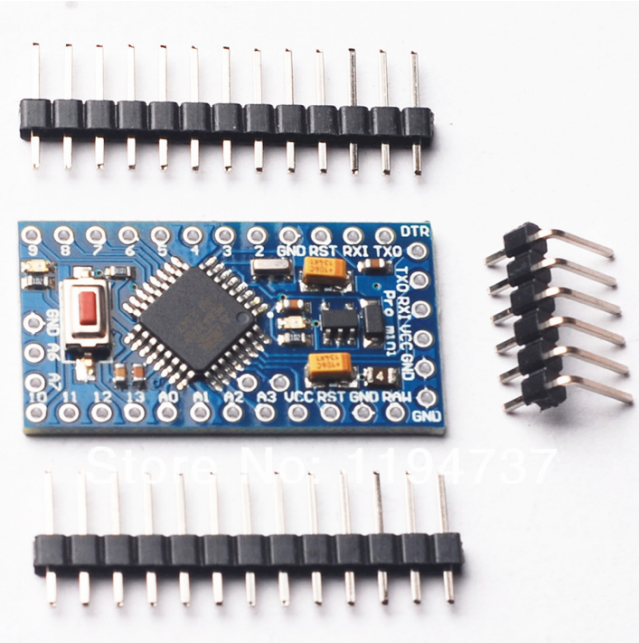
Low-cost LoRa IoT device: A step-by-step tutorial

Prof. Congduc Pham
http://www.univ-pau.fr/~cpham
Université de Pau, France

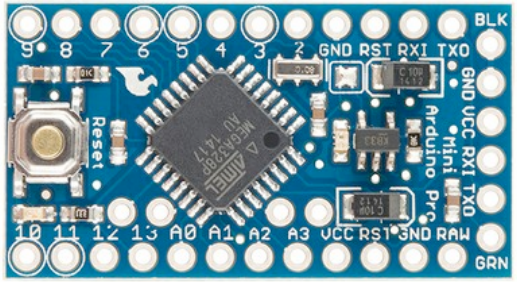
Contents

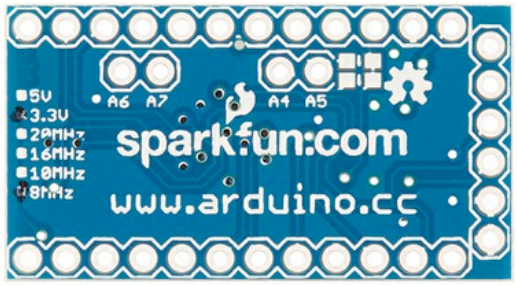
- We will show how to build a low-cost IoT device for sensing surveillance applications using LoRa radio technology
- The gateway part will be shown in a separate tutorial
- The target hardware platform is an Arduino Pro Mini in its 3.3v, 8MHz version: the original from Sparkfun or a clone from various providers
- Let's get started…

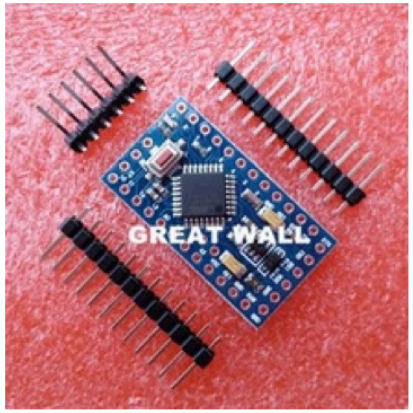
Assembling the hardware



Get the Arduino board







With the bootloader 1pcs **pro mini** atmega328 **Pro Mini** 328 **Mini** ATMEGA328 3.3V/8MHz for **Arduino**

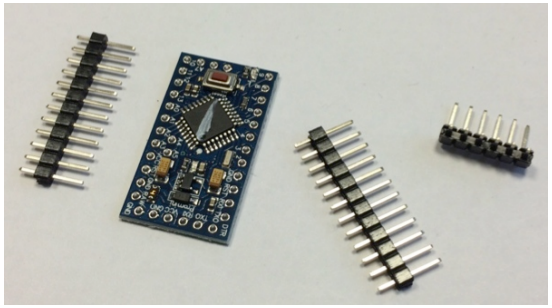
GREAT WALL Electronics Co., Ltd.

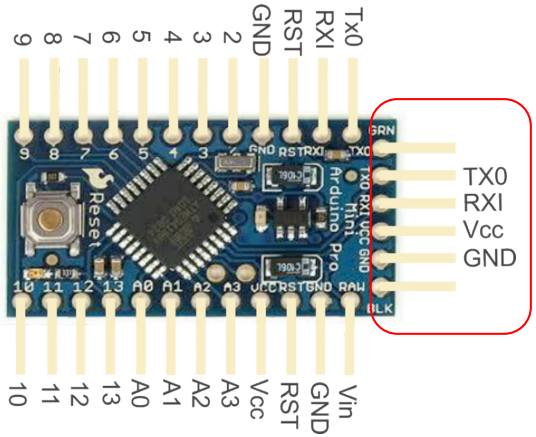
Chat now!

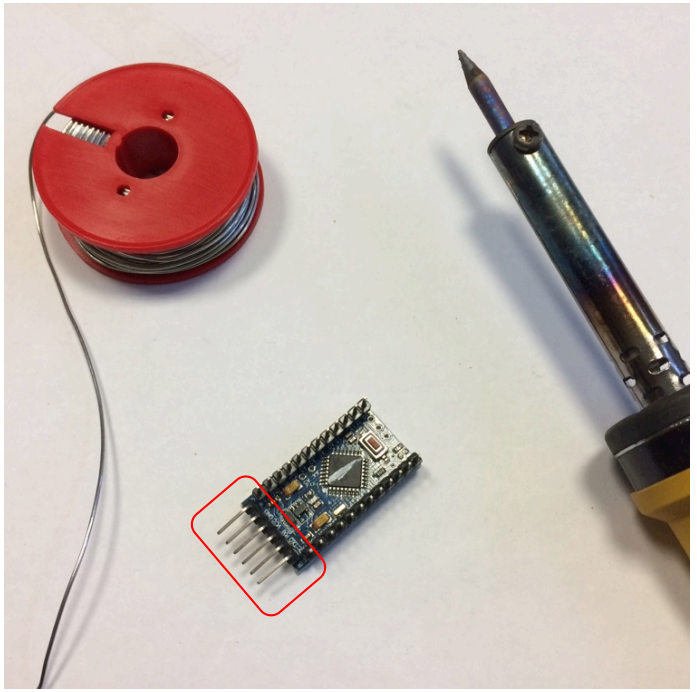
Be sure to get the 3.3v and 8MHz version

You can get the original board designed by Sparkfun or get one of the various clones available mainly from Chinese manufacturer. The last solution is very cost-effective as the Pro Mini board can be obtained for less than 2€ a piece. Some boards may not be working as reported by some people but in my own experience, all the boards I got from Chinese manufacters have been working great.

Prepare the board (1)

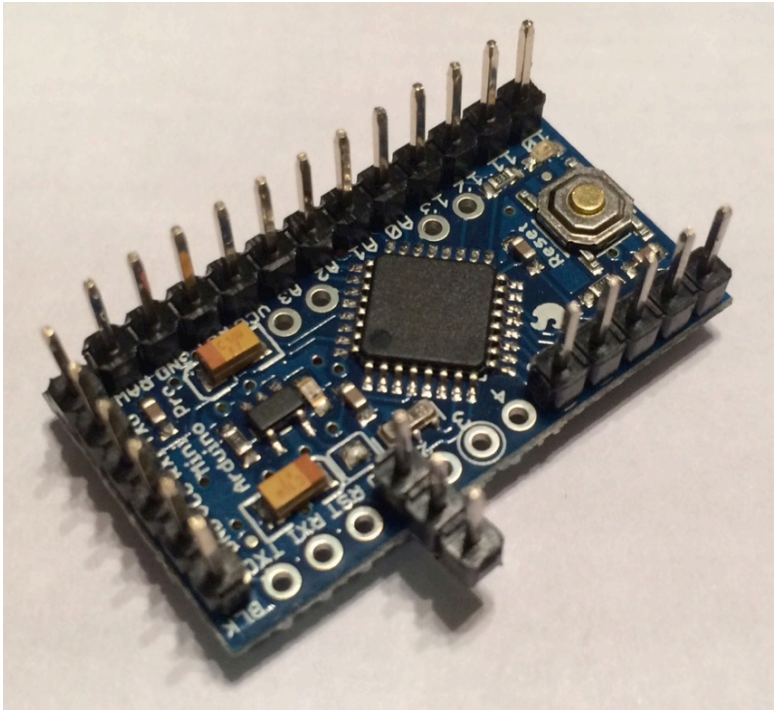


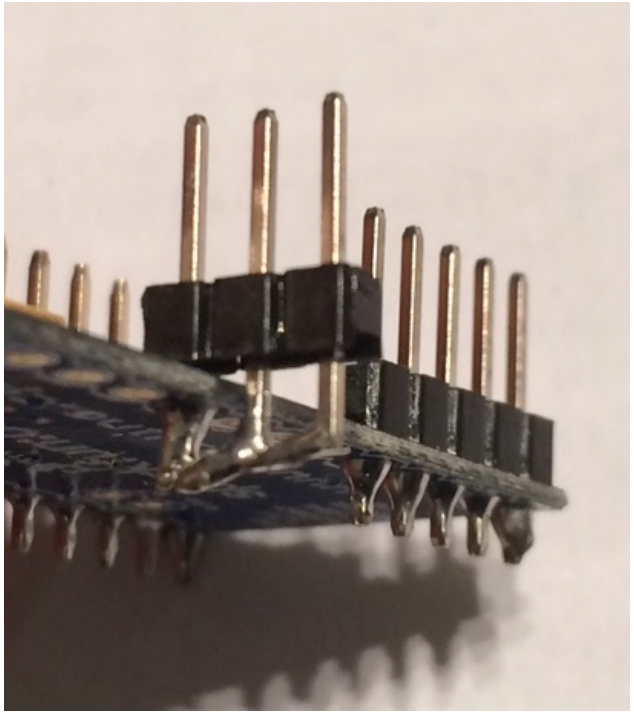




When you receive the board, it will probably come with the appropriate header pins that must be soldered to the board. Just use a regular soldering station to solder the header pins to the board. The 6-pin header on one side of the board (see red rectangle) will be used to connect an FTDI cable to program the board. This will be explained in the « software » section.

