CanSat Feather M0 Guide

A comprehensive technical guide to assemble, use and get the best of the Feather M0 Express for CanSat launch.

(version 0.3)

CANSAT Wiki - welcome

Sommaire

What is CanSat?

CanSat Contest simulates the fly of miniaturized satellites named CanSat (Can Satellite).

The CanSat is an autonomous devices enclosed within the volume of a soda can.

The volume of a CanSat has the following characteristics: 66mm diameter, 115mm height for a mass of 350gr.

As the CanSat have small volume and are very affordable, the CanSat contest is great for learning more about space technologies.

The CanSats are deployed from a rocket (the launch vehicle) at a height of about 3000m depending on the competition (see all the details in the contest rules).

The CanSats are not orbited and are always deployed in the athmosphere. So they always comes back to the earth.

The CanSat volume cannot increase until the CanSat is deployed out of the rocket.

This means that external antenna is allowed only after the CanSat left the rocket.

A parachute (that increase the volume of the CanSat) is usually used to limit damages. The aim is to reuse the CanSat for several missions.

In Europe, the CanSat contest includes two missions:

- Mission 1: mesure pressure and temperature and transmit data in real time.
- Mission 2: free choice mission (using Intertial Measurement Unit, GPS, MPX differential pressure sensor, etc).

How to subscribe the contest?

In Europe, the CanSat competition is promoted by the ESA (source https://en.wikipedia.org/wiki/CanSat#_Europe).

- **For Belgium**: the CanSat Belgium competition is promoted by **InnovIris** (CanSat Belgium NL http://www.innoviris.be/fr/promotion/cansat-belgium , CanSat Belgium FR http://www.innoviris.be/nl/promotie/cansat-belgium , CANSAT Belgium FaceBook https://www.facebook.com/CanSat.Belgium?fref=ts).
  Innoviris is the Brussels Institute for the encouragement of scientific research and innovation.

- **For Luxembourg**: the CanSat Luxembourg competition is promoted by **Esero Luxembourg** (www.cansat.lu) https://www.cansat.lu .

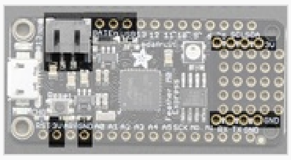
About this Wiki

MC Hobby does promote, mainly in French, the Arduino Open-Source plateform, MicroPython, Raspberry-Pi, coding, electronics to made knowledge freely available de the mass.

This CanSat Belgium Wiki is one of the MC Hobby https://shop.mchobby.be documentation project https://wiki.mchobby.be partially funded by Innoviris.

Getting Started

Hardware discovery



Discover the various items included within the kit.

Cliquez ici

Arduino IDE

Prepare your Arduino IDE environment

Cliquez ici

Feather User Guide

The **Feather M0 Express** user guide for Arduino IDE.

Cliquez ici

Forcing Flash Mode

Useful tip to know.

Cliquez ici

M0 Sketch tips

Tips and tricks to write sketch for the M0.

Cliquez ici

Using SPI Flash

Programing advice to work with integrated SPI Flash.

Cliquez ici

The Feather M0 Express can also been used used with **CircuitPython**, a Flavor of MicroPython. It's means that you can also write **Python script on this microcontroler**. This point is not covered in this tutorial series.

You may learn more from Adafruit Industries https://learn.adafruit.com/adafruit-feather-m0-express-designed-for-circuit-python-circuitpython/what-is-circuitpython (or this translation).

Test the devices

BMP280 sensor

Test the BMP280 pressure and elevation sensor.

TMP36 sensor

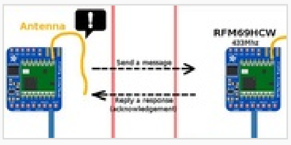
Test the TMP36 analog temperature sensor

Cliquez ici

RFM69HCW radio

User guide for the **RFM69HCW** radio module.

RFM69HCW Testing



Testing the communication with **RFM69HCW** and sending data through the

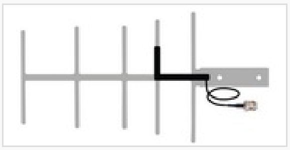
Cliquez ici

Cliquez ici

radio module.

Cliquez ici

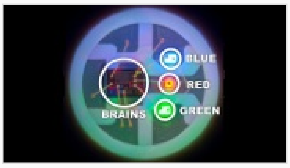
Radio Antenna



A well designed Antenna can increase the communication distance.

Cliquez ici

NeoPixel



Using the NeoPixel LED available on the board.

Cliquez ici

Mission 1

The team must build a CanSat and program it to accomplish the primary (madatory) mission, as follows:

After release and during descent,

- the CanSat shall measure several parameters,
- the data shoudld be transmitted as telemetry information to the ground station.
- the telemetry should, at least, be send once every second.

The following informations should be captured:

- Air temperature
- Air pressure

It must be possible for the team to analyse the data obtained (for example, make a calculation of altitude) and display it in graphs (for example, altitude vs. time and temperature vs. altitude).

So, don't forget to also capture the time for each data collected. This information is as critical than pressure and temperature

Frequency Plan

Be courteous, share the radio bandwidth.

Cliquez ici

Mission 1: Emitter

Wiring sensors, capturing datas and sending over radio.

Cliquez ici

Mission 1: Receiver

Receiving the transmitted data.

Cliquez ici

Mission 1: Going autonomous

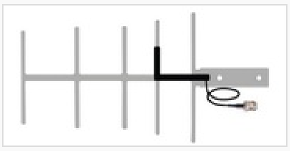
Receiving the transmitted data.

Cliquez ici

Resources

CanSat 3D

Radio Antenna



Parachute

Shopping

CanSat 3D models to print your own one

Cliquez ici

A well designed Antenna can increase the communication distance.

Cliquez ici

Some reference to design the parachute

Cliquez ici

Need to refill some parts?

Cliquez ici

Optional: Adding a GPS module to transmit the CanSat position with the telemetry data would ease to positioning when back on the earth.

Other resources:

- CanSat Europe https://en.wikipedia.org/wiki/CanSat#_Europe on WikiPedia - Lot of informations
- This wiki as PDF https://df.mchobby.be/wiki-export/Eng-Cansat/Eng-Cansat.pdf (*pdf*)
- Install & configure **ARDUINO IDE** for Feather M0 Express https://df.mchobby.be/wiki-export/Eng-Cansat/ENG-CANSAT-ARDUINO-IDE.pdf (*pdf*)
- CanSat Kit Presentation (Belgium) https://df.mchobby.be/wiki-export/Eng-Cansat/Cansat-kit-presentation.pdf (*pdf*)
- CanSat Kit Presentation (Luxembourg) https://df.mchobby.be/wiki-export/Eng-Cansat/Cansat-kit-presentation-Lux.pdf (*pdf*)
- Cansat 2022 Luxembourg - Kit Report https://df.mchobby.be/wiki-export/Eng-Cansat/Esero_SnT_McHobby_CanSat.pdf (*pdf*)

Hardware Discovery

Sommaire

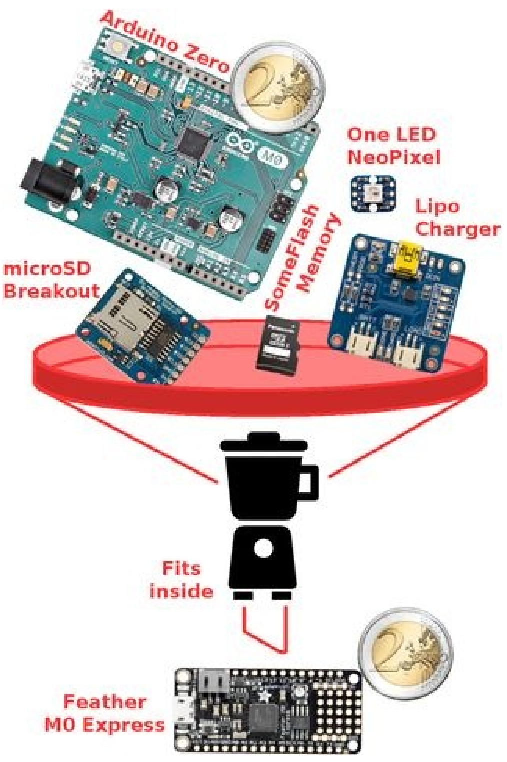
Feather M0 Express in few words

The CanSat kit it build around the Adafruit Industries **Feather M0 Express** plateform.

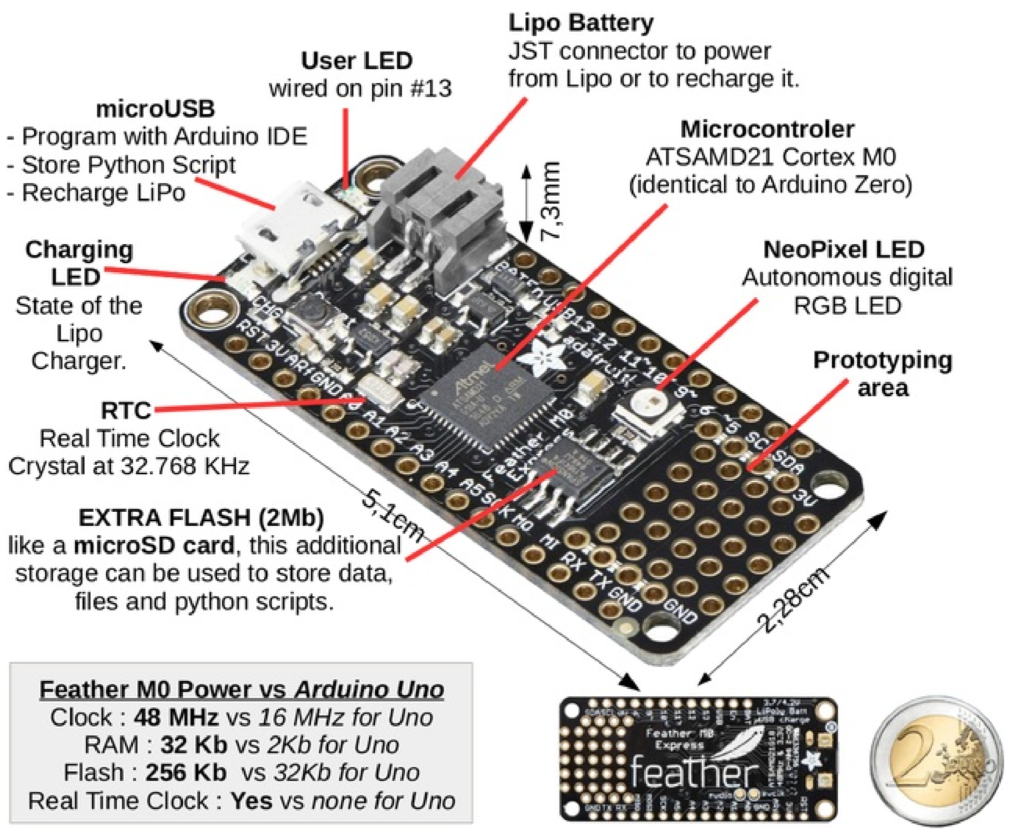
- **Feather** is a new emerging standard -Arduino compatible plaform- for embedded projet.
- **Feather** is small, light and already brings lot useful features.
- **Feather M0** is compatible with Arduino M0, so compatible with Arduino IDE.
- **Feather M0 Express** also embed FLASH memory that can act like SD Card (Arduino IDE) or USB Stick (CircuitPython)

As Feather M0 is compatible Arduino IDE, everything learned for Arduino Uno can be applied to Feather M0. Just care about voltage, the UNO is 5V Logic and the **Feather 3.3V logic**.

Feather Board content



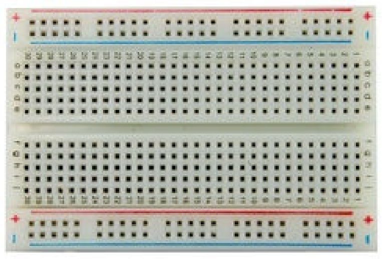
Feather board items

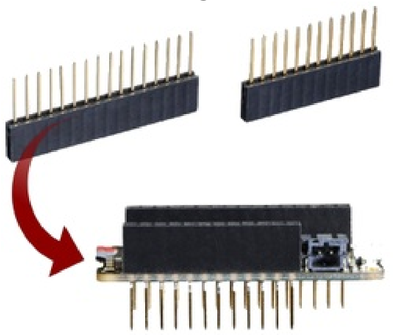
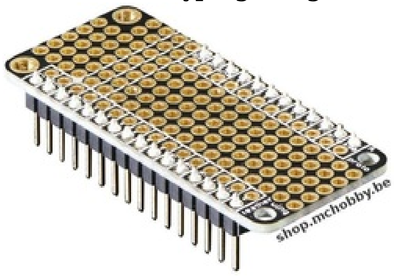
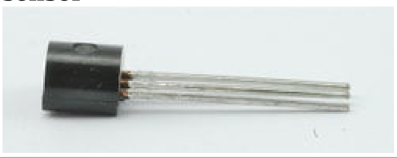
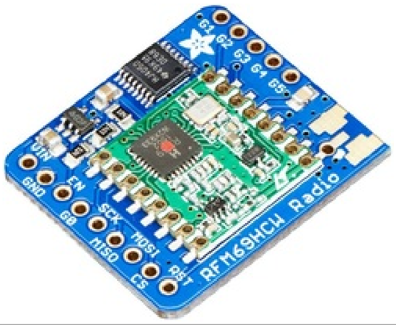

Lipo Battery
JST connector to power from Lipo or to recharge it.
User LED
wired on pin #13
microUSB
- Program with Arduino IDE
- Store Python Script
- Recharge LiPo
Microcontroler
ATSAMD21 Cortex M0
(identical to Arduino Zero)
7,3mm
Charging LED
State of the Lipo Charger.
NeoPixel LED
Autonomous digital RGB LED
Prototyping area
RTC
Real Time Clock
Crystal at 32.768 KHz
EXTRA FLASH (2Mb)
like a microSD card, this additional storage can be used to store data, files and python scripts.
5,1cm
2,28cm

Feather M0 Power vs *Arduino Uno*
Clock : **48 MHz** vs *16 MHz for Uno*
RAM : **32 Kb** vs *2Kb for Uno*
Flash : **256 Kb** vs *32Kb for Uno*
Real Time Clock : **Yes** vs *none for Uno*

See the Feather User Guide section for more information.

Kit content

	Description	Quantité
Feather M0 Express	New Arduino M0 compatible on a standard platform for embedded project. Also compatible with CircuitPython. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=1119	1
USB A/microB 1m cable	Can be used to plug your feather on a computer to program it or to reload the Lipo. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=145	1
Half Size Breadboard 	Solderless breadboard are used for fast prototyping. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=53	1
Multi-functional breadboard wires	Set of wires with plug that can be modified from female to male. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=82	1

Feather Stacking Headers 	Plug your feather or prototype wing on breadboard and still having a female connector under the hand. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=832	1
Feather Prototyping Wing 	Prototyping board for feather platform. Create your own extension board (wing) by soldering connectors and components. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=861	1
Lithium Polymer Battery	Transform the Feather into an autonomous plateform with this 800mAh Lipo. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=1302	1
BMP280 – Barometric pressure sensor	Easily evaluate pressure, altitude and temperature. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=1118	1
TMP36 – analog temperature sensor 	Transform the sensor voltage read on analog input into an easy-to-read temperature. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=82	1
RFM69HCW Transceiver Radio 	Transport data over long distance with packet radio. One breakout act as emitter, the second one as receiver. disponible ici chez MCHobby https://shop.mchobby.be/product.php?id_product=1390	2

Arduino IDE

Sommaire

Getting prepared for a Training

Installing the Arduino IDE environment + dependencies would involve more than 220 Mb download (100 Mb for Arduino, 120 Mb for Arduino SAMD support, 20 Mb for Adafruit SAMD support).

It is important to get prepared before the training. It is not possible to rely on the guest network to download such amount of data for each of the participant.

So, if you intend to follow a training, get prepared by downloading and installing the complete environment as suggested here follow.

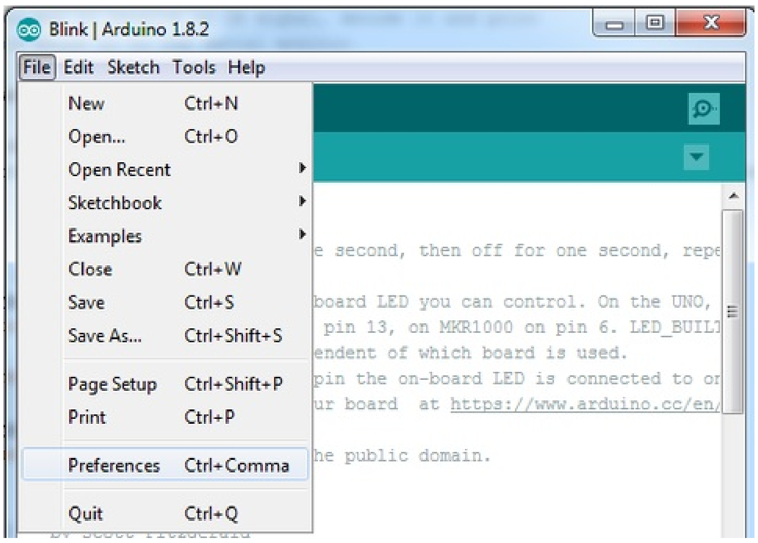
Install Arduino IDE (the .CC version)

As first operation, you have to install the **Arduino IDE from Arduino.CC** (not Arduino.ORG). To follow this guide, you must have the **version 1.8** or higher.

Arduino IDE Download
http://www.arduino.cc/en/Main/Software

Register Additional Boards

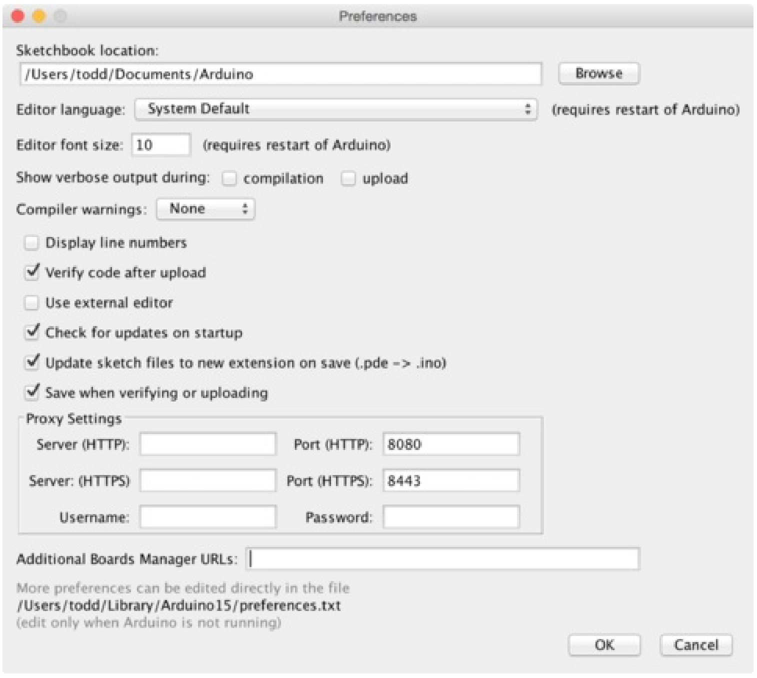
Once the **last version of Arduino IDE** installed, open the IDE and select the **Preference** menu (available in the **File** menu for *Windows* and *Linux* --or-- under the **Arduino** menu for *OS X*).



```
e second, then off for one second, repe

board LED you can control. On the UNO,
 pin 13, on MKR1000 on pin 6. LED_BUILT
endent of which board is used.
pin the on-board LED is connected to on
ur board  at https://www.arduino.cc/en/

he public domain.
```

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

You should see a dialog box like the following.


Preferences

Sketchbook location: /Users/todd/Documents/Arduino [Browse]

Editor language: System Default (requires restart of Arduino)

Editor font size: 10 (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers
✓ Verify code after upload
☐ Use external editor
✓ Check for updates on startup
✓ Update sketch files to new extension on save (.pde -> .ino)
✓ Save when verifying or uploading

Proxy Settings
Server (HTTP): Port (HTTP): 8080
Server: (HTTPS) Port (HTTPS): 8443
Username: Password:

Additional Boards Manager URLs:

More preferences can be edited directly in the file
/Users/todd/Library/Arduino15/preferences.txt
(edit only when Arduino is not running)

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

We will add an URL in the new option **Additional Boards Manager URLs** (*the URL to handle additional boards*).

This field contains an URL list (coma separated). Each new URL can only be added once in this list.

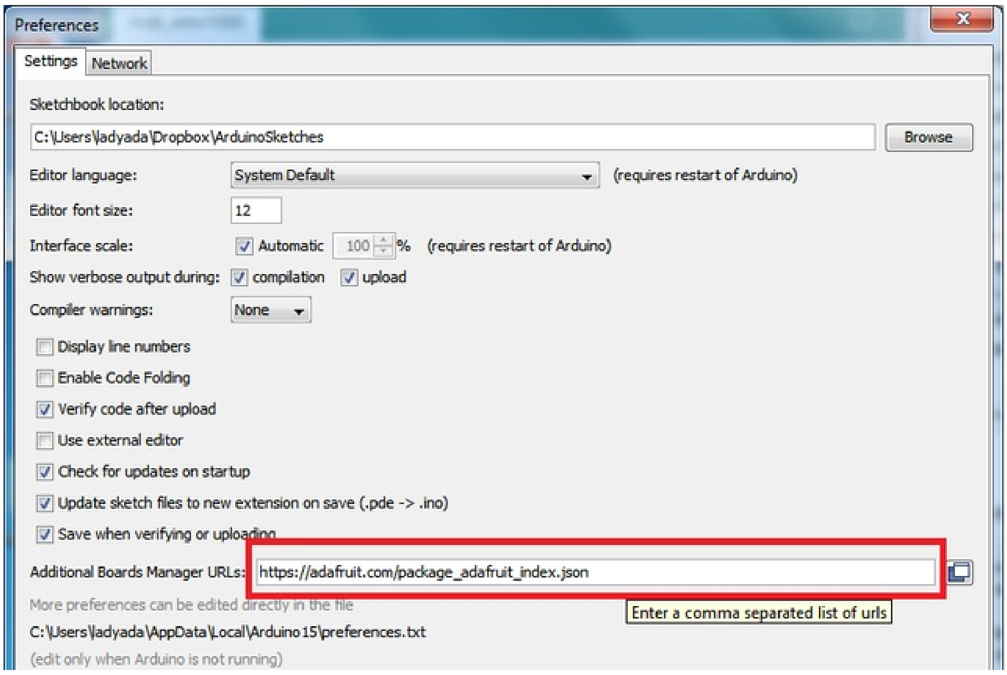
This new Adafruit's board and updates of existing boards will be collected by the "Board Manager" (each time you open it). The URLs point to the index files used by the board manager to build the list of the board available to download.

If you want to know the Arduino IDE's supported boards then browse the list of URLs of managed boards https://github.com/arduino/Arduino/wiki/Unofficial-list-of-3rd-party-boards-support-urls#list-of-3rd-party-boards-support-urls (Arduino Wiki page).

For this Feather M0 Express, we only need to add a single URL. However, it is possible to add several URLs separated by a coma.

Copy/paste the link here below in the field **Additional Boards Manager URLs** (of the Arduino IDE "preference" window).

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


Preferences

Sketchbook location: C:\Users\ladyada\Dropbox\ArduinoSketches [Browse]

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ✓ Automatic 100 % (requires restart of Arduino)

Show verbose output during: ✓ compilation ✓ upload

Compiler warnings: None

☐ Display line numbers
☐ Enable Code Folding
✓ Verify code after upload
☐ Use external editor
✓ Check for updates on startup
✓ Update sketch files to new extension on save (.pde -> .ino)
✓ Save when verifying or uploading

Additional Boards Manager URLs: https://adafruit.com/package_adafruit_index.json

Enter a comma separated list of urls

More preferences can be edited directly in the file
C:\Users\ladyada\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Here follows a small description of the boards available with the URLs:

- **Adafruit AVR Boards** - support for Flora, Gemma, Feather 32u4, Trinket and Trinket Pro.
- **Adafruit SAMD Boards** - support for the Feather M0, Metro M0, Circuit Playground Express, Gemma M0 and Trinket M0
- **Arduino Leonardo & Micro MIDI-USB** - Add the MIDI over USB support for Flora, Feather 32u4, Micro &

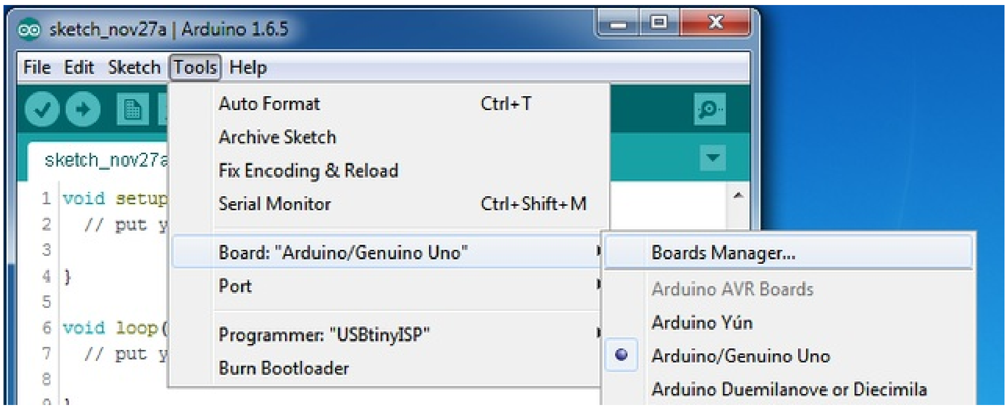
Leonardo (use the projet arcore https://github.com/rkistner/arcore).

Once the "OK" button pressed, the new preferences are saved.

We can now install the needed board into the *Board Manager*.

Install the Feather M0 board

Open the *Boards Manager* available via the menu *Tools->Board* .



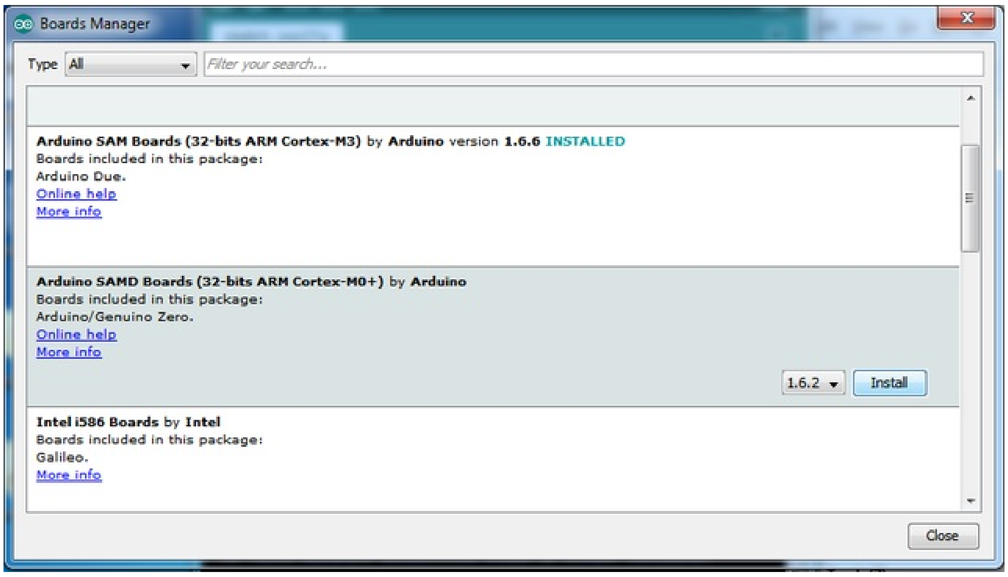
Crédit: AdaFruit Industries www.adafruit.com *http://www.adafruit.com*

Once the *Boards Manager* opened, select the *Contributed* type. Once done, you will be able to install the board attached to `package_adafruit_index.json` URL.

Install the SAMD boards

Now, we will install the **Arduino SAMD board** version **1.6.15** or higher.

You can type in the **Arduino SAMD** in the search box to quickly find the package, then press the **Install** button.



Arduino SAM Boards (32-bits ARM Cortex-M3) by **Arduino** version **1.6.6** INSTALLED
Boards included in this package:
Arduino Due.
Online help
More info

Arduino SAMD Boards (32-bits ARM Cortex-M0+) by **Arduino**
Boards included in this package:
Arduino/Genuino Zero.
Online help
More info

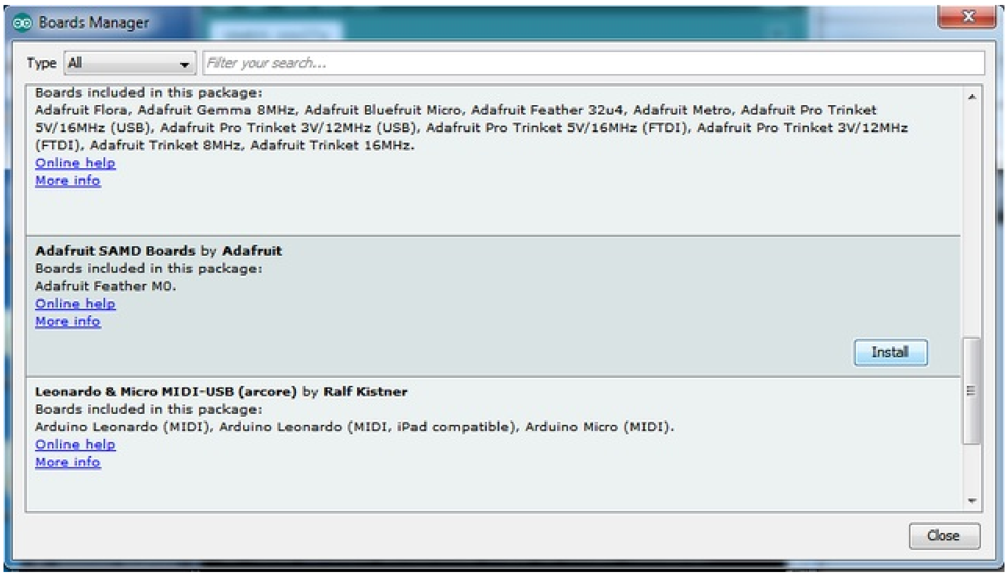
Intel i586 Boards by **Intel**
Boards included in this package:
Galileo.
More info

Crédit: AdaFruit Industries www.adafruit.com *http://www.adafruit.com*

Install the ADAFRUIT SAMD board

After the SAMD board, it's time to install the "Adafruit SAMD" package to support the Adafruit boards.

You can type in the **Adafruit SAMD** in the search field to find the package. Once located, press the **Install** button.



Boards included in this package:
Adafruit Flora, Adafruit Gemma 8MHz, Adafruit Bluefruit Micro, Adafruit Feather 32u4, Adafruit Metro, Adafruit Pro Trinket 5V/16MHz (USB), Adafruit Pro Trinket 3V/12MHz (USB), Adafruit Pro Trinket 5V/16MHz (FTDI), Adafruit Pro Trinket 3V/12MHz (FTDI), Adafruit Trinket 8MHz, Adafruit Trinket 16MHz.
Online help
More info

Adafruit SAMD Boards by **Adafruit**
Boards included in this package:
Adafruit Feather M0.
Online help
More info

Leonardo & Micro MIDI-USB (arcore) by **Ralf Kistner**
Boards included in this package:
Arduino Leonardo (MIDI), Arduino Leonardo (MIDI, iPad compatible), Arduino Micro (MIDI).
Online help
More info

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

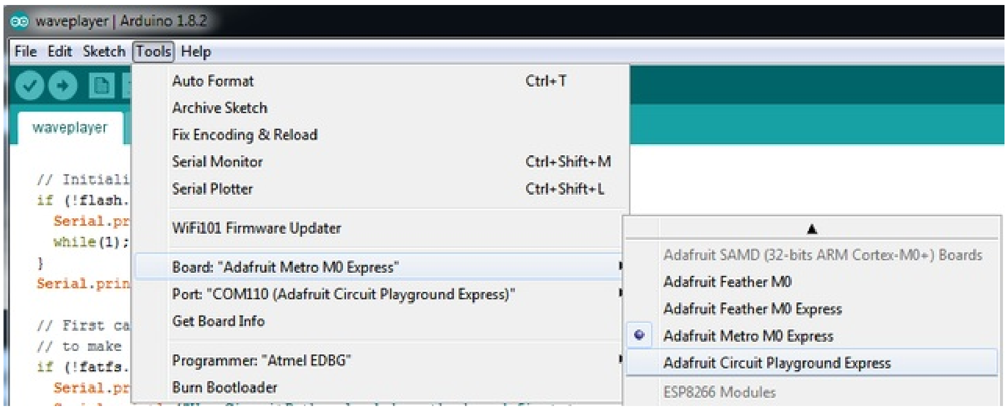
We strongly recommand to restart the Arduino IDE (it is not required but it is better to do it anyway).

Check installed boards

Once the Arduino IDE restarted to be sure that boards are properly installed, you should be able to select the new boards in the interfaces (and to upload code) via the menu *Tools -> Board*.

