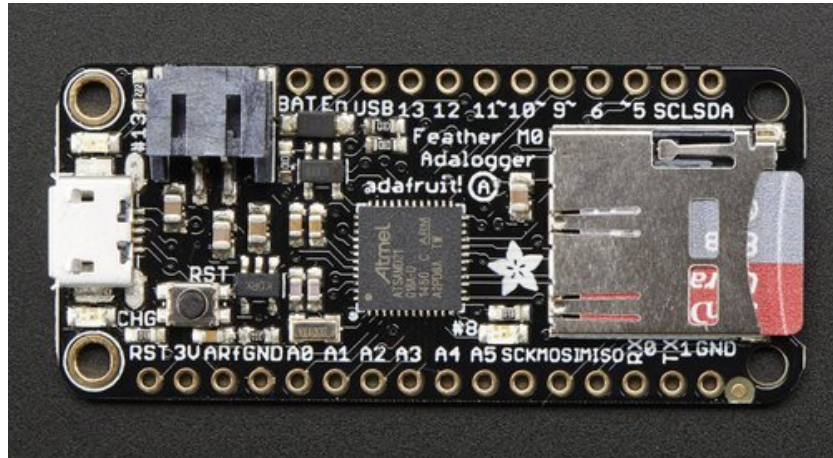


□

Adafruit Feather M0 Adalogger

Created by lady ada



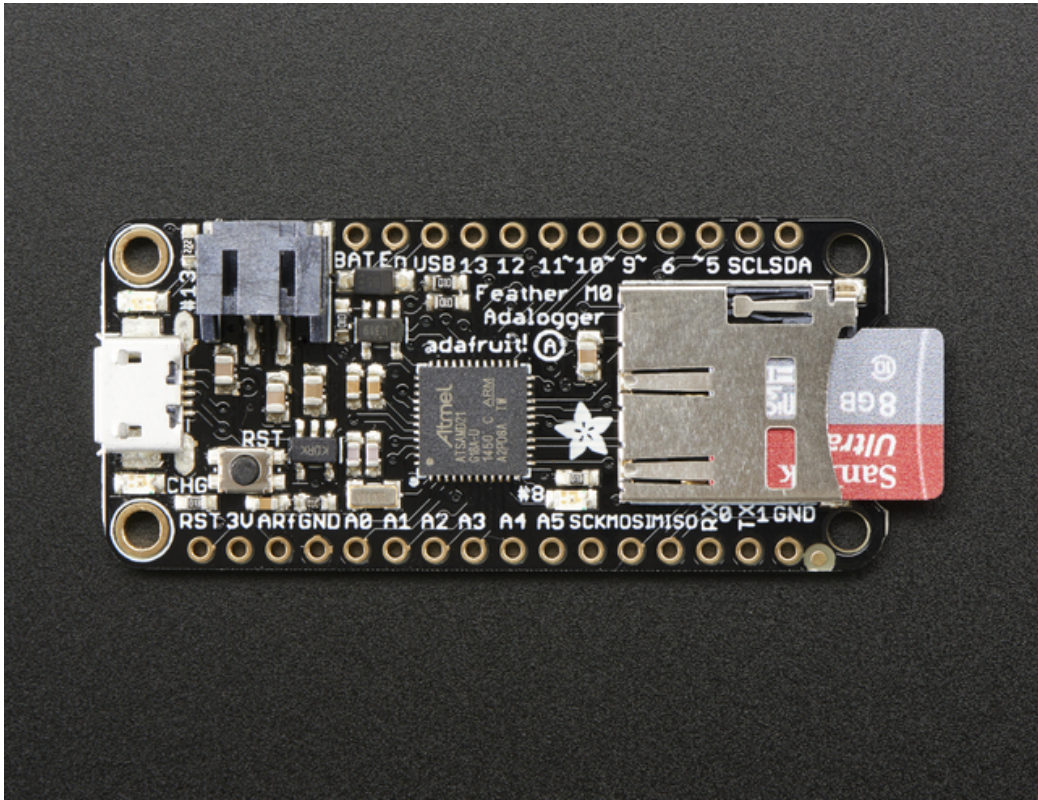
Last updated on 2016-09-19 08:22:57 PM UTC

Guide Contents

Guide Contents	2
Overview	4
Pinouts	8
Power Pins	9
Logic pins	9
Micro SD Card + Green LED	10
Other Pins!	10
Assembly	12
Header Options!	12
Soldering in Plain Headers	14
Prepare the header strip:	14
Add the breakout board:	15
And Solder!	15
Soldering on Female Header	17
Tape In Place	17
Flip & Tack Solder	18
And Solder!	19
Power Management	21
Battery + USB Power	21
Power supplies	22
Measuring Battery	22
Average Power Draw w/SD Card	23
ENable pin	25
Arduino IDE Setup	27
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json	28
Using with Arduino IDE	30
Install SAMD Support	30
Install Adafruit SAMD	31
Install Drivers (Windows Only)	32
Blink	33
Sucessful Upload	34
Compilation Issues	34

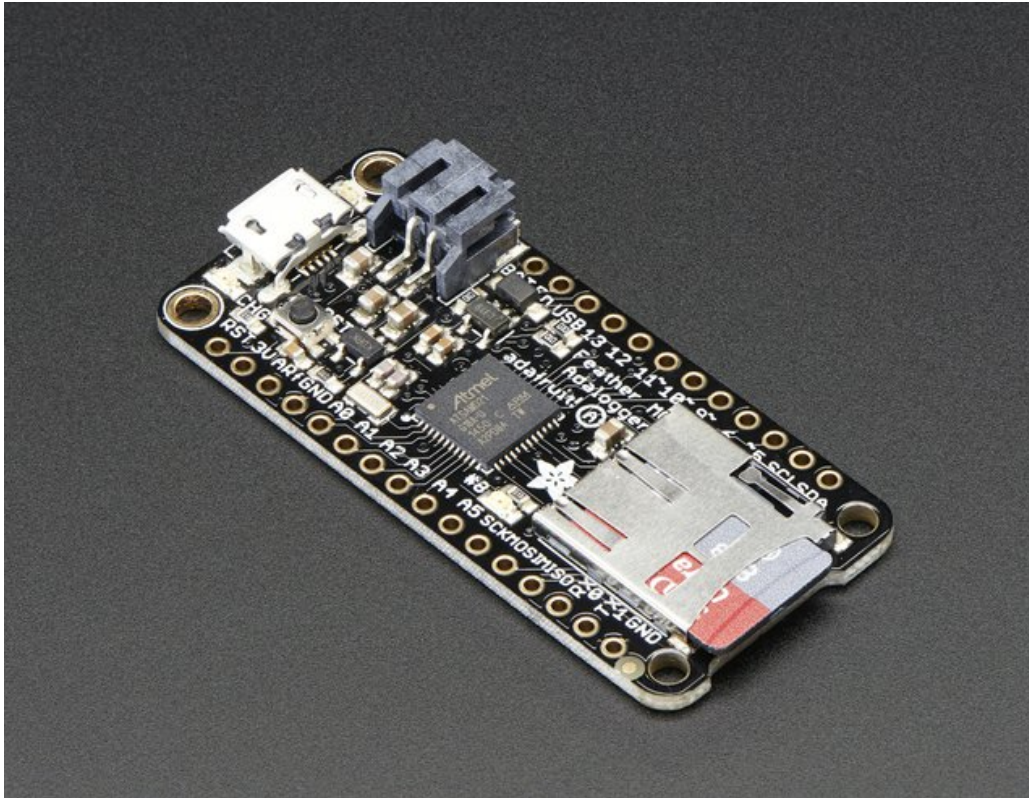
Manually bootloading	35
Ubuntu & Linux Issue Fix	35
Using the SD Card	36
Example logging sketch	37
Adapting Sketches to M0	39
Analog References	39
Pin Outputs & Pullups	39
Serial vs SerialUSB	39
AnalogWrite / PWM	40
Missing header files	40
Bootloader Launching	40
Aligned Memory Access	40
Floating Point Conversion	41
How Much RAM Available?	41
Storing data in FLASH	41
Downloads	43
Datasheets	43
Schematic	43
Fabrication Print	43
HELP!	45

Overview

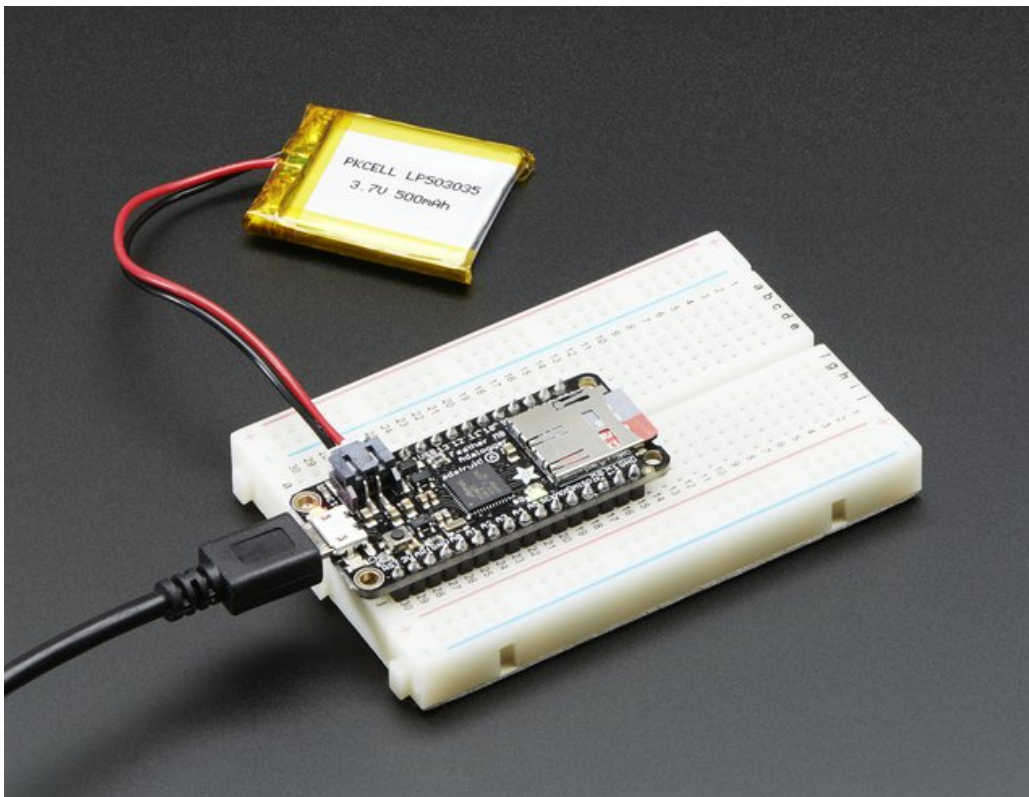


Feather is the new development board from Adafruit, and like it's namesake it is thin, light, and lets you fly! We designed Feather to be a new standard for portable microcontroller cores.

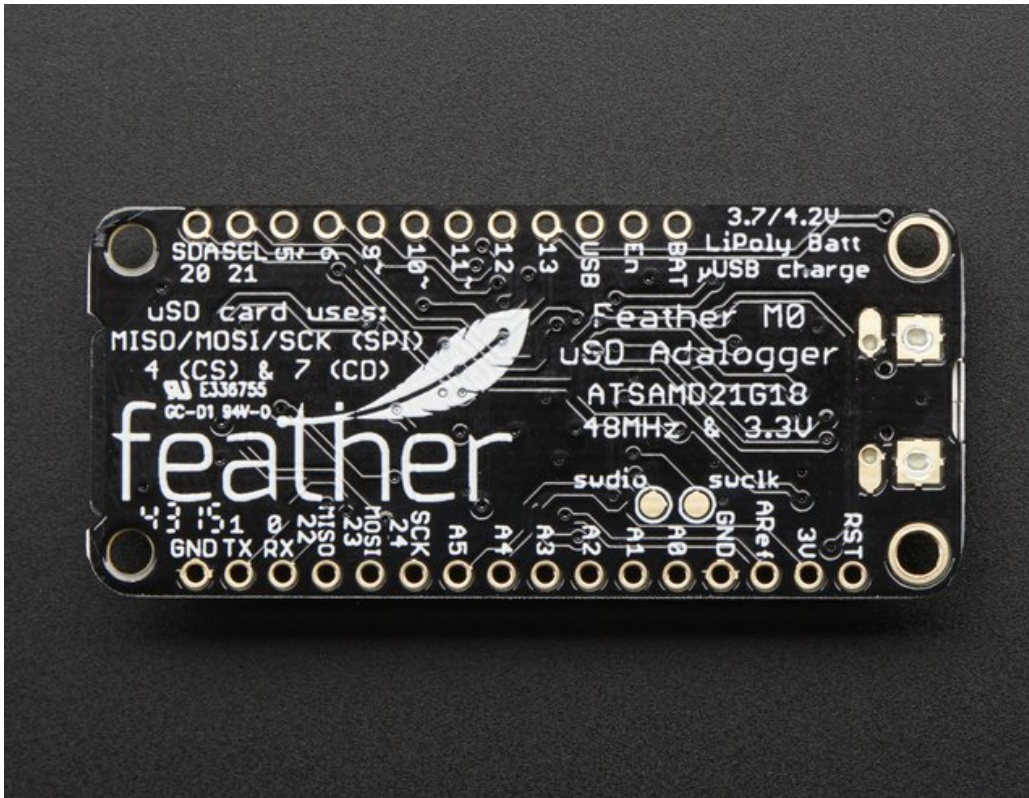
This is the **Adafruit Feather M0 Adalogger** - our take on an 'all-in-one' Cortex M0 datalogger (or data-reader) with built in USB and battery charging. Its an Adafruit Feather M0 with a microSD holder ready to rock! [We have other boards in the Feather family, check'em out here \(http://adafru.it/jAQ\)](http://adafru.it/jAQ)



At the Feather M0's heart is an ATSAM D21G18 ARM Cortex M0 processor, clocked at 48 MHz and at 3.3V logic, the same one used in the new [Arduino Zero](http://adafru.it/2843) (<http://adafru.it/2843>). This chip has a whopping 256K of FLASH (8x more than the Atmega328 or 32u4) and 32K of RAM (16x as much)! This chip comes with built in USB so it has USB-to-Serial program & debug capability built in with no need for an FTDI-like chip.