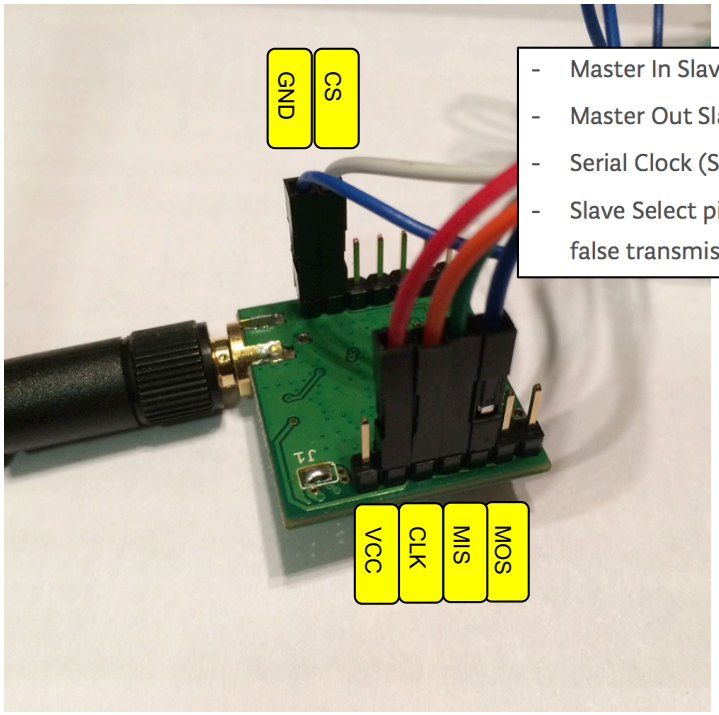
Prepare the board (2)



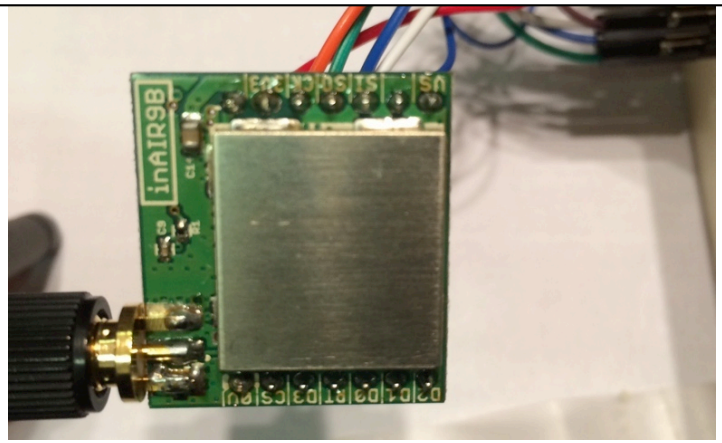


You can solder a row of header pins (left) on the ground pin (GND) in order to have several ground pins for the various sensors that will be connected to the device. But don't forget to link all the pins together to get the GND signal on all the pins (right).

NOW THE RADIO MODULE (1)

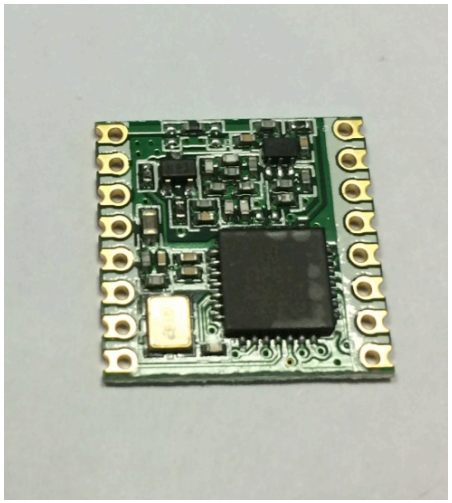


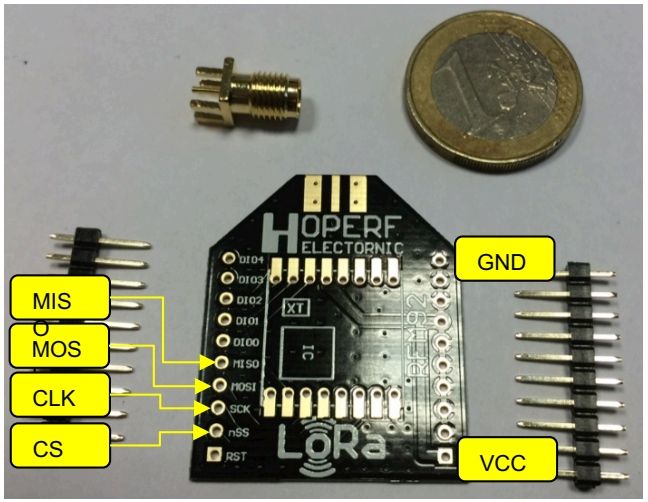
- Master In Slave Out (MISO) - The Slave line for sending data to the master,
- Master Out Slave In (MOSI) - The Master line for sending data to the peripherals,
- Serial Clock (SCK) - The clock pulses which synchronize data transmission generated by the master, and
- Slave Select pin - allocated on each device which the master can use to enable and disable specific devices and avoid false transmissions due to line noise.



If you go for the inAir9 from Modtronix, then the header pins can come fully assembled. Order with the 6mm header pins to have enough length to connect F/F breadboard cables (left). Connect the SPI pins with the F/F cables. Try to use different colors. I use the following colors: MOSI (blue), MISO (green), CS (white), CLK (orange). Then connect also the VCC (red) and the GND (black or any other dark color) of the radio board.

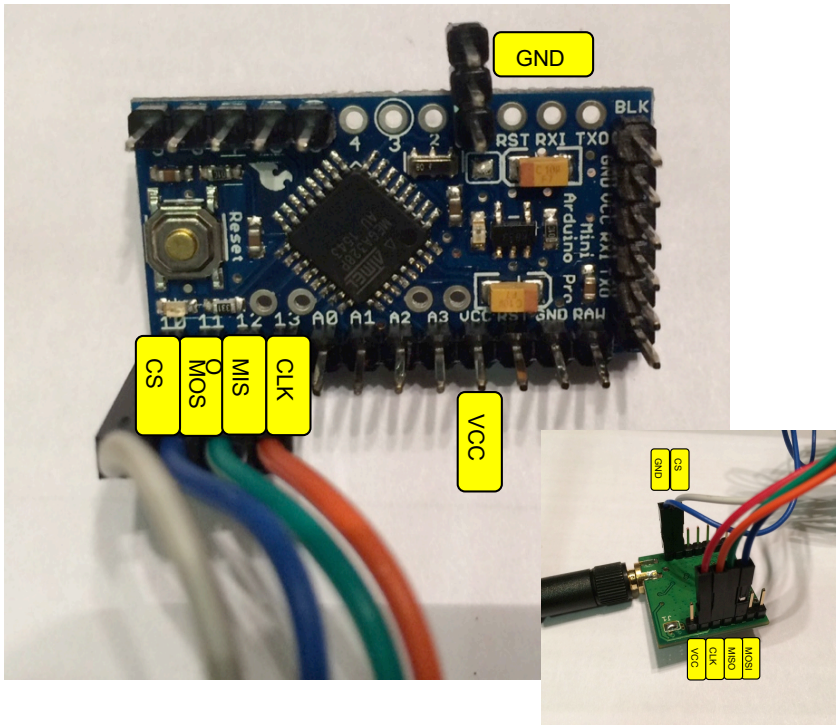
NOW THE RADIO MODULE (2)

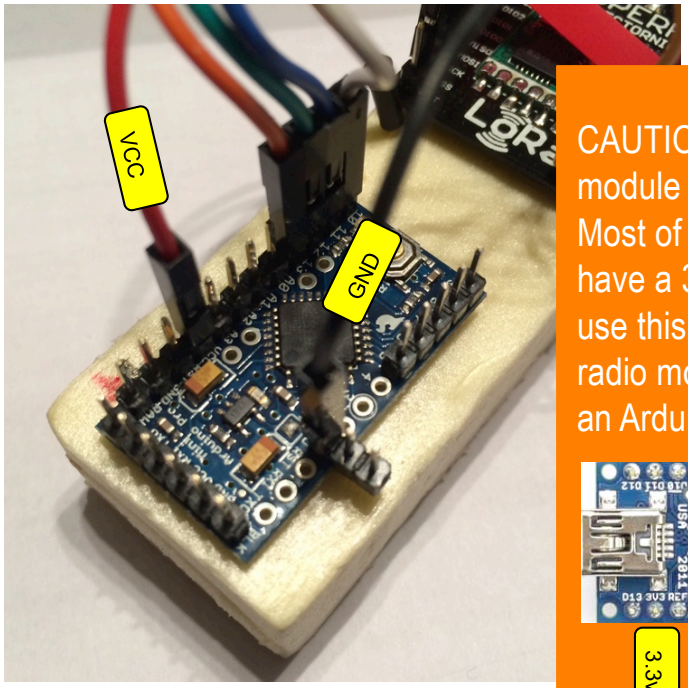




If you take the HopeRF RFM 92W/95W you will need the adaptor breakout and you have to go though some delicate but simple soldering tasks! It is not difficult but you have to train a bit before! Then, like for the inAir9, use F/F breadboard cable to connect the SPI pins, using different colors as explained previously.