Select the board for the kit among those now available:

- Feather M0 (for the Feather M0 boards other than Feather M0 Express)
- **Feather M0 Express**
- Metro M0 Express
- Circuit Playground Express
- Gemma M0
- Trinket M0


waveplayer | Arduino 1.8.2
File Edit Sketch Tools Help
waveplayer
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Adafruit Metro M0 Express"
Port: "COM110 (Adafruit Circuit Playground Express)"
Get Board Info
Programmer: "Atmel EDBG"
Burn Bootloader
Adafruit SAMD (32-bits ARM Cortex-M0+) Boards
Adafruit Feather M0
Adafruit Feather M0 Express
Adafruit Metro M0 Express
Adafruit Circuit Playground Express
ESP8266 Modules

```
// Initiali
if (!flash.
  Serial.pr
  while(1);
}
Serial.prin

// First ca
// to make
if (!fatfs.
  Serial.pr
```

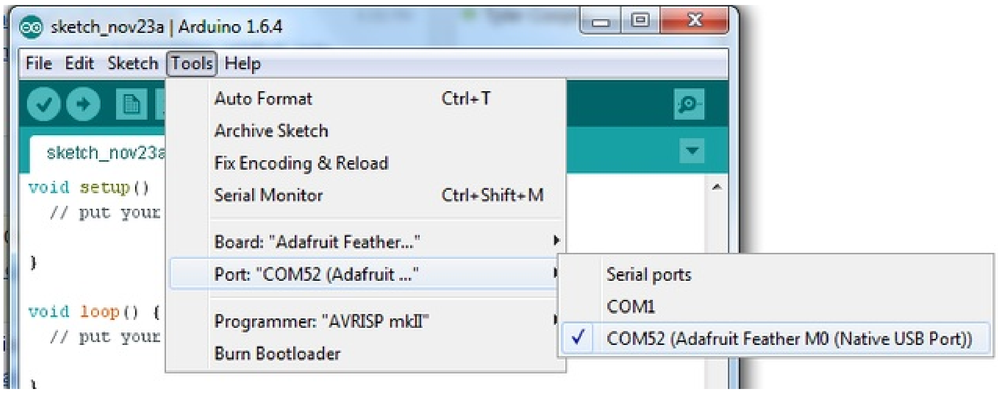
Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Test with BLINK

If you are using WINDOWS then you should have a look to the "Windows Driver" section here below!

Now, we can upload your first sketch to the board (the "blink" sketch)!

Wire your board to the computer and wait for the operating system to identify it (this may take few seconds). Once identified, the Serial port/COM is available in the list of serial port available.


sketch_nov23a | Arduino 1.6.4
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Board: "Adafruit Feather..."
Port: "COM52 (Adafruit ..."
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
COM52 (Adafruit Feather M0 (Native USB Port))

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

It is now time to upload the Blink sketch

```
void setup() {
  // init the digital pin #13 as OUTPUT
  pinMode(13, OUTPUT);
}

// the "loop" function is executed again and again (in a infinite loop)
void loop() {
  digitalWrite(13, HIGH);   // Light up the LED (HIGH level = 3.3v)
  delay(1000);              // Wait 1 second
  digitalWrite(13, LOW);    // Switch off the LED (LOW level = 0V)
  delay(1000);              // Wait 1 second
}
```

Then click on the "upload" button! You should be able to see the board LED blinking. You can tune the blink speed by updating the value for the `delay()` function.

If you get upload issue then you should check if you selected the proper board type.

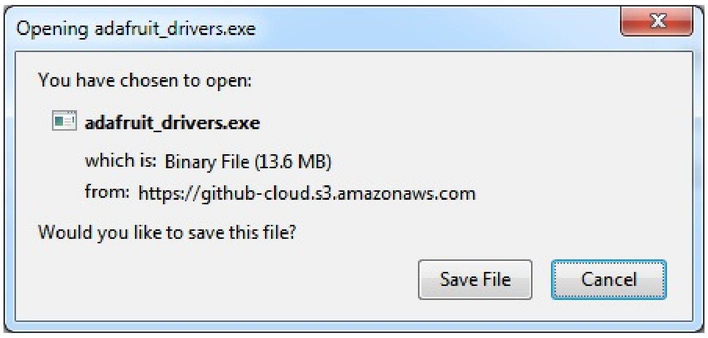
Install the Windows Driver (Win 7 only)

You will certainly have to install Windows Driver before plugin the board on the computer.

You can download the Windows Driver from the Adafruit Industries server :

Download the Adafruit Drive
https://github.com/adafruit/Adafruit_Windows_Drivers/releases/download/2.0.0.0/adafruit_drivers_2.0.0.0.exe

Download and start the setup software.


Opening adafruit_drivers.exe

You have chosen to open:

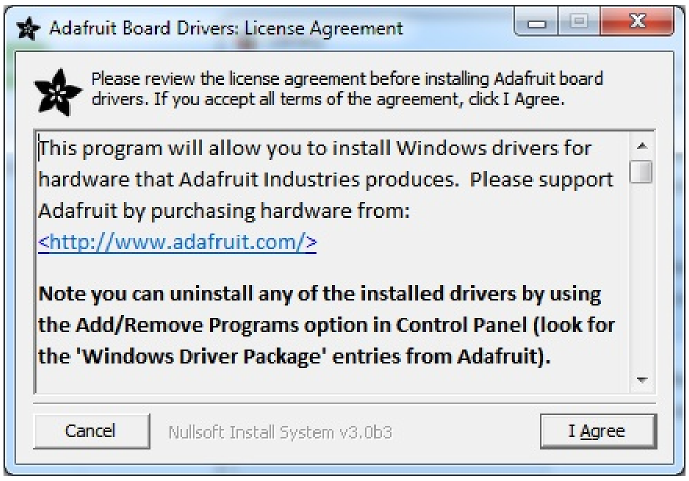
adafruit_drivers.exe

which is: Binary File (13.6 MB)
from: https://github-cloud.s3.amazonaws.com

Would you like to save this file?

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Execute the setup software! It will be necessary to review the licensing instruction as it contains the SiLabs setup and FTDI driver.


Adafruit Board Drivers: License Agreement

Please review the license agreement before installing Adafruit board drivers. If you accept all terms of the agreement, click I Agree.

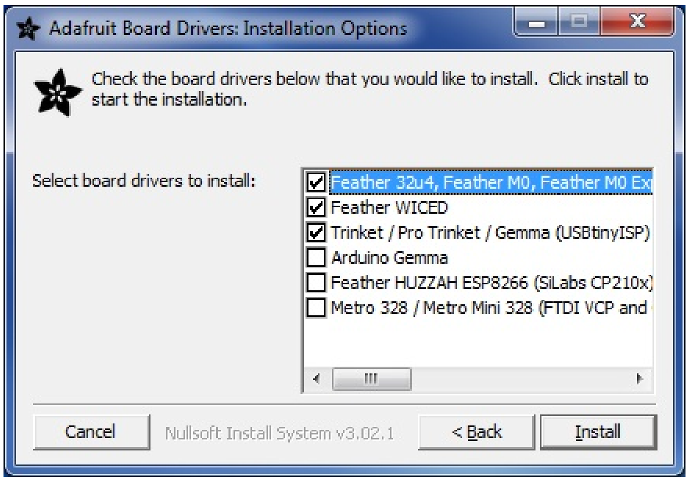
This program will allow you to install Windows drivers for hardware that Adafruit Industries produces. Please support Adafruit by purchasing hardware from: <http://www.adafruit.com/>

Note you can uninstall any of the installed drivers by using the Add/Remove Programs option in Control Panel (look for the 'Windows Driver Package' entries from Adafruit).

Cancel | Nullsoft Install System v3.0b3 | I Agree

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Select the driver you want to install, the default selection would be perfect for the Adafruit boards!


Adafruit Board Drivers: Installation Options

Check the board drivers below that you would like to install. Click install to start the installation.

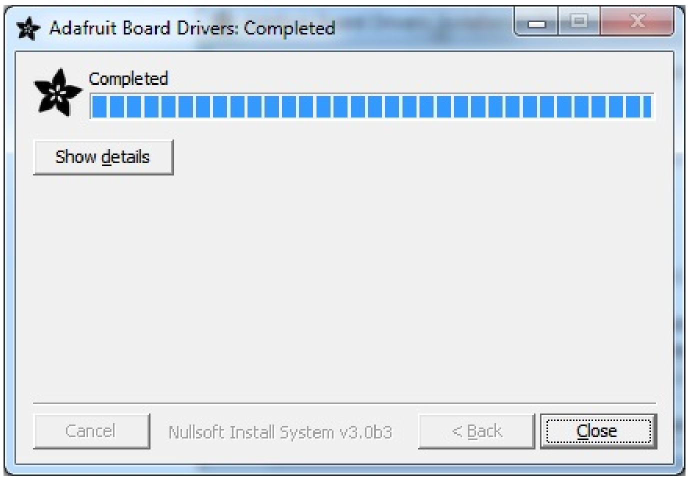
Select board drivers to install:

- [x] Feather 32u4, Feather M0, Feather M0 Ex
- [x] Feather WICED
- [x] Trinket / Pro Trinket / Gemma (USBtinyISP)
- [ ] Arduino Gemma
- [ ] Feather HUZZAH ESP8266 (SiLabs CP210x)
- [ ] Metro 328 / Metro Mini 328 (FTDI VCP and

Cancel | Nullsoft Install System v3.02.1 | < Back | Install

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Push the '*Install* button to proceed.


Adafruit Board Drivers: Completed

Completed

Show details

Cancel | Nullsoft Install System v3.0b3 | < Back | Close

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Feather User Guide

Sommaire

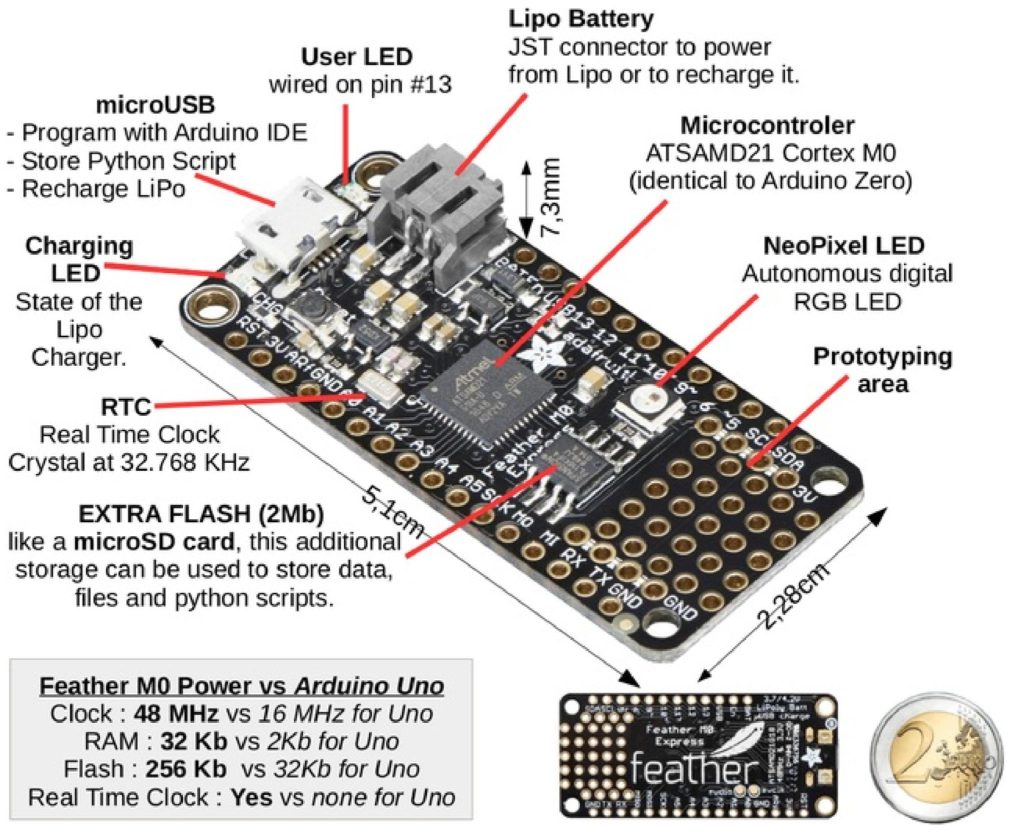
Feather M0 Express

The Feather M0 use the ATSAMD21G18 ARM Cortex M0+ processor with 3.3V logic and 48 MHz. This chip embed 256K of FLASH (8x more than the Atmega328/Uno) and has 32K RAM (16x more than UNO)!

As Feather M0 is compatible Arduino IDE, everything learned for Arduino Uno can be applied to Feather M0. Just care about voltage, the UNO is 5V Logic and the **Feather 3.3V logic**.

Feather M0 Features
20 GPIOs
6 Analog inputs 12bits value from 0 to 4095.
1 Analog output 10 bits value from 0 to 1023.
PWM on all pins
Hardware I2C et SPI buses
UART

Feather are:
* Small (5c2xm)
* Light (4.7Gr)
* Powerful
* Versatile
* Polyvalent
* Available with complete ecosystem https://shop.mchobby.be/category.php?id_category=87



Lipo Battery
JST connector to power from Lipo or to recharge it.
User LED
wired on pin #13
microUSB
- Program with Arduino IDE
- Store Python Script
- Recharge LiPo
Microcontroler
ATSAMD21 Cortex M0 (identical to Arduino Zero)
7,3mm
Charging LED
State of the Lipo Charger.
NeoPixel LED
Autonomous digital RGB LED
Prototyping area
RTC
Real Time Clock
Crystal at 32.768 KHz
EXTRA FLASH (2Mb)
like a **microSD card**, this additional storage can be used to store data, files and python scripts.
5,1cm
2,28cm
feather

Feather M0 Power vs *Arduino Uno*
Clock : **48 MHz** vs *16 MHz for Uno*
RAM : **32 Kb** vs *2Kb for Uno*
Flash : **256 Kb** vs *32Kb for Uno*
Real Time Clock : **Yes** vs *none for Uno*

This ATSAMD21G18 has USB built-in which offers USB-to-Serial and debug feature (so no need for FTDI-like chip).

The feather has the following specs:

- Size: 51mm x 23mm x 8mm
- Weight: 5 grams
- ATSAMD21G18 @ 48MHz
- 3.3V logic/power
- 256KB of FLASH
- 32KB of RAM
- No EEPROM
- RTC & Clock based on 32.768 KHz crystal
- 3.3V regulator (500mA peak current) with Power/enable pin
- USB native support (has USB bootloader)
- 20 GPIO pins (PWM on all pins)
- Hardware Serial + Hardware I2C + Hardware SPI support
- 6 x 12-bit analog inputs (from 0 to 4095)
- 1 x DAC 10-bit analog ouput (from 0 to 1024)
- Lipo Charger included: 100mA charging with status LED
- Red LED attached on pin #13 (like Arduino UNO)
- Reset button
- Mini NeoPixel
- 2 MB SPI additional Flash storage.

The Feather M0 Express can be be programmed with Arduino IDE -OR- with CircuitPython (a flavor of MicroPython). This tutorial focus on Arduino IDE development. Check the Adafruit Tutorial if you are interested in **CircuitPython Programming** https://learn.adafruit.com/adafruit-feather-m0-express-designed-for-circuit-python-circuitpython/what-is-circuitpython .

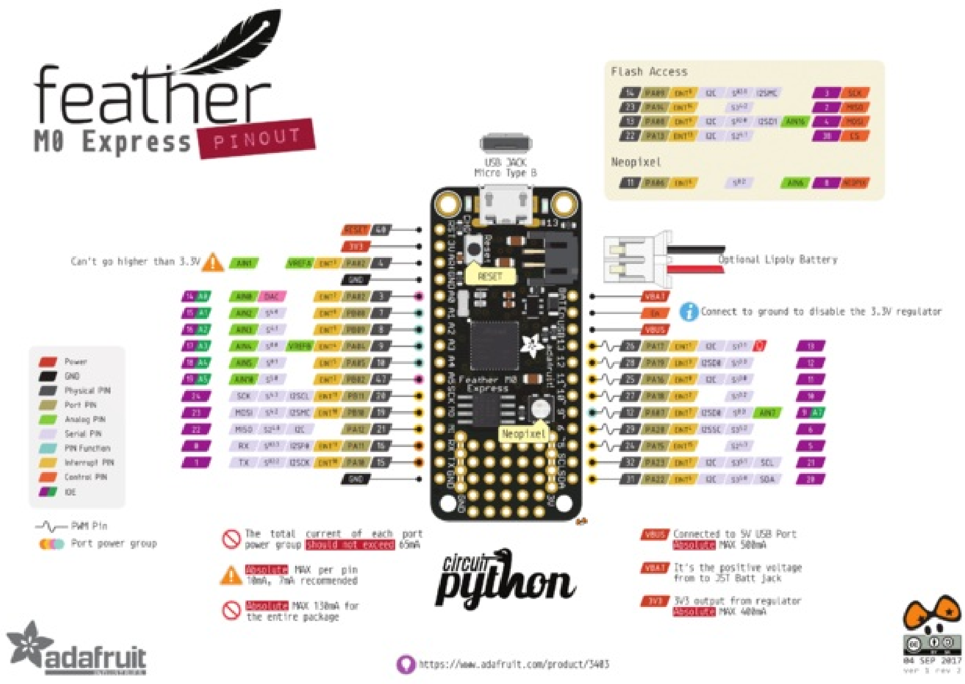
About the additionnal Flash:
The SPI Flash storage act like a tiny hard drive.

- Under CircuitPython, the 2 MB Flash is used to the Python scripts, libraries and other files.
- With Arduino, the 2 MB Flash is used to read/write files to it (like a SD card). Adafruit has helper program to access those files over USB.

The UF2 bootloader:
The Feather M0 is pre-loaded with an UF2 bootloader which looks like a USB Flash Drive. Simply drag an UF2 firmware on the USB Flash drive to completely reprogram de board firmware (No special tools, No special drivers needed)! The UF2 bootloader can be used to load up CircuitPython, MakeCode PXT file or Arduino IDE (bossa-compatible).

Feather M0 Express PINOUT

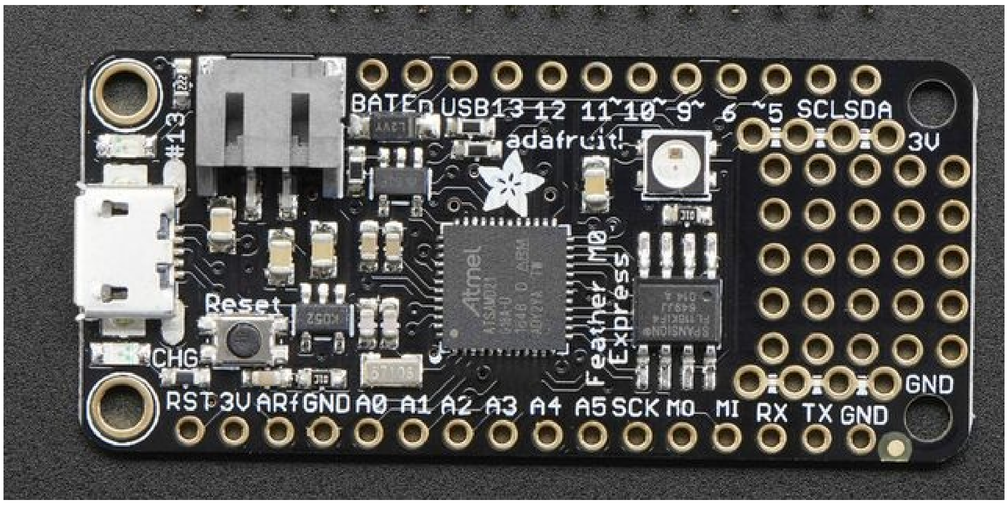
Introduction


feather
M0 Express PINOUT
Flash Access
Neopixel
USB JACK Micro Type B
RESET
Optional Lipoly Battery
Can't go higher than 3.3V
Connect to ground to disable the 3.3V regulator
Power
GND
Physical PIN
Port PIN
Analog PIN
Serial PIN
PIN Function
Interrupt PIN
Control PIN
IDE
PWM Pin
Port power group
The total current of each port power group should not exceed 65mA
Absolute MAX per pin 10mA, 7mA recommended
Absolute MAX 130mA for the entire package
Connected to 5V USB Port Absolute MAX 500mA
It's the positive voltage from to JST Batt Jack
3V3 output from regulator Absolute MAX 400mA
circuit python
adafruit
https://www.adafruit.com/product/3403
04 SEP 2017

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

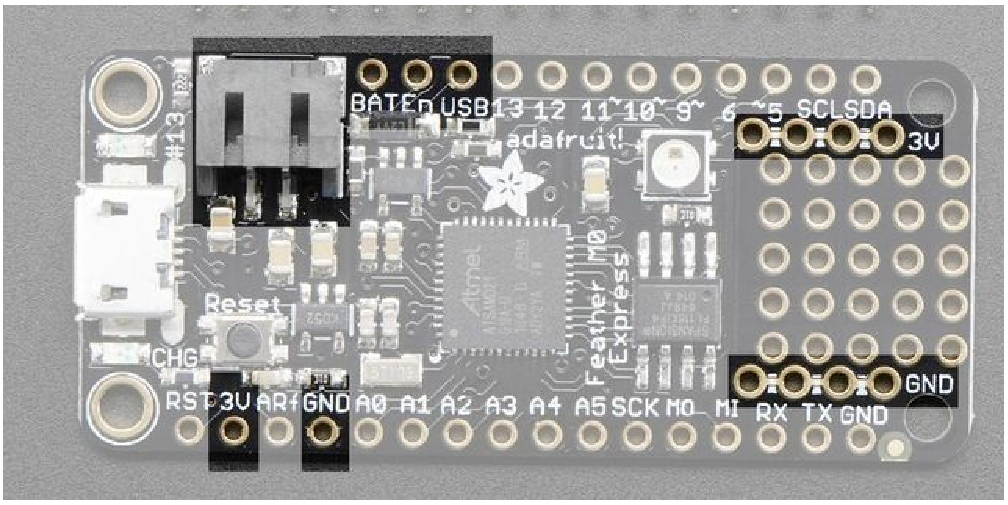
Click to enlarge

(Please note that AREF is PA03 (and not PA02)


BAT EN USB 13 12 11 10 9 6 5 SCL SDA
3V
GND
RST 3V AREF GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND
Reset
CHG
#13
adafruit
Feather M0 Express
Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

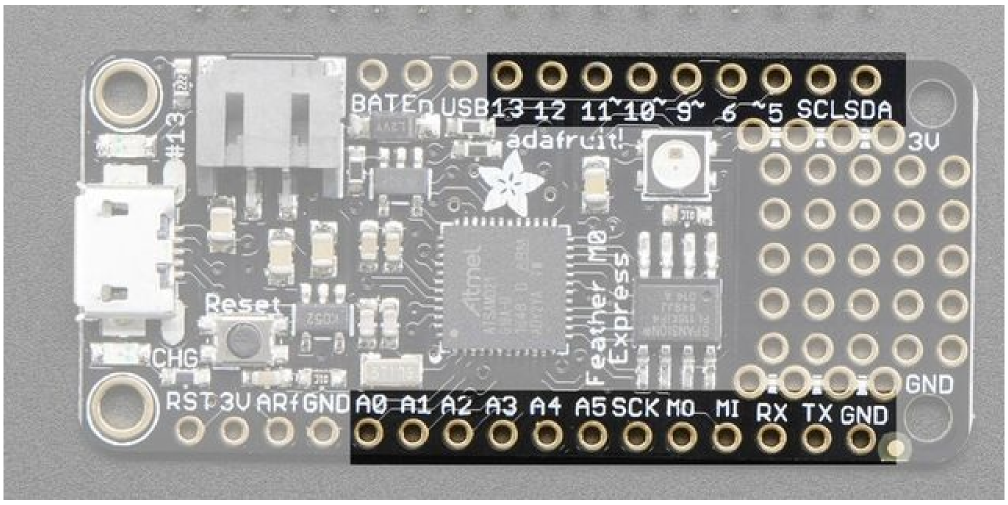
The Feather M0 benefits from the Cortex M0 microcontroler hardware. If has many buses and many pins. Let's review them togheter!

Power Pin


BAT EN USB 13 12 11 10 9 6 5 SCL SDA
3V
GND
RST 3V AREF GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND
Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

- **GND** : is the common ground. The 0v reference voltage for all the logic and power supplies.
- **BAT** : positive pin from the JST connector (the optional Lipo).
- **USB** : positive pin from the USB connector. Will allow you to detect if the board is connected on USB.
- **EN** : *Enable* pin of the 3.3V regulator. This pin has a Pull-Up résistor. Connect the **EN** pin to the ground will shutdown the 3.3V regulator.
- **3V** : Output of the 3.3V regulator (500mA peak)

Logical pins


BAT EN USB 13 12 11 10 9 6 5 SCL SDA
3V
GND
RST 3V AREF GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND
Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

This concerns all the I/O pins on the microcontroler.

All PINS are 3.3V Logic

- **Almost all pins can generate PWM signal** (also called "*analog output*" on Arduino UNO board).
- **All pins can be used for Interrupt Request (IRQ).**

The following pins have specific features:

- **#0 / RX** - GPIO #0, also receiving pin of the **Serial1** UART (hardware *serial port*, input). Also an analog input.
- **#1 / TX** - GPIO #1, also transmitting pins of the **Serial1** UART (hardware *serial port*, output). Also an analog input.
- **SDA** - Data pin for the I2C bus. There is no pull-up resistor on that pin, so you need to add a 2.2K-10K pull-up when used for I2C bus.
- **SCL** - Clock pin for the I2C bus. There is no pull-up resistor on that pin, so you need to add a 2.2K-10K pull-up when used for I2C bus.
- **#5** - GPIO #5
- **#6** - GPIO #6
- **#9** - GPIO #9, also analog input **A7**. This analog input is wired on a divider resistor bridge to read the Lipoly battery voltage. As a consequence, the voltage of that pin is fixed to a voltage around ~2V.
- **#10** - GPIO #10
- **#11** - GPIO #11
- **#12** - GPIO #12
- **#13** - GPIO #13, also connected to the **red LED** near if the microUSB située près du connecteur micro USB.
- **A0** - **Analog input A0** and also a real **Analog output** since the DAC is wired on this pin (DAC = Digital to Analog Converter). The output voltage can be set with a value between 0 and 3.3V. Unlike the PWM output, this pin is a real analog output. So you can experiment with signal generator.
- **A1 to A5** - each pin is an Analog input and also a Digital Input/Output.
  It is a 12-bit analog inputs (so with values from 0 to 4095)
- **SCK/MOSI/MISO** - those are the pins for the hardware SPI bus. The pin can also been used to as Digital Input/Output.

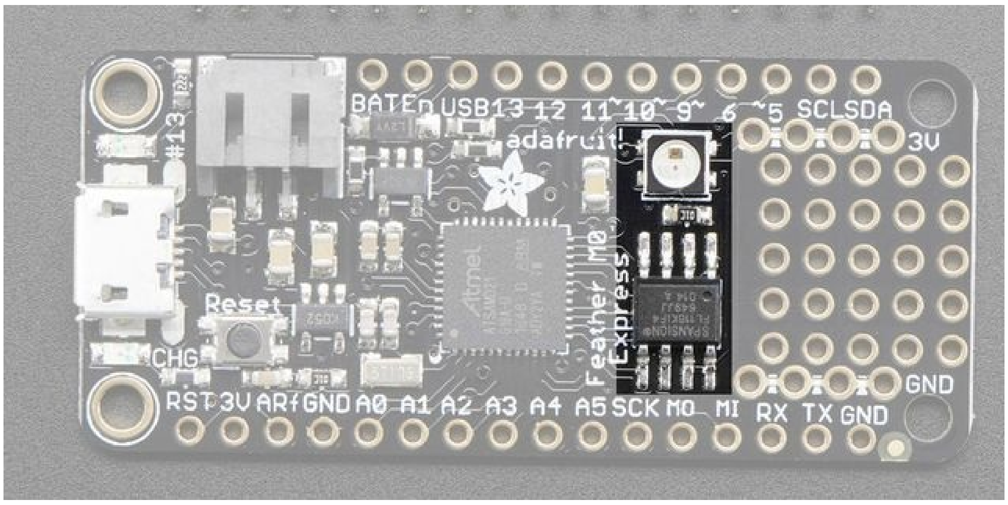
About buses:

- The SPI bus can reach really high speed so it is a good idea to preserve it, mostly useful with TFT display requiring high throughput to display the content.
- The I2C bus can be shared among several devices (usually sensors). So it is also a good idea to preserve the SDA & SCL pins.

SPI Flash and NeoPixel

The "Express" product line is designed to also run CircuitPython. To ease the prototyping, Adafruit added 2 additional items on the Feather M0:

- a mini NeoPixel LED (NeoPixel are Digital RGB LEDs that can be controled with only one data pin)
- a 2 MB additional memory (Flash memory on SPI bus.



Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

The **NeoPixel** LED is wired on the #8 (in Arduino), so you can use the library https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use%7CNeoPixel by configuring a ribbon of only 1 pixel. The NeoPixel is powered with 3.3V but this doesn't impact the color or the brightness.

NeoPixel

The NeoPixel is also used by the bootloader to inform the user about the bootloader state:

- Green : The device has been proprely enumerated over the USB interface.
- Red : error with the USB.

Note: In CircuitPython, the LED is used to mention the running status.

The SPI Flash memory

The Flash memory is wired on 4 reserved pins. Those pins are not made available as GPIO, so no worries about collision risk when using device on the Feather SPI bus. The SPI Flash memory is wired on a different SPI bus... so no worries.

Under Arduino

The Flash pins are:

- **SCK** = pin #3,
- **MISO** = pin #2,
- **MOSI** = pin #4
- **CS** = pin #38.

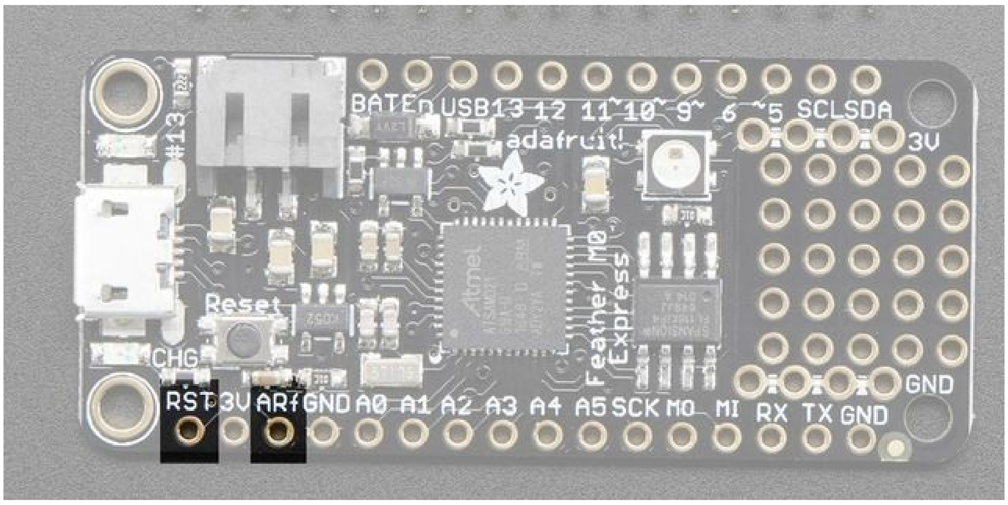
On the **Feather M0 Express** you will be able to access this SPI bus (and the Flash) under the name **SPI1**. So the Flash device is totally distinct from the Feather GPIO's.

When used with Arduino, this SPI Flash memory would allow read/write operations.

Under CircuitPython (MicroPython Flavor)

Under CircuitPython, the SPI Flash memory is a native storage for the Python interpreter. The Flash memory does appears as read only for the user code.

Other pins!



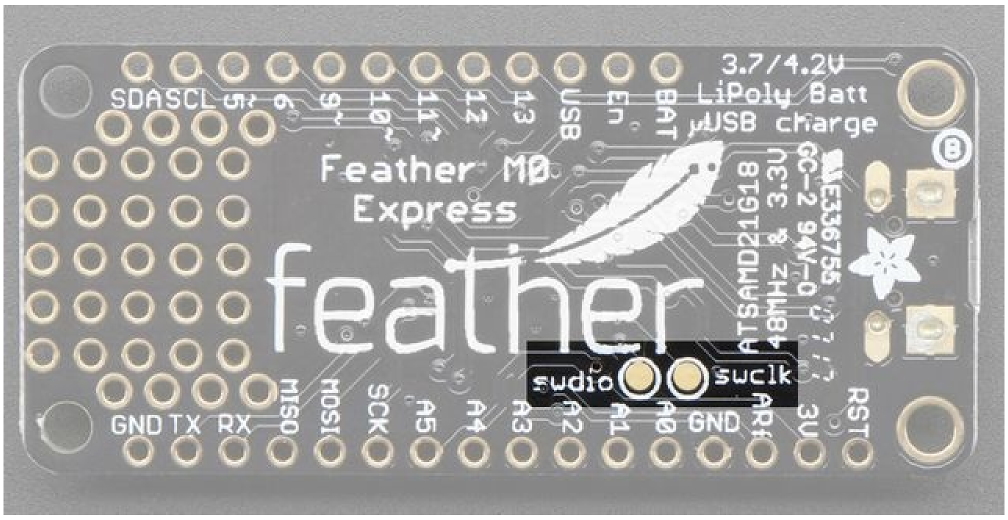
Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

- **RST** - "*Reset*" pin. Wire this pin to the ground (GND) to reset the microcontroler and start the bootloader.
- **ARef** - "Analog Reference" pin used when the microcontroler reads analog voltage. As default behavior, the reference voltage is identical to the logic level (so 3.3V). However, you can use a another analog reference voltage (eg: 1.5V) and inform you software to use the external reference AREF EXTERNE. In such case, the 12bits analog reads (value from 0 to 4095) would cover a voltage range from 0 to 1.5V, so a resolution of 1.5/4095=0.366mV.