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.

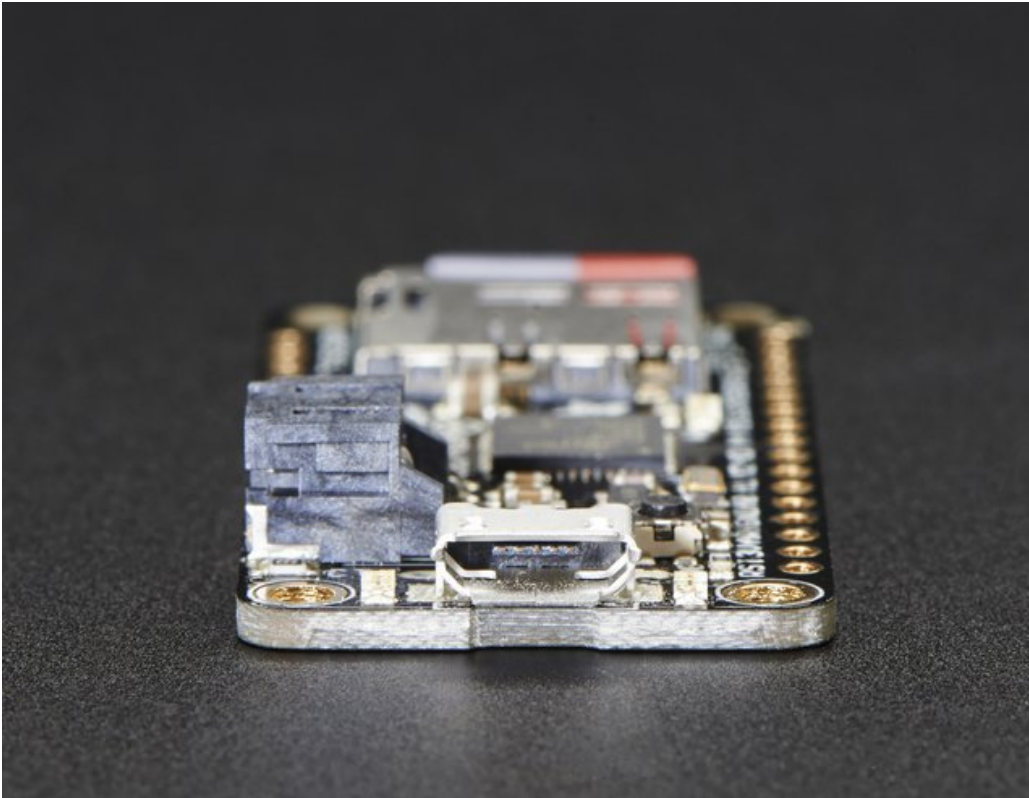


Here's some handy specs! Like all Feather M0's you get:

- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.3 grams
- ATSAM21G18 @ 48MHz with 3.3V logic/power
- 256KB of FLASH + 32KB of RAM
- 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
- 8 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 M0 Adalogger** uses the extra space left over to add MicroSD + a green LED:

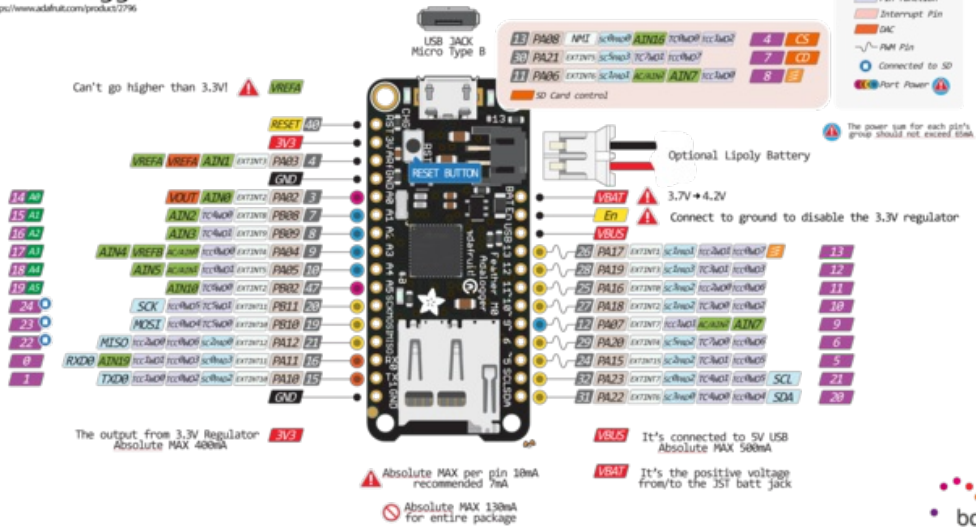
- Pin #8 green LED for your blinking pleasure
- MicroSD card holder for adding as much storage as you could possibly want, for reading or writing.



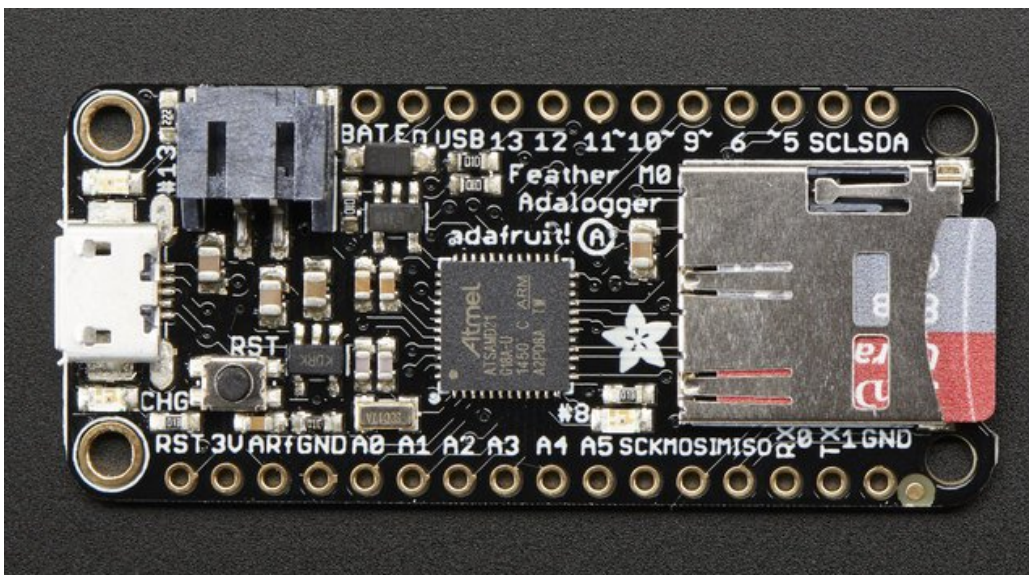
Comes fully assembled and tested, with a USB bootloader that lets you quickly use it with the Arduino IDE. We also toss in some header so you can solder it in and plug into a solderless breadboard. **Lipoly battery, MicroSD card and USB cable not included** (but we do have lots of options in the shop if you'd like!)

Check out our tutorial for all sorts of details, including schematics, files, IDE instructions, and more!

Pinouts

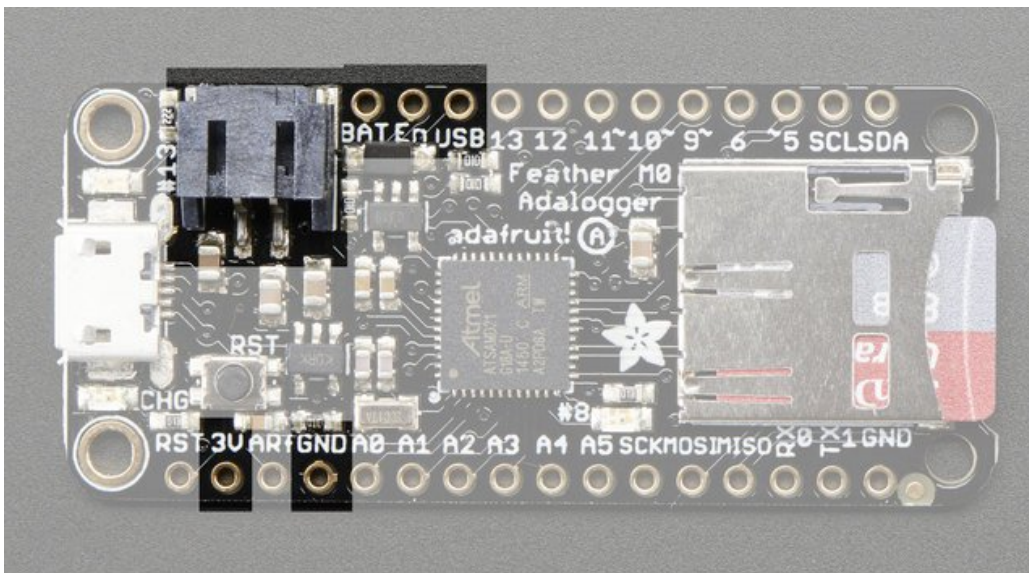


The Feather M0 Adalogger is chock-full of microcontroller goodness. There's also a lot of pins and ports. We'll take you a tour of them now!





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

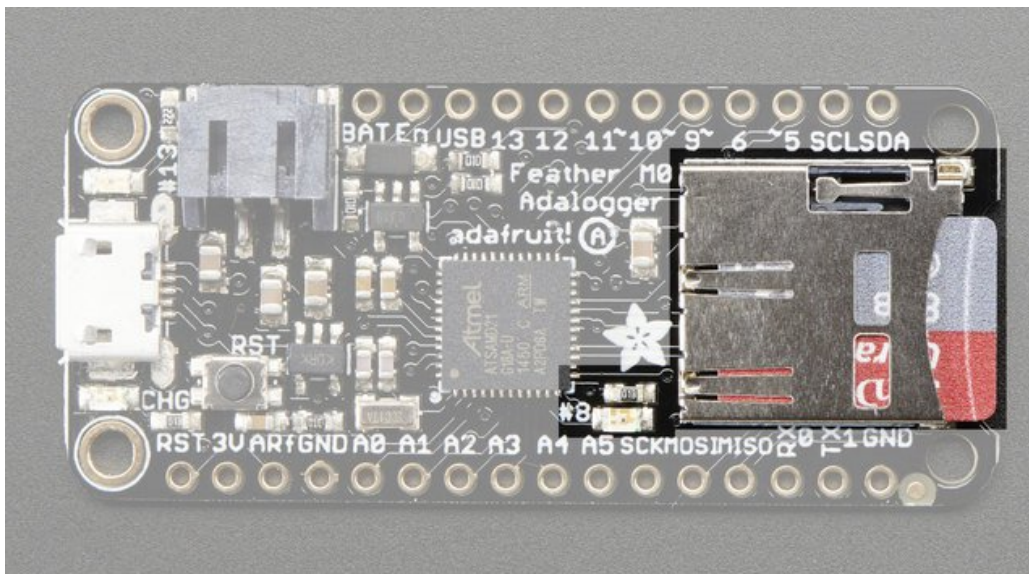
All pins can do PWM output

All pins can be interrupt inputs

- **#0 / RX** - GPIO #0, also receive (input) pin for **Serial1** (hardware UART), also can be analog input
- **#1 / TX** - GPIO #1, also transmit (output) pin for **Serial1**, also can be analog input

- **#20 / SDA** - GPIO #20, 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.
- **#21 / SCL** - GPIO #21, 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.
- **#5** - GPIO #5
- **#6** - GPIO #6
- **#9** - GPIO #9, also analog input **A7**. 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
- **#11** - GPIO #11
- **#12** - GPIO #12
- **#13** - GPIO #13 and is connected to the **red LED** next to the USB jack
- **A0** - This pin is analog *input* **A0** but is also an analog *output* due to having a DAC (digital-to-analog converter). You can set the raw voltage to anything from 0 to 3.3V, unlike PWM outputs this is a true analog output
- **A1 thru A5** - These are each analog input as well as digital I/O pins.
- **SCK/MOSI/MISO** (GPIO **24/23/22**)- These are the hardware SPI pins, you can use them as everyday GPIO pins (but recommend keeping them free as they are best used for hardware SPI connections for high speed.

