

# TRACK YOUR ATMOSPHERE

*104: Sensors and data servers for microcontrollers and embedded systems*



# Contents

<b>IO4: SENSORS AND DATA SERVERS FOR MICROCONTROLLERS AND EMBEDDED SYSTEMS</b>	<b>1</b>
<b>CONTENTS</b>	<b>2</b>
<b>TRYAT PROJECT</b>	<b>4</b>
<b>1. PRESENTATION</b>	<b>4</b>
<b>2. PEDAGOGICAL MODEL</b>	<b>5</b>
2.1. STRUCTURAL SCHEME	7
2.2. PROGRAMMING	8
2.2.a. Sample programme	8
<b>3. AUTONOMOUS MEASURING SYSTEM</b>	<b>12</b>
3.1. DIAGRAM OF USE CASES	13
3.2. REQUIREMENTS DIAGRAM	14
3.3. AUTONOMOUS POWER SUPPLY PART	15
3.3.a. Solution 1: LT1513 switch mode battery charger	15
3.3.b. Solution 2: Battery charger with linear regulator version L200	17
3.4. SENSOR CARD	18
3.4.a. Structural diagram of the sensor board	18
3.4.b. Choice of sensors	20
<b>4. AUTONOMOUS LOW CONSUMPTION MEASUREMENT SYSTEM</b>	<b>25</b>
4.1. STRUCTURAL DIAGRAM	28
4.2. IMPLEMENTATION OF THE LAYOUT	29
4.3. SOFTWARE DEVELOPMENT	29
4.4. TESTS AND MEASUREMENTS	30
4.4.a. Current measurement	30
<b>5. AUTONOMOUS MEASURING SYSTEM WITH WIFI CONNECTION</b>	<b>31</b>
5.1. STRUCTURAL SCHEME	32
5.2. IMPLEMENTATION OF THE LAYOUT	34
5.3. SOFTWARE DEVELOPMENT	34
5.4. TESTS AND MEASUREMENTS	35
5.4.a. Current measurement	35
<b>6. STAND-ALONE MEASURING SYSTEM MKRFOX VERSION</b>	<b>36</b>
6.1. STRUCTURAL SCHEME	38
6.2. IMPLEMENTATION OF THE LAYOUT	39
6.3. SOFTWARE DEVELOPMENT	39
6.4. TESTS AND MEASUREMENTS	40
6.4.a. Current measurement	40

---

**APPENDIX 1: SIGFOX** **41**

<b>1. A BIT OF HISTORY</b>	<b>41</b>
1.1. SIGFOX COVER	41
<b>2. SIGFOX MODULE REGISTRATION</b>	<b>43</b>
2.1. SIGFOX'S BACKEND INFRASTRUCTURE	46
<b>3. VIEWING RECEIVED MESSAGES</b>	<b>47</b>
<b>4. DATA RECOVERY AND EXPLOITATION</b>	<b>49</b>
<b>5. CALLBACK CONFIGURATION</b>	<b>50</b>

---

**APPENDIX 2: PLATFORMIO AND VSCODE** **53**

<b>1. WHAT IS IT ALL ABOUT?</b>	<b>53</b>
<b>2. INSTALLATION</b>	<b>53</b>
<b>3. USING PLATFORMIO</b>	<b>54</b>

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Co-funded by the  
Erasmus+ Programme  
of the European Union



# TRYAT project

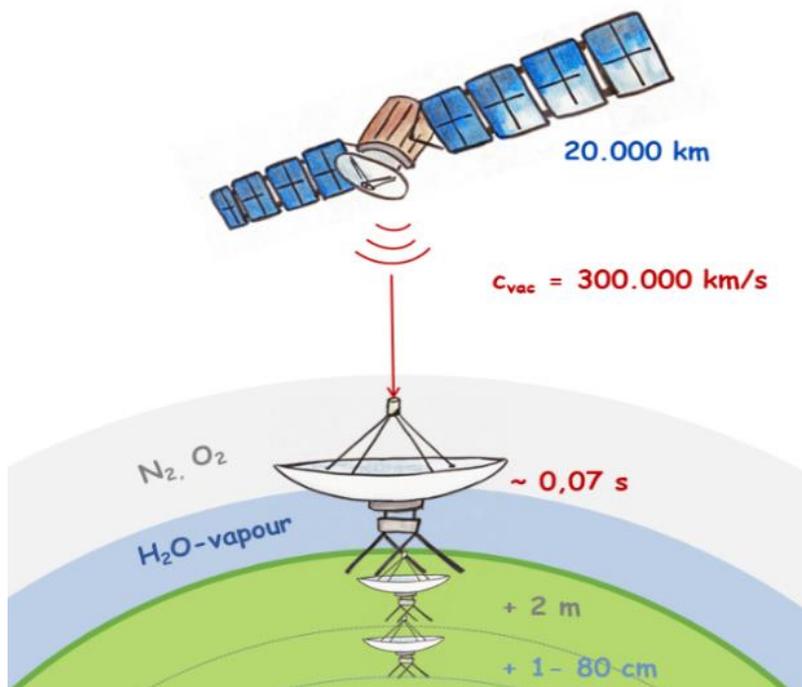
## 1. Presentation

A **GNSS** (*Global Navigation Satellite System*) is a system of geolocation and navigation by **satellites**. It must be capable of providing **real-time location information** for **navigation** purposes **anywhere**. The **GNSS** system must also ensure the **integrity** of this information.

The **signal** emitted by a **satellite** **travels** at the **speed of light**. Since the time of **transmission** is included in the **signal**, a receiver can then **calculate** the **distance** to the **satellite** against its **own clock**.

The **calculated distance** is somewhat **wrong** because :

- problems with the synchronisation of the transmit and receive clocks
- the **crossing** of the **atmosphere** which **degrades** the **signal**.



The **European** project (*Erasmus+*) TRYAT aims to use the **signal degradation** during its **crossing** of the **atmosphere** to make **weather forecasts**.

Indeed, the **presence of water vapour** in the **troposphere** (*see image opposite*) slows down the **movement** of the **satellite signal**.

This delay is almost **proportional** to the **amount of water vapour** present and can therefore be **evaluated**. **Quantifying** the quantity of **water vapour** makes it **possible** to improve **weather forecasts** and the **prevention of extreme phenomena**.

As **water vapour** is an important **greenhouse gas**, these **data** are also useful for studying **climate change**.

Calculations to evaluate this water vapour rate require knowledge of the temperature and atmospheric pressure at the satellite signal receiving antenna.

So we need to design a weather station that will record temperature, air pressure, humidity, wind speed and direction.

We offer two platforms:

- a model for didactic purposes on the operation and implementation of various sensors,
- an energy autonomous measuring system that will be used to carry out the various measurements on the sites.

## 2. Pedagogical model

---

This model is in the form of an "Arduino shield" and can be used with various microcontroller development boards compatible with this format.

The layout that we have developed makes it possible to implement :

1. an analogue temperature sensor (Pt1000 sensor),
2. a DS18B20 digital temperature sensor,
3. a digital barometric sensor (MPL3115A2),
4. an analogue humidity sensor (HTM2500LF),
5. a digital humidity sensor T9602 or AM2315,
6. an anemometer (flexible blade switch controlled by a magnet during rotation),
7. a weather vane (16 positions per turn),
8. a rain gauge (pulse counting).

This layout will enable the development of a series of practical works (TP) centred on the use of different sensors and their programming. We propose to work in groups of 2 students with a model and an Mbed compatible development board (FRDM KL25Z or Nucleo 64).

For example :

TP\_N°1 implementation of a digital temperature sensor DS18B20

- Analysis of the manufacturer's documentation (after Internet research)
- Search for a library to evaluate the sensor
- Carry out the programming using the Mbed platform to take the temperature of the classroom and display it on the PC screen using the serial link present thanks to the USB connection via the "Open SDA" interface of the Mbed compatible cards.
- Carry out 1-Wire link measurements by reading the DQ signal with the digital oscilloscope.
- Analysis of the 1 Wire connection.

TP\_N°2 implementation of a digital pressure sensor MPL3115A2

- Analysis of the manufacturer's documentation (after Internet research)
- Search for a library to evaluate the sensor
- Carry out the programming using the Mbed platform to record the pressure and temperature of the classroom and display them on the PC screen using the serial link present thanks to the USB connection via the "Open SDA" interface of the Mbed compatible cards.
- Perform I2C link measurements by taking SCL and SDA signals from the digital oscilloscope.
- I2C link analysis

#### TP\_N°3 use of a humidity and temperature sensor (T9602 or AM 2315)

- Analysis of the manufacturer's documentation (after Internet research)
- Search for a library to evaluate the sensor
- Carry out the programming using the Mbed platform to take the humidity and temperature outside and display them on the PC screen using the serial link present thanks to the USB connection via the "Open SDA" interface of the Mbed compatible cards.
- Perform I2C link measurements by taking SCL and SDA signals from the digital oscilloscope.
- Add the pressure and temperature measurement inside the classroom by repeating the work done during TP2.

#### TP\_N°4 implementation of a wind speed sensor (anemometer)

- Justify the sensor wiring
- Explain how to initialize the devices to operate this sensor using interrupts.
- Carry out the programming using the Mbed platform to record the wind speed (a small fan controlled by a variable power supply - 0 to 12 V can be used) and display the wind speed on the PC screen using the serial link present thanks to the USB connection via the "Open SDA" interface of the Mbed compatible cards.
- Read the output signal from the sensor with a digital oscilloscope.
- Compare the measured signal with the measurements transmitted to the PC.

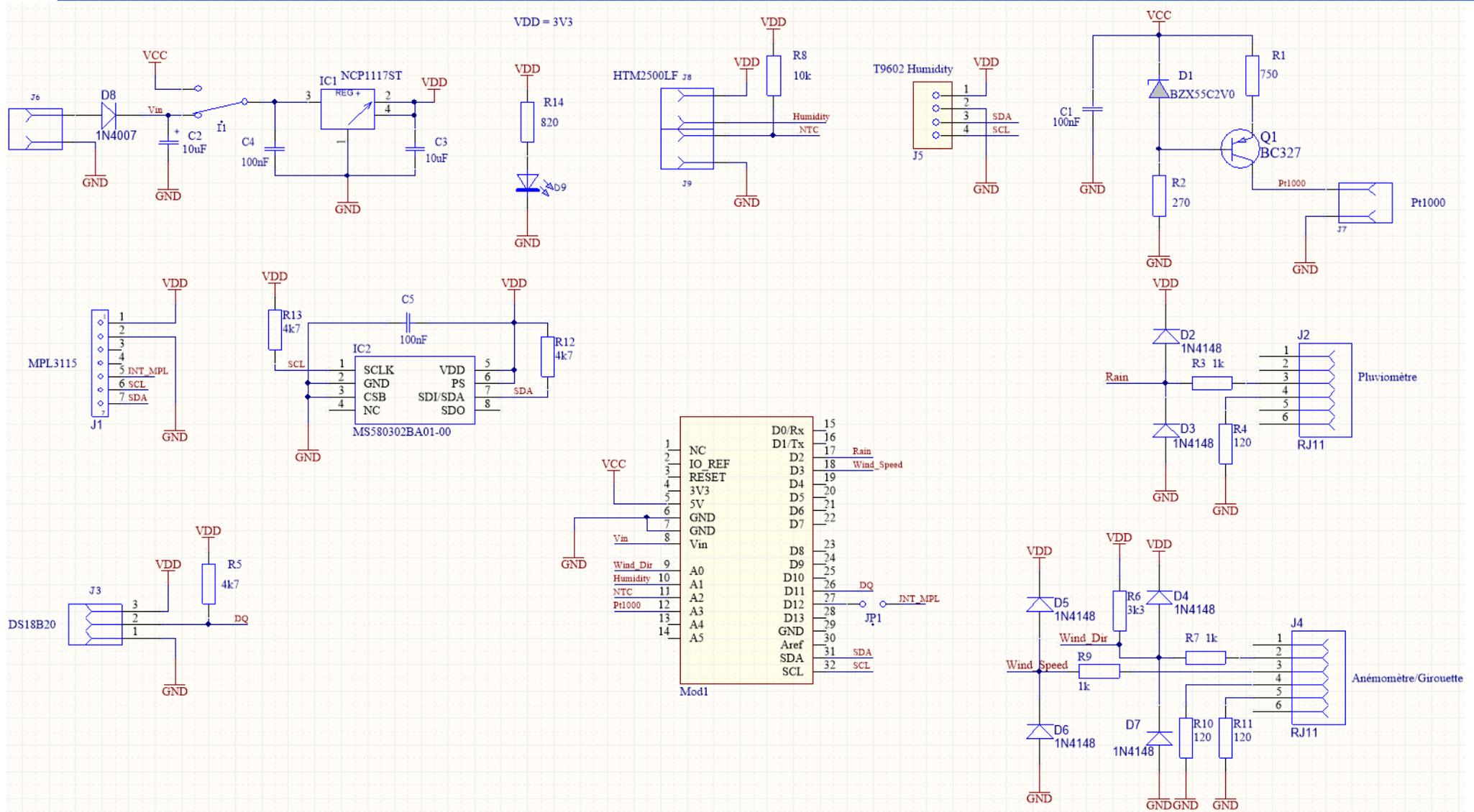
#### TP\_N°5 implementation of a wind direction sensor

- Find documentation to explain how this sensor works,
- Justify the windvane wiring.
- Explain how to initialize the devices to operate this sensor.
- Carry out the programming using the Mbed platform to take the wind direction (a small fan controlled by a variable power supply - 0 to 12 V can be used) displayed on the PC screen using the serial link present thanks to the USB connection via the "Open SDA" interface of the Mbed compatible cards.
- Perform sensor output voltage measurements for the 16 possible windvane positions.
- Use a spreadsheet to calculate the theoretical voltage value based on the data provided by the manufacturer and the external components used. Compare with previous measurements.

#### TP\_N°6 use of an analogue temperature sensor (Pt1000 sensor)

- Find documentation to explain how this sensor works,
- Analyze the installation diagram (conditioner) for operating this sensor.
- Explain how to initialize the devices to operate this sensor.
- Carry out the programming using the Mbed platform to take the wind direction (a small fan controlled by a variable power supply - 0 to 12 V can be used) displayed on the PC screen using the serial link present thanks to the USB connection via the "Open SDA" interface of the Mbed compatible cards.
- Take a measurement of the conditioner output voltage and then measure the resistance of the Pt1000 sensor (after disconnecting it).
- Make changes to the initial programme to display the temperature based on previous measurements.
- Use a spreadsheet to calibrate your sensor against a standard thermometer.

## 2.1. Structural scheme



## 2.2. Programming

ARM has developed a programming environment that allows rapid prototyping using cards based on microcontrollers from the Cortex M family.

Many manufacturers (NXP, ST, Silicon Labs, etc.) offer a variety of evaluation and prototyping boards that are compatible with the popular and easy to use ARM Mbed development platform. With an online Software Development Kit, open source software libraries, hardware designs and online tools, this is the fastest way to create products based on ARM microcontrollers.



See the document entitled "Development chain for STM32" and in particular the chapter devoted to the Mbed platform.

We have tested this board for a NUCLEO F401RE (ST) development board.

### 2.2.a. Sample programme

In the following code, we have also added a fine particle measurement using a sensor (PMS5003) connected to a serial link of the NUCLEO F401RE board. This sensor is not present on the pedagogical model.

```
#include "mbed.h"
#include "MPL3115A2.h".
#include "DS1820.h".

#define adr_T9602 0x50 //8bits mode <=> 0x28 << 1
const int MPLAddressI2C = 0xC0; //This is the slave address of the device
// -----
// Hyperterminal configuration
// 9600 bauds, 8-bit data, no parity
// -----
Serial pc(SERIAL_TX, SERIAL_RX);

InterruptIn anemo(D3, PullUp);
Ticker PasTemporary;
AnalogIn wind_Dir(A0);
Serial PM25(PA_11, PA_12);
AnalogIn A_Humidity(A1);
AnalogIn NTC(A2);
AnalogIn pt1000(A3);
DigitalOut myled(LED1);

MPL3115A2 mpl3115(MPLAddressI2C, D14, D15, D12, D13);

I2C i2c(D14, D15);

DS1820 ds1820(D11); // DQ on spindle D11

char I2C_Read [8];
float time = 0.25;
float meas_r;
float temppt1000;
float tempNTC;
float AnHumidity;
int compt_anemo, impVentpSec,timeanemo;
const float tab_dirvent [16][2] = {
    {0.0, 31000}, {22.5, 6570}, {45.0, 8100}, {67.5, 891},
    {90.0, 1000}, {112.5, 688}, {135.0, 2200}, {157.5, 1410},
```

```

{180.0, 3900}, {202.5, 3140}, {225.0, 15600}, {247.5, 14120},
{270.0, 100000}, {292.5, 40000}, {315.0, 60000}, {337.5, 21000}
};
typedef struct {
    uint16_t framelen;
    uint16_t pm10_standard, pm25_standard, pm100_standard;
    uint16_t pm10_env, pm25_env, pm100_env;
    uint16_t particles_03um, particles_05um, particles_10um, particles_25um, particles_50um,
particles_100um;
    uint16_t unused;
    uint16_t checksum;
} PM25data;

volatile uint8_t data = '\0', pmbuffer[32] = {0};
bool go = false, pret = false;
volatile uint8_t Nbr;

void pms5003Read(void)
{
    data = PM25.getc();
    if (go) {
        pmbuffer [Nbr++] = data;
        if(Nbr>=32) {
            ready = true;
            Nbr=0;
            go = false;
        }
    } else {
        if( data == 0x42 && !pret) {
            go = true;
            pmbuffer [Nbr++] = data;
        }
    }
}

void int_anemo()
{
    compt_anemo++;
}
void int_timer()
{
    timeanemo++;
}
float vitesseVent (void)
{
    float lives;
    //1 implusion per second -> 2.4 km/h
    vit = (float)compt_anemo*2.4/((float)timeanemo*tpsmes);
    compt_anemo = 0;
    timeanemo = 0;
    return (vit);
}

```

```

float dir_vent(float rint)
{
    char i;
    for (i = 0; i < 16; i++) {
        if (rint > tab_dirvent[i][1] * 0.9 && rint < tab_dirvent[i][1] * 1.1) {
            return tab_dirvent[i][0];
        }
    }
    return -1;
}

float calcRin (void)
{
    float rint;
    rint = wind_Dir.read()*3300;
    rint = rint*3300/(3300-rint);
    return(rint);
}
    
```

```

int main()
{
    chariot wake_cmd [1];
    char data [4];
    float humidity;
    float temperatureC;
    ind. tank
    uint16_t buffer16bits [15];
    PM25data pm25data;
    compt_anemo = 0;
    impVentpSec = 0;
    timeanemo = 0;
    anemo.fall(&int_anemo);

    PasTemporel.attach(&int_timer, tpsmes);
    PM25.attach(&pms5003Read, Serial::RxIrq);
    myled = 0;
    Nbr = 0;

    if(!Impl3115.Init()) {
        pc.printf( Barometer Initialized\n\r);
    } else {
        pc.printf ("Barometer Failed To Initialize\n\r");
        while(1);
    }
    if(ds1820.begin()) {
        pc.printf( DS18B20 Found !!\n\r);
    } else {
        pc.printf( No DS&8B20 Sensor\n\r);
        while(1);
    }

    while(true) {
    
```

```

pc.printf("\n \n-----");
pc.printf("Temperature and Humidity and Pressure");
pc.printf ( Analog Sensors );
meas_r = pt1000.read(); // Read the analog input value (value from 0.0 to 1.0 = full ADC conversion
range)
temppt1000 = ((meas_r * 3317/1.97)-1000)/3.75; // Converts value in temperature range
pc.printf( %.2f Degrees, temppt1000);
meas_r = A_Humidity.read(); // Read the analog input value (value from 0.0 to 1.0 = full ADC conversion
range)
AnHumidity = (meas_r * 187.5)-37.7; // Converts value in RH range
pc.printf( %.2f RH\t\t, AnHumidity);
pc.printf(" \n----");
pc.printf("Digital Sensors");
ds1820.startConversion(); // start temperature conversion on DS18b20
wait_ms(800);
mpl3115.Read_MPL_Data();
wait_ms(100);
//**** Digital Humidity Sensor T9602 ****
data [0] = 0x00;
data [1] = 0x00;
data [2] = 0x00;
data [3] = 0x00;
wake_cmd[0] = 0x00;
i2c.write(adr_T9602, wake_cmd, 1);
wait_ms(50);
i2c.read(adr_T9602, data, 4);
humidity = (((data [0] & 63) << 8) + data [1]) / 163.84;
temperatureC = (((data [2] << 6) + (data [3] / 4)) / 99.29) - 40;

pc.printf( %.2f %.2f Deg\t\t,mpl3115.Temp_C(), temperatureC);
pc.printf( %.2f RH\t\t, humidity);
pc.printf( %.2f Pa\t\n,mpl3115.Pressure());
pc.printf( DS18B20 sensor\t);
pc.printf(" %.2f Deg\t\n",ds1820.read());
pc.printf(" \n-----");
if(loan) {

    for(ind = 0; ind <15; ind++) {
        buffer16bits [ind] = pmbuffer [2 + 2*ind + 1];
        buffer16bits [ind] = buffer16bits [ind] + pmbuffer [2 + 2*ind] < 8;
    }
    memcpy((void*)&pm25data, buffer16bits, 30);
pc.printf(" \n-----");
pc.printf("Concentration Units (standard)\n");
pc.printf("PM 1.0: %d",pm25data.pm10_standard);
pc.printf("\t\t\tPM 2.5: %d ", pm25data.pm25_standard);
pc.printf("\t\t\tPM 10: %d\n", pm25data.pm100_standard);
pc.printf(" \n-----");
pc.printf("Concentration Units (environmental)\n");
pc.printf("PM 1.0: %d", pm25data.pm10_env);
pc.printf("\t\t\tPM 2.5: %d", pm25data.pm25_env);
pc.printf("\t\t\tPM 10: %d\n", pm25data.pm100_env);

```

```

pc.printf(" \n-----");
pc.printf("Particles > 0.3um / 0.1L air: %d\n", pm25data.particles_03um);
pc.printf("Particles > 0.5um / 0.1L air: %d\n", pm25data.particles_05um);
pc.printf("Particles > 1.0um / 0.1L air: %d\n", pm25data.particles_10um);
pc.printf("Particles > 2.5um / 0.1L air: %d\n", pm25data.particles_25um);
pc.printf("Particles > 5.0um / 0.1L air: %d\n", pm25data.particles_50um);
pc.printf("Particles > 10 um / 0.1L air: %d\n", pm25data.particles_100um);
pc.printf(" \n-----");

pret = false;

}
wait_ms(2000);

pc.printf( WindSpeed = %.2f km/h\t, WindSpeed());
pc.printf("Direction Wind = %.2f degre(s) \n", dir_vent(calcRin()));
}
}
    
```

The measurement results are transmitted via a virtual UART serial link via the USB connection to a PC (the USB connection via the "Open SDA" interface includes a serial link which appears as an asynchronous serial port on the PC -com xx-).

There are many programs for managing serial ports on PCs (Termite, Tera Term, PuTTY,...).

### 3. Autonomous measuring system

The initial aim of the TRYAT project is to evaluate the water vapour rate in the troposphere. This measurement can be carried out by analysing GNSS signals received at various points around the globe.

As the signals transmitted by satellites pass through layers of the atmosphere, they undergo alterations that result in a positioning error (altitude).

Alterations suffered during the ionosphere crossing can be compensated for by using a multi-frequency receiver.

