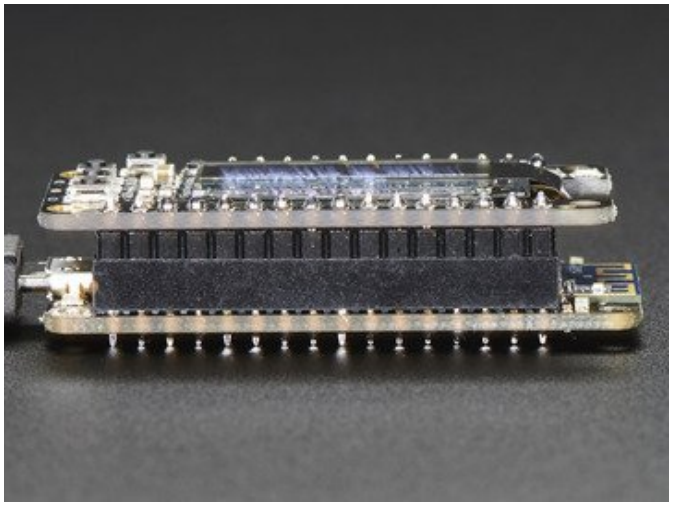


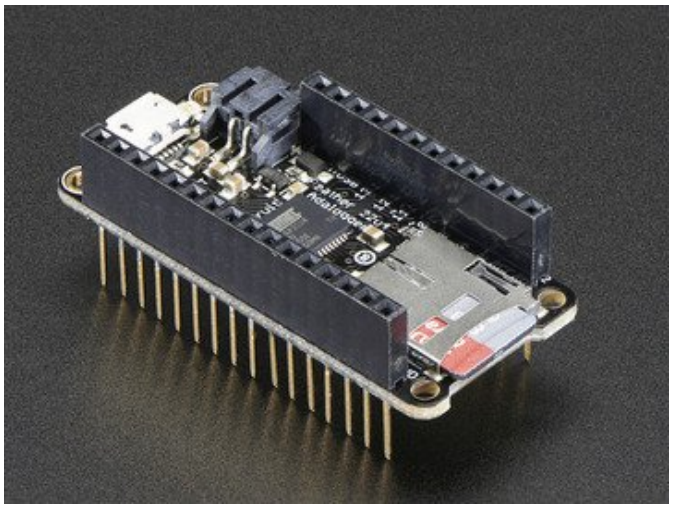


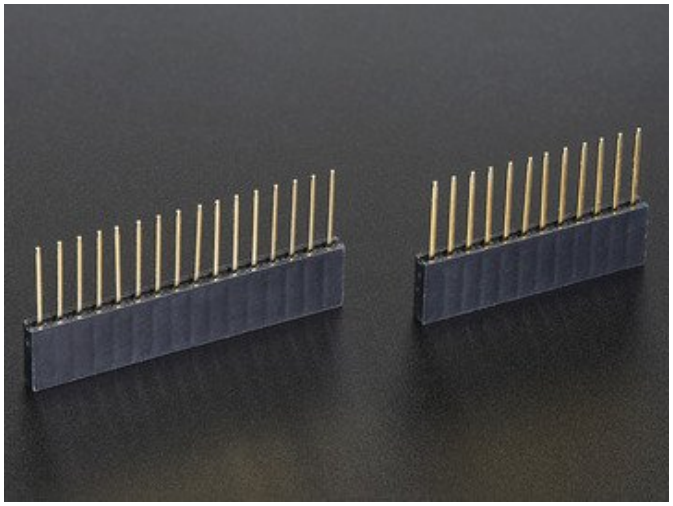
Another option is to go with socket female headers. This won't let you plug the Feather into a breadboard but it will let you attach featherwings very easily



We also have 'slim' versions of the female headers, that are a little shorter and give a more compact shape

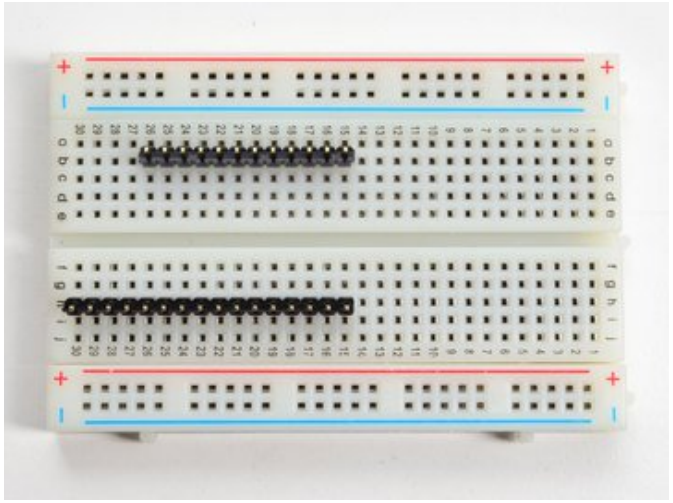






Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard *and* plug a featherwing on top. But its a little bulky

Soldering in Plain Headers

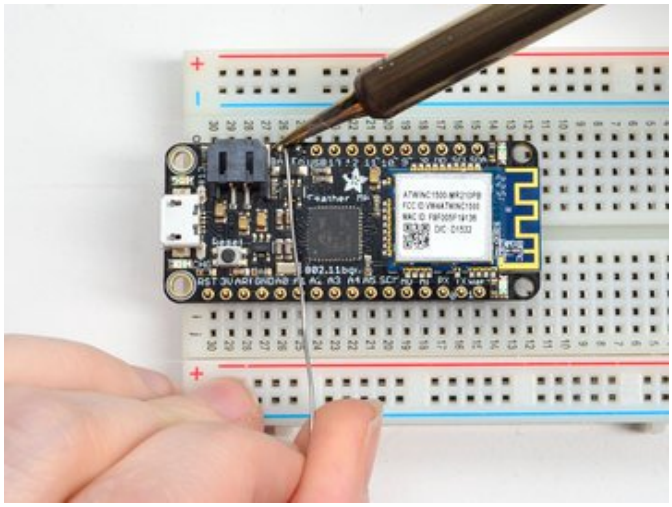


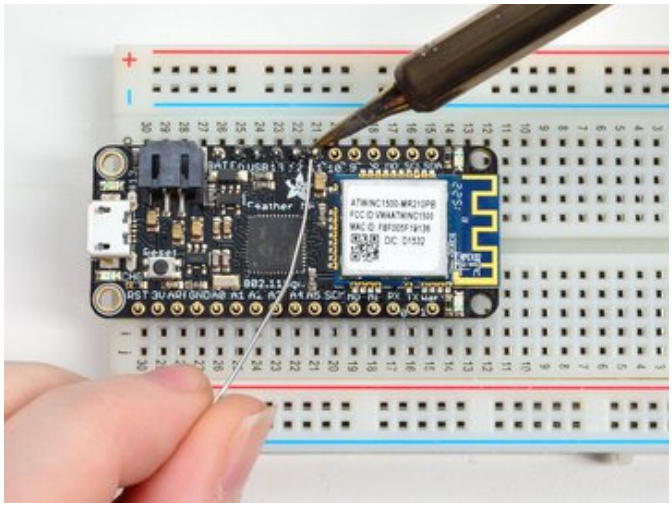
Prepare the header strip:

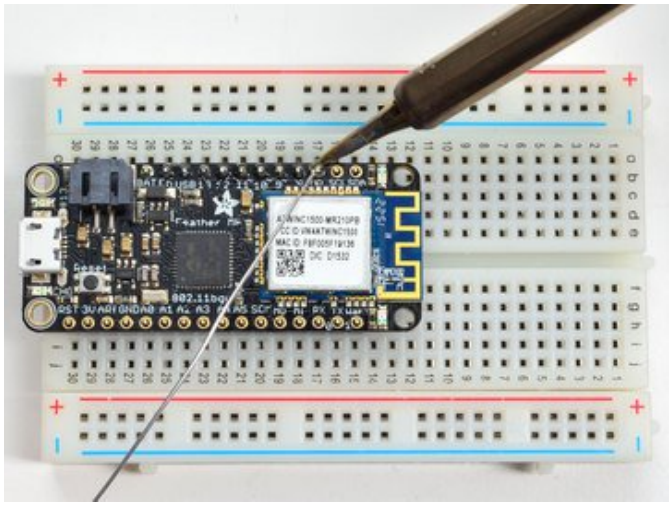
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

Add the breakout board:

Place the breakout board over the pins so that the short





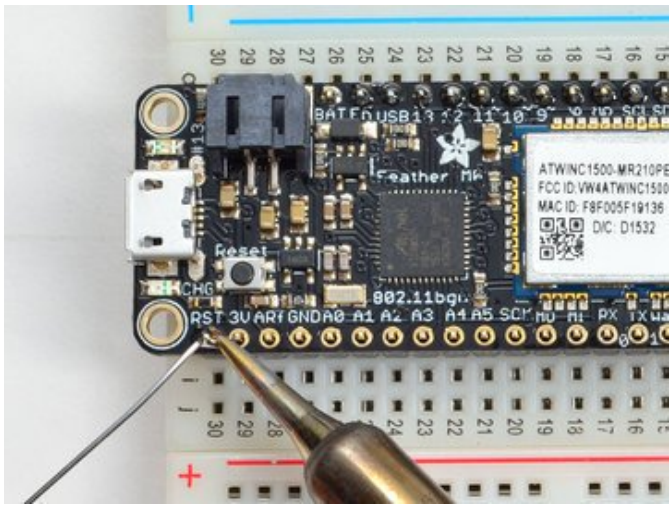


pins poke through the breakout pads

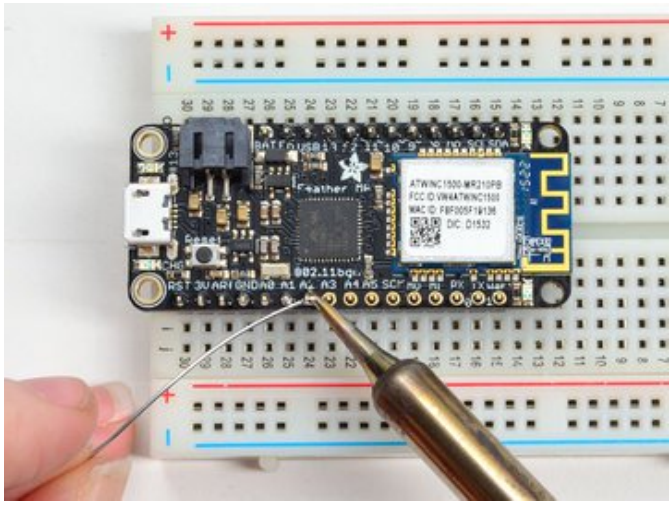
And Solder!

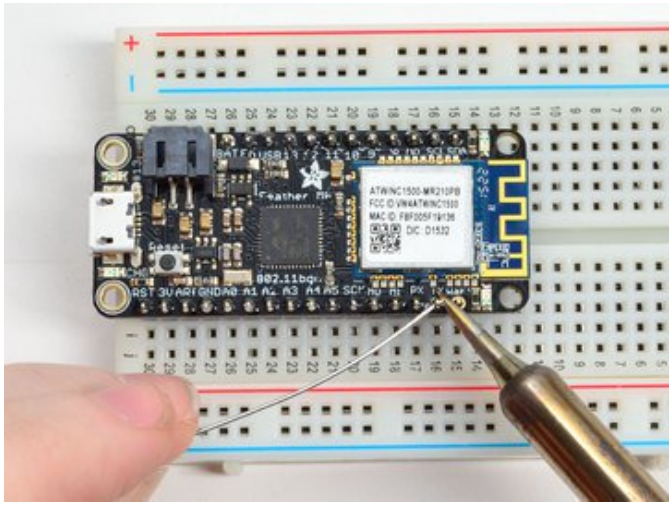
Be sure to solder all pins for reliable electrical contact.

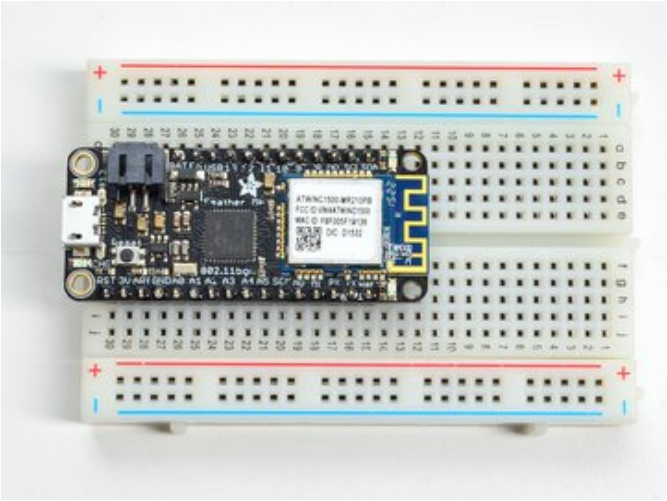
(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk).