Micro SD Card + Green LED



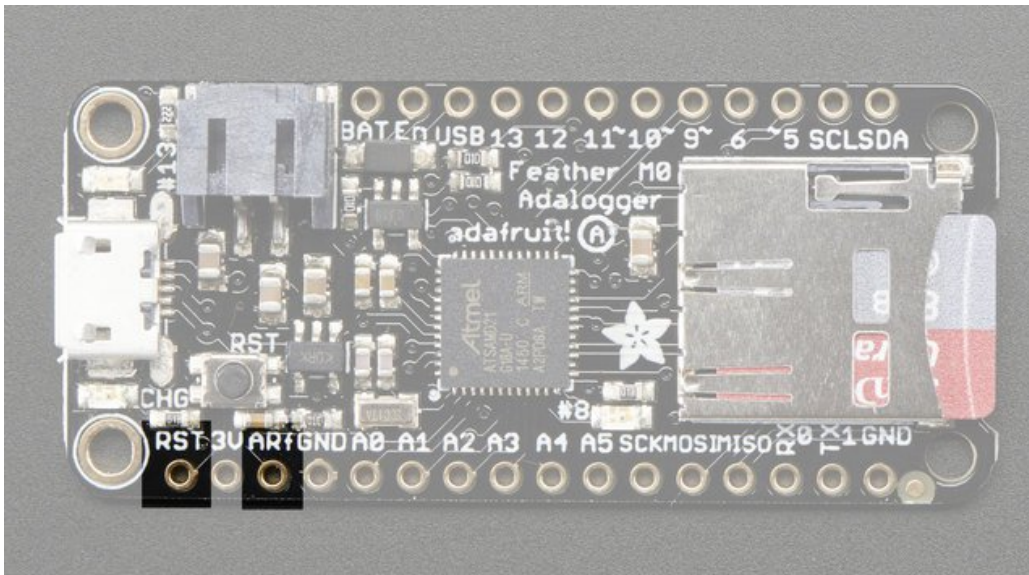
Since not all pins can be brought out to breakouts, due to the small size of the Feather, we use these to control the SD card!

- **#4** - used as the MicroSD card **CS** (chip select) pin
- **#7** - used as the MicroSD card **CD** (card detect) pin. If you want to detect when a card is inserted/removed, configure this pin as an input with a pullup. When the pin reads low (0V) then there is no card inserted. When the pin reads high, then a card is in place. It will not tell you if the card is valid, its just a mechanical switch
- **#8** - This pin was also left over, so we tied it to a green LED, its next to the SD card. It might be handy to blink this LED when writing / reading valid data or some other user-alert!

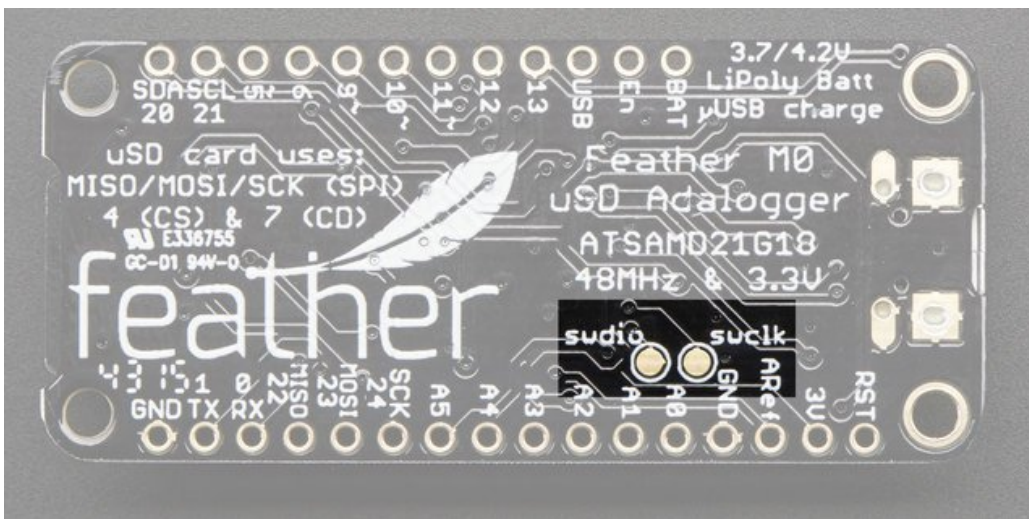
Other Pins!

- **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!



SWCLK & SWDIO - These pads on the bottom are used to program the chip. They can also be connected to an SWD debugger.

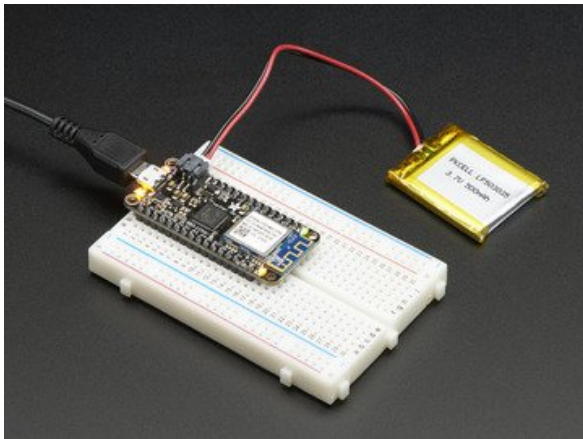


Assembly

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

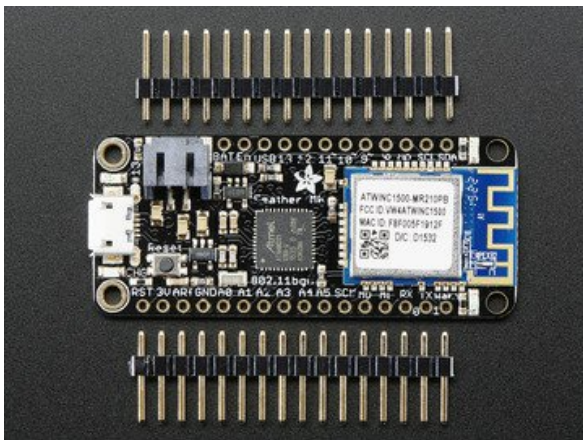
Header Options!

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

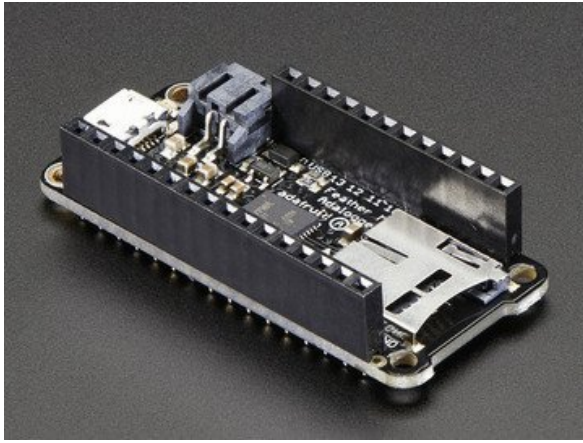


-

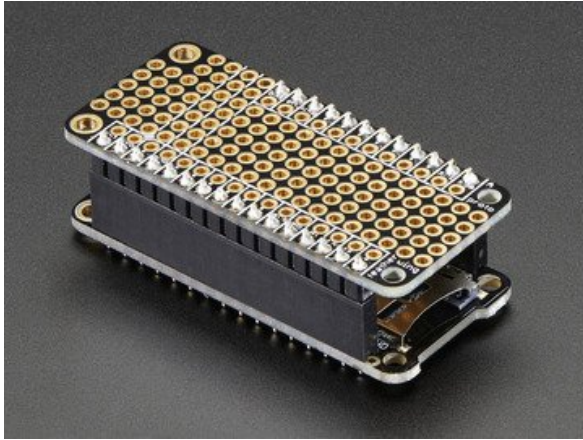
The first option is soldering in plain male headers, this lets you plug in the Feather into a solderless breadboard



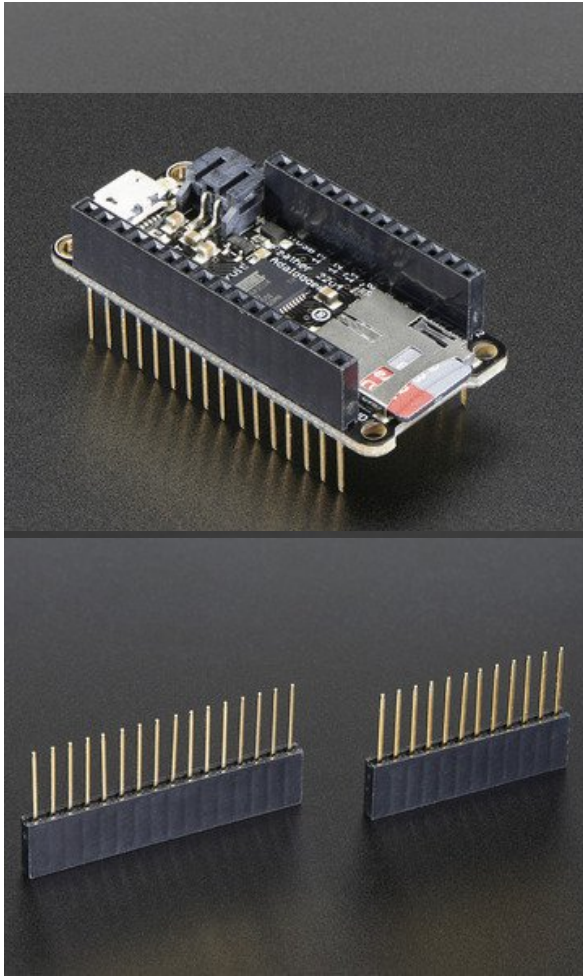
-



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

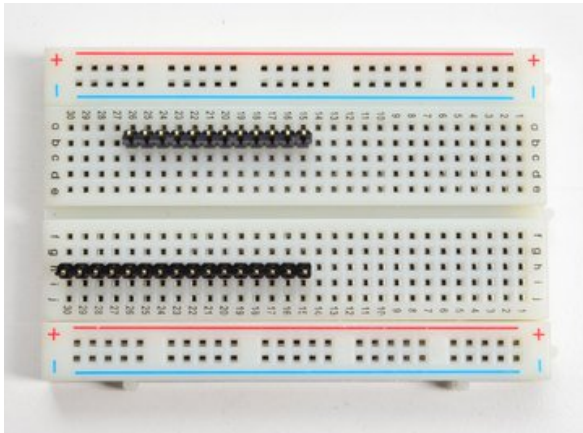


We also have 'slim' versions of the female headers, that are a little shorter and give a more compact shape

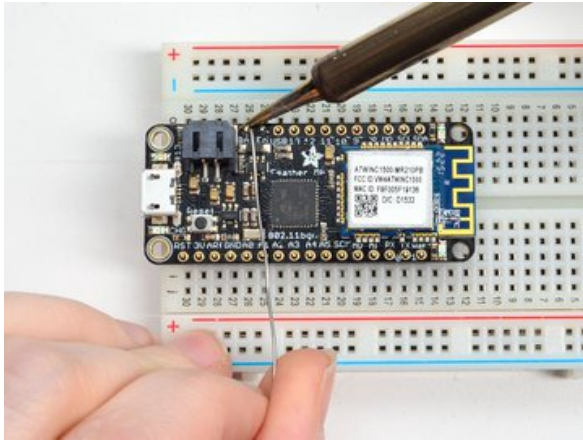


Finally, there's the "Stacking Header" option. This one is sort of the best-of-both-worlds. You get the ability to plug into a solderless breadboard *and* plug a featherwing on top. But its a little bulky

Soldering in Plain Headers

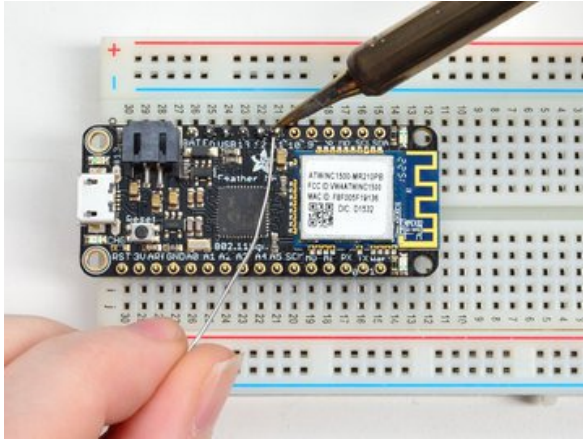


Prepare the header strip:
 Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



Add the breakout board:

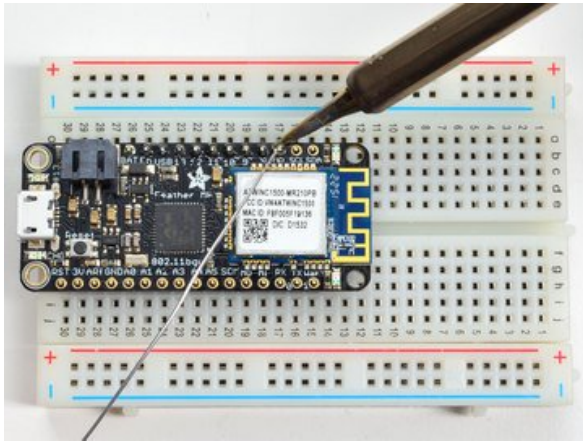
Place the breakout board over the pins so that the short pins poke through the breakout pads

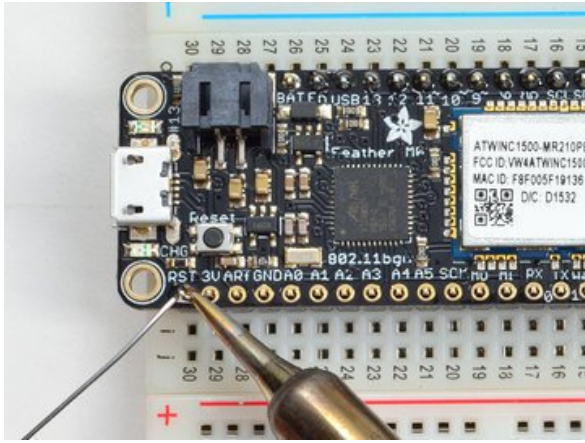


And Solder!