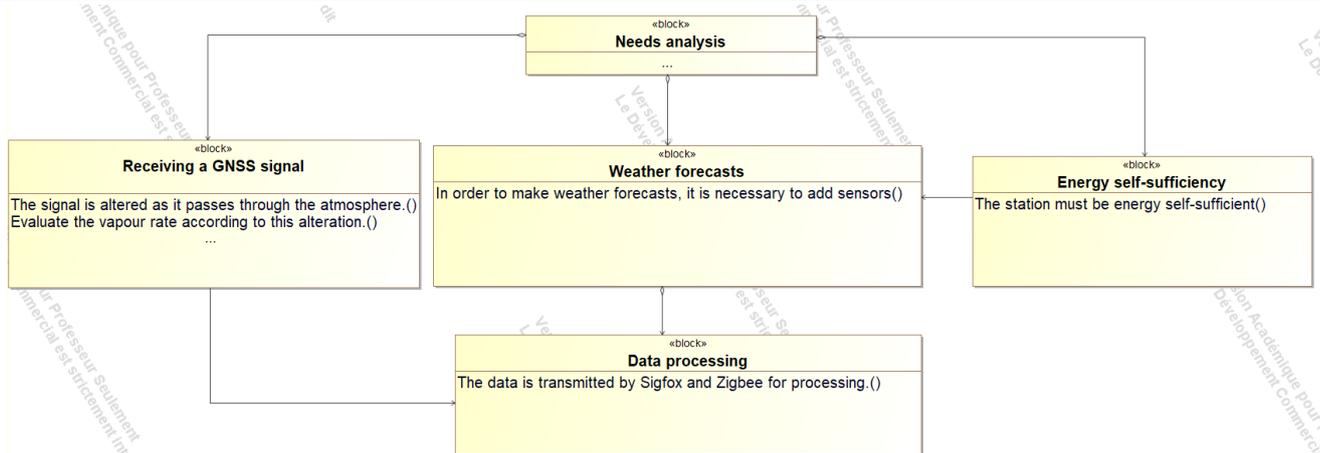
All that remains is the delay due to the crossing of the troposphere. This delay, which will give an altitude error of 2 to 3 m (for a satellite at the zenith) can be broken down into two terms (according to F. Duquenne, S. Botton, F. Peyret, D. Bétaille, and P. Willis, *GPS satellite positioning and navigation, second revised and expanded edition*. Lavoisier, 2005). :

- The dry component (due to the presence of nitrogen, oxygen, ...) at 90% of the tropospheric retardation.
- The wet component (due to the presence of water vapour) which is the one we are interested in.

Calculations to find these components require knowledge of atmospheric pressure and temperature.

We are therefore going to create a weather station that will enable us to acquire the necessary data (atmospheric pressure and temperature). In addition, we will measure humidity, wind speed and direction, as well as the levels of fine particles present in the air, allowing us to make weather forecasts. This station will be completely energy self-sufficient.

Needs assessment :



### 3.1. Diagram of use cases

The use case diagram allows you to define the different actions performed in our system:

- Acquire the sizes of all the sensors of the weather station.
- Allow power to be supplied by a solar panel and then store enough energy to be self-sufficient.
- Acquire a NMEA frame of a GNSS signal and extract the time data from it.

**NMEA:** National Marine Electronics Association is an American association of marine electronics manufacturers. **NMEA 0183** is a specification for communication between marine equipment, including GPS equipment. All data is transmitted as **ASCII** characters, all of which are printable, as well as **[CR]** carriage return and **[LF]** line feed. The data is transmitted as frames (sentences, phrases).

Each frame starts with the **\$** character followed by a group of 2 letters for the origin of the signal (GP for GPS, BD or GB for Beidou, GA for Galileo, GL for Glonass). Then a group of 3 letters for the frame identifier. Let's quote two of the most commonly used examples:

- **GGA:** for GPS location and date.
- **RMC:** recommended minimum data for gps.

This is followed by a number of fields separated by a "comma". The maximum size of a frame is 82 characters. For example :

**\$GPGGA,123519,4807.038,N,01131.324,E,1,08,0.9,545.4,M,46.9,M, , \*42**

**123519** = Acquisition of FIX at 12:35:19 UTC

**4807.038,N** = Latitude 48 deg 07.038' N

**01131.324,E** = Longitude 11 deg 31.324' E

**1** = Fix qualification: (0 = invalid, 1 = Fix GPS, 2 = Fix DGPS)

**08** = Number of satellites in pursuit.

**0.9** = DOP (Horizontal dilution of position) Dilution horizontale.

**545.4,M** = Altitude, in Metres, above the MSL (mean sea level) mean sea level.

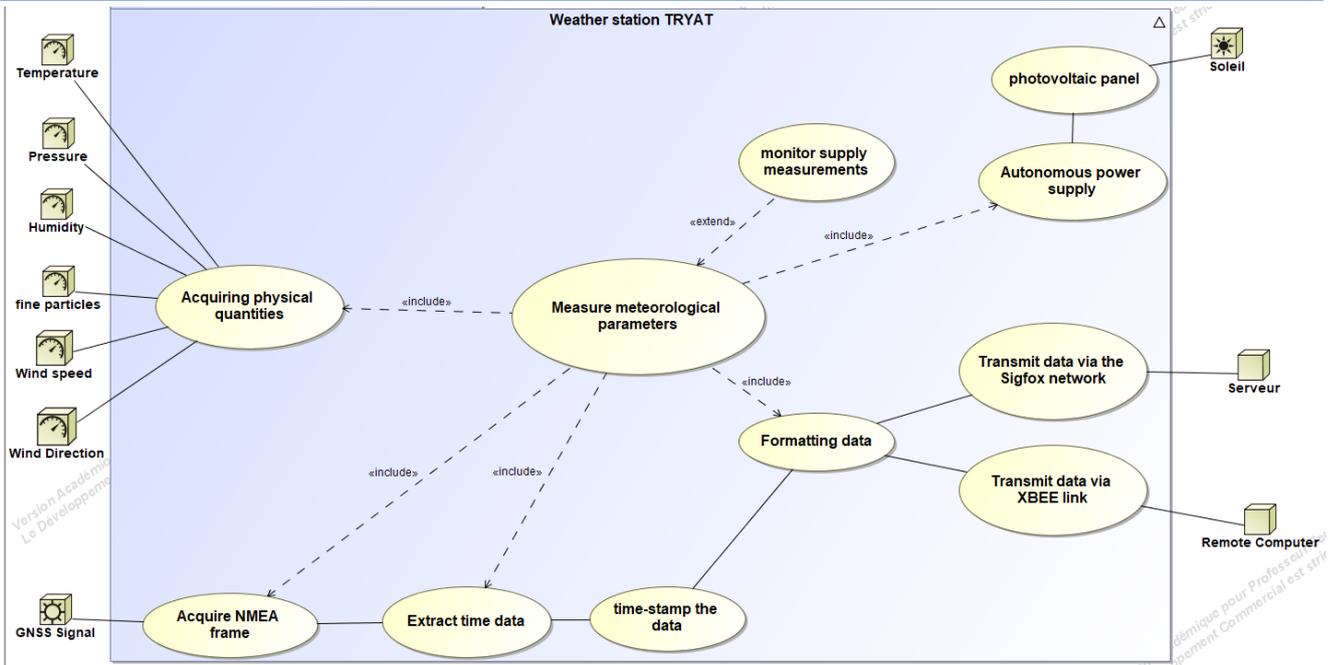
**46.9,M** = Correction of the geoid height in Metres with respect to the WGS84 ellipsoid (MSL).

**(Empty field)** = number of seconds since the last DGPS update.

**(Empty field)** = DGPS station identification.

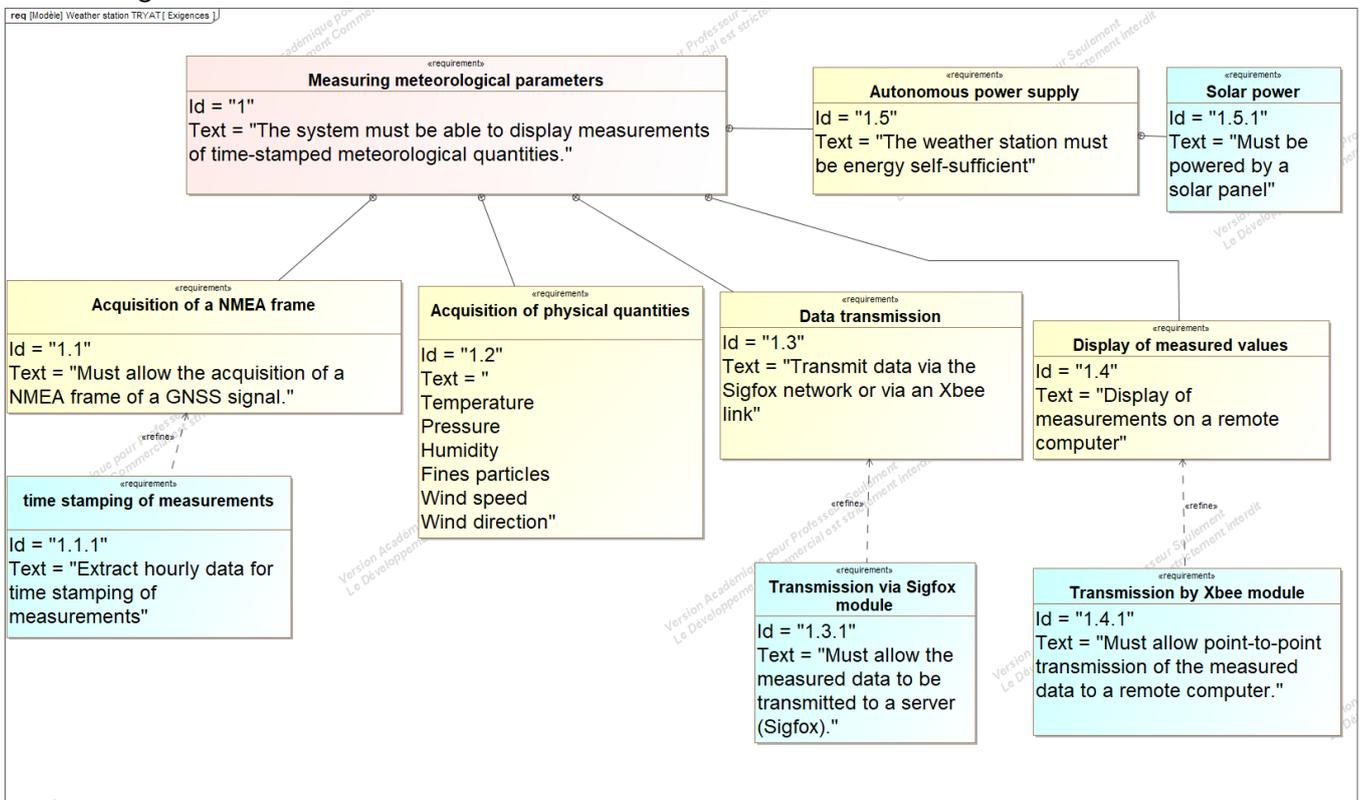
**\*42** = Checksum

- Time stamp all transmitted data.
- Transmit data via wireless links (Sigfox and Xbee).
- The data transmitted by the Sigfox module will go to a server.
- The data transmitted by the Xbee module will be received by a remote PC.



### 3.2. Requirements diagram

This diagram is used to determine the different tasks to be carried out.



### 3.3. Autonomous power supply part

This part will have to allow our system to be powered by solar energy.

The weather station must be able to be powered for 24 hours in the absence of sunshine (without charging the battery).

The maximum consumption of the system is estimated at 200mA. It will of course be possible to reduce this value by energy management by setting up sleep phases for all components between each measurement procedure.

The energy required for 24 hours will therefore be :  $0.2 A \cdot 24 h = 4.8 Ah$ .

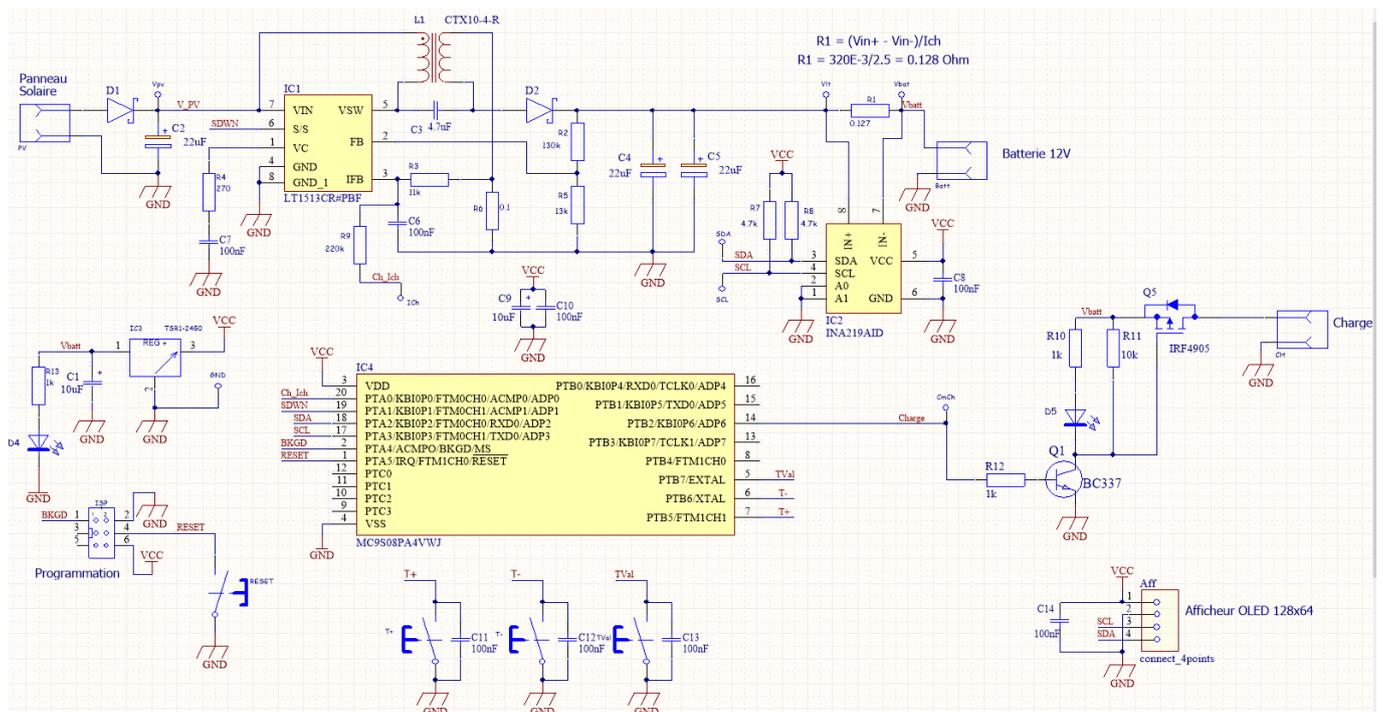
The choice of a 7Ah battery allows the specifications to be respected.

With a maximum charging current equal to one tenth of the battery capacity. Either

$I_{chmax} = 7/10 = 700 mA$

The choice of a 25Wp solar panel with a maximum current of 1.4 A will allow a satisfactory battery charge.

#### 3.3.a. Solution 1: LT1513 switch mode battery charger



The load is managed by a switching regulator (LT1513). It is possible to set the maximum load current by means of a pulse-width modulated control (Ch\_Ich) generated by the microcontroller.

We have also provided a measurement of the charging current and battery voltage via the IC2 component (INA219).

It is also possible to disconnect the load to avoid deep discharge of the battery. (Q1, Q5).

A graphical OLED display will allow the measured quantities to be visualised. The three push-buttons will allow the settings of the maximum charge current according to the selected battery.

This charger is managed by an 8-bit microcontroller from NXP.

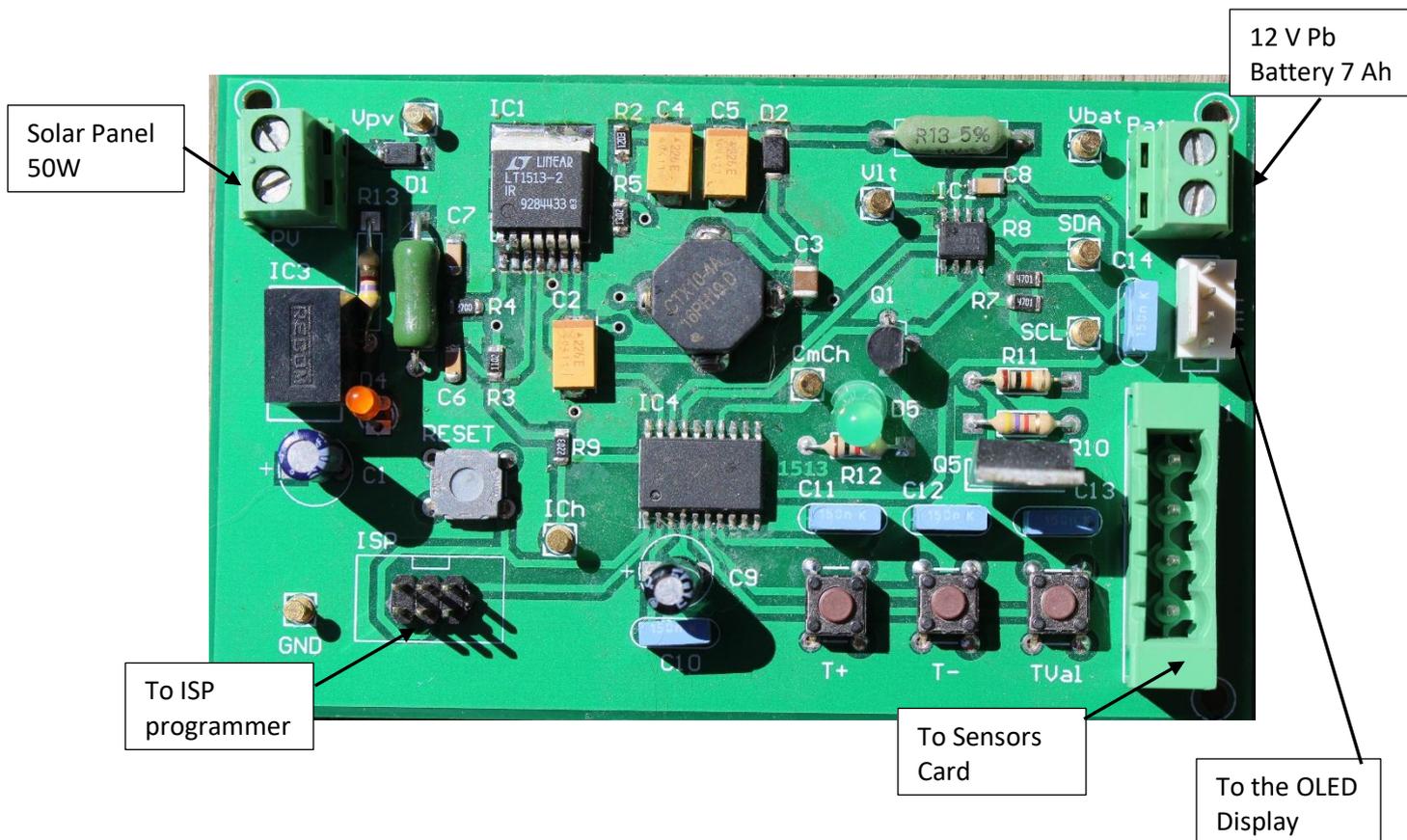
We use the software "CodeWarrior for MCU" Version: 11.0.1 Build Id:180609. It can be downloaded at: [https://www.nxp.com/design/software/development-software/codewarrior-development-tools/downloads:CW\\_DOWNLOADS](https://www.nxp.com/design/software/development-software/codewarrior-development-tools/downloads:CW_DOWNLOADS).

CodeWarrior® for MCUs (Eclipse IDE)	11.1	✓	✓	✓	Evaluation	Special
-------------------------------------	------	---	---	---	------------	---------

You have to choose the " Special " version which is a free version but limited in code size (32ko) which is not disturbing for our application.

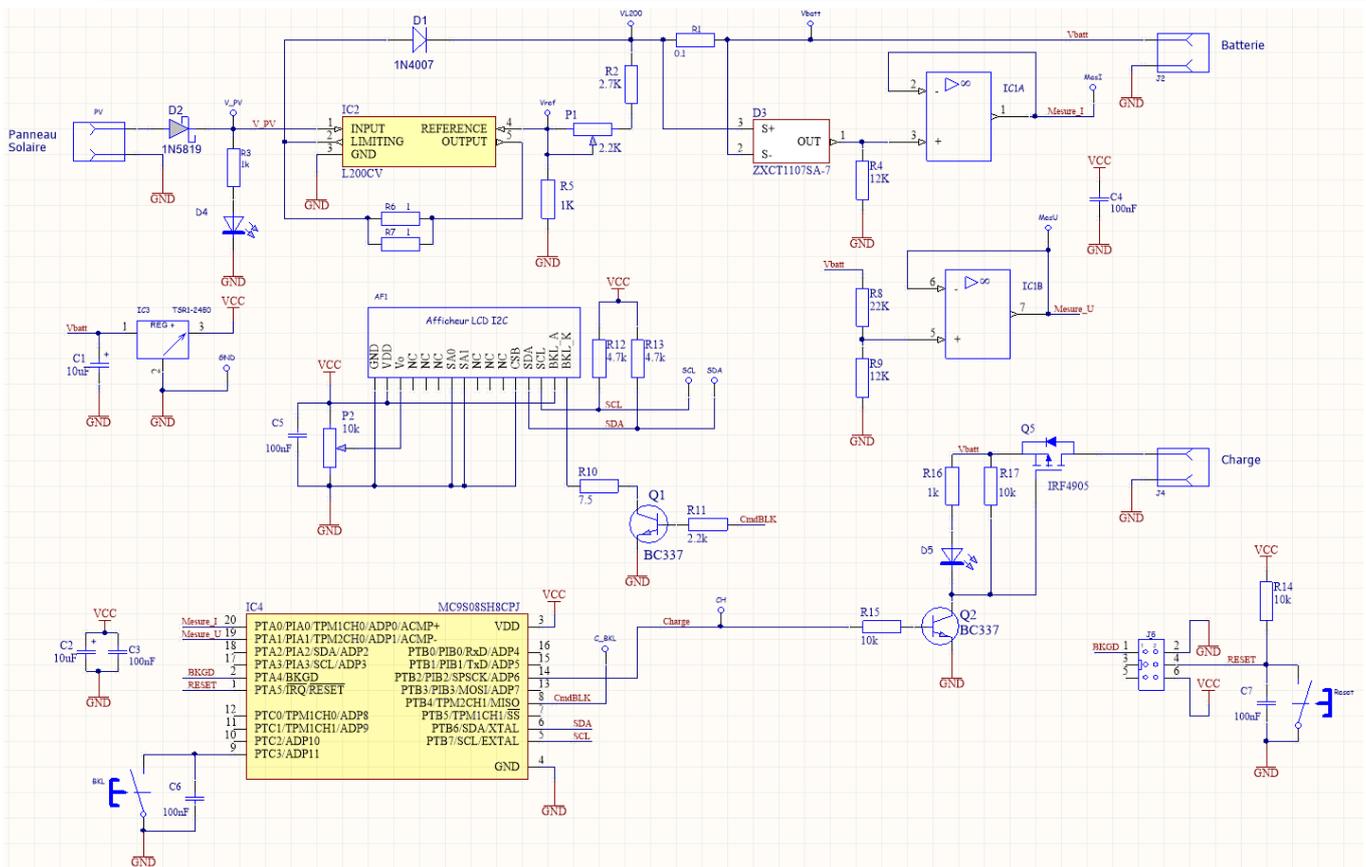
For in-situ programming, there are several possibilities:

1. Buy the version recommended by NXP " USB Multilink " manufactured by PE micro ([http://www.pemicro.com/products/product\\_viewDetails.cfm?product\\_id=15320168&productTab=1](http://www.pemicro.com/products/product_viewDetails.cfm?product_id=15320168&productTab=1)). This model allows the programming of a very large number of microcontrollers from different manufacturers,
2. Buy a cheap USBDM version (<https://witztronics.com/WP/product/wtusbdbl/>). This choice only allows you to program microcontrollers from Freescale (now NXP),
3. Make your own USBDM version ([http://usbdm.sourceforge.net/USBDM\\_V4.12/html/index.html](http://usbdm.sourceforge.net/USBDM_V4.12/html/index.html)). This is the most economical choice but it has the same restrictions as the previous one.



