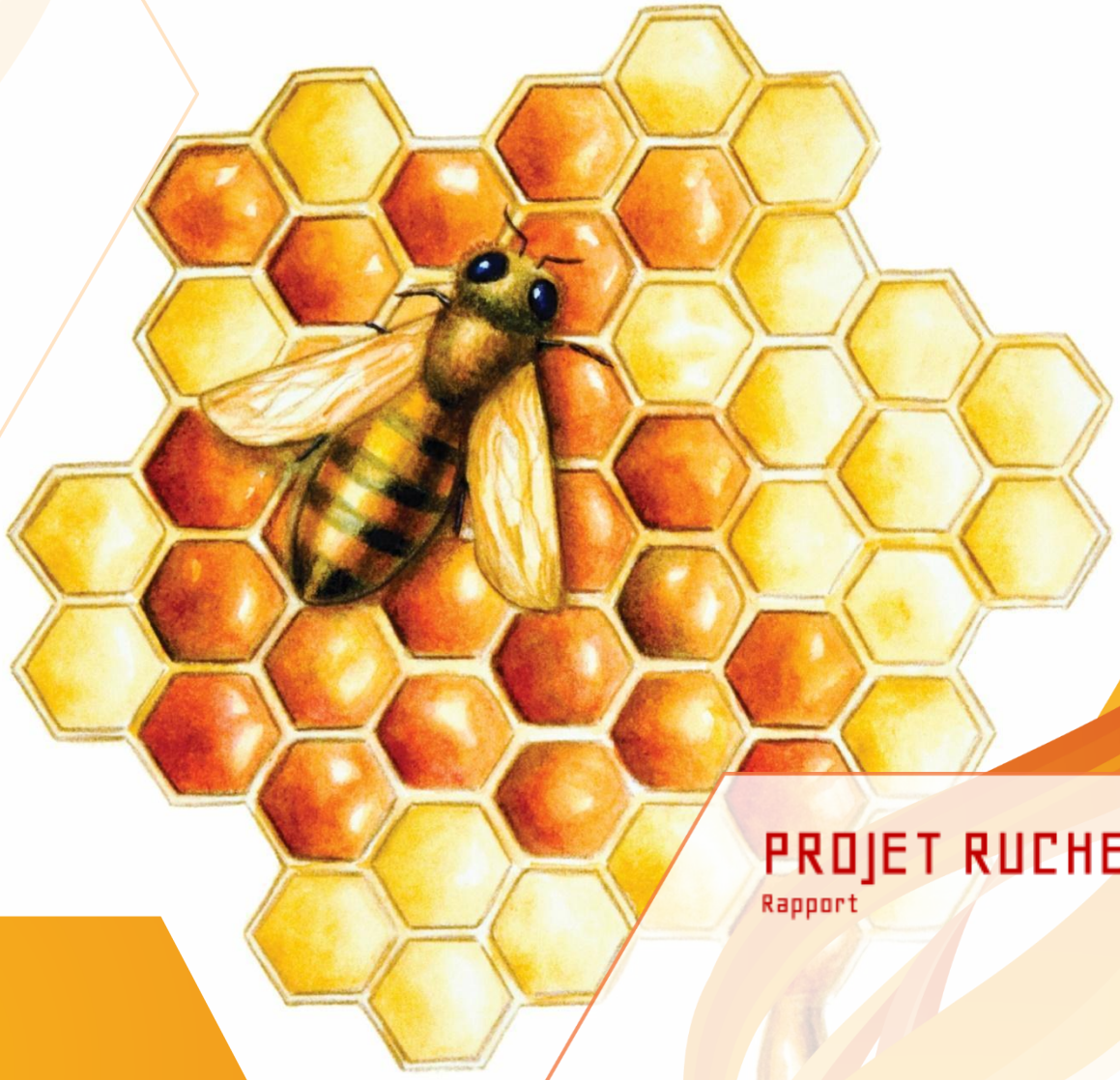


PROJET RUCHE

Rapport

Station Météorologique

BOUCHARD Guillaume
COMBA—ANTONETTI Lucile
HEISSLER Nicolas
IMBARD Nicolas
MAGNE Loris

Sommaire

Introduction

Dans le cadre de notre formation en alternance à Polytech Nice-Sophia, nous avons à réaliser un projet en électronique dont le but est de rendre "intelligente" la gestion des ruches disposées sur les toits de l'établissement.

L'objectif principal est d'optimiser la récolte du miel et d'améliorer la qualité et les conditions de vie des abeilles des 12 ruches.

Notre groupe a choisi de s'occuper de la station météorologique et plus spécifiquement du module météo.
Ce module doit être capable de centraliser un grand nombre d'informations et de les communiquer vers une Gateway qui va elle-même les afficher sur une page web.

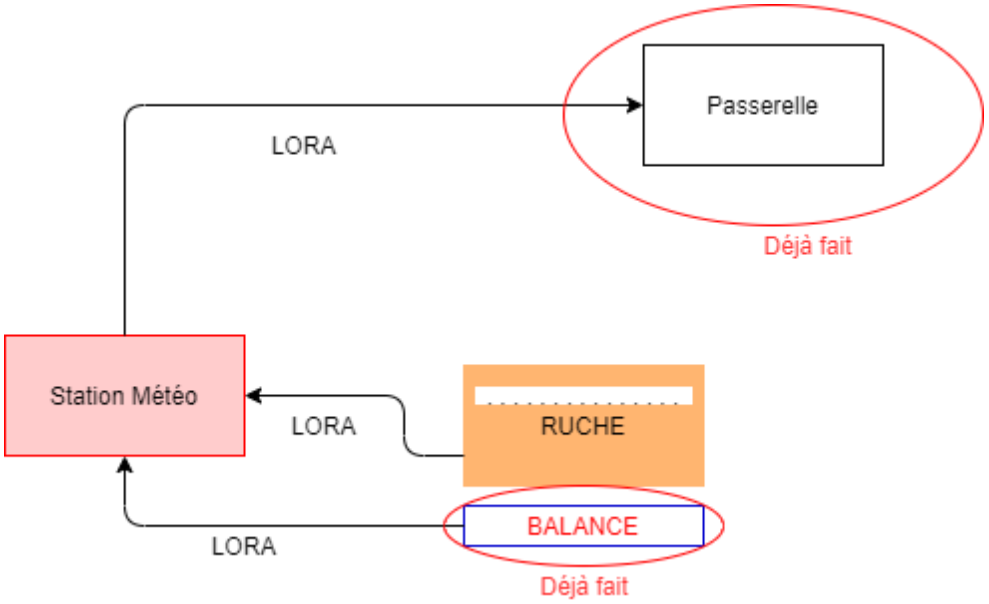
Le protocole de communication se fait exclusivement par du LoRa. Il s'agit d'un protocole de télécommunication radio permettant la communication à bas débit d'objets connectés.

Présentation du projet

Notre groupe est en charge de mettre en place la station météorologique qui s'occupe de :

- Relever la température, l'humidité, l'altitude, de la pluviométrie, la direction et la vitesse du vent
- Réceptionner les données transmises par les autres modules
- Transmettre la totalité des données à la base de données

Voici un schéma explicatif du fonctionnement des ruches connectées et du travail à réaliser :



Comme nous pouvons le constater, les ruches sont équipées de balances, permettant de peser la quantité de miel dans la ruche.

Les informations envoyées par ce module (et d'autres) seront transmises à une certaine fréquence, réceptionnées par le premier module LoRa de la station météorologique et retransmisent sur une autre fréquence radio vers la GateWay (passerelle).

Planning et organisation des tâches

Pour le projet, nous avons décidé de nous répartir certaines tâches afin de pouvoir avancer en parallèle et optimiser le temps de travail lors des séances de projet.

Voici les différentes tâches que nous avions à réaliser :

- Câblage des différents éléments passifs autour du FTDI
- Câblage et code arduino pour les capteurs de pression, altitude, humidité, température
- Câblage et code arduino du capteur de luminosité
- Câblage et code arduino du pluviomètre
- Câblage et code arduino de la girouette
- Câblage et code arduino de l'anémomètre
- Code arduino pour la communication entre les deux modules LoRa
- Code arduino pour la gestion du mode sommeil des éléments
- PCB : conception, design, fabrication et soudures

Voici le temps un planning montrant le temps que nous avons prévu pour réaliser ces tâches et le temps réellement passé :

Tâches ***Mois***	**9**	**10**	**11**	**12**	**1**	**2**	**3**	**4**	**5**
Câblage des différents éléments / FTDI									
Capteur de pression/altitude/humidité/température									
Capteur de luminosité									
Pluviomètre									
Girouette									
Anémomètre									
Module LoRa									
Gestion du mode sommeil									
PCB									

Nous pouvons constater que nous avons été un peu optimiste sur la prévision du temps à passer sur chaque tâche.
En effet, nous avons prit un peu de retard notamment sur :

- La girouette
- La communication LoRa
- La gestion du mode sommeil

Ces retards sont dus à des imprévus que nous avons rencontrés.