Solder the other strip as well.





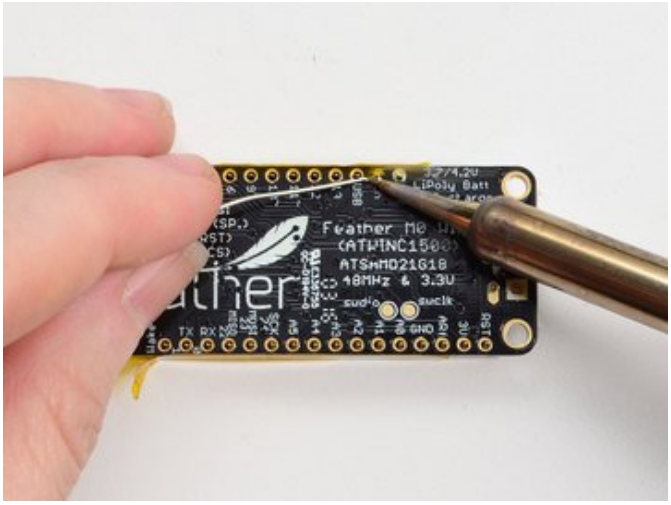


You're done! Check your solder joints visually and continue onto the next steps

Soldering on Female Header

Tape In Place

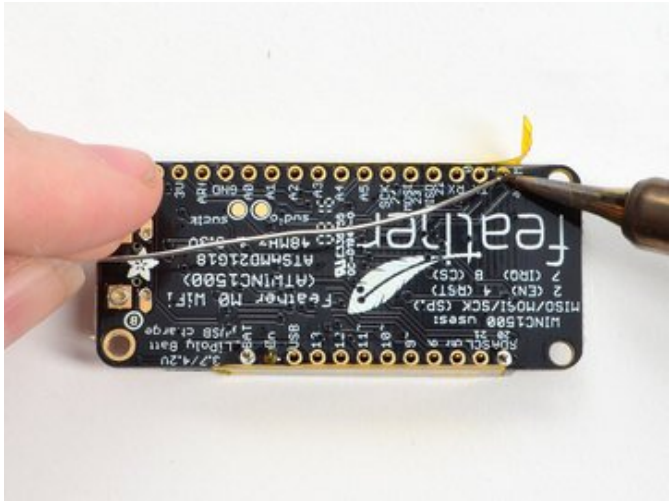
For sockets you'll want to tape them in place so when you flip over the board they don't fall out

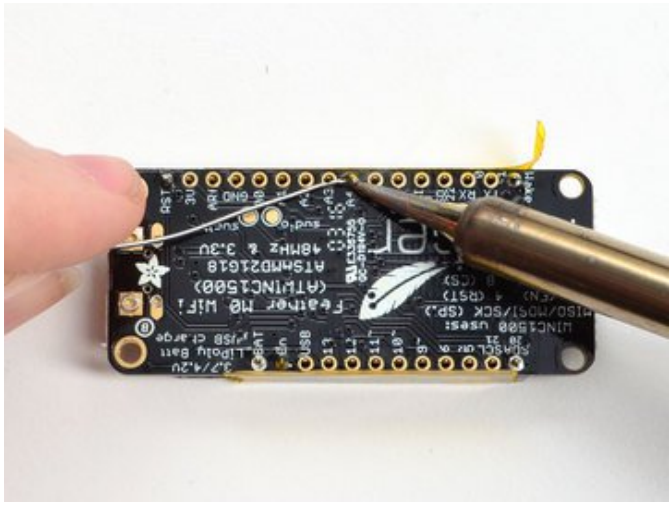


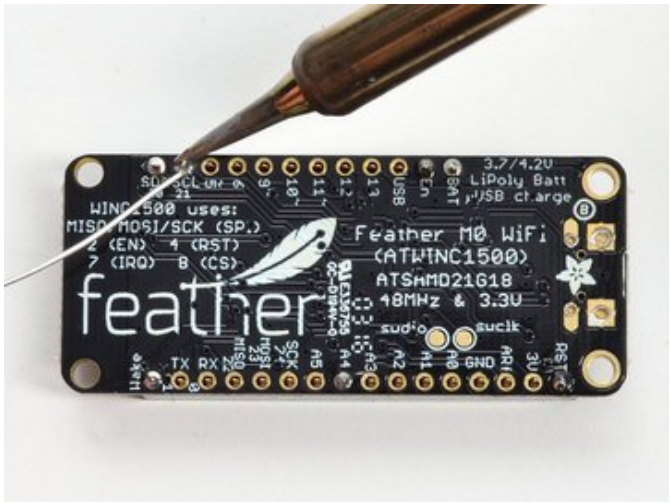
Flip & Tack Solder

After flipping over, solder one or two points on each strip, to 'tack' the header in place


feather
M0 WiFi


feather





And Solder!

Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk).

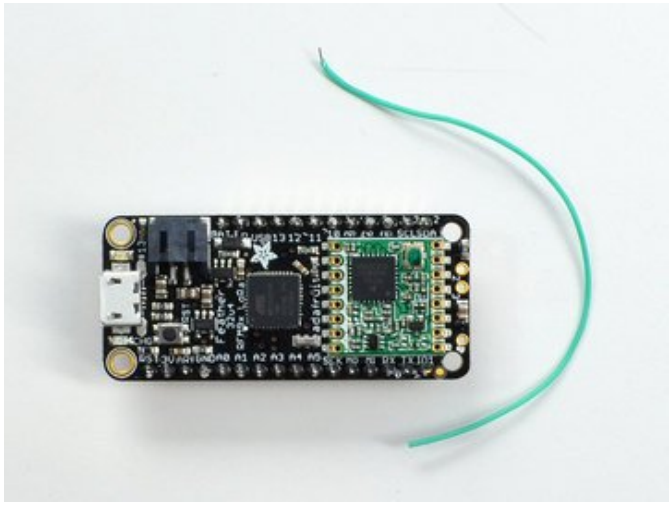
You're done! Check your solder joints visually and continue onto the next steps

Antenna Options

Your Feather Radio does not have a built-in antenna. Instead, you have two options for attaching an antenna. For most low cost radio nodes, a wire works great. If you need to put the Feather into an enclosure, soldering in uFL and using a uFL to SMA adapter will let you attach an external antenna

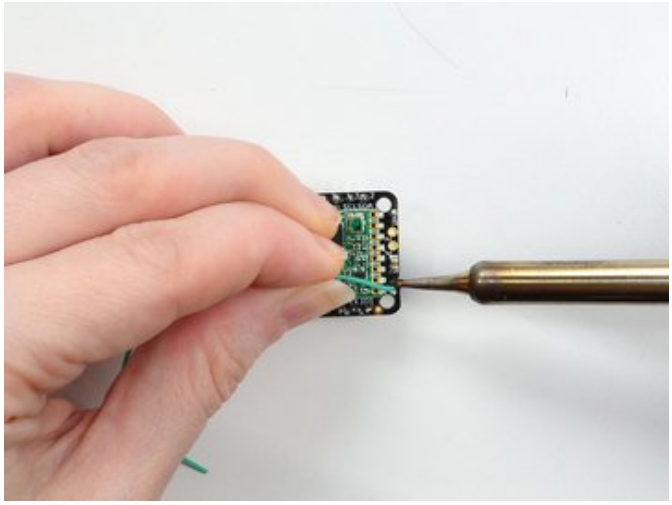
Wire Antenna

A wire antenna, aka "quarter wave whip antenna" is low cost and works very well! You just have to cut the wire down to the right length.

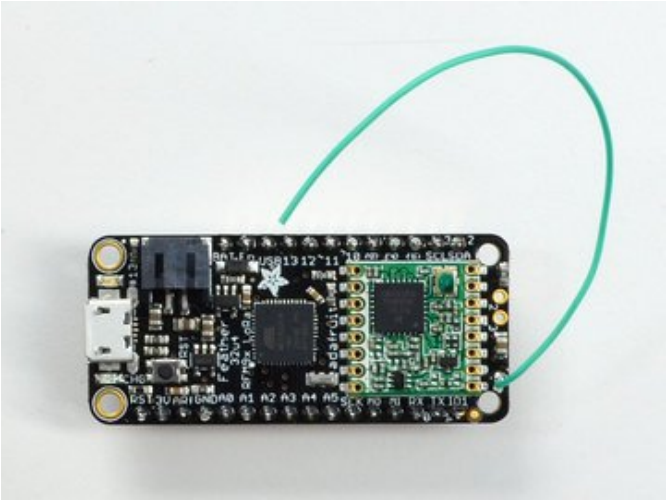


Cut a stranded or solid core wire to the proper length for the module/frequency

- **433 MHz** - 6.5 inches, or 16.5 cm
- **868 MHz** - 3.25 inches or 8.2 cm
- **915 MHz** - 3 inches or 7.8 cm



Strip a mm or two off the end of the wire, tin and solder into the **ANT** pad on the very right hand edge of the Feather



That's pretty much it, you're done!

uFL Antenna

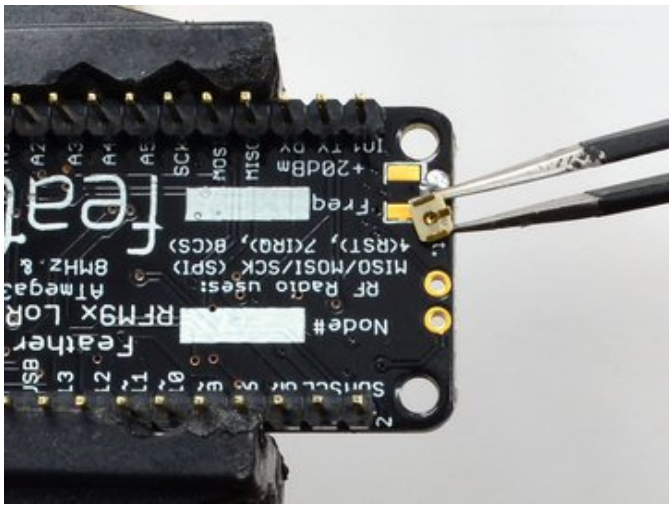
If you want an external antenna, you need to do a tiny bit more work but its not too difficult.

You'll need to get an SMT uFL connector, these are fairly standard (http://adafru.it/1661)

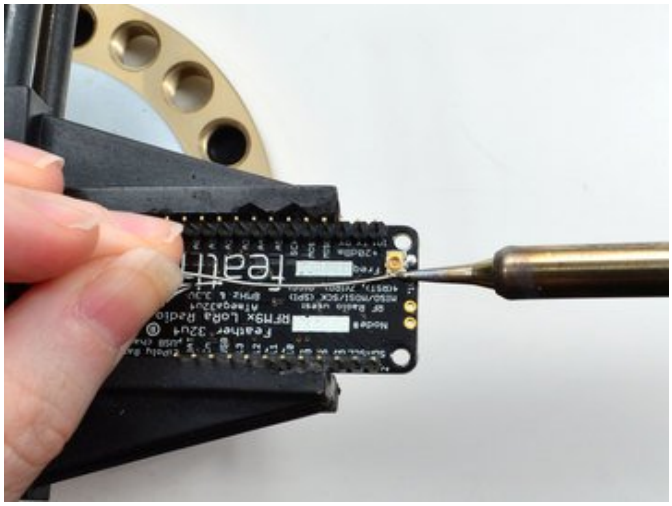
You'll also need a uFL to SMA adapter (http://adafru.it/851) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

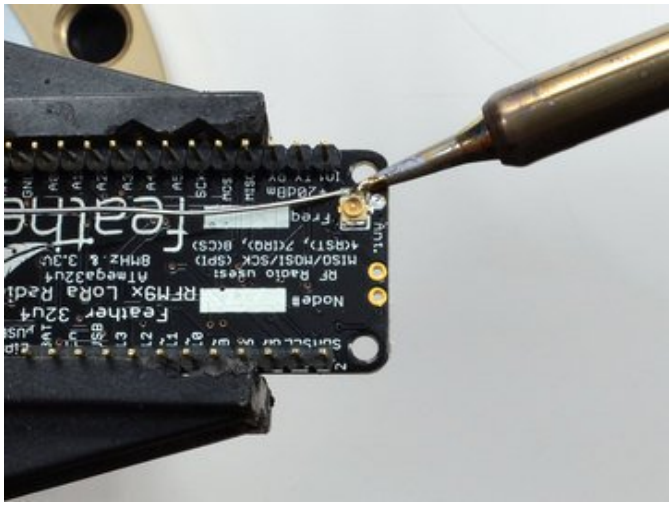
Of course, you will also need an antenna of some sort, that matches your radio frequency

uFL connectors are rated for 30 connection cycles, but be careful when connecting/disconnecting to not rip the pads off the PCB. Once a uFL/SMA adapter is connected, use strain relief!