By default, the maximum charging current of the battery is set to 700 mA (i.e. for a 7Ah battery). However, it is possible to change this choice by pressing a push button to enter the setting mode and then pressing the T+ button to increase the battery capacity value or T- to decrease it. The choice is validated by pressing TVal. The battery capacity can range from 0 to 11 Ah.

### 3.3.b. Solution 2: Battery charger with linear regulator version L200

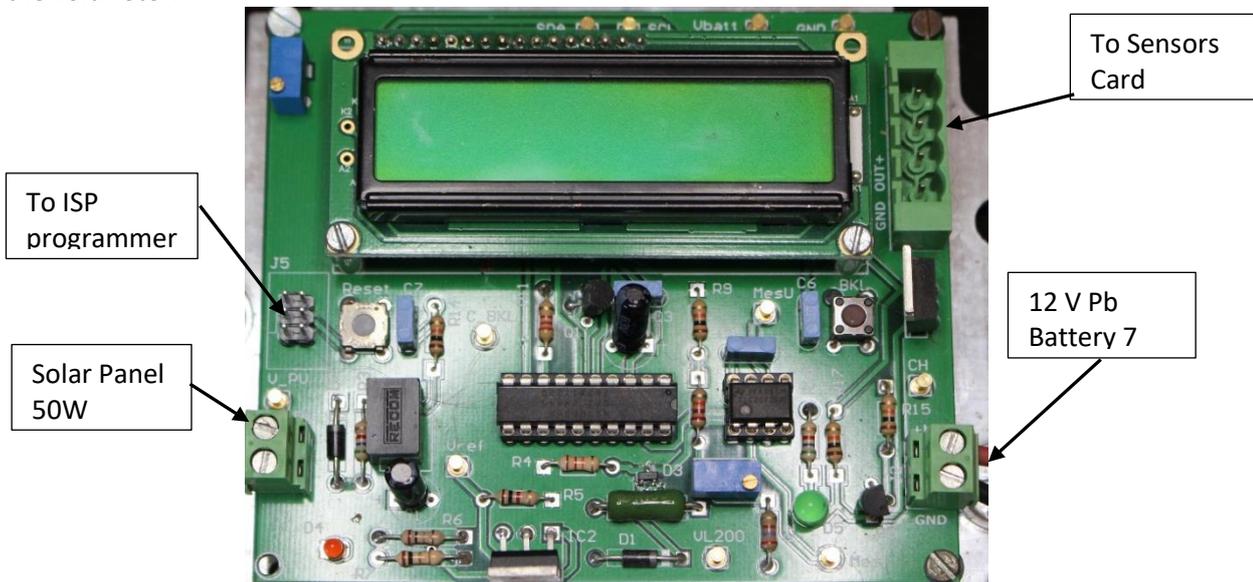


Load management is entrusted to a linear regulator (L200CV). The maximum load current is set by resistors R6 and R7. 
$$I_{Max} = \frac{V_5 - 2}{\frac{R_6 \cdot R_7}{R_6 + R_7}} = \frac{0,52}{0,5} \approx 1 \text{ A.}$$

We have also provided a measurement of charge current and battery voltage via the R1, D3, R4 and IC1A components.

It is also possible to disconnect the load to avoid deep discharge of the battery. (Q2, Q5).

Potentiometer P1 is used to set the end-of-charge voltage to 13.5 V. To carry out this adjustment, the board must be supplied by a laboratory power supply set at a voltage of 15 to 20V and limited in current to 1A. A voltmeter is then placed between the VL200 measuring point and ground and P1 is adjusted to obtain 13.5V on the voltmeter.



A graphic LCD display will allow the measured quantities to be visualised. An 8-bit microcontroller from NXP is used to manage this charger. We use the software " CodeWarrior for MCU " see previous description.

### 3.4. Sensor card

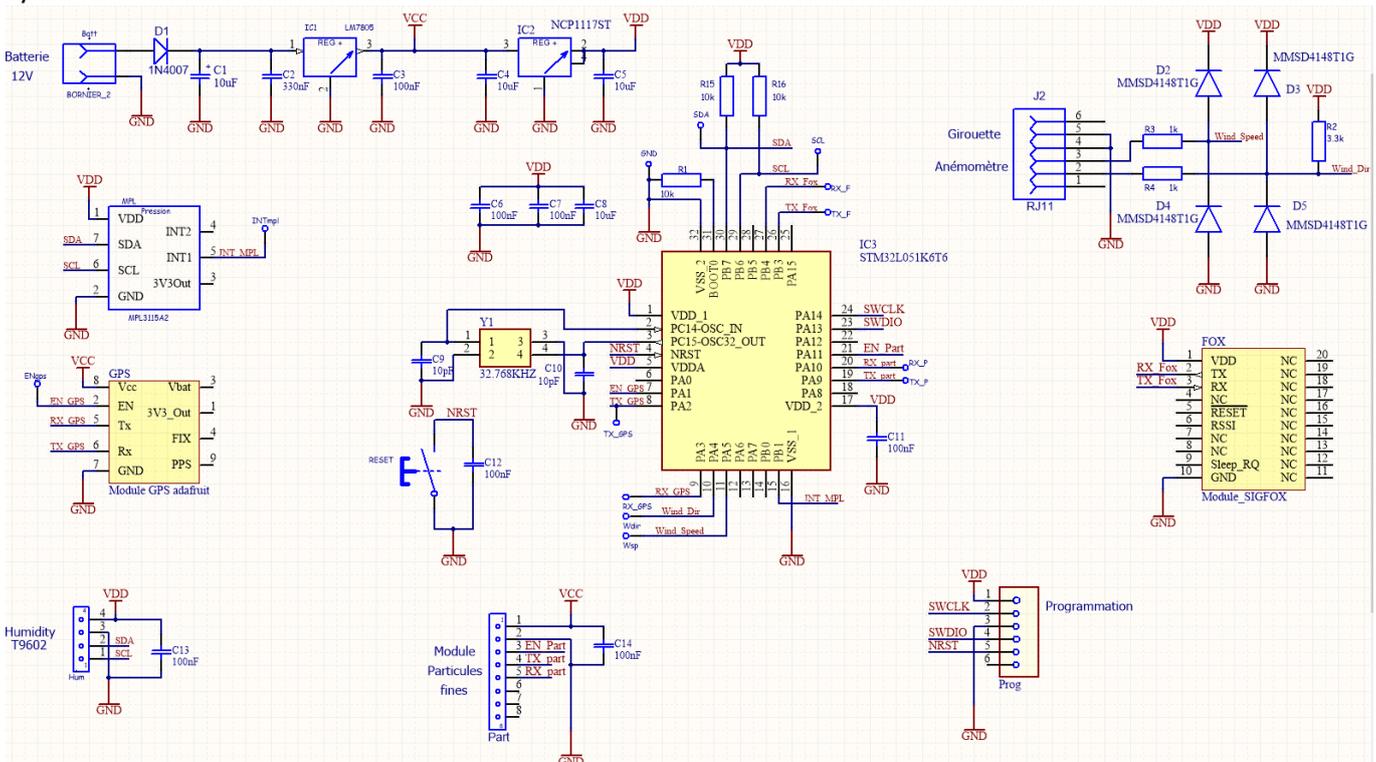
The layout that we have developed makes it possible to implement :

1. a digital barometric sensor (MPL3115A2),
2. a digital humidity sensor (T9602),
3. an anemometer (pulse counting)
4. a weather vane (16 positions per turn),
5. a fine particle sensor (plantower-pms5003),
6. a GPS receiver (Adafruit Ultimate GPS Breakout MTK3339 chipset),
7. a Sigfox module (Sigbee Xbee board module from ATIM),
8. an Xbee module (Xbee® module (series 2) "XB24-CZ7SIT-004" or "XB24-CZ7UIT-004").

The two wireless transmission modules (Sigfox or Xbee) cannot be used simultaneously. Therefore, a choice will have to be made on how to transfer the data.

#### 3.4.a. Structural diagram of the sensor board

In order to ensure continuity with the teaching layout, we wanted to use an ARM processor to manage our system.

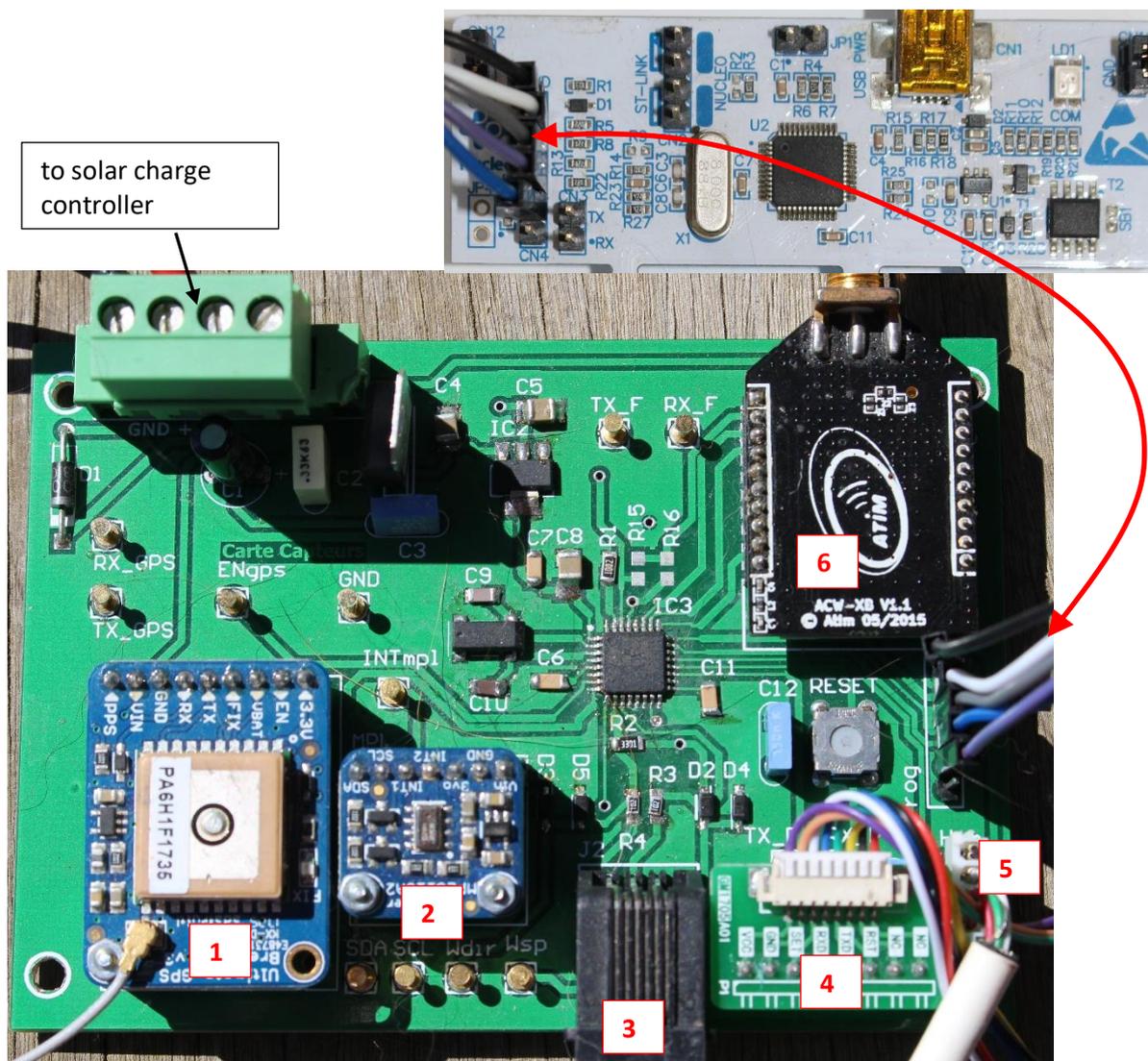


We chose a ST microcontroller from the STM32L0 family.

There are several reasons for this choice:

- The STM32L0 family is based on ARM Cortex M0+ MCUs with low power consumption (ultra low power),

- ST provides a free, cross-platform software development suite ([https://www.st.com/content/st\\_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-ides/stm32cubeide.html](https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-ides/stm32cubeide.html)). See the document entitled "Development chain for STM32" and in particular the chapter on The development environment for STM32.
- Possibility to use the STLINK V2 programmer which is supplied with the development boards (Nucleo)



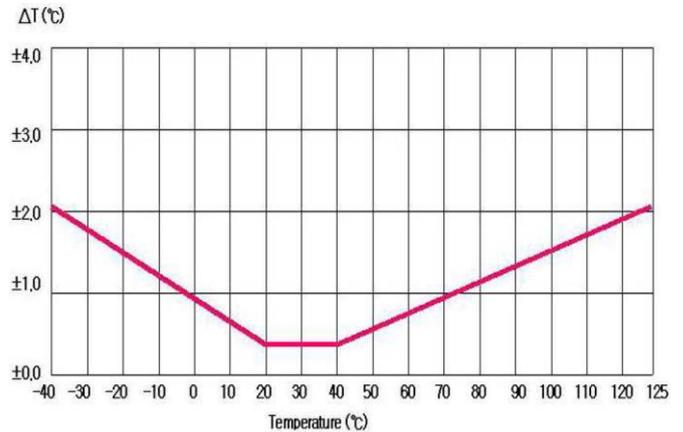
1. GPS module.
2. Pressure sensor.
3. Connector for anemometer and wind vane.
4. Fine particle sensor.
5. Humidity and temperature sensor.
6. Sigfox module.

### 3.4.b. Choice of sensors

#### Humidity and temperature sensor

For the measurement of temperature and humidity, we chose the T9602 model from Telaire. The range of temperature measurement corresponds perfectly to our needs.

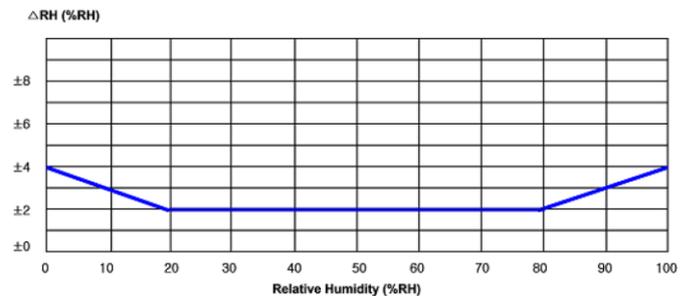
<b>Resolution</b>	14 bit (0.01°C)
<b>Accuracy</b>	±0.3°C <i>(Accuracies measured at 25°C, 5.0V)</i>
<b>Repeatability</b>	±0.1°C
<b>Response Time</b>	5.0 sec (63%) (min. 5.0 sec, max. 20 sec)
<b>Operating Range</b>	- 40 to 125°C
<b>Long Term Drift</b>	<0.05°C/yr (Normal condition)



Accuracy is satisfactory (±0.3°C) in the range of 20 to 40°C, but there is a lack of accuracy in the range of -20 to +100°C (±1.5°C). Nevertheless, we chose this sensor because it is IP67 certified (outdoor use) and has an interesting price/performance ratio.

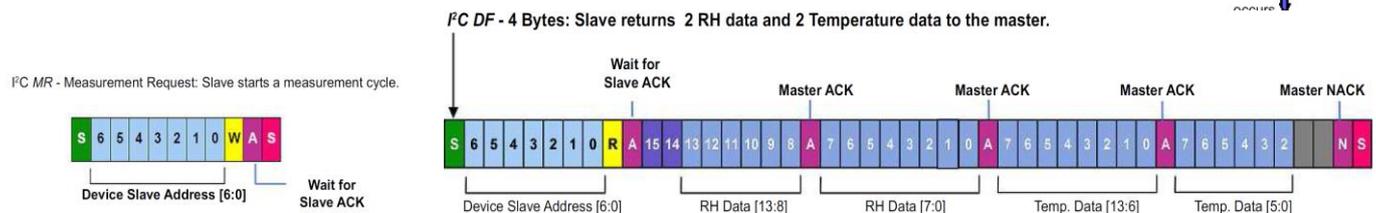
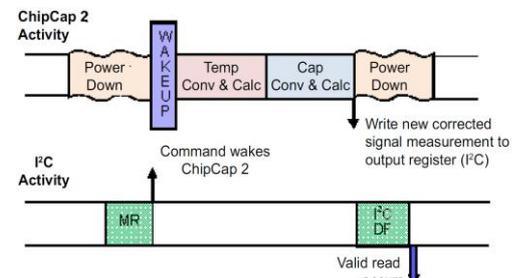
The humidity measurement is completely in accordance with the requirements:

<b>Resolution</b>	14 bit (0.01%RH)
<b>Accuracy</b>	±2.0%RH (20~80%RH) <i>(Accuracies measured at 25°C, 5.0V)</i> <i>(Custom accuracy tolerance available.)</i>
<b>Repeatability</b>	±0.2%RH
<b>Hysteresis</b>	+/- 1.0% RH
<b>Linearity</b>	<2.0%RH
<b>Response Time</b>	4.0 sec (τ 63%) (Measured at 25°C, 1 m/sec airflow for achieving 63% of step from 33%RH to 90%RH)



This sensor interfaces with the microcontroller via an I2C link (address 0x50 → 01010000). Its supply voltage is 3.3 V.

This sensor is in sleep mode by default to limit power consumption. A measurement sequence is triggered by sending the sensor address (MR) on the I2C bus (possibly followed by the sending of a meaningless data byte). This procedure triggers the sensor to wake up and then the temperature and humidity measurement. The data is ready to be read after a maximum of 45ms.



Calculations for temperature and humidity are given in the table below:

## Atmospheric pressure sensor

Humidity & Temperature Conversion Formula	
Humidity Output (%RH)	$(RH\_High [5:0] \times 256 + RH\_Low [7:0]) / 2^{14} \times 100$
Temperature Output (°C)	$(Temp\_High [7:0] \times 64 + Temp\_Low [7:2] / 4) / 2^{14} \times 165 - 40$

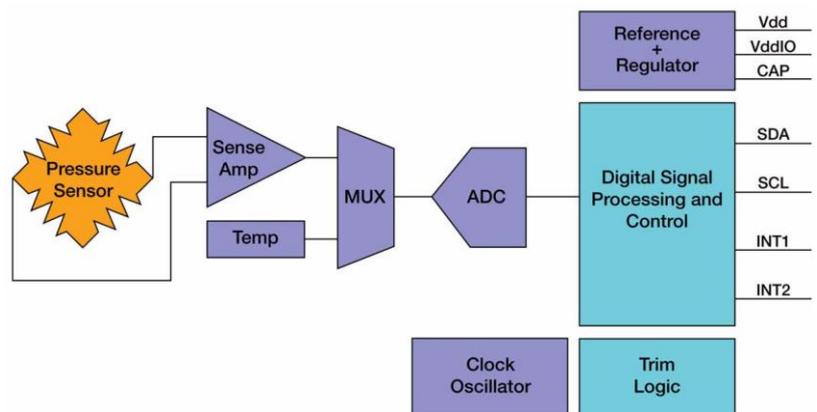
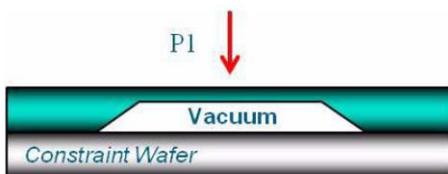
We have chosen the MPL3115A2 sensor which allows us to make the desired measurements with satisfactory accuracy. Indeed its measurement range extends from 50 to 110 kPa which is sufficient to make measurements of atmospheric pressure. Its accuracy is 0.4 kPa.

Like the previous sensor, the MPL3115A2 interfaces with the microcontroller via an I2C link (address 0xC0 → 110000). Its supply voltage is 3.3 V.

This small sensor is quite difficult for amateurs to solder. However, it is available as a small module on which it is already in place. This module is not waterproof, it will be soldered directly to the sensor card which will be placed in a waterproof case. This will have no influence on the pressure measurement.

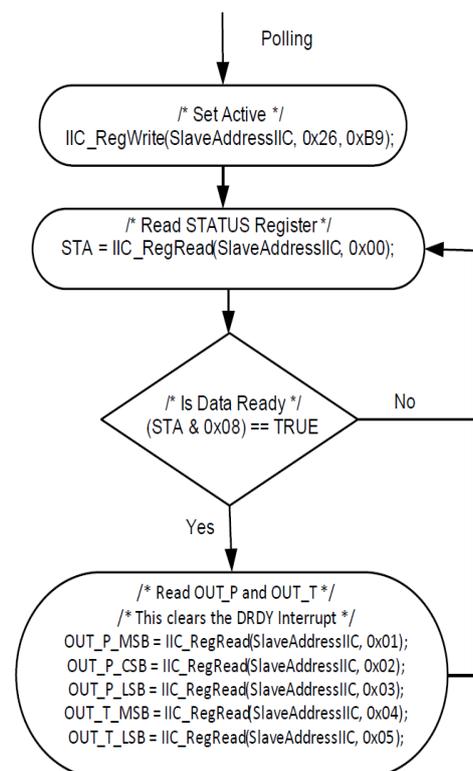
It is an absolute pressure sensor, i.e. with reference to an integrated capsule subjected to a perfect vacuum.

The absolute pressure sensing element of the MPL3115A2 altimeter is based on measuring the output voltage of a 4 strain gauge bridge related to the deformation of a diaphragm subjected to applied



atmospheric pressure (P1) relative to a zero pressure reference (vacuum pressure) sealed inside the reference chamber of the die during manufacture.

The principle of measurement in scanning mode is given in the flowchart opposite.