Partie 1 : Pistes non abouties

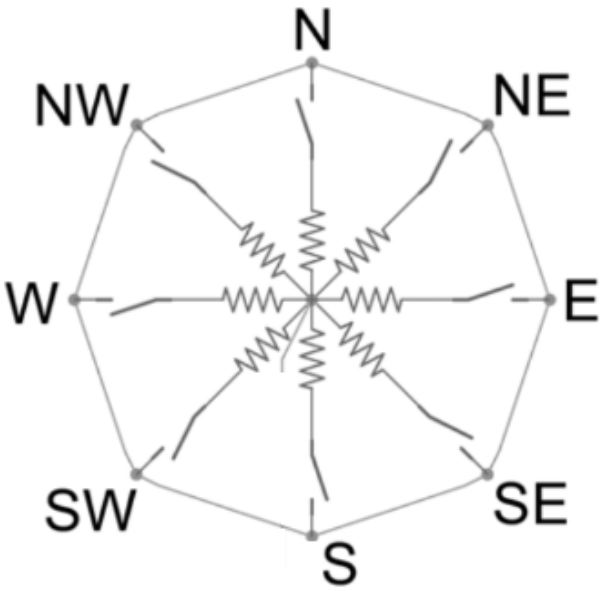
Girouette

Dans cette partie, nous détaillons les recherches et le fonctionnement de la Girouette. Ce capteur n'a malheureusement pas pu être intégré à notre système car plusieurs problèmes nous ont ralentis lors de l'implémentation du code arduino.

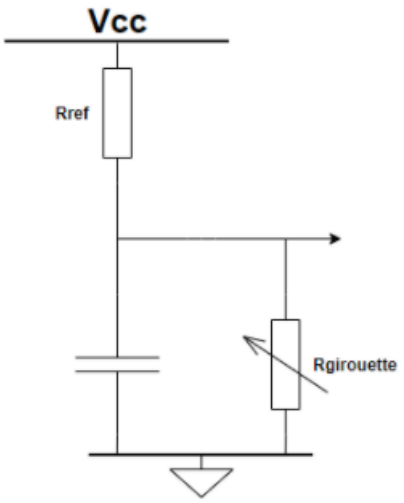
Fonctionnement

Ce capteur permet de connaître l'orientation du vent, il fonctionne de la manière suivante:

- Le vent oriente la pale dans une certaine direction
- L'emplacement fait de la pale va faire évoluer une résistance variable qui va donner une certaine tension mesurée
- Cette tension correspond à une certaine direction (indiqué dans la doc technique).



Voici le câblage associé :

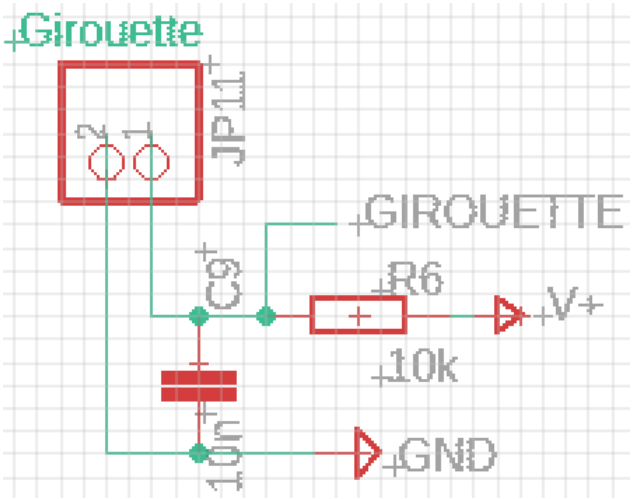


$$VRgirouette = \frac{Rgirouette}{Rgirouette + Rref} * Vcc$$

Pour calculer la tension de la résistance variable, nous avons utilisé un pont diviseur de tension en ajoutant une résistance de référence Rref.

Notre alimentation VCC est de 3,3V.

Schéma de câblage sur la carte en mode “Pull Up”



Détermination de la valeur de Rref

Nous devons maintenant calculer la valeur idéale pour Rref afin d'avoir les plus grands Deltas entre les différentes tensions ce qui apportera un gain de précision et une suppression du bruit.
Pour cela, pour différentes valeurs de Rref, nous avons calculé la tension au borne de la résistance variable pour les 16 variations d'angles remarquables puis nous en avons déduit les deltas suivant :

Delta pour R = 2,2k	Delta pour R = 3,3k	Delta pour R = 7k	Delta pour R = 10k
0,05	0,07	0,14	0,19
0,06	0,08	0,15	0,19
0,04	0,06	0,11	0,13
0,10	0,13	0,22	0,27
0,10	0,13	0,20	0,23
0,05	0,06	0,09	0,10
0,25	0,32	0,43	0,45
0,13	0,16	0,18	0,18
0,36	0,41	0,42	0,38
0,17	0,18	0,16	0,14
0,29	0,29	0,23	0,19
0,36	0,33	0,24	0,19
0,26	0,22	0,14	0,11
0,08	0,07	0,04	0,03
0,17	0,13	0,08	0,06

Nous remarquons qu'avec une résistance de 3,3k ohms, le Delta le plus petit est de 0,06V, c'est le plus élevé de tous les tests. Nous choisissons donc une Rref de 3,3k.

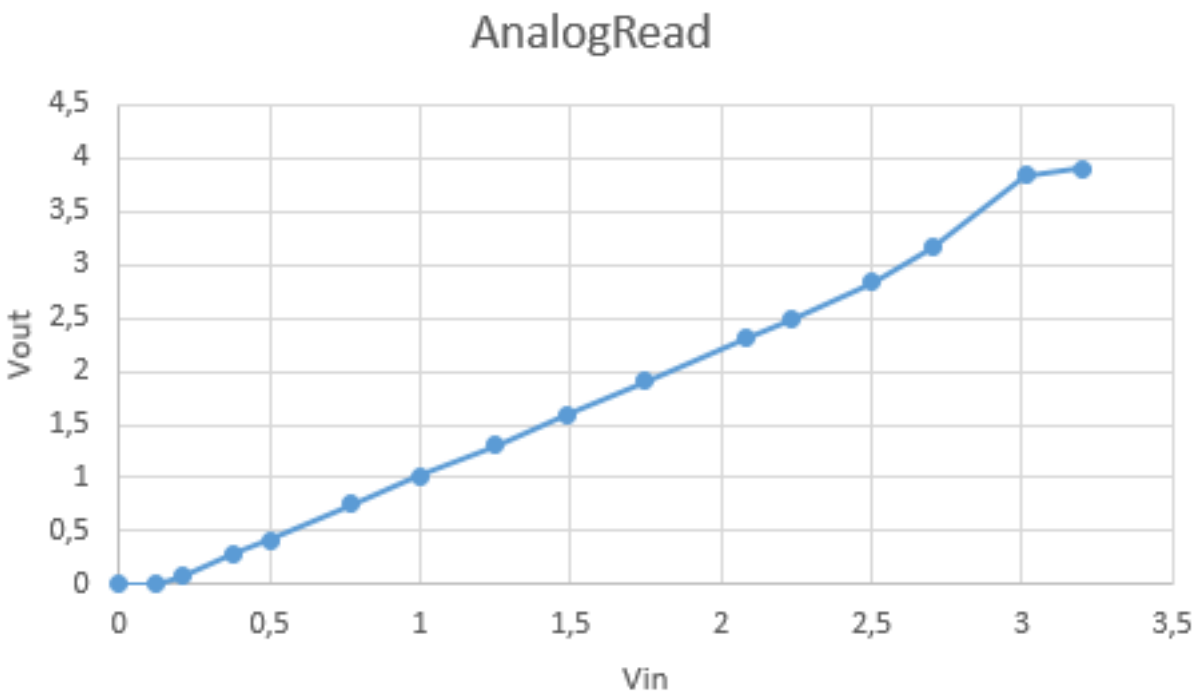
Tableau de correspondance entre tension mesurée et angle

Direction (Degrees)	Résistance (Ohms)	Voltage (V=3,3 R=3,3k)	Voltage (V=5 R=10k)
0	33000	3,00	3,84
22,5	6570	2,20	1,98
45	8200	2,35	2,25
67,5	891	0,70	0,41
90	1000	0,77	0,45
112,5	688	0,57	0,32
135	2200	1,32	0,90
157,5	1410	0,99	0,62
180	3900	1,79	1,40
202,5	3140	1,61	1,19
225	16000	2,74	3,08
247,5	14120	2,67	2,93
270	120000	3,21	4,62
292,5	42120	3,06	4,04
315	64900	3,14	4,33
337,5	21880	2,87	3,43

Comportement non linéaire des broches ESP32 ADC

La lecture d'une valeur analogique avec l'ESP 32 signifie que l'on peut mesurer différents niveaux de tension entre 0V et 3,3V. La tension mesurée est alors affectée à une valeur comprise entre 0 et 4095.