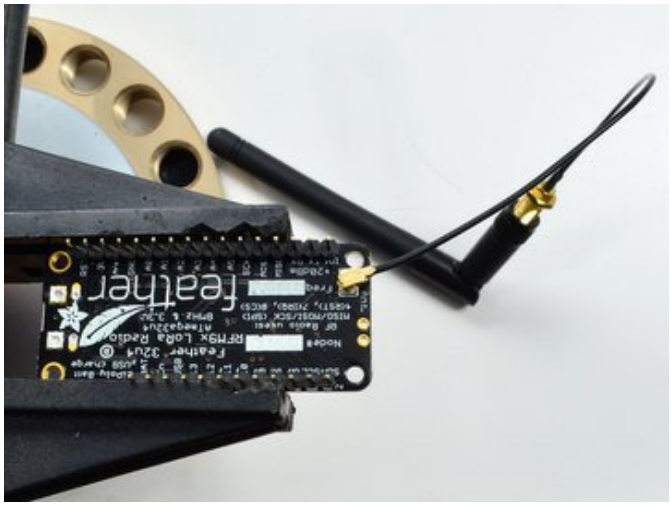


Check the bottom of the uFL connector, note that there's two large side pads (ground) and a little inlet pad. The other small pad is not used!



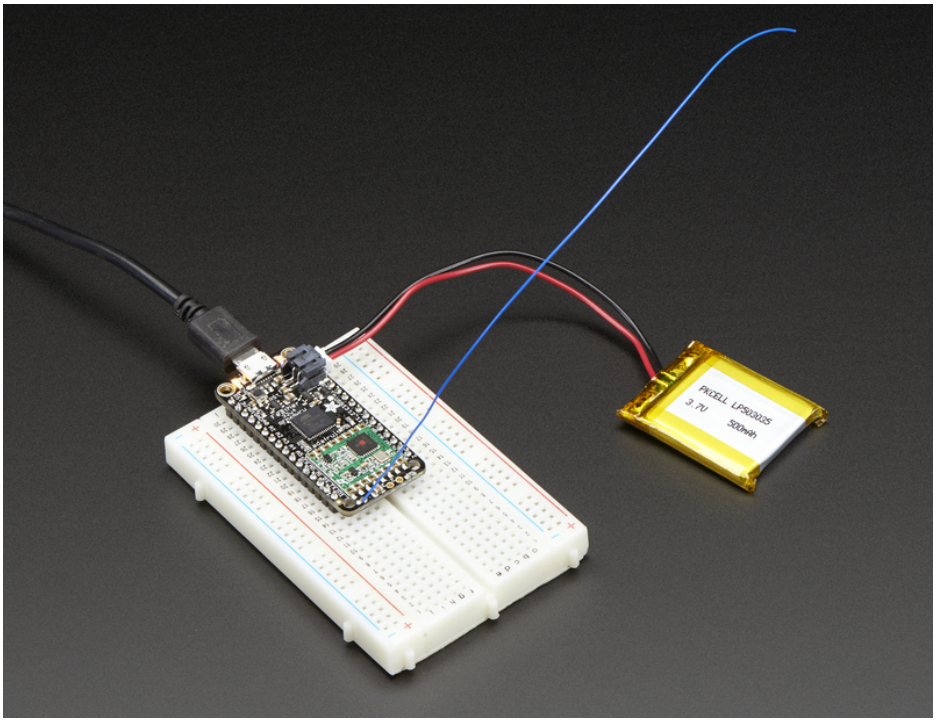


Solder all three pads to the bottom of the Feather



Once done attach your uFL adapter and antenna!

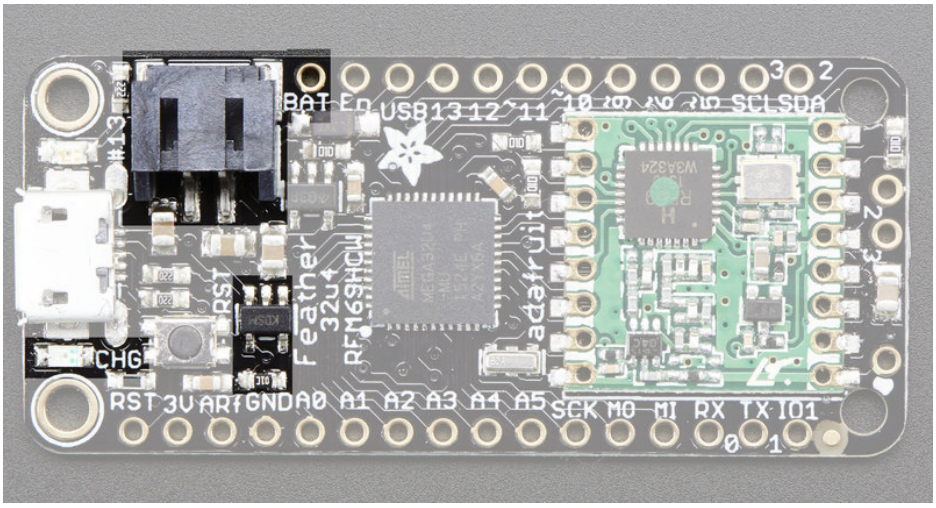
Power Management



Battery + USB Power

We wanted to make the Feather easy to power both when connected to a computer as well as via battery. There's **two ways to power** a Feather. You can connect with a MicroUSB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V. You can also connect a 4.2/3.7V Lithium Polymer (Lipo/Lipoly) or Lithium Ion (Lilon) battery to the JST jack. This will let the Feather run on a rechargable battery. **When the USB power is powered, it will automatically switch over to USB for power, as well as start charging the battery (if attached) at 100mA.** This happens 'hotswap' style so you can always keep the Lipoly connected as a 'backup' power that will only get used when USB power is lost.

The JST connector polarity is matched to Adafruit LiPoly batteries. Using wrong polarity batteries can destroy your Feather

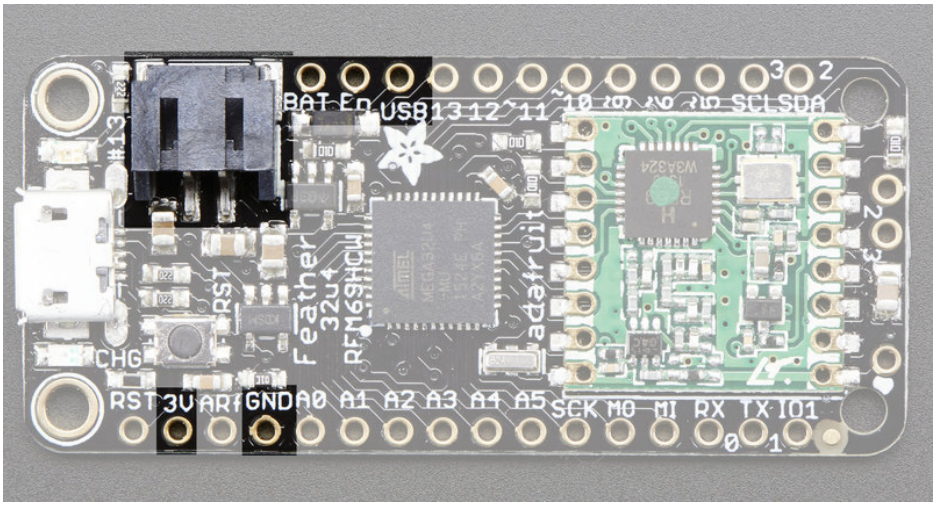


The above shows the Micro USB jack (left), Lipoly JST jack (top left), as well as the 3.3V regulator and changeover diode (just to the right of the JST jack) and the Lipoly charging circuitry (to the right of the Reset button). There's also a **CHG** LED, which will light up while the battery is charging. This LED might also flicker if the battery is not connected.

The charge LED is automatically driven by the Lipoly charger circuit. It will try to detect a battery and is expecting one to be attached. If there isn't one it may flicker once in a while when you use power because it's trying to charge a (non-existant) battery. It's not harmful, and its totally normal!

Power supplies

You have a lot of power supply options here! We bring out the **BAT** pin, which is tied to the lipoly JST connector, as well as **USB** which is the +5V from USB if connected. We also have the **3V** pin which has the output from the 3.3V regulator. We use a 500mA peak AP2112. While you can get 500mA from it, you can't do it continuously from 5V as it will overheat the regulator. It's fine for, say, powering an ESP8266 WiFi chip or XBee radio though, since the current draw is 'spiky' & sporadic.



Measuring Battery

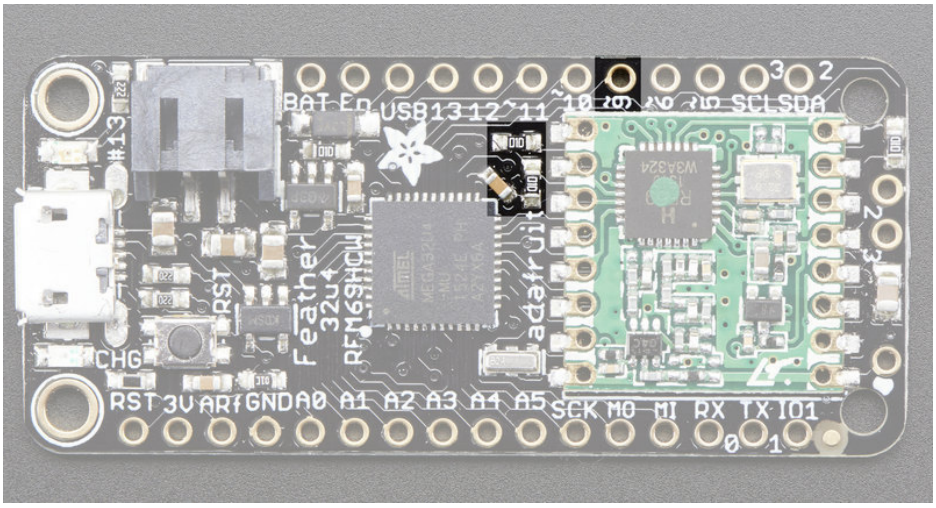
If you're running off of a battery, chances are you wanna know what the voltage is at! That way you can tell when the battery needs recharging. Lipoly batteries are 'maxed out' at 4.2V and stick around 3.7V for much of the battery life,

then slowly sink down to 3.2V or so before the protection circuitry cuts it off. By measuring the voltage you can quickly tell when you're heading below 3.7V

To make this easy we stuck a double-100K resistor divider on the **BAT** pin, and connected it to **D9** (a.k.a analog #7 **A7**). You can read this pin's voltage, then double it, to get the battery voltage.

```
#define VBATPIN A9

float measuredvbat = analogRead(VBATPIN);
measuredvbat *= 2;    // we divided by 2, so multiply back
measuredvbat *= 3.3;  // Multiply by 3.3V, our reference voltage
measuredvbat /= 1024; // convert to voltage
Serial.print("VBat: " ); Serial.println(measuredvbat);
```

This voltage will 'float' at 4.2V when no battery is plugged in, due to the lipoly charger output, so its not a good way to detect if a battery is plugged in or not (there is no simple way to detect if a battery is plugged in)



Alternative Power Options

The two primary ways for powering a feather are a 3.7/4.2V LiPo battery plugged into the JST port *or* a USB power cable.