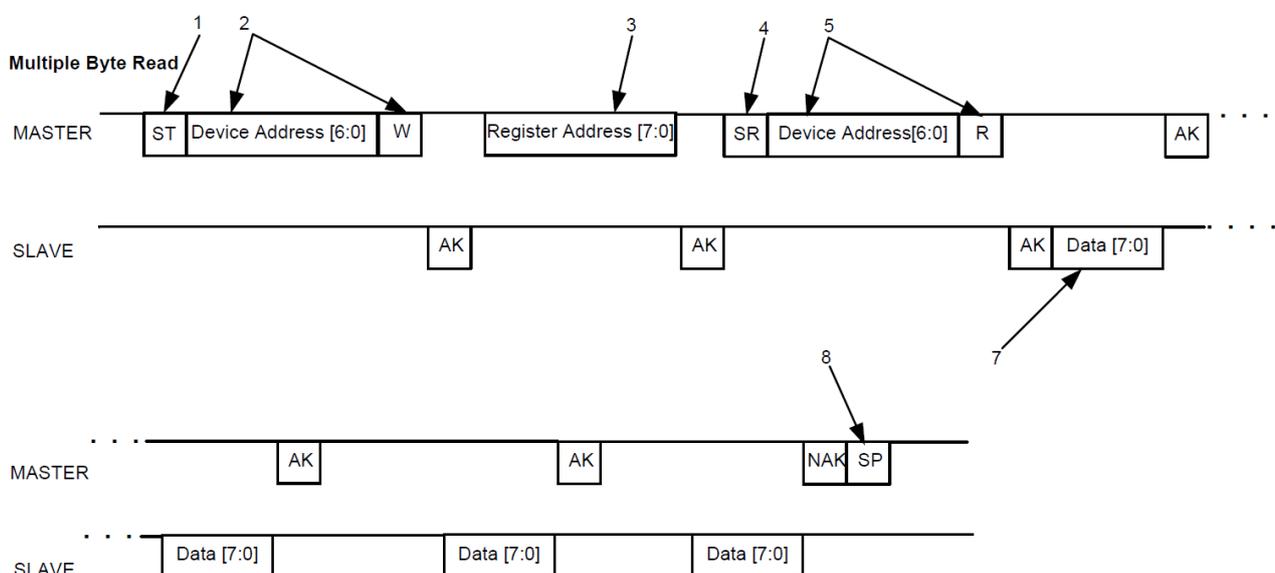
Note that there is no calculation to do, the pressure measurement is direct on 20 user bits (3 bytes OUT\_P\_MSB, OUT\_P\_CSB, OUT\_P\_LSB).

Register Address	Name	Reset	Reset when STBY to Active	Type	Auto-Increment Address		Comment	
0x00	Sensor Status Register (STATUS) <sup>(1)(2)</sup>	0x00	Yes	R	0x01		Alias for DR_STATUS or F_STATUS	
0x01	Pressure Data Out MSB (OUT_P_MSB) <sup>(1)(2)</sup>	0x00	Yes	R	0x02	0x01	Bits 12-19 of 20-bit real-time Pressure sample.	Root pointer to Pressure and Temperature FIFO data.
0x02	Pressure Data Out CSB (OUT_P_CSB) <sup>(1)(2)</sup>	0x00	Yes	R	0x03		Bits 4-11 of 20-bit real-time Pressure sample	
0x03	Pressure Data Out LSB (OUT_P_LSB) <sup>(1)(2)</sup>	0x00	Yes	R	0x04		Bits 0-3 of 20-bit real-time Pressure sample	
0x04	Temperature Data Out MSB (OUT_T_MSB) <sup>(1)(2)</sup>	0x00	Yes	R	0x05		Bits 4-11 of 12-bit real-time Temperature sample	
0x05	Temperature Data Out LSB (OUT_T_LSB) <sup>(1)(2)</sup>	0x00	Yes	R	0x00		Bits 1-3 of 12-bit real-time Temperature sample	
0x06/0x00	Sensor Status Register (DR_STATUS) <sup>(1)(2)</sup>	0x00	Yes	R	0x07		Data Ready status information	

1. Register contents are preserved when transitioning from "ACTIVE" to "STANDBY" mode.
2. Register contents are reset when transitioning from "STANDBY" to "ACTIVE" mode.

To read several registers, the following protocol must be followed:

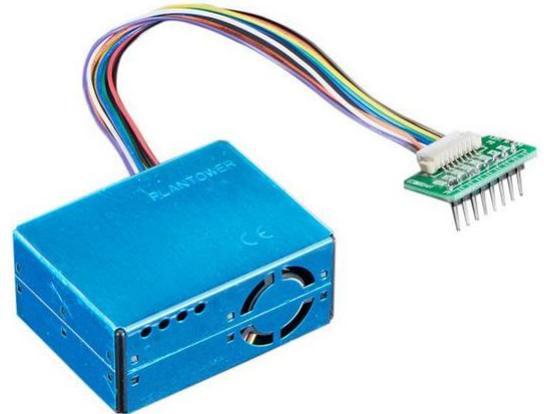
- 1- Sending a departure condition,
- 2- Sending the slave address 0xC0 in write mode
- 3- Sending the address of the first register to be read
- 4- Sending a new departure condition (ReStart)
- 5- Sending the slave address 0xC1 in read mode
- 6- Dummy reading (internal)
- 7- Looped data reading
- 8- Sending a stop condition.



## Fine dust sensor

### Presentation

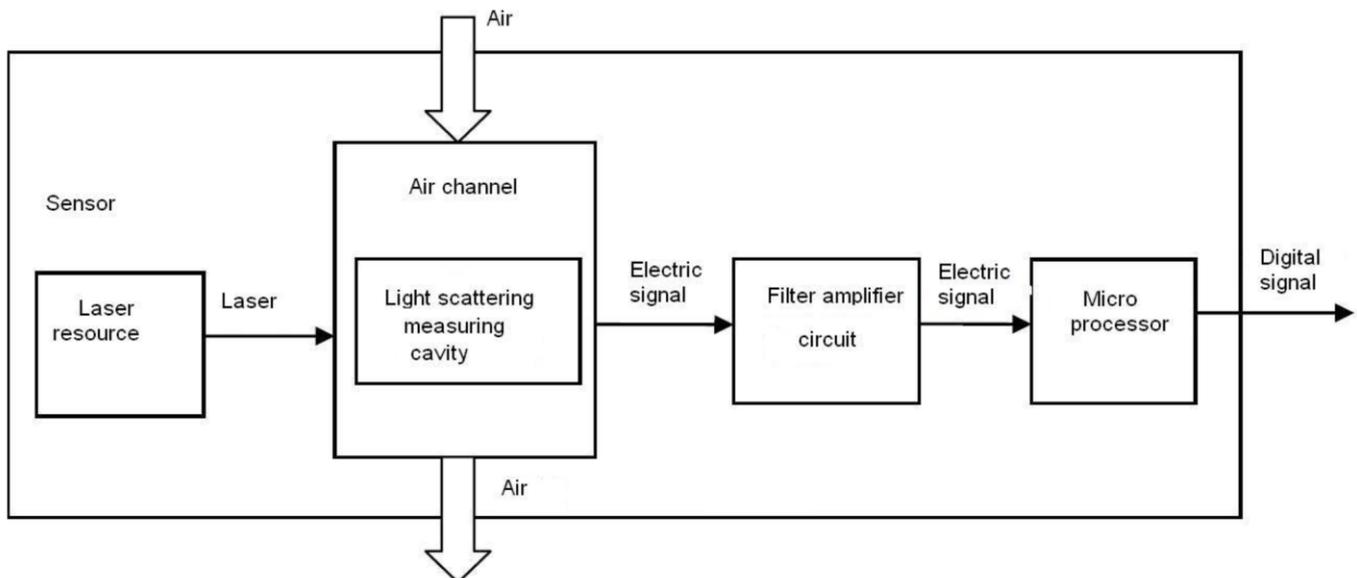
The PMS5003 is a digital particle concentration sensor, which can be used to obtain the number of particles suspended in the air, i.e. the particle concentration. The measured data is presented digitally and can be read using an asynchronous serial link (UART).



### Principle of operation

The principle of laser scattering is used for this type of sensor, i.e. it uses a laser beam to illuminate airborne particles. The particles will scatter the light in all directions, and the sensor measures the received light radiation at a certain angle to the incident light flux. Finally, the variation curve of the scattered light over time is obtained. Finally, the equivalent particle diameter and the number of particles with different diameters per unit volume can be calculated by microprocessor on the basis of the MIE theory<sup>1</sup>.

See the functional diagram of the sensor in the figure below :



The results are presented in three different forms:

- the raw format which gives the number of particles for according to their size per unit of volume, the unit of volume of the number of particles is 0.1L
- the particle concentration the unit of mass concentration is  $\mu\text{g}/\text{m}^3$  factory calibrated (CF = 1)
- the particle concentration the unit of mass concentration is  $\mu\text{g}/\text{m}^3$  in environmental mode.

There are two options for digital output: passive and active. The default mode is active after power on. In this mode, the sensor automatically sends serial data to the host. Passive mode requires the user to request that the data be sent.

The frame transmitted in the active mode is as follows:

<sup>1</sup> MIE theory: (source Wikipedia) In wave optics, **Mie theory**, or **Mie solution**, is a particular solution of Maxwell's equations describing the elastic scattering - i.e. without wavelength change - of a plane electromagnetic wave by a spherical particle characterised by its diameter and complex refractive index. It is named after the German physicist Gustav Mie, who described it in detail in 1908.

Default baud rate: 9600bps Check bit: None Stop bit: 1 bit

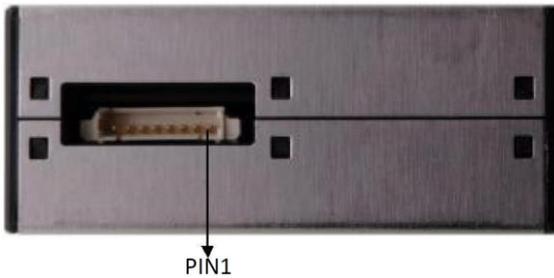
32 Bytes

Start character 1	0x42	(Fixed)
Start character2	0x4d	(Fixed)
Frame length high 8 bits	.....	Frame length=2x13+2(data+check bytes)
Frame length low 8 bits	.....	
Data 1 high 8 bits	.....	Data1 refers to PM1.0 concentration unit $\mu$ g/m3 (CF=1, standard particle) *
Data 1 low 8 bits	.....	
Data2 high 8 bits	.....	Data2 refers to PM2.5 concentration unit $\mu$ g/m3 (CF=1, standard particle)
Data2 low 8 bits	.....	
Data3 high 8 bits	.....	Data3 refers to PM10 concentration unit $\mu$ g/m3 (CF=1, standard particle)
Data3 low 8 bits	.....	
Data4 high 8 bits	.....	Data4 refers to PM1.0 concentration unit * $\mu$ g/m3 (under atmospheric environment)
Data4 low 8 bits	.....	
Data5 high 8 bits	.....	Data 5 refers to PM2.5 concentration unit $\mu$ g/m3 (under atmospheric environment)
Data5 low 8 bits	.....	
Data6 high 8 bits	.....	Data 6 refers to concentration unit (under atmospheric environment) $\mu$ g/m3
Data6 low 8 bits	.....	

Data7 high 8 bits	.....	Data7 indicates the number of particles with diameter beyond 0.3 $\mu$ m in 0.1 L of air.
Data7 low 8 bits	.....	
Data8 high 8 bits	.....	Data 8 indicates the number of particles with diameter beyond 0.5 $\mu$ m in 0.1 L of air.
Data8 low 8 bits	.....	
Data9 high 8 bits	.....	Data 9 indicates the number of particles with diameter beyond 1.0 $\mu$ m in 0.1 L of air.
Data9 low 8 bits	.....	
Data10 high 8 bits	.....	Data10 indicates the number of particles with diameter beyond 2.5 $\mu$ m in 0.1 L of air.
Data10 low 8 bits	.....	
Data11 high 8 bits	.....	Data11 indicates the number of particles with diameter beyond 5.0 $\mu$ m in 0.1 L of air.
Data11 low 8 bits	.....	
Data12 high 8 bits	.....	Data12 indicates the number of particles with diameter beyond 10 $\mu$ m in 0.1 L of air.
Data12 low 8 bits	.....	
Data13 high 8 bits	.....	Data13 Reserved
Data13 low 8 bits	.....	
Data and check high 8 bits	.....	Check code=Start character 1+ Start character 2+.....+data 13 Low 8 bits
Data and check low 8 bits	.....	

Note: CF=1 should be used in the factory environment

Broaching the module :



<b>PIN1</b>	VCC	Positive power 5V
<b>PIN2</b>	GND	Negative power
<b>PIN3</b>	SET	Set pin /TTL level@3.3V, high level or suspending is normal working status, while low level is sleeping mode.
<b>PIN4</b>	RX	Serial port receiving pin/TTL level@3.3V
<b>PIN5</b>	TX	Serial port sending pin/TTL level@3.3V
<b>PIN6</b>	RESET	Module reset signal /TTL level@3.3V, low reset.
<b>PIN7/8</b>	NC	

## 4. Autonomous low consumption measurement system

After our meeting in Berlin in February 2020 and our exchanges between partners, it appeared that a more sober version of this measurement system would be interesting.

We have therefore developed a low-power version that makes extensive use of the low-energy potential of our STM32L0 microcontrollers.

We therefore chose to use a 4.3V LiPo battery (fairly cheap and very common about 5€ for a 1000mA/h battery) which has a higher energy density than a lead battery. This will reduce the size and weight of our system.

We will replace the fine particle sensor which requires a 5V power supply (not available in this version) with a Lux brightness sensor and a UV level sensor.

The layout that we have developed makes it possible to implement :

- a digital barometric sensor ([MPL3115A2 module supplied with 3.3V](#)),
- a digital humidity sensor ([AM2315](#)),
- an anemometer (pulse counting)
- a weather vane (16 positions per turn),
- a light sensor in Lux ([Module TSL2591 Adafruit](#)),
- a UV level sensor ([module VEML6070](#)),
- a GPS receiver ([GY-GPS6MV2](#)),
- a Sigfox module ([Sigbeecard module from ATIM](#)), this module is delivered with a one-year subscription to the Sigfox network,
- [a 1000 mA.h LiPo battery](#),
- a Sunny Buddy solar charger from Sparkfun ([MPPT charger](#)),
- [a 10W photovoltaic panel](#),
- An IP55 housing with transparent cover (L ≥ 110 mm, W ≥ 80 mm h ≥ 45mm).

Here we have chosen a different humidity and temperature sensor AM2315 from the previous one. This sensor has better characteristics than the T9602.

	<b>T9602</b>	<b>AM2315</b>
<b>HR measurement range</b>	0 ... 100 %	0 ... 100 %
<b>Temperature measuring range</b>	-40 ... + 125 °C	-20 ... + 70 °C
<b>HR Accuracy</b>	± 2 %	± 2 %
<b>Temperature accuracy</b>	± 0.5 °C to ± 2 °C	± 0.1 °C to ± 1 °C
<b>Interface</b>	I2C	I2C
<b>Supply voltage</b>	3.3 V or 5 V	3.3 V to 5 V

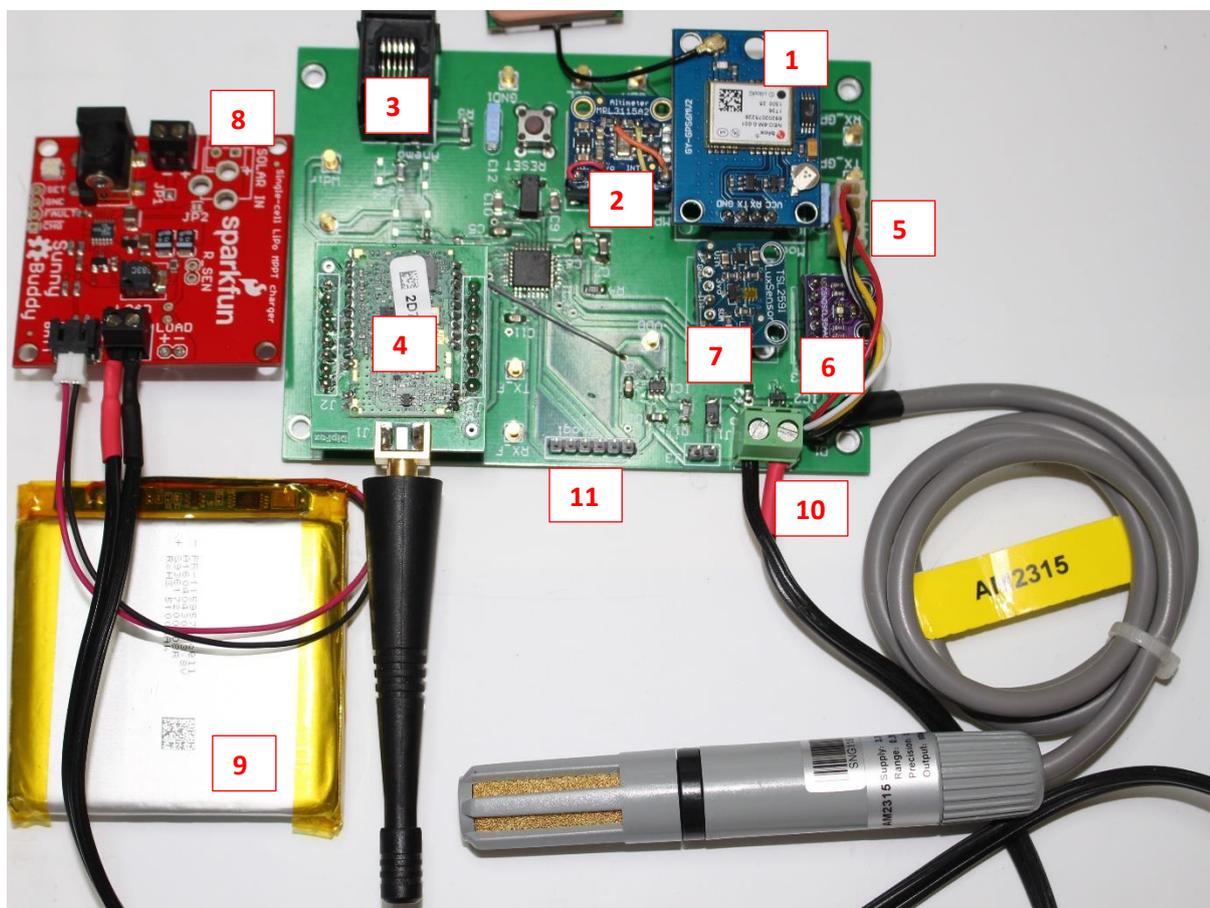
The address on the I2C bus of this sensor is 0xB8. To avoid self-heating and reduce power consumption, the sensor is in sleep mode by default. The user must therefore send an I2C frame to wake up the sensor and then read the measurement data. After reading the humidity and temperature registers (2 x 2 registers), the sensor starts a new measurement and returns to sleep mode.

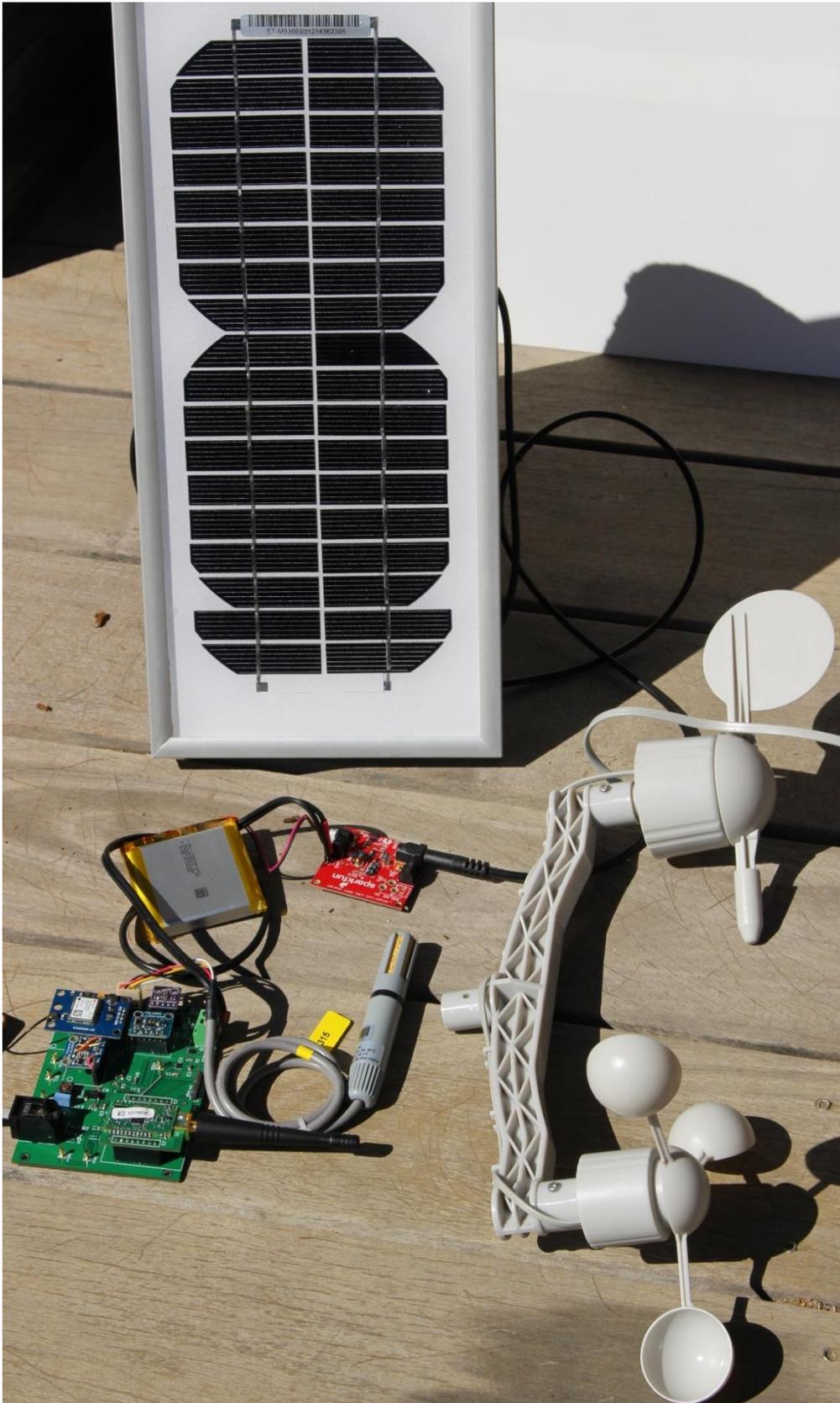
The results of the temperature and humidity measurements are given on 16 bits and are equivalent to 10 times the measured quantity. With regard to temperature, bit 15 (most significant bit) indicates a negative temperature when set to 1.

The useful registers of the AM2315 sensor are given in the table below.