Nous avons utilisé un potentiomètre pour faire varier une tension qu'on mesurait à l'oscilloscope correspondant à VIN et nous avons en parallèle aussi récupéré la tension à l'aide de la fonction AnalogRead() correspondant à VOUT afin de les comparer. Nous somme supposés obtenir un rapport de 1, ce qui n'est pas le cas :



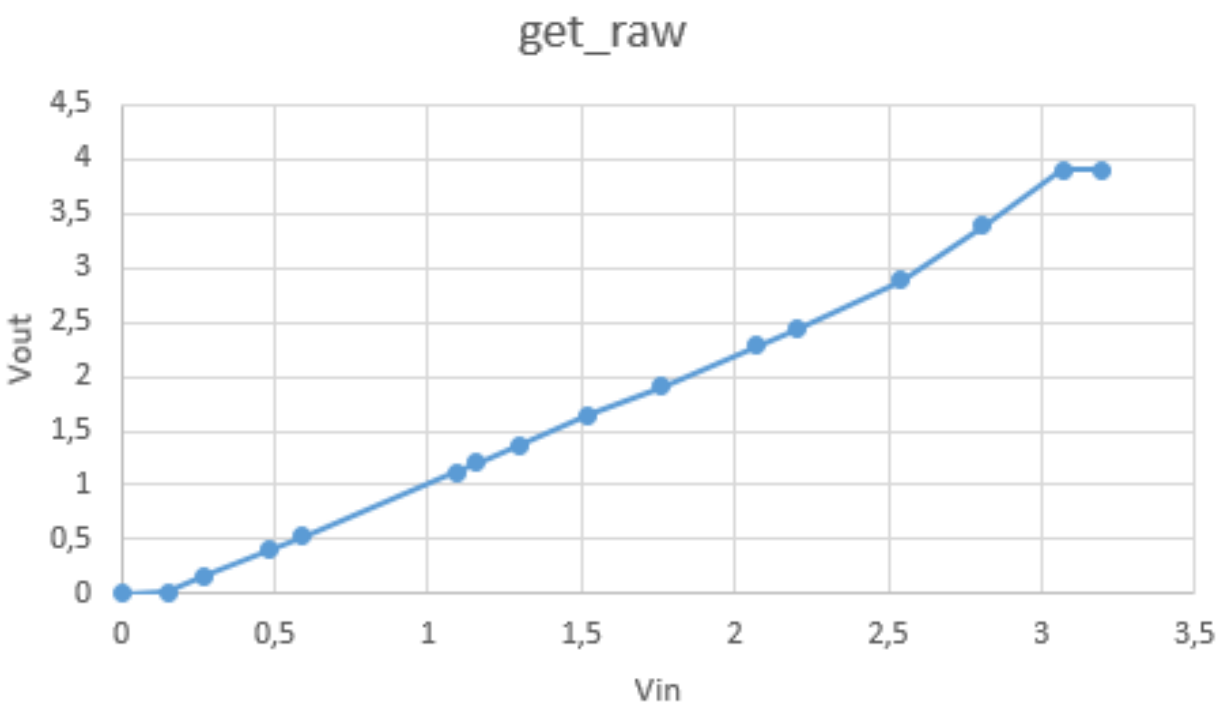
On observe un comportement non linéaire des broches ESP32 ADC.
En effet, l'ESP32 n'est pas capable de distinguer 3,3 V de 3,2 V, la même valeur est lue pour ces deux tensions : 3,8V. De même pour 0 V et 0,1 V, la valeur obtenue est la même : 0V.

Alternative : Utilisation de la fonction get_raw()

Nous avons donc réalisé la même expérience que précédemment en utilisant une autre fonction que analogRead().

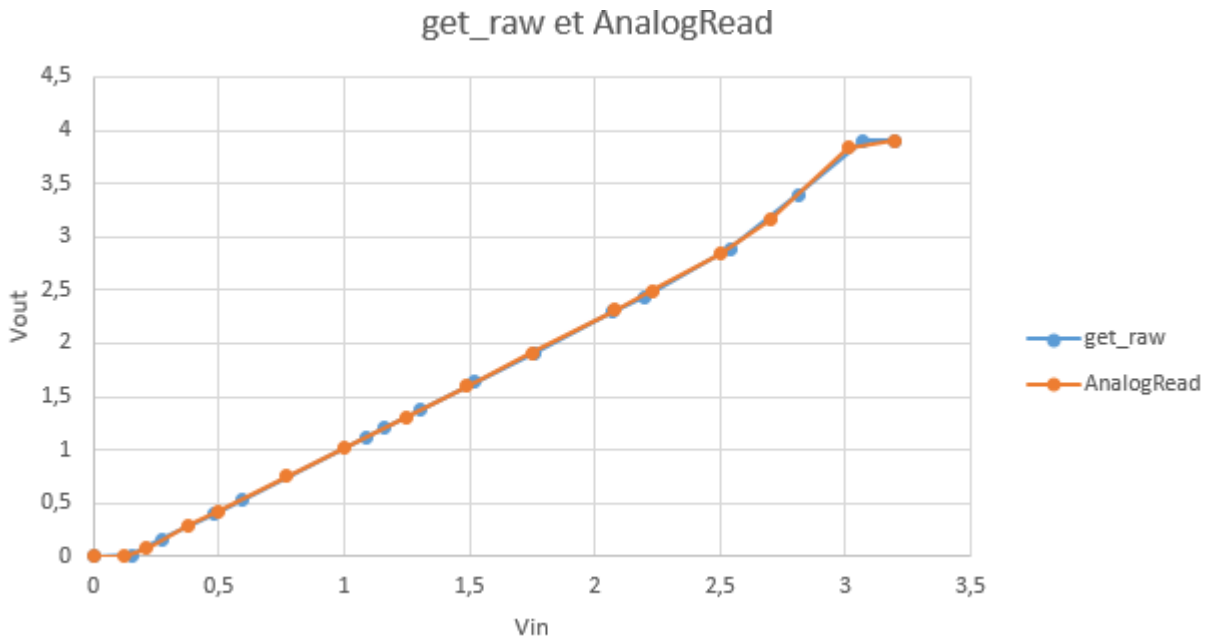
Nous avons utilisé le mode de lecture unique ADC, l'ADC 1 doit être configuré avant la lecture. Nous avons donc configuré la précision et l'atténuation souhaitée en appelant les fonctions *adc1_config_width ()* et *adc1_config_channel_atten ()*.

Ensuite, il est possible de lire le résultat de la conversion ADC avec *adc1_get_raw ()*.



Le tracé n'est pas non plus linéaire, nous devrions obtenir un coefficient directeur de 1, ce qui n'est pas le cas dans les extrémités du tracé.

Superposition des deux tracés :



Étant donné qu'il n'y a pas de différence notable entre l'utilisation de la fonction *get_raw* et *AnalogRead*, nous choisissons d'utiliser *AnalogRead* pour nous simplifier la tâche.

Autres piste / solution envisagée

Une dernière piste envisageable serait de modifier les seuils de tensions en redéfinissant Rref de sorte à travailler qu'avec la plage de valeurs linéaire.

L’objectif est donc d’ajouter un nouveau critère dans la définition de Rref, afin de ne pas atteindre des seuils de tension en dessous de 0,25V ou dépassant 3V.