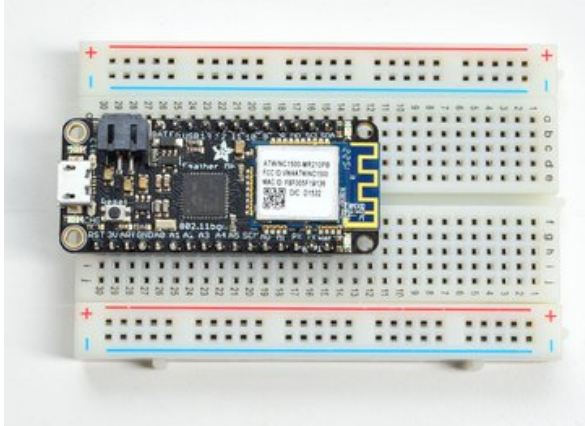
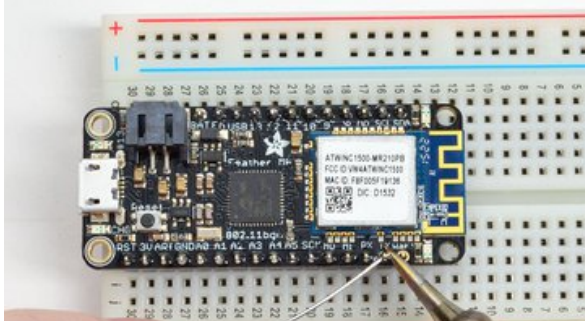
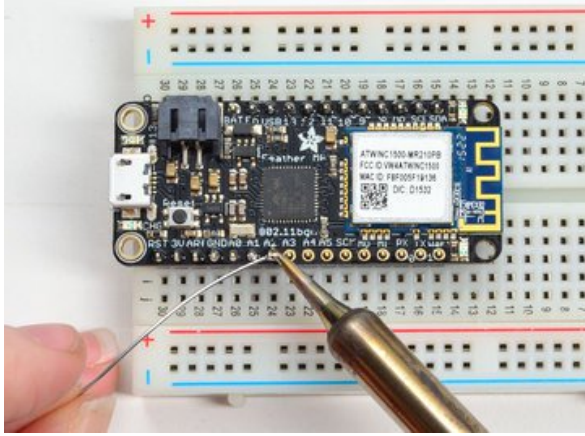
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](http://adafruit.com/guides/adafruit-excellent-soldering) (<http://adafruit.com/guides/adafruit-excellent-soldering>)).





Solder the other strip as well.



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

Soldering on Female Header

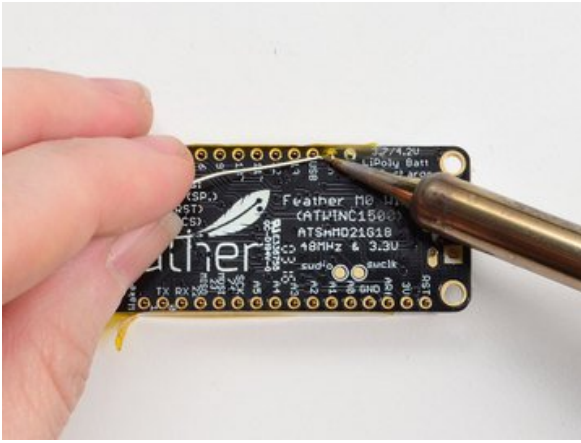


Tape In Place

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

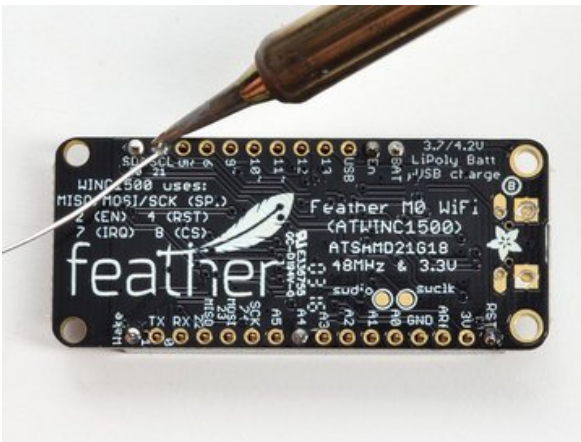
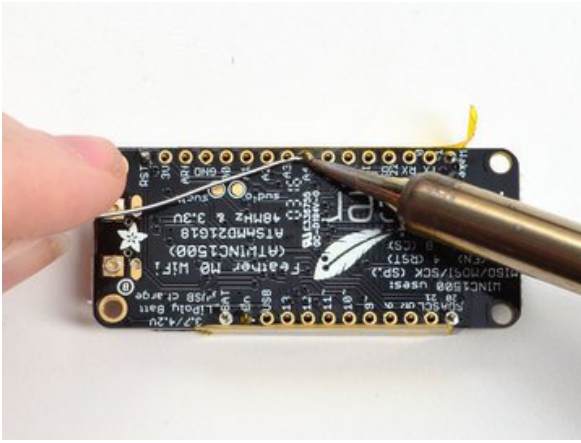
After flipping over, solder one or two points on each strip, to 'tack' the header in place



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](http://adafruit.it/aTk) (<http://adafruit.it/aTk>)).



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

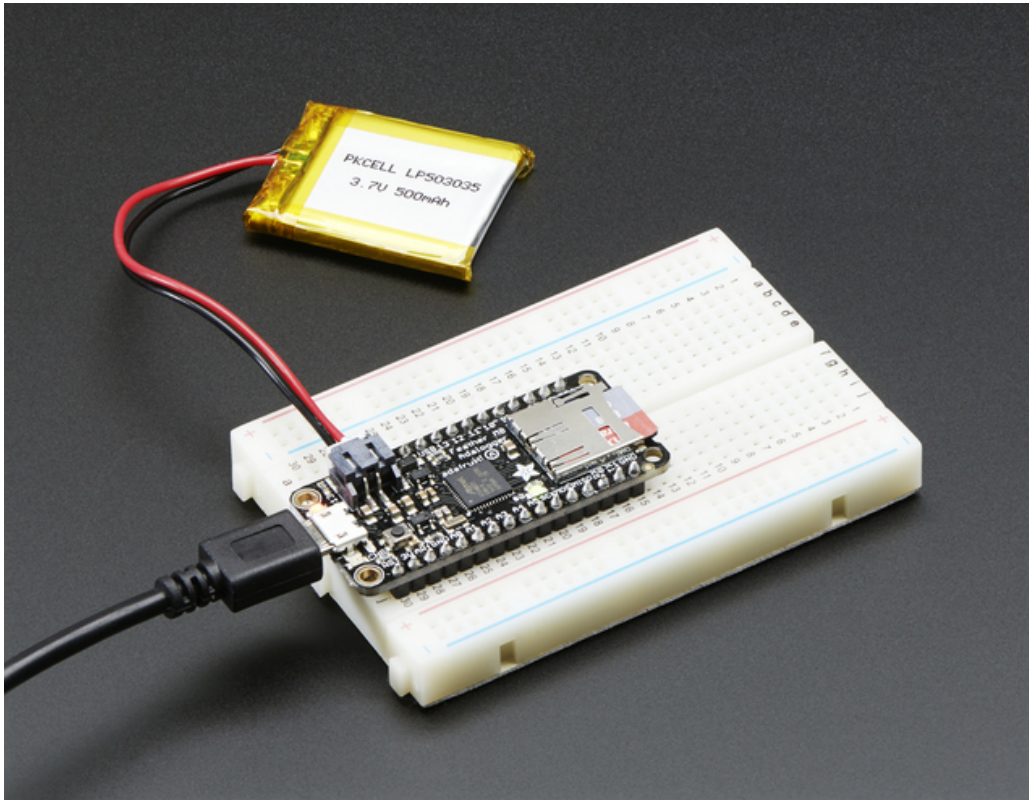


•



•

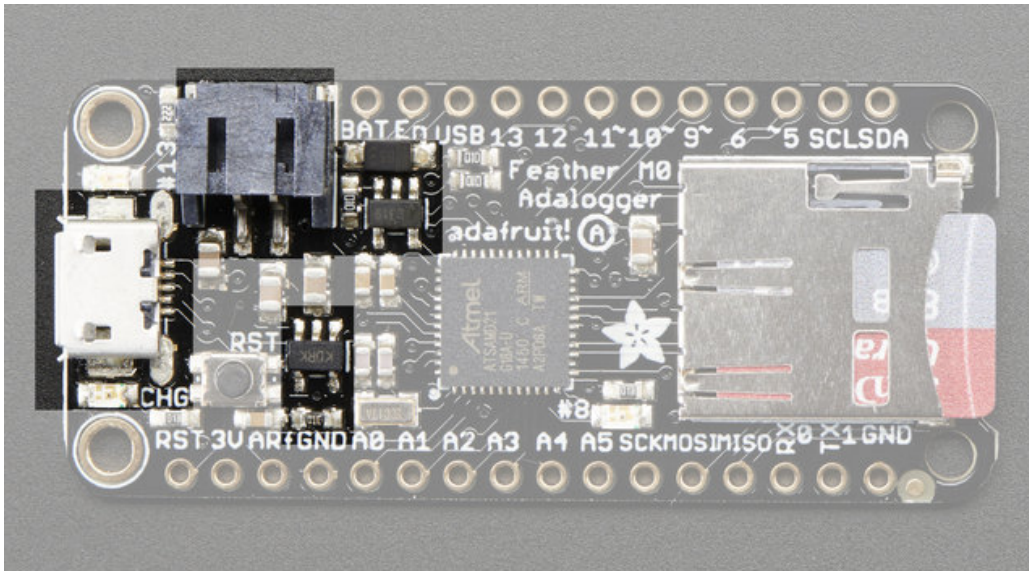
Power Management



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 rechargeable 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.

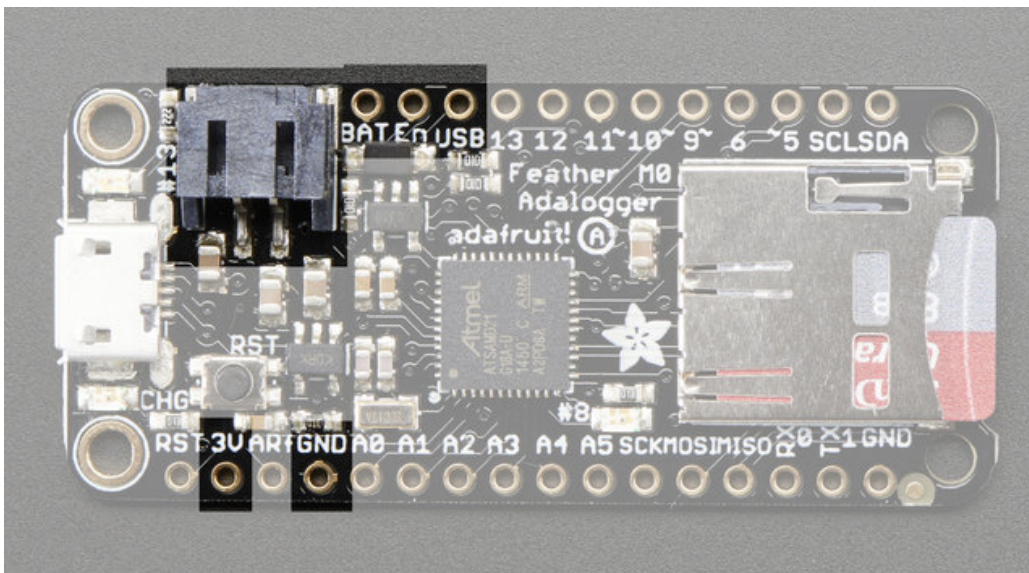
The JST connector polarity is matched to Adafruit LiPoly batteries. Using wrong polarity batteries can destroy your Feather



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.

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 regulator. 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 'spikey' & 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

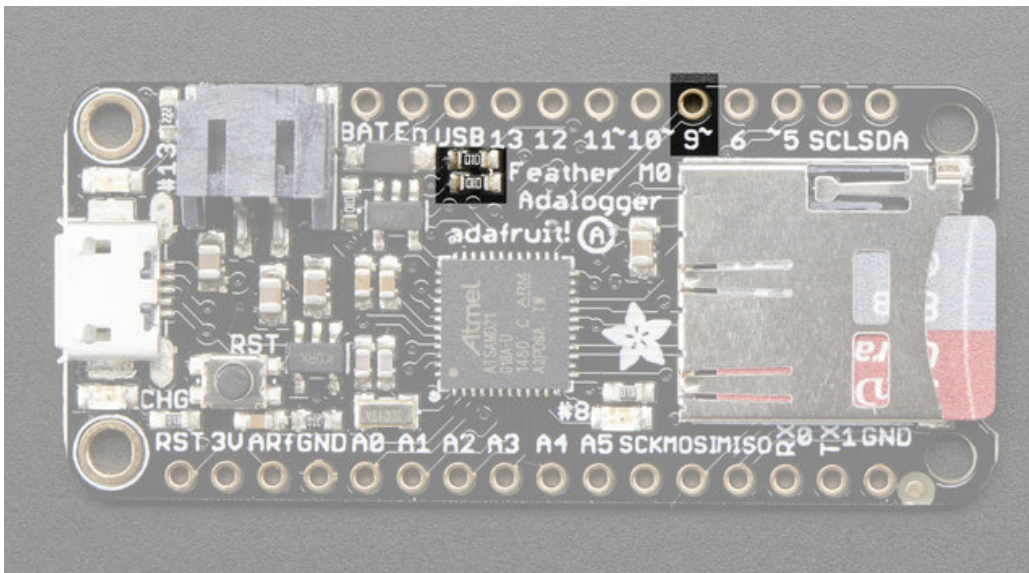
the battery needs recharging. Lipoly batteries are 'maxed out' at 4.2V and stick around 3.7V for much of the battery life, then slowly sink down to 3.2V or so before the protection circuitry cuts it off. By measuring the voltage you can quickly tell when you're heading below 3.7V

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 A7

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)



Average Power Draw w/SD Card

The average power draw of the ATSAM21 + regulator circuitry is 11mA. Both the red and green LED each draw 1mA if you light them up.