  The reference voltage can never exceed 3.3v. The analogic pin voltage can never exceed the ARef reference voltage!

Debug Interface

For advanced users.



Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

- **SWCLK et SWDIO** - those contact points, visible under the board, are used to program the microcontroler. You can also use thoses contact to connect the SWD debugger.

Forcing Flash Mode

Issue Description

Sometime, it happens that compilation phase get complete successfully but the binary can't get uploaded to the board.

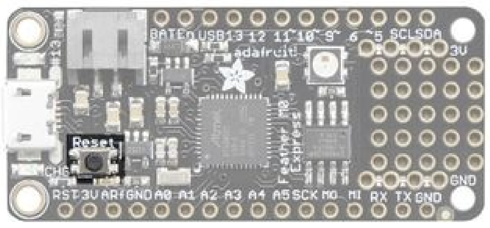
Reason

The M0 board Flash Mode does not get activated from the Arduino IDE environment.

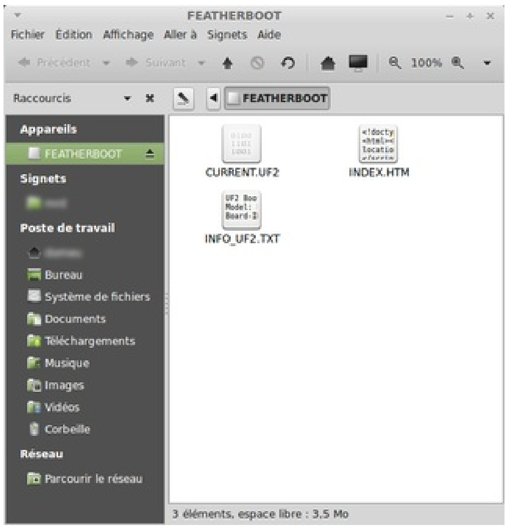
Workaround

Activate the Flash Mode before compiling & uploading your sketch.

To do so, **PRESS TWICE the reset** button.


Reset
BATEn USB 13 12 11 10 9 6 5 SCL SDA
adafruit
Feather M0 Express
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND

Once done, the Flash Mode is activated and the board will show itself as an USB Stick named "FEATHERBOOT" (Windows and other operating system does show a file navigation window).



Then, **press the Upload** button into Arduino IDE environment.

This time, the sketch will be copied to the board... and Feather M0 properly reset.

When Writing Sketch

Sommaire

Forewords

The ATSAMD21 is still a newcomer in the Arduino-compatible world. **Most** of sketch and libraries would work on ATSAMD21 but somes things have to be pointed out!

Le notes here under would apply to the M0 boards.

Analog reference

If you want to use the **ARef** for a voltage reference under 3.3v, the line code to use is `analogReference(AR_EXTERNAL)` with **AR_EXTERNAL** and not *EXTERNAL*.

Pins and pull-up

The old way of activating the pull-up resistor was:

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

Because the *pullup-selection* register was the same register than *output-selection* register.

For the M0 (as for many plateform), the code to use is:

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

Serial ou SerialUSB

99.9% of Arduino Sketch does use **Serial.print** for debugging purpose (or serial output). On the official SAMD/M0 Arduino, this instruction use the Serial5 port which is not exposed on a Feather.

Instead, the USB port on official Arduino M0 is called **SerialUSB**.

Adafruit did fix this on the Adafruit M0 by redirecting Serial call to USB calls. So, when using a Feather M0 everything appears to work to properly without requiring any changes.

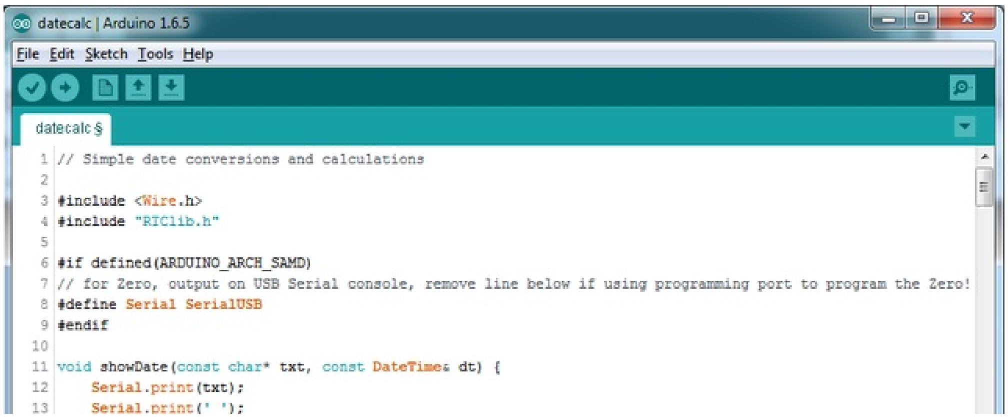
However, if you want to use a, official Arduino SAMD, you will have to use `SerialUSB` instead of `Serial`. So the Adafruit produit is better on this point.

If you want to use an official M0 without the need to change all the `Serial.print()` call to `SerialUSB.print()` , then place the following code:

```
#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
   // required for Serial operations on Zero based board
   #define Serial '''SERIAL_PORT_USBVIRTUAL'''
#endif
```

just before the first function definition in the code.

Example:



```
// Simple date conversions and calculations

#include <Wire.h>
#include "RTClib.h"

#if defined(ARDUINO_ARCH_SAMD)
// for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!
#define Serial SerialUSB
#endif

void showDate(const char* txt, const DateTime& dt) {
    Serial.print(txt);
    Serial.print(' ');
```

Crédit: AdaFruit Industries www.adafruit.com *http://www.adafruit.com*

AnalogWrite / PWM on Feather M0

After looking through the SAMD21 datasheet, it appears that some of the options listed in the multiplexer table don't exist on the specific chip used in the Feather M0.

For all SAMD21 chips, there are two peripherals that can generate PWM signals: The Timer/Counter (TC) and Timer/Counter for Control Applications (TCC). Each SAMD21 has multiple copies of each, called 'instances'.

Each TC instance has one count register, one control register, and two output channels. Either channel can be enabled and disabled, and either channel can be inverted. The pins connected to a TC instance can output identical versions of the same PWM waveform, or complementary waveforms.

Each TCC instance has a single count register, but multiple compare registers and output channels. There are options for different kinds of waveform, interleaved switching, programmable dead time, and so on.

The biggest members of the SAMD21 family have five TC instances with two 'waveform output' (WO) channels, and three TCC instances with eight WO channels:

- TC[0-4],WO[0-1]
- TCC[0-2],WO[0-7]

And those are the ones shown in the datasheet's multiplexer tables.

The SAMD21G used in the Feather M0 only has three TC instances with two output channels, and three TCC instances with eight output channels:

- **TC[3-5],WO[0-1]**
- **TCC[0-2],WO[0-7]**

By following the signals to the pins made available on a Feather M0, **the following pins would not be able to produce PWM signal**:

- **Analog Pin A5**

The following pins can be configured as PWM (without conflict) as long as the SPI, I2C & UART keeps their protocol functions:

- **Digital pins 5, 6, 9, 10, 11, 12 et 13**
- **Analog pins A3 et A4**

When only the SPI keeps the protocol function, you can also do PWM on the following pins:

- **TX** (Digital pin 1)
- **SDA** (Digital pin 20)

analogWrite() and range of PWM value

When using an AVR (like Arduino Uno), the instruction `analogWrite(pin, 255)` on a PWM output would result in a permanent HIGH signal on the output PIN.

On a Cortex ARM microcontroler, the output signal would be 255/256th. As a consequence, there is always a small pulse-down to 0v. If you need a continuously HIGH signal then the `analogWrite(pin, 255)` must be replaced by `digitalWrite(pin, HIGH)`

Missing Header file

You may have some code using a library not supported by the M0 code. As example, if you have some code containing the following line:

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

Then you will get the following error

```
fatal error: util/delay.h: No such file or directory
  #include <util/delay.h>
                         ^
compilation terminated.
Error compiling.
```

Wich allow to identiy the line (and the file) where the error occured. You would just need to include the library loading inside a `#ifdef` structure like showed:

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

The line here upper would not include the header for the listed architectures.

If the `#include` is present in your Arduino sketch then you may try to remove the `#include` line.

Start the Bootloader

On most of the AVRs (like Arduino Uno), simply press the **reset** button with the microcontroler connected on USB would manually start the bootloader. The bootloader would automatically exists after few seconds.

On a M0 microcontroler, you will have to **double click** the **reset** button. You will see the LED pulsing on red meaning that the bootloader is active. Once in this mode, the M0 would stay in the bootloader mode forever (there is no "time out"). Click once again on the "reset" button to restart the microcontroler.

Memory alignment

There is few change that you reach this issue... but being aware of this may help.

If you are using the 8 bits plateform then you probably know that TypeCast can be performed on on variables. Example:

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

But there is no warranty that this may work properly on 32 bits AVR because **mybuffer** may not been aligned on 2 or 4 bytes (voir memory alignment https://en.wikipedia.org/wiki/Data_structure_alignment on Wikipedia).

An ARM Cortex-M0 can only directly access to data by bloc of 16-bits (every 2 or 4 bytes). Trying to access an even byte (byte in position 1 or 3) would cause an hardware fault and will stop a MCU.

Thankfully, there is a very simple workaround... by using the `memcpy` function!

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

Floating point conversion (dtostrf)

As for the Arduino AVR, the M0 libraries doesn't offer a full support to convert floating point value to string.

The functions like `sprintf` would not convert floating point values. Thankfully, the standard AVR-LIBC includes the `dtostrf` function able to handle this conversion.

Inconveniently, the M0 AVR run-time does not have the dtostrf function! You may see some thread suggesting to **#include <avr/dtostrf.h>** the dtostrf function. But this will not work on M0 even if it compiles**.**

Instead, have a look to this discussion thread to find a `dtostrf` function running proprely:

- http://forum.arduino.cc/index.php?topic=368720.0

How many RAM available ?

The ATSAMD21G18 does have 32K of RAM but you may need to monitor the memory usage for some raison. You can to this with the following function:

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

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

Thanks to this discussion thread http://forum.arduino.cc/index.php?topic=365830.msg2542879#msg2542879 on the Arduino forums for the trick!

Store data in the microcontroler Flash

If you use an AVR (Arduino) on regular basis, you may have a chance to use **PROGMEM**. PROGMEM inform the compiler to store the content of a variable (or a string) into the FLASH memory (to save RAM).

It is a bit more easy on an ARM microcontroler. Just add the word **const** before the variable name:

```
const char str[] = "A quite lonnnnngggggg striiiiiinnnnnnng";
```

The string is now stored in the FLASH. You can handle the string as it was stored inside the RAM, the compiler would automagically read it from the FLASH (no need for special reading function like those required for PROGMEM variables).

You can easily check where the data is stored! Just print the storage address of the variable:

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

If the adress is:

- equal or greater than $2000000 then the data is in the SRAM.
- between $0000 and $3FFFF then the data is stored in FLASH

Using the SPI Flash

Sommaire

Forewords

One of the most exciting feature of the M0 Express board is this small FLASH chip wired on the SPI bus. That memory could be used to provide lot of services like storing files, python script and many more.

You can see that additional FLASH chip like a small SD card continously wired on the board. This flash memory is available through a library very similar to the Arduino's SD card https://www.arduino.cc/en/reference/SD . You can even read and write the files on the CircuitPython filesystem (Circuit Python use this Flash to store the Python Script and files)!

You will need to install the Adafruit SPI Flash Memory library https://github.com/adafruit/Adafruit_SPIFlash in Arduino IDE to use the SPI Flash with your Arduino sketch. Click on the link below to download the library source code, open the zip file and copy the file files into a subfolder named **Adafruit_SPIFlash** (*remove the '-master' added by GitHub on the front of the folder name). Place this new library next to your other Arduino libraries:*

Adafruit Adafruit SPI Flash library
https://github.com/adafruit/Adafruit_SPIFlash

Once the library installed, start your Arduino IDE to access the various examples available in the library:

- **fatfs_circuitpython**
- **fatfs_datalogging**
- **fatfs_format**
- **fatfs_full_usage**
- **fatfs_print_file**
- **flash_erase**

Thoses exemples would allow you to format the Flash memory with the FAT filesystem (the same filesytem than used on SD card), read and write files like we do with SD card.

Format the Flash memory

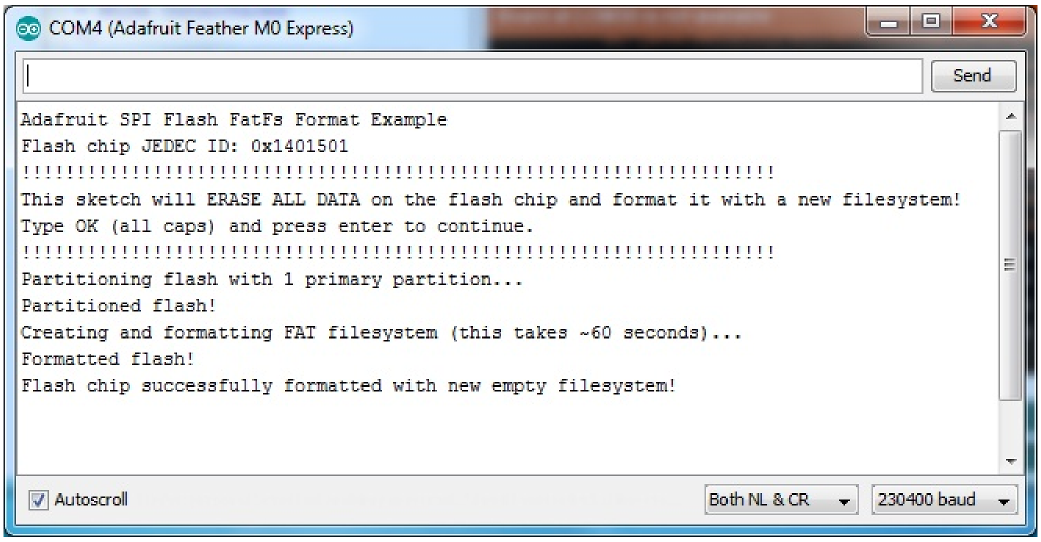
The example **fatfs_format** will format the SPI FLASH with a brand new FileSystem. **WARNING: this sketch will erase all the data stored in the FLASH memory, including any data, python script!**.

The sketch is useful when you need to erase ALL the items to start a fresh setup. This sketch would also allow you to recover the board when the file system is corrupted.

The sketch **fatfs_format** and exemples here under are not compatible with the CircuitPython file system!**. If you need to share data between Arduino and CircuitPython then you should have a look to the example** fatfs_circuitpython **described here below.**

To execute the formatting sketch, just load the Arduino IDE and updload it on the Feather M0 board. Then open the serial monitor (at 115200 baud). You should see a message requiring a confirmation before formatting the Flash.

If you do not see the message, close the serial monitor, press the reset button then open the serial monitor again.



```
Adafruit SPI Flash FatFs Format Example
Flash chip JEDEC ID: 0x1401501
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
This sketch will ERASE ALL DATA on the flash chip and format it with a new filesystem!
Type OK (all caps) and press enter to continue.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Partitioning flash with 1 primary partition...
Partitioned flash!
Creating and formatting FAT filesystem (this takes ~60 seconds)...
Formatted flash!
Flash chip successfully formatted with new empty filesystem!
```

Crédit: AdaFruit Industries www.adafruit.com http://www.adafruit.com

Type in **OK** in the serial monitor and press the "send" button to confirm the format operation of the Flash memory. **It is necessary to type the OK in capital!**

Once done, the sketch would start to format the SPI Flash memoryt. The formating procedure need 1 minute to get complete. The sketch will display a message when done. Great, you have a drand new file system.

Error while formatting

If you can't get the Flash memory formated and receive the following error:

```
Adafruit SPI Flash FatFs Format Example
Flash chip JEDEC ID: 0x1401501
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
This sketch will ERASE ALL DATA on the flash chip and format it with a new filesystem!
Type OK (all caps) and press enter to continue.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Partitioning flash with 1 primary partition...
Couldn't read sector before performing write!
Error, f_fdisk failed with error code: 1
```

Then you will need to use an older library version (until a fixed version is released).

See this thread on the Adafruit forums https://forums.adafruit.com/viewtopic.php?f=57&t=128979&p=641983#p641983 .

Datalogging example

A common usage of the SPI Flash memory is the datalogging. The example **fatfs_datalogging** shows some datalogging/writing operation. Open the sketch into Arduino IDE then upload the Feather M0 board. Next, open the serial monitor (at 115200 baud) and you should see a message displayed every minutes when the sketch writes a new line in the SPI Flash file system.

See the content of the `loop()` function to understand how to write into a file:

```
// Open the datalogging file in write mode.  the FILE_WRITE mode will
// open the file for appending (it will add the data a the end
//  of the file).
File dataFile = fatfs.open(FILE_NAME, FILE_WRITE);
// Check if the file is open and write datas.
if (dataFile) {
  // Grab the data from sensors. In this sample
  // the data would be a random number.
  int reading = random(0,100);
  // Write a new line in the file.
  // The user can use the same functions as print function
  // sending data to the serial monitor.
  // EG: to write 2 CSV entries (coma separated):
  dataFile.print("Sensor #1");
  dataFile.print(",");
  dataFile.print(reading, DEC);
  dataFile.println();
  // The file must be closed at the end of writing operation.
  // This is the right way to ensure that data are writtebn
  // into the file.
  dataFile.close();
  Serial.println("New value written to the file!");
}
```

As you would do with the SD library for Arduino, you first need to create a **File** object by calling the **open** function with the filename and file access mode (**FILE_WRITE** mode appends data at the end of the file). **However** instead of calling the `open` global function, you have to call the `fatfs.open()` on a fatfs object created to access the file system on the SPI Flash (see the config values just behing the #define).

Once the file opened, calling the **print** and **println** method on the file object would write the data to the file. It is the exact same way than sending data over the serial monitor for sending text, numerical values and other types of data.

Just check twice that you closed the file system (otherwise you way lost data or even the complete file)!

Read and display file content

The example **fatfs_print_file** would open a file (the file `data.csv` per default, the file created with the **fatfs_datalogging** example) then it displays the file content in the serial monitor. Open the **fatfs_print_file** sketch and upload it to the Feather M0 board before opening the serial monitor (at 115200 baud).

You should see the content of the data.csv file (if the Flash Memory doesn't yet have the data.csv file, then run the datalogging example to create it).

Please, see the content of `setup()` function to understant how to read a file:

```
// Open the file in read only mode (check if opens succeed).
// The FILE_READ opens the file to read it.
File dataFile = fatfs.open(FILE_NAME, FILE_READ);
if (dataFile) {
  // The file is now open.
  // Display the content char by char until the
  // end of file.
  Serial.println("File open, content displayed here under:");
  while (dataFile.available()) {
    // Use the read() function to extract next char.
    // The readUntil or readString functions can also be use.
    // See the exemple fatfs_full_usage for more information.
    char c = dataFile.read();
    Serial.print(c);
  }
}
```

In the same way as datalogging example, you need to create a **File** object by calling the **open** method of the `fatfs` object.

This time, the mode used to access the file is **FILE_READ** indicating our intention to read the content of the file.

Once the file open in read mode, the **available** function let you know if some data are available in the file. Then, the **read** function read a byte from the file. The combination of those 2 functions allow us to create a read loop which check the availability of date before reading it (on byte at the time).

It also exists advanced read functions like those used in the **fatfs_full_usage** and explained indide the Arduino SD class documentation https://www.arduino.cc/en/reference/SD (the Flash SPI library implements the same functions).

Full Flash example

The **fatfs_full_usage** is a complete example demonstrating reading and writing operations on files. This example use all the library functions and advanced feature like file existence, folder creation, file wiping, etc.

Remember that SPI Flash library is designed to expose the same interface than Arduino's SD library https://www.arduino.cc/en/reference/SD . So the codes and samples storing data on SD card would would be easy to adapt to the SPI Flash library. Just create a `fatfs` object like the examples here upper. You will also have to use the `open` method on the object (instead of the global open function). Once the reference to the file object, all the functions and usages would be identifcal between the SPI Flash and Arduino's SD library!

Read and Write CircuitPython files

The example **fatfs_circuitpython** demonstrate how to read and write the files from the SPI Flash from CircuitPython file system. This means that you can execute CircuitPython script to store data in the CircuitPython file system, then use an Arduino sketch using this library to interact with those data.

Note: before running the exemple **fatfs_circuitpython** you **must** have loaded the CircuitPython on the board. see the Adafruit's M0 Express guide https://learn.adafruit.com/adafruit-feather-m0-express-designed-for-circuit-python-circuitpython/what-is-circuitpython%7CPlease, to initialize the CircuitPython file system in the SPI Flash. Once the CircuitPython loaded on the board, you can execute the sketch **fatfs_circuitpython**.

To execute the sketch, you have to load it inside Arduino IDE then upload it to the Feather M0 board. Then, you have to open a serial monitor a 115200 baud. You should see messages displayed when the sketch tries to read and write files on the Flask memory.

Specifically, the example looks for the files **boot.py** and **main.py** (since CircuitPython execute thos file when starting the board) to display their content. After, the sketch add a line at the end of the **data.txt** file available in the CircuitPython file system (the file is created if not yet existing).

When done, you can reload CircuitPython on the board to load and read the exécuté le croquis, vous pouvez recharger CircuitPython sur la carte pour lire le fichier **data.txt** directement depuis CircuitPython!

Let's have a loot to the sketch code to understant how to read and write files in CircuitPython. First, an instance of the **Adafruit_M0_Express_CircuitPython** class is created with the instance of the SPIFlash class (SPIFlash is used to access the Flash content):

```
#define FLASH_SS       SS1                    // SSP pin from Flash
#define FLASH_SPI_PORT SPI1                   // SPI port where the Flash is wired

Adafruit_SPIFlash flash(FLASH_SS, &FLASH_SPI_PORT);     // Use the hardware SPI bus

// Other pins can also be used for the SPI bus (software SPI)!
//Adafruit_SPIFlash flash(SCK1, MISO1, MOSI1, FLASH_SS);

// Finally, create an Adafruit_M0_Express_CircuitPython object to gain access
// to a SD alike interface. The Adafruit_M0_Express_CircuitPython would allow
// the sketch to access the CircuitPython file system stored inside the Flash.
Adafruit_M0_Express_CircuitPython pythonfs(flash);
```

By using the **Adafruit_M0_Express_CircuitPython** class, you get a "File System" objet type compatible with read/write operations over a CircuitPython file system. This point is important for the interoperability between CirctuitPython and Arduino. CircuitPython use a particular partitioning of the Flash which is not compatible simpler library (like those mentionned in another examples).

One the **Adafruit_M0_Express_CircuitPython** class instance created (instance named **pythonfs** in the sketch) you can interact with the file system like an Arduino's SD library https://www.arduino.cc/en/Reference/SD . You can open files in read/write mode, create folder, drop files and folders (and even more).

Here a sketch looking for the **boot.py** file and displaying its content on the screen (char by char):

```
  // Check the boot.py file existence THEN display it on the screen
  if (pythonfs.exists("boot.py")) {
    File bootPy = pythonfs.open("boot.py", FILE_READ);
    Serial.println("Display boot.py...");
    while (bootPy.available()) {
      char c = bootPy.read();
      Serial.print(c);
    }
    Serial.println();
  }
  else {
    Serial.println("No boot.py file...");
  }
```

Le file write operation is also very simple, the following sketch will add data to the **data.txt** file:

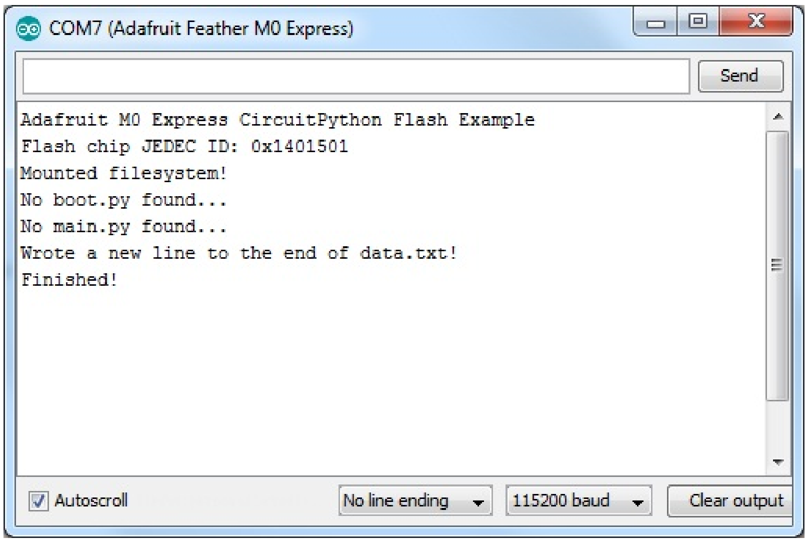
```
  // Create and add data in the file data.txt
  // then append a carriage return.
  // Later, the CircuitPython script will be able to
  // read the file content!
  File data = pythonfs.open("data.txt", FILE_WRITE);
  if (data) {
    // Add a new line of data:
    data.println("A great day to CircuitPython from our beloved Arduino sketch!");
    data.close();
    // See also the examples from the fatfs_full_usage
    // and fatfs_datalogging for mode information about
    // interaction with files.
    Serial.println("A new line was added to the data.txt file!");
  }
  else {
    Serial.println("Erreur, ne sais pas ouvrir le fichier en écriture!");
  }
```

Access to the SPI Flash

Arduino is not able to expose himself as a storage device (a "*mass storage*" device). Instead you will have to switch to CircuitPython to expose the SPI Flash as storage device. Here is the technique to use:

- Start the bootloader of the Express board. Drag and drop the last version of the **circuitpython** (the UF2 file).
- After a while, you should see a **CIRCUITPY** drive containing the file **boot_out.txt**. Great, the Circuit Python filesystem is initialized on the SPI Flash.
- Open the Arduino IDE and upload the **fatfs_circuitpython** sketch available in the Adafruit's SPI FLASH library. Open the serial console and start the sketch. Voila! the CircuitPython file system is properly mounted and the file **data.txt** created and initialized.

So, the Arduino Sketch did manipulated a file onto the SPI Flash owning the CircuitPython filesystem!

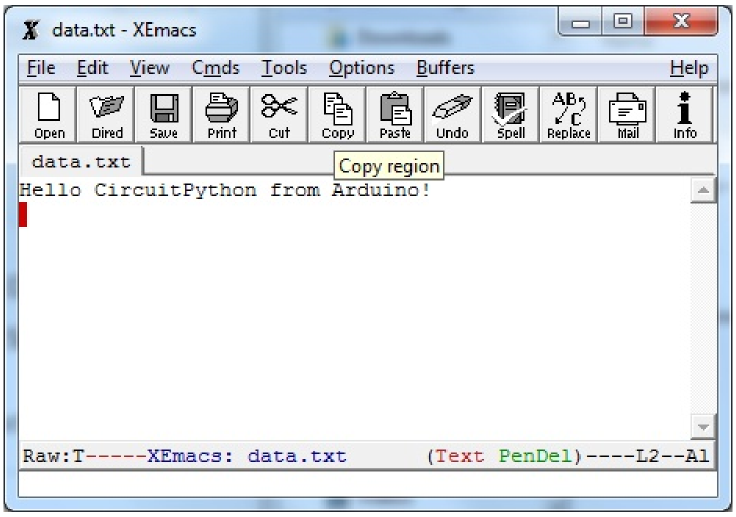


```
Adafruit M0 Express CircuitPython Flash Example
Flash chip JEDEC ID: 0x1401501
Mounted filesystem!
No boot.py found...
No main.py found...
Wrote a new line to the end of data.txt!
Finished!
```

Crédit: AdaFruit Industries www.adafruit.com *http://www.adafruit.com*

This time, we will open the file created by the Arduino sketch.

- Let plug again the board on the computer, restart the bootloader on the Express board --AND-- drag/drop the **circuitpython.uf2** on the drive **BOOT** made accessible by the bootloader. Great, CircuitPython is now installed (again) on the board.
- After a a while, the **CIRCUITPY** drive is made available by CircuitPyhton. This would expose the SPI Flash content as a Mass Storage device. You can now see the file **data.txt** created by our Arduino Sketch, Open it and read it's content :-) !



```
Hello CircuitPython from Arduino!
```

```
Raw:T-----XEmacs: data.txt       (Text PenDel)----L2--A1
```

Crédit: AdaFruit Industries www.adafruit.com *http://www.adafruit.com*

Once your datalogging sketch finish, you can simplify the procedure by copying the **CURRENT.UF2** from the **BOOT** drive to make "ready to use copy" of your sketch. You could now load the CircuitPython to access the CircuitPython file system and then switch back to you Arduino sketch by restoring the **CURRENT.UF2** on the Express Board!

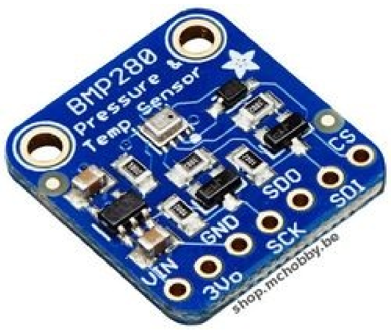
BMP280 sensor

Sommaire

About the BMP280 breakout

The BMP280 sensor can measure the atmospheric pressure and temperature (because the temperature also impact the sensor physics).

The BMP280 is a Bosch sensor upgraded from the BMP085/BMP180/BMP183 serie. This sensor is really great to make environmental or weather measurements. Best of all, it can be used over an I2C or a SPI bus!



It is one of the best sensor, it offer good accuracy for an affordable price. The accuracy is ±1 hPa for the pressure and ±1.0°C for the temperature. This sensor is not really made to measure the temperature but you can estimate its range with the BMP280.

As the pressure also change with the altitude, the sensor accuracy allows you to use the BMP280 to make an altimeter (with accuracy of ±1m at worste, about 0.25m in best conditions).

To ease the usage of this sensor, the SMD component is solder on a breakout board with some additional passive electronics. The board also bring a level shifter and 3V voltage regulator so it is save for 3V and 5V logic microcontrolers.

technical details

- Fiche technique du BMP280 https://df.mchobby.be/datasheet/bmp280.pdf (Bosch, pdf)
- Size: 19.2mm x 17.9mm x 2.9mm
- weight: 1.3 gr

Install the Library

Manual Installation

The Adafruit's BMP280 is provided with a library available on GitHub.

You can download and install the library manually by dowloading it from the BMP280 Repository.

Download the BMP280 lib from the repository
https://github.com/adafruit/Adafruit_BMP280_Library

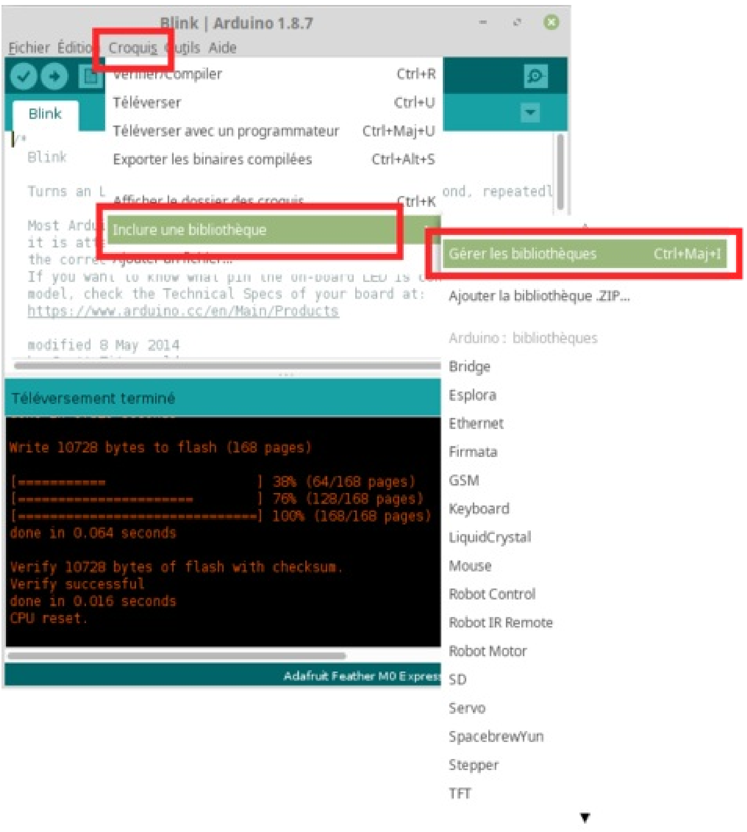
Note that BMP280 library rely on the **Adafruit_Sensor** library which declare an unified data structure to store data across all the Adafruit's Sensors Libraries.