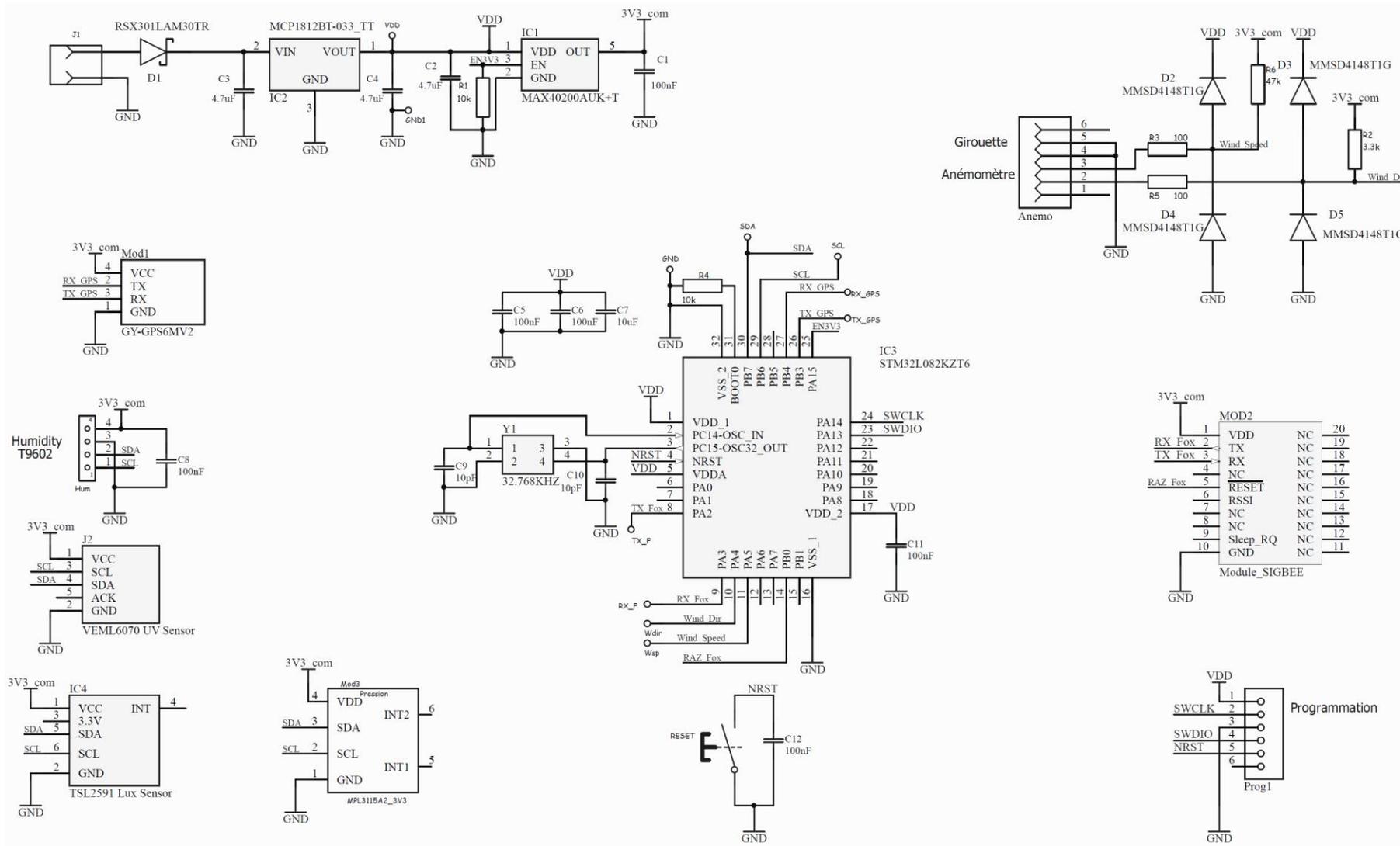
Register information	Add	Register information	Add	Register information	Add	Register information	Add
High RH	0x00	Model high	0x08	Users register a high	0x10	Retention	0x18
Low RH	0x01	Model low	0x09	Users register a low	0x11	Retention	0x19
High temp.	0x02	Version number	0x0A	Users register 2 high	0x12	Retention	0x1A
Low temp.	0x03	ID(24–31) Bit	0x0B	Users register 2 low	0x13	Retention	0x1B

The images below show the system as a whole. This is a photo of the first prototype where the Adafruit MPL3115A2 module had to be modified to overcome a malfunction when using all the sensors on the I2C bus. The correction of the problem is to opt for an MPL3115A2 module powered directly in 3V3 (without the voltage matching transistors only necessary with a microcontroller powered in 5V).





## 4.1. Structural diagram



## 4.2. Implementation of the layout

All documents necessary for the manufacture and development of the software are provided with the IO4.

The 'AltiumDesigner' folder contains the source files in native Altium Designer software (version 17.1.6) format, as well as a pdf version of the PCB and schematic.

The "Fabrication Documents" folder contains the necessary files (in a ZIP archive) to have the cards manufactured, for example through JLCPCB (<https://jlcpcb.com/>).

The "SoftWare Sources" folder contains all the files for developing the software:

"STM32\_LP Version" --> "STM32\_LP\_Software.zip" which is a complete archive of the project developed with the "STM32CubeIDE" IDE.

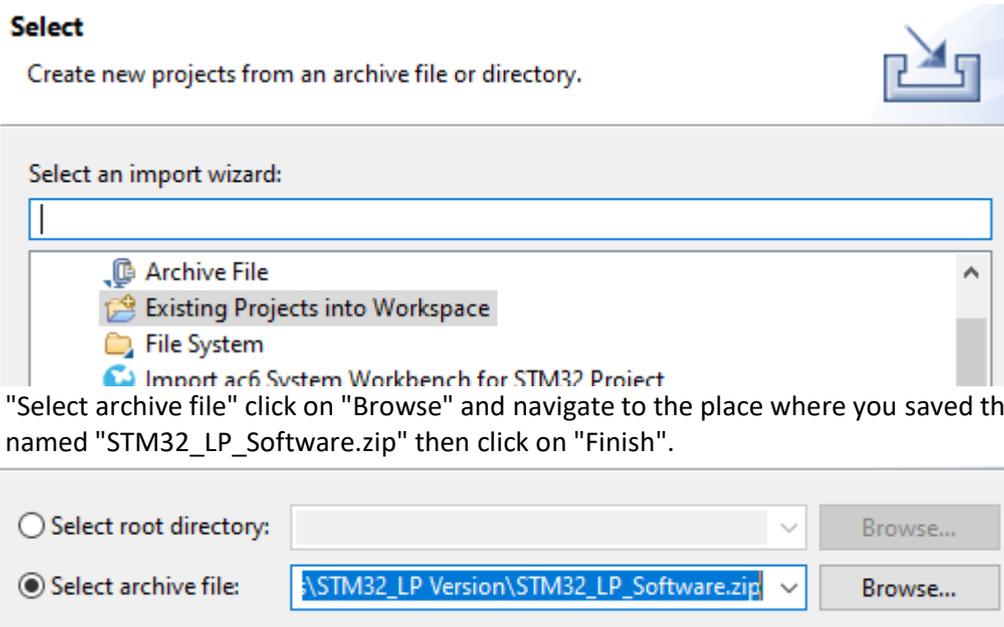
## 4.3. Software development

To carry out the operating programme for this card, we will use the free IDE from ST: STM32CubeIDE.

See the document entitled "Development Chain for STM32" for more information.

You need :

- Install the software,
- Create your workspace
- In your workspace click on "File→Import" then "existing Project into Workspace", tick the line



"Select archive file" click on "Browse" and navigate to the place where you saved the archive named "STM32\_LP\_Software.zip" then click on "Finish".

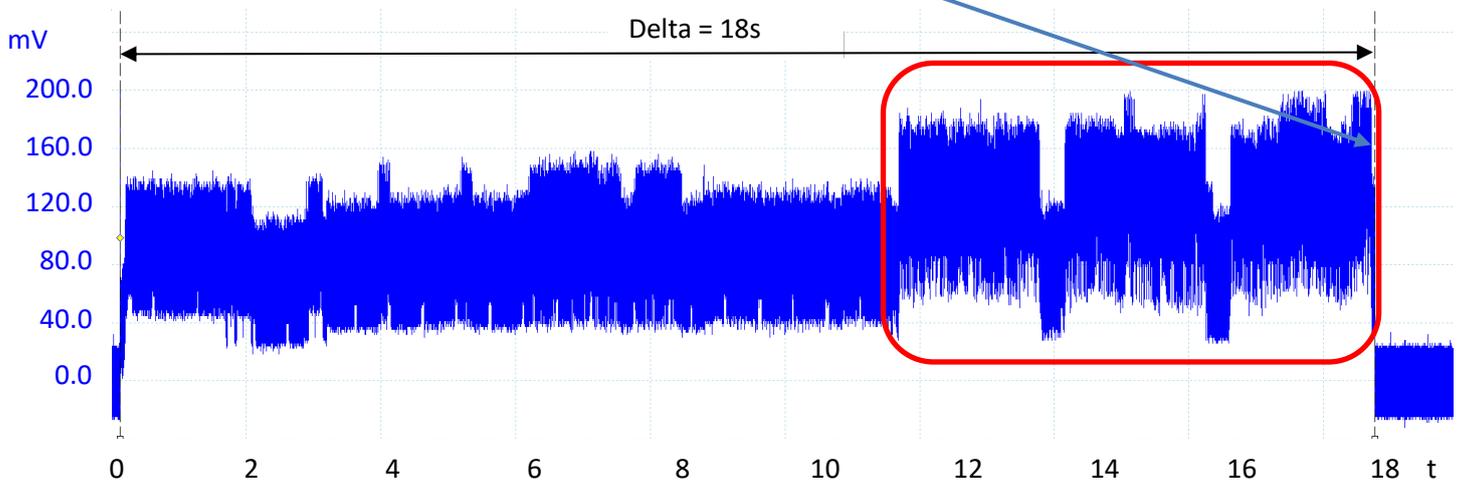
- Compile and transfer the code to the target,
- Test operation,
- Data processing by the Sigfox network requires the registration of the module used on the site and the transfer of the data to a database (for example on the Thingier.io site) see document appendix 1: "Sigfox".

## 4.4. Tests and measurements

### 4.4.a. Current measurement

To measure the current supplied by the power supply, we used an ammeter called "Current ranger" from LowPowerLab (<https://lowpowerlab.com/guide/currentranger/>). This device can measure currents from a few nA up to 3300 mA. It is also possible to visualize the current rate using an oscilloscope. This device is particularly useful for viewing and measuring currents in microcontroller systems (e.g. switching from normal to sleep mode).

The chronogram below shows the change from measurement mode (110 mA to 120 mA on average) to the card's sleep mode (700  $\mu$ A).



The three successive shipments to the Sigfox network (circled in red) can be clearly distinguished for a duration of about 8s. The total duration in "measurement" mode is 18s.

It can be estimated that our system consumes 120 mA for about 20 s, i.e. :

$$E_{mesure} = \frac{120 \cdot 10^{-3} \cdot 20}{3600} = 0,7 \cdot 10^{-3} A \cdot h$$

then 700  $\mu$ A for 15 min, i.e. :

$$E_{sleep} = \frac{700 \cdot 10^{-6} \cdot 15}{60} = 0,18 \cdot 10^{-3} A \cdot h$$

Overall this card requires an energy of  $0.7 + 0.18 = 0.88$  mAh so with a 1000 mAh battery, the system can operate for 500 hours, i.e. around 40 days. Combined with a 10 W solar collector, this system will be completely autonomous.

## 5. Autonomous measuring system with Wifi connection

This system has been developed in such a way as to add to the initial project a Wifi link which allows the measurements to be transmitted to a remote server accessible by an internet link via the local network.

The choice of the microcontroller was made on a module based on ESP32. In order to reduce the energy requirement (solar and battery power supply) as much as possible, we opted for the Wifi module:

- [DFRobot DFR0478](#).

This module from the DFRobot FireBeetle range is based on a 240 MHz ESP32 and features WiFi and Bluetooth connectivity. The FireBeetle ESP32 is lightweight, small in size and low power consumption, ideal for connected object projects.

### Characteristics:

- supply: - 3.7 Vdc via LiPo battery (not included) - 5 Vdc via microUSB plug (cable not included)  
 - consumption: - at work: 80 mA - standby: 10  $\mu$ A-  
 microcontroller: Tensilica LX6-  
 microprocessor: ESP32 at 240 MHz - flash memory: 16 MB - SRAM memory: 520 KB-  
 microUSB connector-  
 WiFi: - 802.11 b/g/n/d/e/l/k/r (802.11n up to 150 Mbps) - WPA/WPA2 - WEP/TKIP/AES-  
 Bluetooth 4.2 compatible BLE (Bluetooth Low Energy)-  
 inputs/outputs: - 10 x digital pins - SPI, I2C and I2S interface-  
 integrated LiPo charger with status led-  
 2 pins JST connector for LiPo batteries-  
 reset button-  
 dimensions: 58 x 29 mm-  
 weight: 24 g

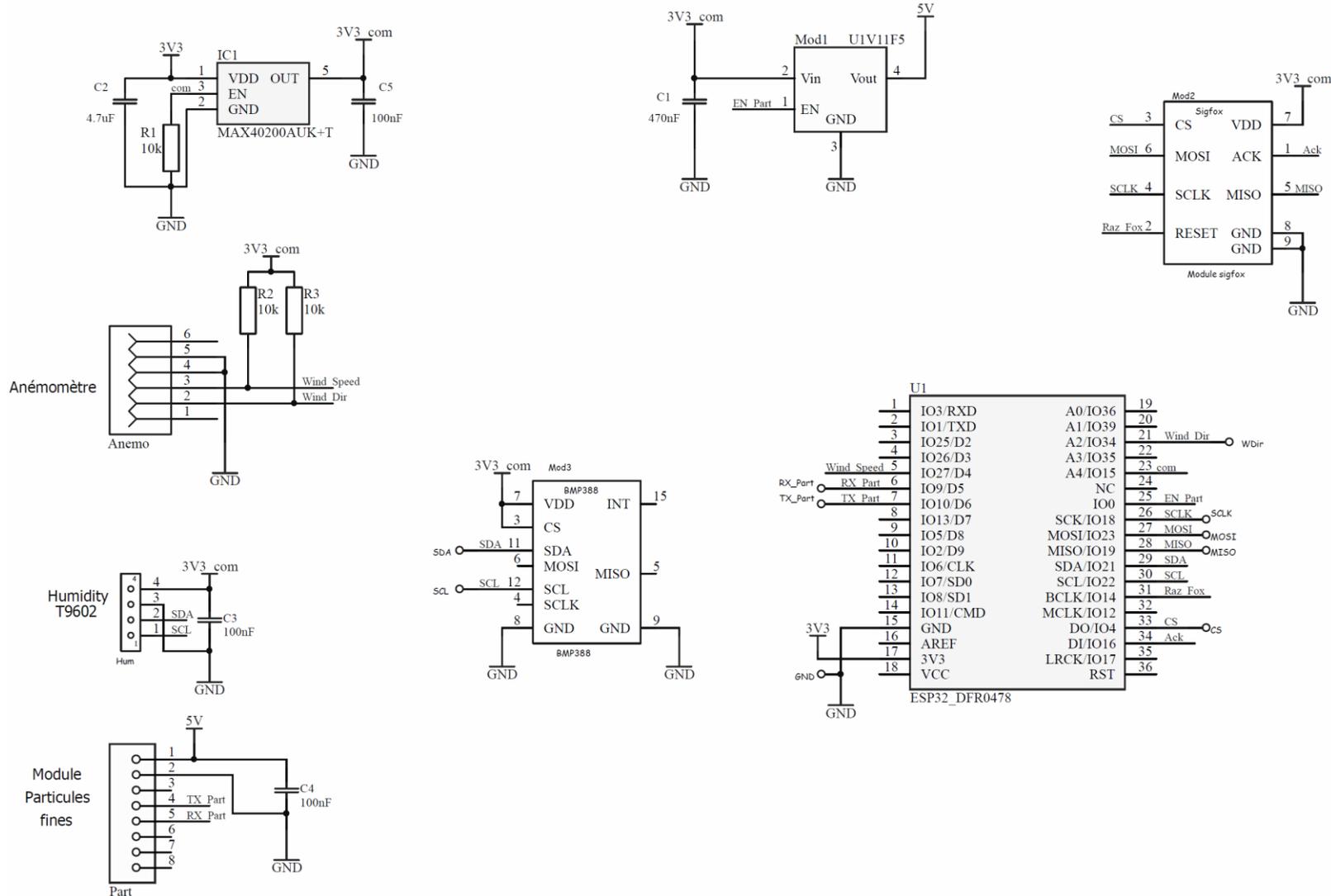
The sensors present on this card are :

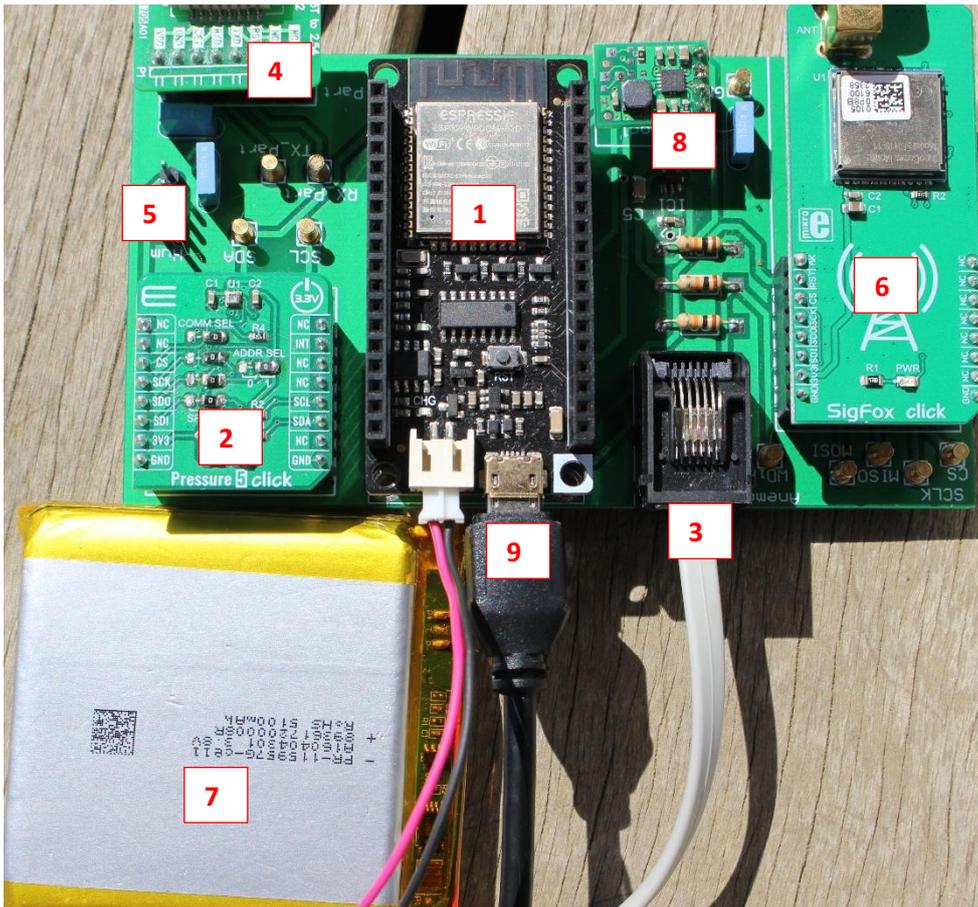
- a digital barometric sensor ([BMP388](#)),
- a digital humidity sensor ([T9602](#)),
- an anemometer (pulse counting)
- a weather vane (16 positions per turn),
- a fine particle sensor ([plantower-pms5003](#)),
- a Sigfox module ([ModuleSigFox click from mikroelektronica](#)),

The pressure sensor is different from the one on the STM32 board. We have indeed chosen a sensor with better characteristics :

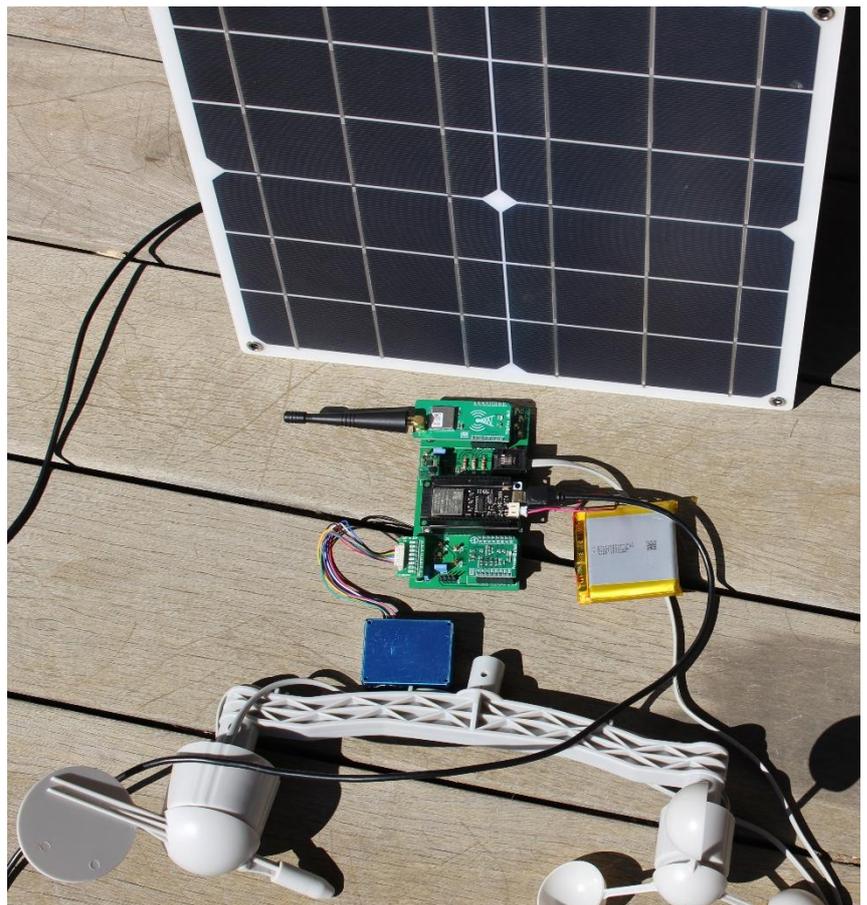
	<b>BMP388</b>	<b>MPL3115A2</b>
<b>Operation range</b>	300... 1250 hPa	200 ... 1100 hPa
<b>Supply voltage VDD</b>	1.65 V ... 3.6 V	1.95 V...3.6 V
<b>Interface</b>	I2C and SPI	I2C
<b>Absolute accuracy pressure (typ.)</b>	$\pm$ 0.40 hPa	$\pm$ 4.0 hPa
<b>Relative accuracy pressure (typ.)</b>	$\pm$ 0.08 hPa	$\pm$ 0.5 hPa

## 5.1. Structural scheme





1. ESP32 Firebeetle module.
2. Pressure sensor.
3. Connector for anemometer and wind vane.
4. Fine particle sensor.
5. Humidity and temperature sensor.
6. Sigfox module.
7. 3.8 V LiPo battery
8. DC/DC converter (3.3V → 5V).
9. Micro USB connector to connect to the solar collector 5 V/10 W.



## 5.2. Implementation of the layout

All documents necessary for the manufacture and development of the software are provided with the IO4.

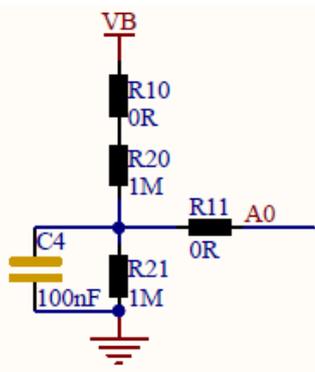
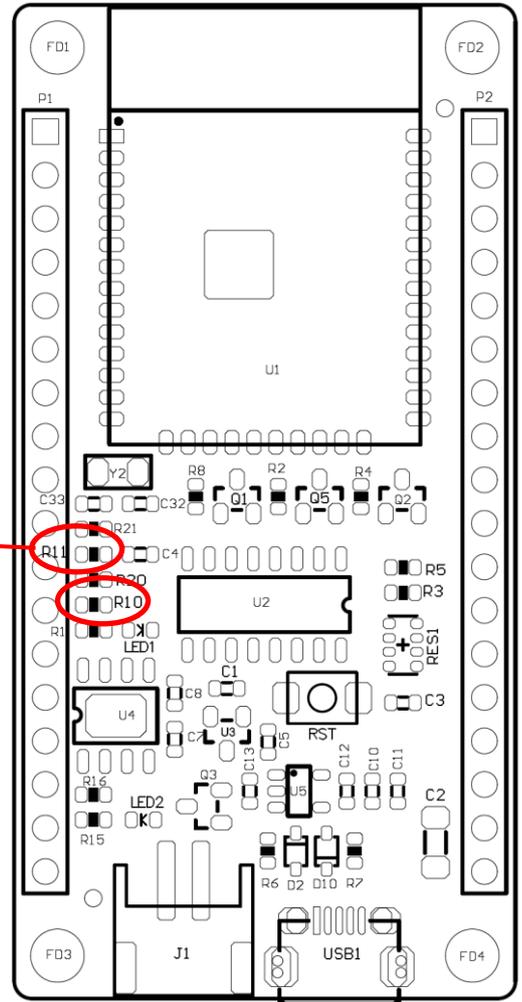
The 'AltiumDesigner' folder contains the source files in native Altium Designer software (version 17.1.6) format, as well as a pdf version of the PCB and schematic.