Prise en compte de la variation de la tension d’alimentation

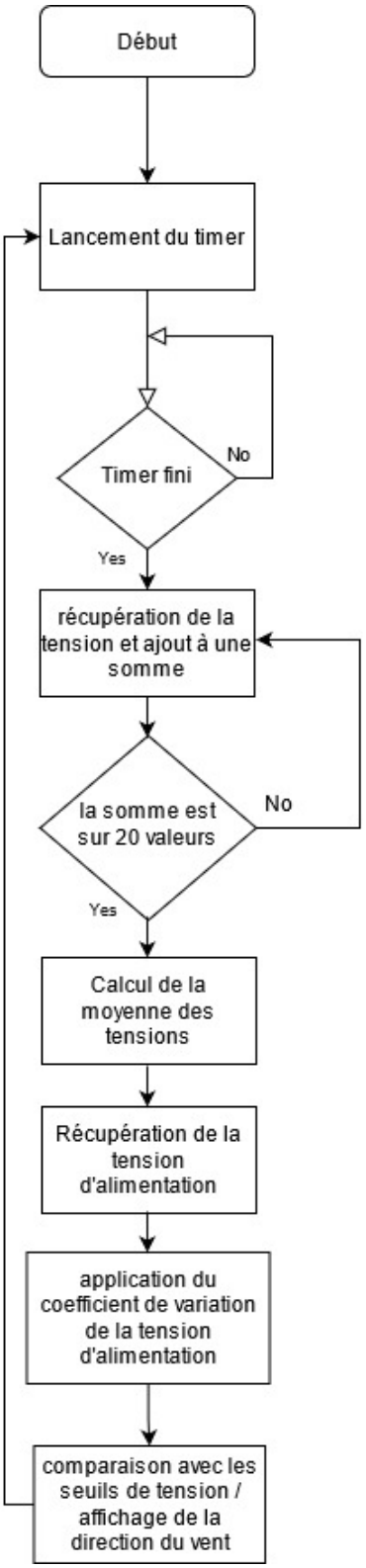
L'ESP32 étant alimenté par batterie, cette dernière peut être variable au cours du temps, selon le niveau de batterie. Or la tension mesurée est proportionnelle à la tension d’alimentation, il faut donc prendre en compte le coefficient de variation lors de la comparaison avec les seuils de tension.

Il est donc nécessaire de récupérer la tension d’alimentation qui peut se faire à l’aide d’un pont diviseur de tension (déjà présent dans le câblage), nous permettant d’en déduire le rapport avec la tension nominale d’alimentation de 3,3V et d’appliquer le coefficient aux tensions mesurées de la résistance interne de la girouette avant de les comparer aux tension de seuils.

Architecture du code

Pour un résultat plus précis, une moyenne sur 20 valeurs de tensions relevées est réalisée avant de faire la correspondance avec les seuils correspondants aux angles.

Finalement, Rref n’ayant pas été définitivement déterminé, le code de la girouette n’a pas été intégré au code final, ni dans le système.



Le code du fonctionnement du pluviomètre est en Annexe 7 : “Girouette”

Partie 2 : Rapport technique

Présentation des capteurs et matériels utilisés

Capteur BME

Ce capteur permet de mesurer :

- Pression (300 à 1100 hPa)
- Altitude
  Humidité (%)
- Température (-40 à 85°C)

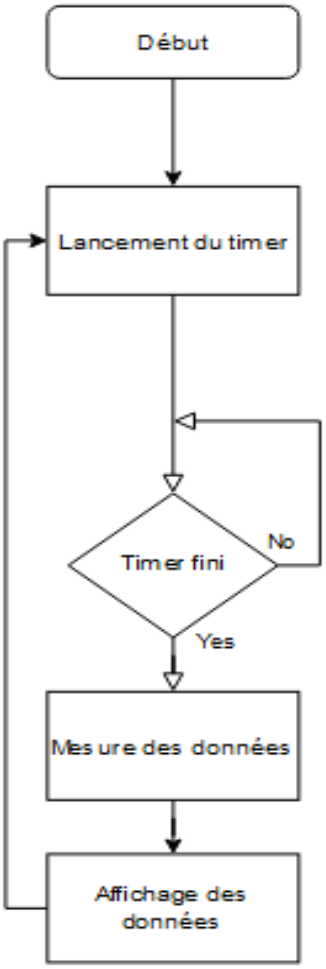
Son alimentation est :

- minimum 1.7v
- maximum 3.6v

Il possède plusieurs interfaces I²C :

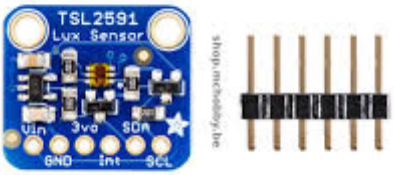
- SCK: serial clock (SCL)
- SDI: data (SDA)
- SDO: Slave address LSB

Logigramme du code Arduino



Le code complet du fonctionnement du capteur BME est en Annexe 2 : “Capteur BME et TSL”

Capteur TSL



Ce capteur permet de mesurer :

- La luminosité (0,1 à 40 000 lux)

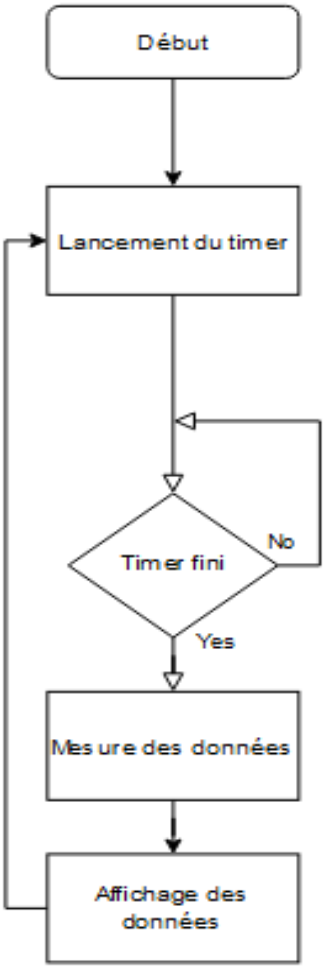
Son alimentation est :

- minimum 1.7v
- maximum 3.6v

Il possède plusieurs interfaces I²C :

- SCL: serial clock
- SDA: data

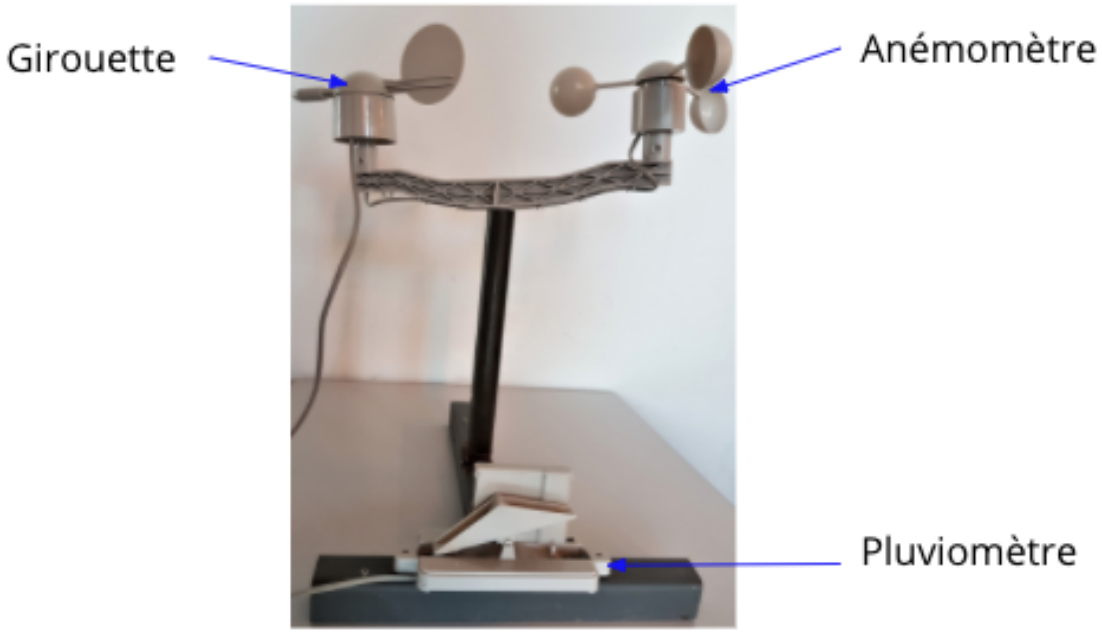
Logigramme du code Arduino



Le code complet du fonctionnement du capteur BME est en Annexe 2 : “Capteur BME et TSL”

Girouette

La Girouette permet de connaître l'orientation du vent.
Son fonctionnement est détaillée dans la "Partie 1 : Pistes non abouties"



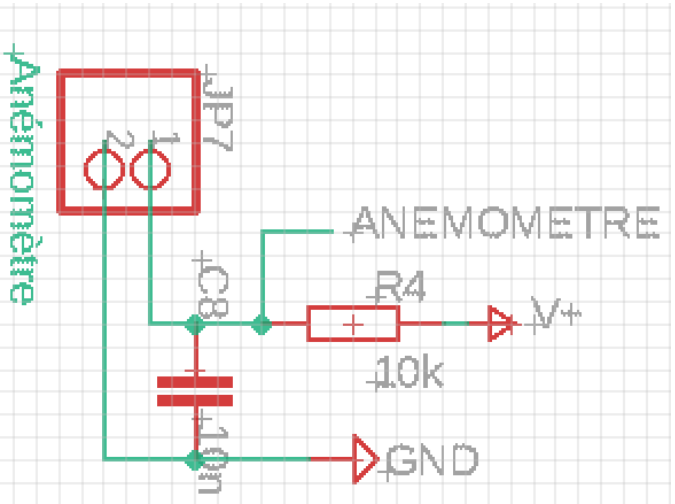
Anémomètre

L'Anémomètre permet de mesurer la vitesse du vent.