If you need other ways to power the Feather, here's what we recommend:

- For permanent installations, a 5V 1A USB wall adapter (https://adafru.it/duP) will let you plug in a USB cable for reliable power
- For mobile use, where you don't want a LiPoly, use a USB battery pack! (https://adafru.it/e2q)
- If you have a higher voltage power supply, use a 5V buck converter (https://adafru.it/DHs) and wire it to a USB cable's 5V and GND input (https://adafru.it/DHu)

Here's what you cannot do:

- **Do not use alkaline or NiMH batteries** and connect to the battery port - this will destroy the LiPoly charger and there's no way to disable the charger
- **Do not use 7.4V RC batteries on the battery port** - this will destroy the board

The Feather *is not designed for external power supplies* - this is a design decision to make the board compact and low

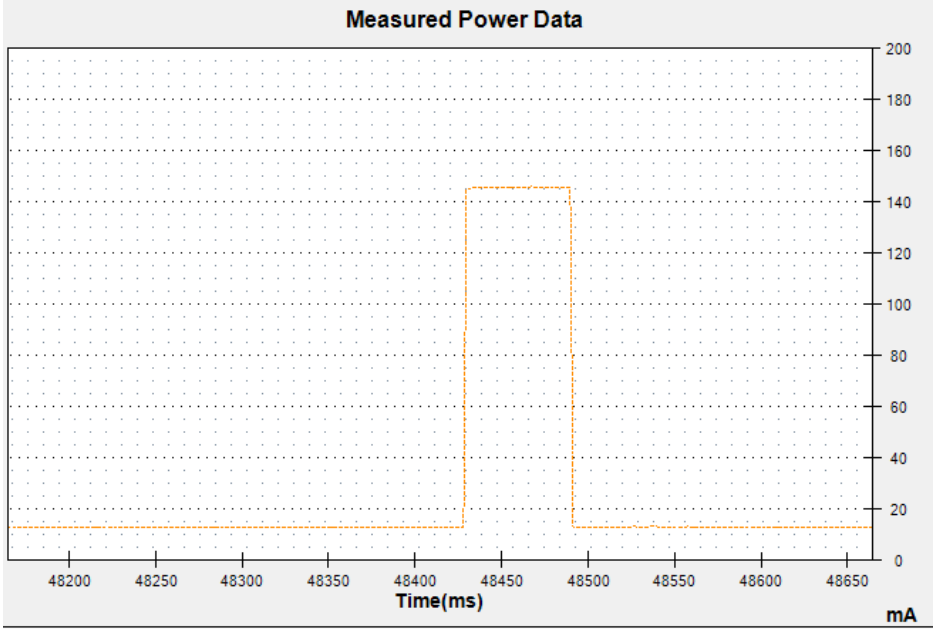
cost. It is not recommended, but technically possible:

- **Connect an external 3.3V power supply to the 3V and GND pins.** Not recommended, this may cause unexpected behavior and the **EN** pin will no longer. Also this doesn't provide power on **BAT** or **USB** and some Feathers/Wings use those pins for high current usages. You may end up damaging your Feather.
- **Connect an external 5V power supply to the USB and GND pins.** Not recommended, this may cause unexpected behavior when plugging in the USB port because you will be back-powering the USB port, which *could* confuse or damage your computer.

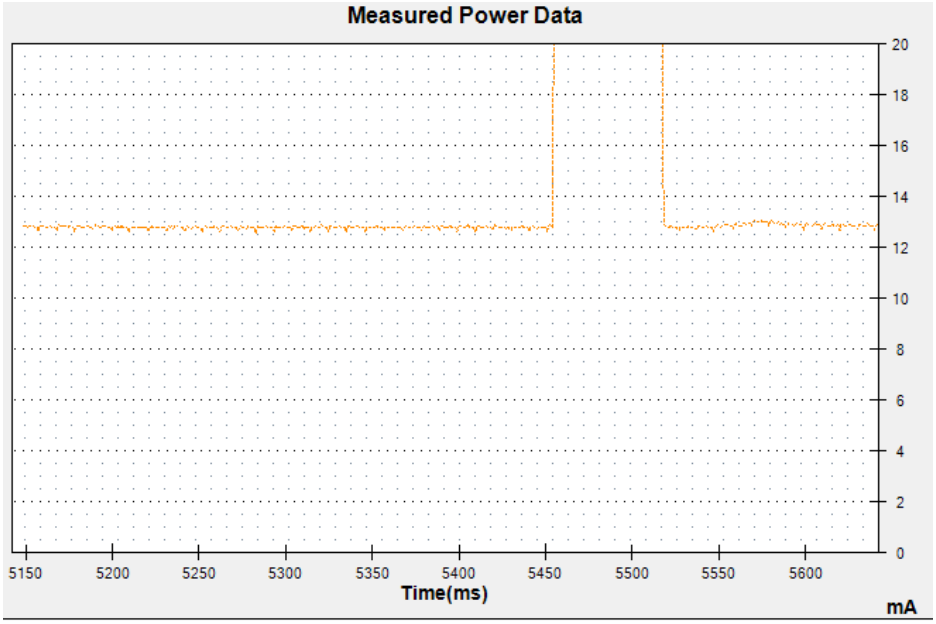
Radio Power Draw

You can select the power output you want via software, more power equals more range but of course, uses more of your battery.

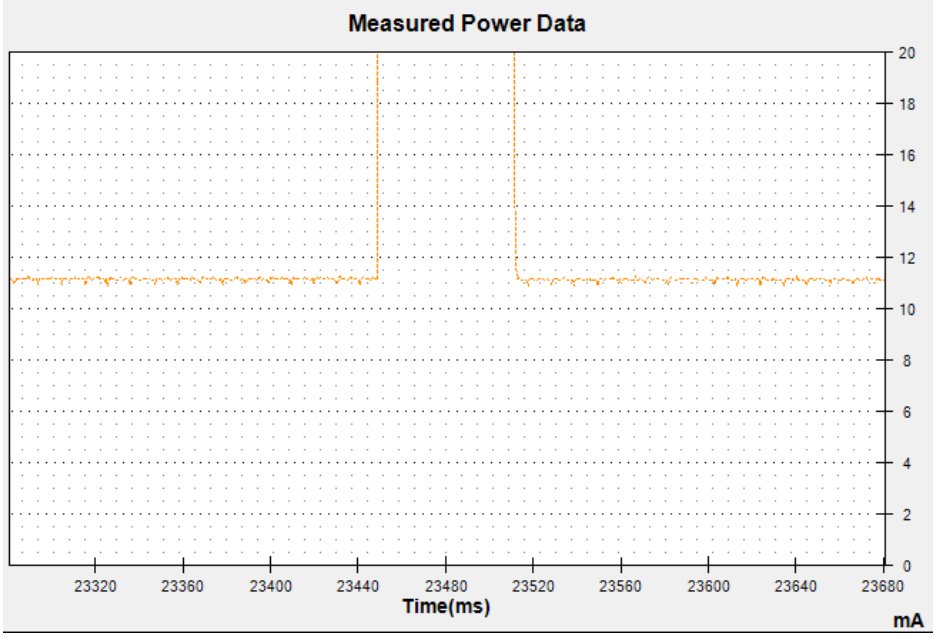
For example, here is the feather 32u4 with RFM9x 900MHz radio set up for +20dBm power, transmitting a data payload of 20 bytes. Transmits take about 130mA for 70ms



The ~13mA quiescent current is the current draw for listening (~2mA) plus ~11mA for the microcontroller. This can be reduce to amost nothing with proper sleep modes and not putting the module in active listen mode!



You can put the module into sleep mode by calling `radio.sleep();` which will save you about 2mA



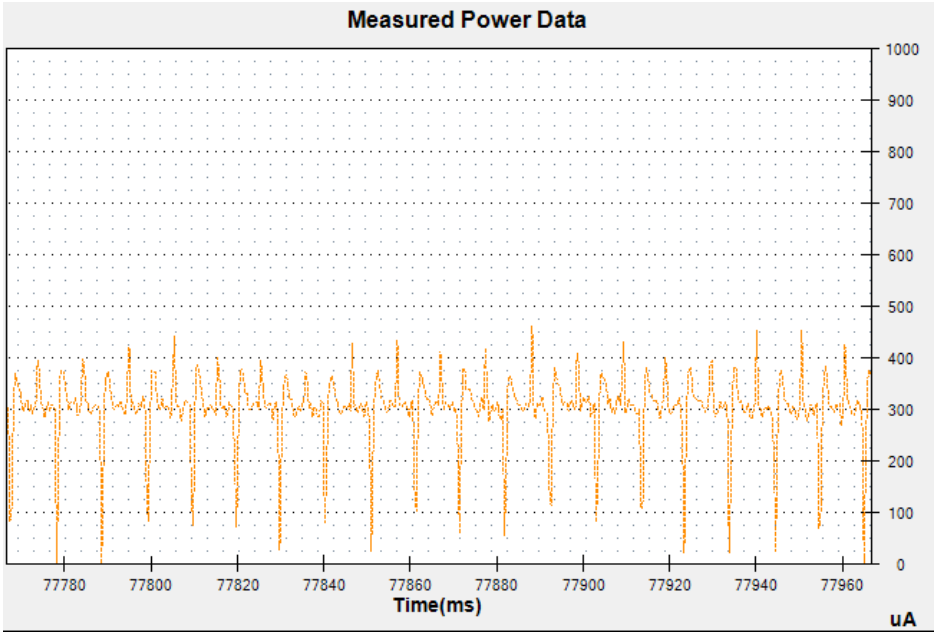
If you want to reduce even more power, use the Adafruit Sleepdog (https://adafru.it/fp8)library by installing and adding #include "Adafruit_SleepyDog.h" at the top of your sketch and replace

```
delay(1000);
```

with

```
radio.sleep();
Watchdog.sleep(1000);
```

To put the chip into ultra-low-power mode. Note that USB will disconnect so do this after you have done all your debugging!



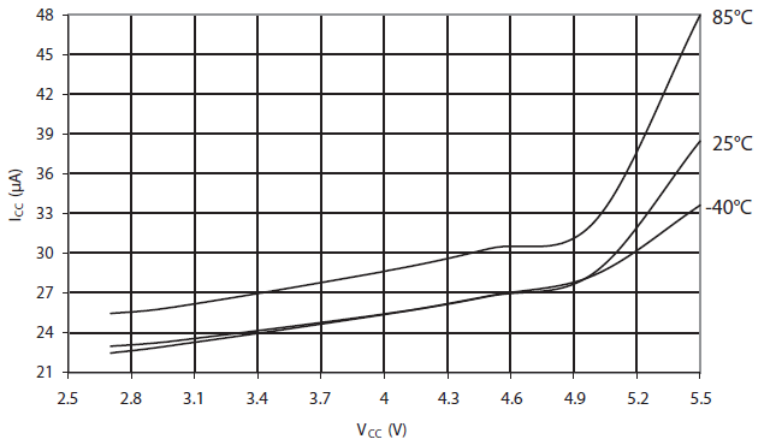
During the super sleepy mode you're using only 300uA (0.3mA)!

While its not easy to get the exact numbers for all of what comprise the 300uA there are a few quiescent current items on the Feather 32u4:

- 2 x 100K resistors for VBAT measurement = **25uA**
- AP2112K 3.3V regulator = **55uA**
- MCP73871 batt charger = **up to 100uA** even when no battery is connected

The rest is probably the Atmega32u4 peripherals including the brown-out detect and bandgap circuitry, ceramic oscillator, etc. According to the datasheet, with the watchdog and BrownOutDetect enabled, the lowest possible current is **~30uA** (at 5V which is what we're testing at)

Figure 30-12. Power-down Supply Current vs. V_{CC} (WDT Enabled, BOD EN)



ENable pin

If you'd like to turn off the 3.3V regulator, you can do that with the **EN**(able) pin. Simply tie this pin to **Ground** and it will disable the 3V regulator. The **BAT** and **USB** pins will still be powered

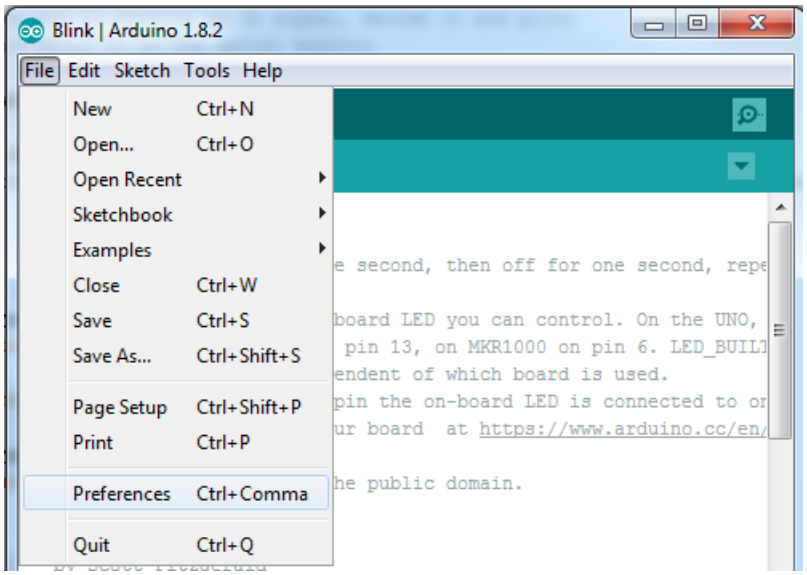
Arduino IDE Setup

The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.8** or higher for this guide