Download the Unified Sensor lib from the repository
https://github.com/adafruit/Adafruit_Sensor

Easier Installation

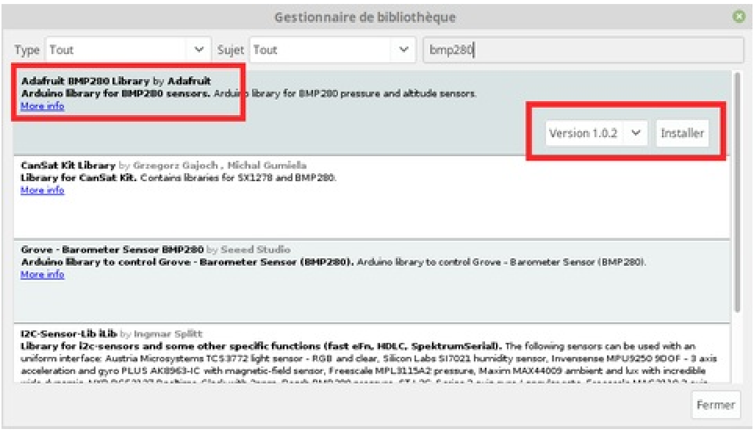
You can also use the "**Library Manager**" to ease the installation of the BMP280 library.

From the "**Sketch**" menu, select the sub-menu "**Include library**" --> "**Library Manager**" like shown on the picture here under.



```
Write 10728 bytes to flash (168 pages)

[==========                    ] 38% (64/168 pages)
[=====================         ] 76% (128/168 pages)
[==============================] 100% (168/168 pages)
done in 0.064 seconds

Verify 10728 bytes of flash with checksum.
Verify successful
done in 0.016 seconds
CPU reset.
```

In the library manager, key-in the value "**BMP280**" in the search box. Then click on the install button in the front of the **Adafruit BMP280 Library by Adafruit**.



Adafruit BMP280 Library by **Adafruit**
Arduino library for BMP280 sensors. Arduino library for BMP280 pressure and altitude sensors.
More info

CanSat Kit Library by Grzegorz Gajoch , Michal Gumiela
Library for CanSat Kit. Contains libraries for SX1278 and BMP280.
More info

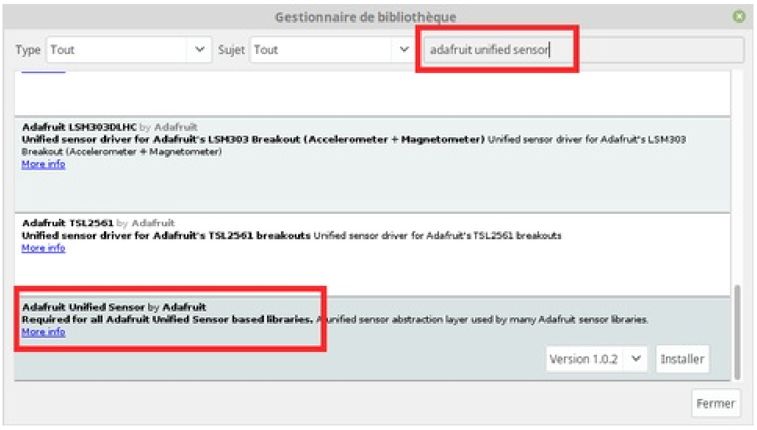
Grove - Barometer Sensor BMP280 by Seeed Studio
Arduino library to control Grove - Barometer Sensor (BMP280). Arduino library to control Grove - Barometer Sensor (BMP280).
More info

I2C-Sensor-Lib iLib by Ingmar Splitt
Library for i2c-sensors and some other specific functions (fast eFn, HDLC, SpektrumSerial). The following sensors can be used with an uniform interface: Austria Microsystems TCS3772 light sensor - RGB and clear, Silicon Labs SI7021 humidity sensor, Invensense MPU9250 9DOF - 3 axis acceleration and gyro PLUS AK8963-IC with magnetic-field sensor, Freescale MPL3115A2 pressure, Maxim MAX44009 ambient and lux with incredible

Great, the BMP280 library is now installed!

You must also install the unified sensor library! If not yet done, proceed the installation of the library as described here under.

From "**Library Manager**", search for the "**Adafruit Unified Sensor**" like shown on the picture here under.



adafruit unified sensor

Adafruit LSM303DLHC by Adafruit
Unified sensor driver for Adafruit's LSM303 Breakout (Accelerometer + Magnetometer) Unified sensor driver for Adafruit's LSM303 Breakout (Accelerometer + Magnetometer)
More info

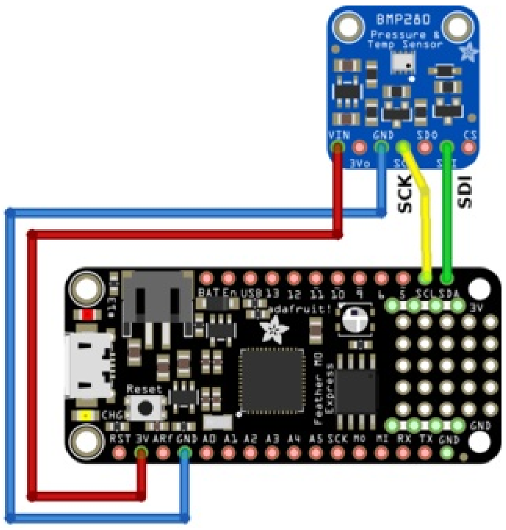
Adafruit TSL2561 by Adafruit
Unified sensor driver for Adafruit's TSL2561 breakouts Unified sensor driver for Adafruit's TSL2561 breakouts
More info

Adafruit Unified Sensor by **Adafruit**
Required for all Adafruit Unified Sensor based libraries. A unified sensor abstraction layer used by many Adafruit sensor libraries.
More info

Then install it.

Wiring the sensor

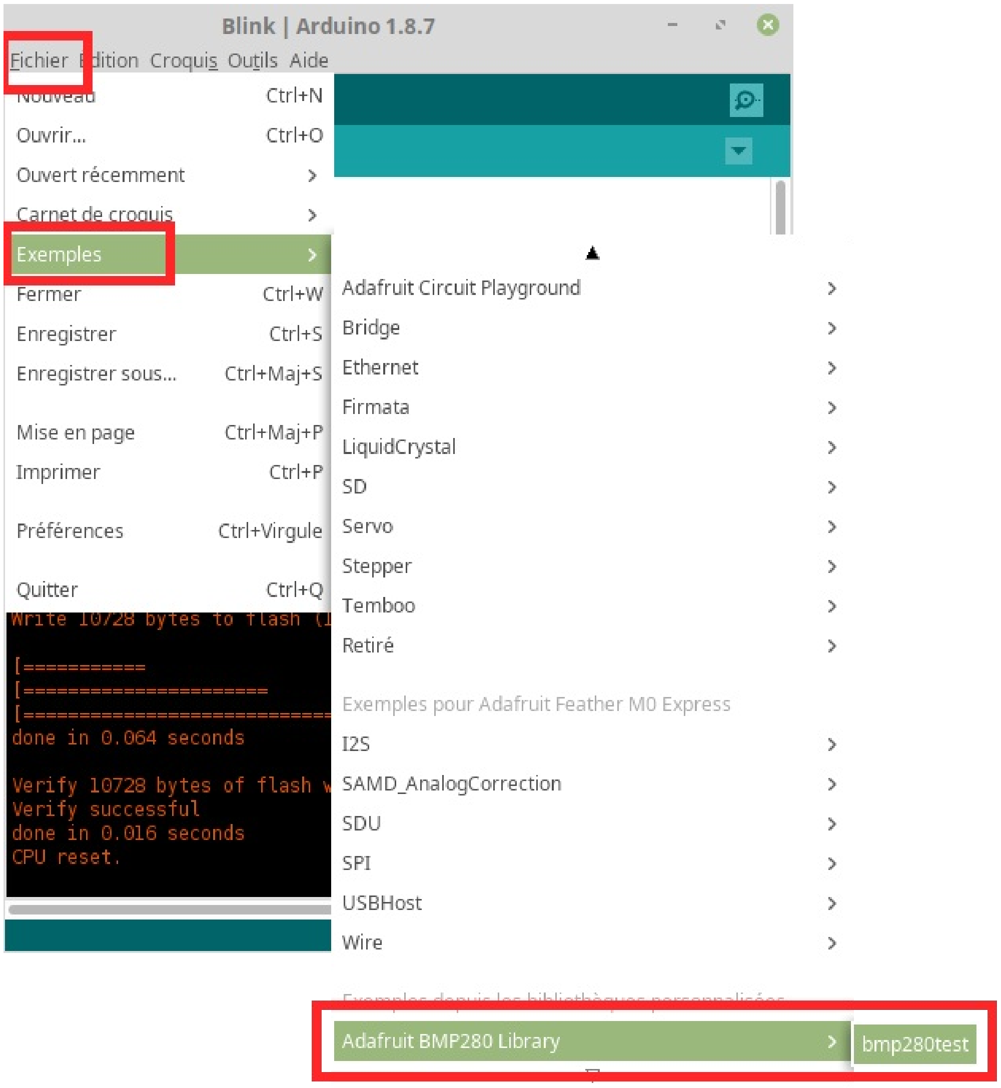
The BMP280 is wired on the I2C bus of the Feather.



Testing the sensor

The sensor can be easily tested with the Adafruit Example code (installed with the library).

The sample code is available through the **File** menu under the sub-menu **Example** --> **Adafruit BMP280 Library** --> **BMP280test**.

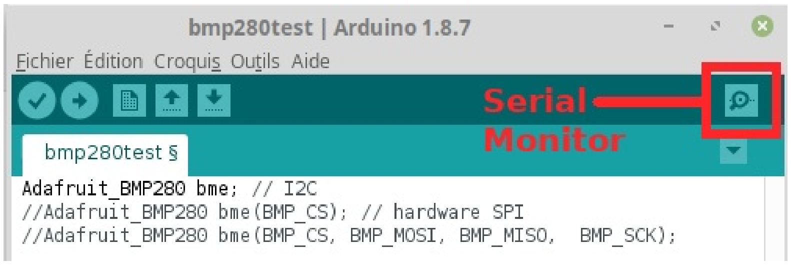


```
Write 10728 bytes to flash (1
[===========
[=======================
[==============================
done in 0.064 seconds

Verify 10728 bytes of flash w
Verify successful
done in 0.016 seconds
CPU reset.
```

The content of the example code is displayed here under (and reduced to the minimum lines for better reading).

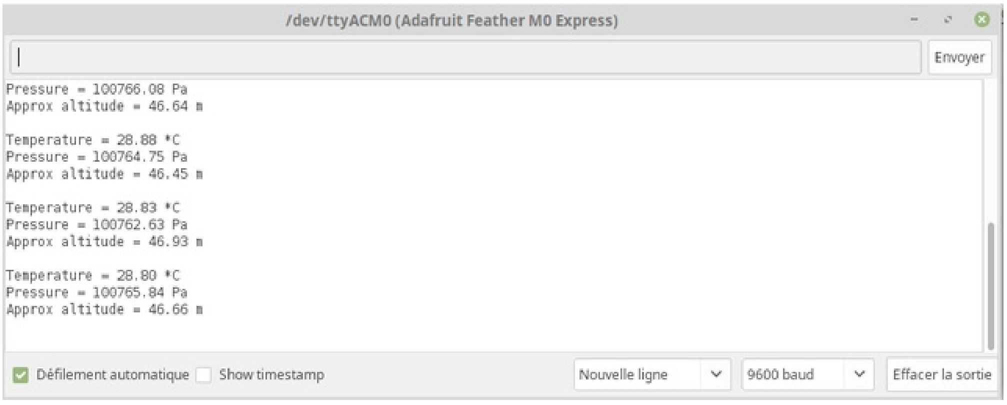
```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>


Adafruit_BMP280 bme; // I2C

void setup() {
  Serial.begin(9600);
  Serial.println(F("BMP280 test"));

  if (!bme.begin()) {
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");
    while (1);
  }
}

void loop() {
    Serial.print("Temperature = ");
    Serial.print(bme.readTemperature());
    Serial.println(" *C");

    Serial.print("Pressure = ");
    Serial.print(bme.readPressure());
    Serial.println(" Pa");

    Serial.print("Approx altitude = ");
    // 1013.25 is the pressure at sea level. It should be ajusted
    // with your local forecast for a corect evaluation of altitude
    Serial.print(bme.readAltitude(1013.25));
    Serial.println(" m");

    Serial.println();
    delay(2000);
}
```

Compile and upload to sketch the board.


bmp280test | Arduino 1.8.7
Fichier Édition Croquis Outils Aide
Serial Monitor
bmp280test §

```
Adafruit_BMP280 bme; // I2C
//Adafruit_BMP280 bme(BMP_CS); // hardware SPI
//Adafruit_BMP280 bme(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);
```

Then open the serial monitor (configured at 9600 baud) and you should see the following on the screen.



```
Pressure = 100766.08 Pa
Approx altitude = 46.64 m

Temperature = 28.88 *C
Pressure = 100764.75 Pa
Approx altitude = 46.45 m

Temperature = 28.83 *C
Pressure = 100762.63 Pa
Approx altitude = 46.93 m

Temperature = 28.80 *C
Pressure = 100765.84 Pa
Approx altitude = 46.66 m
```

The sketch display pressure and other parameter every 2 seconds.

Pressure and altitude

This section could also be named "the Weather Station OR the Altimeter".

Depending on the use case (weather station or altimeter", the way of using the BMP280 is slightly different.

About pressure reading

The pressure is returned in Pascals (an unit from International System of Units https://en.wikipedia.org/wiki/International_System_of_Units). 100 Pascals = 1 hPa = 1 millibar. The barometric pressure is often using the millibar or mm of mercury as unit. Just note that 1 pascal = 0.00750062 mm of mercury mercure.

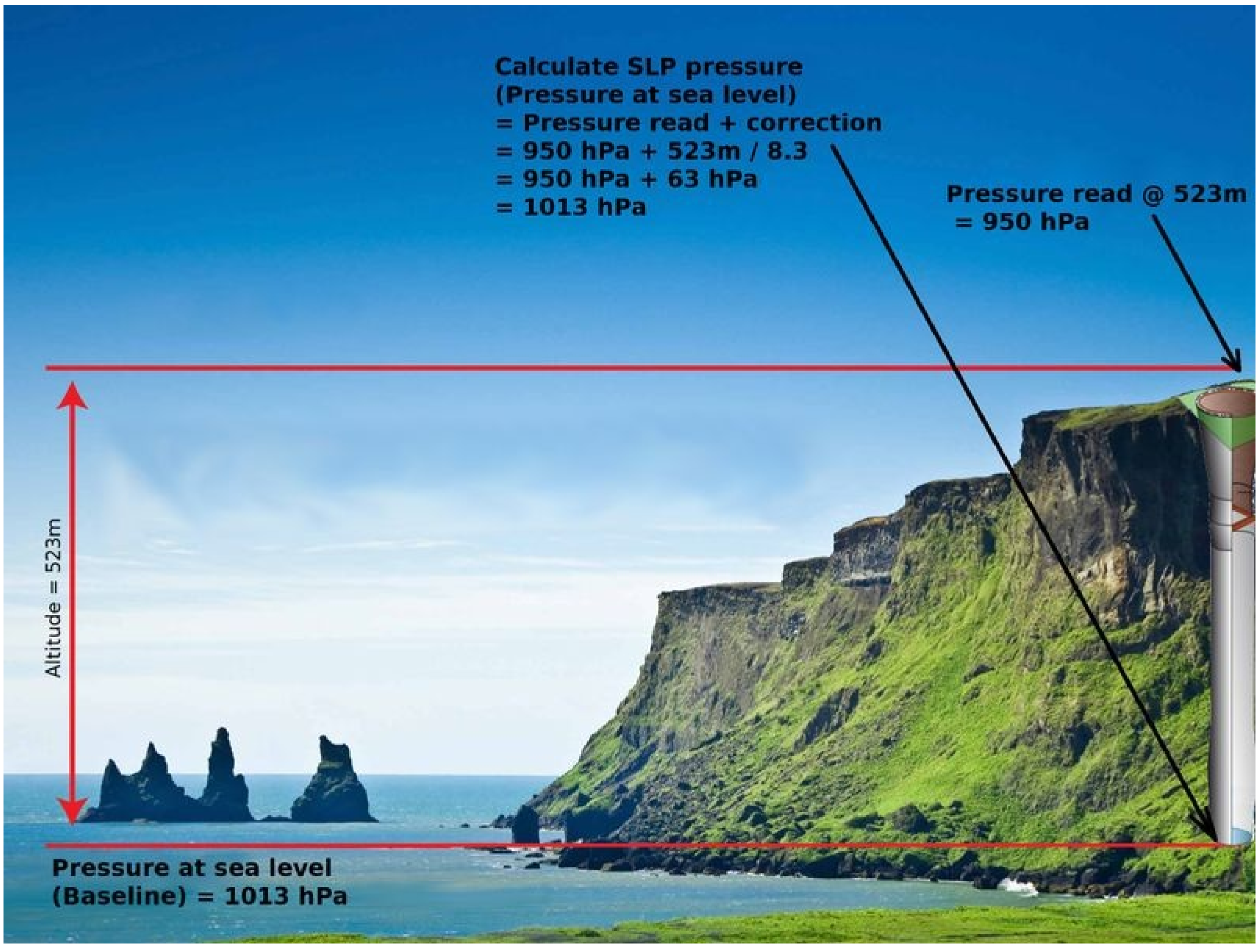
SLP: Sea Level Pressure

You can also calculate the altitude from the pressure. However, to proprely measure the altitude, you need to know the pressure at the sea level (hPa, pressure that change every day)!. The BMP280 is really precise, however it may be difficult to have precise evaluation of the altitude if you don't know the pressure at the sea level (the pressure of day at the sea level).

Pressure, SLP and altitude

SLP means Sea Level Pressure (also knwon as PNM pressure). Most of the advanced the weather station does transform the current pressure to normalized SLP pressure before displaying it. If all pressure in the country are expressed as SLP pressure then it is more easy to determine the wind (and cloud) movements across the country, from higher pressure to lower pressure.

Here is a small picture that shows the relation between normalized SLP pressure, altitude and pressure given by a sensor (like the BMP280).



Let's say that we readed the pressure of 950 hPa at our house with the BMP280. The house sit at 523m of altitude.

If we want to know the SLP pressure, it is like sink a well under the house (down to the sea level). At the bottom of the well, we have more air over our head so the pressure will be greater than 950 hPa. In real world, we do not have to sink a well, we do know the altitude of our house (at 523m) so we can estimate the correction to apply (corresponding to a column of 523m of air). So, from the 950hPa read at house, we can calculate the corresponding pressure at sea level (SLP pressure). Together with the other SLP values, we can estimate the movement of clouds :-) .

In real world, we cannot sink a well under the house, neither know exact altitude of the house.

Here the steps to follow with the BMP280 to estimate the altitude of the house:

1. Find the hPa (or mmbar) pressure at Sea Level on a WebSite
2. Use that value as baseline (don't hesitate to multiply it by 100)
3. Use the sensor and the BMP280 library to read the altitude.
4. Calculate the correction value (for the air's colonne) in hPa = height-in-meter / 8.3

When you have to "correction" value you don't have to care anymore about the baseline.

Indeed, we can read the current pressure (without caring about the baseline) THEN we add the "correction" value --> Tadaaa! We have the SLP pressure (the same than displayed on Reference Weather Station).

```
# mean pressure at the sea level (not critical for SLP pressure)
p.baseline = 101325
# SLP pressure
p = bmp280.pressure + compensation
```

where `p` would contains pressure normalized at sea level (so the SLP value).

The downside of the "mean pressure value" approach (101325 Pa) is that you cannot estimate the altitude with accuracy. However, by updating the baseline value every day with the "day's pressure" at sea level (see on Internet broadcast website) then the altitude calculation would be fairly precise.

Just remind:

- **If you plan to do a Weather station** then you have to care about the correction in order calculate the normalized value at sea level. Altitude is not useful since the weather station doesn't move.
- **if you need a flying sensor** in a rocket then you have to care about the baseline value (sea level pressure of the day) for have a accurate measure the the altitude.

The sea level pressure change every day!

The usual pressure at sea level is about 1013.25 mbar (or 1013.25 hPa or 101325 Pa).

However, this value depends on the weather conditions and quantity of steam in the air.

By example, today the pressure is 1002.00 hPa at the Belgian's sea level.

this value is critical if you want to evaluate the altitude of the sensor.. It is also important to know the current altitude if you plan to calculate the "correction" for the normalized SLP pressure.

So I have fixed the baseline as follow before reading the altitude:

```
Serial.print(bme.readAltitude(1002.00));
Serial.println(" m");
```

It is quite easy to know the current sea level pressure by using an Internet Weather Broadcast like this link to meteobelgique.be http://www.meteobelgique.be/observations/temps-reel/stations-meteo.html

The sensor doesn't give the right altitude!

My sensor does indicates an altitude of 189m whereas the reference weather station (next to house) is known to be at 120m height (at the top of the tower)! What's wrong with the sensor?

The altitude is deduced from the difference of local pressure and pressure at the sea level.

If you want to obtain an accurate altitude value with the BMP280 then you need to know the pressure at the sea level (*the baseline*) with precision.

Once the baseline value corrected, you will have the correct altitude:

```
bme.readAltitude(1002.00)
```

The sensor would return the right altitude (104m) for the sensor which is quite good since we are not in the tower (like the reference weather station).

The pressure is not correct atmosphérique semble incorrecte!

My sensor returned a pressure of 98909 pascal (so 989.09 hPa) whereas the reference weather station does mentino 1002 hPa!

The sensor value is right, it just not apply the correction to indicates the Normalized SLP pressure (equivalent pressure at the sea level). The reference reference weather station does applies this correction for yo (so they displays normalized SLP).

Let's do the correction on the value...

First: read the preceding point, from it we know :

- that we have to set the baseline to the current pressure at day's sea level pressure (baseline=100200)
- once done, we can use the sensor to calculate the current altitude of the sensor (in our case, it is 104 m)
- the pressure decrease of 1hPa every time we increase the altitude of 8.3m .

Next, on the reference Weather station:

The weather station does normalize the atmospheric pressure calculate local pressure at the Sea Level altitude (called SLP for "Sea Level Pressure" also named "PNM" for *Pression Niveau Mer*).

This means that reference weather station applies the correction (compensating) to the read value.

For the reference station next to house, its height is 120m. so the correction adds the air column of 120m height over the sensor value to get the normalized SLP pressure. So the correction is evaluated to (120 / 8.3) hPa = 14.45 hPa.

Let's apply the same principle to our BMP280 readings:

In our case, we know that the altitude of the BMP280 sensor is 104m. Remind, the pressure is reduced of 1hPa every time our altitude increase of 8.3m.

For 104m, the air column until the sea level correspond to 104 / 8.3 = 12.53 hPa additional pressure.

As the sensor give the 989.09 hPavalue, the correction to have the Normalized SLP is 989.09 + 12.53 = 1001.62 hPa. Great! it is almost the same value than reference weather station next to home (to remind, it communicates 1002 hPa).

Please note that the today's pressure is 1002 hPa and our normalized pressure is also 1002 hPa. This is rare situation and means that the air will not flow in any way between our location and the sea.

TMP36 sensor

Sommaire

About the TMP36 Sensor

The TMP36 is the reference analogue temperature sensor in the Arduino world. It is affordable, small et power efficient. For sure there are better temperature sensors but this one will do the job for almost nothing :-)

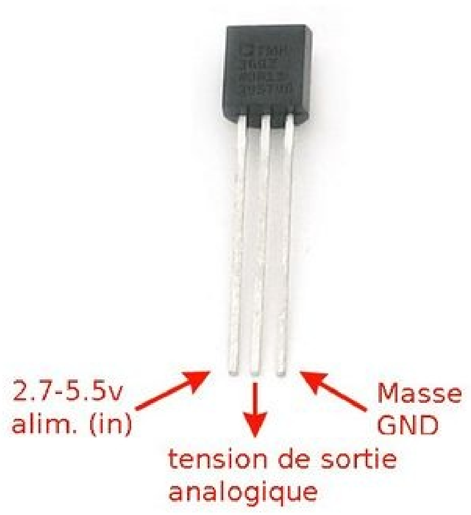
This sensor is very common and easy to use. It is also one of the components of the ARDX development kit https://shop.mchobby.be/product.php?id_product=11 .

With the TMP36, it is possible to measure a temperature from -50°C to 125°C, the output voltage is proportional to the temperature.

Don't be fooled, the TMP36 looks like a transistor (eg: P2N2222AG) but it isn't a transistor. It is a complex sensor within a package identical to a transistor.

There are 3 pins on the TMP36.

- the **ground** (on the left),
- the **output** signal (center position),
- the **+5 volts** (on the right)



The sensor output signal does output 10 millivolts per degree (with 500mV offset for temperature under 0°C).

Eg:

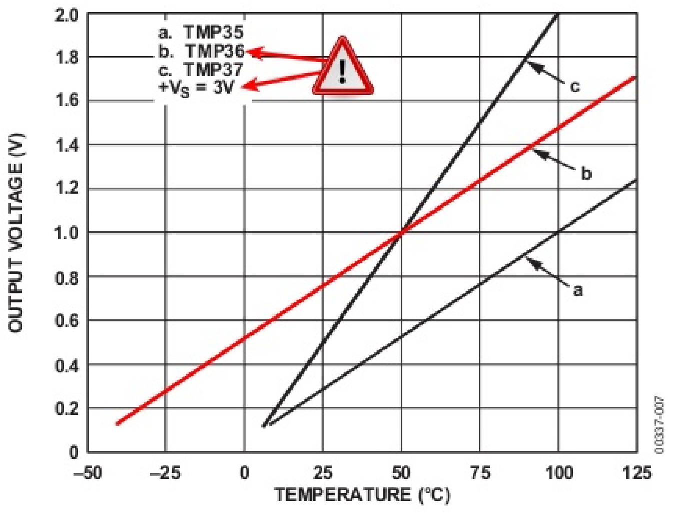
- 25° C --> output = 750 mV
- 0° C --> output = 500mV

Technical detail

- Analog output (see graphics)
- Temperature range: from -50°C to 125°C
- Power supply range: 2.7 to 5.5v
- TMP36 datasheet http://www.analog.com/en/mems-sensors/digital-temperature-sensors/tmp36/products/product.html (*analog.com*, html)

How to measure the temperature

It will be necessary to convert the analogue voltage intro degree. As the TMP36 can also measure negative temperature, the 0 degree Celcius is placed at 500 mV offset. So, any voltage under 0.5 Volt is a negative temperature.



Here is the formula to use with a TMP36 powered at 3.3v:

```
Temp in °C = ( output_voltage_in_mV - 500) / 10
```

So, if we do have an output voltage of exactly 1 Volt (1000 mV) then the temperature would be

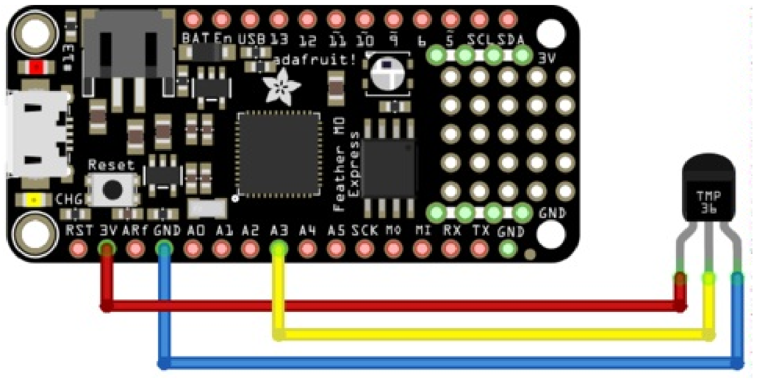
```
temp = (1000 - 500)/10
```

So 50 Celcius degrees.

Wiring

To use the TMP36, connect:

- The pin 1 (on the left) to a power source (3.3V),
- The pin 3 (the the right droite) to the ground/GND.
- The pin 2 (middle one) to the A3 analogue input.



Feather M0 Express
adafruit!
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND
BAT En USB 13 12 11 10 9 6 5 SCL SDA
Reset
CHG
TMP 36

The TMP36 output voltage would range from 0V @ -50°C to 1.75V @ 125°C. So no risk for our 3V based microcontroler.

Testing the sensor

In the both case show here under, the measured temperature would be identical.

However, if your project does need a high resolution analog reads then it may be appropriate to explore the "High Resolution Reading" example.

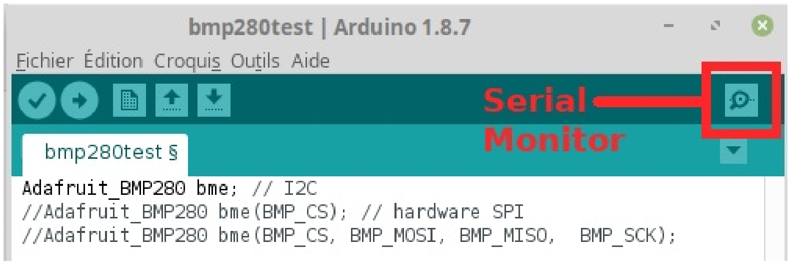
By default, the Arduino's `analogRead()` use a 10 bit coding. So the range of possible value return by `analogRead()` is 0 to 1024 (for 0 to 3.3v). This means that the accuracy of reading is 3.3 / 1024 = 0.0032 Volts, so 3.2 mV.

As the M0 does have an ADC (*Analog-to-Digital Converter*) with a precision of 12 bits, we could also use the `analogReadResolution( 12 )` to upgrade the `analogRead()` resolution to 12 bits. In such case, the range of possible value return by `analogRead()` is 0 to 4095 (for 0 to 3.3v). As we have a real 12bit ADC, we can rely on that accuracy (it is not a 10 bits ADC storing the data into a 12 bits integer). with 12bits we have an reading accuracy of 3.3 / 4095 = 0.000805 Volts, so 0.805 mV.

Low resolution reading

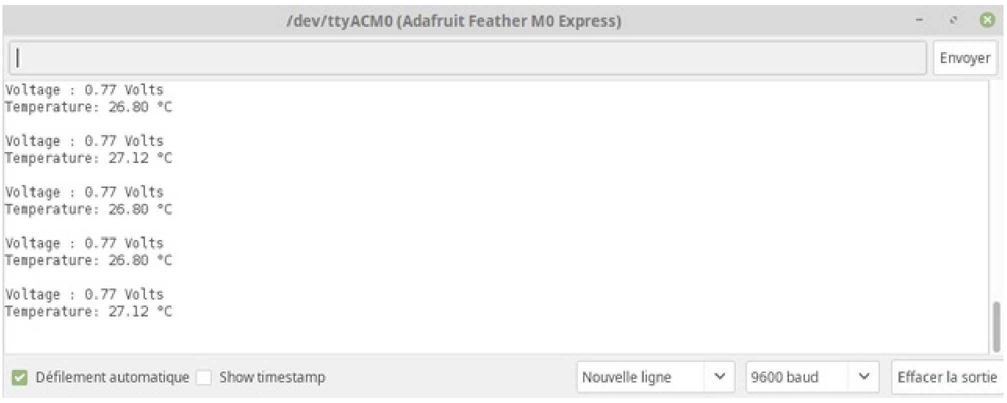
```
// where is wired the TMP36
const int temperaturePin = A3; // analogue input

// Executed once when starting the microcontroler
void setup() {
   // start serial connexion with the computer
   Serial.begin(9600);
}

// Executed again and again
void loop() {
   // read the voltage of TMP36
   float voltage = getVoltage(temperaturePin);
   Serial.print( "Voltage : " );
   Serial.print( voltage );
   Serial.println( " Volts" );
   // convert voltage to temperature
   //   Degrees = (voltage - 500mV) multiplied by 100
   float temperature = (voltage - .5) *100;
   Serial.print( "Temperature: " );
   Serial.print(temperature);
   Serial.println( " °C" );
   Serial.println( " " );
   delay(1000); // wait 1 second
}

/*
 * getVoltage() - return the voltage of an analog pin
 */
float getVoltage(int pin){
   // AS the sketch does not call the analogReadResolution()
   //    function to change the analog reading resolution
   // THEN Arduino use the defaut 12 bits resolution!
   // Under 12 bits resolution, the analogRead() returns
   //    a value between 0 & 1024.
   //
   // Convert digital value between 0 & 1024 to
   //    voltage between 0 & 3.3 volts.
   //    (each unit equal 3.3 / 1024 = 3.2 millivolts)
   return (analogRead(pin) * .0032);
}
```

Once the sketch uploaded to the board, you can start the Serial Monitor


bmp280test | Arduino 1.8.7
Fichier Édition Croquis Outils Aide
Serial Monitor
bmp280test §

```
Adafruit_BMP280 bme; // I2C
//Adafruit_BMP280 bme(BMP_CS); // hardware SPI
//Adafruit_BMP280 bme(BMP_CS, BMP_MOSI, BMP_MISO,  BMP_SCK);
```

Which produce the following result on the Serial Monitor



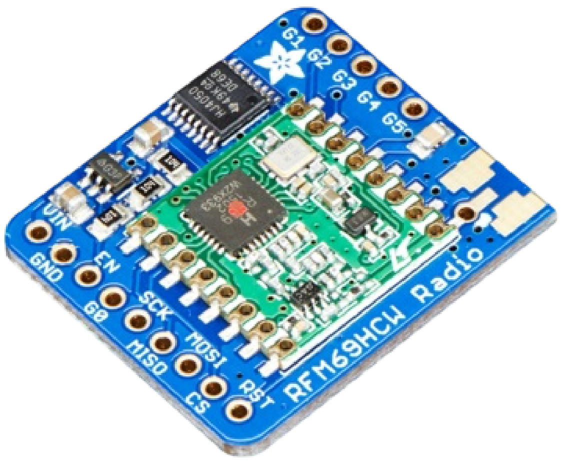
```
Voltage : 0.77 Volts
Temperature: 26.80 °C

Voltage : 0.77 Volts
Temperature: 27.12 °C

Voltage : 0.77 Volts
Temperature: 26.80 °C

Voltage : 0.77 Volts
Temperature: 26.80 °C

Voltage : 0.77 Volts
Temperature: 27.12 °C
```

RFM69HCW radio

Sommaire

RFM69 Radio Module

Before starting make sure you have your Feather and Arduino working properly with basic functionalities. This will make this part more easier and you can upgrade your project to radio transmission.