Connecting the radio module





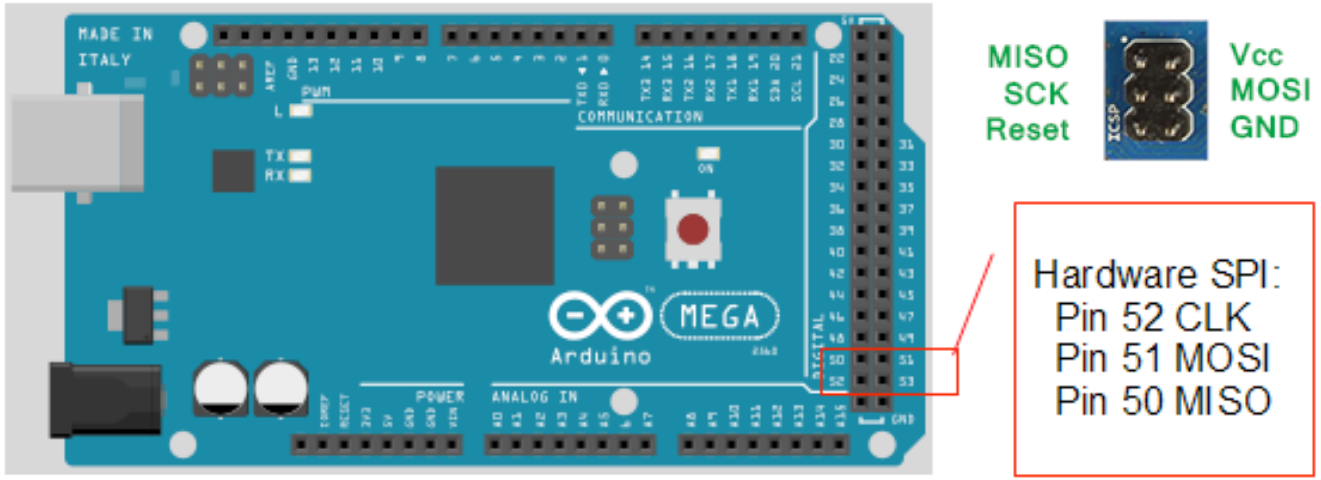
CAUTION: the radio module needs 3.3v, not 5v! Most of Arduino boards have a 3.3v pin so you can use this pin to power the radio module. Example wth an Arduino Nano:

3.3v

Now, just connect the corresponding SPI pins of the radio module to the SPI pins on the board. MOSI (blue) is pin 11, MISO (green) is pin 12, CS (white) is pin 10 and CLK (orange) is pin 13 (left picture). Then connect also the VCC (red) and the GND (black) of the radio board to the VCC and the GND of the board (right picture). The VCC of the board gets 3.3v from the on-board voltage regulator.

Connecting the radio module on other boards

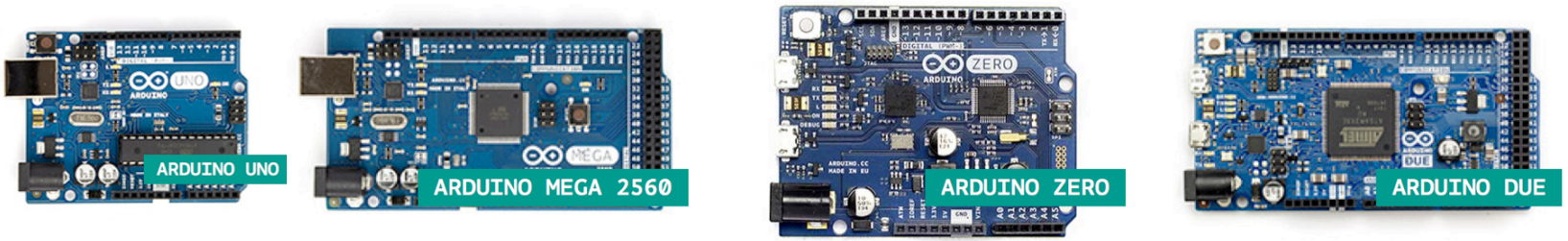
- If you use other Arduino boards, check on the specific Arduino web page where are the SPI pins. Example with Arduino MEGA:



SPI: 50 (MISO), 51 (MOSI), 52 (SCK). These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Arduino /Genuino Uno and the old Duemilanove and Diecimila Arduino boards.

Connecting the SPI CS

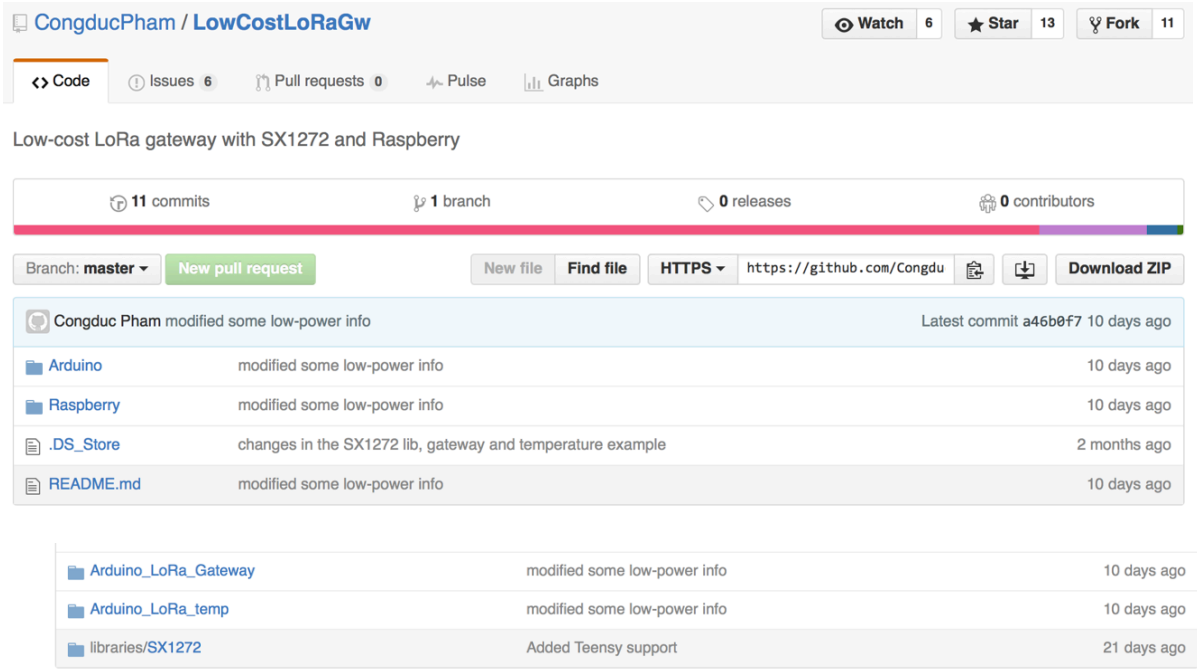
- The SPI Chip Select (CS) or SS (Slave Select) of the radio module needs to be connected to a specific pin on the board.
- For boards with UNO format, CS -> pin 2



- For small form factor boards, CS-> pin 10

Getting, compiling & uploading the software

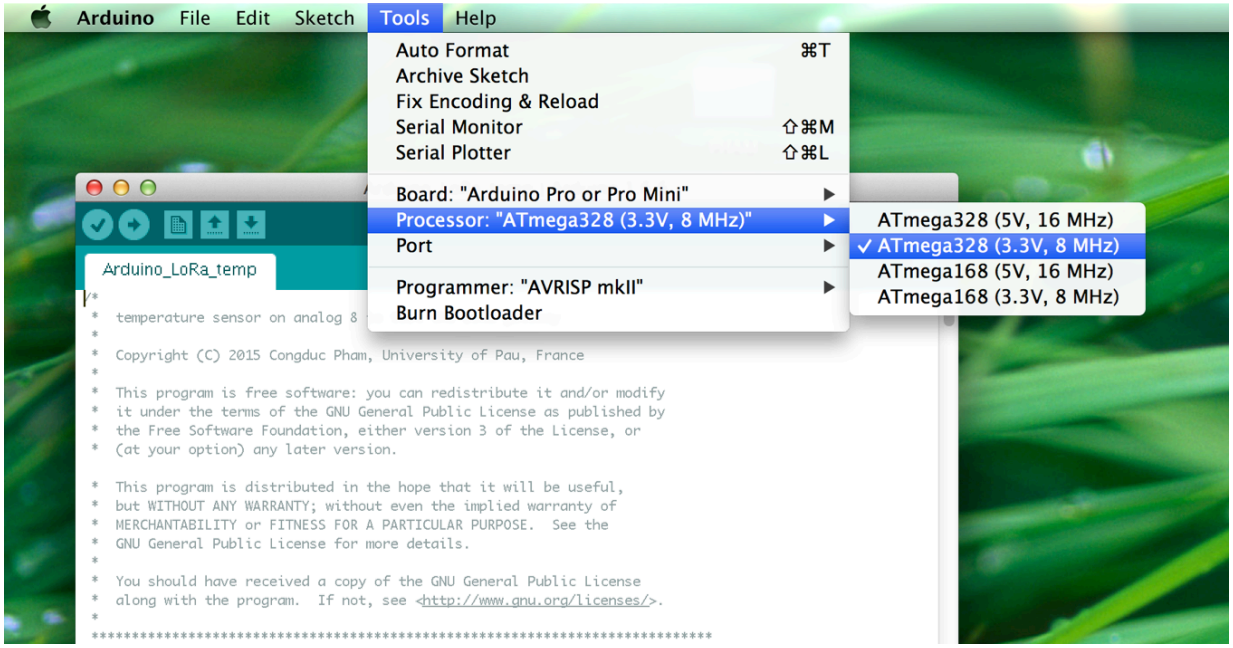
GETTING THE SOFTWARE



Fisrt, you will need the Arduino IDE 1.6.6 or later (left). Then get the LoRa library from our github: https://github.com/CongducPham/LowCostLoRaGw (right).