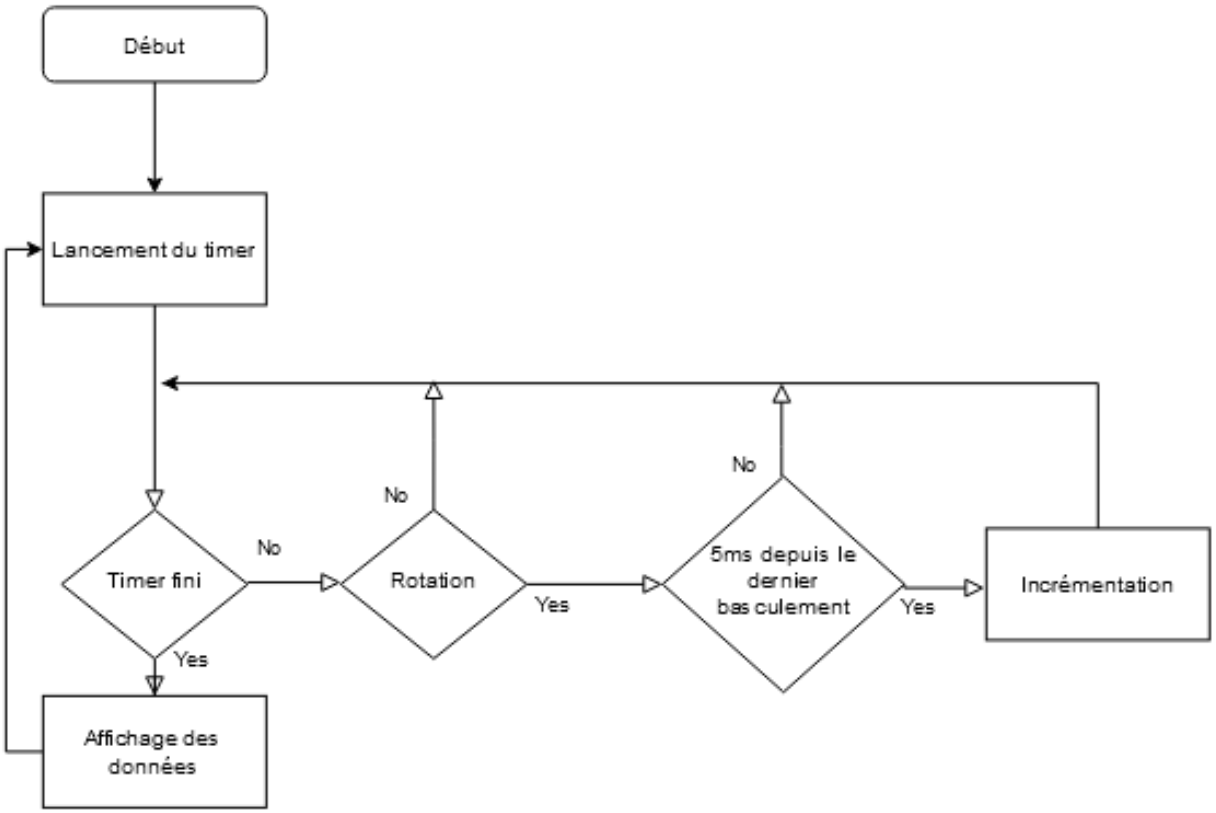
Fonctionnement

Ce capteur est composé de 3 pales, qui elles même contiennent des aimants. En tournant, chaque aimant va passer devant un ILS et va déclencher une interruption.
On peut alors en déduire la vitesse de rotation de l'anémomètre et donc calculer la vitesse du vent.

Schéma de câblage



Logigramme du code Arduino



Le code complet du fonctionnement de l'anémomètre est en Annexe 1 : "Test anémomètre"

Pluviomètre

Le pluviomètre permet de mesurer la quantité d'eau qu'il tombe (en m^3/L). Cette donnée est récupérée sur une entrée digitale (cf. schéma de câblage)

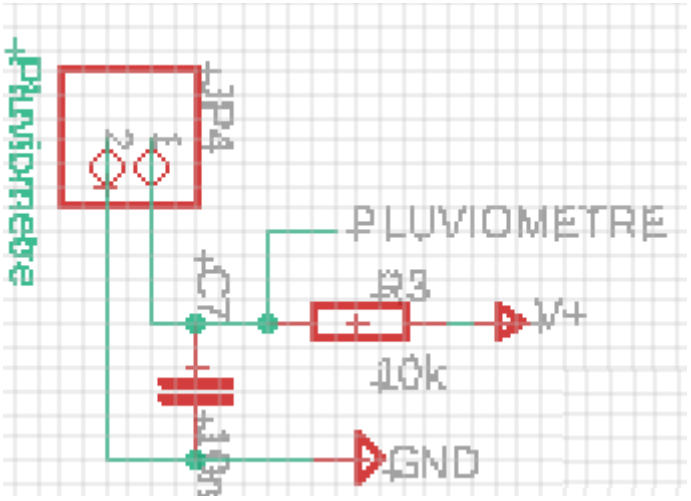
Fonctionnement

Lorsqu'il pleut, le réservoir d'eau du pluviomètre va se remplir petit à petit. Lorsqu'il sera plein, le poids va le faire basculer du côté opposé. Un aimant placé au milieu des deux réservoirs va alors passer devant un ILS, qui va déclencher une interruption.

On estime le calcul de la quantité d'eau qui tombe à cette formule :

*Quantité = 0.2794 * Nombre de basculements*

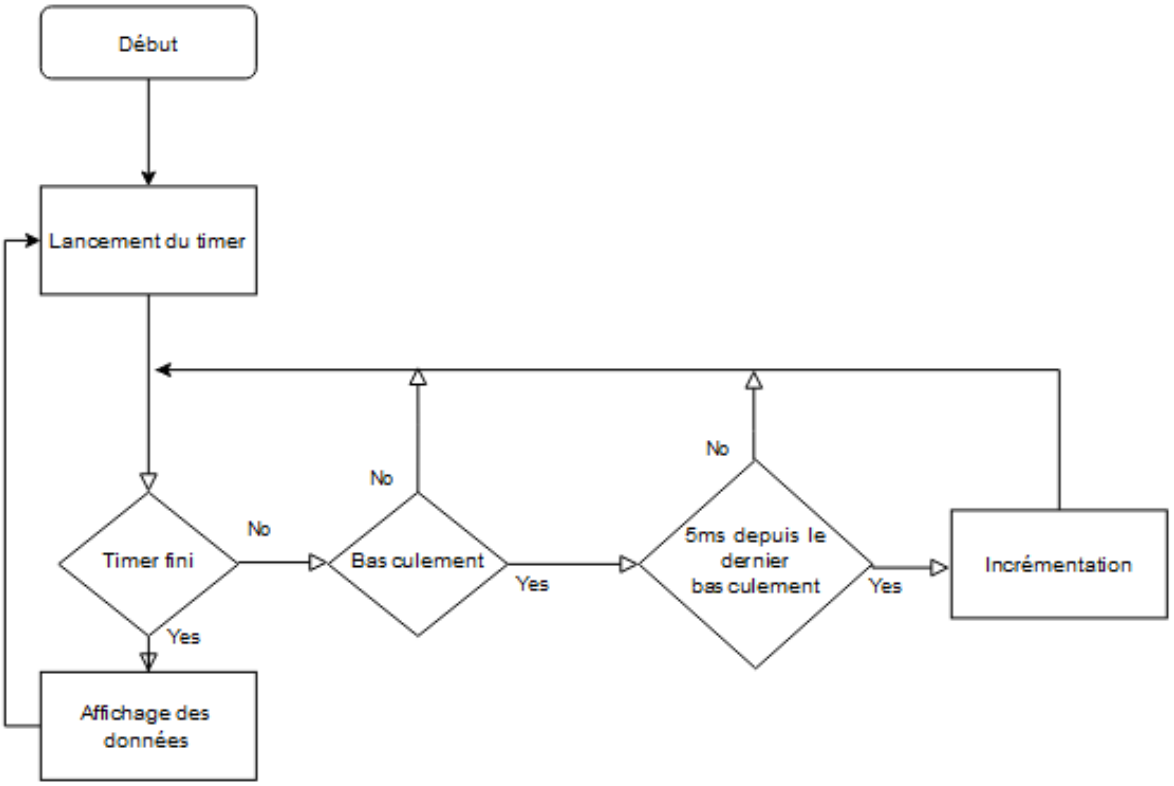
Voici son schéma électronique :



Explication des composants :

La résistance de 10K permet d'avoir un état logique haut en repos. C'est ce qu'on appelle résistance de tirage (pull-up). Le condensateur de 10nF permet d'éviter tous les rebonds des mesures parasites. C'est un condensateur de découplage. Cette valeur de 10 nF a été choisie à la suite de tests sur un oscilloscope.