Say you are running this sample sketch which logs the analog voltage on A0 to an SD card file once a second.

```
#include <SPI.h>
#include <SD.h>

// Set the pins used
#define cardSelect 4

File logfile;

// blink out an error code
void error(uint8_t errno) {
  while(1) {
    uint8_t i;
```



```

for (i=0; i<errno; i++) {
digitalWrite(13, HIGH);
delay(100);
digitalWrite(13, LOW);
delay(100);
}
for (i=errno; i<10; i++) {
delay(200);
}
}
}

// This line is not needed if you have Adafruit SAMD board package 1.6.2+
// #define Serial SerialUSB

void setup() {
// connect at 115200 so we can read the GPS fast enough and echo without dropping
chars
// also spit it out
Serial.begin(115200);
Serial.println("\r\nAnalog logger test");
pinMode(13, OUTPUT);

// see if the card is present and can be initialized:
if (!SD.begin(cardSelect)) {
Serial.println("Card init. failed!");
error(2);
}
char filename[15];
strcpy(filename, "ANALOG00.TXT");
for (uint8_t i = 0; i < 100; i++) {
filename[6] = '0' + i/10;
filename[7] = '0' + i%10;
// create if does not exist, do not open existing, write, sync after write
if (!SD.exists(filename)) {
break;
}
}

logfile = SD.open(filename, FILE_WRITE);
if( ! logfile ) {
Serial.print("Couldnt create ");
Serial.println(filename);
error(3);
}
Serial.print("Writing to ");
Serial.println(filename);
pinMode(13, OUTPUT);

```



```

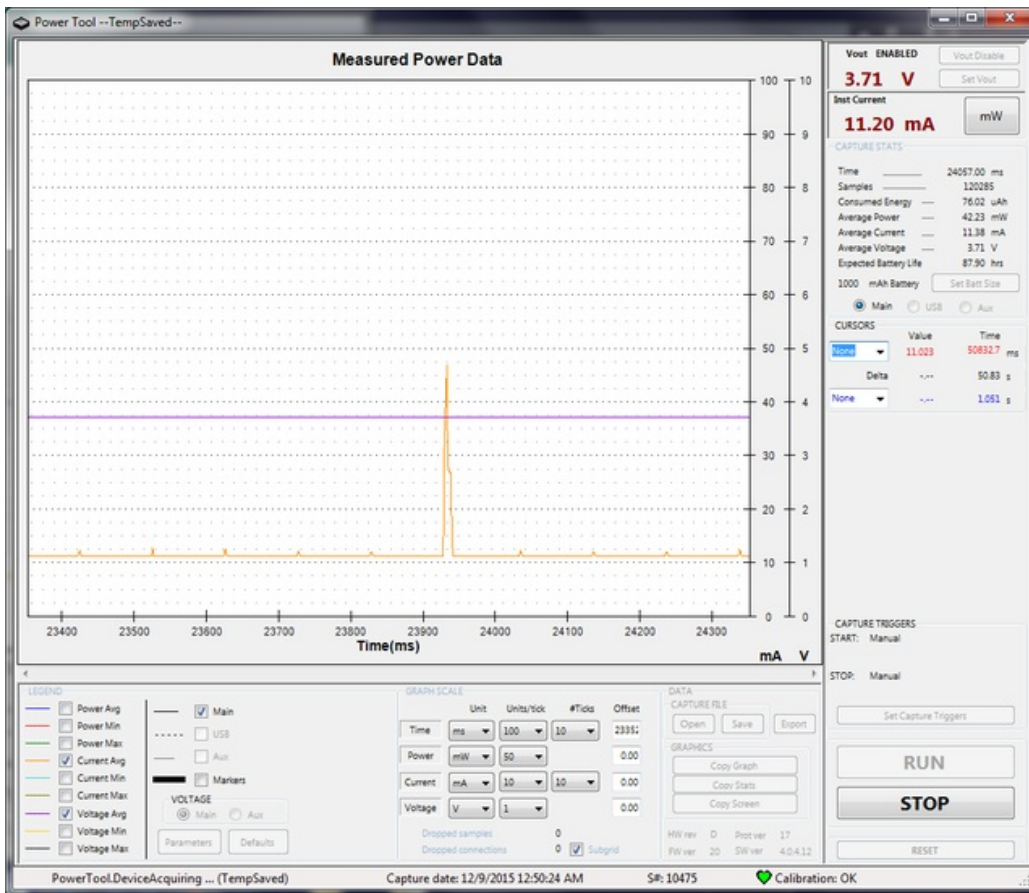
pinMode(8, OUTPUT);
Serial.println("Ready!");
}
uint8_t i=0;
void loop() {
digitalWrite(8, HIGH);
logfile.print("A0 = "); logfile.println(analogRead(0));
Serial.print("A0 = "); Serial.println(analogRead(0));
digitalWrite(8, LOW);
delay(100);
}

```

[adalogger.ino](https://github.com/adafruit/adalogger.ino) hosted with ♥ by [GitHub](https://github.com)

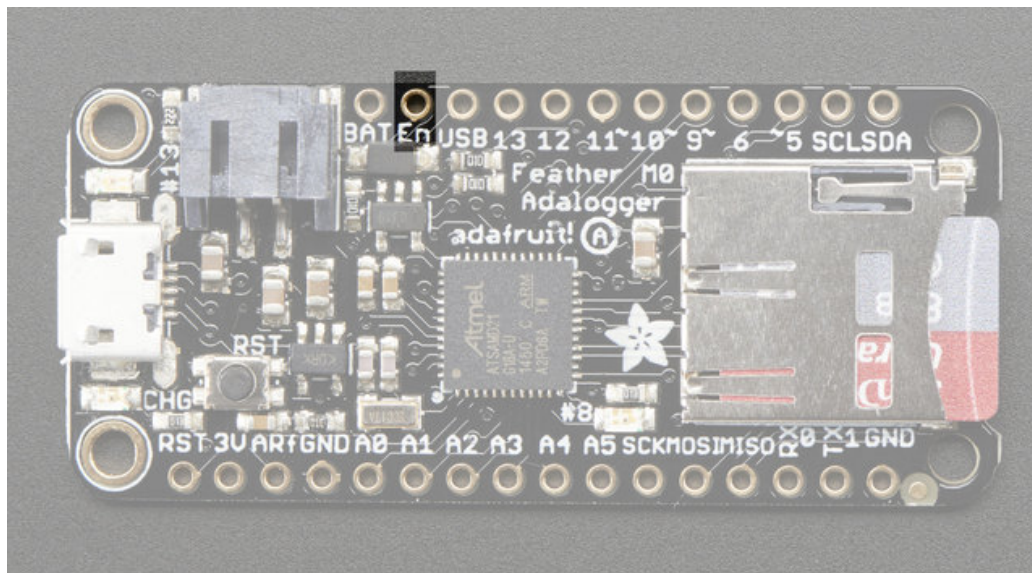
[view raw](#)

You'll draw 11mA or so for 100ms during the delay(100) line. Since we blink the pin #8 LED, we'll also get a 11 mA blip for about 10ms. The data for the SD card is *buffered* which means that whenever we reach 512 bytes of log file that needs to be written, the SD card will actually write the data. When this happens you'll see a 50mA pulse for 10ms. If you use **flush()** to write the log file, this pulse will be much longer, as you have to write the file as well as the file table sectors. Your 50mA spike could end up being 500ms or longer. So basically, keep your file writes to a minimum if you can avoid it!



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

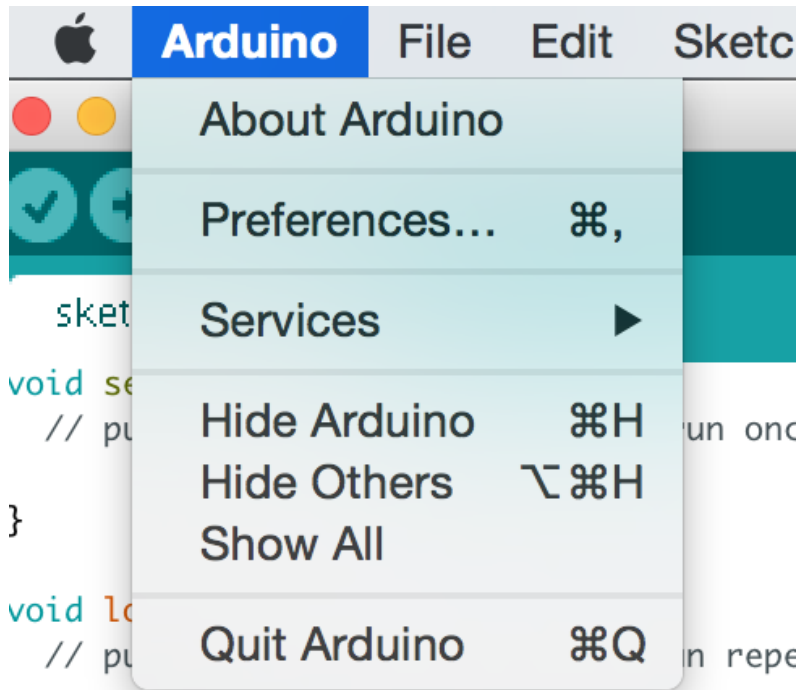


Arduino IDE Setup

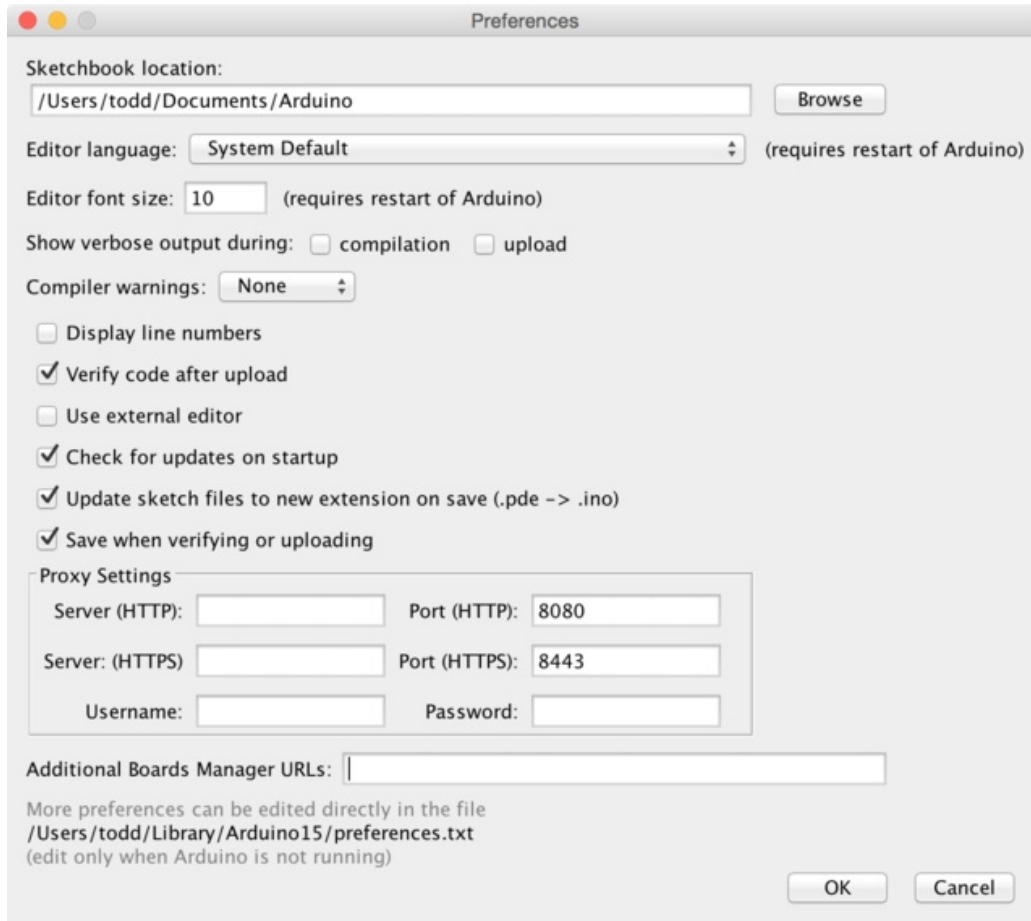
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.6.4** or higher for this guide.

[Arduino IDE v1.6.4+ Download](http://adafru.it/f1P)
<http://adafru.it/f1P>

After you have downloaded and installed **v1.6.4**, 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*.