The "Fabrication Documents" folder contains the necessary files (in a ZIP archive) to have the cards manufactured, for example through JLCPCB (<https://jlcpcb.com/>).

The "SoftWare Sources" folder contains all the files for developing the software:

"FireBeetle Version" --> "TRYAT\_ESP32\_FireBeetle" for the project developed under VSCode + PlatformIO.

The Firebeetle module has a hardware function to measure the battery voltage (voltage divider bridge). Unfortunately, two 0 Ohm resistors (R10 and R11) are not mounted on the board. In order to use this function, a "jumper" must be soldered instead of R10 and R11.



## 5.3. Software development

The Arduino environment can be used to carry out the operating programme for this board. However, we have preferred a more professional but free IDE that allows more functionality than the Arduino tool, including the possibility of real debugging and OTA (Over The Air) downloading of the firmware.

Another advantage of this system is that the libraries needed for the project are downloaded. This allows extreme portability of the code.

See Annex 2 "PlatformIO\_VSCode" for more information.

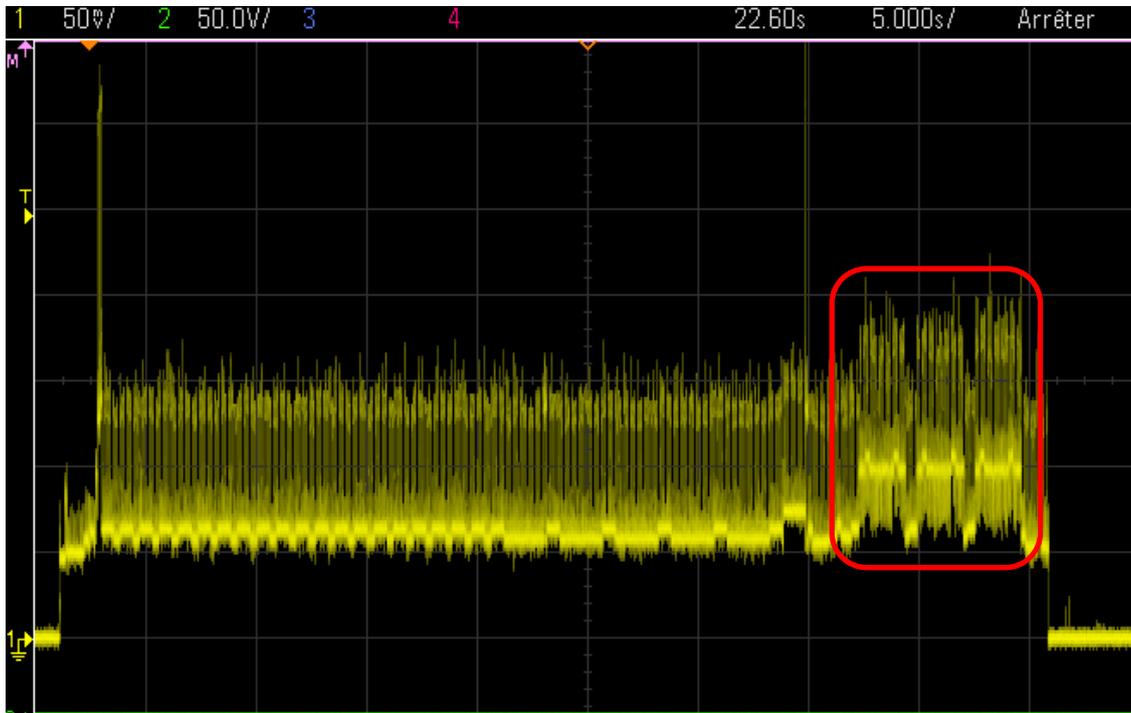
You need :

- Install the software,
- Copy the "TRYAT\_ESP32\_FireBeetle" folder from the "SoftWare Sources" folder → "FireBeetle Version" to your workspace,
- Open the "TRYAT\_ESP32\_FireBeetle" project,
- Change the code lines for the SSID and password of the wireless network,
- Specify the address of the site containing the database and its access port,
- Compile and transfer the code to the target,
- Test operation,
- The use of data by the Sigfox network requires the registration of the module on the site and the transfer of data to a database (e.g. on the Thingier.io site - see appendix entitled "Sigfox").

## 5.4. Tests and measurements

### 5.4.a. Current measurement

The chronogram below shows the change from sleep mode (500  $\mu$ A) to "run" mode of the Firebeetle module (150 mA on average).



The first peak corresponds to the connection to the Wifi network. On this chronogram, we do not see the transmission of information to the database by Wifi because this function had been deactivated at the time of the reading (database inaccessible). On the other hand, the three successive transmissions to the Sigfox network (circled in red) for a duration of about 8s are clearly visible. The total duration in "run" mode is 45 seconds.

We can estimate that our system consumes 150 mA for 45 s, i.e. :

$$E_{run} = \frac{150 \cdot 10^{-3} \cdot 45}{3600} = 1,9 \cdot 10^{-3} A \cdot h$$

then 500  $\mu$ A for 12 min, i.e. :

$$E_{sleep} = \frac{500 \cdot 10^{-6} \cdot 12}{60} = 0,1 \cdot 10^{-3} A \cdot h$$

Overall this card requires an energy of 2 mAh, so with a 1000 mAh battery, the system can operate for 500 hours, i.e. about 20 days. Combined with a 10 W solar collector, this system will be completely autonomous.

## 6. Stand-alone measuring system MKRFox version

This latest version of our measurement system has been developed to work with the Arduino environment. It is based on a module of the Arduino MKR family: the MKRFox 1200.

The use of this module will eventually allow the development of a WAN system (IoT) that uses a LoRaWAN network rather than the Sigfox network without changing the design of the board for a future project. But this is another story.

The Arduino MKR FOX1200 board is based on a SAMD21 Cortex M0+ clocked at 48 MHz under 32 bits and has a SigFox ATA8520 wireless module.

This 868 MHz low-speed wireless network protocol is long-range and dedicated to connected objects. The SigFox network covers a large area of France and Europe. This protocol is easy to configure and is energy efficient.

The Arduino MKR FOX 1200 board can be programmed via its micro-USB port (USB cable not included) with the Arduino software available for download on the Arduino [.cc](https://www.arduino.cc) website.

### Note:

- **Unlike other Arduino cards, the inputs/outputs operate at 3.3 Vdc. It is not possible to use 5 Vdc as input or output. A higher voltage will damage the card.**
- **- the purchase of this card includes a 1-year subscription to the SigFox network, see the registration procedure of the module on the document entitled "Sigfox". You can send up to 140 messages per day.**

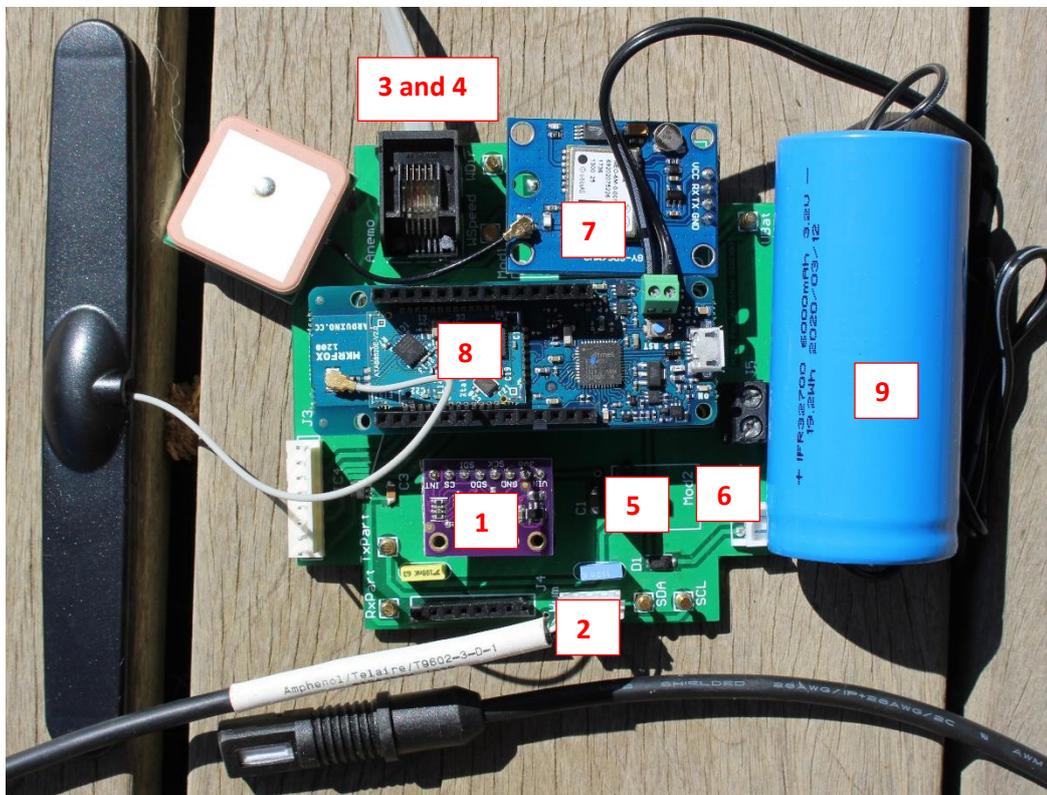
### Characteristics:

- supply:
  - via micro-USB port or - via screw terminal block, 3 Vdc (2 AA or AAA batteries not included or 1 LiFePo4 3.3V battery) or - via VIN pin (5 Vdc)
- microcontroller: SAMD21 Cortex M0+ 32bits at 48 MHz
- flash memory: 256 kB
- SRAM memory: 32 kB
- 8 I/O pins
- 12-pin PWM
- 7 ADC 8, 10 and 12 bit analogue inputs
- 1 DAC 10-bit analogue output
- serial bus, I2C and SPI
- disruption management
- micro-USB plug
- dimensions: 80 x 55 x 7 mm

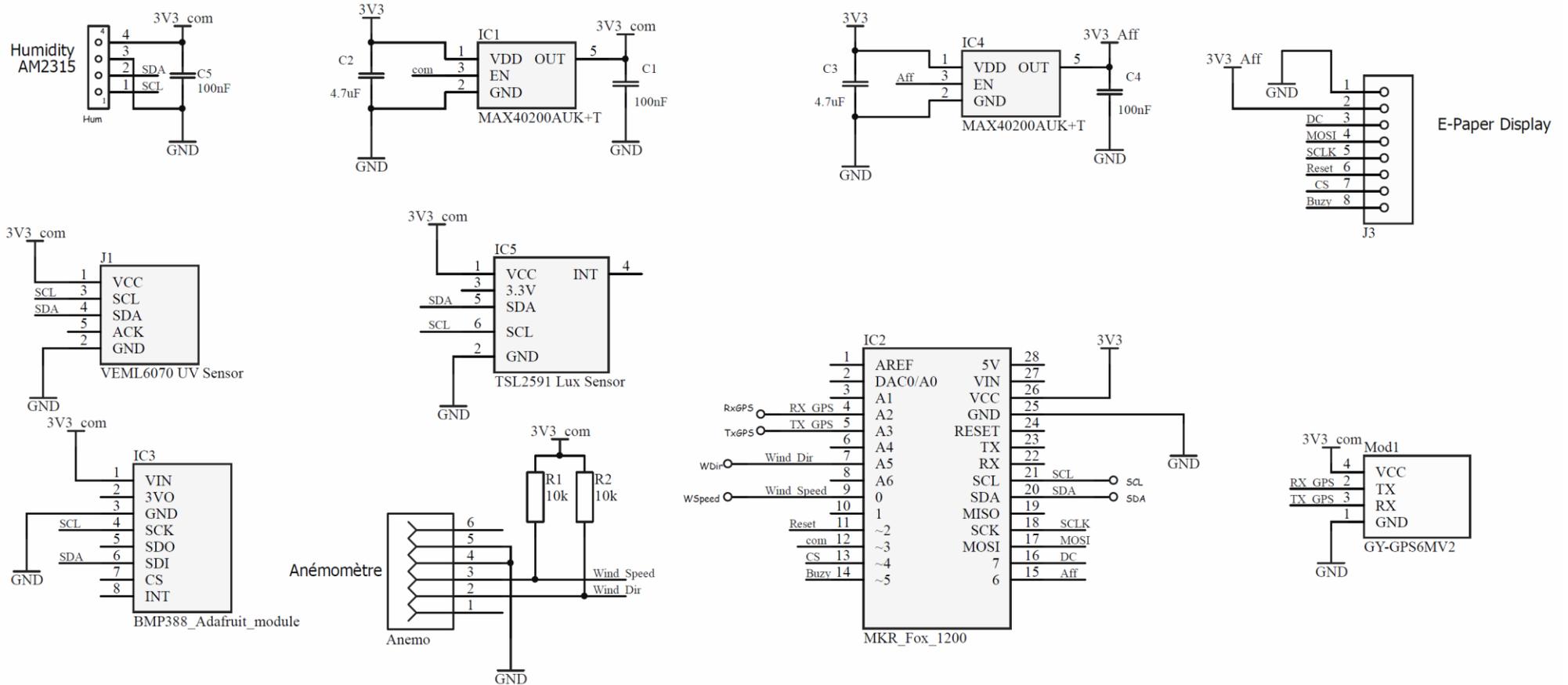
This layout is used to implement :

1. a digital barometric sensor ([BMP388](#)),
2. a digital humidity sensor ([AM2315](#)),
3. an anemometer (pulse counting)
4. a weather vane (16 positions per turn),
5. a light sensor in Lux ([Module TSL2591 Adafruit](#)),
6. a UV level sensor ([module VEMML6070](#)),
7. a GPS receiver ([GY-GPS6MV2](#)),

8. a Sigfox module integrated into the [MKR Fox 1200](#) card, this module is delivered with a one-year subscription to the Sigfox network,
9. [a LiFePo battery](#) 1100 mA.h,
10. a solar charger for LiFePo batteries ([LiFePocharger](#) or [MPPT module](#)),
11. [a 10W photovoltaic panel](#),
12. An IP55 housing with transparent cover (e.g. TK PS L: 94 mm, W: 94 mm, H: 57mm).



## 6.1. Structural scheme



## 6.2. Implementation of the layout

---

All documents necessary for the manufacture and development of the software are provided with the IO4.

The 'AltiumDesigner' folder contains the source files in native Altium Designer software (version 17.1.6) format, as well as a pdf version of the PCB and schematic.

The "Fabrication Documents" folder contains the necessary files (in a ZIP archive) to have the cards manufactured, for example through JLCPCB (<https://jlcpcb.com/>).

The "SoftWare Sources" folder contains all the files for developing the software:

" MKRFox Version" --> "TRYAT\_MKR\_Fox2" for the project developed under VSCode + PlatformIO

## 6.3. Software development

---

The Arduino environment can be used to carry out the operating programme for this card. However, we have preferred a more professional but free IDE that allows more functionality than the Arduino tool, including the possibility of real debugging and OTA (Over The Air) downloading of the firmware.

Another advantage of this system is that the libraries needed for the project are downloaded. This allows extreme portability of the code.

See Annex 2 "PlatformIO\_VSCode" for more information.

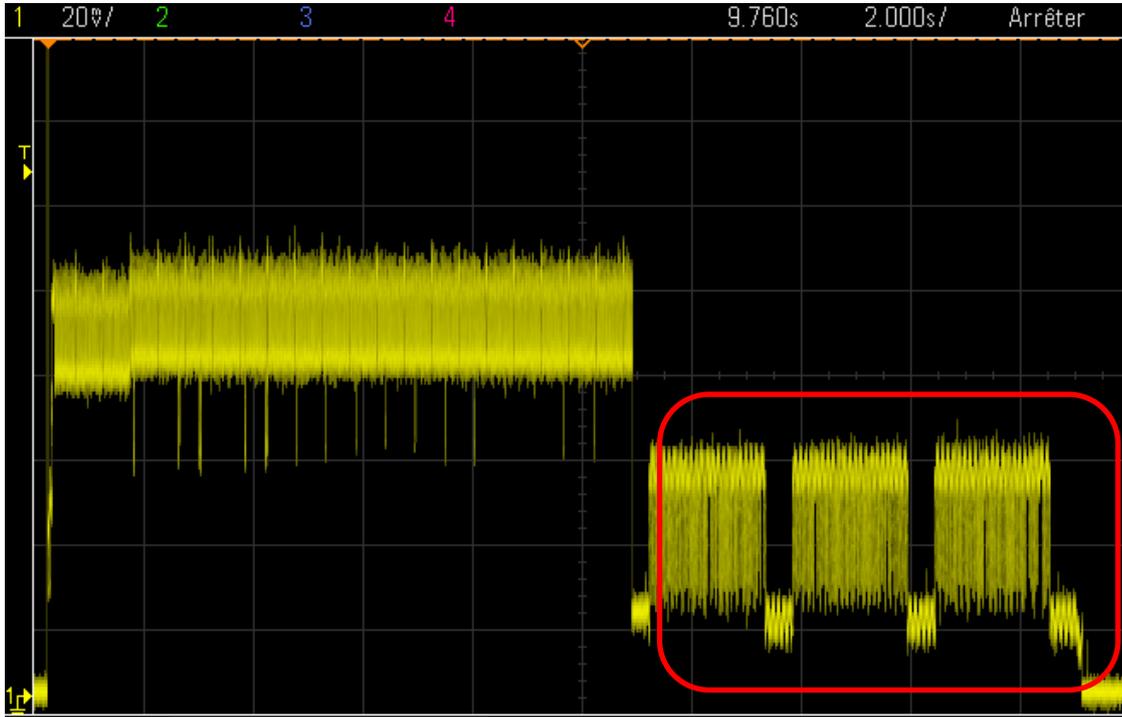
You need :

- Install the software,
- Copy the "TRYAT\_MKR\_Fox2" folder, which is located in the "SoftWare Sources" folder → "MKRFox Version", to your workspace,
- Open the project "TRYAT\_MKR\_Fox2",
- Compile and transfer the code to the target,
- Test operation,
- The use of data by the Sigfox network requires the registration of the module used on the site and the transfer of the data to a database (for example on the Thingier.io site, see the document entitled "Sigfox").

## 6.4. Tests and measurements

### 6.4.a. Current measurement

The chronogram below shows the transition from sleep mode (500  $\mu$ A) to run mode of the MKRFox module (100 mA on average).



The first peak corresponds to the connection to the Wifi network. On this chronogram, we do not see the transmission of information to the database by Wifi because this function had been deactivated at the time of the reading (database inaccessible). On the other hand, the three successive transmissions to the Sigfox network (circled in red) for a duration of about 8s are clearly visible. The total duration in "run" mode is 20 s.

We can estimate that our system consumes 150 mA (on estimate) for 20 s, i.e. :

$$E_{run} = \frac{150 \cdot 10^{-3} \cdot 20}{3600} = 0,8 \cdot 10^{-3} \cdot h$$

then 500  $\mu$ A for 12 min, i.e. :

$$E_{sleep} = \frac{500 \cdot 10^{-6} \cdot 12}{60} = 0,1 \cdot 10^{-3} A \cdot h$$

Overall this card requires an energy of 1 mAh, so with a 1000 mAh battery, the system can operate for 1000 hours, i.e. around 40 days. Combined with a 10 W solar collector, this system will be completely autonomous.

# Appendix 1: Sigfox

## 1. A bit of history

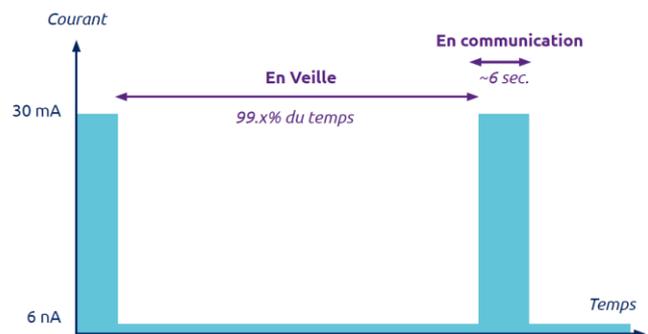
Founded in 2010 by Ludovic Le Moan and Christophe Fournet, this French company is based in Labège, near Toulouse, in the *IoT Valley*. It has created a low-speed cellular network, on licence-free frequency bands, dedicated to machine-to-machine (M2M) communications and IoT.

Tricolour operator Sigfox operates its long-range, low-speed IoT network in 70 countries. Temperature, vibrations, location... compatible connected objects can transmit small packets of information (the ceiling is 12 bytes) a maximum of 140 times a day, at low cost.

Devices connected to Sigfox are on standby almost all the time, which allows them to consume only a small amount of energy. They do not need to be connected to the mains, their life span can be up to ten years with a small battery. In its strategy for 2023, Sigfox plans to connect a billion objects to its network. This orientation is based on the opening of three countries - India, Russia and China. The company, which celebrates its 10th anniversary in 2019, has more than 30 million objects

contracted, including 15.4 million connected to the network and returning 26 million messages per day. Securitas is Sigfox's largest customer, with 2.8 million alarms. Sigfox has set itself the target of operating in 70 countries by the end of 2019, ten more than at present.

The network is bidirectional, but only on request of the object. It should be understood that in the Sigfox protocol, the network is at the service of the object, and not the opposite. The object thus asks itself an "update" to the network, which answers it if need be. The rest of the time, it is "deaf".



### 1.1. Sigfox cover

More than 2,000 antennas have been deployed by Sigfox in France, according to figures provided by the operator. The group's IoT network covers 94% of the tricolour population, as shown on the map below (updated in [real time on the Sigfox site](#)). The French enterprise is also present internationally, in 69 countries in addition to France. 21 have national coverage, such as Spain, Ireland, Denmark and Belgium. In the United States, Sigfox covered 30% of the population at the beginning of 2019. More than 1.1 billion people around the world can now benefit from an IoT Sigfox network, the company says.



**We've got you covered!**

Sigfox is already available in over 70 countries and regions and aims to cover 100% of the globe in the next few years...