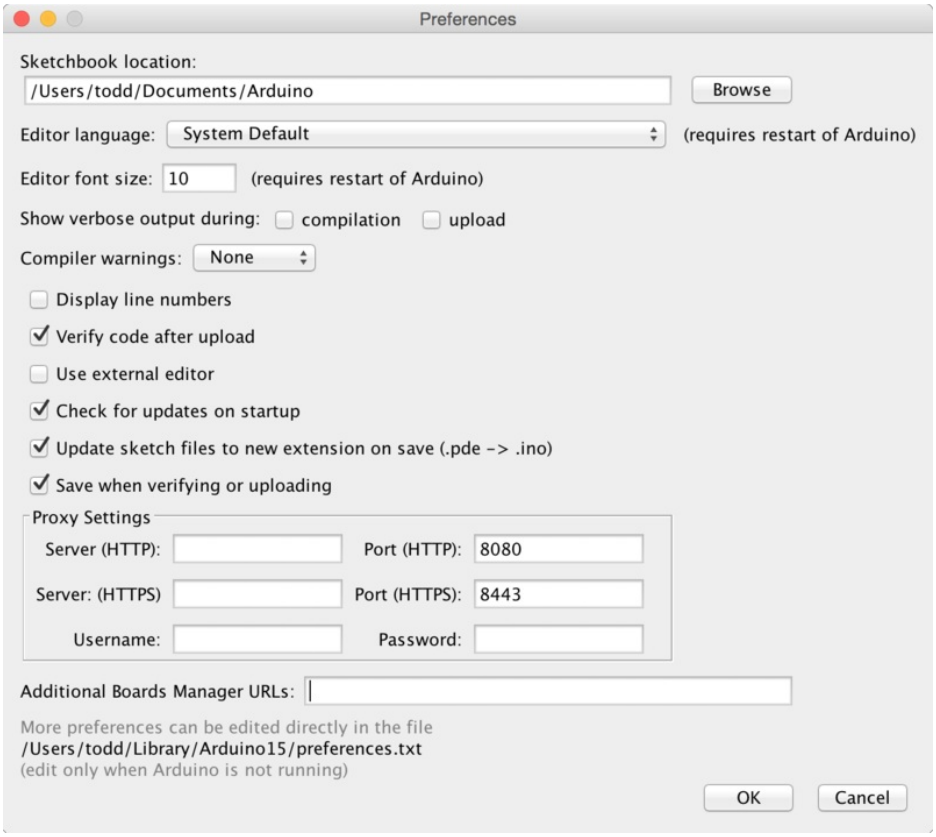
https://adafru.it/f1P

https://adafru.it/f1P

After you have downloaded and installed **the latest version of Arduino IDE**, you will need to start the IDE and navigate to the **Preferences** menu. You can access it from the **File** menu in *Windows* or *Linux*, or the **Arduino** menu on *OS X*.



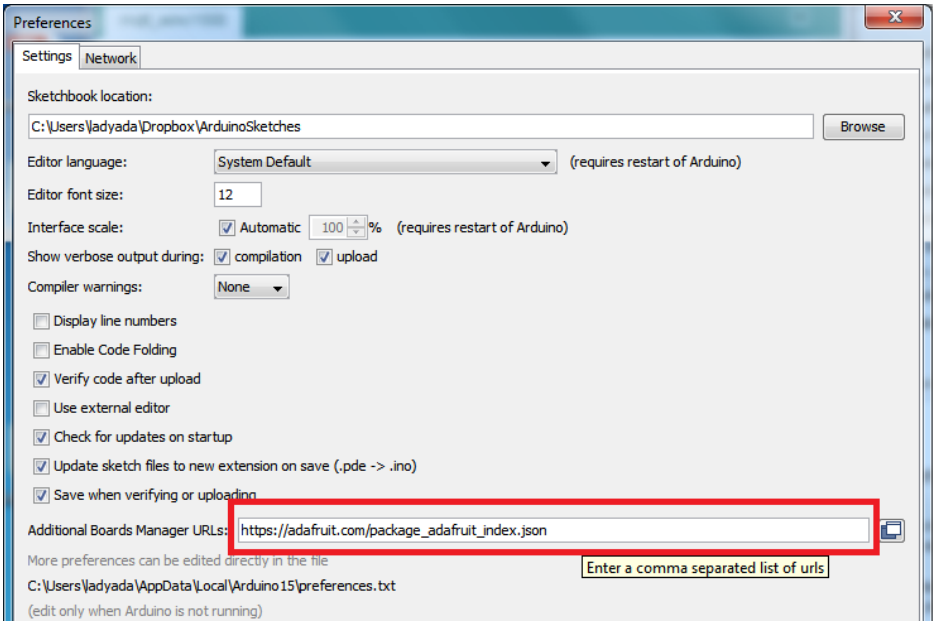
A dialog will pop up just like the one shown below.



We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and *you will only have to add each URL once.* New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list of third party board URLs on the Arduino IDE wiki (https://adafru.it/f7U). We will only need to add one URL to the IDE in this example, but ***you can add multiple URLS by separating them with commas***. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

```
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json
```



Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0 and M4, Metro M0 and M4, ItsyBitsy M0 and M4, Circuit Playground Express, Gemma M0 and Trinket M0
- **Arduino Leonardo & Micro MIDI-USB** - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the arcore project (https://adafru.it/eSI).

If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

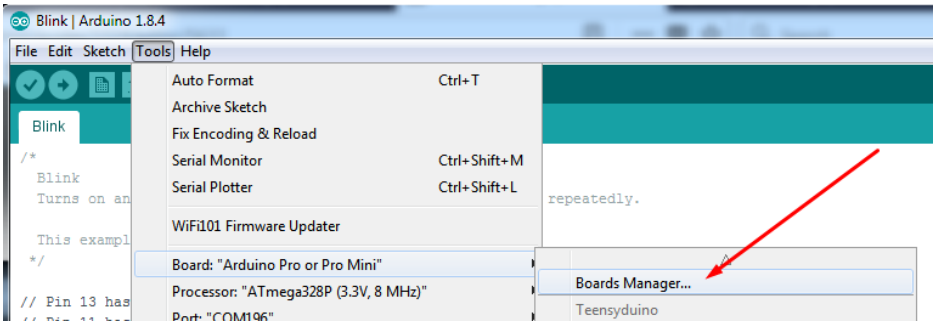
Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

Now continue to the next step to actually install the board support package!

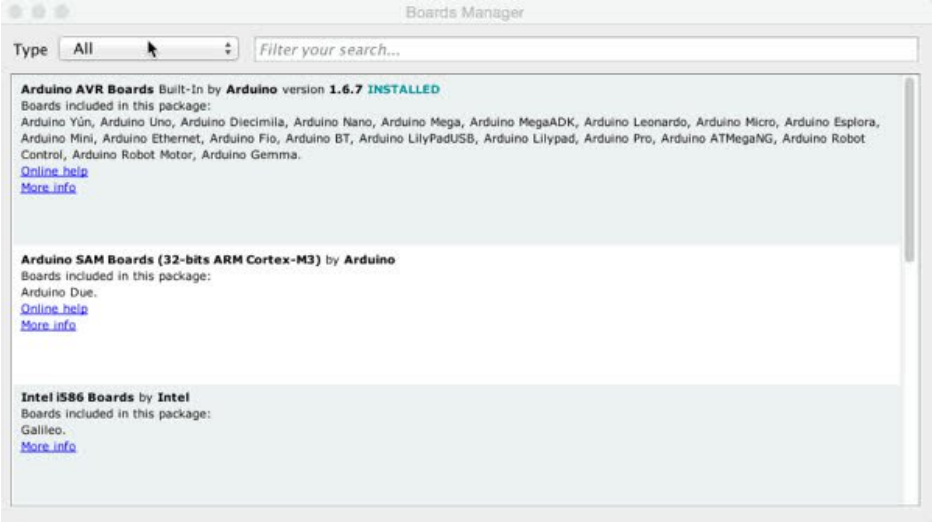
Using with Arduino IDE

Since the Feather 32u4 uses an ATmega32u4 chip running at 8 MHz, you can pretty easily get it working with the Arduino IDE. Many libraries (including the popular ones like NeoPixels and display) work great with the '32u4 and 8 MHz clock speed.

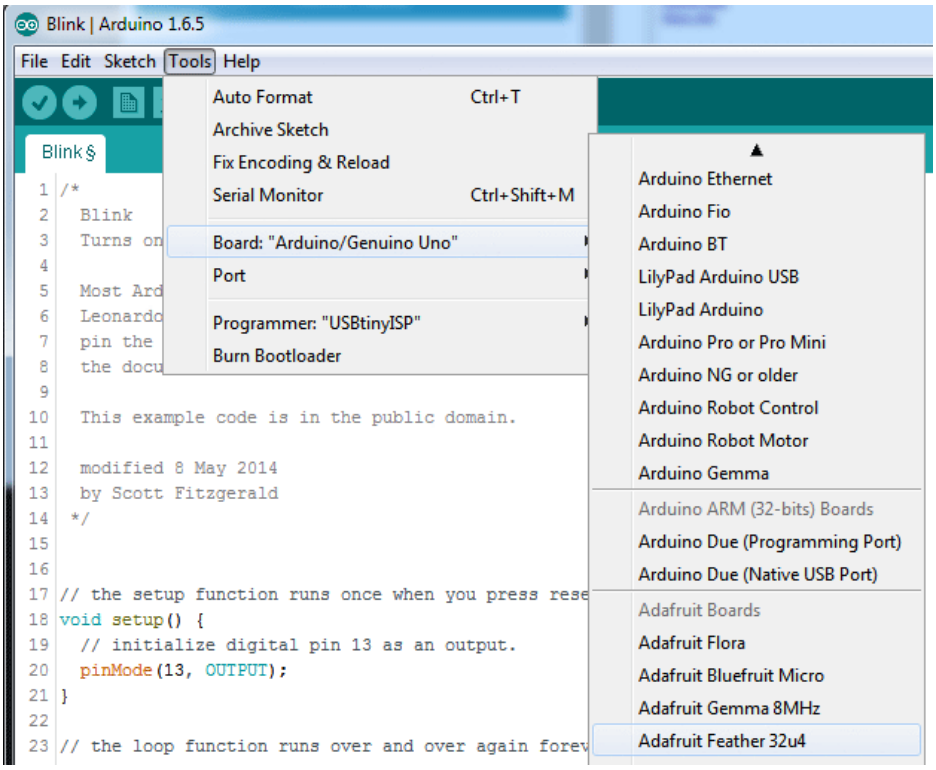
Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the **Boards Manager** by navigating to the **Tools->Board** menu.



Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the prefrences. In the example below, we are installing support for ***Adafruit AVR Boards***, but the same applies to all boards installed with the Board Manager.



Next, **quit and reopen the Arduino IDE** to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the **Tools->Board** menu.