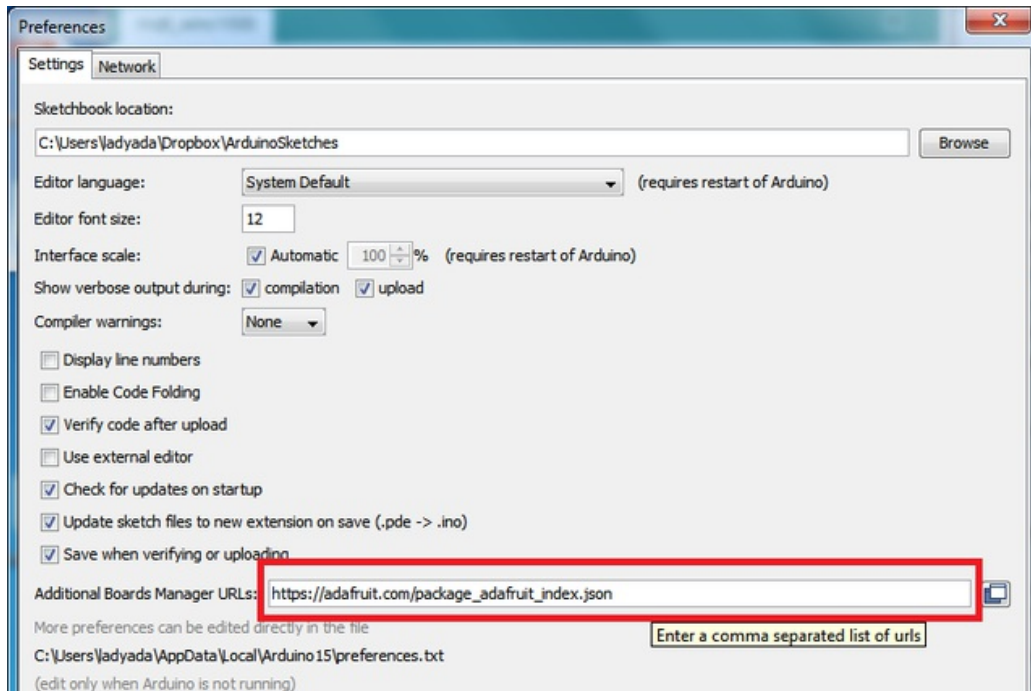
A dialog will pop up just like the one shown below.



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 \(http://adafru.it/f7U\)](http://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
- **Arduino Leonardo & Micro MIDI-USB** - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the [arcore project](http://adafru.it/eSI) (<http://adafru.it/eSI>).

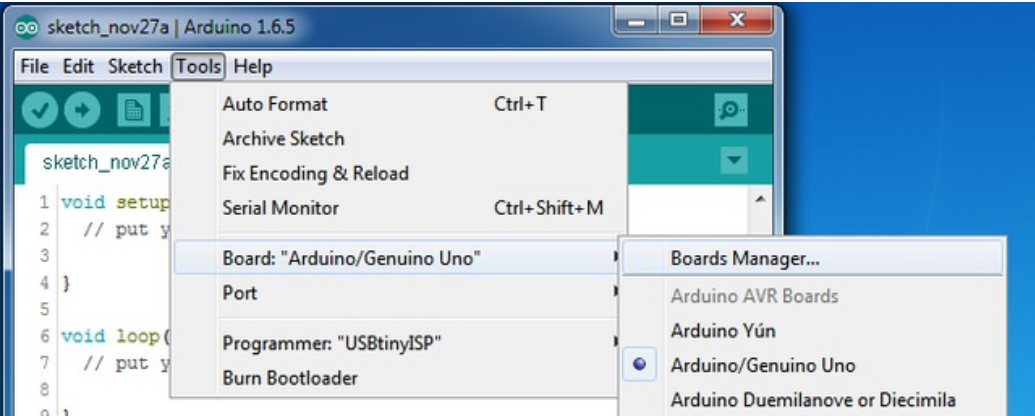
If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

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

Using with Arduino IDE

Since the Feather M0 uses an ATSAM21 chip running at 48 MHz, you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0, especially devices & sensors that use i2c or SPI.

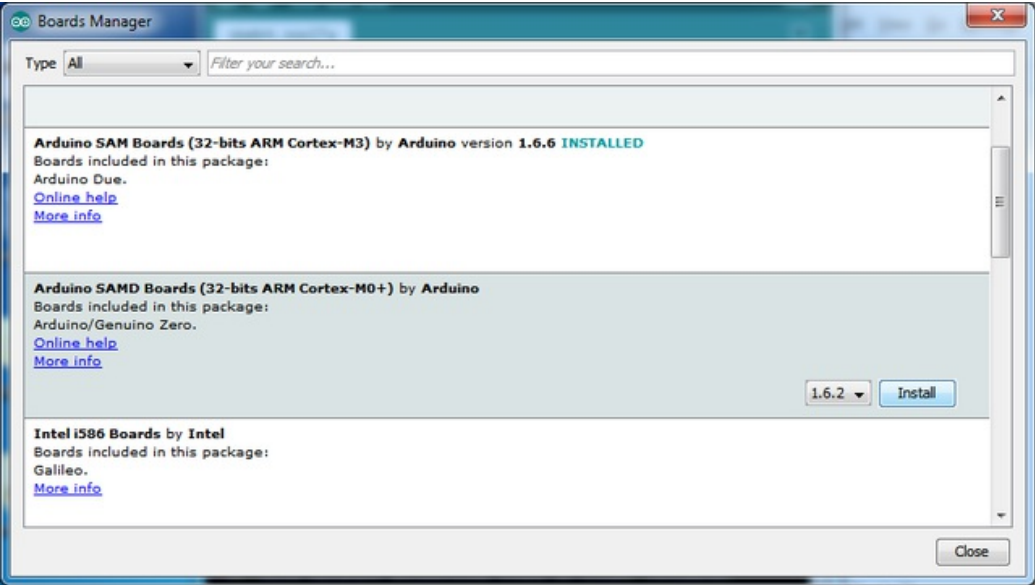
Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the **Boards Manager** by navigating to the **Tools->Board** menu.

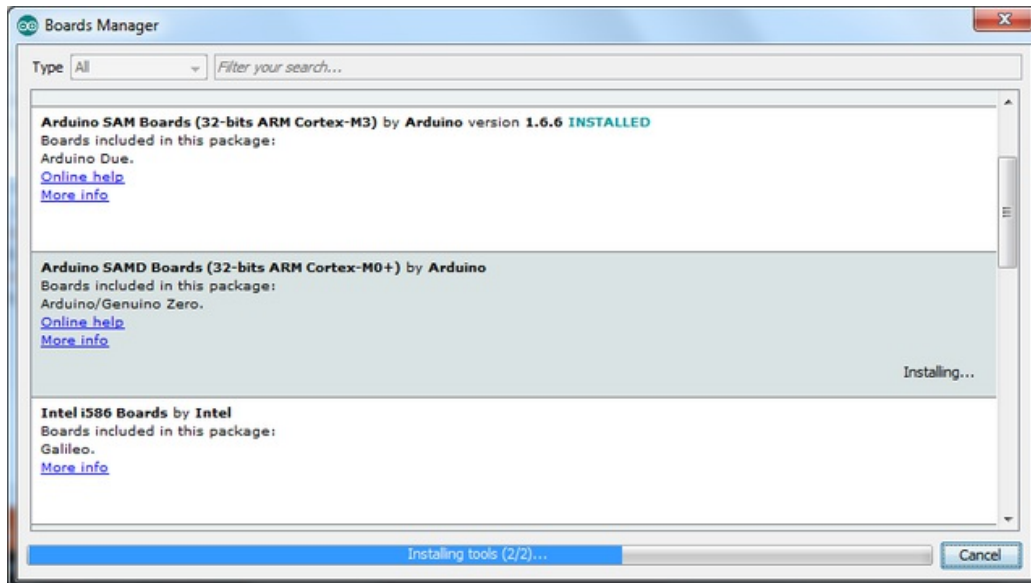


Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Install SAMD Support

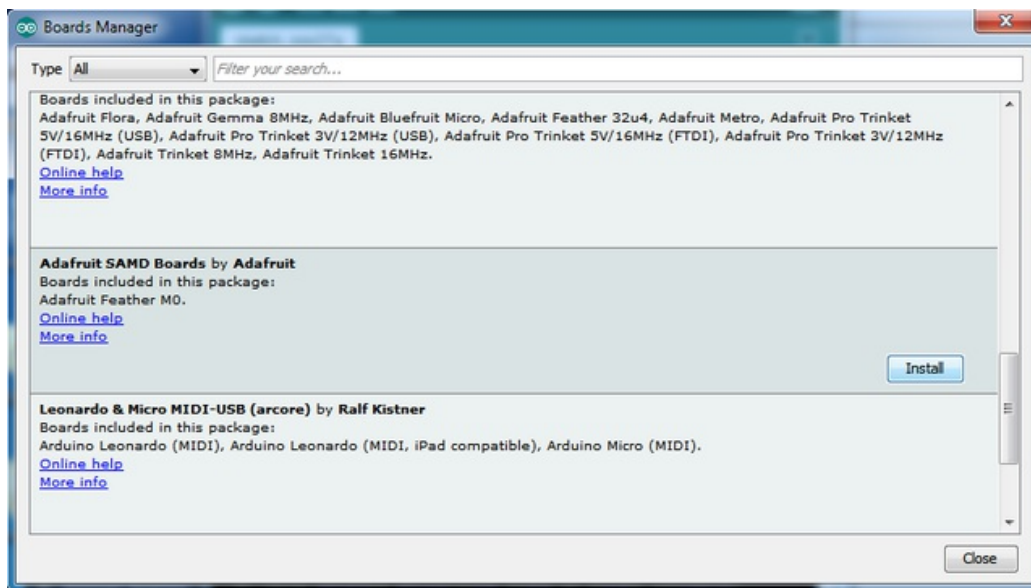
First up, install the **Arduino SAMD Boards** version **1.6.2** or later





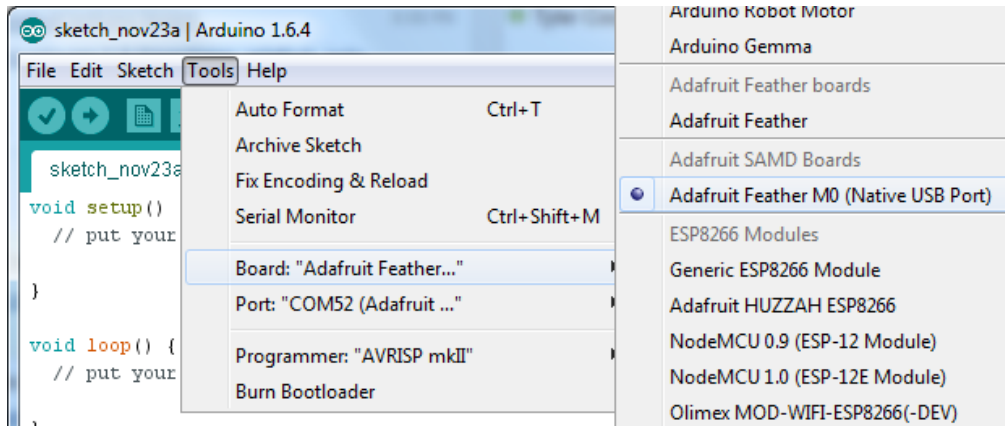
Install Adafruit SAMD

Next you can install the Adafruit SAMD package to add the board file definitions



Even though in theory you don't need to - I recommend rebooting the IDE

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)

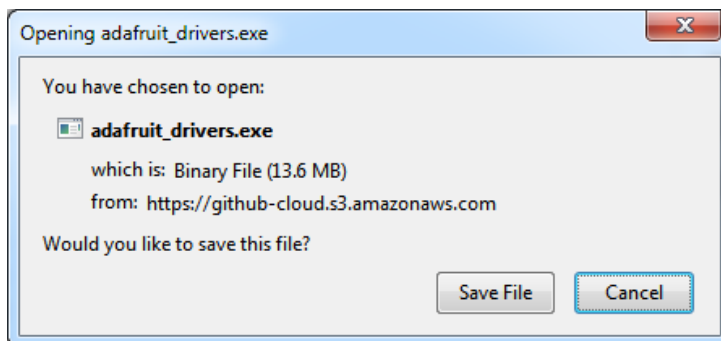
When you plug in the Feather, you'll need to possibly install a driver

Click below to download our Driver Installer

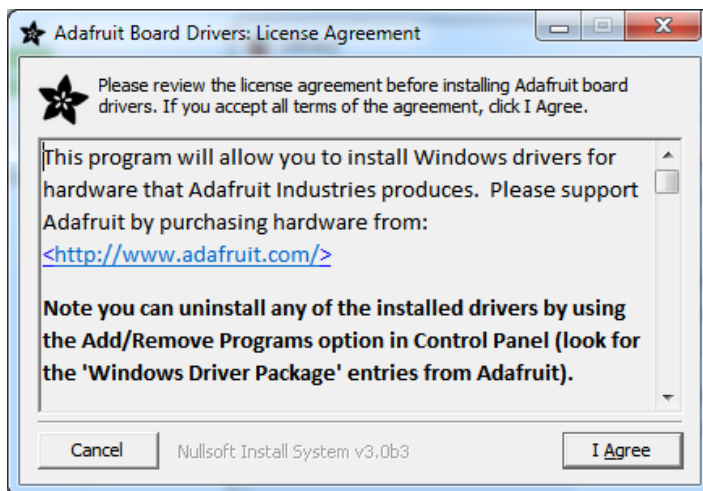
[Download Adafruit Driver Installer](http://adafru.it/mai)