## Ultra Narrow Band

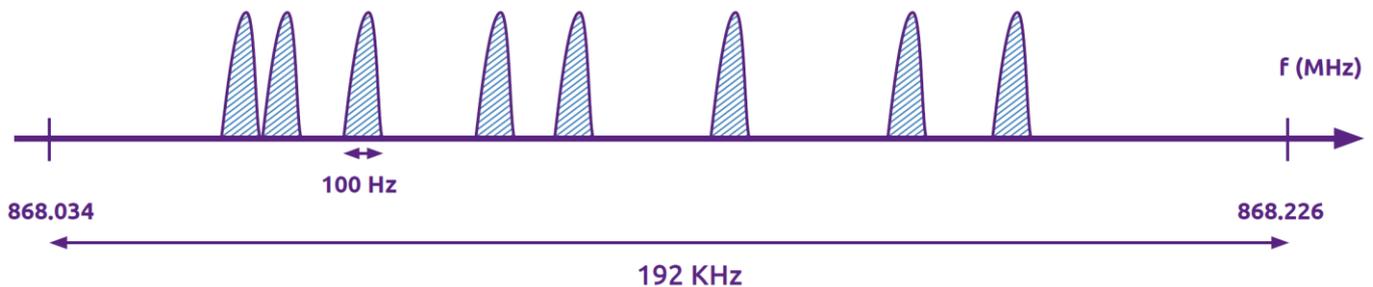
This technology allows Sigfox base stations to communicate over long distances without being affected by noise. The band used varies according to geographical areas:

Frequency (in MHz)	RC1	RC2	RC3	RC4	RC5	RC6	RC7
UL low boundary	868.034	902.104	923.104	920.704	923.204	865.104	868.704
UL center	868.130	902.200	923.200	920.800	923.300	865.200	868.800
UL high boundary	868.226	902.296	923.296	920.896	923.396	865.296	868.896
DL low boundary	869.429	905.104	922.104	922.204	922.204	866.204	869.004
DL center	869.525	905.200	922.200	922.300	922.300	866.300	869.100
DL high boundary	869.621	905.296	922.296	922.396	922.396	866.396	869.196

The seven geographical areas are:

- RC1:
  - Europe
  - Overseas France: French Guiana, French Polynesia, Guadeloupe, Martinique, Mayotte, New Caledonia, Reunion.
  - Middle East and Africa: Botswana, Kenya, Nigeria, Oman, South Africa, Swaziland, United Arab Emirates.
- RC2: Brazil, Canada, Mexico, Puerto Rico, USA.
- RC3: Japan.
- RC4:
  - Latin America: Argentina, Chile, Colombia, Costa Rica, Ecuador, El Salvador, Guatemala, Honduras, Nicaragua, Panama, Peru, Trinidad & Tobago, Uruguay.
  - Asia Pacific: Australia, Hong Kong, Indonesia, Malaysia, New Zealand, Singapore, Taiwan, Thailand.
- RC5: South Korea.
- RC6: India.
- RC7: Russia.

Sigfox uses 192 KHz of the unlicensed ISM band to exchange messages by radio link. The transmission technique is the so-called Ultra Narrow Band (UNB). Each message occupies 100 Hz (ETSI Europe zones) or 600 Hz (FCC USA zones) and is transferred at a rate of 100 or 600 bits per second depending on the region.



Sigfox base stations monitor the entire 192 kHz spectrum and search for UNB signals to be demodulated.

The **Ultra Narrow Band** allows :

- to have a good immunity to interference ;
- due to the low spread, to avoid data collisions. This makes it a reliable protocol.

## 2. Sigfox module registration

In each SIGFOX module, the **ID** and **PAC** couple is essential for the identification and referencing of an object. Each module (or product) has a unique identifier (ID). This is linked to the CAP (Porting Authorization Code). The CAMP proves the identity of the module owner. This ID/PAC pair is obligatorily provided when provisioning a module on the SIGFOX portal (<https://buy.sigfox.com/>). Without these identifiers, it is impossible to communicate in SIGFOX.

These identifiers are either given by the supplier of the product, or obtained by querying the SIGFOX module. You will find two types of modules (or objects) : those with a one-year subscription to the site and those without a subscription. In the latter case, you will have to subscribe to Sigfox (prices range from 6 € per year per object with 2 messages per day to 14 € per object with 140 messages per day).

	Discovery € 6.00 / Device	Discovery € 14.00 / Device
Number of devices:	1	1
Your Price:	€ 6.00 (excl. VAT)	€ 14.00 (excl. VAT)
Messages per device per day	2, 50, 100, 140	2, 50, 100, 140
Subscription duration (years)	1 year	1 year
Atlas Native	Off <input type="checkbox"/> On	Off <input type="checkbox"/> On
Global connectivity	✓	✓
Activation period	1 year	1 year
Access to SigFox Cloud	✓	✓
Online Support	✓	✓
Support	✓	✓

In case your module is sold with a subscription included, you have to register it in the Sigfox infrastructure at the following address:

<https://buy.sigfox.com/activate>

Then simply fill in the country where the module will work,

On the next page, fill in the necessary fields (figure opposite) and click on the "Next" button at the bottom right. The ID and CAMP data will be checked. If everything is OK, you will get a message indicating that your kit is recognised.



Country

Devkit

Account

Confirmation

Next >

### Provide your DevKit's details for identification

Device ID \*

Up to 8 numbers and letters (from A to F)

PAC \*

Exactly 16 numbers and letters (from A to F)

### Tell us about your project

Purpose of your project \*

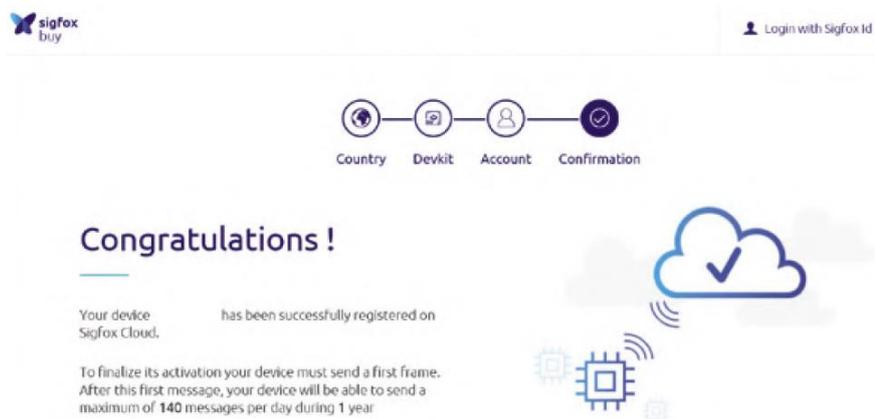
Description \*

Tell us more about your project!

< Back
Next >

Click on the "Next" button again and go to the second activation window (figure opposite). Fill in the fields and click on the "Activate your kit" button.

If there are no typing errors, you will receive a confirmation of your registration (figure below).



The activation of your module will not be completed until the card has sent its first message to the Sigfox infrastructure (see below). Only then is the device recognised and activated in the Sigfox network, and you can download your 140 12-byte messages per day free of charge for one year (four 8-byte messages per day can be received as a download, e.g. for remote configuration).

You should have received a confirmation at the email address provided. You must use this message to assign a password to log into the Sigfox *backend*. Click on the *SET YOUR SIGFOX ID PASSWORD* field and the input mask will appear (figure below).

Fill in the fields as usual and enter your login data in the next window. After clicking *Login*, the created profile is displayed. The login procedure to the Sigfox infrastructure is finished! You can now log out.

If your module is sold without a subscription, it is up to you to take out a subscription with Sigfox at the following address:

<https://buy.sigfox.com/buy>

Once you have chosen your country and clicked on "Subscribe", you must choose your subscription level according to the number of daily messages you wish to receive. Then click on "Buy".

You will then be asked to create your account on the Sigfox platform. On this window, you will see a summary of what your subscription will cost you (see figure on next page). Fill in the fields and click on the "Proceed to payment" button.

**Where is your company based?**

Choose the country of domiciliation of your company

Country	Status
France	Active
Germany	Active
Hong Kong	Active
Ireland	Active

Your country is **Active**: you can purchase subscriptions, get support from your local Sigfox Operator, and make your devices communicate in all other **Active** countries.

Your country is **Inactive**: contact your Sigfox Operator.

Your country is not listed: [contact us](#).

[Check Coverage](#)

**Sigfox France**  
 France  
 Sigfox, the world's leading Internet of things (IoT) connectivity service.

**Sigfox France main office**  
 Bat E-volution, 425 rue Jean Rostand  
 31670 Labège  
<https://www.sigfox.com>

### Create your account

Already have a partner account or a Sigfox backend account? [Log in](#)

#### User information

First Name \*  Last Name \*

Email \*  Phone Number (international format) \*

#### Company information

Business Name \*  Street Address \*

Country \*  City \*

Your operator must be in the same country as your company

Zip Code \*  Business ID

International VAT number

- Keep me informed about service and offer updates
- I have read, understood and agree to the [Terms & Conditions](#)

[< Back](#)

[Proceed to payment >](#)

### Your Order

Sigfox Operator: Sigfox France		
Subscription duration	1 year	
Messages/day	100	
Atlas Native	no	
Unit Price	€ 11.00	x 1 units
Subtotal (excl. VAT):	1 x € 11.00	€ 11.00
VAT (20%):		€ 2.20
<b>OrderTotal:</b>		<b>€ 13.20</b>

The next step is the checkout...

You will then have to register a module with its ID and CAP code and choose a password for your access to the sigfox backend.

## 2.1. Sigfox's backend infrastructure

Once you have registered your module, you can send messages to the Sigfox cloud (the sending procedure varies from module to module). To view the messages you have sent, go to the Sigfox portal and enter your email address and the password you entered during registration.

The screenshot shows the Sigfox portal interface. At the top, there is a navigation bar with the Sigfox logo and menu items: DEVICE, DEVICE TYPE, USER, and GROUP. On the right side of the navigation bar, there are icons for user profile, help, and a share icon. The main content area features a large banner with the text "Welcome to sigfox portal" and a decorative graphic of overlapping purple and blue lines. Below the banner, there is a "Release 9.8" section with a sub-heading "Devices and contract" and a link "Simultaneous operations on a device [correction]". On the right side of the main content area, there is a date indicator "21 SEPTEMBER 2020". A left sidebar contains a menu with items: NEWS, SERVICE MAPS, and KNOWN ISSUES.

Now click on the Device tab. You will see an overview of the Sigfox devices registered or activated by you (below).  
 - **Device type:** Within the group, you can sort and classify identical stations (type, structure, functions) according

Communication status	Device type	Group	Id	Last seen	Name	Token state
<input type="radio"/>	TRYAT_MKRFFox	Joly Dominique	1DAD70	2020-09-18 22:48:30	MKRFFox1	<input checked="" type="checkbox"/>
<input type="radio"/>	TRYAT_Firebeetle	Joly Dominique	26C59D	2020-09-16 10:05:53	MikroElektronika_DevKit_2-device	<input checked="" type="checkbox"/>
<input type="radio"/>	TRYAT_MKRFFox	Joly Dominique	1DC4E1	2020-08-29 18:21:35	MKRFFox2	<input checked="" type="checkbox"/>
<input type="radio"/>	TRYAT_STM32	Joly Dominique	2D78DE	2020-08-27 23:45:15	Atim_Nano_N8	<input checked="" type="checkbox"/>

to the device type.

- **Group :** The name of the group has been automatically created by Sigfox and you cannot modify it. This group now contains all your Sigfox devices. Click on the name of the group to get information (cannot be modified).
- **Id:** the unique identification number of the station.
- **Last seen :** the infrastructure enters here the date and time of the last communication from the station, i.e. the date on which the last telegram was received by the station. N/A means that the infrastructure has not yet received any telegrams from the station.
- **Name:** The name of the station.
- **Token state:** a token is a transmission right for a station on the Sigfox network. When you sign a contract with Sigfox, you receive a certain number of tokens. In order for a station in the Sigfox network to become active, a token is assigned to it. It can then send and receive data over the network. When you purchase certain modules and register them in the Sigfox infrastructure, you receive a free token for one year when you register this station.

### 3. Viewing received messages

In the list of active Sigfox devices, click (exactly) on the Station *Id* field to bring up the device information page and there, on the left, on *Messages*.

Time	Seq Num	Data / Decoding	LQI	Callbacks	Location
2020-08-27 23:45:15	381	00d201808951030d00000000			
2020-08-27 23:30:14	380	00dc01808a4f000000000000			

The following window (above) lists all telegrams received by the Sigfox infrastructure. The *Time* field is self-explanatory; the *LQI* (Link Quality Indicator) field shows the field strength with which the Sigfox telegram was

received by the base stations. If you place the cursor on this field, you will see a description of the reception quality and the local operator of the Sigfox network in which the Sigfox station is located will be displayed. This operator is not necessarily called Sigfox. When you click on *Location*, a map shows the approximate location of the Sigfox station.

Now we come to the important information in the Data/Decoding field, where the content of the payload is displayed, once as raw data and once in decoded form. The infrastructure does not yet know how to interpret the raw data, it needs to be taught. It is important that the content of the message is well detailed and that the order and size of each piece of information transmitted is respected. For example, in our TRYAT STM32LP application, we decided to transmit according to the following table:

Name of the data	Number of bytes	Type of data	unit
Temperature	2	(signed integer) int16_t	Tenth degree
Pressure	3	(unsigned integer) uint24_t	Pascal
Humidity	1	(unsigned integer) uint8_t	% HR
Wind speed	1	(unsigned integer) uint8_t	km/h
Wind direction	1	(unsigned integer) uint8_t	N x 22.5 °
Light intensity	2	(unsigned integer) uint16_t	Lux
UV rate	2	(unsigned integer) uint16_t	Without

Knowing the structure of the data sent to the network, it is possible to improve the presentation of the raw data by associating a name for the data and its size. To do this, click on the tab " DEVICE TYPE ".

#### Device type 'TRYAT\_STM32' - Information

Disengage sequence number Restart Edit Delete

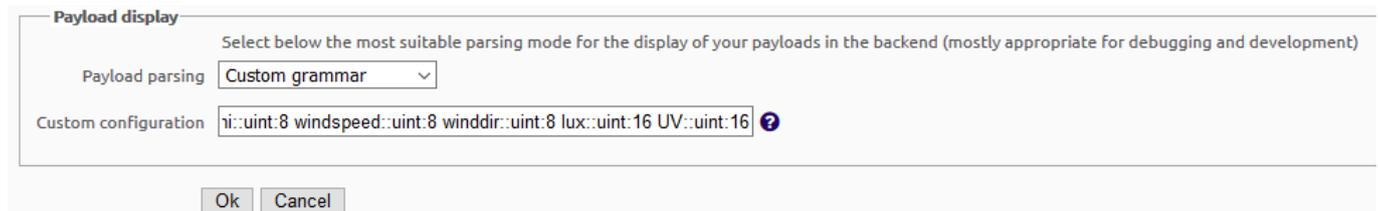
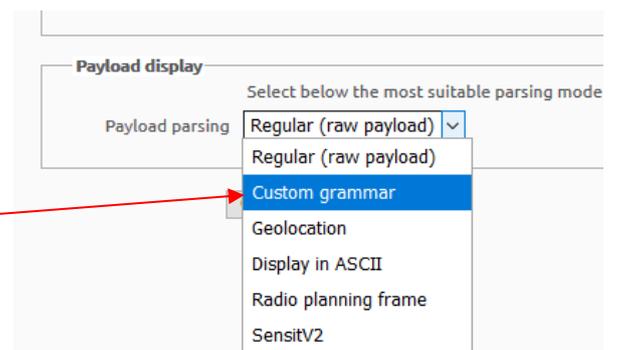
Id: 5f0eb758e833d97984593d9c  
Name: TRYAT\_STM32

Then on the type to be modified and finally on Edit.

In the next editing window, you can adjust the datatype. In the "Payload display" area, you can change the "Payload parsing" by selecting "Custom grammar".

All that remains is to adapt the formatting of the data according to the information transmitted in the message (according to the previous table). This gives :

temp::int:16 pres::uint:24 humi::uint:8 windspeed::uint:8 winddir::uint:8 lux::uint:16 UV::uint:16



Back in the *Messages* window, you will see one of the additional lines for this station with the payload contents correctly formatted (see figure below).

Time	Seq Num	Data / Decoding	LQI
2020-08-27 23:45:15	381	00d201808951030d00000000 temp: 210 pres: 98441 humi: 81 windspeed: 3 winddir: 13 lux: 0 UV: 0	
2020-08-27 23:30:14	380	00dc01808a4f000000000000 temp: 220 pres: 98442 humi: 79 windspeed: 0 winddir: 0 lux: 0 UV: 0	

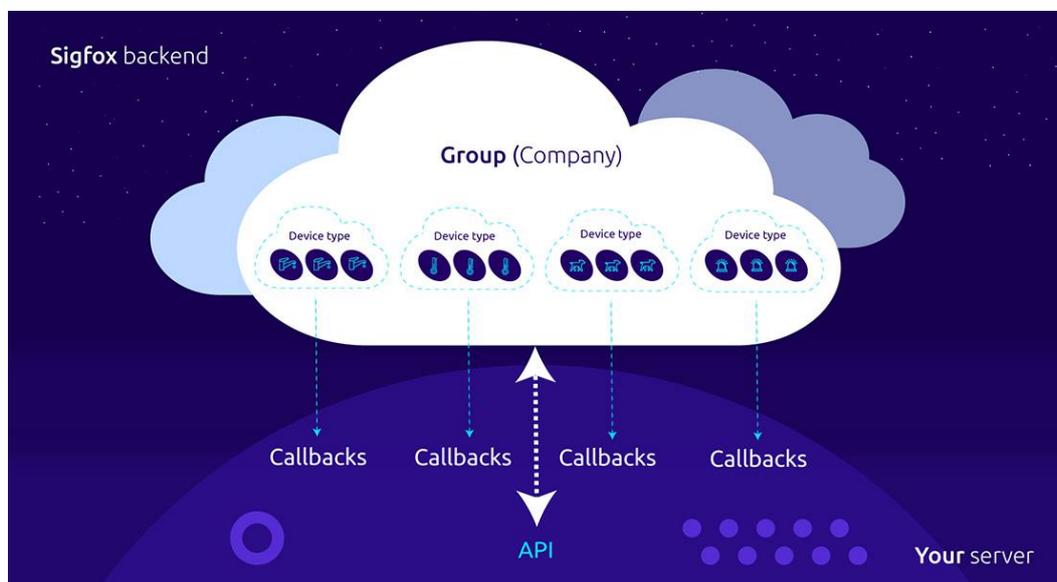
#### 4. Data recovery and exploitation

Two methods can be distinguished for recovering data from the Sigfox Cloud:

The Callback method or the method using API REST.

Let's come back to the functioning of the Sigfox Cloud. A user (company) is generally represented by a "Group", which contains at least one "Device type". A device type groups all the units of a same product, to allow them to behave in exactly the same way when the Sigfox network receives a message.

Recall-based integration is defined by device type. Subscription management is also assigned on a per-device basis. All this is shown in the diagram below:



### The "Callback" method

Callback requests are HTTP requests that are (one-way) notification messages and are linked to a type of device: your server can only obtain data through callbacks, not manage devices through the cloud.

When Sigfox Cloud receives a message from a sending device, it instantly generates a reminder message and sends it to your platform servers. This way, you don't have to regularly check for new data: the system sends you any new content as soon as it is received.

### API REST

REST API requests are bi-directional HTTP requests that work at group level: your server can request and receive data from the Sigfox Cloud via the REST API.

The REST API allows you to automatically register and manage all your devices on the Sigfox Cloud, including their subscription lifecycle. This way, you don't need to connect to the Sigfox backend to perform recurring operations such as recall declaration, device management, etc. You need to use the REST API to manage your registered devices directly from your platform.

## 5. Callback configuration

To enable the transfer of information from the Sigfox Cloud to our server, the infrastructure must of course be correctly configured, ensuring that the data is sent to the desired recipient.

To configure the callback, you must first connect to the Sigfox infrastructure. On the main page, click on the "Device Type" tab and in the list that appears click on the name of the type you wish to configure. The preview page of this device will then appear. In the list on the left, click on Callbacks. If you now click on the small New button at the top right of this window, a list will appear with all the available callback types that work with both dashboard and cloud programs. From these, choose "Custom callback".

**Device type 'TRYAT\_STM32' - Information**

- Id:** 5f0eb758e833d97984593d9c
- Name:** TRYAT\_STM32
- Description:** joly\_dom\_8609\_ff3f
- Keep alive:** N/A
- Subscription automatic renewal:**  ⓘ
- Group:** Joly Dominique
- Payload display:** Custom
- Downlink mode:** CALLBACK
- Contracts:**
  - 1. joly\_dom\_8609\_ff3f (no token left - geoloc: no, end date: 2022-07-15)
- Alert Email:**
- Creation date:** 2020-07-15 09:59:20
- Created by:** buy.sigfox.com
- Last edition date:** 2020-09-19 22:53:33
- Last edited by:** Dominique Joly

Navigation: DEVICE | **DEVICE TYPE** | USER | GROUP

Device type 'TRYAT\_STM32' - Callbacks New

In the window that appears now, you will be able to configure the reminder and the way the data is sent from the Sigfox infrastructure to your database.

### Device type TRYAT\_STM32 - Callback edition

**Callbacks**

Type:

Channel:

Custom payload config: temp::int:16 pres::uint:24 humi::uint:8 windspeed::uint:8 winddir::uint:8 lux::uint:16 UV: ?

URL syntax: [https://host/path?id={device}&time={time}&key1={var1}&key2={var2}...](#)  
 Available variables: device, time, data, seqNumber, deviceId  
 Custom variables: customData#temp, customData#humi, customData#windspeed, customData#winddir, customData#lux, customData#UV

Url pattern: <https://api.thinger.io/v1/users/Numerobis/buckets/myTRYAT/data>

Use HTTP Method:

Send SNI:  (Server Name Indication) for SSL/TLS connections

Headers: Authorization: Bearer eyJhbGciOiJIUzI1NiIs...  
 header: value

Content type:

Body:

```
{
  "Device-ID" : "{device}",
  "Date" : {time},
  "Temperature" : {customData#temp},
  "Pression" : {customData#pres},
  "Humidite" : {customData#humi},
  "Vit_Vent" : {customData#windspeed},
  "Dir_Vent" : {customData#winddir},
  "Lux" : {customData#lux},
  "UV" : {customData#UV}
}
```

In this presentation, we will only show one (most common) example of Sigfox recall use (for more information go to the Sigfox website: <https://support.sigfox.com/docs/custom-callback-creation>).

There are free platforms available to quickly and easily create dashboards to use the recovered data. In this example we used "thinger.io" (<https://thinger.io/>).

Let's start at the top of the figure above. Under "Type", the correct parameters are DATA and UPLINK, and for Channel, choose URL, as it is through an HTTP connection that we will talk to *thinger.io*.

As seen in the previous paragraph, the data transmitted by the module must be matched to their meaning (type of information and format). Internally, a freely chosen variable name is assigned to each value. These names are used to specify the further processing that the infrastructure performs on the variable values. The way in which the Sigfox payload is decoded is specified in the *Custom payload* field and should reflect the way in which the data values are incorporated into the payload on the module. For the Sigfox payload to be correctly decomposed into variables, the sequence of the data, the relevant data types and the storage format must be specified precisely.

In our example, the *Custom payload config* should contain seven items, each appearing **precisely** as follows. :

```
temp::int:16 pres::uint:24 humi::uint:8 windspeed::uint:8 winddir::uint:8 lux::uint:16 UV::uint:16
```

Under *URL pattern*, you specify the destination URL to your platform (in our example on *thinger.io*).

As the HTTP method, choose *POST*. Depending on the security protocol provided by your platform, it is possible in the next step to set up authentication for data transfer. This is done in the two fields next to the *headers*.

In our example, for the *header* field enter the word *Authorization* and next to it, in the *value* field, enter *Bearer* followed by a space and the access token that is created by the target site (here by *thinger.io*) *Bearer eyJhbGciOiJIUzI1NiIs...* etc.

In the *Content type* field, enter "application/json", indicating that the body of the message is in JSON format.