Logigramme du code Arduino



Le code complet du fonctionnement du pluviomètre est en Annexe 3 : “Pluviomètre”

PCB

Schématique

Un projet ruche était déjà existant, nous avons dans un premier temps étudié les schémas et les composants qui avaient été choisis. Puis nous avons déterminé ce qui pouvait être réutilisé et ce qui était trop éloigné de nos besoins. Certains designs ont été repris en changeant les composants pour améliorer la fiabilité des mesures.

Avant de se plonger dans la schématique et le design du PCB nous avons testé et validé chaque composant sur breadboard. Une fois les composants fonctionnels, nous avons commencé la schématique sur le logiciel de “CAO EAGLE.”

Une fois les schéma et le fonctionnement de tous les composants validé. Il fallait s'assurer que nous pouvions relier les différents éléments au microcontrôleur. En effet, le micro est la pièce maîtresse de la carte. Cependant, il y a des différences entre les sorties du micro. C’est pourquoi nous avons dû minutieusement choisir quel pin devait être relié à quel capteur.

Ci-dessous le tableau récapitulatif des pin choisis et quel facteur nous a permis d’associer le capteur à la broche du micro.

Numéro de broche	Le nom du composant	Le besoin
TX	FTDI	Réponse du micro au FDTI
RX	FTDI	Réception du code dans le micro
IO 0	Bouton poussoir de prog	Entrée digitale
IO 2	Validation du téléversement	Entrée digitale
IO 4	LORA	Entrée digitale
IO 5	LORA	Entrée digitale
IO 12	Batterie	Entrée analogique
IO 13	Batterie	Entrée analogique
IO 14	LORA	Entrée digitale
IO 15	Batterie	Entrée digitale

IO 16	Capteur TSL	Entrée analogique + Pull up + Interruption
IO 17	Capteur TSL	Entrée analogique + Pull up + Interruption
IO 18	LORA	Entrée digitale
IO 19	LORA	SPI + Interruption
IO 20	///////	/////
IO 21	Capteur BME	Entrée analogique + Pull up + Interruption
IO 22	Capteur BME	Entrée digitale + Pull up + Interruption
IO 23	LORA	SPI + Interruption
IO 25	LED verte	Entrée analogique + Pull up
IO 26	LORA	Entrée digitale
IO 27	Anémomètre	Entrée analogique + Pull up + Interruption
IO 32	Pluviomètre	Entrée digitale + Pull up + Interruption
IO 33	LED rouge	Entrée analogique + Pull up
IO 34	Batterie	Entrée analogique
IO 35	Batterie	Entrée analogique
SENSOR_VP	Girouette	Entrée analogique et ADC1
SENSOR_VN	Capteur TSL	Entrée digitale

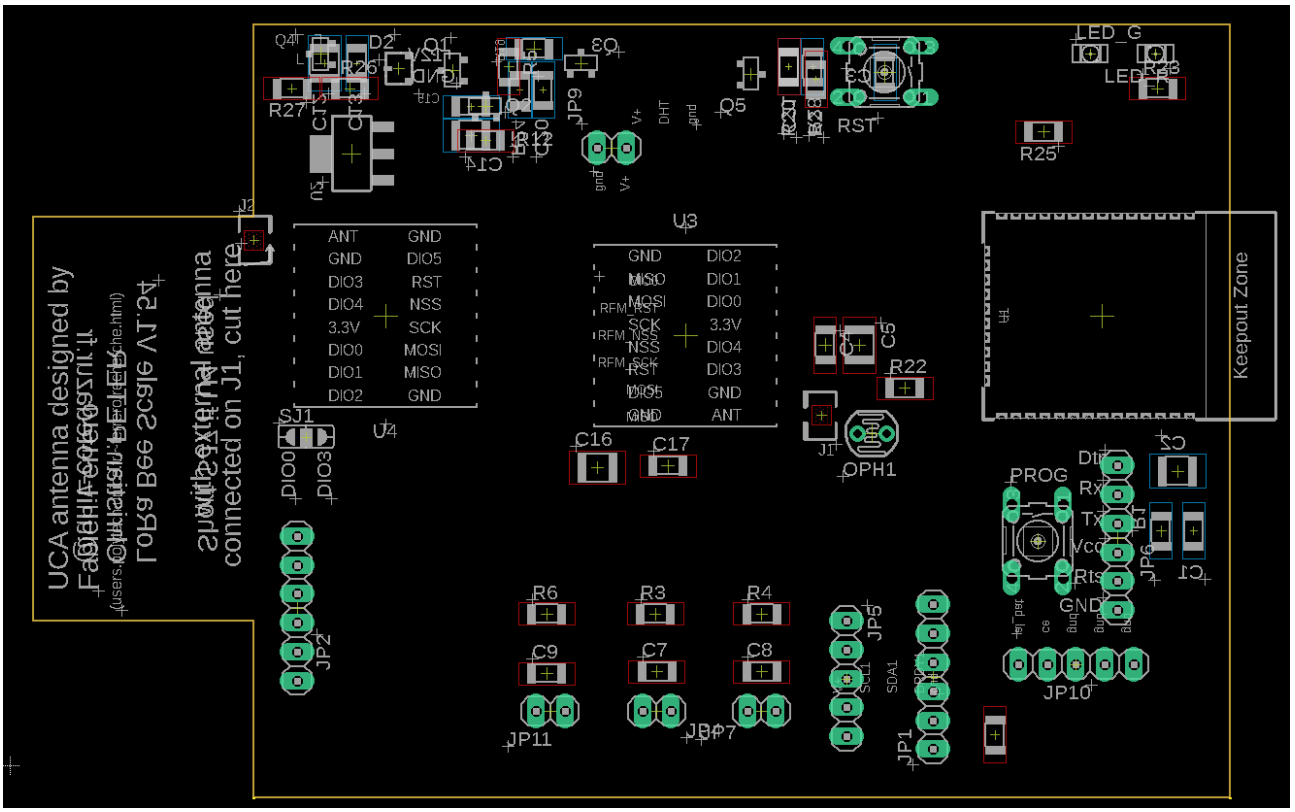
Dans ce tableau, il manque des IO qui sont des IO cachés du microcontrôleur, nous avons fait le choix de ne pas utiliser ces IO, car nous avions assez de place avec les broches visibles.

Placement des composants

Après avoir validé le fonctionnement de la partie schématique, nous nous sommes plongés dans le placement des composants sur la carte. Il faut savoir dans un premier temps que nous avons réalisé des recherches en interne pour savoir s'il n'y avait pas un design de carte à respecter. En effet, un design de carte existe et il faut l'utiliser car celui-ci possède une antenne intégrée ce qui va nous faire gagner du temps sur la modélisation de l'antenne de communication du module LORA.

Ainsi nous avons commencé avec un design de carte déjà existant, cependant, cette carte était trop petite pour que l'on puisse mettre tous les composants sur cette carte. En effet, ce modèle a été réalisé pour ne mettre qu'un seul module LORA, alors qu'on en possède deux.

Donc nous avons élargi la carte de 3 cm en largeur et en longueur pour garder un design homogène. Ce qui nous a permis de poser les composants sur cette carte sans trop de difficulté. Il est important de noter que les composants ne sont pas posés au hasard sur la carte. Nous avons respecté les connexions entre les composants qui existent sur la schématique. Ainsi, nous avons placé les composants passif tels que les résistances et les capacités au plus proche de leur module. Nous avons essayé de réduire la distance entre les modules de communication pour éviter toutes les perturbations sur les signaux de communication..