Both RFM69 and RFM9x LoRa breakouts have the exact same pinouts! And they exists in 900 MHz or 433 MHz flavor.

- The silkscreen identify the RFM69HCW -OR- LoRa
- The 900 MHz modules have a green or blue dot on top.
- The 433 Mhz modules have **a red dot** on top.

The sub-GHz radio transmission does have lower throughput so it is not made to stream audio or video! The sub-GHz is suited for small packets of data. The data rate is adjustable but its common to stick to around 19.2 Kbit per second. Lower is the rate and better woud be the transmissions.

To use such modules you will need both of them! The radios must be matched in frequency (eg: 433 MHz & 433 MHz will match, 433 MHz & 900 MHz will not match).

The both module must use the same encoding schemes. You cannot use a RFM69 900 MHz packet radio together with a RFM9x packet radio (LoRa).

In Belgium, you cannot use the RFM69 900 Mhz without having the appropriate license, see this "IBTP Frequency Plan" link for more details https://www.bipt.be/en/operators/radio/frequency-management/frequency-plan .

According to the same "IBTP Frequency Plan https://www.bipt.be/en/operators/radio/frequency-management/frequency-plan " the RFM69HCW should be used between 430-440 MHz.

Raw vs Packet Transmission

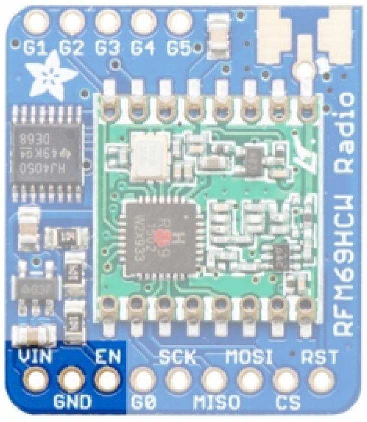
The SX1231 module used on the RFM69 breakout board can be used in 'raw Rx/Tx' where it modulates incoming bits (from pin #2) and sends them on the radio. In 'raw Rx/Tx' there is no error correction and no addressing. This mode is weak and error prone so it will not be covered.

Packet mode will be suited for almost 99% of use cases. When packetized, the code can setup a recipient for the data, ensure error correction (*data transmitted correctly*), automatic retries on transmission error and acknowledgement when the packet is delivered. In packet mode, you got a reliable data pipe, transparent communication without getting care about the complex details of data transmission over radio frequencies.

With a SX1231 module, the complexity is reduced to 4 main characteristics more easy to handle:

- the frequency to use
- the power level to use
- the encryption key to use
- the appropriate antenna (depending on the expected transmission range)

Power Pins

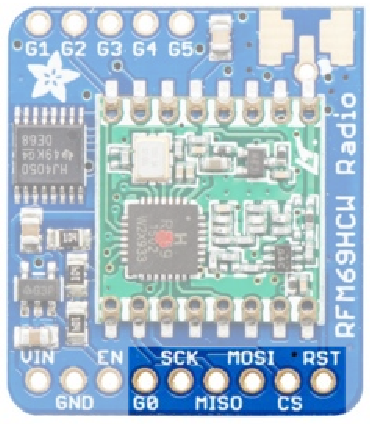


- **GND**: Common ground between logic and power.
- **Vin**: Power In 3.3 to 6V. The board regulated it to 3.3V. **Be sure you can supply up to 150mA** on this pin since Radio Emitting can reach this current level.
- **EN**: Enbable Pin used to switch off the Power supply (and the radio module). EN pin use a pull-up resistor to maintain the regulator enabled, just set it to LOW to switch off the radio.

SPI Interface Pins

Those pins are use to communicate with the host micro controller. The standard SPI bus pins are MISO, MOSI, CLK and ChipSelect.

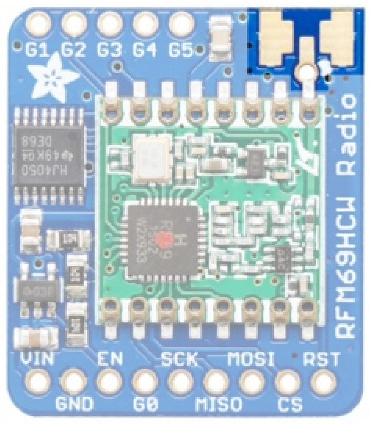
This radio breakout also expose the radio RESET pin and radio INTERRUPT pin (named G0) to offer a better control over the Radio communication.



All the pins (except extra GPIOs) are wired thought a level shifter. This means that logical level (3.3 or 5V) will be compatible with the voltage applied on Vin!

- **MISO**: Master In Slave Out. This pin is used to send data from the radio (the slave) to the micro controller (the master).
- **MOSI**: Master Out Slave In. This pin is used by the micro controller to send data to the radio.
- **SCK**: Clock signal used to synchronise bits exchanges between the Master and the Slave.
- **CS**: This is the **C**hip **S**elect. Place it to LOW level to initiate transaction with the radio module. The CS pin make sense when several slaves (eg: radio module + LCD display) are present on the SPI bus.
- **RST**: The **reset** pin of the radio module. Set it to LOW to reset the radio module.
- **G0**: This is the GPIO 0 pin of the radio module. This pin has a special interruption function attached to it because it act as **IRQ** (Interrupt Request) pin. The radio module manipulate it to send to notification toward the micro controller. This pin is in 3.3V logic.

Antenna spot



There are 3 ways to connect an antenna on this spot.

The cheapest is the wire antenna (a simple dipole) and the best option is the SMA connector.

An antenna is ABSOLUTELY REQUIRED to allow any data transmission. No communication possible (even 1 meter) without antenna.

For testing purpose, we suggest to twist a wire inside the antenna hole (take care to not touch the ground spots with the wire).

Later on, you would focus on antenna choice:

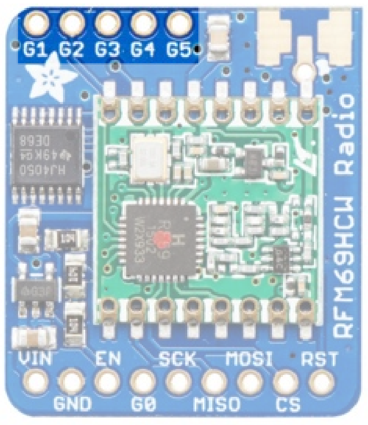
- A wire inside the antenna hole (said a *wire dipole*).
- A µFl connector https://shop.mchobby.be/product.php?id_product=1418 to plug antenna.
- A PCB SMA connector https://shop.mchobby.be/product.php?id_product=1419 also to plug other kind of antenna.

A µFl connector (also named uFl) is looking to this:

A PCB SMA Connector is looking to this:

The CanSat tutorials also have a section dedicated to the Antenna and alternative communication devices

Extra GPIOs Pins



The radio module also feature 5 additionnal GPIOs (from G1 to G5).

As they are usually free for use (not used for notification or radio functions) you can control them for the purpose of your project.

Those pins are **3.3V logic without level shifting**.

RFM69HCW Testing

Sommaire

Forewords

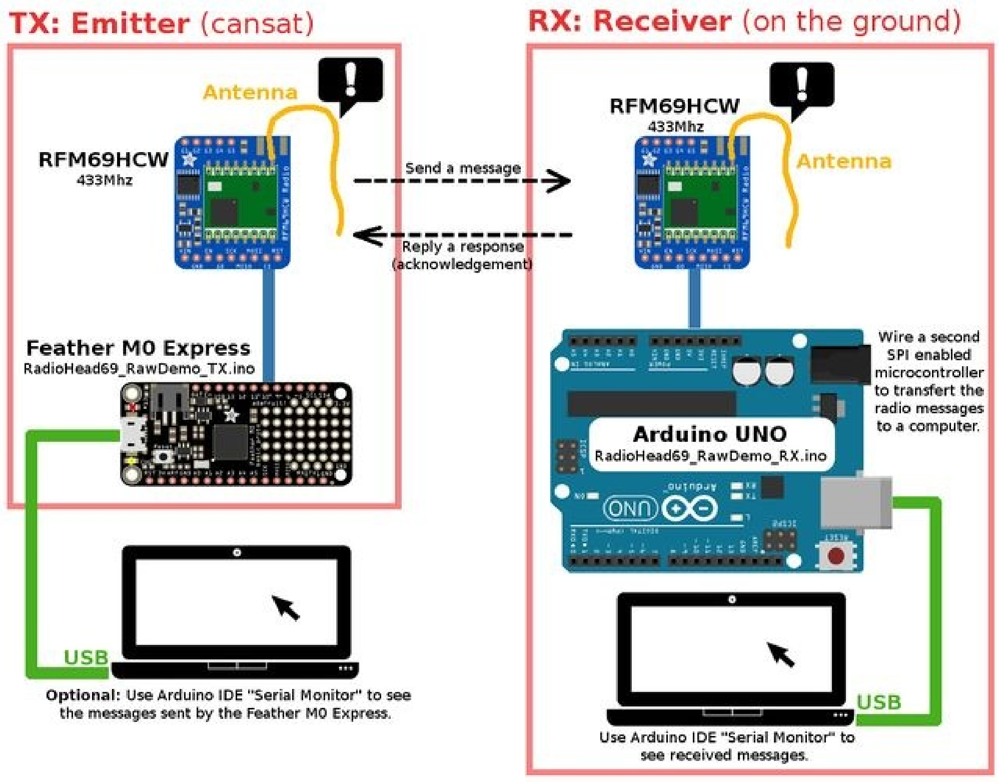
It is now time to establish a communication between:

- a **Data Emitter** made with a Feather M0 Express + RFM69HCW-433MHz.
- a **Data Receiver** made with the second RFM69HCW that should be linked to a second micro controller.

As the kit contains only one micro controller (the Feather M0 Express), we will use the very common **Arduino UNO** (not included) as micro controller for the **Data Receiver**.

In this simple exemple:

1. The **Data Emitter** will send a message and wait 500ms for a response.
2. The **Data Receiver** will receive the message.
3. The **Data Receiver** will send a reply.


TX: Emitter (cansat)
RX: Receiver (on the ground)
Antenna
RFM69HCW
433Mhz
Send a message
Reply a response (acknowledgement)
RFM69HCW
433Mhz
Antenna
Feather M0 Express
RadioHead69_RawDemo_TX.ino
Arduino UNO
RadioHead69_RawDemo_RX.ino
Wire a second SPI enabled microcontroller to transfert the radio messages to a computer.
USB
Optional: Use Arduino IDE "Serial Monitor" to see the messages sent by the Feather M0 Express.
USB
Use Arduino IDE "Serial Monitor" to see received messages.

As we will see, there are 2 key items will be highlighted:

1. The frequency must be identical in the emitter and receiver (eg: 433.0 MHz in this example).
2. The encryption key must be identical on the both side.

Installing the RadioHead library

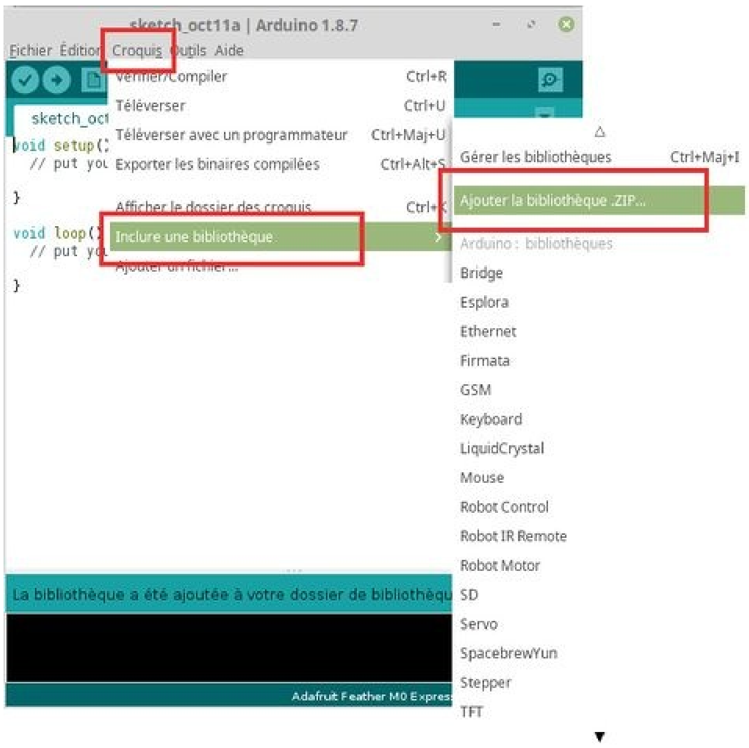
If not done yet, we will have to install the RadioHead library in Arduino IDE.

That library support lot of RFM modules including our RFM69HCW.

Adafruit did fork the RadioHead library https://github.com/adafruit/RadioHead and add some useful sample, so we will install the small|Adafruit's RadioHead forked library.

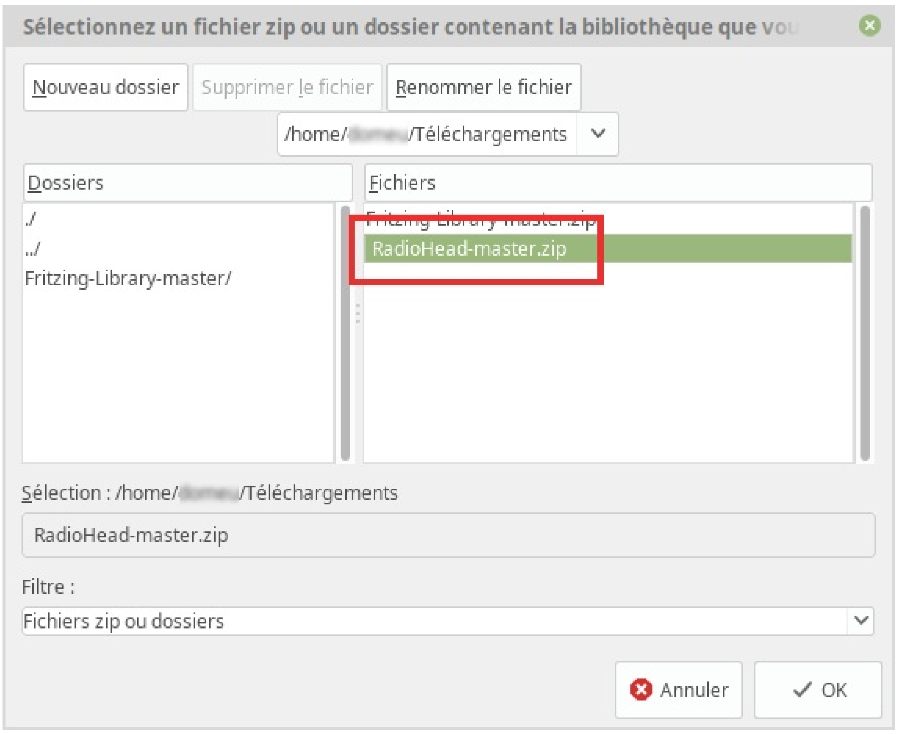
Download RadioHead forked library
https://github.com/adafruit/RadioHead/archive/master.zip

For easy install, you can run Arduino IDE and open the menu "**Sketch -> Add a .ZIP library...**"


sketch_oct11a | Arduino 1.8.7
Fichier Édition Croquis Outils Aide
Vérifier/Compiler Ctrl+R
Téléverser Ctrl+U
Téléverser avec un programmateur Ctrl+Maj+U
Exporter les binaires compilées Ctrl+Alt+S
Afficher le dossier des croquis Ctrl+K
Inclure une bibliothèque
Ajouter un fichier...
Gérer les bibliothèques Ctrl+Maj+I
Ajouter la bibliothèque .ZIP...
Arduino : bibliothèques
Bridge
Esplora
Ethernet
Firmata
GSM
Keyboard
LiquidCrystal
Mouse
Robot Control
Robot IR Remote
Robot Motor
SD
Servo
SpacebrewYun
Stepper
TFT
La bibliothèque a été ajoutée à votre dossier de bibliothèqu
Adafruit Feather M0 Expres

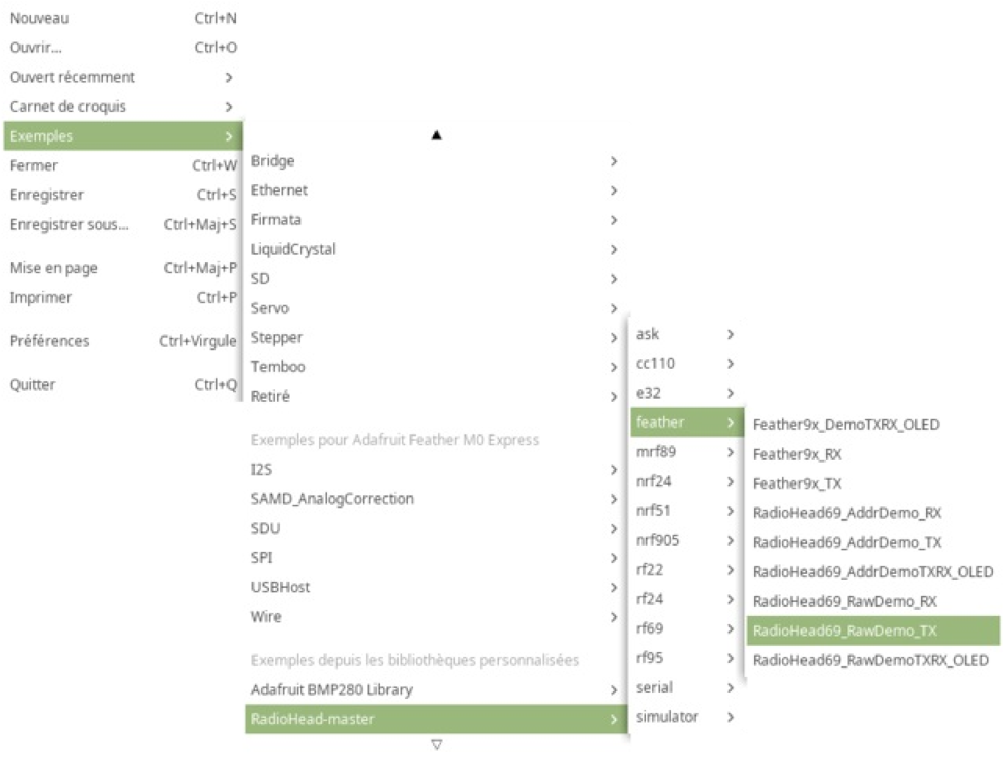
```
void setup(
  // put you
}

void loop()
  // put yo
}
```

Then pick-up the downloaded RadioHead ZIP file.



A good idea would be to rename the RadioHead-master.zip to RadioHead.zip before adding it to Arduino IDE.

Once installed the RFM69 examples are available from the menu "**File -> Examples**".



We will focus our interest in the following examples:

- File -> Examples -> RadioHead -> Feather -> RadioHead69_RawDemo_RX
- File -> Examples -> RadioHead -> Feather -> RadioHead69_RawDemo_TX

About Antennas

The RFM69HCW will not work without antenna, even at 1m distance of each other.

For this example, a simple wire twisted in the antenna hole will do a great job for testing.

Please wait before soldering the wire inside the antenna hole!. The antenna hole can be populated with:

- a simple wire
- a µFl SMT antenna connector https://shop.mchobby.be/product.php?id_product=1418 where you could plug various kind of antenna.
- a PCB SMA Connector https://shop.mchobby.be/product.php?id_product=1419 where you could plus various kind of antenna.

The antenna design is a key feature to ensure a reliable communication over a long distance.

A µFl connector (also named uFl) is looking to this:

A PCB SMA Connector is looking to this:

Frequency, Encryption & Power

To make the module communicating together:

- The module must be identical. You cannot mix them.
- The tuned frequency must be identical.
- The encryption key must be identical.

Tuned frequency

The tuned frequency is declared with a line like this:

```
#define RF69_FREQ 433.0

...

if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("setFrequency failed");
}
```

where the tuned frequency is declared with the constant **RF69_FREQ**.

Use the frequency assigned to your team by the instructor.

In packet radio, several teams can share the same frequency if they use distinct encryption key.

Like TCP (from TCP/IP network), the packet radio is able to detect packet colission try to recover from it.

However, more teams share the same frequency, more collision we have.

Encyption Key

The module encrypts the data with AES-128.

The encryption key is defined into the following lines.

```
// The encryption key has to be the same as the one in the server
uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
rf69.setEncryptionKey(key);
```

It is highly recommended for each team to define its own encryption key.

When all the teams do use the same frequency and the same key then they will all receives the messages from the other teams sending messages. Your messages will also been received by all the other teams.

Transmission Power

The transmission power is set with the function call.

The range of power is 14 to 20 (in dBi). Lowest values requires less power. Means higher battery life but also smaller transmission distance.

```
rf69.setTxPower(20, true);
```

Notice: the second parameter concerns the HCW radio modules and indicates that extra amplifier is present.

The Emitter

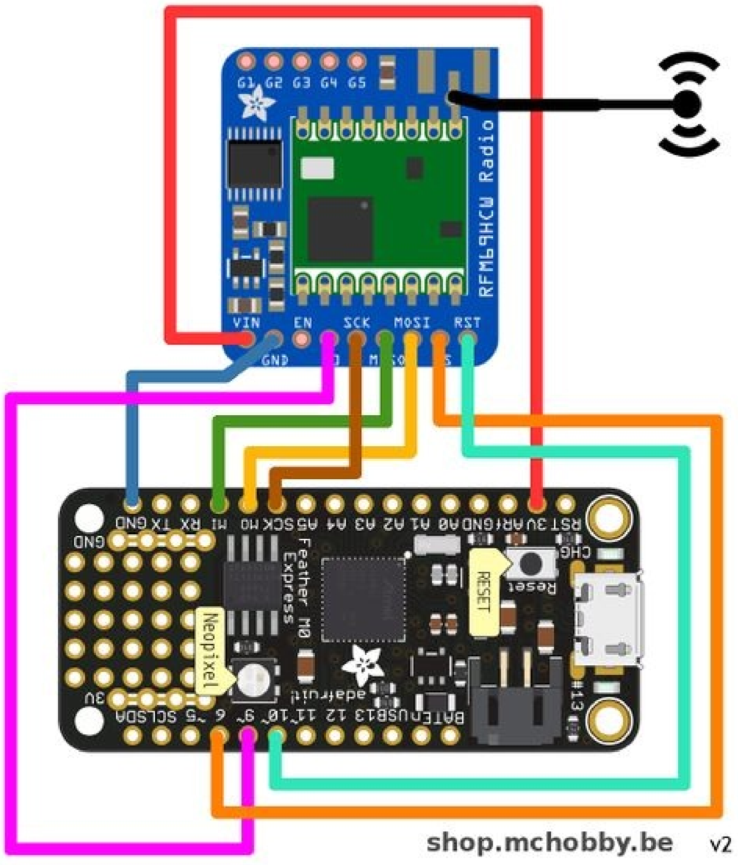
We will prepare our message emitter, typically stored inside the CanSat can.

This will involve:

- The Feather M0 Express plateform
- One of the RFM69HCW 433 Mhz module
- An wire antenna

To ease the learning, we will also connect the Feather to the computer to spy the emitted messages (which implies additional code to activate the serial port.

Wiring

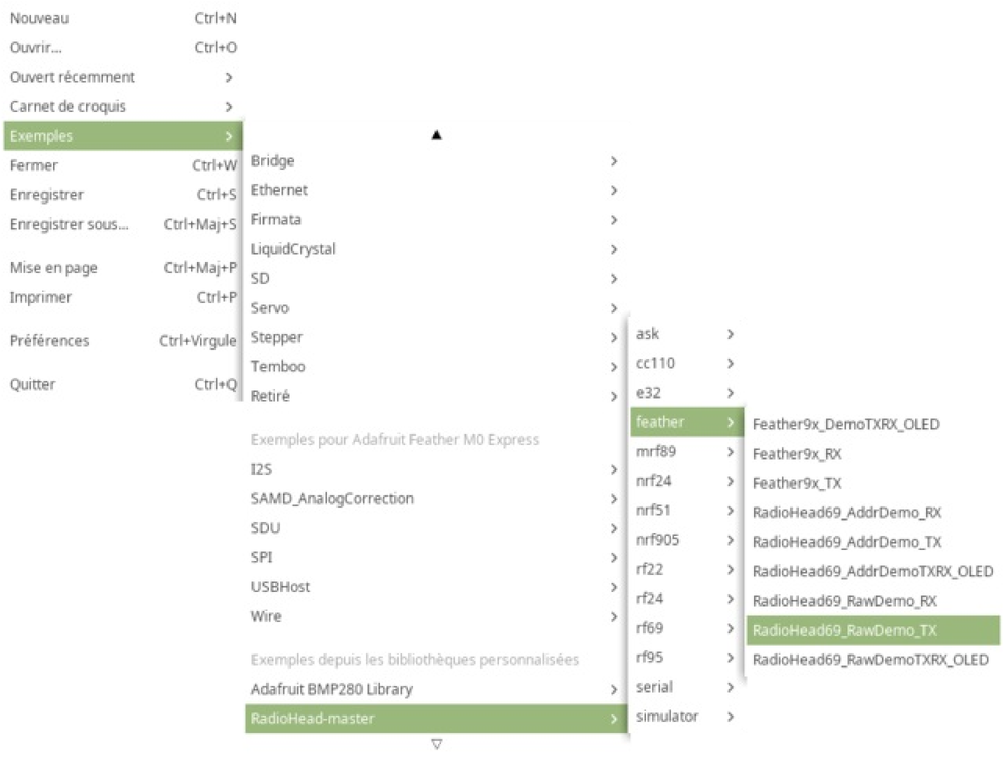


Feather M0 Express	RFM69
3V	VIN
GND	GND
MO	MOSI
MI	MISO
SCK	SCK
6	CS
9	G0
10	RST

The code

Now we will load the **emitter** example code from the RadioHead Library.

Load the sketch **file -> Examples -> RadioHead (or RadioHead-master) -> feather -> RadioHead69_RawDemo_TX**



We will have to modify the code before uploading it to the Feather M0 Express!

Indeed, the example code is provided for the **Feather M0** and not the **Feather M0 Express** so we will have to adapt the used pinout because pins 3,4 and 8 are not available on the Feather M0 Express.

Update for Interface:

Locate the following lines in the code:

```
#if defined(ARDUINO_SAMD_FEATHER_M0) // Feather M0 w/Radio
  #define RFM69_CS      8
  #define RFM69_INT     3
  #define RFM69_RST     4
  #define LED           13
#endif
```

and change it as follow ("Feather M0 Express" it is still the a "Feather M0" plateform):

```
#if defined(ARDUINO_SAMD_FEATHER_M0)
  // UPDATE for Feather M0 EXPRESS with RFM69HCW radio module
  // G0 is the Radio Module interrupt pin
  #define RFM69_CS      6
  #define RFM69_INT     9
  #define RFM69_RST     10
  #define LED           13
#endif
```

Update for frequency plan:

The RFM69HCW exists in 2 flavor:

- 900 MHz for United State usage (with a green dot)
- 433 Mhz for "Europe" license-free ISM usage (with a red dot).

The code is the same for the both flavor, you must indicates the right frequency according to the module you have. Trying to generate 900Mhz signal on a 433Mhz would result in "nothing generated"!

The RFM69HCW 433Mhz can generate signal from 424 Mhz to 510 Mhz (see datasheet https://cdn-shop.adafruit.com/product-files/3076/sx1231.pdf). You have to select the Frequency accordingly to the authorised Frequency Plan and Radio License. The 433 Mhz is free for use, please select your own frequency in that range.

Locate the following lignes:

```
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 915.0
```

And update it to:

```
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 433.0
```

Activate the Serial Line:

The begin of the `setup()` function does contains the following lines.

```
void setup()
{
  Serial.begin(115200);
  //while (!Serial) { delay(1); } // wait until serial console is open, remove if not tethered to computer
```

Update it and remove the comment mark in the front of the `while` loop like showed here under.

```
void setup()
{
  Serial.begin(115200);
  while (!Serial) { delay(1); } // wait until serial console is open, remove if not tethered to computer
```

This way, the Feather will wait for the "serial monitor" to be open before starting the sketch.

This `while` loop is important otherwise, none of the serial.print() would be visible in the serial monitor.

Voilà, We are ready to compile and upload.

Compile and upload

Select the proper board in the menu **Tools -> Type of board** : Adafruit Feather M0 Express

Select the proper port in the menu **Tools -> Port**

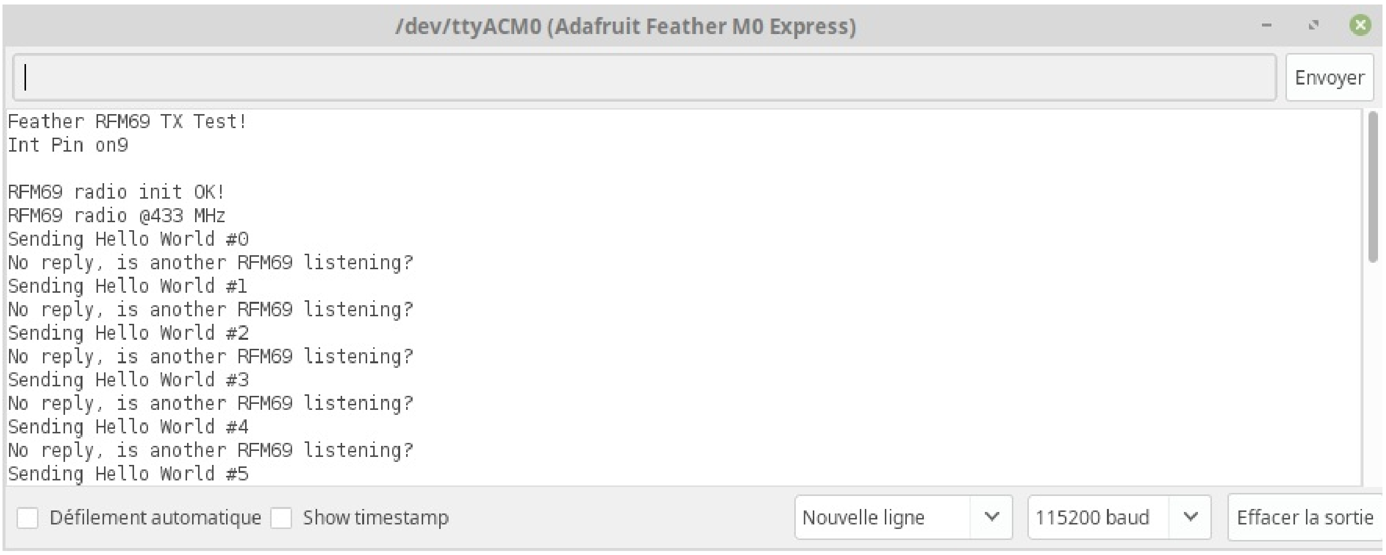
Then press the "compile" button.

If you add trouble to flash the Feather, you can still activate manually the **boot mode** by double pressing the reset button before compiling the sketch.