In the last step, we specify exactly what will be sent to our database (here *thinger.io*). The entry (in JSON format) in the *Body* field must be as follows. Be careful to respect the formalism.

```
{
  "Device-ID": "{device}",
  "Date": {time},
  "Temperature": {customData#temp},
  "Pressure": {customData#pres},
  "Humidity": {customData#humi},
  "Vit_Vent": {customData#windspeed},
  "Dir_Vent": {customData#winddir},
  "Lux": {customData#lux},
  "UV": {customData#UV}.
}
```

We have therefore

- ✚ "Device-ID": "{device}" which will give the unique device number of the Sigfox module, which is known to the infrastructure. This information is sent along with the measured data so that the sigfox station can be properly identified afterwards.
- ✚ "Date": {time} which will give the date and time of reception of the data by the Sigfox Cloud (in Unix Timestamp format). This information will allow a time stamping of the data,

Then follows the information recovered from the frame. For example, a variable with the name *Temperature* will be sent. The value of this variable corresponds to the value of the infrastructure variable called *temp*, which we previously specified in the *Custom payload config* field. The keyword *customData#* indicates that the value of a variable unpacked using the *Custom payload config field* should be inserted. ).

Now our system is ready to use the data from our weather station. If everything is correct on the Sigfox side, a green arrow will appear in the "Callback" column when you go to the reading of the received messages (see the paragraph visualization of the received messages). If there is a problem, this arrow will be red...

2020-08-27 18:14:51	362	01400180832d03000995014d temp: 320 pres: 98435 humi: 45 windspeed: 3 winddir: 0 lux: 2453 UV: 333		
2020-08-27 17:59:48	361	01400180852a030205f500ca temp: 320 pres: 98437 humi: 42 windspeed: 3 winddir: 2 lux: 1525		

# Appendix 2: PlatformIO and VSCode

## 1. What is it all about?

PlatformIO is an integrated development environment (IDE) for embedded systems that can be integrated with many text editors or IDEs. According to the official website (<https://docs.platformio.org/en/latest/>), the most complete and simple integration is with Microsoft Visual Studio (VS).

- It is cross-platform and includes an integrated debugger.
- It supports over 800 cards and automatically installs everything that is needed once a card is selected.

Without going into too much detail about the implementation of PlatformIO, the work cycle of the project developed using PlatformIO is as follows:

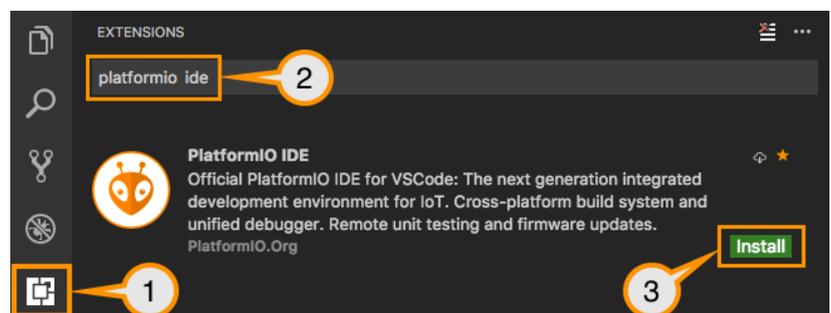
- Users choose the layout they are interested in from "platformio.ini" (project configuration file).
- Based on this list of cards, PlatformIO downloads the necessary tool strings and installs them automatically.
- Users develop the code and PlatformIO ensures that it is compiled, prepared and uploaded to the chosen card.

Visual Studio Code is a lightweight but powerful source code editor that runs on your PC and is available for Windows, MacOS and Linux. It supports JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Python, PHP, Go) and runtime environments (such as .NET and Unity).

## 2. Installation

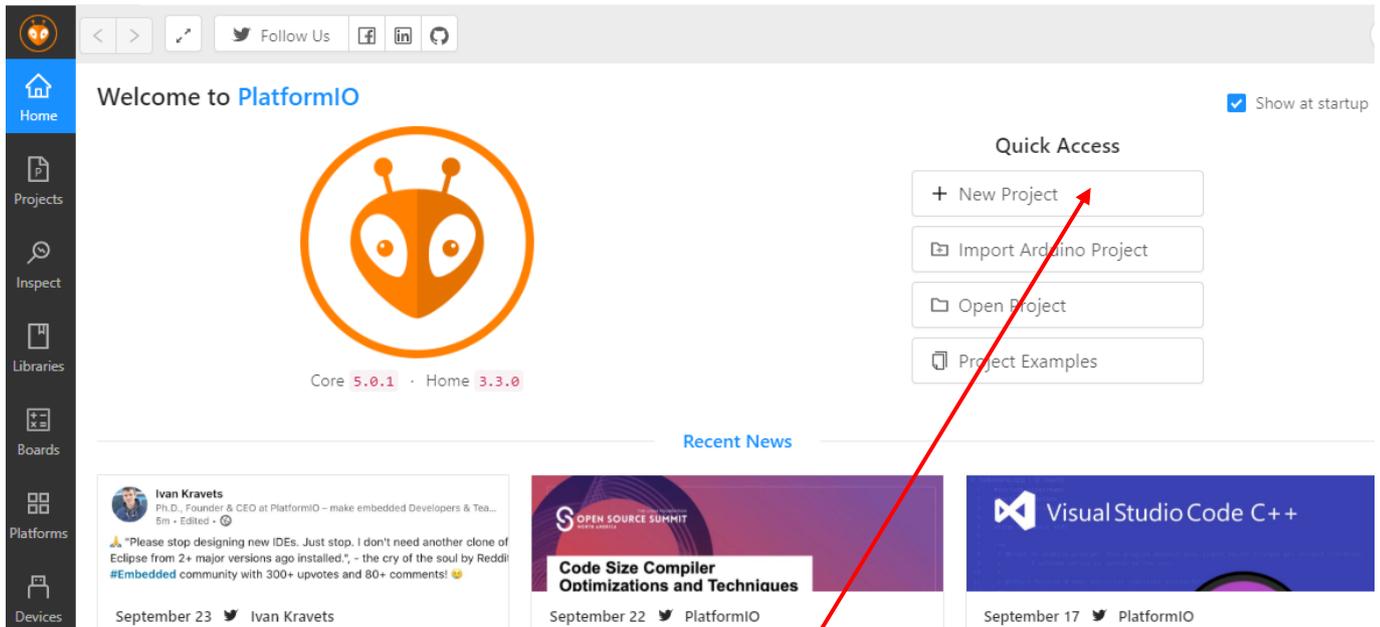
The installation is quite simple :

1. Download and install VSCode for your operating system, (<https://code.visualstudio.com/>),
2. Launch VSCode,
3. Step 1 open the VSCode extension pane Step 2 choose PlatformIO IDEE step 3 install the extension
4. Restart VSCode to complete the installation.
5. Welcome to PlatformIO...



### 3. Using PlatformIO

The window that opens should look like the figure below:



The grip is quite easy. To start a project, just click on "New Project".

In the window that opens, you have to fill in the following information:

- the name of your project,
- the card you wish to use
- the FDI model (varies according to the type of card, but Arduino is present for the majority of cards)

#### Project Wizard



This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name:

Board:

Framework:

Location:

Cancel

Finish

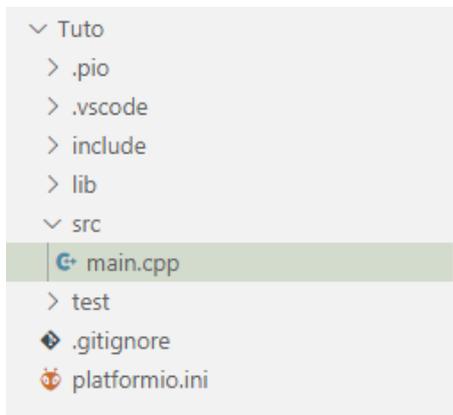
For this tutorial, I chose a Fire beetle ESP32 (DFRobot) board and the Arduino tool model.

It is also necessary to choose the location where this project will be stored, the default location is specified by placing the mouse cursor on the icon.

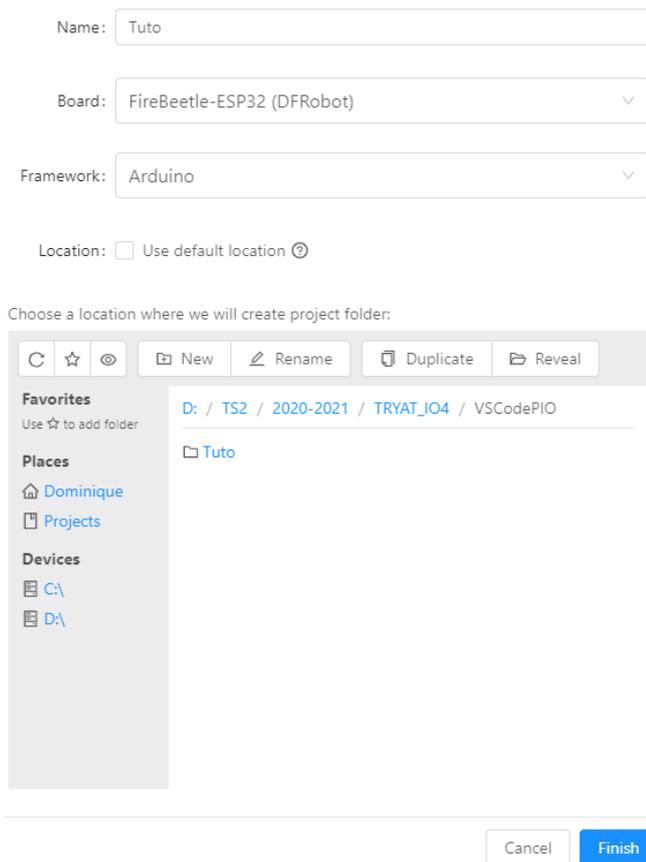


Once you have chosen the place to store the project, you just have to validate by clicking on Finish.

The software initiates the project in a few seconds and creates several folders and files :



- .pio is a folder that contains the binary files generated by the compiler. One sub-folder is created per target (development board).
- Include will contain the header files (xx.h) of the libraries useful for the project.
- lib is the folder in which the libraries (.c or .cpp) needed for the project will be installed. This makes it easier to manage versioning problems from one project to another. On the other hand, pay attention to the consumption of disk space.
- src. This folder contains the source code of your project. This is your working folder
- platformio.ini is the PIO configuration file.



The main.cpp file is as expected in "Arduino" format... The only notable difference is the presence of the header file: #include <Arduino.h>.

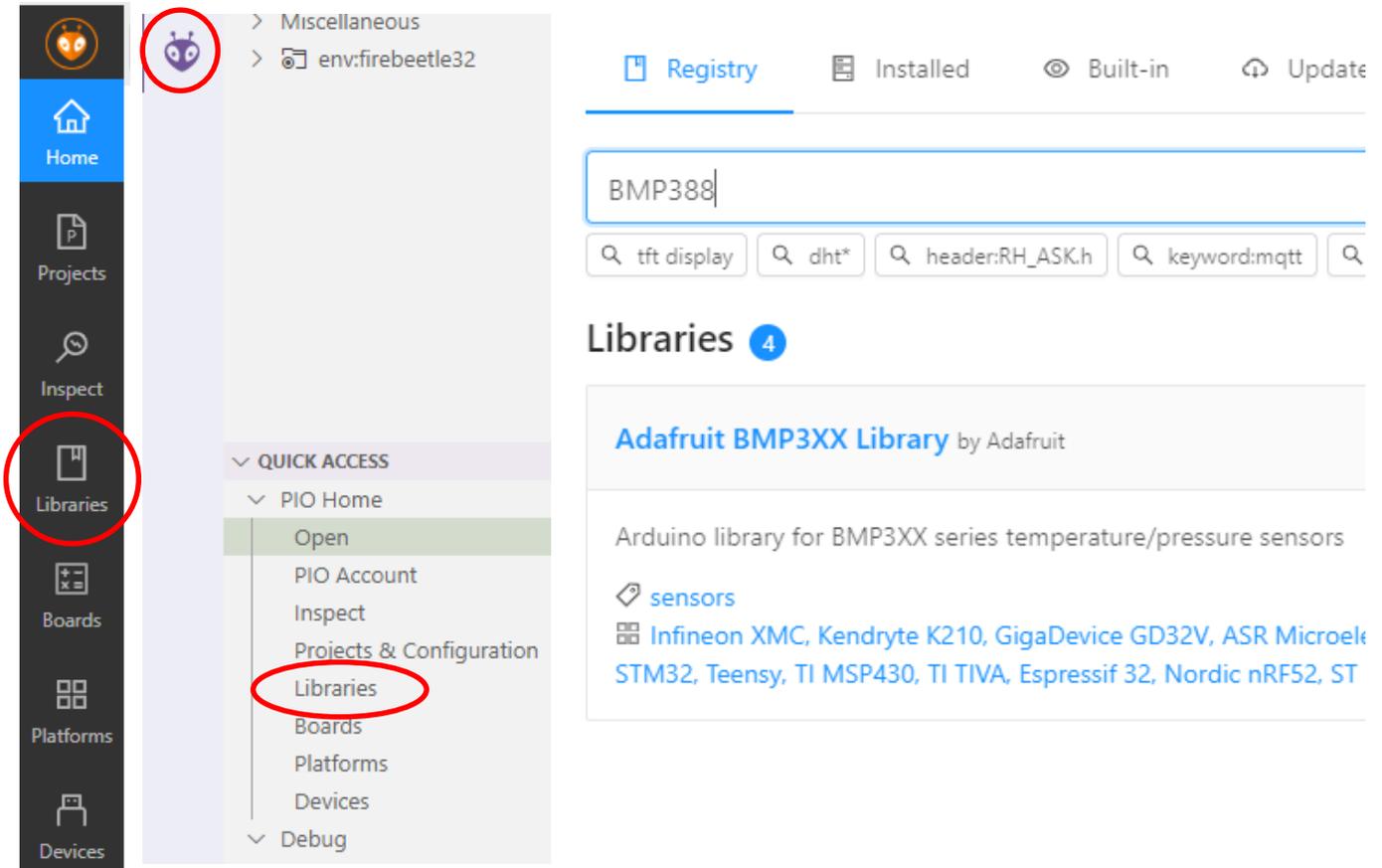
I propose to carry out a simple project which consists of implementing a BMP388 atmospheric pressure sensor and displaying the pressure, altitude, and temperature on the terminal (using the serial link via the USB cable).

By default, the bit rate of the serial link with the terminal is set to 115200 bits/s. This must therefore be specified to the terminal by adding the line :

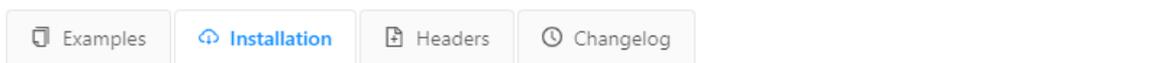
```
monitor_speed = 115200
```

In the file platformio.ini

We will need a library to manage the pressure sensor. We will therefore use the "libraries" function in the PIO menu (platformIO) accessible by clicking on the icon of the same name (either in the side banner or by clicking on the icon in the shape of an ant's head and then on "libraries").



In the search window, type the name of the component you are looking for here BMP388 and then type enter. At the time of writing this tutorial there are 4 libraries that match this keyword. We choose the Adafruit BMP3XX library. In the next window, there is a tab "Examples" which shows an example of use and in the tab



### Library Dependencies platformio.ini

The PlatformIO Registry is fully compatible with [Semantic Versioning](#) and its "version" scheme `<major>.<minor>.<patch>`. You can declare library dependencies in "platformio.ini" configuration file using `lib_deps` option.

```
; platformio.ini - project configuration file

[env:my_build_env]
platform = infineonxmc
framework = arduino
lib_deps =
  # RECOMMENDED
  # Accept new functionality in a backwards compatible manner and patches
  adafruit/Adafruit BMP3XX Library @ ^1.1.0
```

"Installation" we find the best way to install this library in the project in order to make it easily transportable from one workstation to another by specifying the dependencies in the platformio.ini file. This tells PIO to download the resources needed for the project.

The "platformio.ini" file must correspond to the figure opposite.

Let's go back to our "main.cpp" file and add the headers necessary to use the BMP3XX library.

A very interesting feature of VSCode is that it offers suggestions at the time of typing (autocomplete). For example, when you start typing Serial, then the software suggests the methods associated with this word (object), you click on begin and all you have to do is complete with the value of the desired bit rate. Here again, VSCode helps us by indicating the arguments necessary for the operation of the chosen function.

The main programme therefore becomes :

```
#include < Arduino.h>
#include < Wire.h>
#include < Adafruit_BMP3XX.h>.

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BMP3XX bmp388; // I2C388;

float temperature, altitude;
uint32_t pascals;

void setup() {
  Serial. begin(115200);
  Serial. println ( BMP388 test );
  if (! bmp388.begin()) {
    Serial. println ("Could not find a valid BMP3 sensor, check wiring!");
    while (1);
  }
  // Set up oversampling and filter initialization
  bmp388.setTemperatureOversampling(BMP3_OVERSAMPLING_8X);
  bmp388.setPressureOversampling(BMP3_OVERSAMPLING_4X);
  bmp388.setIIRFilterCoeff(BMP3_IIR_FILTER_COEFF_3);
  //bmp.setOutputDataRate(BMP3_ODR_50_HZ);
}

void loop() {
  if (! bmp388.performReading()) {
    Serial. println( Failed to perform reading (:);
    return;
  }
}
```

```
Tuto > platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:firebeetle32]
12 platform = espressif32
13 board = firebeetle32
14 framework = arduino
15
16 monitor_speed = 115200
17
18 lib_deps =
19 # RECOMMENDED
20 # Accept new functionality in a backwards compatible manner and patches
21 adafruit/Adafruit BMP3XX Library @ ^1.1.0
```

```
Tuto > src > main.cpp > ...
1 #include <Arduino.h>
2 #include <Wire.h>
3 #include <Adafruit_BMP3XX.h>
4
5 #define SEALEVELPRESSURE_HPA (1013.25)
6
7 Adafruit_BMP3XX bmp388; // I2C388;
8
9 void setup() {
10
11 }
12
13
14 void loop() {
15 // put your main code here, to run repeatedly:
16 }
```

```
void setup() {
  Serial.
  available
  availableForWrite
  baudRate
  begin
  clearWriteError
  end
  find
  findUntil
  flush
  setTimeout
  getWriteError
  operator bool
}
```

```

}
temperature = bmp388.readTemperature();
pascals = (uint32_t)bmp388.readPressure();
altitude = bmp388.readAltitude(SEALEVELPRESSURE_HPA);
Serial. print( Temp: );
Serial. print(temperature);
Serial. println( °C);
Serial. print( Pres: );
Serial. print(pascals); Serial. println("Pa");
Serial. print( Approx. Altitude = );
Serial. print(altitude);
Serial. println("m");
Serial. println();
delay(2000);
}

```

There are several methods to control the different actions to be carried out:  
Using the toolbar at the bottom of the PIO window

To go to the main PIO window

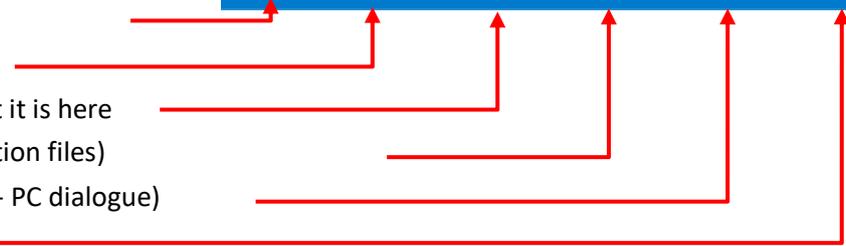
To compile the project it is here

To transfer the code to the target it is here

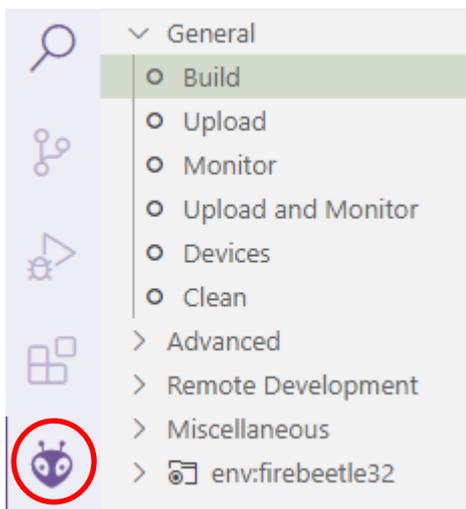
To clean up (destroy the compilation files)

To open the serial monitor (card - PC dialogue)

To open a terminal



Using the PIO menu



- Build : compile
- Upload: compiles and transfers the programme to the target
- Monitor: opens the serial monitor
- Upload and monitor: compile, transfer and open the serial monitor
- Devices: lists the connected cards
- Clean: Cleans up the project (deleting compilation files)

And of course the keyboard shortcuts :

- ctrl+alt+b: for the compilation of the project
- cmd-shift-d: to start debugging the project
- ctrl+alt+u: compiles and transfers the programme to the target
- ctrl+alt+s: opens the serial monitor

### The compilation

We launch a compilation to check the correct syntax of our code.

## The result of the compilation :

Processing firebeetle32 (platform: espressif32; board: firebeetle32; framework: arduino)

```

Verbose mode can be enabled via `--verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/espressif32/firebeetle32.html
PLATFORM: Espressif 32 (2.0.0) > FireBeetle-ESP32
HARDWARE: ESP32 240MHz, 320KB RAM, 4MB Flash
DEBUG: Current (esp-prog) External (esp-prog, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd, olimex-arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny, tumpa)
PACKAGES:
- framework-arduinoespressif32 3.10004.200129 (1.0.4)
- tool-esptoolpy 1.20600.0 (2.6.0)
- toolchain-xtensa32 2.50200.80 (5.2.0)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 40 compatible libraries
Scanning dependencies...
Dependency Graph
|-- <Adafruit BMP3XX Library> 1.1.0
| |-- <Wire> 1.0.1
| 1. 0
|-- Wire> 1.0.1
Building in release mode
Compiling .pio\build\firebeetle32\src\main.cpp.o
Linking .pio\build\firebeetle32\firmware.elf
Retrieving maximum program size .pio\build\firebeetle32\firmware.elf
Checking size .pio\build\firebeetle32\firmware.elf
Building .pio\build\firebeetle32\firmware.bin
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [ = ] 5.0% (used 16508 bytes from 327680 bytes)
Flash: [ == ] 18.4% (used 241334 bytes from 1310720 bytes)
esptool.py v2.6
===== [SUCCESS] Took 10.99 seconds =====

```

## Code transfer to the target

To finish this tutorial, all that remains to do is to implement the code on the Firebeetle card by clicking for example on the icon : 

```

Writing at 0x00010000... (12 %)
Writing at 0x00014000... (25 %)
Writing at 0x00018000... (37 %)
Writing at 0x0001c000... (50 %)
Writing at 0x00020000... (62 %)
Writing at 0x00024000... (75 %)
Writing at 0x00028000... (87 %)
Writing at 0x0002c000... (100 %)
Wrote 241456 bytes (125181 compressed) at 0x00010000 in 2.9 seconds (effective 663.3 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 11.50 seconds =====

```

Then open the serial monitor: 

And here is the result of this small example programme

```

> Executing task in folder Tuto: C:\Users\Dominique\.platformio\penv\Scripts\platformio.exe device monitor <

--- Available filters and text transformations: colorize, debug, default, direct, esp32_exception_decoder, hexlify, log2file, nocontrol, printable, send_on_enter, time
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on COM6 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Temp : 22.17 °C
Pres : 98203 Pa
Approx. Altitude = 263.13 m

```