Get into the Arduino folder and get both Arduino_LoRa_temp and SX1272 folder. Copy Arduino_LoRa_temp into your “sketch” folder and SX1272 into “sketch/libraries”

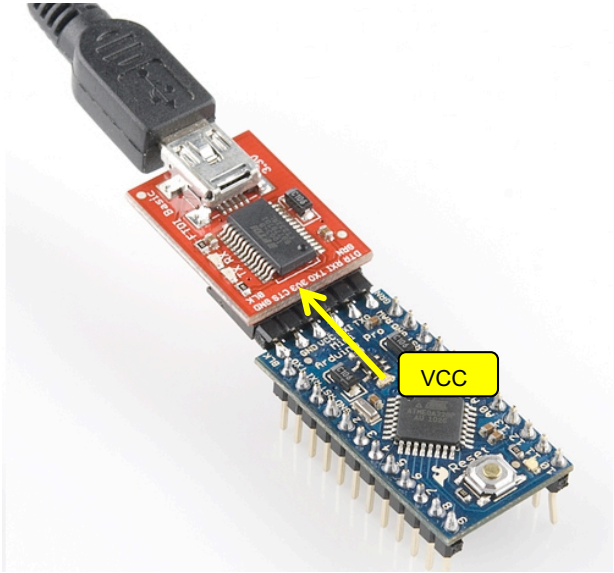
Compiling



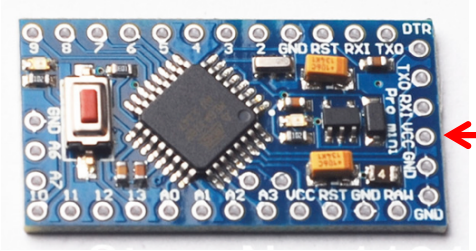
Open the Arduino_LoRa_temp sketch and select the Arduino Pro Mini board with its 3.3V & 8MHz version.

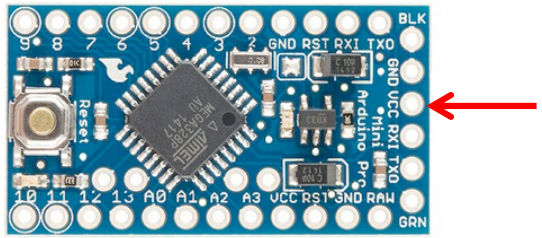
Then, click on the « verify » button

Uploading (1)



Some clone version, check the VCC pin

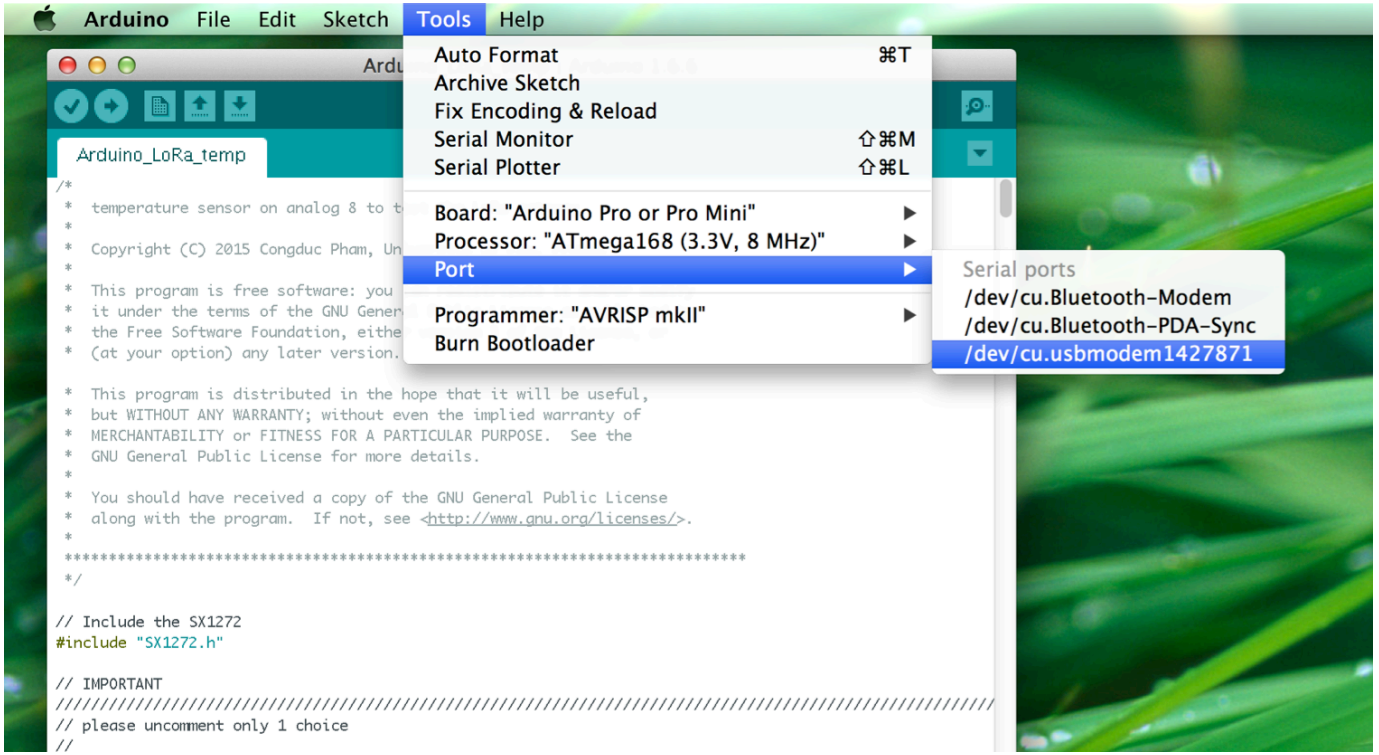




Original Sparkfun version

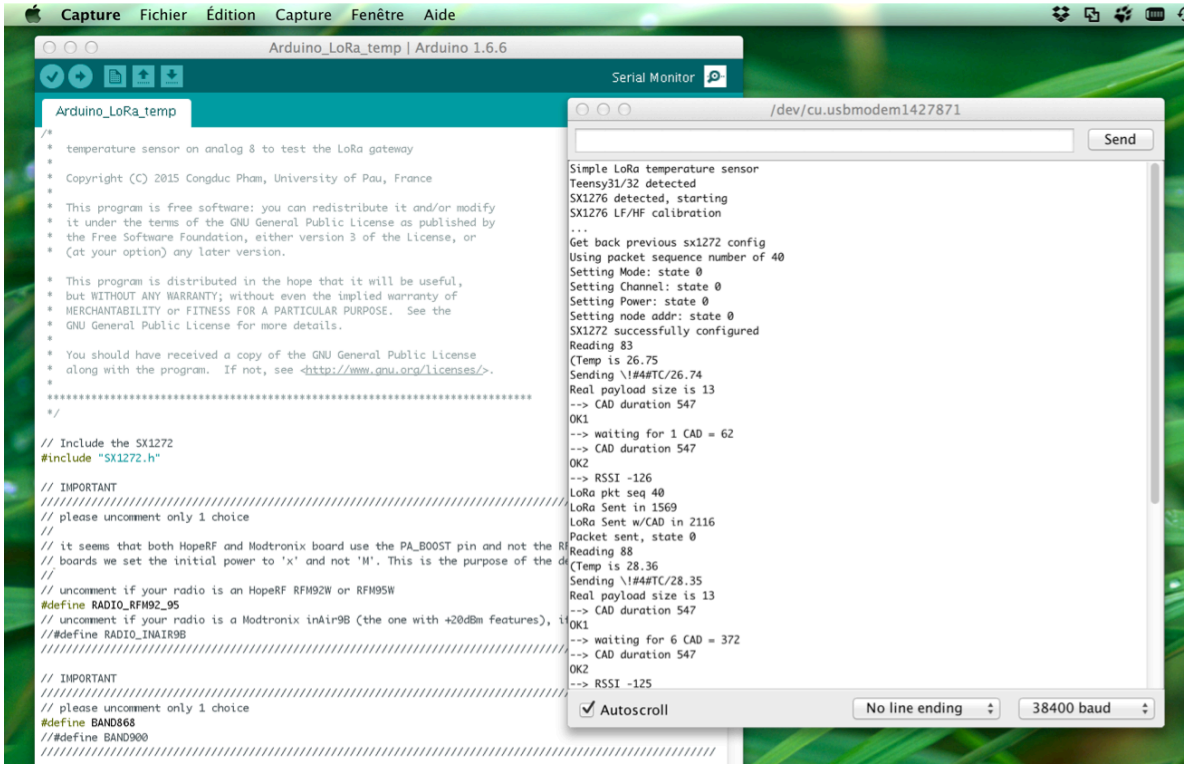
For the Pro Mini, you need to have an FTDI breakout cable working at 3.3v level (there is also 5v version but our advised Pro Mini version is running at 3.3v to reduce energy consumption). Be careful, on some low-cost Pro Mini version (Chinese manufacturer for instance) the pins may be in reversed order. The simplest way in to check the VCC pin and make it to correspond to the VCC pin of the FTDI breakout.