Install Drivers (Windows Only)

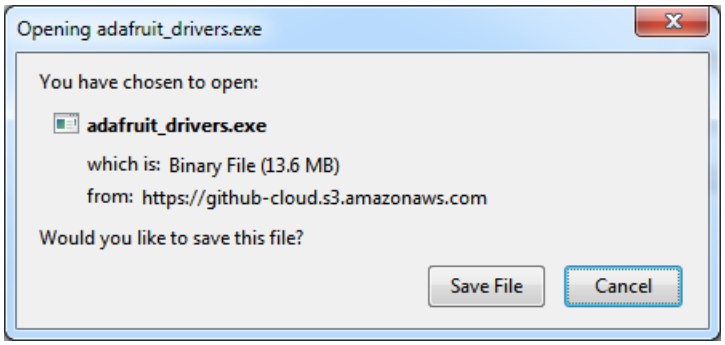
When you plug in the Feather, you'll need to possibly install a driver

Click below to download our Driver Installer

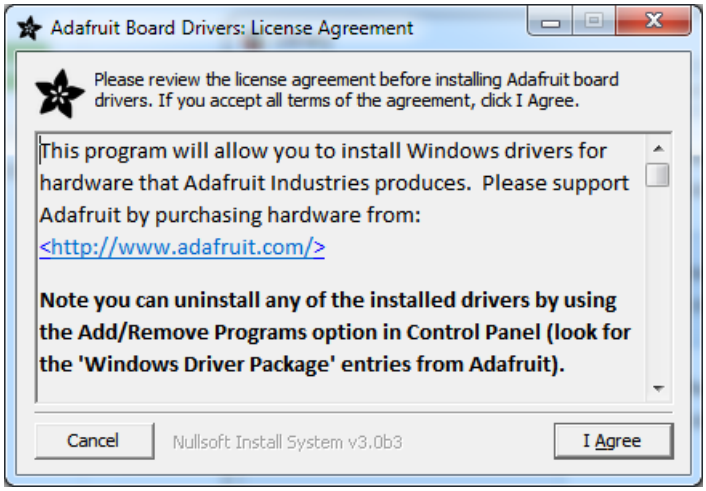
https://adafru.it/AB0

https://adafru.it/AB0

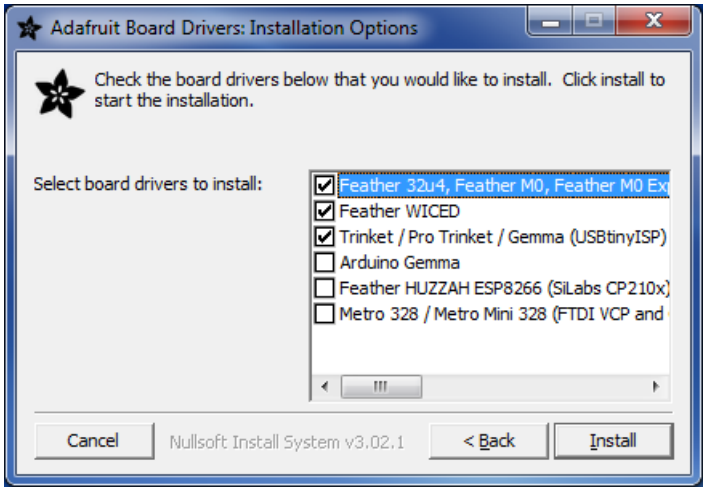
Download and run the installer



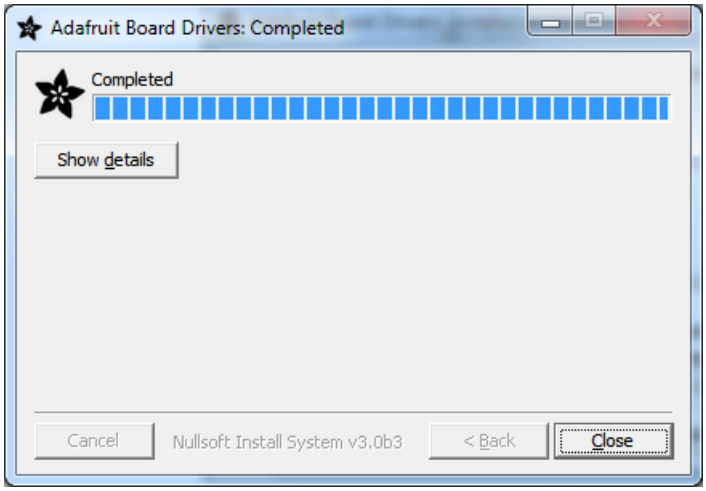
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install:



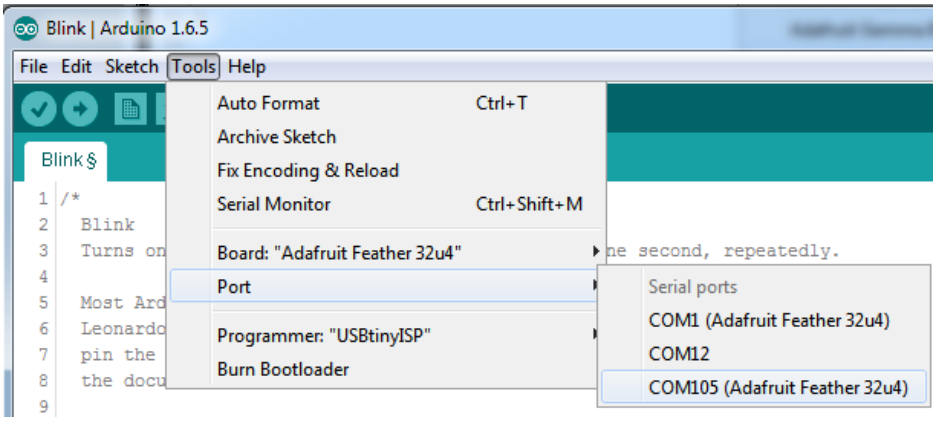
Click **Install** to do the installin'



Blink

Now you can upload your first blink sketch!

Plug in the Feather 32u4 and wait for it to be recognized by the OS (just takes a few seconds). It will create a serial/COM port, you can now select it from the dropdown, it'll even be 'indicated' as Feather 32u4!



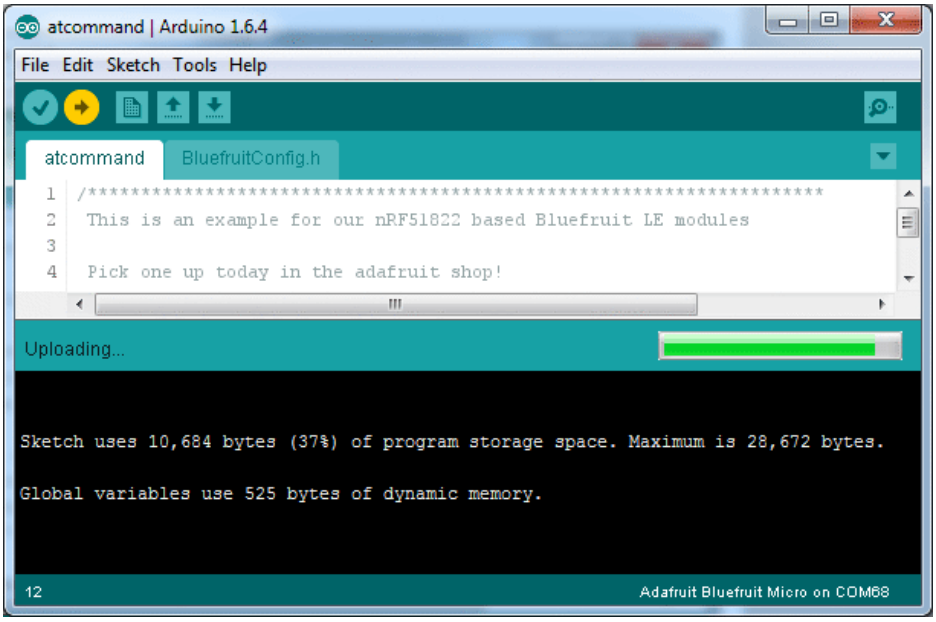
Now load up the Blink example

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

And click upload! That's it, you will be able to see the LED blink rate change as you adapt the **delay()** calls.

Manually bootloading

If you ever get in a 'weird' spot with the bootloader, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, **double-click the RST** button to get back into the bootloader. The red LED will pulse, so you know that its in bootloader mode. Do the reset button double-press right as the Arduino IDE says its attempting to upload the sketch, when you see the Yellow Arrow lit and the **Uploading...** text in the status bar.



Don't click the reset button **before** uploading, unlike other bootloaders you want this one to run at the time Arduino is trying to upload

Ubuntu & Linux Issue Fix

Note if you're using Ubuntu 15.04 (or perhaps other more recent Linux distributions) there is an issue with the modem manager service which causes the ATmega32u4 micro to be difficult to program. If you run into errors like "device or resource busy", "bad file descriptor", or "port is busy" when attempting to program then you are hitting this issue. (https://adafru.it/sHE)

The fix for this issue is to make sure Adafruit's custom udev rules are applied to your system. One of these rules is made to configure modem manager not to touch the board and will fix the programming difficulty issue. Follow the steps for installing Adafruit's udev rules on this page. (https://adafru.it/iOE)

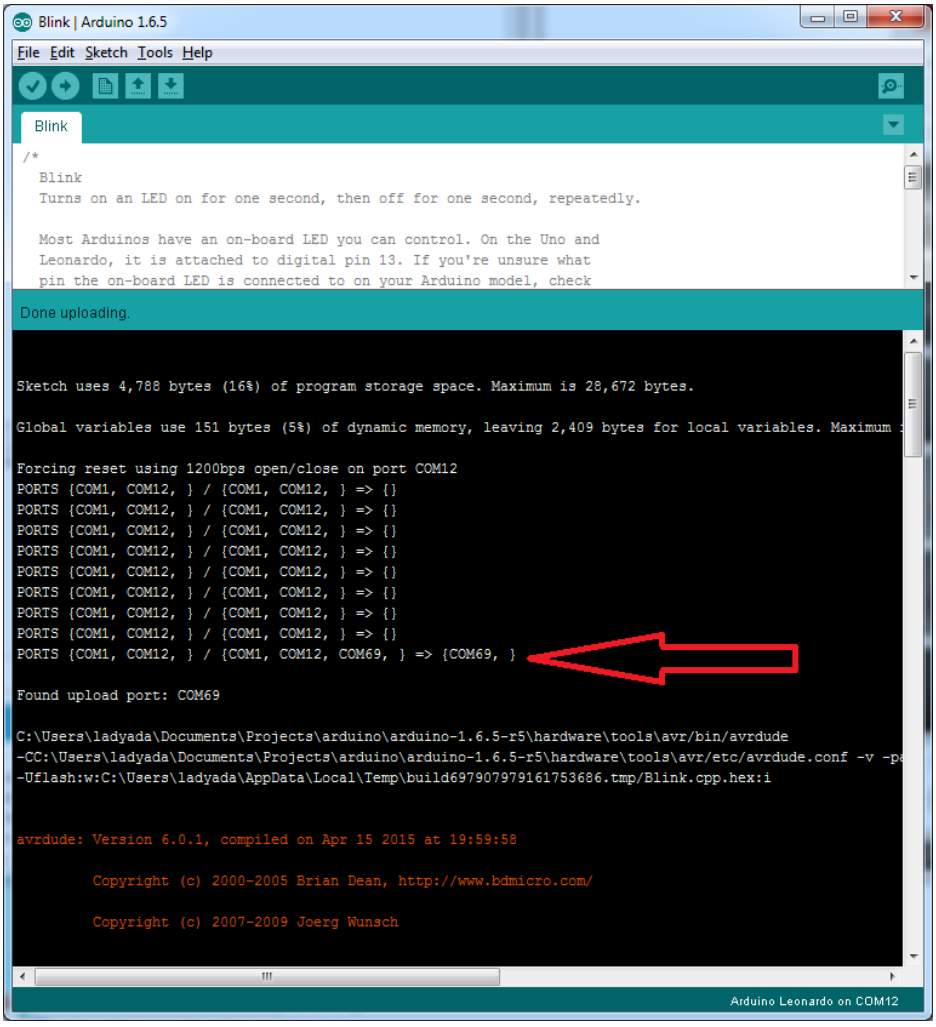
Feather HELP!

Even though this FAQ is labeled for Feather, the questions apply to ItsyBitsy's as well!

My ItsyBitsy/Feather stopped working when I unplugged the USB!

My Feather never shows up as a COM or Serial port in the Arduino IDE

- Ack! I "did something" and now when I plug in the Itsy/Feather, it doesn't show up as a device anymore so I cant upload to it or fix it...
  -



I can't get the Itsy/Feather USB device to show up - I get "USB Device Malfunctioning" errors!

I'm having problems with COM ports and my Itsy/Feather 32u4/M0

- I don't understand why the COM port disappears, this does not happen on my Arduino UNO!

- I'm trying to upload to my 32u4, getting "avrdude: butterfly_recv(): programmer is not responding" errors

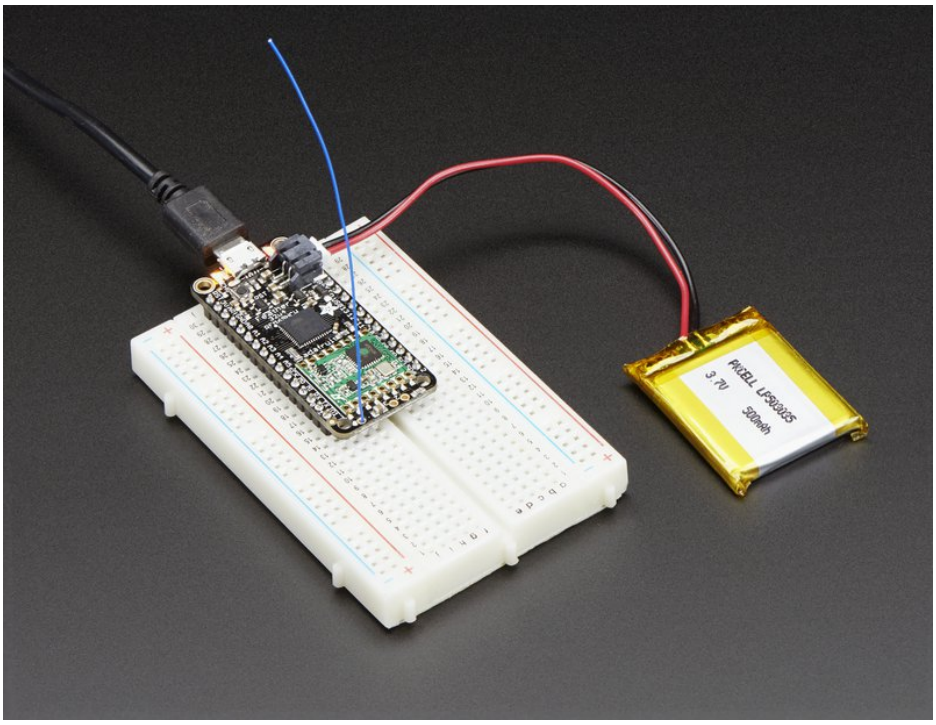
I'm trying to upload to my Feather M0, and I get this error "Connecting to programmer: .avrdude: butterfly_recv(): programmer is not responding"

- I'm trying to upload to my Feather and i get this error "avrdude: ser_recv(): programmer is not responding"
  -

I attached some wings to my Feather and now I can't read the battery voltage!