Running the sketch

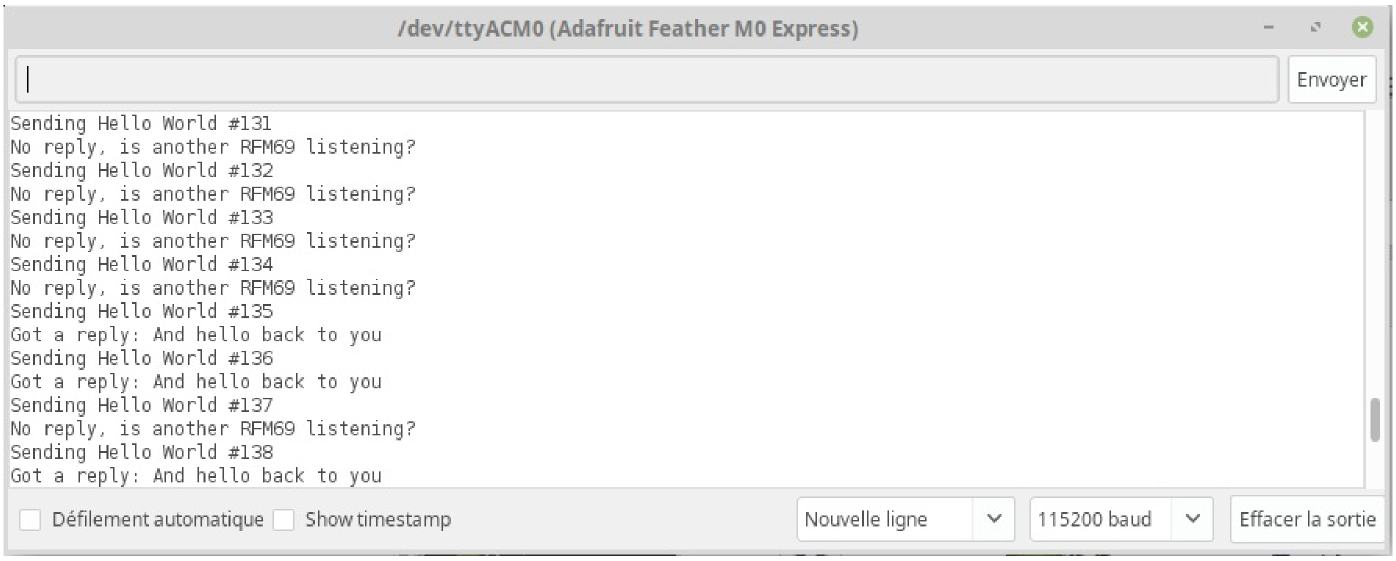
Now open the Serial Monitor and set the baud rate to 115200 baud.

As there is no board listening and answering, you should see the following results on the screen.



```
Feather RFM69 TX Test!
Int Pin on9

RFM69 radio init OK!
RFM69 radio @433 MHz
Sending Hello World #0
No reply, is another RFM69 listening?
Sending Hello World #1
No reply, is another RFM69 listening?
Sending Hello World #2
No reply, is another RFM69 listening?
Sending Hello World #3
No reply, is another RFM69 listening?
Sending Hello World #4
No reply, is another RFM69 listening?
Sending Hello World #5
```

Later, when the the receiver would be ready, the message will turn from "Is another RFM69 Listening?" to "Got reply:"



```
Sending Hello World #131
No reply, is another RFM69 listening?
Sending Hello World #132
No reply, is another RFM69 listening?
Sending Hello World #133
No reply, is another RFM69 listening?
Sending Hello World #134
No reply, is another RFM69 listening?
Sending Hello World #135
Got a reply: And hello back to you
Sending Hello World #136
Got a reply: And hello back to you
Sending Hello World #137
No reply, is another RFM69 listening?
Sending Hello World #138
Got a reply: And hello back to you
```

The Receiver

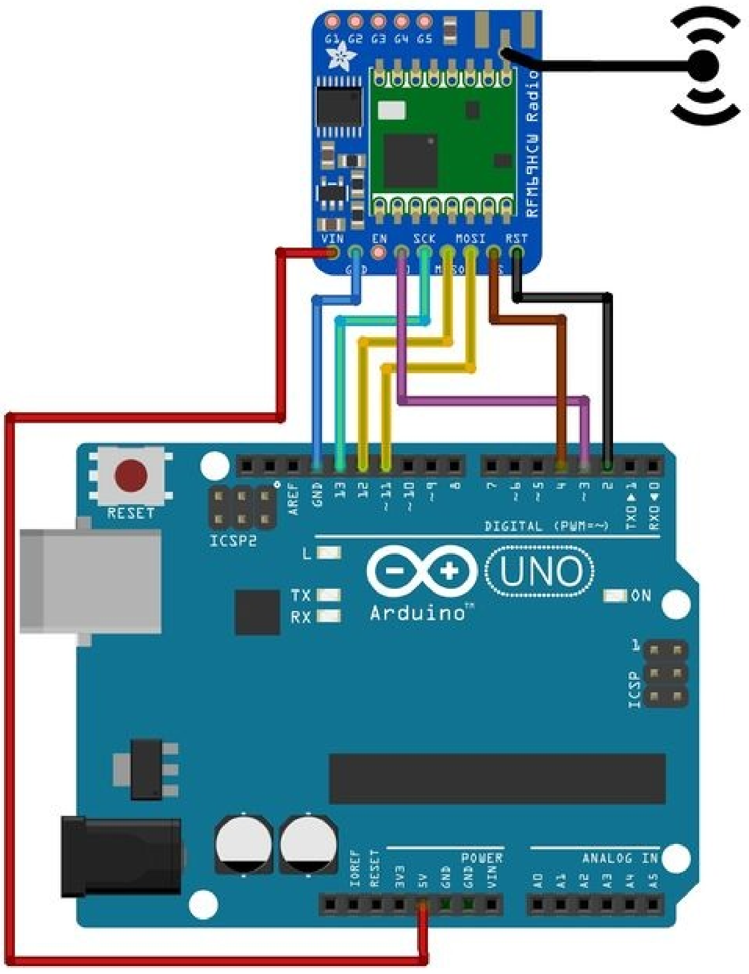
Now we will prepare our receiver station.

The receiver stays on the ground and receive the messages sent by the Emitter and forward them to a computer.

This will involve:

- The second RFM69HCW 433 Mhz module
- An arduino compatible microcontroler (we selected an Arduino Uno)
- A computer to read the messages
- An wire antenna

Wiring



Feather M0 Express	RFM69
5V	VIN
GND	GND
11	MOSI
12	MISO
13	SCK
4	CS
3	G0
2	RST

The code

Now we will load the **receiver** example code from the RadioHead Library.

Load the sketch **file -> Examples -> RadioHead (or RadioHead-master) -> feather -> RadioHead69_RawDemo_RX**

We will have to modify the code before uploading it to the Arduino Uno!

Update for Interface:

No update are required for the interface as we the following the wiring for ATmega328P:

```
#if defined (__AVR_ATmega328P__)  // Feather 328P w/wing
  #define RFM69_INT     3  //
  #define RFM69_CS      4  //
  #define RFM69_RST     2  // "A"
  #define LED           13
#endif
```

Update for frequency plan:

The frequency used by the receiver RFM69HCW must be exactly the same as th emitter! z is free for use, please select your own frequency in that range.

Locate the following lignes:

```
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 915.0
```

And update it to:

```
// Change to 434.0 or other frequency, must match RX's freq!
#define RF69_FREQ 433.0
```

Activate the Serial Line:

No need to change here as we are using an Arduino UNO.

Voilà, We are ready to compile and upload.

Compile and upload

Select the proper board in the menu **Tools -> Type of board** : Arduino/Genuino UNO

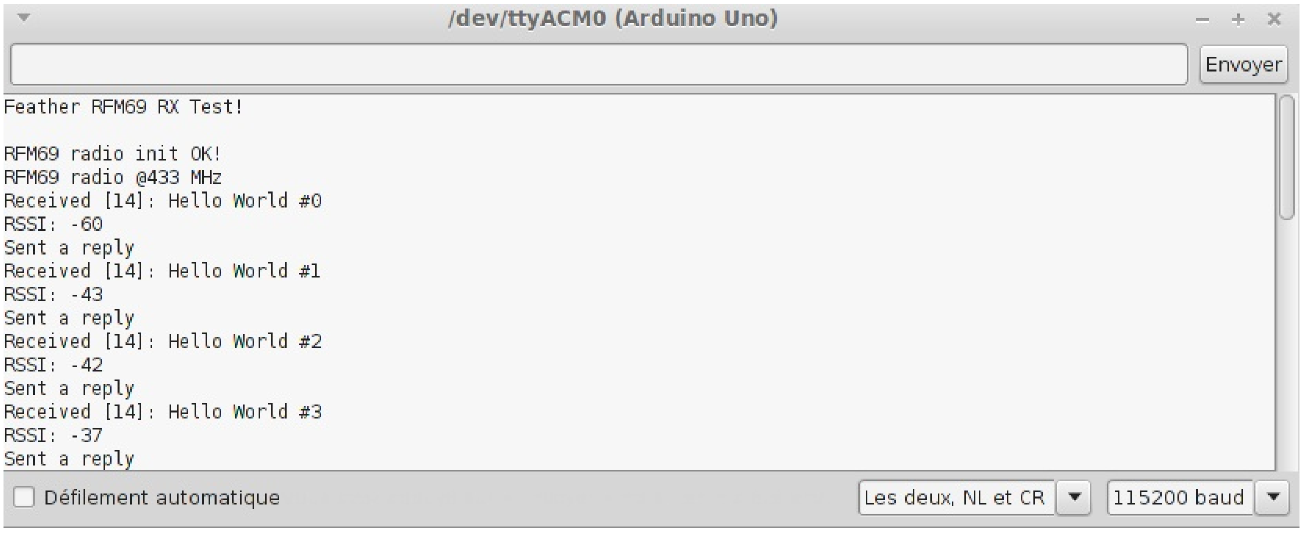
Select the proper port in the menu **Tools -> Port**

Then press the "compile" button.

Running the sketch

Now open the Serial Monitor and set the baud rate to 115200 baud.

As we did already started the "emitter" board, the receiver board will immediately display the received message and sends replies.



```
Feather RFM69 RX Test!

RFM69 radio init OK!
RFM69 radio @433 MHz
Received [14]: Hello World #0
RSSI: -60
Sent a reply
Received [14]: Hello World #1
RSSI: -43
Sent a reply
Received [14]: Hello World #2
RSSI: -42
Sent a reply
Received [14]: Hello World #3
RSSI: -37
Sent a reply
```

The Serial Console also displays the RSSI which indicates the quality of the radio signal (-15 is the best signal we could have, -60 as displayed on the screen is a really bad signal).

More info

Understanding sketch content

Reading the sketch would help to understand how the code works.

Those Adafruit examples codes are also documented on the on this page of the *Adafruit Learning System* https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts/using-the-rfm69-radio#setup-9-36 .

Addressed & Reliable Communication

More complex setup could used addressed communication and **Reliable Datagram**.

- **Addressed communication** allows you to associate a unique identifier (an integer value) to each RFM69 module. This allows detect the sender when receiving a message on the frequency and to act properly.
- **Reliable Datagram** do a lot of management with connection to make sure that the packets were received. You do not have have to send the acknowledgement in your code, the Reliable Datagram take care of it for you.

The **RadioHead** library contains the examples `RadioHead69_AddrDemo_RX` and `RadioHead69_AddrDemo_TX` that demonstrate the adressed and reliable communication. See the Addressed RX and TX Demo https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts/using-the-rfm69-radio#addressed-rx-and-tx-demo-9-50 on the Adafruit's learning system.

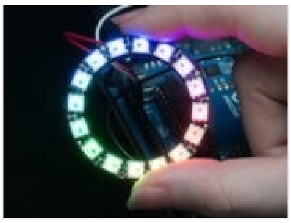
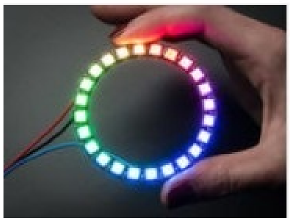
Onboard NeoPixel

Sommaire

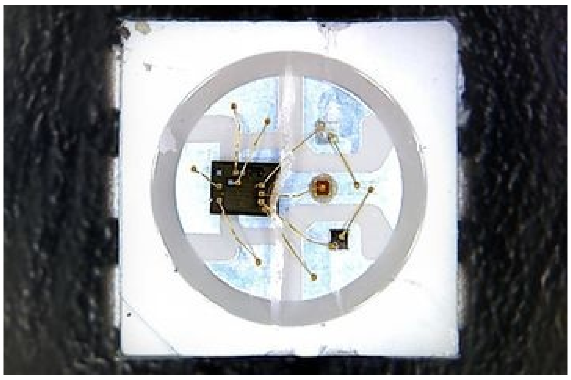
What are NeoPixels?

NeoPixel is a brand from Adafruit Industries. It concerns individually addressable LEDs that can be controlled with a single data line. Individually addressable means that you can control the color of each LED independently of the other.

NeoPixel are usually sold in strand but is also very popular is various form-factor.

The NeoPixels contains a small microcontroller used to decode the information send on the Data Line and using PWM generator to drive the LEDs.

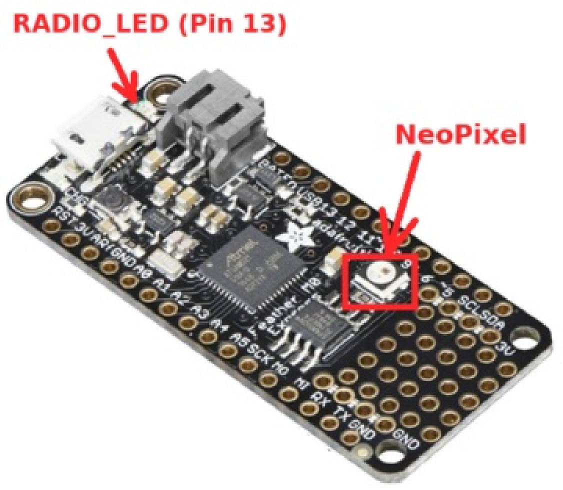


It also include a constant current supply for the LED, which means that the color would not change with the supply voltage.

NeoPixel is a very powerful technology. You can learn more about it from the Adafruit's neoPixel User Guide https://learn.adafruit.com/adafruit-neopixel-uberguide or the translated NeoPixel user guide.

Feather M0 NeoPixel

The Feather M0 Express also contains a NeoPixel LED. This LED is used by the Microcontroller to inform the user about its running status (red or green).



The great thing about this NeoPixel is that we can take the control of it from your Arduino sketch.

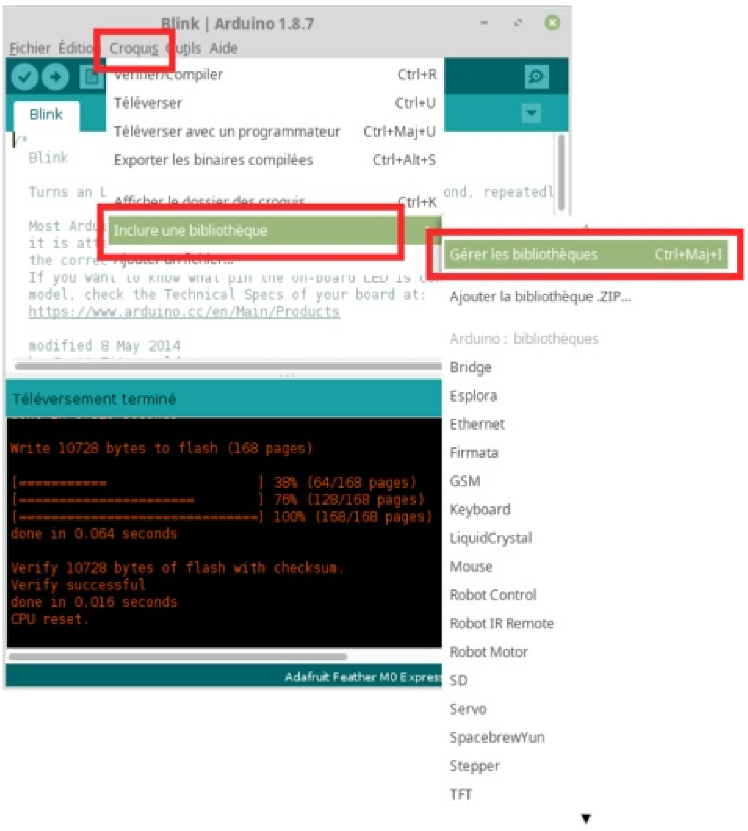
We just need to know the pin to use (**pin 8**) and install the Adafruit NeoPixel Library for ATSAMD21 microcontroller :-)

The NeoPixel should be viewed as the very first LED of a NeoPixel Strand having only one single LED.

Installing the library

You can also use the "**Library Manager**" to ease the installation of the BMP280 library.

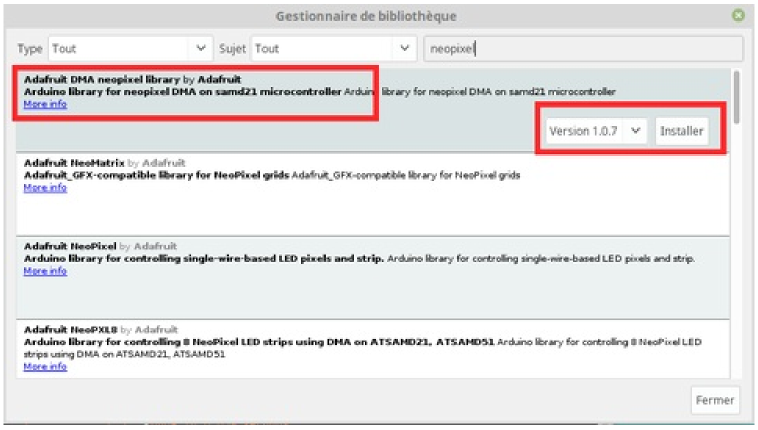
From the "**Sketch**" menu, select the sub-menu "**Include library**" --> "**Library Manager**" like shown on the picture here under.



```
modified 8 May 2014
```

```
Write 10728 bytes to flash (168 pages)

[============                  ] 38% (64/168 pages)
[========================      ] 76% (128/168 pages)
[==============================] 100% (168/168 pages)
done in 0.064 seconds

Verify 10728 bytes of flash with checksum.
Verify successful
done in 0.016 seconds
CPU reset.
```

In the library manager, key-in the value "**neopixel**" in the search box. Then click on the install button in the front of the **Adafruit DMA NeoPixel Library by Adafruit**. This library is suited for the ATSAMD21 microcontroller as used on this Feather M0 plateform.


Gestionnaire de bibliothèque

Type Tout Sujet Tout neopixel

Adafruit DMA neopixel library by **Adafruit**
Arduino library for neopixel DMA on samd21 microcontroller Arduino library for neopixel DMA on samd21 microcontroller
More info

Version 1.0.7 Installer

Adafruit NeoMatrix by Adafruit
Adafruit_GFX-compatible library for NeoPixel grids Adafruit_GFX-compatible library for NeoPixel grids
More info

Adafruit NeoPixel by Adafruit
Arduino library for controlling single-wire-based LED pixels and strip. Arduino library for controlling single-wire-based LED pixels and strip.
More info

Adafruit NeoPXL8 by Adafruit
Arduino library for controlling 8 NeoPixel LED strips using DMA on ATSAMD21, ATSAMD51 Arduino library for controlling 8 NeoPixel LED strips using DMA on ATSAMD21, ATSAMD51
More info

Fermer

Test script

The following script demonstrate how to manipulate the onboard NeoPixel.

As there is only one pixel (one pixel on the strand), the function `pixel.setPixelColor()` will always have the first parameter is set to 0!

This means that we change the color of the LED #0 on the *pixel strand*.

```
#include <Adafruit_NeoPixel.h>

#define NEOPIXEL       8
#define NUMPIXELS      1

Adafruit_NeoPixel pixel = Adafruit_NeoPixel(NUMPIXELS, NEOPIXEL, NEO_GRB + NEO_KHZ800);

void setup() {
  pixel.begin();
  // switch OFF the pixel 0
  pixel.setPixelColor(0, pixel.Color(0,0,0));
  pixel.show();
}

int red = 0;
int green = 0;
int blue = 0;

void loop(){
  pixel.setPixelColor(0, pixel.Color(red,green,blue));
  pixel.show();
  red += 10;
  if( red > 255 ){
    red = 0;
    green += 10;
  }
  if( green > 255 ){
    green = 0;
    blue += 10;
  }
  if( blue > 255 ){
    blue = 0;
  }
  // Wait 100ms
  delay( 100 );
}
```

Frequency Plan

Forewords

The RFM69 module and Packet Radio can do a lot to secure message transmission and message content (with the encryption key).

As for wired network (aka TCP/IP) the packet radio enclose your data into datagram before sending it over the air. This helps the hardware to detect error and possibly recover when collisions occurs.

Obviously, more we are talking on a same frequency, more we will have collision, less the communication will be efficient.

The best solution would be to use a frequency plan where each team receives its own frequency range like showed in the following table.

Suggested Frequency Plan

Together with the attributed Frequency, we do recommend the Teams to define their own encryption KEY for radio transmission (this will be clarified further in the documentation).

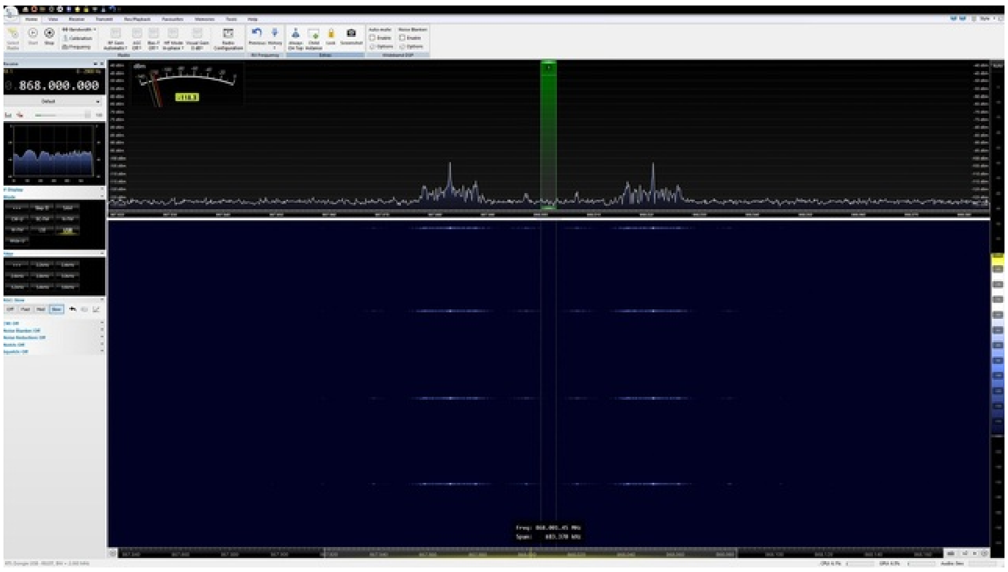
Team	Freq (MHz)	Team name
Team #1	433.1	. .
Team #2	433.2	
Team #3	433.3	
Team #4	433.4	
Team #5	433.5	
Team #6	433.6	
Team #7	433.7	
Team #8	433.8	
Team #9	433.9	
...	434.0	...

Frequency plan explained

Why do we space the frequencies of 0.1 MHz (so 100 KHz)? Spacing more (>100 KHz) will be best, spacing less (<100 KHz) is not recommended.

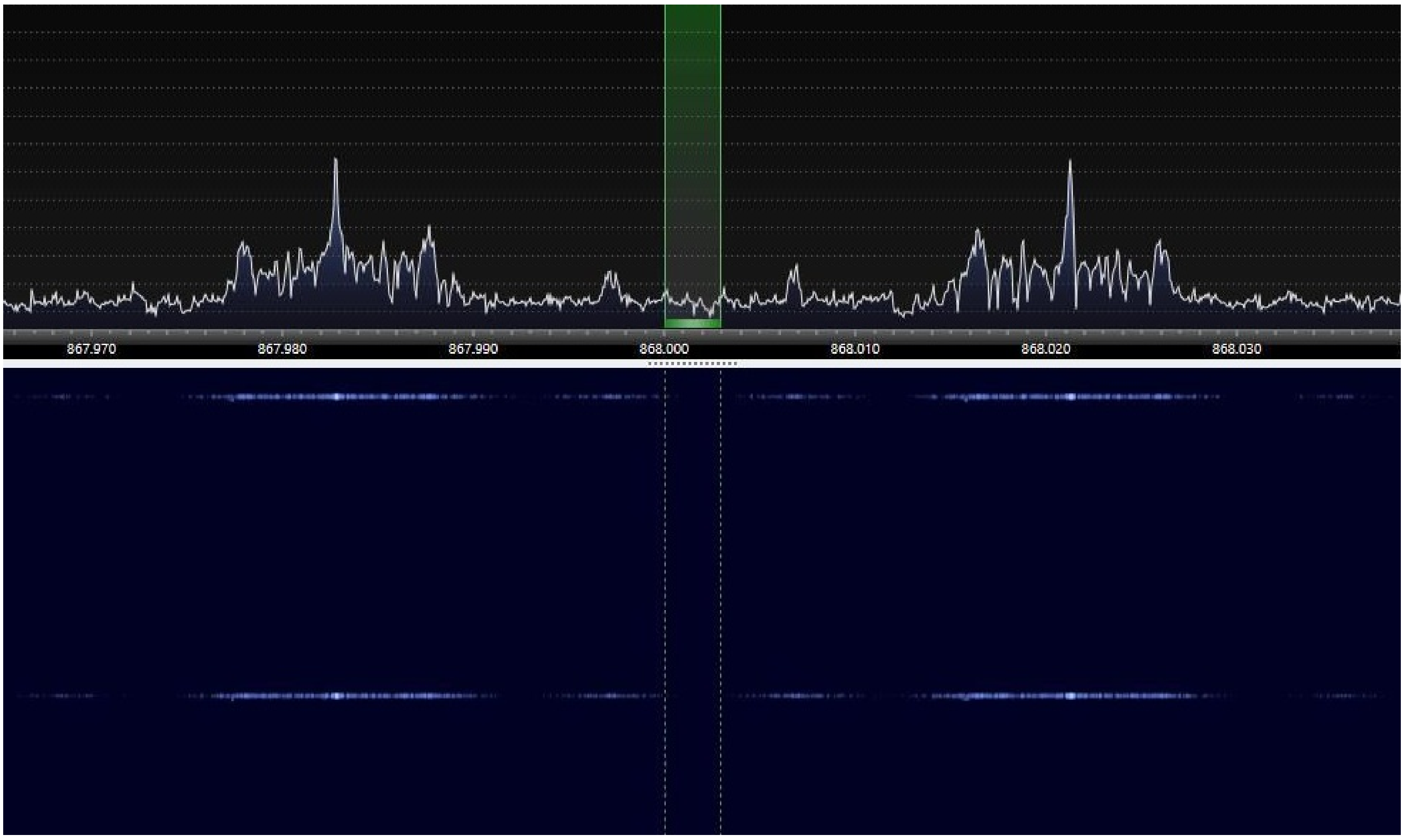
The following capture coming from USA shows the spectrum view (and waterfall view) of a RFM69 emiting on the 868.0 MHz frequency. Just remember that it works the same for CanSat around the 433 Mhz.

Note: the 868.0 Mhz is a FREE ISM band in USA. In Europe, that frequency range is reserved for LoRa transmission!



Source: this thread in the mysensors.org forum https://forum.mysensors.org/topic/11501/rfm69-range-issues

This second capture does focus on the interesting part of the picture (spectrum around 868 Mhz and corresponding waterfall).



Source: this thread in the mysensors.org forum https://forum.mysensors.org/topic/11501/rfm69-range-issues

As you can see, the transmission does take place on the right and left side around the central 868.0 MHz axis. A bit like a mirroring image. This is called "Double Side Band" (DSB) communication in the radio area with the carrier wavelength set to 868.0 Mhz.

The carrier wavelength doesn't ship any data/information (no peak in that position) since the "Double Side Bands" are enough to rebuild the transmited information.

By comparing the spectrum and waterfall (on the right part), we can see the communication take places between 868.0 MHz and 868.030 MHz. This is the same on the opposite side of the picture.

So RFM69 Packet radio transmission takes place between:

- Carrier WaveLength + 30 KHz
- Carrier WaveLength - 30 KHz

So a total of 60 KHz around the carrier Wave length.

the waterfall section does show some activities over 30 KHz which is flooded inside the noise (no real peak visible on the spectrum). It is probably some radio emiting harmonics.

So, with the starting Carrier Frequency = 433.1 MHz, we would have the following unperfect series: 433.1, 433.16, 433.22, 433.28, and so on. This proposal is unperfect because the transmission spectrums of the proposed frequencies are just "touching" each other.

The best would be to keep some room between the transmissions spectrums, so we could space the frequencies of 80 KHz or even better and simplier 100 KHz!

So the suggested frequency plan is 433.1, 433.2, 433.3, 433.4, 433.5, and so on.

Mission 1 - the Emitter

Sommaire

Introduction

Before starting this point, we recommand to follow all the sensors testing steps (BMP280 sensor, TMP36 Sensor, RFM69HCW radio, RFM69HCW Testing and onboard NeoPixel). It contains all the details about the wiring, install needed libraries and conduct basic testing.

The following Wiring is used to capture

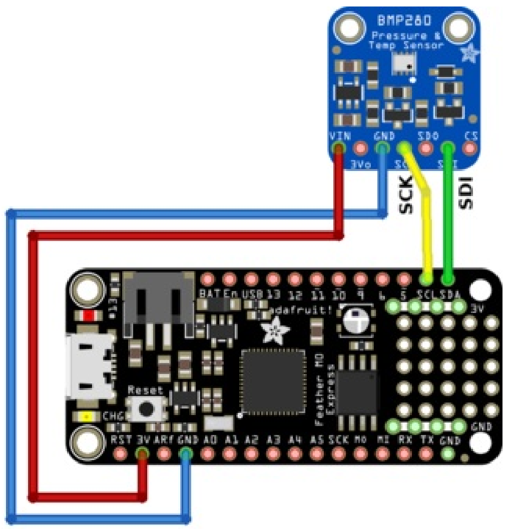
- Air temperature
- Air pressure

and transmitting the information via the RFM69HCW radio module.

Wiring

Wire the barometric sensor

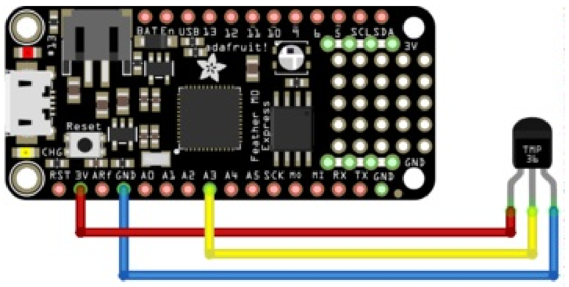
The BMP280 is wired on the I2C bus of the Feather.


BMP280
Pressure & Temp Sensor
VIN
GND
SDO
CS
3Vo
SCK
SDI
Feather M0 Express
adafruit!
Reset
CHG
BAT En USB 13 12 11 10 9 6 5 SCL SDA
RST 3V ARf GND A0 A1 A2 A3 A4 A5 SCK MO MI RX TX GND
3V
GND

Wire the temperature sensor

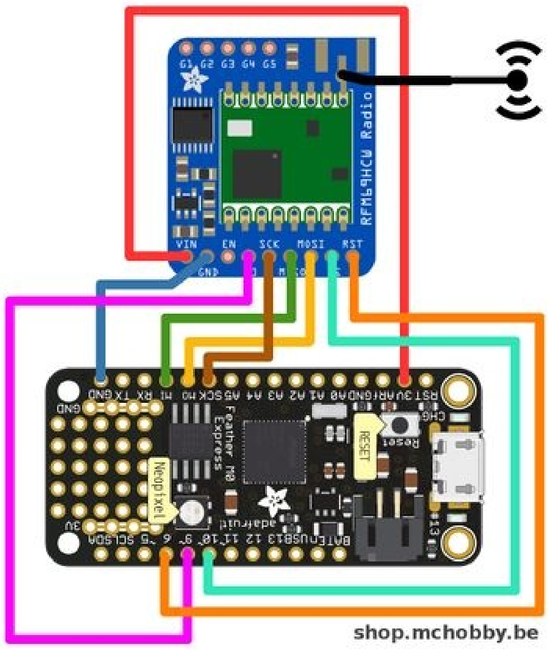
Then connect the TMP36 sensor as follows:

- The pin 1 (on the left) to a power source (3.3V),
- The pin 3 (the the right droite) to the ground/GND.
- The pin 2 (middle one) to the A3 analogue input.



Wire the radio module

Finally wire the RFM69HCW radio as follows:



Feather M0 Express	RFM69
3V	VIN
GND	GND
MO	MOSI
MI	MISO
SCK	SCK
6	CS
9	G0
10	RST

Download the code

The code is available for download on the GitHub associated to this wiki https://github.com/mchobby/cansat-belgium .