Uploading (2)



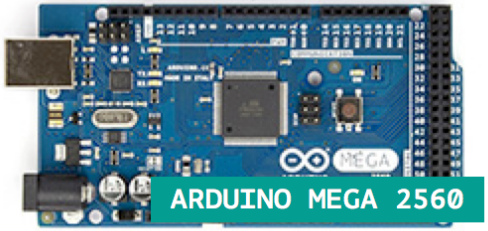
Connect the USB end to your computer and the USB port should be detected in the Arduino IDE. Select the serial port for your device. It may have another name than what is shown in the example. Then click on the « upload » button

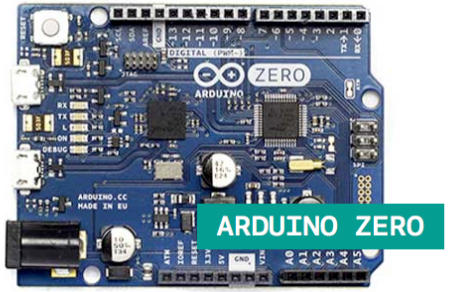
Serial monitor

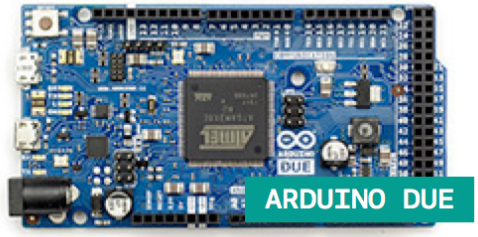


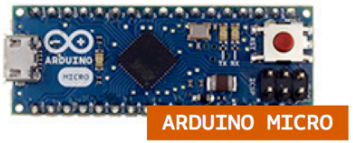
You can see the output from the sensor if it is connected to your computer. Use the Arduino IDE « serial monitor » to get such output, just to verify that the sensor is running fine, or to debug new code. Be sure to use 38400 baud.

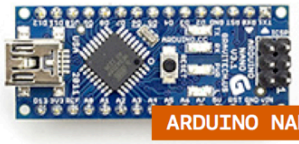
MORE PLATFORMS ARE SUPPORTED!

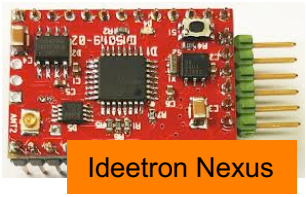




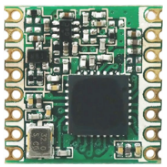




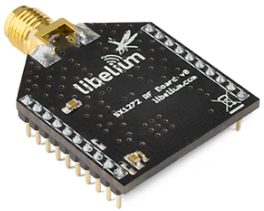




LoRa radios that our library already supports

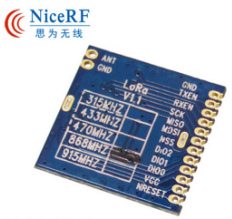


HopeRF RFM92W/95W



Libelium LoRa

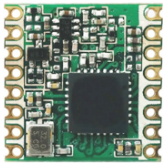
Modtronix inAir9/9B



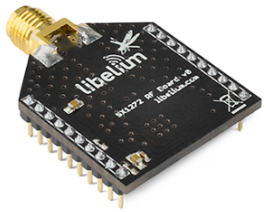
NiceRF LoRa1276

Long-Range communication library
(mostly sending functions)