- The yellow LED Is flickering on my Feather, but no battery is plugged in, why is that?

Using the RFM9X Radio



Before beginning make sure you have your Feather working smoothly, it will make this part a lot easier. Once you have the basic Feather functionality going - you can upload code, blink an LED, use the serial output, etc. you can then upgrade to using the radio itself.

Note that the sub-GHz radio is not designed for streaming audio or video! It's best used for small packets of data. The data rate is adjustbale but its common to stick to around 19.2 Kbps (thats bits per second). Lower data rates will be more successful in their transmissions

You will, of course, need at least two paired radios to do any testing! The radios must be matched in frequency (e.g. 900 MHz & 900 MHz are ok, 900 MHz & 433 MHz are not). They also must use the same encoding schemes, you cannot have a 900 MHz RFM69 packet radio talk to a 900 MHz RFM96 LoRa radio.

Arduino Library

These radios have really excellent code already written, so rather than coming up with a new standard we suggest using existing libraries such as AirSpayce's Radiohead library (https://adafru.it/mCA) which also supports a vast number of other radios

This is a really great Arduino Library, so please support them in thanks for their efforts!

RadioHead RFM9x Library example

To begin talking to the radio, you will need to download the RadioHead library (https://adafru.it/mCA). You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip corresponding to version 1.62

Note that while all the code in the examples below are based on this version you can visit the RadioHead documentation page to get the most recent version which may have bug-fixes or more

functionality (https://adafru.it/mCA)

https://adafru.it/q6f

https://adafru.it/q6f

Uncompress the zip and find the folder named **RadioHead** and check that the **RadioHead** folder contains **RH_RF95.cpp** and **RH_RF95.h** (as well as a few dozen other files for radios that are supported)

Place the **RadioHead** library folder your **arduinosketchfolder/libraries/** folder.
You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

Basic RX & TX example

Lets get a basic demo going, where one Feather transmits and the other receives. We'll start by setting up the transmitter

Transmitter example code

This code will send a small packet of data once a second to node address #1

Load this code into your Transmitter Arduino/Feather!

Before uploading, check for the #define RF95_FREQ 915.0 line and change that to 433.0 if you are using the 433MHz version of the LoRa radio!

Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

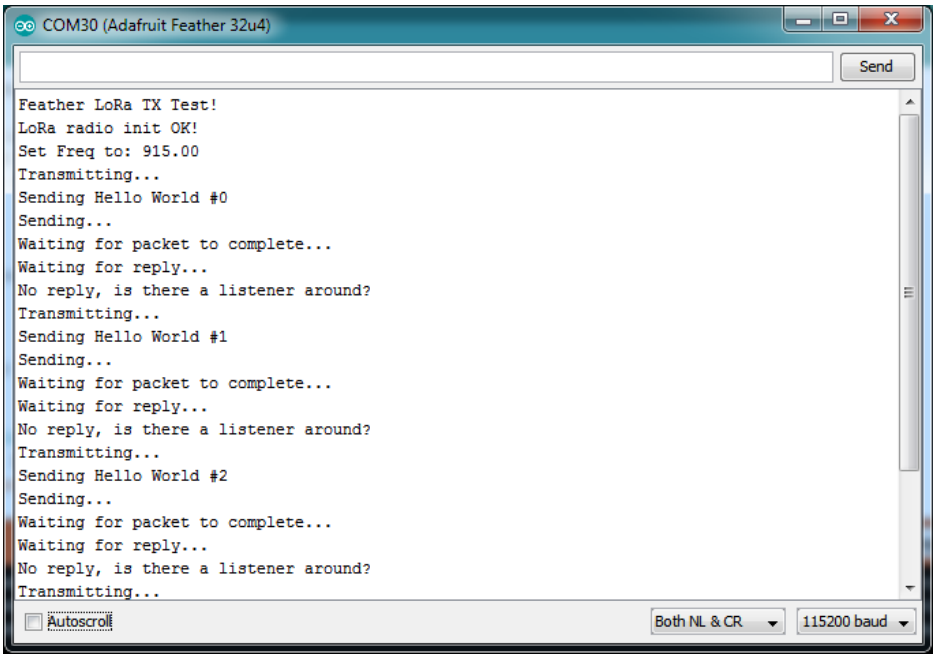
```
// Feather9x_TX
// -*- mode: C++ -*-
// Example sketch showing how to create a simple messaging client (transmitter)
// with the RH_RF95 class. RH_RF95 class does not provide for addressing or
// reliability, so you should only use RH_RF95 if you do not need the higher
// level messaging abilities.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

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

/* for feather32u4
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7
*/

/* for feather m0
#define RFM95_CS 8
```

```
#define RFM95_RST 4
#define RFM95_INT 3
*/

/* for shield
#define RFM95_CS 10
#define RFM95_RST 9
#define RFM95_INT 7
*/

/* Feather 32u4 w/wing
#define RFM95_RST     11   // "A"
#define RFM95_CS      10   // "B"
#define RFM95_INT     2    // "SDA" (only SDA/SCL/RX/TX have IRQ!)
*/

/* Feather m0 w/wing
#define RFM95_RST     11   // "A"
#define RFM95_CS      10   // "B"
#define RFM95_INT     6    // "D"
*/

#if defined(ESP8266)
  /* for ESP w/featherwing */
  #define RFM95_CS  2    // "E"
  #define RFM95_RST 16   // "D"
  #define RFM95_INT 15   // "B"

#elif defined(ESP32)
  /* ESP32 feather w/wing */
  #define RFM95_RST     27   // "A"
  #define RFM95_CS      33   // "B"
  #define RFM95_INT     12   //  next to A

#elif defined(NRF52)
  /* nRF52832 feather w/wing */
  #define RFM95_RST     7   // "A"
  #define RFM95_CS      11   // "B"
  #define RFM95_INT     31   // "C"

#elif defined(TEENSYDUINO)
  /* Teensy 3.x w/wing */
  #define RFM95_RST     9   // "A"
  #define RFM95_CS      10   // "B"
  #define RFM95_INT     4    // "C"
#endif


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

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

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

  Serial.begin(115200);
```

```
  while (!Serial) {
    delay(1);
  }

  delay(100);

  Serial.println("Feather LoRa TX Test!");

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

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

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

  // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

  // The default transmitter power is 13dBm, using PA_BOOST.
  // If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
  // you can set transmitter powers from 5 to 23 dBm:
  rf95.setTxPower(23, false);
}

int16_t packetnum = 0;  // packet counter, we increment per xmission

void loop()
{
  delay(1000); // Wait 1 second between transmits, could also 'sleep' here!
  Serial.println("Transmitting..."); // Send a message to rf95_server

  char radiopacket[20] = "Hello World #      ";
  itoa(packetnum++, radiopacket+13, 10);
  Serial.print("Sending "); Serial.println(radiopacket);
  radiopacket[19] = 0;

  Serial.println("Sending...");
  delay(10);
  rf95.send((uint8_t *)radiopacket, 20);

  Serial.println("Waiting for packet to complete...");
  delay(10);
  rf95.waitPacketSent();
  // Now wait for a reply
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  Serial.println("Waiting for reply...");
  if (rf95.waitAvailableTimeout(1000))
```

```
  if (rf95.waitAvailableTimeout(1000))
  {
    // Should be a reply message for us now
    if (rf95.recv(buf, &len))
   {
      Serial.print("Got reply: ");
      Serial.println((char*)buf);
      Serial.print("RSSI: ");
      Serial.println(rf95.lastRssi(), DEC);
    }
    else
    {
      Serial.println("Receive failed");
    }
  }
  else
  {
    Serial.println("No reply, is there a listener around?");
  }

}
```

Once uploaded you should see the following on the serial console



Now open up another instance of the Arduino IDE - this is so you can see the serial console output from the TX Feather while you set up the RX Feather.

Receiver example code

This code will receive and acknowledge a small packet of data.

Load this code into your **Receiver** Arduino/Feather!

Make sure the #define RF95_FREQ 915.0 matches your transmitter Feather!

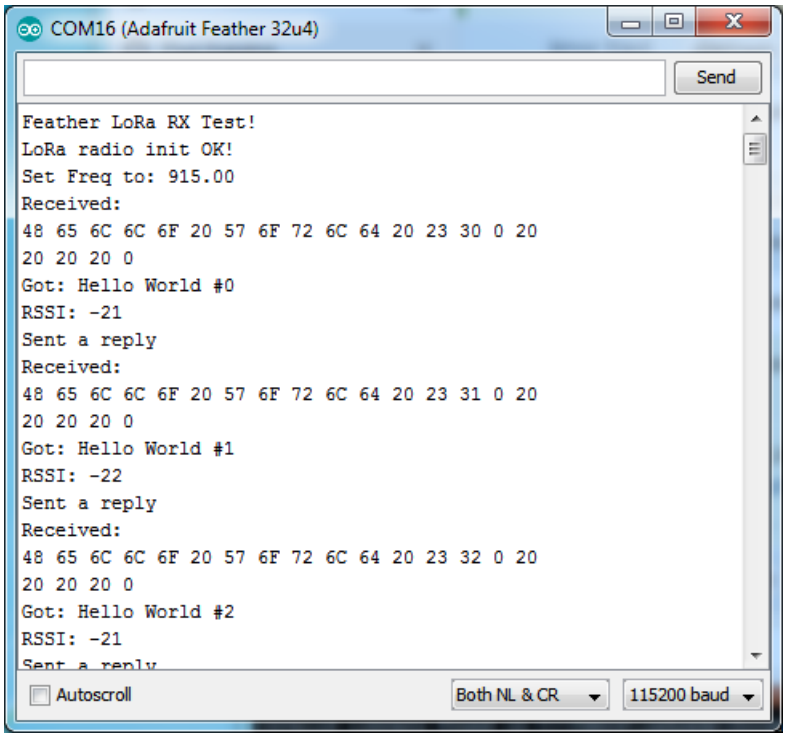
Uncomment/comment the sections defining the pins for Feather 32u4, Feather M0, etc depending on which chipset and wiring you are using! The pins used will vary depending on your setup!