Téléchargez mission1-serial-radio-capture.ino
https://raw.githubusercontent.com/mchobby/cansat-belgium/master/mission1-serial-radio-capture/mission1-serial-radio-capture.ino

About testing

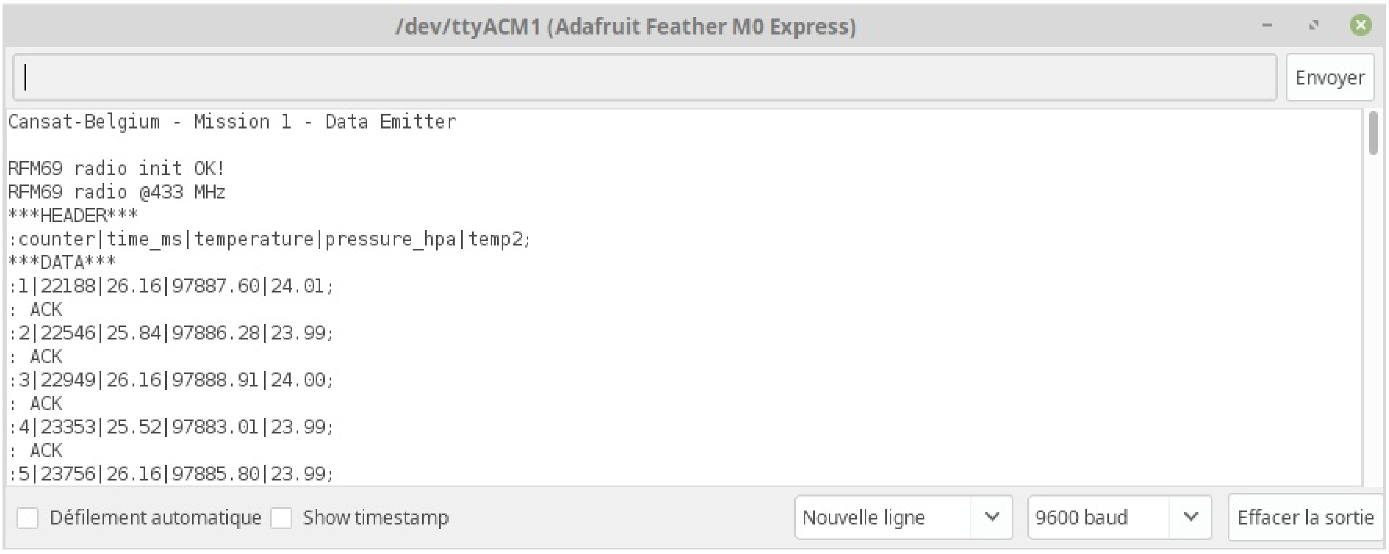
Now, we will move forward in several steps.

1. Getting data from sensors + send them it over the serial connexion (to confirm good working) + transmit over radio
2. Testing the radio reception
3. Going autonomous (removing Serial Connexion waiting) + add the Lipo

The code proposed here under has been tested up to 23197 iterations without issue, time when we decided to ends the test :-) .

Once uploaded to your Feather, open the Serial Monitor and set it to 9600 bauds. **The sketch would wait until you open the Serial Monitor to start transmitting the data**.

You should see the following messages appears on the Serial Monitor.



```
Cansat-Belgium - Mission 1 - Data Emitter

RFM69 radio init OK!
RFM69 radio @433 MHz
***HEADER***
:counter|time_ms|temperature|pressure_hpa|temp2;
***DATA***
:1|22188|26.16|97887.60|24.01;
: ACK
:2|22546|25.84|97886.28|23.99;
: ACK
:3|22949|26.16|97888.91|24.00;
: ACK
:4|23353|25.52|97883.01|23.99;
: ACK
:5|23756|26.16|97885.80|23.99;
```

Where we could see the transmitted messages with the packetnum packet index, timing and data.

The screen also displays the **ACK** acknowledgement send back by the receiver.

Structuring the data

The radio module only sends buffer of binary data to the receiver. This is a bit rough but efficient.

So to transport the data to the receiver, we need to transform the values (float, integer) into their string representation.

When having multiple data in their string representation is not enough, they must also been organized.

The final format must be easy to parse and very compact (smaller is the radio message and higher is the chance for him to get to the ground without error).

We propose the following format:

```
:data1|data2|data3|data4;/r/n
```

where:

- **:** is the begin of data stream
- **;** is the end of data stream
- **/r/n** are optional carriage return + line feed characters.
  This will would make the messages user friendly when the the messages are viewed in a console or terminal.
- **|** is the separator between data items.
- **datax** are the string representation of the various data. The characters ;:| are forbidden in this area.

we would also recommend to use:

- **packetnum** as data1. packetnum is a simple variable increment of one unit after each transmission. This would allow the receiver to detect lost message (since it would exist holes in the numbering of received messages).
- **timing_info** as data2. This would help to create timing chart or time base data analysis. We suggest to use the Arduino's `millis()` function which count the number of milliseconds since the last microcontroler reset.

As explained later in the code the `packet_str` variable contains the message to be transmitted to the ground. The Arduino's `String` class would ease the transformation of data to their string representation.

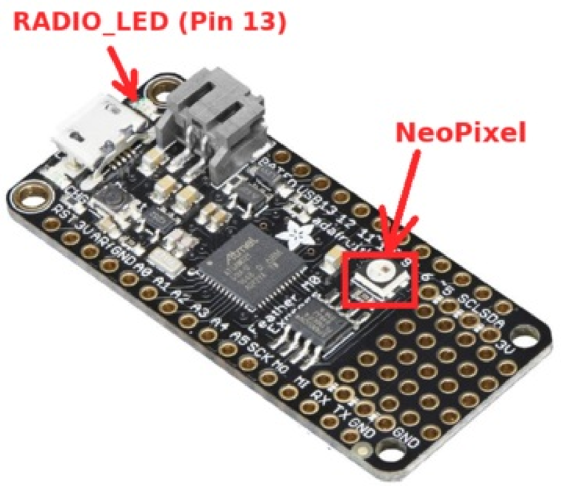
```
String packet_str = String( ":"+String(packetnum,DEC)+"|" );
packet_str.concat( String( ms,DEC)+"|" );
packet_str.concat( String( temperature, 2 )+"|" );
packet_str.concat( String( bme_hpa, 2 )+"|" );
packet_str.concat( String( bme_temp, 2 )+";\r\n" );
```

LEDs and Error management

Being able to understand rapidly what's happening inside your object is essential to rapidly fix the issue.

The best is to figure out what's happening is to use LED, blink status, heartbeat.

By doing so, no need to open a Serial Monitor or diagnostic tool to figure out the status of the object.



The NeoPixel LED does turn GREEN when the Feather M0 switch on to normal operation (when it runs your Arduino Sketch).

In the following sample, we do take the control over the NeoPixel LED to switch it off at the end of `setup()` function. This means that all buses and devices are properly initialized.

The RADIO_LED wired on the Pin 13 is used to signal radio status when emitting a message.

LED operation	Description	Fix the issue
NeoPixel GREEN	The `setup()` function did not complete initialization because of a crash.	Check the wiring of sensors. Test each sensor separately (with their tests code). If this not working, remove all sensors except the one you are testing.
NeoPixel OFF	The `setup()` did complete successfully. The main `loop()` is not running.	*Nothing to do here, just check the RADIO_LED for more informations.*
RADIO LED = 1 pulse 50ms	The LED is pulsed for each successfully send message + getting ACK from the receiver. The code wait 500ms max for the ACK.	*Nothing to do here.*
RADIO LED = 2 pulse 50ms + pause 100ms	Message send but error while decoding the ACK response.	*This is not critical, the most important is that the message was sent successfully.*
RADIO LED = 3 pulse 50ms + pause 150ms	Not ACK message received within the 500ms after message was sent. This can be interpreted as "Is there someone listening the message?" because there are not reply.	*This is not critical, the most important is that the message was sent successfully.*

The code explained

Here some explanation about the {fname|mission1-serial-radio-capture.ino}} sketch used in the CanSat.

This Arduino sketch would:

1. **Wait for the serial connexion** to be established before starting the sketch
2. Collect the sensor data
3. Send it to serial connexion
4. Send it over the radio connexion

Don't forget to update the radio frequency `RF69_FREQ` and the encryption key `key[]`

First, the script will includes all the needed libraries.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_NeoPixel.h>
#include <RH_RF69.h>
```

Then, it defines the parameters for the radio module and the pinout used to wire the RFM69HCW radio module to the Feather M0.

The last line create the object `rf69` to control the module.

```
#define RF69_FREQ 433.0
```

```
#define RFM69_CS      6
#define RFM69_INT     9
#define RFM69_RST     10
#define RADIO_LED     13

RH_RF69 rf69(RFM69_CS, RFM69_INT);
```

Defining the parameters to control the NeoPixel LED available on the board. That LED is wired on the Pin 8.

The last line creates an objet pixel which is a Pixels Strand of only 1 pixel length.

```
#define NEOPIXEL       8
#define NUMPIXELS      1

Adafruit_NeoPixel pixel = Adafruit_NeoPixel(NUMPIXELS, NEOPIXEL, NEO_GRB + NEO_KHZ800);
```

Defining the parameter and objects for temperature and pressure sensor.

```
#define temperaturePin A3

Adafruit_BMP280 bme; // wired with I2C
```

Initialize the serial connexion @ 9600 bauds, the BMP sensor, the radio module (`init_radio_module()`) and pixel.

The `while(!Serial)` waits that you open the serial monitor so effectively starts the sketch.

The NeoPixel is turned when the `setup()` function is complete.

```
void setup() {
  Serial.begin(9600);

  // wait until serial console is open
  while (!Serial) { delay(1); }

  if (!bme.begin()) {
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");
    while (1);
  }

  init_radio_module();

  // everything is right! So switch off neopixel
  pixel.begin();
  pixel.setPixelColor(0, pixel.Color(0,0,0)); // switch off
  pixel.show();
}
```

Now, we can focus on the main loop.

The first step is to send the column header (so we know what are the data) if not done yet.

Then we reads the sensors (as we have tested them, this should not be a surprise). We also capture the time with the function `millis()`, so the `ms` variable contains the number of milliseconds since the last reset.

Finally, we do increment the `packetnum` variable. This would allows to track lost packets on the receiver side.

`packet_str` is the message to send via radio. It is composed with `String` objects and concatenation operations. `String` are welcome to transform **float** into string representation since most common float to string C standard functions would fail to work properly onto Arduino alike plateforms.

The key function to transform the **String object' *into a*** *C buffer* is `packet_str.c_str()` which offer an access to the underlying array of bytes (exactly what the radio module library would need).

The remaining of the radio transmission code is almost the same as the RFM69HCW module testing code (except that error messages are remplaced by Blinking LED).

```
bool header_send = false;
// packet number increment at each data transmission
int16_t packetnum = 0;
void loop() {
    // --- SEND COLUMNS HEADER -------------------
    if( !(header_send) ){
        send_header();
        header_send = true;
    }

    // --- READ SENSORS ---------------------------
    float voltage = getVoltage(temperaturePin);
    float temperature = (voltage - .5) *100;

    float bme_temp = bme.readTemperature();
    float bme_hpa  = bme.readPressure();

    unsigned long ms = millis();
    packetnum += 1; // increment

    // --- Compose the Message to send ------------
    String packet_str = String( ":"+String(packetnum,DEC)+"|" );
    packet_str.concat( String(ms,DEC)+"|" );
    packet_str.concat( String( temperature, 2 )+"|" );
    packet_str.concat( String( bme_hpa, 2 )+"|" );
    packet_str.concat( String( bme_temp, 2 )+";\r\n" );

    // send to Serial
    Serial.print( packet_str.c_str() );
    // Send over Radio
    rf69.send((uint8_t *)(packet_str.c_str()), packet_str.length());
    rf69.waitPacketSent();

    // Now wait for a reply
    uint8_t buf[4]; // We limit the quantity received data
    uint8_t len = sizeof(buf);

    if (rf69.waitAvailableTimeout(500))  {
      // Should be a reply message for us now
      if (rf69.recv(buf, &len)) {
          Serial.print(": ");
          Serial.println((char*)buf);
          Blink(RADIO_LED, 50, 1); //blink LED once, 50ms between blinks
      } else {
          Serial.println("Receive failed");
          Blink(RADIO_LED, 50, 1); //blink LED once, 50ms between blinks
      }
    } else {
        Serial.println("No reply, is another RFM69 listening?");
        Blink(RADIO_LED, 50, 3 ); // blink 3 times, 50ms between blinks
    }

    // Going to next round
}
```

The `init_radio_module()` function is called from the `setup()`.

This function does all the stuff to initialize the RFM69HCW modules. Set the transmission power, the frequency and the **encryption key**.

```

void init_radio_module() {
  pinMode(RADIO_LED, OUTPUT);
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, LOW);

  Serial.println("Feather RFM69 TX Test!");
  Serial.println();

  // manual reset
  digitalWrite(RFM69_RST, HIGH);
  delay(10);
  digitalWrite(RFM69_RST, LOW);
  delay(10);

  if (!rf69.init()) {
    Serial.println("RFM69 radio init failed");
    while (1);
  }
  Serial.println("RFM69 radio init OK!");
  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM (for low power module)
  // No encryption
  if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("setFrequency failed");
  }

  // If you are using a high power RF69 eg RFM69HW, you *must* set a Tx power with the
  // ishighpowermodule flag set like this:
  rf69.setTxPower(20, true);  // range from 14-20 for power, 2nd arg must be true for 69HCW

  // The encryption key has to be the same as the one in the server
  uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
  rf69.setEncryptionKey(key);

  pinMode(RADIO_LED, OUTPUT);

  Serial.print("RFM69 radio @");
  Serial.print((int)RF69_FREQ);
  Serial.println(" MHz");
}
```

This function send the header information to the Serial monitor.

Ideally, this function should also send it via the radio.

```
void send_header() {
  String s1 = String( F("***HEADER***\r\n") );
  Serial.print( s1 );
  String s2 = String( F(":counter|time_ms|temperature|pressure_hpa|temp2;\r\n") );
  Serial.print(s2);
  String s3 = String( F("***DATA***\r\n") );
  Serial.print( s3 );

}
```

Helper function used to blink a LED. Note that a pause of 3 time the blinking time. This will ease the identification of blink code into other blinking patterns.

```
void Blink(byte PIN, byte DELAY_MS, byte loops) {
  for (byte i=0; i<loops; i++)  {
    digitalWrite(PIN,HIGH);
    delay(DELAY_MS);
    digitalWrite(PIN,LOW);
    delay(DELAY_MS);
  }
  // exit: wait 3 times the delay
  delay( 3* DELAY_MS );
}
```

This function returns the voltage for the analog Pin.

It converts a digital value between 0 & 1024 (from ADC) to voltage between 0 & 3.3 volts.

```
float getVoltage(int pin){
   //  each unit equal 3.3 / 1024 = 3.2 millivolts
   return (analogRead(pin) * .0032);
}
```

Fault tolerant design

The goal is to transmit the data to the ground station.
The code of the Emitter (this section) and Receiver (next section) are doing the job.

However, what would happens to your data if the antenna did break? All the data are lots!

This is where the "Extra Flash" would be a great help!

As showed earlier, it is also possible to store/save the data into the Flash.

A good approach would be:

1. to save the data in the Flash
2. then send it over Radio.

In this way, the data stays available inside the CanSat and could be extracted as suited.

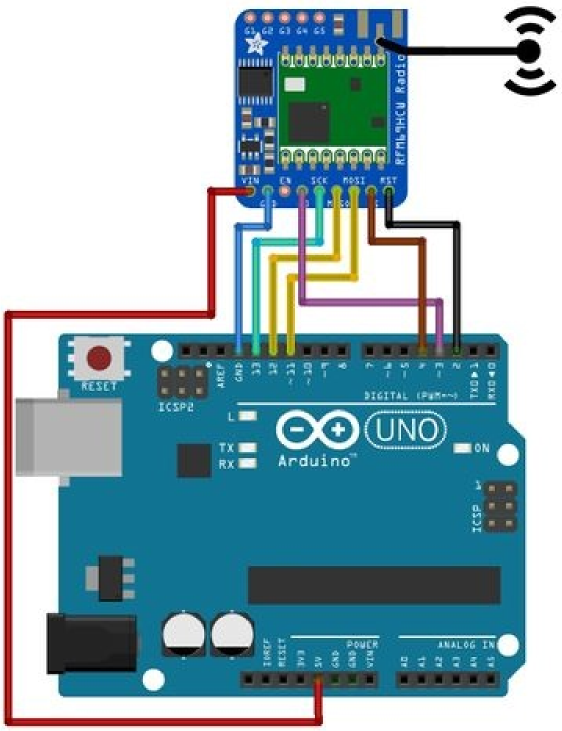
Mission 1 - the Receiver

Sommaire

Introduction

The following wiring will prepare the "Receiver Station" for the mission 1. From the "RFM69HCW Testing" section, we will use an Arduino UNO and RFM69HCW module to **redirect the Radio Messages to the serial port**.

Wiring



Feather M0 Express	RFM69
5V	VIN
GND	GND
11	MOSI
12	MISO
13	SCK
4	CS
3	G0
2	RST

Download the code

The code is available for download on the GitHub associated to this wiki https://github.com/mchobby/cansat-belgium .

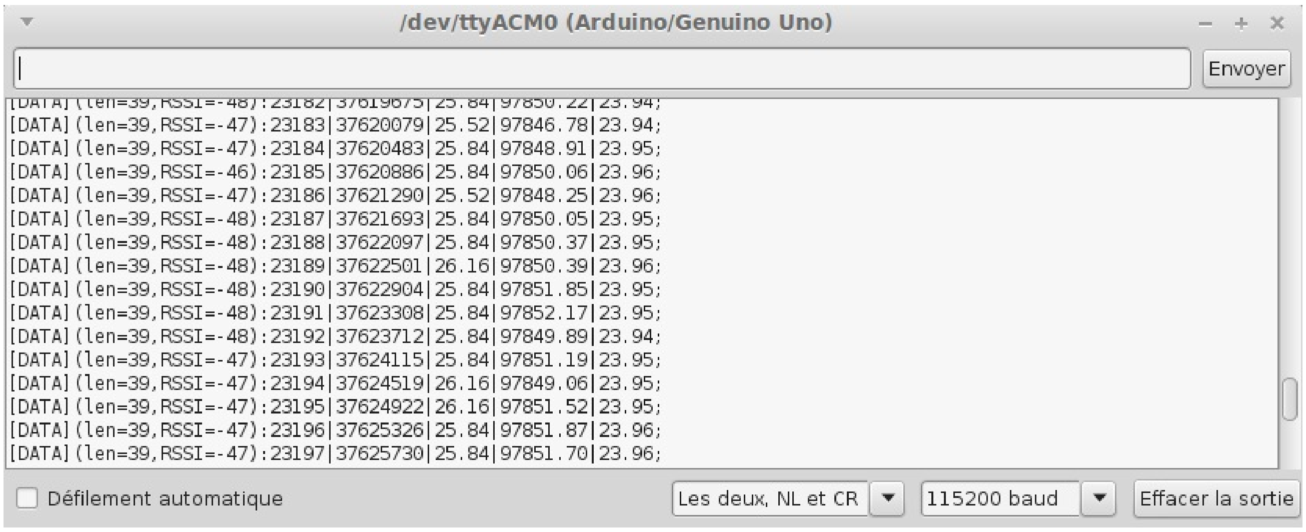
Téléchargez mission1-serial-radio-

receiver.ino
https://raw.githubusercontent.com/mchobby/cansat-belgium/master/mission1-serial-radio-receiver/mission1-serial-radio-receiver.ino

About testing

Once uploaded to your Arduino, open the Serial Monitor and set it to 115200 bauds.

You should see the following messages appears on the Serial Monitor.



```
[DATA](len=39,RSSI=-48):23182|37619675|25.84|97850.22|23.94;
[DATA](len=39,RSSI=-47):23183|37620079|25.52|97846.78|23.94;
[DATA](len=39,RSSI=-47):23184|37620483|25.84|97848.91|23.95;
[DATA](len=39,RSSI=-46):23185|37620886|25.84|97850.06|23.96;
[DATA](len=39,RSSI=-47):23186|37621290|25.52|97848.25|23.96;
[DATA](len=39,RSSI=-48):23187|37621693|25.84|97850.05|23.95;
[DATA](len=39,RSSI=-48):23188|37622097|25.84|97850.37|23.95;
[DATA](len=39,RSSI=-48):23189|37622501|26.16|97850.39|23.96;
[DATA](len=39,RSSI=-48):23190|37622904|25.84|97851.85|23.95;
[DATA](len=39,RSSI=-48):23191|37623308|25.84|97852.17|23.95;
[DATA](len=39,RSSI=-48):23192|37623712|25.84|97849.89|23.94;
[DATA](len=39,RSSI=-47):23193|37624115|25.84|97851.19|23.95;
[DATA](len=39,RSSI=-47):23194|37624519|26.16|97849.06|23.95;
[DATA](len=39,RSSI=-47):23195|37624922|26.16|97851.52|23.95;
[DATA](len=39,RSSI=-47):23196|37625326|25.84|97851.87|23.96;
[DATA](len=39,RSSI=-47):23197|37625730|25.84|97851.70|23.96;
```

Where we could see the received messages with additional information.

```
[DATA](len=<data_len>,RSSI=<radio_rssi>)<transmitted_data>
```

- Each data received and send to the serial connexion are prefixed with **[DATA]**
- The prefix is followed by information enclosed between parenthesis **()**, this concerns the *received data*. Entries are key=value pairs separated by coma.
- At the end, we retrieve the transmitted data (as they have been sent).

In the informations:

- **data_len**: length of the data stream received.
- **RSSI**: indicated the strength of the signal https://en.wikipedia.org/wiki/Received_signal_strength_indication (-15 at best, -90 at worst).
- **transmitted_data**: the data as transmitted by the emitter. As designed in the emitter, it starts with **:** and ends with **;\r\n**

In the **transmitted_data**, we can identify:

- The packet counter
- The time counter (milliseconds)
- The temperature (from tmp36)
- The atmospheric pressure (from bmp280)
- The temperature2 (from bmp280)

The code explained

Here some explanation about the `mission1-serial-radio-receiver.ino` sketch used in the CanSat.

This Arduino sketch would:

1. Collect the sensor data over the radio connexion
2. Reply an ACK to the Emitter
3. Send it the data to the serial connexion

Don't forget to update the radio frequency `RF69_FREQ` and the encryption key `key[]`

First, the script will includes all the needed libraries.

```
#include <SPI.h>
#include <RH_RF69.h>
```

Then, it defines the parameters for the radio module and the pinout used to wire the RFM69HCW radio. The code adapt himself to the the board selected in the compiler.

The last line create the object `rf69` to control the module.

```
#define RF69_FREQ 433.0

#if defined (__AVR_ATmega32U4__) // Feather 32u4 w/Radio
  #define RFM69_CS      8
  #define RFM69_INT     7
  #define RFM69_RST     4
  #define LED           13
#endif

#if defined(ARDUINO_SAMD_FEATHER_M0) // Feather M0 w/Radio
  #define RFM69_CS      8
  #define RFM69_INT     3
  #define RFM69_RST     4
  #define LED           13
#endif

#if defined (__AVR_ATmega328P__)  // Feather 328P w/wing (or Arduino UNO)
  #define RFM69_INT     3  //
  #define RFM69_CS      4  //
  #define RFM69_RST     2  // "A"
  #define LED           13
#endif

#if defined(ESP8266)    // ESP8266 feather w/wing
  #define RFM69_CS      2    // "E"
  #define RFM69_IRQ     15   // "B"
  #define RFM69_RST     16   // "D"
  #define LED           0
#endif

#if defined(ESP32)    // ESP32 feather w/wing
  #define RFM69_RST     13   // same as LED
  #define RFM69_CS      33   // "B"
  #define RFM69_INT     27   // "A"
  #define LED           13
#endif

// Singleton instance of the radio driver
RH_RF69 rf69(RFM69_CS, RFM69_INT);
```

The `setup()` function:

- initialize the serial connexion @ 115200 bauds
- initialze the radio module

```
void setup() {
  Serial.begin(115200);

  pinMode(LED, OUTPUT);
  pinMode(RFM69_RST, OUTPUT);
  digitalWrite(RFM69_RST, LOW);

  Serial.println("[INFO] CanSat Belgium Radio Receiver (Radio to Serial)!");

  // manual reset
  digitalWrite(RFM69_RST, HIGH);
  delay(10);
  digitalWrite(RFM69_RST, LOW);
  delay(10);

  if (!rf69.init()) {
    Serial.println("[ERROR] RFM69 radio init failed");
    while (1);
  }
  Serial.println("[INFO] RFM69 radio init OK!");

  // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM (for low power module)
  // No encryption
  if (!rf69.setFrequency(RF69_FREQ)) {
    Serial.println("[ERROR] setFrequency failed");
  }

  // When using High Power RF69, RFM69HW then the Tx power ishighpowermodule
  // flag MUST be with to TRUE
  rf69.setTxPower(20, true);  // Power range 14-20

  // The encryption key has to be the same as the one in the server
  uint8_t key[] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
  rf69.setEncryptionKey(key);

  pinMode(LED, OUTPUT);

  Serial.print("[INFO] RFM69 radio @");
  Serial.print((int)RF69_FREQ);
  Serial.println(" MHz");
}
```

The main `loop()` function just check if a new message arrives.

If so, it read the message and store it into `buf` buffer.

Then, several `Serial.print()` statement are used to send the data over the serial connexion.

Finally, the sketch sends an **ACK** confirmation message.

```
void loop() {
 if (rf69.available()) {
    // Should be a message for us now
    uint8_t buf[RH_RF69_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (rf69.recv(buf, &len)) {
      if (!len) return;
      buf[len] = 0;
      Serial.print("[DATA](len=" );
      Serial.print(len);
      Serial.print(",RSSI=");
      Serial.print(rf69.lastRssi(), DEC);
      Serial.print(")");
      Serial.print((char*)buf); // Data send by remote is supposed to contains the \r\n

      // Send a reply!
      uint8_t data[] = "ACK";
      rf69.send(data, sizeof(data));
      rf69.waitPacketSent();
      Blink(LED, 50, 1); //blink LED 1 times, 50ms between blinks
    }
  }
  else {
    Blink(LED, 50, 3); //blink LED 3 times, 50ms between blinks
  }
}
```

The `Blink()` function is used to signal error code (blink once when a message received, blink 3 times when having a communication error).

```
void Blink(byte PIN, byte DELAY_MS, byte loops) {
  for (byte i=0; i<loops; i++)  {
    digitalWrite(PIN,HIGH);
    delay(DELAY_MS);
    digitalWrite(PIN,LOW);
    delay(DELAY_MS);
  }
}
```

Compile and upload

Select the proper board in the menu **Tools -> Type of board** : Arduino/Genuino UNO

Select the proper port in the menu **Tools -> Port**

Then press the "upload" button.

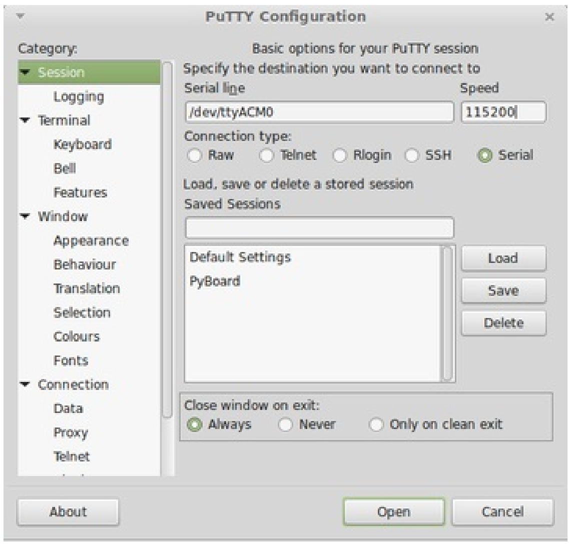
Capturing data to file

Having data available in the Arduino Serial Monitor is great... but capturing it without Arduino would be even better.

Putty

The putty software (available on Windows, Mac, Linux) can be also be used to connect to the Arduino Serial Port interface

Here how it should be configured to capture the data.


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: /dev/ttyACM0

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

Default Settings

PyBoard

Close window on exit: [✓ Always] Never Only on clean exit

Putty also offers some logging capability that may be useful.

Linux command

If you are addict to Linux or Raspberry-Pi board then you can easily view and capture the data with the following commnands.

```
cat /dev/ttyACM0 > output.dat
```

This command will redirect the content of the USB port to a file named `output.dat` .

With Python

The following Python script will capture a serial port (see `baud_rate` variable) and write the content to a file (see `write_to_file_path` variable).

The content of the file is reset when the script is started.

Openning the Serial Port will issue the automatic Reset feature of the Arduino board.

```
import serial

serial_port = '/dev/ttyACM0';
# depend on Serial.begin(baud_rate) in Arduino
baud_rate = 115200;
write_to_file_path = "output.txt";

output_file = open(write_to_file_path, "w+")
ser = serial.Serial(serial_port, baud_rate)
try:
    while True:
        line = ser.readline()
        line = line.encode("utf-8") #ser.readline returns a binary, convert to string
        print(line)
        output_file.write(line)
except KeyboardInterrupt:
    print( 'User abord' )
output_file.close()
ser.close()
```

Other options

You may find many other capture methods from Internet:

- Free Software available on Internet
- Source Code example for your favourite programming language.

Mission 1 - going autonomous

Sommaire

Introduction

Until now, we have worked with the emitter linked to a computer via USB cable.

Using the USB cable was very useful to capture the debugging message into the serial console.

As designed in the code, the Feather **wait for the serial connexion** to start the software.

This means that you cannot make your object flying into the CanSat without packing the computer together with the feather into the can... GLOUPS!!!

Don't panic, we will solve this in a minute ;-)

Removing the "Serial Connection Wait"

The serial connection is only useful when you need to track the debugging messages send by the Feather.

As we are going autonomous... we do not need to wait for the serial connection to be established.

So locate the following lines inside the `setup()` function of your "mission1-serial-radio-capture.ino" sketch (the emitter code running inside the CanSat).