Routage de la carte

Pour effectuer le routage d'une carte électronique sur les deux couches, il faut respecter quelques règles.

1. Taille des pistes d'alimentation en 1.2mm
2. Taille des pistes de communication en 1mm
3. Espace entre les pistes de 1.2mm
4. Interdit de passer sous un composant CMS avec une piste
5. Faire des vias avec assez de cuivre (ajuster la taille du perçage avec la taille du via)
6. Étendre le plan de masse au maximum

Une fois ces règles bien établies au préalable, nous avons commencé le routage des composants.

Fabrication

En ce qui concerne la fabrication de la carte, déjà nous avons opté pour une fabrication interne dans les locaux de l'université pour gagner un maximum de temps. Nous sommes conscients de la différence de qualité entre des machines professionnelles et les machines de l'université. Ainsi nous avons reçu la carte avec tous les composants pour la fabrication.

Dans un premier temps nous avons fait un test de continuité sur la carte pour s'assurer que celle-ci était fonctionnelle. Ensuite nous avons supprimé le plan de masse qui venait s'insérer sous les composants CMS. En effet, lors de la fabrication nous n'avions pas sélectionné une exigence assez élevée sur la taille du plan de masse autorisé sous les composants CMS. Ainsi, si nous n'enlevons pas ce plan de masse nous avons un risque de court-circuit.

Une fois cette partie réalisée nous avons désoxydé la carte avec de la paille pour pouvoir souder tous les composants. Si cette action n'est pas réalisée, il est quasiment impossible de souder les composants, l'étain n'accroche pas sur la carte. Malheureusement, nous avons manqué de temps pour finaliser la soudure de tous ces composants.

Conclusion

Pour conclure ce projet, nous avons réparti les tâches efficacement entre nous en prenant compte nos compétences respectives. Cela nous a permis d'être au plus proche du planning prévisionnel malgré certaines surprises. En effet, certaines tâches qui nous paraissaient rapides à mettre en place se sont avérées plus complexes. Le code de la girouette ainsi que la mise en sommeil de l'ESP 32 en sont de bon exemples. Le retard pris ne nous a pas permis de finaliser le projet et de souder l'ensemble des composants sur la carte. Nous tirons donc de ce projet une leçon concernant la prévision et la prise en compte d'éventuels retards. Pour pallier ce problème et rester dans les temps, une marge de manœuvre plus importante aurait dû être affectée aux tâches auxquelles nous n'avons encore jamais été confrontés.

Annexes

Annexe n°1 : Code de test anémomètre

```
#include <Wire.h>
#include <SPI.h>
#include <driver/adc.h>

///--- Définitions des PINS
# define PinA 35

///--- VARIABLES
int nbInterrupt = 0;   //number of interruption & interruption per second
long measurementPeriod = 1000;           //sampling period in ms
unsigned long instant = 0;               //current time
unsigned long previous_loop = 0;         //time of the last loop
unsigned long previous_interrupt = 0;    //time of the last interruption
float Speed = 0, interruptSec = 0;       //wind speed

///--- FUNCTIONS
void launchAnemometerProcess();


// Initiate mod with pins management and communication speed
void setup()
{


  pinMode(PinA, INPUT);
  Serial.begin(9600);
  while (!Serial);
  Serial.println();

  //at each lap: call of the function to increment the number of rotation
  attachInterrupt(digitalPinToInterrupt(PinA), launchAnemometerProcess, FALLING);

}

void loop()
{
  instant = millis();
  //Serial.println(nbRotation);

  if ((instant - previous_loop) >= measurementPeriod)     //if the measurement period has passed
  {
    //A wind speed of 2.4 km/h causes the switch to close once per second
    interruptSec = nbInterrupt / (measurementPeriod * 0.001);
    // measurementPeriod * 0.001 is used to define interrupt/sec depending on the periode
    Speed = interruptSec*2.4;

    //Display of the speed
    //Serial.println(nbInterrupt);
    //Serial.println(interruptSec);
    Serial.print("wind speed: ");
    Serial.print(Speed);
    Serial.println(" km/h");

    previous_loop = instant;     //resets the time and counter for the next measurement period
    nbInterrupt = 0;
   }

}

void launchAnemometerProcess()
{
  // if it is the first interruption or the next one after 5 ms (debouncing)
  if(previous_interrupt == 0 || instant-previous_interrupt >= 5)
  {
    previous_interrupt = instant;
    nbInterrupt = nbInterrupt + 1;  //increment of the number of interruptions
  }
}
```

Annexe n°2 : Code de tests capteurs TSL et BME

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <Adafruit_TSL2591.h>

///--- Définitions des PINS

//-- BME PINS ASSIGNEMENTS

//--  BME280 wiring
// connect SCL to I2C Clock (PIN 36 on ESP32)
// connect SDA to I2C Data  (PIN 33 on ESP32)
// connect Vcc to 3.3V
// connect GROUND to GND

//--  TSL2591 wiring
// connect SCL to I2C Clock (PIN 27 on ESP32)
// connect SDA to I2C Data  (PIN 28 on ESP32)
// connect Vin to 3.3V
// connect GROUND to GND

///--- CONST
#define SEALEVELPRESSURE_HPA (1022)

///--- FUNCTIONS
void launchTslProcess();
void launchBmeProcess();

Adafruit_BME280 bme;  // I2C
Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591); // pass in a number for the sensor identifier (for your use later)

// Initiate mod with pins management and communication speed
void setup()
{
  //pinMode(DI0, OUTPUT);
  Serial.begin(9600);
  while (!Serial);
  Serial.println();

  ///--- BME
  Serial.println("Beginning BME280 module settings...");

  // Try to find BME280 sensor
  while (!bme.begin(0x77, &Wire))
  {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
  }
  Serial.println("BME280 is founded and set");

  ///--- TSL
  Serial.println("Beginning TSL2591 module settings...");

  // Try to find Adafruit TSL2591
  while (!tsl.begin())
  {
  Serial.println(F("No sensor found ... check your wiring?"));
  }
  // Configures the gain and integration time for the TSL2591
  tsl.setGain(TSL2591_GAIN_MED);  //Medium gain (LOW for bright light and HIGH for dim light)
  tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS); //Medium intergration time  (100MS for bright light to 600MS for dim light)
  Serial.println(F("TSL2591 sensor founded and set"));
}

void loop()
{
  Serial.print("\n");

  //-- LAUNCHING BME SENSOR PROCESS
  launchBmeProcess();
  //--- LAUNCHING TSL SENSOR PROCESS
  launchTslProcess();

  delay(2000);
}
```

```
///--- FUNCTIONS
void launchBmeProcess()
{
  bme.takeForcedMeasurement();

  //-- DISPLAY PRESSURE
  Serial.print("Pressure : ");
  Serial.print(bme.readPressure());
  Serial.println(" Pa");
  Serial.print("Pressure at level 0 : ");
  Serial.print(bme.readPressure()/0.987);
  Serial.println(" Pa");

  //-- DISPLAY HUMIDITY
  Serial.print("Humidity : ");
  Serial.print(bme.readHumidity());
  Serial.println(" %");

  //-- DISPLAY TEMPERATURE
  Serial.print("Temperature : ");
  Serial.print(bme.readTemperature());
  Serial.println(" °C");

  //-- DISPLAY ALTITUDE
  Serial.print("Height = ");
  Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");
}

void launchTslProcess()
{
  uint32_t lum = tsl.getFullLuminosity();
  uint16_t ir, full;
  ir = lum >> 16;
  full = lum & 0xFFFF;
  Serial.print(F("IR: ")); Serial.print(ir);  Serial.print(F("  "));
  Serial.print(F("Full: ")); Serial.print(full); Serial.print(F("  "));
  Serial.print(F("Visible: ")); Serial.print(full - ir); Serial.print(F("  "));
  Serial.print(F("Lux: ")); Serial.println(tsl.calculateLux(full, ir), 6);
}
```

Annexe n°3 : Code de tests pluviomètre

```
#include <Wire.h>
#include <SPI.h>
#include <driver/adc.h>

///--- Définitions des PINS
# define PinP 34

///--- VARIABLES
unsigned long nbRocker = 0;             //number of rockers
float qtyRain = 0;              //quantity of rain (mm)
long measurementPeriod = 5000;          //sampling period in ms
unsigned long instant = 0;              //current time
unsigned long previous_loop = 0;        //time of the last loop
unsigned long previous_interrupt = 0;   //time of the last interruption