LoRa radios that our library already supports

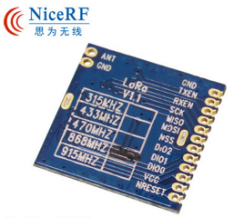


HopeRF RFM92W/95W

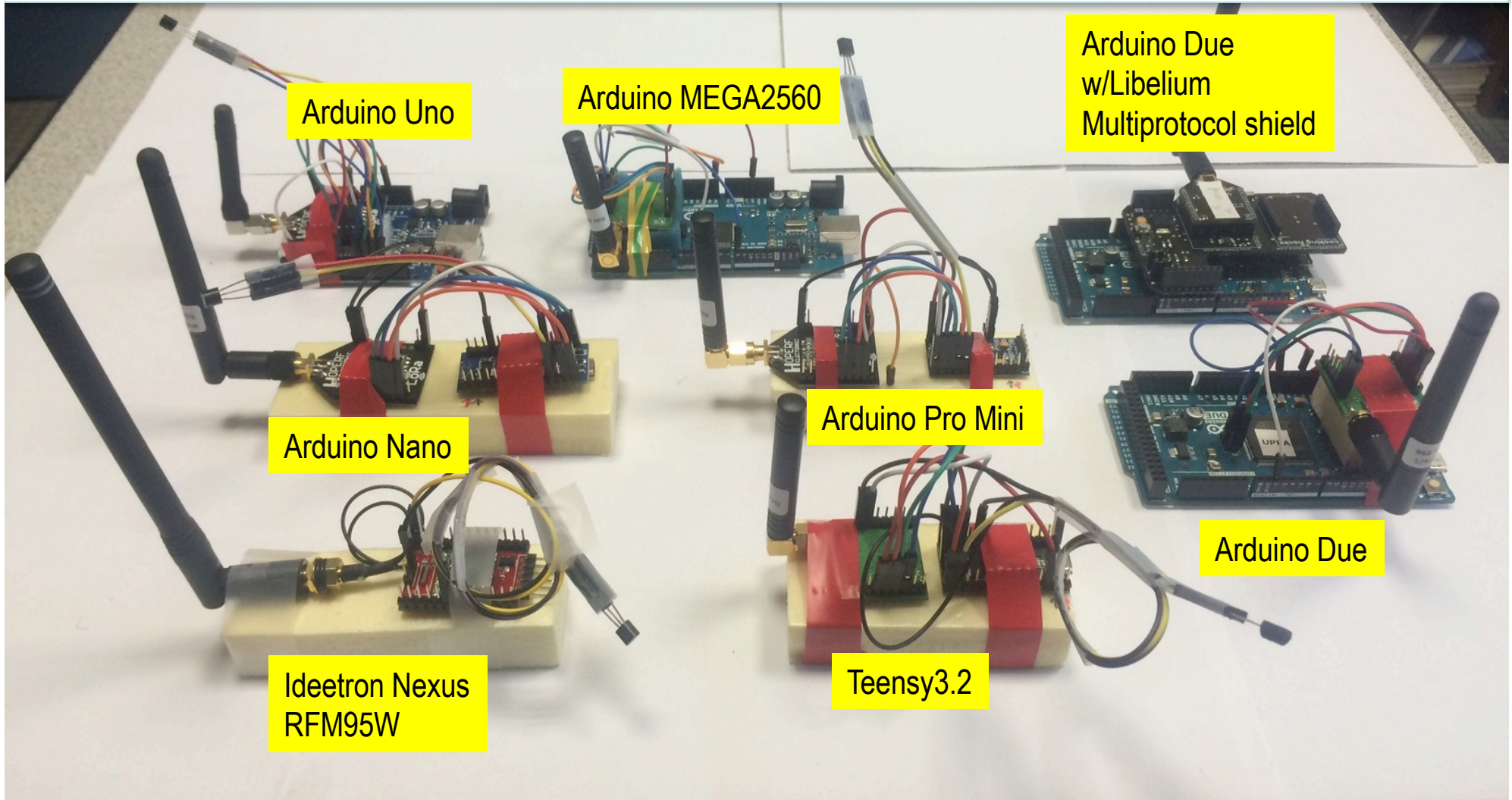


Libelium LoRa

Modtronix inAir9/9B

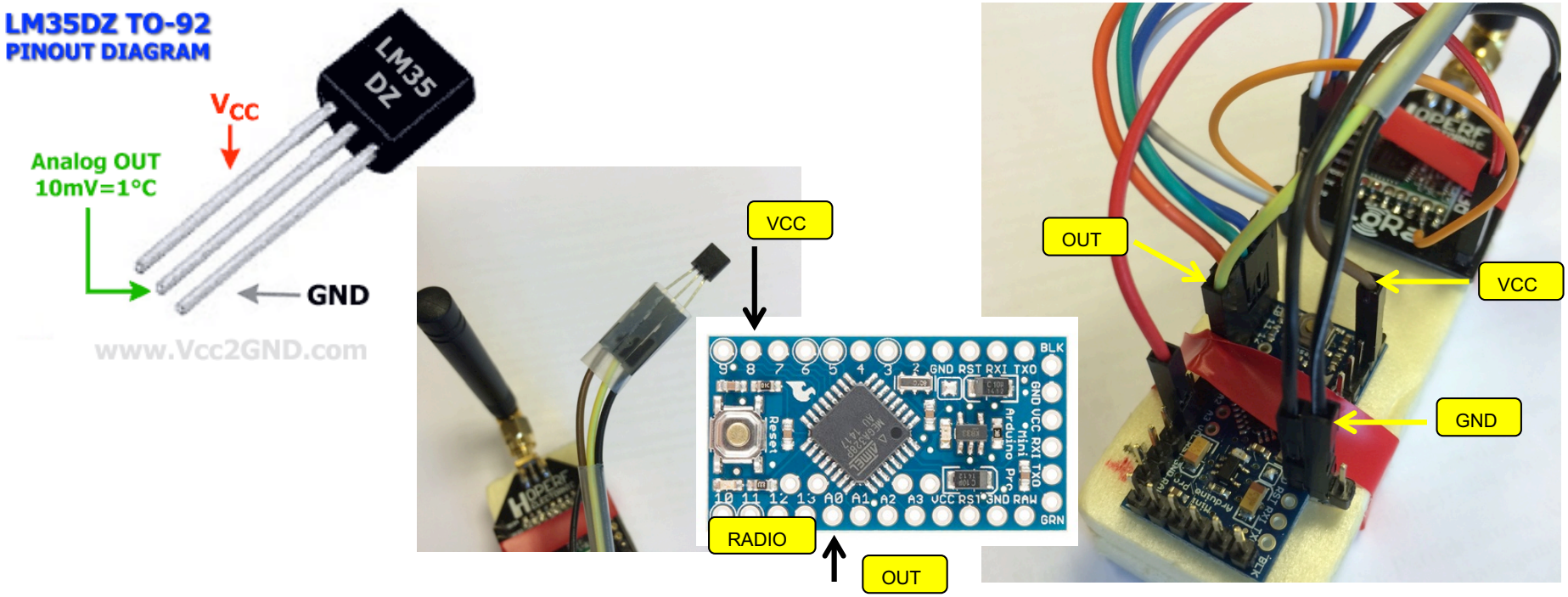


NiceRF LoRa1276



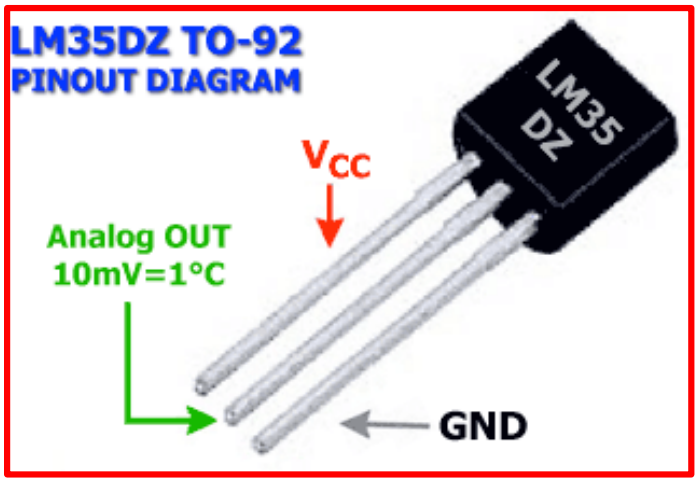
Adding a physical sensor

Temperature sensor



For the moment, there is no physical sensor connected to the board, so you will probably get random value when running the sensor. The Arduino_LoRa_temp example uses the LM35DZ physical sensor to get the ambient temperature. The GND should be connected to one of the board's GND, the VCC should be connected to the digital pin 8 and the OUT pin should be connected to the analog A0 pin.

Understanding analog output

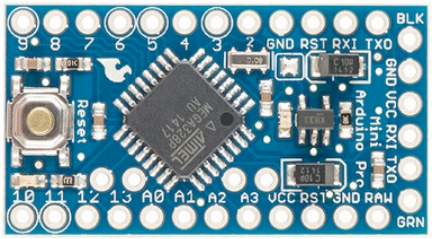


Vcc is 3.3V (the output of digital 8 to power the sensor)

If 0 means 0V and 1024 means 3300mV (10-bit resolution) then 3300mV/1024=3.22mV is the granularity of the measure

A digital value of 100 means 100*3.22mV=322mV
If the sensor output is 10mV/1°C then the physical temperature is 322mV/10mV=32.2°C

READING ANALOG PIN VALUE




```
// sensor output connected to A0 analog pin

value = analogRead(A0);

// now need to convert to Celcius degree
```

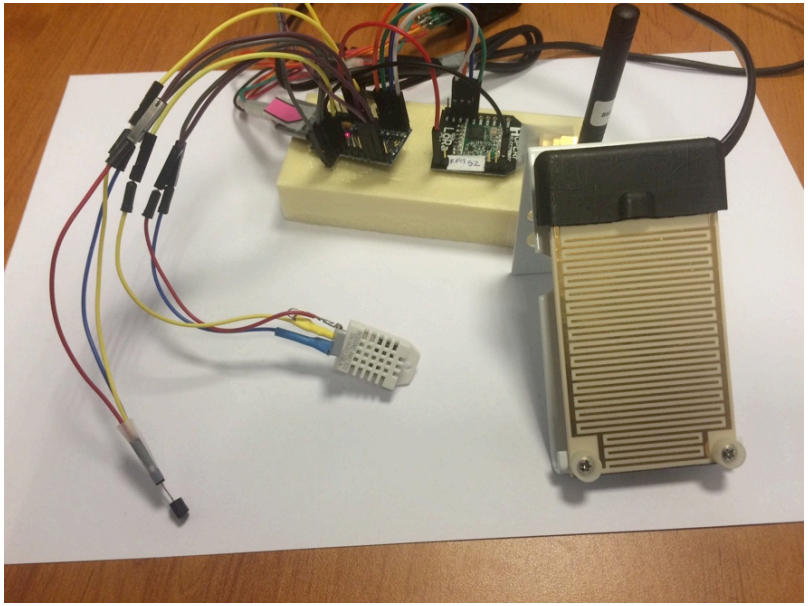
And converting into Celcius

```
Temp = value * 3300.0/1024.0; // 3300/1024=3.22

Temp = Temp / 10;                  // 10mV means 1°C

// now process and transmit the data
```

Generalization

- Depending on the sensor type, getting the physical measure from the analog/digital value follows a specific function provided by the sensor's manufacturer
- Depending on the microcontroller board, the number of I/O pins and the operating voltage may differ
- However the process is always the same:
  - Connect the sensor to the microcontroller board
  - Read analog or digital pin
  - Convert read value into meaningful physical measure
  - Then process and/or transmit

Several physical sensors per device



Physical sensor will be derived from `Sensor` base class and offers a unique `get_value()` function that will be called periodically

A sensor with several physical sensor will simply loop over all the connected sensors to call the `get_value()` function

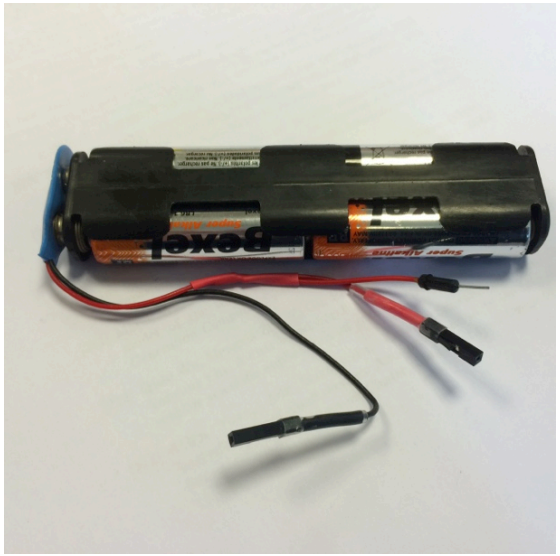
Several physical sensors can be connected to a board. Currently supported sensors:

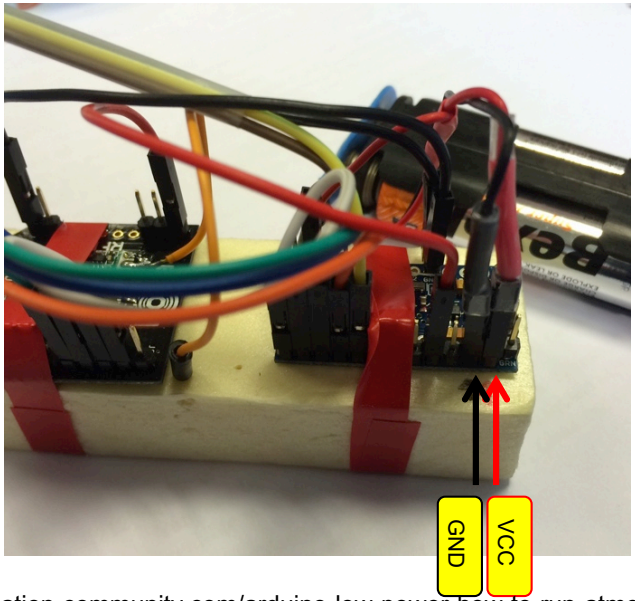
- All simple analog sensors (value on 1 analog wire: e.g. LM35Z temp. sensor or Davis leaf wetness Vantage)
- All simple digital sensors (binary value low/high)
- Most of one-wire digital sensor such as DHT22 sensor