```
void setup() {
  Serial.begin(9600);

  // wait until serial console is open, remove if not tethered to computer
  while (!Serial) { delay(1); }

  ...
}
```

Then place the `while` under comment to disable it.

Proceed by placing a **//** in the front of the `while` instruction.

When done, the code should look to this:

```
void setup() {
  Serial.begin(9600);

  // wait until serial console is open, remove if not tethered to computer
  // while (!Serial) { delay(1); }

  ...
}
```

Compile and upload the modified version to your Feather.

Great! You are ready. The program would now starts without waiting for the Serial Monitor.

Plug the battery

It is now time to plug the Lipo in the appropriate connector.

When the Feather **is plugged via USB**:

- The Feather runs over the USB power.
- The LiPo battery is loaded from the USB.

When the Feather **is unplugged from USB**:

- The Feather is instantaneously powered from the LiPo battery.
- The LiPo battery is now discharging to power up the Feather.

The feather will run until the battery is discharged.

At best, the battery have 4.2V and discharge until 3.0V (when LiPo the protection circuit shutdown the power.

The time it takes to discharge depend on the power requirement of your LiPo.

If the LiPo can store 2500mAh and a project requiring 130mA to run will last after 2500mAh/130mA = 19 Hour of working.

This is an estimate. In real life, the current sink by the project is not constant so energy is not sink out constantly from the LiPo.

Conclusion

You should now be able to unplug the Feather from the USB connector then still receive the telemetric data over the air :-)

CanSat 3D

Sommaire

Introduction

The entire project must fit into a can! The size and dimensions are available in CanSat settlement.

This page contains various resources offered by contest participants.

3D CanSat

One of participant of the previous CanSat 2018 edition did share some 3D models.

Thank to Docopol from Institut Saint Michel for this contribution.

01/04/2019 : On request of DESANG team of St Michel Institute of Bruxelles, this 3D model is removed from the GitHub!

Cansat_BaseModel

This is a standard CanSat accredit on required dimension.

- Diameter: 66mm.
- Height: 115mm.

The hook firmness is good for the parachute.

This can have 3mm wall thickness which offer a good balance between we firmness, available space and weight.

Cansat_1.0

This modeling is an upgrade of the previous one. It includes small edge to insert circular 59mm PCBs.

This model did fly at Elsenborn.

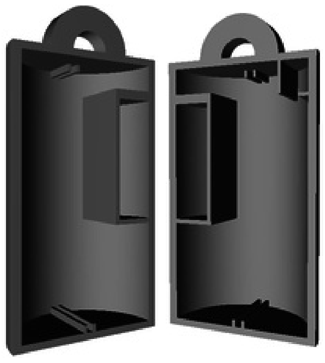
Cansat_2.0

Composed of right and left part, this allows to use a vertical PCB.

The PCB size is 108mm height and 58mm weight.

An additional chamber can be used as ballast. Adding some load may be useful in some circumstance.

This model have been used for CanSat Europe (Açores).



Download Models

All the CanSat 3D models (stl format) https://github.com/mchobby/cansat-belgium/tree/master/cansat-3d can be downloaded from this sub-folder of the GitHub.

01/04/2019 : On request of DESANG team of St Michel Institute of Bruxelles, this 3D model is removed from the GitHub!

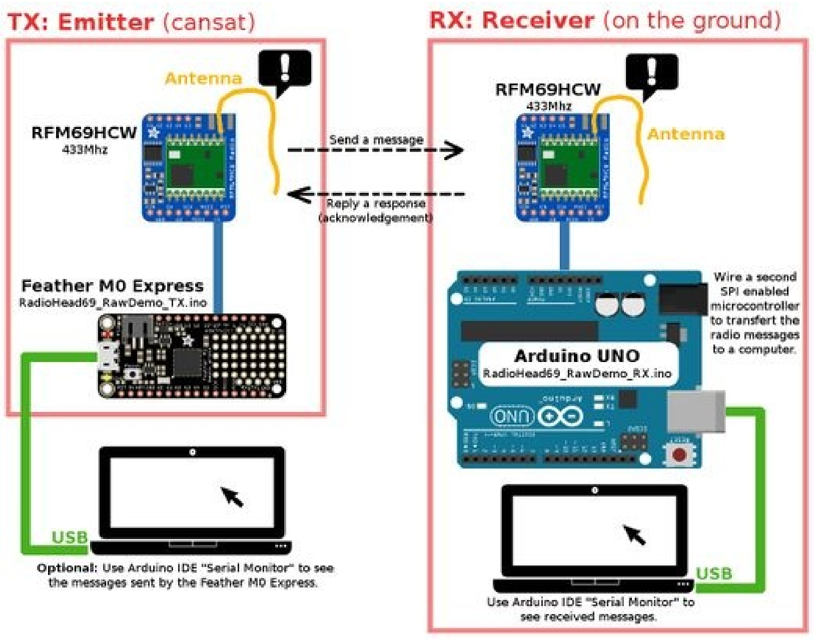
Radio Antenna

Sommaire

Introduction

What ever you do, the antenna is always one of the most sensitive component when working in the radio area.

The following schema shows how to wire the RFM69HCW to transmit data.



TX: Emitter (cansat)
RX: Receiver (on the ground)
Antenna
RFM69HCW
433Mhz
Send a message
Reply a response (acknowledgement)
RFM69HCW
433Mhz
Antenna
Feather M0 Express
RadioHead69_RawDemo_TX.ino
Wire a second SPI enabled microcontroller to transfert the radio messages to a computer.
Arduino UNO
RadioHead69_RawDemo_RX.ino
USB
Optional: Use Arduino IDE "Serial Monitor" to see the messages sent by the Feather M0 Express.
USB
Use Arduino IDE "Serial Monitor" to see received messages.

As you can see, the antenna is made of a simple wire.

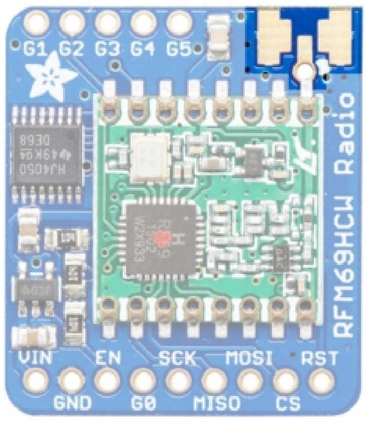
Remove the wire and the data would not transmit at 1 meter of distance, even on the same desk!

Remember, **the antenna is the key to transmit data over a long distance!**

Antenna connector

The radio modules usually offer a spot to solder antenna.

With the RFM69HCW the antenna spot allow you to connect an antenna in 3 different ways.



Later on, you would focus on antenna choice:

- A wire inside the antenna hole (also said a *wire dipole* or *simple dipole*).
- A µFl connector https://shop.mchobby.be/product.php?id_product=1418 to plug antenna.
- A PCB SMA connector https://shop.mchobby.be/product.php?id_product=1419 also to plug other kind of antenna.

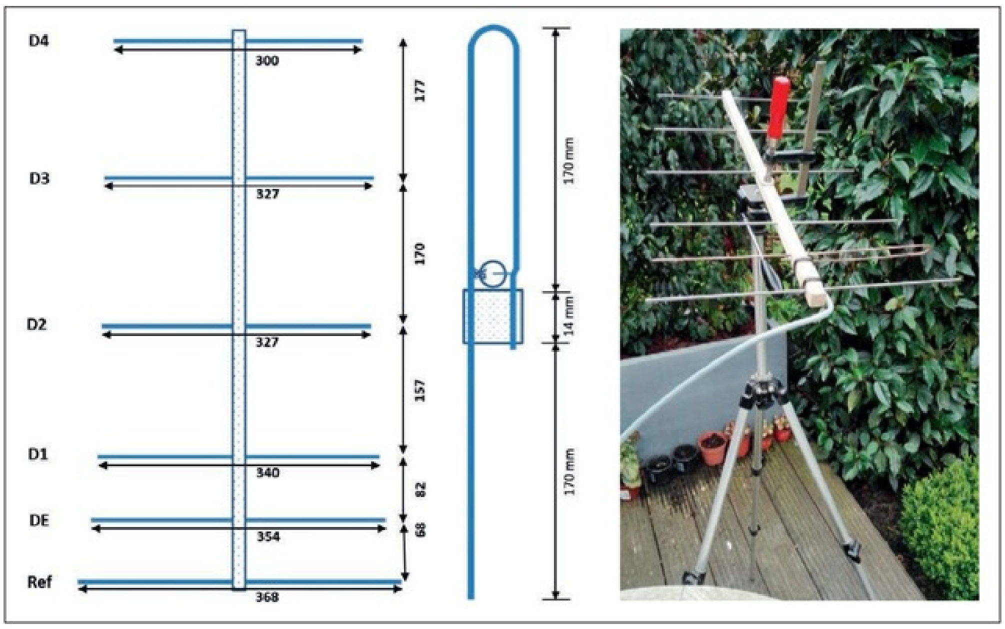
The cheapest antenna is the wire antenna (*simple dipole*) and the best option is the SMA connector (also mode heavy).

The µFl connector https://shop.mchobby.be/product.php?id_product=1418 (also named uFl) is looking to this:

A PCB SMA connector https://shop.mchobby.be/product.php?id_product=1419 is looking to this:

Ground Station - Yagi Antenna

While reading some resources on the QSO magazine https://arduino103.blogspot.com/2018/08/ballons-meteorologiques-tout-savoir-sur.html (Magazine from Belgian Radio Amateur Association written in dutch and french), we did discover a Yagi Antenna tuned for 433Mhz.



Source: QSO Magazine via this publication https://arduino103.blogspot.com/2018/08/ballons-meteorologiques-tout-savoir-sur.html - **click to enlarge**

This antenna was engineered from the "Cheap Yagi https://df.mchobby.be/ballon-meteo/cheapyagi.pdf " (*pdf*), issue from the "Controlled Impedance 'Cheap' Antennas https://www.wa5vjb.com/yagi-pdf/cheapyagi.pdf " article written by Kent Britain WASVJB.

A simple rule of three can be used to adapt the antenna to other frequencies (the antenna was already adapted from a 432 Mhz design).

6 or 11 elements

The Yagi antenna showed here upper does have 6 elements.

From a deeper read of the QSO magazines series https://arduino103.blogspot.com/2018/08/ballons-meteorologiques-tout-savoir-sur.html , we learned that 6 elements Yagi can offer a gain up to 11.2 dBi.

It seems that the 11 elements Yagi (see original document https://df.mchobby.be/ballon-meteo/cheapyagi.pdf) **will double the gain**. Whoaw!!!

About Yagi Antenna

Here some very basic information about Yagi antenna. It brings fundamental concepts that you will deal off with such antenna.

Radiation Pattern

The radiation pattern is the direction or direction**s** where the signal will be emitted (or received).

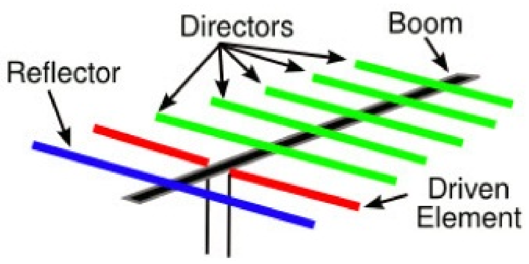
So for a given transmission power, a directional antenna would send the signal further than an hemisphere antenna. Indeed, directional antenna would concentrate all the power is one direction where as hemisphere antenna would spread the same power all around.

The Yagi antenna does support several radiation pattern but "Yagi" is often used in place of "Directional Yagi".

- **Isotropic Antenna** : radiate the same in all the direction.
- **Directional Antenna** (or Beam Antenna) : radiate most of its power in one or more direction. This increase the performances in the direction while reducing the interference coming from the other directions.
- **Omni Directional antenna** : uniformly radiates the power in one plan. With a directive pattern shape in perpendicular plane. This antenna radiates equally in all the directions and have some angle of elevation (dixit elprocus.com https://www.elprocus.com/design-of-yagi-uda-antenna/)
- **Hemispherical antenna** : radiates one half of the hemisphere (the lower or the upper one)

Common Design

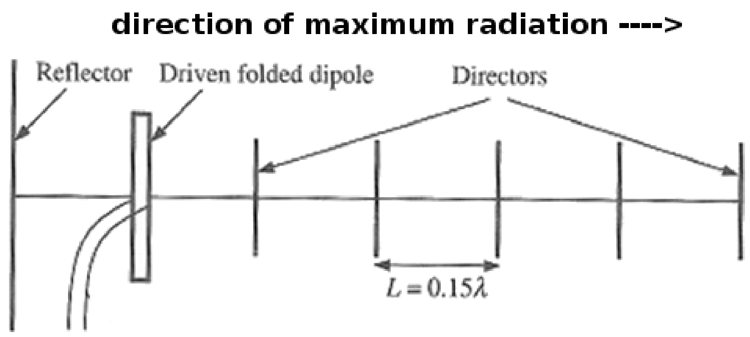
The antenna lengths and spacing are depending on the wavelength to capture.

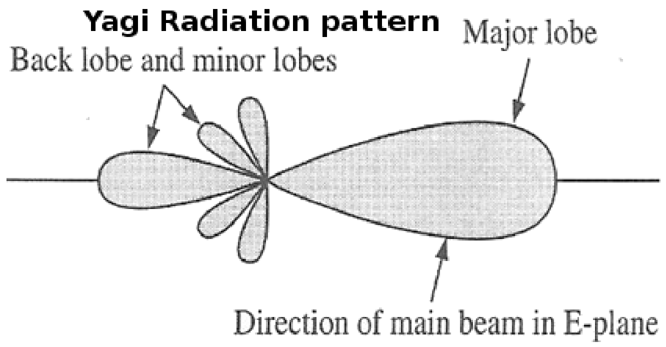


Source: design of Yagi Uda Antenna https://www.elprocus.com/design-of-yagi-uda-antenna/

- **Directors** : used to drive the signal in the given direction. Those directors are usually shorter than driven element (about 5%).
- **Driven element** : is the element that capture or emit the signal.
- **Reflector** : reflect the signal toward (or coming from) the driven element. The reflector is larger than driven element (about 5%).

Because of the antenna design, the maximum of radiation is in the way of director.





Needed Material

A cheap Yagi antenna could be realised with plastic material (or wood) and some very tick wire.

However, the standard Yagy Antenna (and the best ones) are made with aluminium pipes.

Resource

- A closer look at the "black magic" of antennas, how they work, what is essential, and how to test them https://youtu.be/J3PBL9oLPX8 (Youtube)
  Very pratical and affordable video. They explain the power lost (or gain) in the antenna, in the **antenna cable**.
- Yagi Uda Antenna https://www.elprocus.com/design-of-yagi-uda-antenna/ (*elprocus.com*)
- Advantages & disadvantages of YAGI antenna http://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-YAGI-UDA-Antenna.html (*rfwireless-world.com*)
- RFM69HCW antenna hookup guide https://learn.sparkfun.com/tutorials/rfm69hcw-hookup-guide/the-antenna (*Sparkfun, also thread the Dipole antenna*)
- Explaining the Dipole Radioation pattern https://youtu.be/8Eae3lqdtBY (*Youtube*) really great explanation.

CanSat and Antenna

As specified by the CanSat settlement, everything should be contained within a defined volume (the Can).

So, the antenna should also fit into the volume.

However, **once the can is released in the air, your project can deploy an antenna**.

A well designed antenna for sending the data would also reduce the error ratio on the ground station.

Quarter WaveLength Antenna

It is possible to create a small antenna made of a simple thread of Wire.

The thickness is not especially critical but the length is very important.

The ideal Length can be calculated with the following formula:

```
       c
L = -------
     4 x f
```

Where:

- **c** : is the light speed (in m/s)
- **f** : the frequency (in hertz) for the antenna
- 4 : because it's a quarter length antenna.

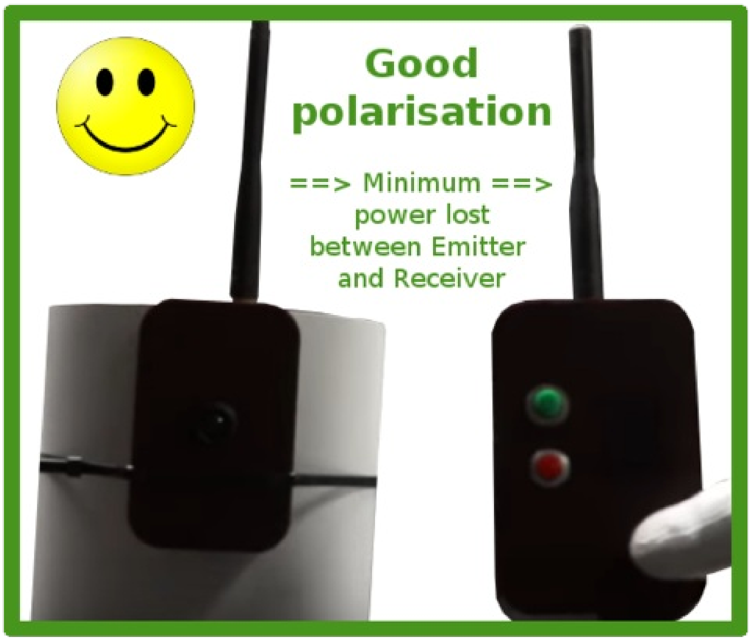
So for a 433 Mhz, this will gives:

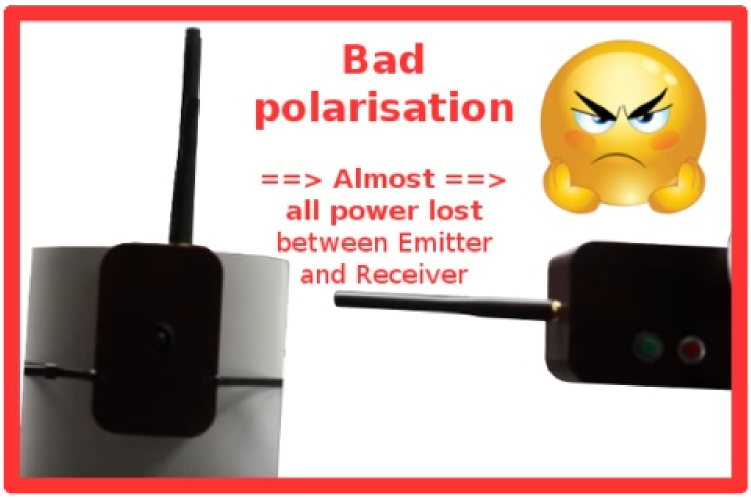
```
          3x10E8
L = -------------- = 0.1732m
      4 x 433*10E6
```

So an antenna length of 17.32 cm.

Quarter WaveLength & Polarization

When using such antenna, you have to think about the polarisation of your signal, Andeas Spiess shows it into its YouTube video 'A closer look at the “black magic” of antennas, how they work, what is essential, and how to test them https://youtu.be/J3PBL9oLPX8 '. If you miss this very simple point, you will lost lot of gain and loose data.





Other Antennas

We encourage you to search antenna with better performance, the Dipole Antenna https://learn.sparkfun.com/tutorials/rfm69hcw-hookup-guide/the-antenna is one of best choice when starting (easy to do with good performance).

Being creative may offers significant advantages in antenna design. A good antenna will maximize the transmitted power to the ground station.

7 Rules for Radio Antennas

- Rule #1: Use short, high quality and thick antenna cables.
- Rule #2: An SWR below 2 is acceptable (less than 11% of power is reflected so we have much of the power available for transmission).
- Rule #3: **Always connect an antenna to the sender** (otherwise 100% of signal is reflected, which may kill the sender)
- Rule #4: Keep the polarization of your antennas the same way.
- Rule #5: The more dBi, the more power in one direction.
- Rule #6: With a proper antenna setup, the distance in air is not an issue if we have a line of sight.
- Rule #7: Longer is not always better for antennas. Smarter is better.

All coming from the famous video of Andreas Spiess https://youtu.be/J3PBL9oLPX8 .

What is SWR?

SWR (*Standing Wave Radio*) measure the performance of signal emitted in the atmosphere. This value is available on the antenna technical datasheet or can be evaluated with appropriate measurement device.

On a radio board, the signal is generated by the transceiver, flowing into the antenna cable and finally through the antenna to be emitted into the atmosphere.

Depending on the antenna, there is some rejection of the signal back to the transceiver. This means than a portion of the signal is not emitted to the air! This is why the SWR measurement is done.

A poor antenna design will have greater SWR (>2) meaning that range of transmission (or reception) will be reduced! Very bad SWR may even damage the transceiver!

The worse case is when there is no antenna, in this case, the rejection ratio is 100%. Nothing is emitted to the atmosphere and all the signal is send back to the transceiver (which may possibly destroy it!

SWR	Description
1.0-1.5	The ideal range! an SWR under 1.5 is really great. Getting under 1.5 (closer of 1) is possible but difficult, you may consider additionnal tuning, other equipment and different mounting location. However dropping under 1.5 (to 1.0) would not increase the performance in significant manner.
1.5-1.9	This range will provide an adequate/acceptable performance. Due to installations, it's sometime impossible to get an SWR under that range. This SWR range often means that your tuned antenna is not mounted in a less-than-ideal location. To troubleshoot, search for paper dealing about "problematic antenna mounting locations". Going to from 2.0 down to 1.5 will really offer noticeable performance improvement.
2.0-2.4	While not good, this likely won't damage your radio with casual use. However, you should definitely try to improve it if you can. SWR in this range is usually caused by a poor antenna mounting location and/or a poor choice of equipment for your specific vehicle. To troubleshoot, you'll likely need to move the mounting location and/or use a more suitable antenna. It's by no means a good tuning job, but will function if you've exhausted all other troubleshooting possibilities.
2.5-2.9	**Do not operate radio in this range**. SWR in this range offer antenna with decreased performance. With range in 2.5 - 2.9 you may even damage the transceiver in case of long period (or frequent) transmitting. This bad SWR range may be caused by poor mounting location, poor equipment. To solve: change your location and/or antenna.
3.0+	**DO NOT OPERATE RADIO IN THIS RANGE**. SWR in this range would have bad performance and this will probably damage the radio when use. You SHOULD NOT transmit with a SWR levels above 3.0. This is almost always the result of a poor ground or incorrectly assembled material (it may also indicates a faulty coax, faulty antenna, SWR meter not properly attached).

The SWR can be measured with appropriate device, it is maybe time to find a Radio Amateur club around your location.

Learn more about:

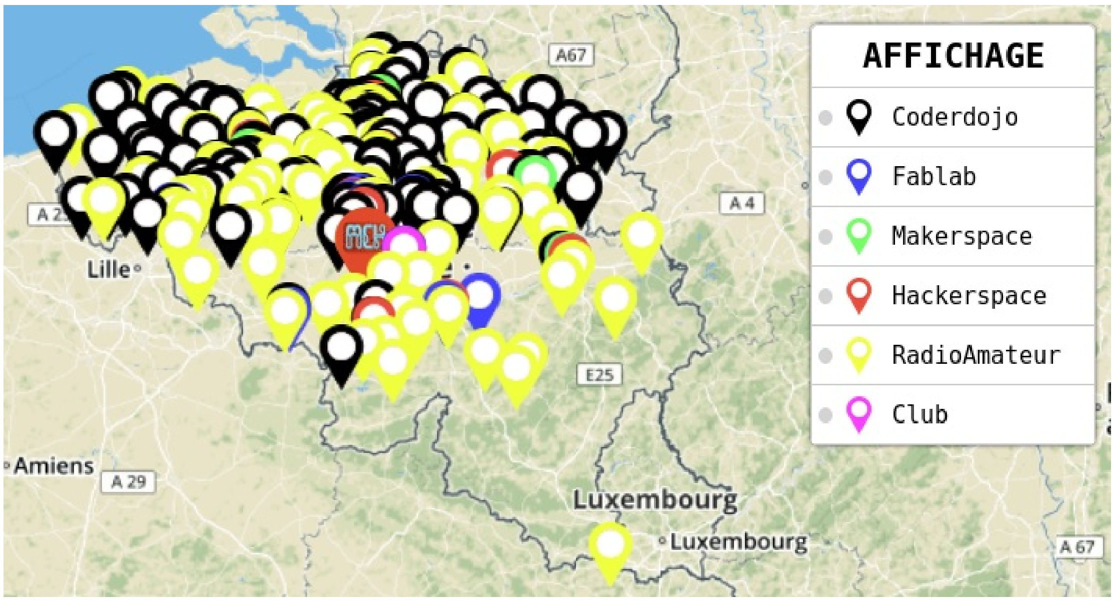
- SWR explained on telecomhall.com http://www.telecomhall.com/what-is-vswr.aspx
- SWR on wearecb.com https://www.wearecb.com/what-is-swr.html .

Getting Help from Radio Amateur

Designing and testing an antenna for long distance communication is an intensive work.

You could find some help from our Radio Amateurs friends.

To locate a Radio Amateur Club http://map.mchobby.be near of your home then have a look the "Maker's Maps" http://map.mchobby.be owned at MC Hobby.



A **yellow mark** indicates the Radio Amateur clubs.

Parachute

Sommaire

Introduction

The CanSat use a parachute to slow it down when getting back to the earth. Otherwise the CanSat would crash!

The parachute is also used to maintain the CanSat in the proper position to properly orientate the antenna! This is essential to receive telemetry data.

The most simple parachutes to create are:

- the "Flat Shape" Parachute
- the "Cross" Parachute

Parachute opening & tenseness

The process of parachute opening and air flow capturing is a violent process. The tenseness on the fabric, wire, hook may be really impressive at the opening.

So you need to select strong fibres and realise strong assemblies.

The ideal materials are those used for human parachute (Nylon Cord, Ripstop Nylon https://en.wikipedia.org/wiki/Ripstop).

Fundamentals parameters

The followings parameters are primary values to use when designing the CanSat's parachute.

- **Mass** : between 300gr and 350gr.
- **Velocity** : between 8 m/s and 11 m/s (about 29 Km/H and 40 Km/H).
- **Drag Coefficient** : the drag coefficient https://en.wikipedia.org/wiki/Drag_coefficient depends on the parachute shape and the fluid used (*Air* in this case).

Drag Coefficient

- Semi Spherical Parachute : 1.5 (narom), 0.77 (in the table below)
- Cross Shape Parachute : 0.8
- Flat, Hexagone Parachute : 0.8

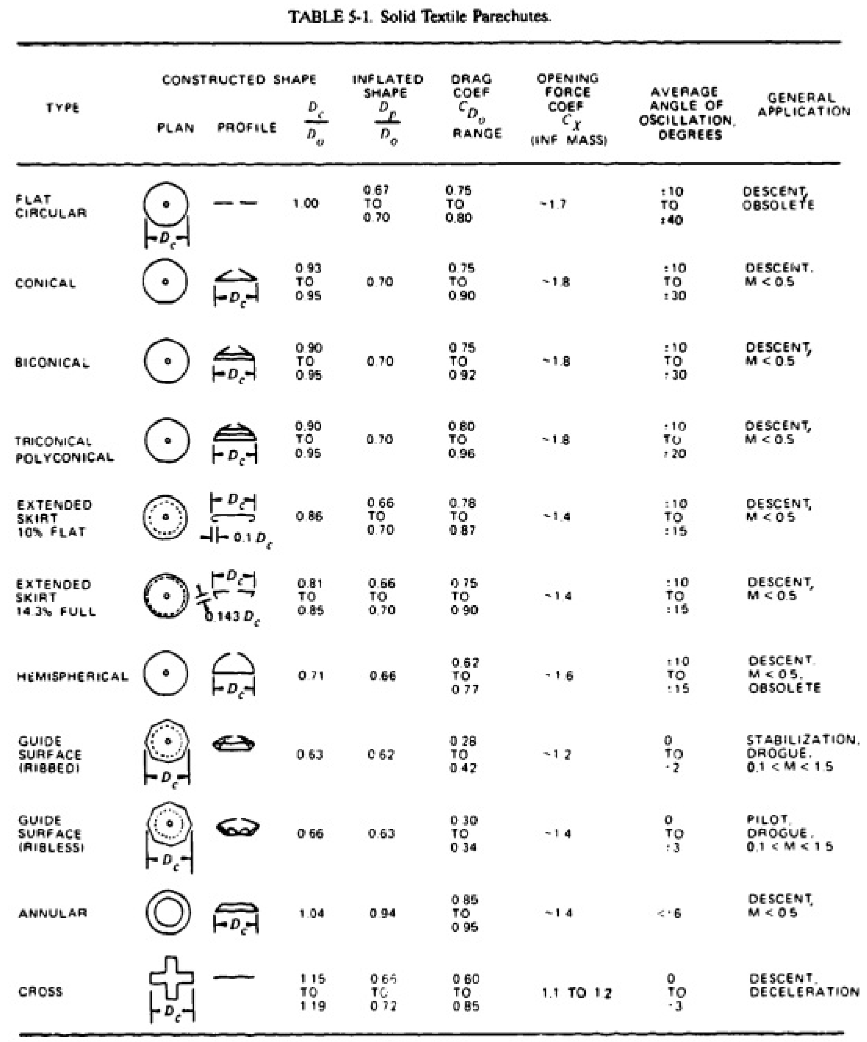


TABLE 5-1. Solid Textile Parechutes.

TYPE	CONSTRUCTED SHAPE PLAN	CONSTRUCTED SHAPE PROFILE	CONSTRUCTED SHAPE $\frac{D_c}{D_o}$	INFLATED SHAPE $\frac{D_p}{D_o}$	DRAG COEF C_{D_o} RANGE	OPENING FORCE COEF C_x (INF MASS)	AVERAGE ANGLE OF OSCILLATION, DEGREES	GENERAL APPLICATION
FLAT CIRCULAR	D_c		1.00	0.67 TO 0.70	0.75 TO 0.80	~1.7	±10 TO ±40	DESCENT, OBSOLETE
CONICAL		D_c	0.93 TO 0.95	0.70	0.75 TO 0.90	~1.8	±10 TO ±30	DESCENT, M < 0.5
BICONICAL		D_c	0.90 TO 0.95	0.70	0.75 TO 0.92	~1.8	±10 TO ±30	DESCENT, M < 0.5
TRICONICAL POLYCONICAL		D_c	0.90 TO 0.95	0.70	0.80 TO 0.96	~1.8	±10 TO ±20	DESCENT, M < 0.5
EXTENDED SKIRT 10% FLAT		D_c 0.1 D_c	0.86	0.66 TO 0.70	0.78 TO 0.87	~1.4	±10 TO ±15	DESCENT, M < 0.5
EXTENDED SKIRT 14.3% FULL		D_c 0.143 D_c	0.81 TO 0.85	0.66 TO 0.70	0.75 TO 0.90	~1.4	±10 TO ±15	DESCENT, M < 0.5
HEMISPHERICAL		D_c	0.71	0.66	0.62 TO 0.77	~1.6	±10 TO ±15	DESCENT, M < 0.5, OBSOLETE
GUIDE SURFACE (RIBBED)	D_c		0.63	0.62	0.28 TO 0.42	~1.2	0 TO ±2	STABILIZATION, DROGUE, 0.1 < M < 1.5
GUIDE SURFACE (RIBLESS)	D_c		0.66	0.63	0.30 TO 0.34	~1.4	0 TO ±3	PILOT, DROGUE, 0.1 < M < 1.5
ANNULAR		D_c	1.04	0.94	0.85 TO 0.95	~1.4	<±6	DESCENT, M < 0.5
CROSS	D_c		1.15 TO 1.19	0.65 TO 0.72	0.60 TO 0.85	1.1 TO 1.2	0 TO ±3	DESCENT, DECELERATION

Source: Parachute Size Estimator https://www.launchwithus.org/lwu-blog/2016/02/17/parachute-size-estimator-for-high-altitude-balloons

Estimate the Drag Coefficient

You can make a drop test from a given height of your CanSat with the parachute.

When terminal velocity is known (measured), you can deduce the *Drag Coefficient* from the estimated velocity in free chute (if the CanSat were not equipped with a parachute).

Flat Circular Parachute

The Flat Parachutes are extremely common in the hobby rocketry fields. Thanks to their simple design, they are cheap and easy to manufacture. They also offers reliable parachute.

The trade-off is the Draft Coefficient (Cd) which is not as high for a given cloth diameter. It is hard to go wrong with a flat circular (or flat hexagonal) parachute.

The typical drag coefficients range goes from 0.75 to 0.80.

You may also find some informations about flat parachute on this NAROM page https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/flat-parachute-design/ .

Parachute Design @ NAROM

The NAROM site in Norway https://www.narom.no did publish "The CanSat Book" containing lot of useful information.

The most interesting parts concerns the Parachute Design https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/ .

- Parachute Design Calculations https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/parachute-design-calculations/
- Descent Physics https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/descent-physics/
- Semi-spherical parachute Design https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/semi-spherical-parachute-design/
- Cross Parachute Design https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/cross-parachute-design/
- Flat Parachute Design https://www.narom.no/undervisningsressurser/the-cansat-book/the-primary-mission/parachute-design/flat-parachute-design/

Parachute Design @ Esero Luxembourg

The Esero Luxembourg http://www.cansat.lu site did publish ressources for cansat contest.

- Parachute calculation https://youtu.be/J-HmIxQ10ec?t=11107 *YouTube*
  Excellent introduction with fundamental concept to some advance parachute calculation. A Great ressource.

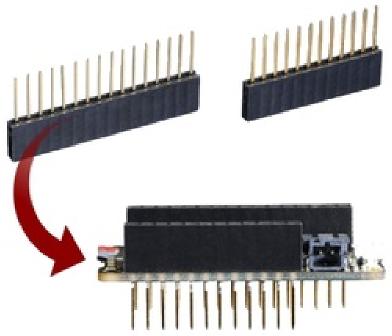
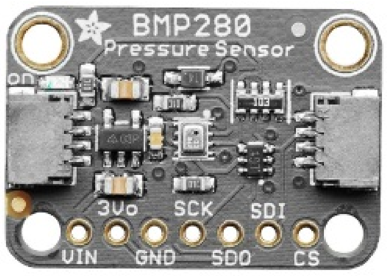
Resources

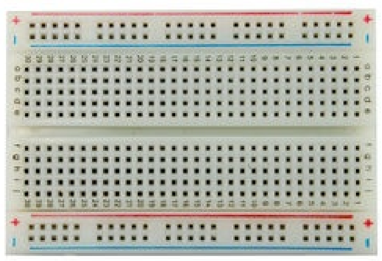
- Parachute Design and Construction https://www.nakka-rocketry.net/paracon.html by Richard Nakka
- Parachute Size Estimator tool https://gallery.mailchimp.com/c4ab65df2b61279e33d7ee72b/files/HAB_Parachute_Size_Estimator_v1.0.xlsx using imperial unit system (Launchwithus.org https://www.launchwithus.org/lwu-blog/2016/02/17/parachute-size-estimator-for-high-altitude-balloons)
  See this article which explains how to use it https://www.launchwithus.org/lwu-blog/2016/02/17/parachute-size-estimator-for-high-altitude-balloons .
- The Mathematics of Parachute https://www.sunward1.com/imagespara/The%20Mathematics%20of%20Parachutes%28Rev2%29.pdf (pdf).

Shopping

Need to refill your Cansat kit with some items ?

Here is the complete product list of boards enclosed inside the kit.

	Description	Quantité
Feather M0 Express	New Arduino M0 compatible on a standard platform for embedded project. Also compatible with CircuitPython. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=1119	1
Feather Stacking Headers 	Plug your feather or prototype wing on breadboard and still having a female connector under the hand. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=832	1
Feather Prototyping Wing	Prototyping board for feather platform. Create your own extension board (wing) by soldering connectors and components. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=861	1
USB A/microB 1m cable	Can be used to plug your feather on a computer to program it or to reload the Lipo. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=145	1
BMP280 Barometric pressure sensor 	Easily evaluate pressure, altitude and temperature. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=1118	1
TMP36 - analog temperature sensor	Transform the sensor voltage read on analog input into an easy-to-read temperature. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=82	1

RFM69HCW Transceiver Radio	Transport data over long distance with packet radio. One breakout act as emitter, the second one as receiver. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=1390	2
Lithium Polymer Battery 0.8 Ah	Transform the Feather into an autonomous plateform with this 800mAh Lipo. Less power, smaller and lighter. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=1302	1
Lithium Polymer Battery 1.3 Ah	Transform the Feather into an autonomous plateform with this 1300mAh Lipo. More power but bigger and bit heavier. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=277	1
Half Size Breadboard 	Solderless breadboard are used for fast prototyping. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=53	1
Multi-functional breadboard wires	Set of wires with plug that can be modified from female to male. disponible ici chez MCHobby http://shop.mchobby.be/product.php?id_product=82	1