<http://adafru.it/mai>

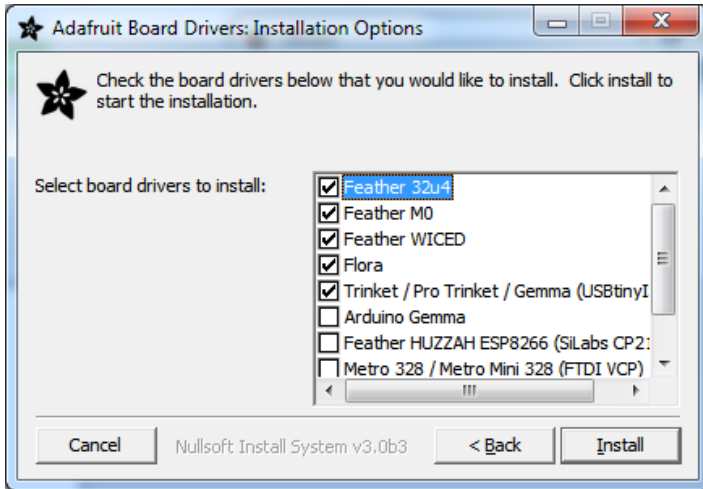
Download and run the installer



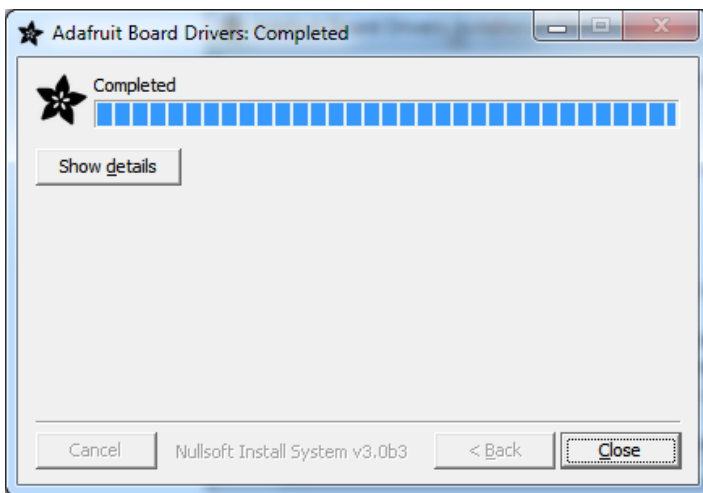
Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license



Select which drivers you want to install:



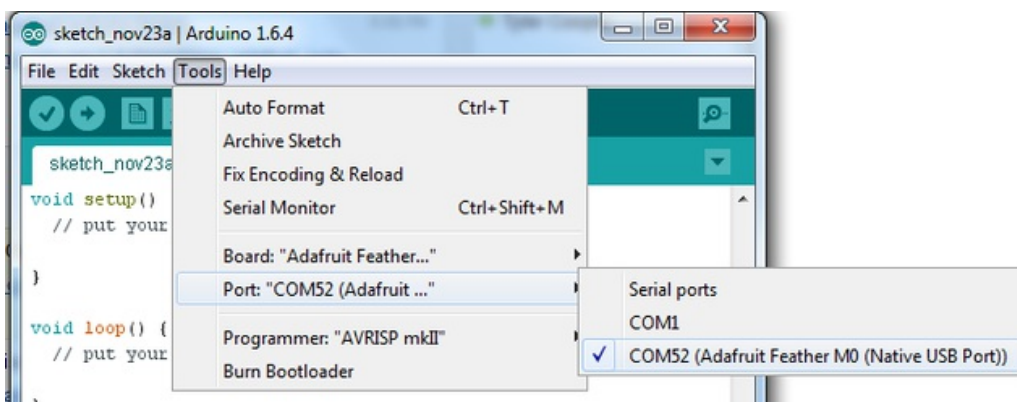
Click **Install** to do the installin'



Blink

Now you can upload your first blink sketch!

Plug in the Feather M0 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 M0!



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.

Successful Upload

If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset

```

Done uploading.
Write 11024 bytes to flash (173 pages)

[=====] 36% (64/173 pages)
[=====] 73% (128/173 pages)
[=====] 100% (173/173 pages)

done in 0.097 seconds

Verify 11024 bytes of flash with checksum.

Verify successful

done in 0.049 seconds

CPU reset.

6
Adafruit Feather M0 (Native USB Port) on COM54

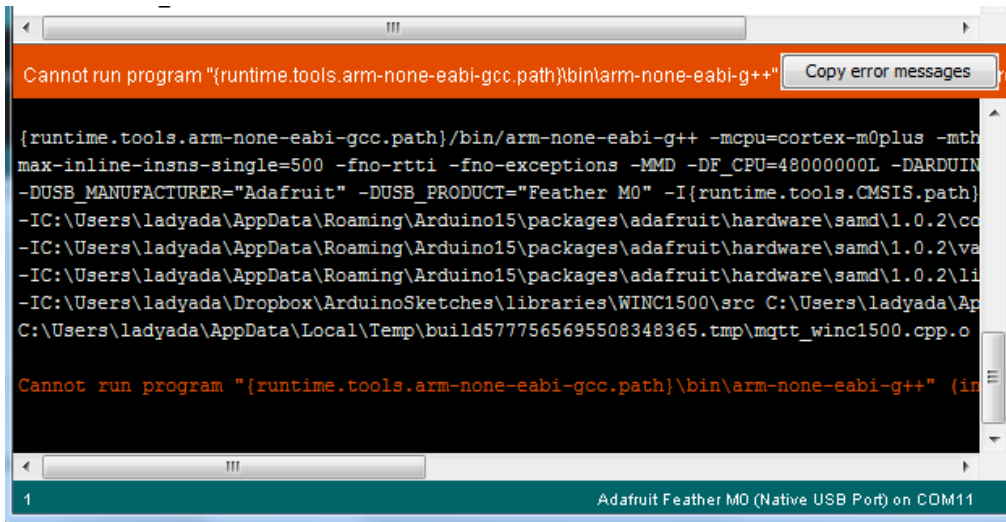
```

Compilation Issues

If you get an alert that looks like

Cannot run program "{runtime.tools.arm-none-eabi-gcc.path}\bin\arm-non-eabi-g++"

Make sure you have installed the **Arduino SAMD** boards package, you need *both* Arduino & Adafruit SAMD board packages

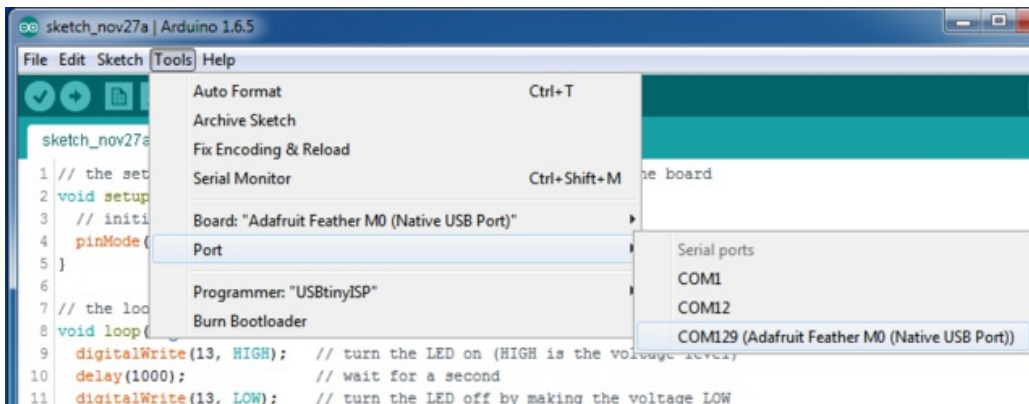


Manually bootloading

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, click the **RST** button **twice** (like a double-click) to get back into the bootloader.

The red LED will pulse, so you know that its in bootloader mode.

Once it is in bootloader mode, you can select the newly created COM/Serial port and re-try uploading.



You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal upload.

Ubuntu & Linux Issue Fix

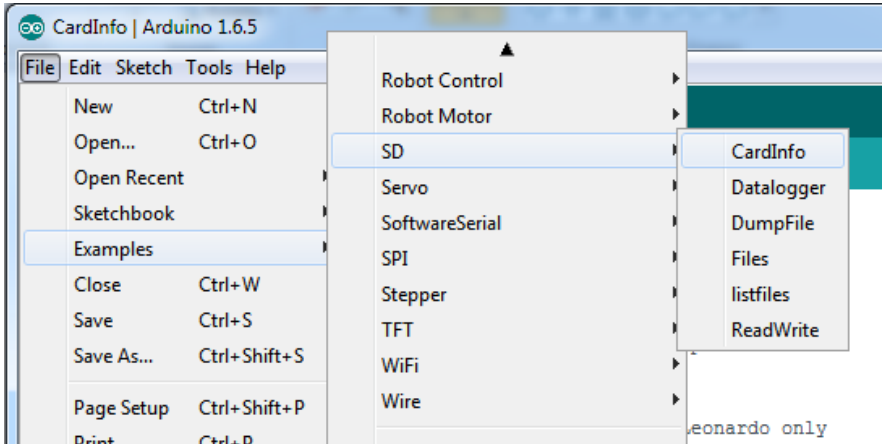
Note if you're using Ubuntu 15.04 (or perhaps other more recent Linux distributions) there is an issue with the modem manager service which causes the Bluefruit LE micro to be difficult to program. If you run into errors like "device or resource busy", "bad file descriptor", or "port is busy" when attempting to program then [you are hitting this issue](http://adafru.it/fP6). (<http://adafru.it/fP6>)

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 Feather board and will fix the programming difficulty issue. [Follow the steps for installing Adafruit's udev rules on this page](http://adafru.it/iOE). (<http://adafru.it/iOE>)

Using the SD Card

Once you have your Feather working, you probably want to rock out with some SD card reading and writing! Luckily, the Arduino IDE has an SD card library that works great, and it even comes with the IDE!

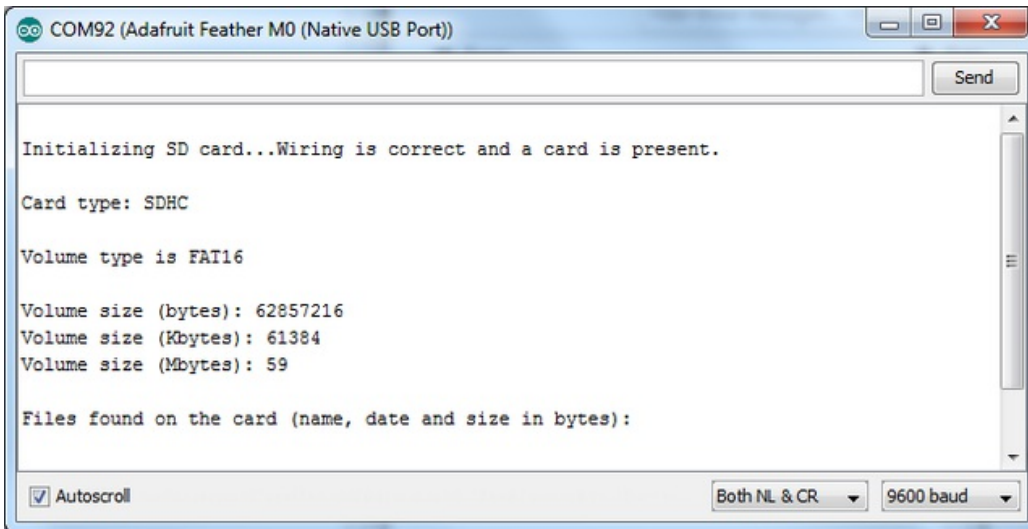
You can start with **CardInfo** which is very detailed



Luckily many of the default examples already have **chipSelect = 4** For other sketches, do check to make sure that CS is set to 4! The SD card uses hardware SPI for the remaining pins.

Make sure you have Adafruit SAMD board package version 1.6.2 or higher, so that Serial debug data goes out on Serial not SerialUSB!

One done, upload & open up the serial console and you'll get all this info including a list of files



Note that it may not print out the files, that's because the example **root.ls(LS_R | LS_DATE | LS_SIZE)** expects to print to Serial rather than SerialUSB.

If you want to list the files, use **listFiles** example

Once you have that working, check out the other examples, such the **Datalogger** example (saving analog data to

SD card) and **Dumpfile** example (reading back data from an SD card)

Example logging sketch

If you want to try saving data to the SD card in the simplest sketch, try this example. You can adjust the `delay()` to set how often analog data is read from pin **A0** and saved to the SD card. The red LED will blink if there's an error, and the green LED will blink when data is written to the SD card.