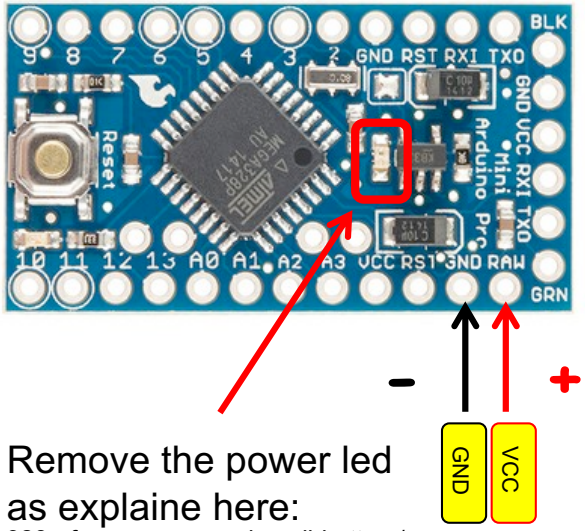
For specific sensors, `get_value()` will simply use specific provided/developped library

Running on battery & using low-power mode

CONNECTING A BATTERY







Remove the power led as explaine here:

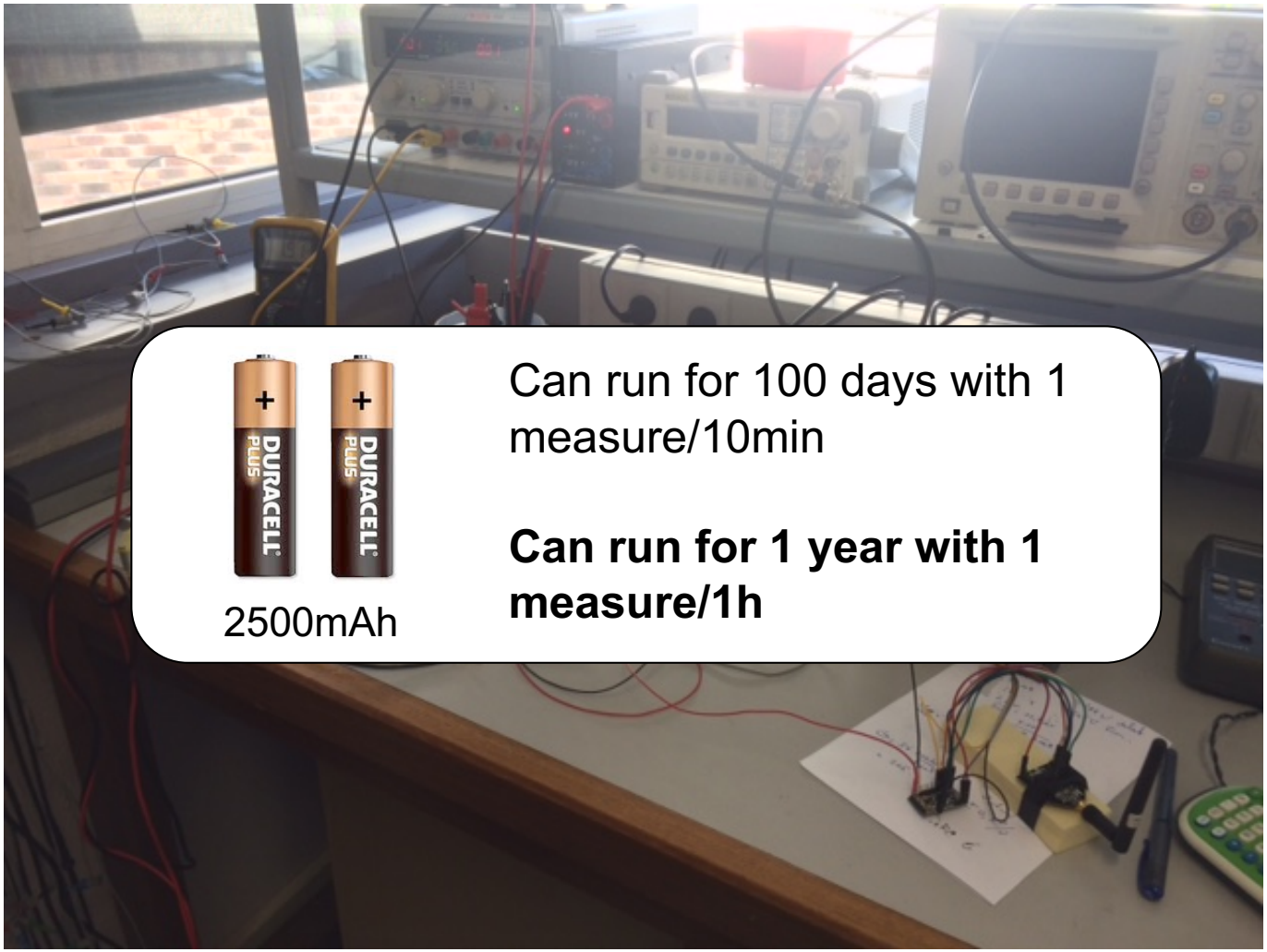
http://www.home-automation-community.com/arduino-low-power-how-to-run-atmega328p-for-a-year-on-coin-cell-battery/

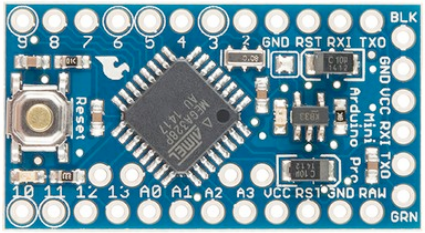
The advantage of using the Arduino Pro Mini running at 3.3v is that energy consumption can be low especially when the board is put in « sleep mode » for low-power operation. Remove the power led to further reduce consumption.

You can use 4 AA batteries to provide 4x1.5v=6v. Using only 3 batteries may lead to insufficient voltage difference.This will be injected into the RAW pin of the board, therefore using the on-board voltage regulator to get the 3.3v.

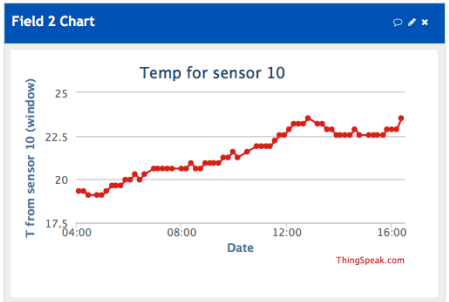
Running for 1 year!

Low-Power library from RocketScream





Wakes-up every 10min, take a measure (temp) and send to GW



146µA in deep sleep mode, 93mA when active and sending

Thanks to T. Mesplou and P. Plouraboué for their help

COMPILING FOR LOW-POWER

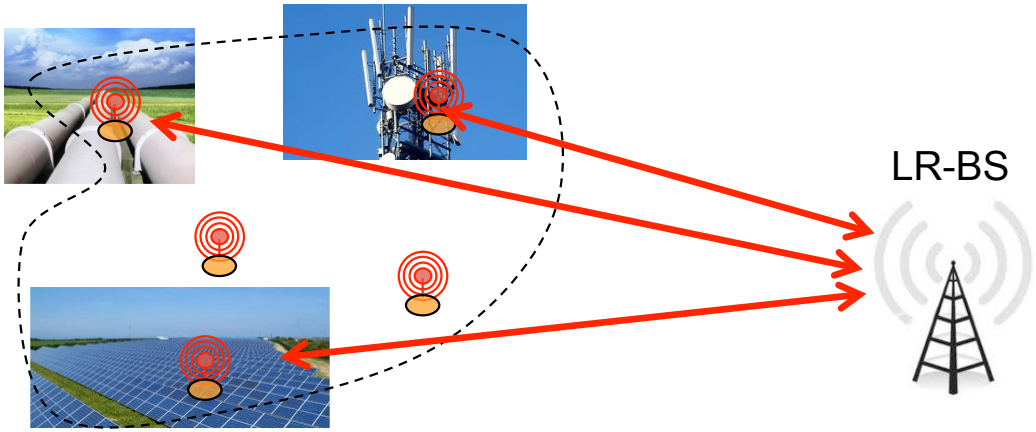
```
//////////////////////////////////////////////////////////////////
// COMMENT OR UNCOMMENT TO CHANGE FEATURES.
// ONLY IF YOU KNOW WHAT YOU ARE DOING!!! OTHERWISE LEAVE AS IT IS
#if not defined _VARIANT_ARDUINO_DUE_X_ && not defined __SAMD21G18A__
#define WITH_EEPROM
#endif
#define WITH_APPKEY
#define LOW_POWER
#define LOW_POWER_HIBERNATE
//#define WITH_ACK
//////////////////////////////////////////////////////////////////
```

Uncomment the « #define LOW_POWER » statement, compile and upload

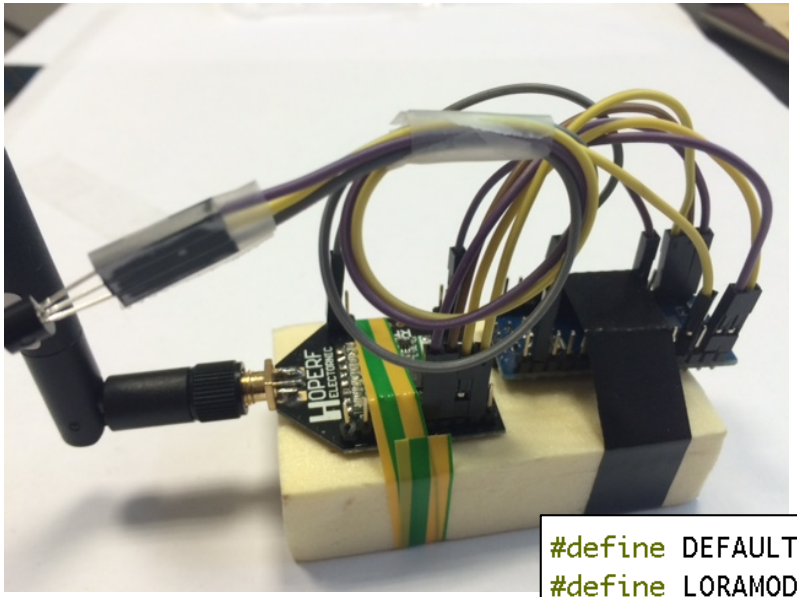
```
//////////////////////////////////////////////////////////////////
// CHANGE HERE THE TIME IN MINUTES BETWEEN 2 READING & TRANSMISSION
unsigned int idlePeriodInMin = 60;
//////////////////////////////////////////////////////////////////
```