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

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

/* for Feather32u4 RFM9x
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7
*/

/* for feather m0 RFM9x
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3
*/

/* for shield
#define RFM95_CS 10
#define RFM95_RST 9
#define RFM95_INT 7
*/

/* Feather 32u4 w/wing
#define RFM95_RST     11   // "A"
#define RFM95_CS      10   // "B"
#define RFM95_INT     2    // "SDA" (only SDA/SCL/RX/TX have IRQ!)
*/

/* Feather m0 w/wing
#define RFM95_RST     11   // "A"
#define RFM95_CS      10   // "B"
#define RFM95_INT     6    // "D"
*/

#if defined(ESP8266)
  /* for ESP w/featherwing */
  #define RFM95_CS  2    // "E"
  #define RFM95_RST 16   // "D"
  #define RFM95_INT 15   // "B"

#elif defined(ESP32)
  /* ESP32 feather w/wing */
  #define RFM95_RST     27   // "A"
  #define RFM95_CS      33   // "B"
  #define RFM95_INT     12   //  next to A
```

```
#elif defined(NRF52)
  /* nRF52832 feather w/wing */
  #define RFM95_RST     7   // "A"
  #define RFM95_CS      11   // "B"
  #define RFM95_INT     31   // "C"

#elif defined(TEENSYDUINO)
  /* Teensy 3.x w/wing */
  #define RFM95_RST     9   // "A"
  #define RFM95_CS      10   // "B"
  #define RFM95_INT     4    // "C"
#endif


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

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// Blinky on receipt
#define LED 13

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  Serial.begin(115200);
  while (!Serial) {
    delay(1);
  }
  delay(100);

  Serial.println("Feather LoRa RX Test!");

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

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

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

  // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

  // The default transmitter power is 13dBm, using PA_BOOST.
```

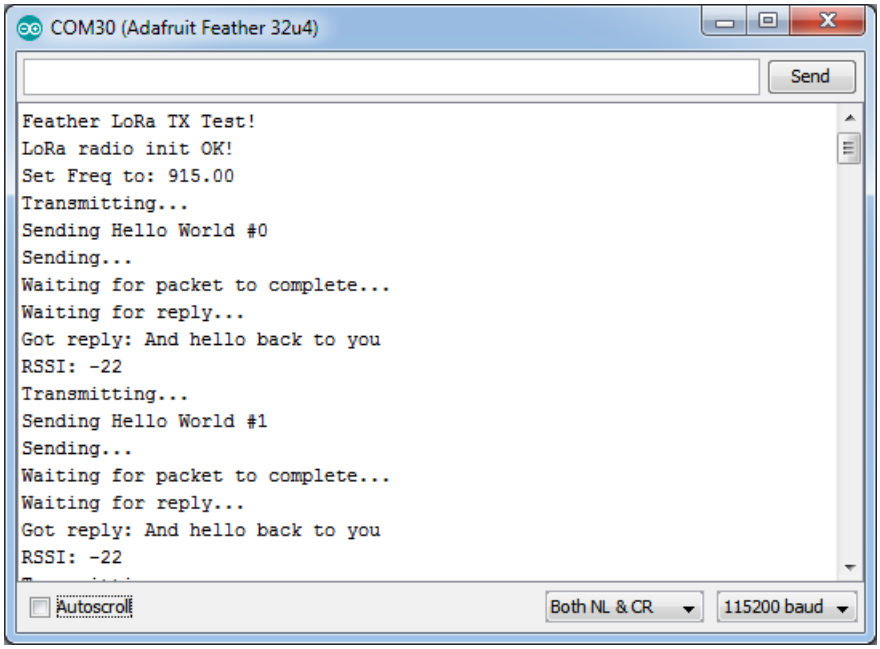
```
  // If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
  // you can set transmitter powers from 5 to 23 dBm:
  rf95.setTxPower(23, false);
}

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

    if (rf95.recv(buf, &len))
    {
      digitalWrite(LED, HIGH);
      RH_RF95::printBuffer("Received: ", buf, len);
      Serial.print("Got: ");
      Serial.println((char*)buf);
       Serial.print("RSSI: ");
      Serial.println(rf95.lastRssi(), DEC);

      // Send a reply
      uint8_t data[] = "And hello back to you";
      rf95.send(data, sizeof(data));
      rf95.waitPacketSent();
      Serial.println("Sent a reply");
      digitalWrite(LED, LOW);
    }
    else
    {
      Serial.println("Receive failed");
    }
  }
}
```

Now open up the Serial console on the receiver, while also checking in on the transmitter's serial console. You should see the receiver is...well, receiving packets



You can see that the library example prints out the hex-bytes received `48 65 6C 6C 6F 20 57 6F 72 6C 64 20 23 30 0 20 20 20 20 0`, as well as the ASCII 'string' `Hello World`. Then it will send a reply.

And, on the transmitter side, it is now printing that it got a reply after each transmisssion `And hello back to you` because it got a reply from the receiver



That's pretty much the basics of it! Lets take a look at the examples so you know how to adapt to your own radio setup

Feather Radio Pinout

This is the pinout setup for all **Feather 32u4** RFM9X's:

```
/* for feather32u4 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 7
```

This is the pinout for all **Feather M0** RFM9X's:

```
/* for feather m0 */
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3
```

Frequency

You can dial in the frequency you want the radio to communicate on, such as 915.0, 434.0 or 868.0 or any number really. Different countries/ITU Zones have different ISM bands so make sure you're using those or if you are licensed, those frequencies you may use

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

You can then instantiate the radio object with our custom pin numbers.

```
// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);
```

Setup

We begin by setting up the serial console and hard-resetting the Radio

```
void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  while (!Serial); // wait until serial console is open, remove if not tethered to computer
  Serial.begin(9600);
  delay(100);
  Serial.println("Feather LoRa RX Test!");

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

Remove the **while (!Serial);** line if you are not tethering to a computer, as it will cause the Feather to halt until a USB

connection is made!

Initializing Radio

The library gets initialized with a call to **init()**. Once initialized, you can set the frequency. You can also configure the output power level, the number ranges from 5 to 23. Start with the highest power level (23) and then scale down as necessary

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

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

  // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on

  // The default transmitter power is 13dBm, using PA_BOOST.
  // If you are using RFM95/96/97/98 modules which uses the PA_BOOST transmitter pin, then
  // you can set transmitter powers from 5 to 23 dBm:
  rf95.setTxPower(23, false);
```

Transmission Code

If you are using the transmitter, this code will wait 1 second, then transmit a packet with "Hello World #" and an incrementing packet number

```
void loop()
{
  delay(1000); // Wait 1 second between transmits, could also 'sleep' here!
  Serial.println("Transmitting..."); // Send a message to rf95_server

  char radiopacket[20] = "Hello World #      ";
  itoa(packetnum++, radiopacket+13, 10);
  Serial.print("Sending "); Serial.println(radiopacket);
  radiopacket[19] = 0;

  Serial.println("Sending..."); delay(10);
  rf95.send((uint8_t *)radiopacket, 20);

  Serial.println("Waiting for packet to complete..."); delay(10);
  rf95.waitPacketSent();
```

Its pretty simple, the delay does the waiting, you can replace that with low power sleep code. Then it generates the packet and appends a number that increases every tx. Then it simply calls **send** to transmit the data, and passes in the array of data and the length of the data.

Note that this does not any addressing or subnetworking - if you want to make sure the packet goes to a particular

radio, you may have to add an identifier/address byte on your own!

Then you call **waitPacketSent()** to wait until the radio is done transmitting. You will not get an automatic acknowledgement, from the other radio unless it knows to send back a packet. Think of it like the 'UDP' of radio - the data is sent, but its not certain it was received! Also, there will not be any automatic retries.

Receiver Code

The Receiver has the same exact setup code, but the loop is different

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

    if (rf95.recv(buf, &len))
    {
      digitalWrite(LED, HIGH);
      RH_RF95::printBuffer("Received: ", buf, len);
      Serial.print("Got: ");
      Serial.println((char*)buf);
       Serial.print("RSSI: ");
      Serial.println(rf95.lastRssi(), DEC);
```

Instead of transmitting, it is constantly checking if there's any data packets that have been received.**available()** will return true if a packet with proper error-correction was received. If so, the receiver prints it out in hex and also as a 'character string'

It also prints out the RSSI which is the receiver signal strength indicator. This number will range from about -15 to about -100. The larger the number (-15 being the highest you'll likely see) the stronger the signal.

Once done it will automatically reply, which is a way for the radios to know that there was an acknowledgement

```
// Send a reply
      uint8_t data[] = "And hello back to you";
      delay(200);
      rf95.send(data, sizeof(data));
      rf95.waitPacketSent();
      Serial.println("Sent a reply");
```

It simply sends back a string and waits till the reply is completely sent

Radio Range F.A.Q.

- Which gives better range, LoRa or RFM69?

What ranges can I expect for RFM69 radios?

What ranges can I expect for RFM9X LoRa radios?

I don't seem to be getting the range advetised! Is my module broken?

- How do I pick/design the right antenna?

What frequency is my module?

My radio has a burnt blob on it, is it damaged?

Downloads

Datasheets & Files

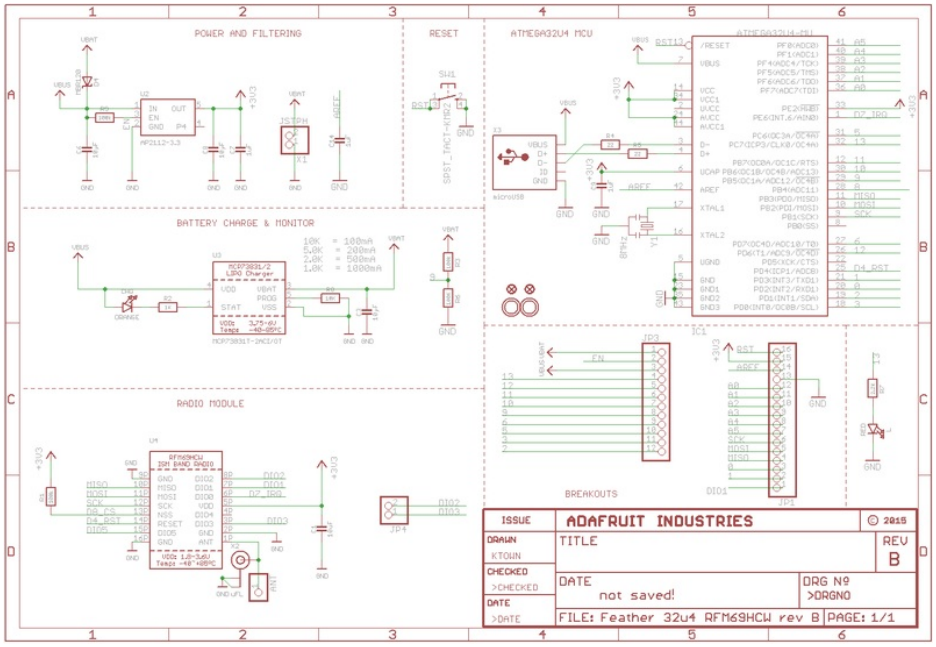
- SX127x Datasheet (https://adafru.it/BjG) - The radio chip itself
- RFM9X (https://adafru.it/mFX) - The radio module, which contains the SX1272 chipset
- FCC Test Report (https://adafru.it/qla) - 13dBm
- ETSI Test Report (https://adafru.it/qlb)
- CE Report (https://adafru.it/qlc)
- FCC Test Report (https://adafru.it/qld)
- RoHS Test Report (https://adafru.it/qld)
- EagleCAD PCB Files on GitHub (https://adafru.it/obt)
- Fritzing object in Adafruit Fritzing library (https://adafru.it/c7M)

https://adafru.it/z3C

https://adafru.it/z3C

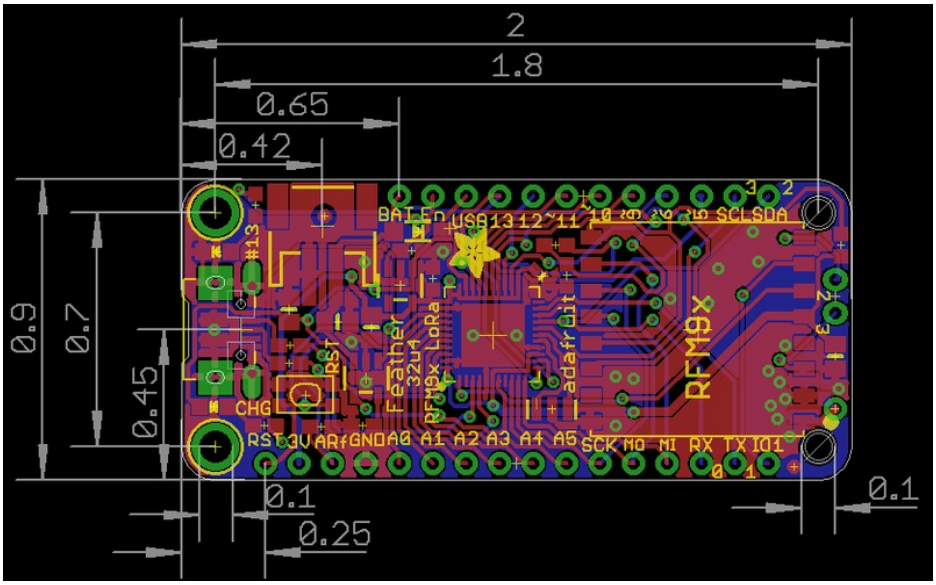
Schematic

The RFM69/RFM9x modules have identical pinouts



Fabrication Print

Dimensions in Inches


2
1.8
0.65
0.42
0.9
0.7
0.45
0.1
0.25
0.1
BAT_EN
USB
13 12 11 10 9 6 5 SCL SDA
#13
RST
CHG
Feather
32u4
RFM9x LoRa
adafruit
RFM9x
RST 3V AREF GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX IO1