Note that to save power, we *buffer* the data, so you will only 'save' data truly every 50 datapoints (512 total characters written)

```
#include <SPI.h>
#include <SD.h>

// Set the pins used
#define cardSelect 4

File logfile;

// blink out an error code
void error(uint8_t errno) {
  while(1) {
    uint8_t i;
    for (i=0; i<errno; i++) {
      digitalWrite(13, HIGH);
      delay(100);
      digitalWrite(13, LOW);
      delay(100);
    }
    for (i=errno; i<10; i++) {
      delay(200);
    }
  }
}

// This line is not needed if you have Adafruit SAMD board package 1.6.2+
// #define Serial SerialUSB

void setup() {
  // connect at 115200 so we can read the GPS fast enough and echo without dropping
  // chars
  // also spit it out
  Serial.begin(115200);
  Serial.println("\r\nAnalog logger test!");
  pinMode(13, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(cardSelect)) {
    Serial.println("Card init. failed!");
    error(2);
  }
}
```

```

char filename[15];
strcpy(filename, "ANALOG00.TXT");
for (uint8_t i = 0; i < 100; i++) {
  filename[6] = '0' + i/10;
  filename[7] = '0' + i%10;
  // create if does not exist, do not open existing, write, sync after write
  if (!SD.exists(filename)) {
    break;
  }
}

logfile = SD.open(filename, FILE_WRITE);
if( ! logfile ) {
  Serial.print("Couldnt create ");
  Serial.println(filename);
  error(3);
}
Serial.print("Writing to ");
Serial.println(filename);
pinMode(13, OUTPUT);
pinMode(8, OUTPUT);
Serial.println("Ready!");
}

uint8_t i=0;
void loop() {
  digitalWrite(8, HIGH);
  logfile.print("A0 = "); logfile.println(analogRead(0));
  Serial.print("A0 = "); Serial.println(analogRead(0));
  digitalWrite(8, LOW);
  delay(100);
}

```

[adalogger.ino](#) hosted with ♥ by [GitHub](#)

[view raw](#)

If you really want to make sure you save every data point, put a

```
logfile.flush();
```

right after the **logfile.print**'s however this will cause the adalogger to draw a lot more power, maybe about 3x as much on average (30mA avg rather than about 10mA)

Adapting Sketches to M0

The ATSAM21 is a very nice little chip but its fairly new as Arduino-compatible cores go **Most** sketches & libraries will work but here's a few things we noticed!

Analog References

If you'd like to use the **ARef** pin for a non-3.3V analog reference, the code to use `isAnalogReference(AR_EXTERNAL)` (it's `AR_EXTERNAL` not `EXTERNAL`)

Pin Outputs & Pullups

The old-style way of turning on a pin as an input with a pullup is to use

```
pinMode(pin, INPUT)
digitalWrite(pin, HIGH)
```

This is because the pullup-selection register is the same as the output-selection register.

For the M0, you can't do this anymore! Instead, use

```
pinMode(pin, INPUT_PULLUP)
```

which has the benefit of being backwards compatible with AVR.

Serial vs SerialUSB

99.9% of your existing Arduino sketches use **Serial.print** to debug and give output. For the Official Arduino SAMD/M0 core, this goes to the Serial5 port, which isn't exposed on the Feather. The USB port for the Official Arduino M0 core, is called **SerialUSB** instead.

In the Adafruit M0 Core, we fixed it so that Serial goes to USB when you use a Feather M0 so it will automatically work just fine.

However, on the off chance you are using the official Arduino SAMD core & you want your Serial prints and reads to use the USB port, use **SerialUSB** instead of Serial in your sketch

If you have existing sketches and code and you want them to work with the M0 without a huge find-replace, put

```
#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
// Required for Serial on Zero based boards
#define Serial SERIAL_PORT_USBVIRTUAL
#endif
```

right above the first function definition in your code. For example:



AnalogWrite / PWM

We've noticed that some PWM outputs are not working with the current SAMD core, its something that is being worked on!

Missing header files

there might be code that uses libraries that are not supported by the M0 core. For example if you have a line with

```
#include <util/delay.h>
```

you'll get an error that says

```
fatal error: util/delay.h: No such file or directory
```

```
#include <util/delay.h>
```

^

```
compilation terminated.
```

```
Error compiling.
```

In which case you can simply locate where the line is (the error will give you the file name and line number) and 'wrap it' with `#ifdef`'s so it looks like:

```
#if !defined(ARDUINO_ARCH_SAM) && !defined(ARDUINO_ARCH_SAMD) && !defined(ESP8266) && !defined(ARDUINO_ARCH_STM32F2)
#include <util/delay.h>
#endif
```

The above will also make sure that header file isn't included for other architectures

If the `#include` is in the arduino sketch itself, you can try just removing the line.

Bootloader Launching

For most other AVR's, clicking **reset** while plugged into USB will launch the bootloader manually, the bootloader will time out after a few seconds. For the M0, you'll need to *double click* the button. You will see a pulsing red LED to let you know you're in bootloader mode. Once in that mode, it wont time out! Click reset again if you want to go back to launching code

Aligned Memory Access

This is a little less likely to happen to you but it happened to me! If you're used to 8-bit platforms, you can do this nice thing where you can typecast variables around. e.g.

```
uint8_t mybuffer[4];
float f = (float)mybuffer;
```

You can't be guaranteed that this will work on a 32-bit platform because **mybuffer** might not be aligned to a 2 or 4-byte boundary. The ARM Cortex-M0 can only directly access data on 16-bit boundaries (every 2 or 4 bytes). Trying to access an odd-boundary byte (on a 1 or 3 byte location) will cause a Hard Fault and stop the MCU. Thankfully, there's an easy work around ... just use `memcpy`!

```
uint8_t mybuffer[4];
float f;
memcpy(f, mybuffer, 4)
```

Floating Point Conversion

Like the AVR Arduinos, the M0 library does not have full support for converting floating point numbers to ASCII strings. Functions like `sprintf` will not convert floating point. Fortunately, the standard AVR-LIBC library includes the `dtostrf` function which can handle the conversion for you.

Unfortunately, the M0 run-time library does not have `dtostrf`. You may see some references to using `#include <avr/dtostrf.h>` to get `dtostrf` in your code. And while it will compile, it does **not** work.

Instead, check out this thread to find a working `dtostrf` function you can include in your code:

<http://forum.arduino.cc/index.php?topic=368720.0> (<http://adafru.it/IFS>)

How Much RAM Available?

The ATSAM D21G18 has 32K of RAM, but you still might need to track it for some reason. You can do so with this handy function:

```
extern "C" char *sbrk(int i);

int FreeRam () {
  char stack_dummy = 0;
  return &stack_dummy - sbrk(0);
}
```

Thx to <http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879> (<http://adafru.it/m6D>) for the tip!

Storing data in FLASH

If you're used to AVR, you've probably used **PROGMEM** to let the compiler know you'd like to put a variable or string in flash memory to save on RAM. On the ARM, it's a little easier, simply add **const** before the variable name:

```
const char str[] = "My very long string";
```

That string is now in FLASH. You can manipulate the string just like RAM data, the compiler will automatically read from FLASH so you don't need special progmem-knowledgeable functions.

You can verify where data is stored by printing out the address:

```
Serial.print("Address of str $"); Serial.println((int)&str, HEX);
```

If the address is \$2000000 or larger, its in SRAM. If the address is between \$0000 and \$3FFFF Then it is in FLASH

Downloads

Datasheets

- [ATSAMD21 Datasheet \(http://adafru.it/kUf\)](http://adafru.it/kUf) (the main chip on the Feather M0)

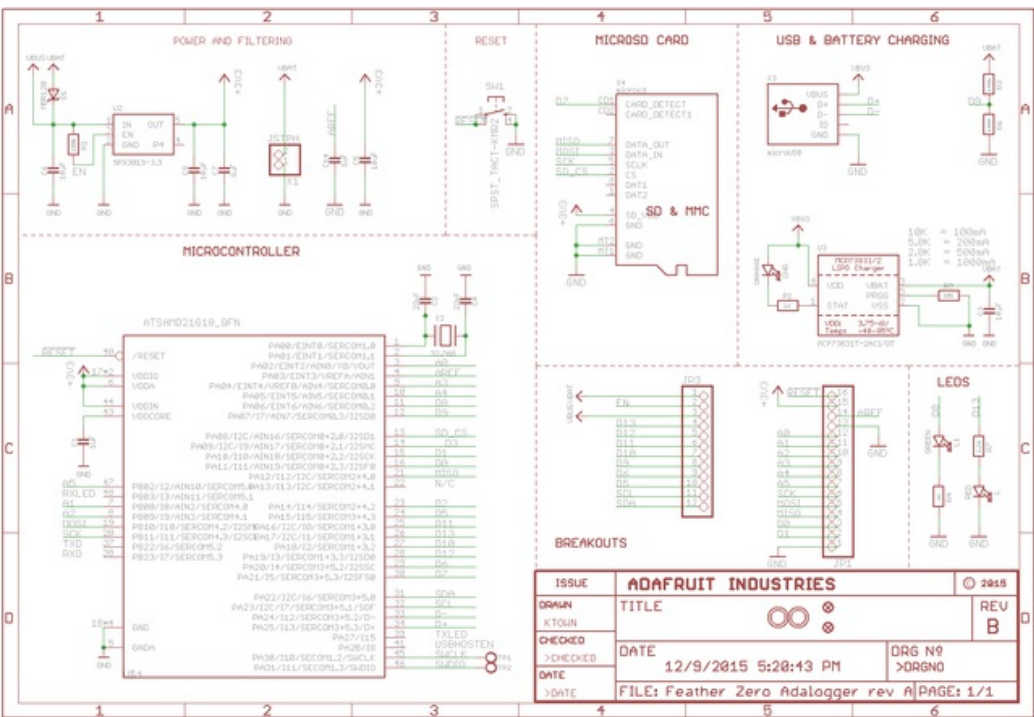
[Feather M0 Adalogger Pinout Diagram](http://adafru.it/IMb)
<http://adafru.it/IMb>

Pinout note: The RX and TX pins are known as **Serial1**

- [PCB Files on GitHub \(http://adafru.it/obn\)](http://adafru.it/obn)
- [Fritzing object in the Adafruit Fritzing Library \(http://adafru.it/aP3\)](http://adafru.it/aP3)

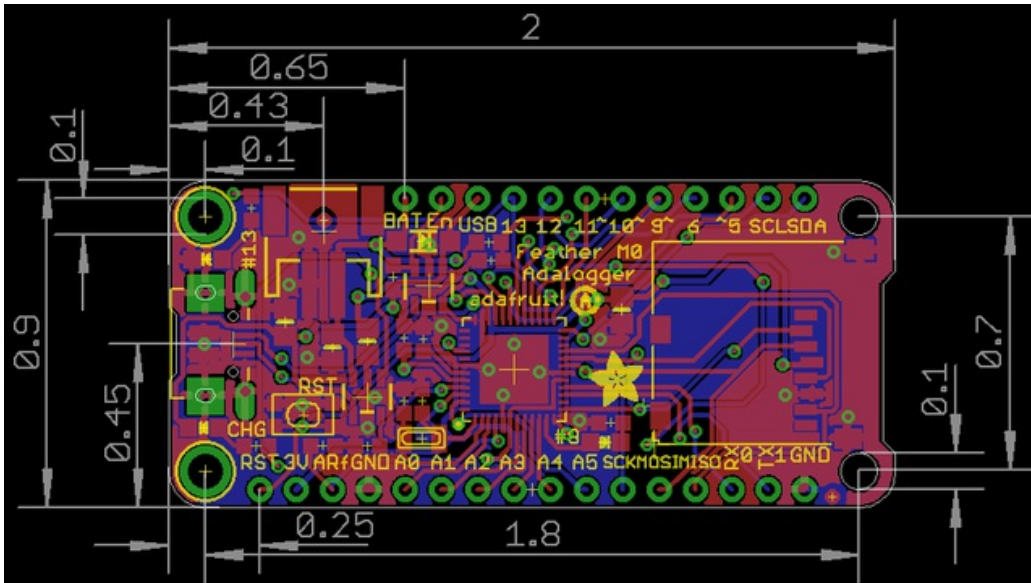
Schematic

Click to enlarge



Fabrication Print

Dimensions in inches



HELP!

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

A vast number of Feather 'failures' are due to charge-only USB cables

Seriously, we get upwards of 5 complaints a day that turn out to be due to charge-only cables

Use only a cable that you **know** is for data syncing

If you have any charge-only cables, cut them in half throw them out. We are serious! They tend to be low quality in general, and will only confuse you and others later, just get a good data+charge USB cable

My Feather M0 won't enumerate anymore and can't be programmed, help!

If something happens and your Feather M0 won't enumerate as a USB serial device, like perhaps you were exploring the ATSAM21's peripherals and accidentally misconfigured something, don't worry you can try a few things to revive it back to normal.

- First try pressing the reset button twice like a mouse double click. The LED should start pulsing red to inform you the chip has entered its bootloader and is waiting a program upload.
- The Feather M0 will show up as a new COM port device (if you haven't already installed the Windows drivers, do that!)
- Select the new bootloader COM port in the Arduino IDE
- Now try uploading a simple blink example from the Arduino IDE to see if that gets the board back into a good state.

If you don't get a pulsing red LED and the board doesn't enumerate as a serial device then something has happened to the bootloader. Unfortunately the best option in this case is to connect to the single-wire debug test points on the back of the board (the SWDIO & SWCLK pads) and manually reprogram the bootloader using a J-Link or ST-Link ARM programmer.

I'm having problems with COM ports and my M0

Theres **two** COM ports you can have with the M0, one is the **user port** and one is the **bootloader port**. They are not the same COM port number!

When you upload a new user program it will come up with a user com port, particularly if you use Serial in your user program.

If you crash your user program, or have a program that halts or otherwise fails, the user com port can disappear.

When the user COM port disappears, Arduino will not be able to automatically start the bootloader and upload new software.

So you will need to help it by performing the double-click to start the bootloader, then selecting the bootloader COM port for uploading. See above

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

UNO-type Arduinos have a *seperate* serial port chip (aka "FTDI chip" or "Prolific PL2303" etc etc) which handles all serial port capability seperately than the main chip. This way if the main chip fails, you can always use the COM port.

M0 and 32u4-based Arduinos do not have a separate chip, instead the main processor performs this task for you. It allows for a lower cost, higher power setup...but requires a little more effort since you will need to 'kick' into the bootloader manually once in a while