Change the value of idlePeriodInMin to X minutes if needed

Sending to a gateway



DEFAULT CONFIGURATION



```
#define DEFAULT_DEST_ADDR 1
#define LORAMODE  1
#define node_addr 6
```

\!##TC/18.5

The default configuration in the Arduino_LoRa_temp example is:

Send packets to the gateway (one or many if in range)
LoRa mode 1
Node short address is 6

Securing with Application Key (1)

- End-device can use application key (app key) on 4 bytes to allow filtering mechanisms at the gateway side.
- The app key is defined in the end-device sketch (`Arduino_LoRa_temp`) and the feature is activated by uncommenting `#define WITH_APPKEY`

```
#ifdef WITH_APPKEY
////////////////////////////////////////////////////////////////
// CHANGE HERE THE APPKEY, BUT IF GW CHECKS FOR APPKEY, MUST BE
// IN THE APPKEY LIST MAINTAINED BY GW.
uint8_t my_appKey[4]={5, 6, 7, 8};
////////////////////////////////////////////////////////////////
#endif
```

- At the gateway side, post_processing_gw.py has a list of allowed app key

```
app_key_list = [
          #change/add here your application keys
          '\x01\x02\x03\x04',
          '\x05\x06\x07\x08' ]
```

Securing with Application Key (2)

- With app key enforcement at gateway, all LoRa data to be uploaded on clouds will need a valid app key, otherwise the data will be discarded as shown below:

```
--- rxlora. dst=1 type=0x12 src=6 seq=136 len=17 SNR=9 RSSIpkt=-56
rcv ctrl pkt info (^p): 1, 18, 6,136,17,9,-56
splitted in:  [1, 18, 6, 136, 17, 9, -56]
(dst=1 type=0x12 src=6 seq=136 len=17 SNR=9 RSSI=-56)
got first framing byte
--> got app key sequence
app key is: [9, 10, 11, 12]
not in app key list
invalid app key: discard data
```

- This is configured in the `gateway_conf.json` file

```
    "freq": 433.3
  },
  "gateway_conf": {
    "gateway_ID": "000000XXXXXXXXXX",
    "ref_latitude": "my_lat",
    "ref_longitude": "my_long",
    "wappkey": false,
    "raw": false,
    "aes": false,
    "log_post_processing": true,
```

How to use app key

- App key can be used to differenciate data from one organization to another
  - Sensing devices of a given organization will use the same app key
  - The gateway is configured to only accept this app key
- App key can be used to distribute the gateway task in case several gateways in the same organization are deployed
  - Sensing devices will be caterorized with 2 app key
  - Each gateway will allow only one of these 2 app key
  - In this way, data that can be received by 2 gateways will be processed by only 1 gateway

Securing by encryption (1)

- `Arduino_LoRa_temp` is an extended version of `Arduino_LoRa_Simple_temp` with data encryption feature.
- Data will be encrypted using 128-bit AES algorithm following the LoRaWAN encryption method.
- Uncomment `#define WITH_AES`

```
////////////////////////////////////////////////////////////////////
// COMMENT OR UNCOMMENT TO CHANGE FEATURES.
// ONLY IF YOU KNOW WHAT YOU ARE DOING!!! OTHERWISE LEAVE AS IT IS
#if not defined _VARIANT_ARDUINO_DUE_X_ && not defined __SAMD21G18A__
#define WITH_EEPROM
#endif
#define WITH_APPKEY
#define LOW_POWER
#define LOW_POWER_HIBERNATE
#define WITH_AES
```

Securing by encryption (2)

- Encryption ensures confidentiality. The two 16-byte encryption keys are defined in the end-device sketch (`Arduino_LoRa_temp`)

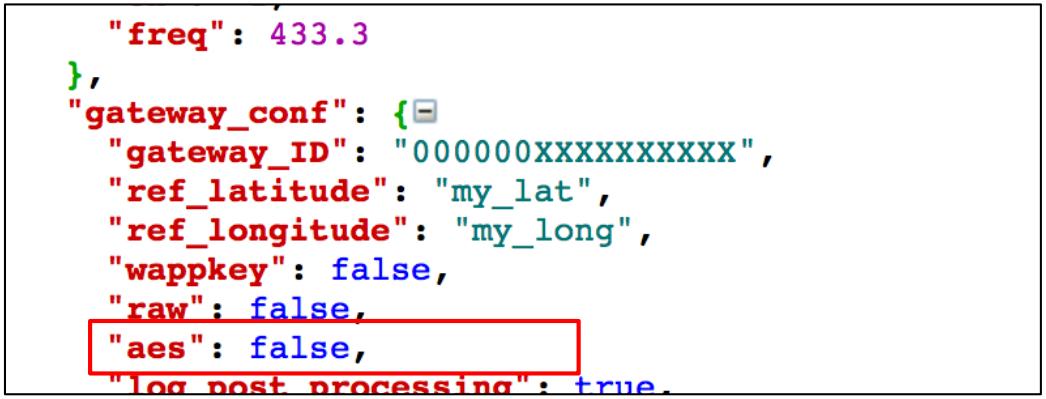
```
unsigned char AppSkey[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};

unsigned char NwkSkey[16] = {
  0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6,
  0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C
};
```

- And should also be declared in the `loraWAN.py` script on the gateway

```
AppSKey = '2B7E151628AED2A6ABF7158809CF4F3C'
NwkSKey = '2B7E151628AED2A6ABF7158809CF4F3C'
```

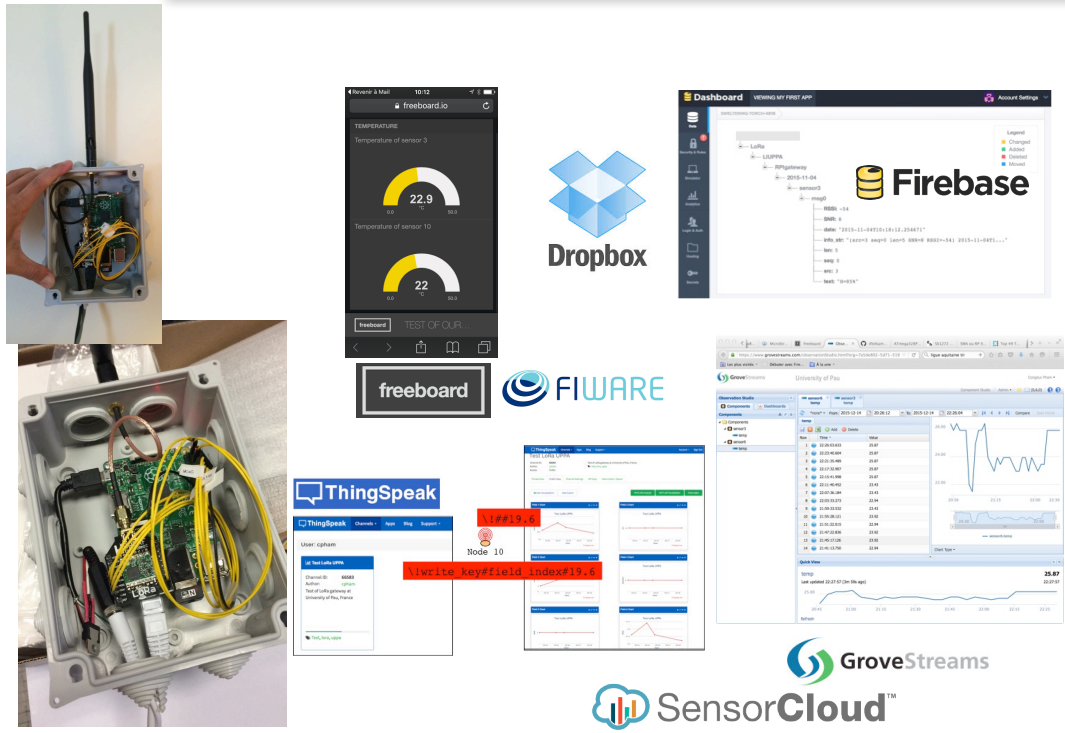
Securing by encryption (3)

- With encryption at device and decryption at gateway, there is more robust integrity check of the messages.
- Note that app key can still be used with AES, even if different gateways may have different encryption keys.
- To enable decryption at gateway, AES feature should be declared in the `gateway_conf.json` file.



- Otherwise, the gateway will not be able to decrypt and therefore will not be able to push meaningfull data to clouds

Gateway to cloud



Data received at the gateway will be pushed to IoT clouds. We provide python script examples for many IoT cloud platforms. Most of clouds with REST API can be easily integrated.

Now, have a look at the « Low-cost LoRa gateway: a step-by-step tutorial »