///--- FUNCTIONS
void launchRainGaugeProcess();


// Initiate mod with pins management and communication speed
void setup()
{
  pinMode(PinP, INPUT_PULLDOWN);
  Serial.begin(9600);
  Serial.println("");
  while (!Serial);

  //at each tilting: call of the function to increment the number of rockers
  attachInterrupt(digitalPinToInterrupt(PinP), launchRainGaugeProcess, FALLING);

}

void loop()
{
  instant = millis();
  if ((instant - previous_loop) >= measurementPeriod)     //if the measurement period has passed
  {
    // Calculate the quantity of rain on the period
    qtyRain = 0.2794*nbRocker;
    // Display of the number of rockers & the rain quantity
    Serial.print("Number of rockers : ");
    Serial.println(nbRocker);
    Serial.print("Quantity of rain : ");
    Serial.print(qtyRain);
    Serial.println(" L/m²");

    previous_loop = instant;    //resets the time and counter for the next measurement period
    nbRocker = 0;
   }

}

void launchRainGaugeProcess()
{
  // if it is the first interruption or the next one after 5 ms (debouncing)
  if(previous_interrupt == 0 || instant-previous_interrupt >= 5)
  {
    previous_interrupt = instant;
    nbRocker += 1;          //increment of the number of rockers
  }
}
```

Annexe n°4 : Code de test deepsleep mode avec pluviomètre

```
#include <Wire.h>
#include <SPI.h>
#include <driver/adc.h>

///--- Définitions des PINS
#define PinP 32 //GPIO 32
#define UP   1
#define DOWN 0

///--- Variables
time_t now;
char strftime_buf[64];

RTC_DATA_ATTR int nbRocker = 0;         //number of rockers (saved on RTC memory)
RTC_DATA_ATTR time_t lastTimeDataSent;
RTC_DATA_ATTR unsigned long timePassed;
float qtyRain = 0;                      //quantity of rain (mm)
esp_sleep_source_t wakeUpReason;        //reason of the waking up

void setup()
{
  pinMode(PinP, INPUT_PULLDOWN);
  Serial.begin(9600);
  Serial.println("");
  while (!Serial);

  //Enable waking up with external interruption
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_34, DOWN);
}

void loop()
{
  time(&now);

  wakeUpReason = esp_sleep_get_wakeup_cause();
  switch(wakeUpReason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 :
      nbRocker += 1;
      Serial.printf("Waked up by external interruption : %d \n",nbRocker);
      Serial.printf("Current time : %d s \n", now);
      timePassed = difftime(now, lastTimeDataSent) * 1000000; //timePassed need to be in us
      Serial.printf("Data passed since last send : %l us \n",timePassed);
      esp_sleep_enable_timer_wakeup(10000000ULL - timePassed);
    break;

    case ESP_SLEEP_WAKEUP_TIMER :
      Serial.println("Waked up by timer, display datas :");
      // Calculate the quantity of rain on the period
      qtyRain = 0.2794*nbRocker;
      // Display of the number of rockers & the rain quantity
      Serial.printf("Number of rockers : %d\n", nbRocker);
      Serial.printf("Quantity of rain : %f L/m²\n", qtyRain);
      nbRocker = 0;
      timePassed = 0;
      time(&lastTimeDataSent);
      esp_sleep_enable_timer_wakeup(10000000ULL - timePassed);
    break;

    default :
      Serial.printf("Wake up was not caused by deepsleep : %d\n", wakeUpReason);
      esp_sleep_enable_timer_wakeup(10000000ULL - timePassed);
    break;
  }
  Serial.println("=== goto sleep ======");

  esp_deep_sleep_start();

}
```

Annexe n°5 : Code de tests d'envoi de message LoRa (2 connectés)

```
#include <LoRa.h>
#include <Wire.h>
#include <SPI.h>

///--- CONST
#define LoraSenderBand 866E6
#define measure 13

LoRaClass LoRa0;
LoRaClass LoRa1;

// Lora Sender
#define sender_rst    14
#define sender_dio0   26
#define sender_ss     5
#define sender_cs     5

// Lora Sender2
#define sender2_rst  14
#define sender2_dio0 27
#define sender2_ss   4
#define sender2_cs   4

// SCK, MISO, MOSI is the same pin for both LoRa devices
#define SCK           18
#define MISO          19
#define MOSI          27

///--- VARIABLES
int messageNumber = 0;
```

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

  //Define both chip select as output
  pinMode (sender_cs,    OUTPUT);
  pinMode (sender2_cs,   OUTPUT);
  pinMode (measure,      OUTPUT);

  SPI.begin();

  digitalWrite(sender_cs, HIGH);
  digitalWrite(sender2_cs, HIGH);

  //Initiate sender LoRa device
  Serial.println("LoRa sender");
  digitalWrite(measure, HIGH);
  digitalWrite(sender_cs, LOW);
  digitalWrite(sender2_cs, HIGH);
  delay(1000);
  LoRa0.setPins(sender_ss, sender_rst, sender_dio0);
  LoRa0.setSPIFrequency(8E6);
  while (!LoRa0.begin(LoraSenderBand))
  {
    Serial.println("Starting sender LoRa failed !");
    delay(1000);
  }
  Serial.println("Starting sender LoRa successfully !");
  digitalWrite(sender_cs, HIGH);
  delay(1000);

  //Initiate sender 2 LoRa device
  Serial.println("LoRa1 sender");
  digitalWrite(sender2_cs, LOW);
  LoRa1.setPins(sender2_ss, sender2_rst, sender2_dio0);
  LoRa1.setSPIFrequency(8E6);
  while (!LoRa1.begin(LoraSenderBand))
  {
    Serial.println("Starting sender 2 LoRa failed !");
    delay(500);
  }
  Serial.println("Starting sender 2 LoRa successfully !");
  digitalWrite(sender2_cs, HIGH);
  delay(1000);
}
```

```
void loop()
{
  //Sender 1 packet code
  Serial.print("Sending packet from sender 1 : ");
  Serial.println(messageNumber);

  //digitalWrite(sender_cs, LOW);
  LoRa0.begin(LoraSenderBand);
  delay(500);
  LoRa0.beginPacket();
  LoRa0.print("Message from first sender number : ");
  LoRa0.print(messageNumber);
  LoRa0.endPacket(true);
  //digitalWrite(sender_cs, HIGH);
  delay(1000);

  //Sender 2 packet code
  Serial.print("Sending packet from sender 2 : ");
  Serial.println(messageNumber);

  //digitalWrite(sender2_cs, LOW);
  LoRa1.begin(LoraSenderBand);
  delay(500);
  LoRa1.beginPacket();
  digitalWrite(sender_cs, HIGH);
  digitalWrite(sender2_cs, LOW);
  LoRa1.print("Message from sender 2 number : ");
  LoRa1.print(messageNumber);
  LoRa1.endPacket(true);
  //digitalWrite(sender2_cs, HIGH);

  messageNumber++;
  delay(3000);
}
```

Annexe n°6 : Code complet

```
///--- LIBRARY INCLUDE
#include <Wire.h>
#include <SPI.h>
#include <config.h>
#include <LoRa.h>
#include <driver/adc.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2591.h>
#include <Adafruit_BME280.h>

///--- PINS ASSIGNEMENT
#define PinP 32 //GPIO 32
#define PinA 35

//---  RFM95W (LORA SENDER)
#define sender_rst    14
#define sender_dio0   26
#define sender_ss     5
#define sender_cs     5

//--- RFM95W (LORA RECEIVER)
#define receiver_rst  14
#define receiver_dio0 27
#define receiver_ss   4
#define receiver_cs   4

// SCK, MISO, MOSI is the same pin for both LoRa devices
#define SCK     18  // GPIO5  -- SX1278's SCK
#define MISO    19  // GPIO19 -- SX1278's MISnO
#define MOSI    27  // GPIO27 -- SX1278's MOSI

//---  TSL2591 wiring
// connect SCL to I2C Clock (PIN 36 on ESP32)
// connect SDA to I2C Data  (PIN 33 on ESP32)
// connect Vin to 3.3V
// connect GROUND to GND

//---  BME280 wiring
// connect SCL to I2C Clock (PIN 36 on ESP32)
// connect SDA to I2C Data  (PIN 33 on ESP32)
// connect VCC to 3.3V
// connect GROUND to GND

///--- CONSTANT
#define UP   1
#define DOWN 0
#define measure 13
#define SEALEVELPRESSURE_HPA (1013.25)
#define SENDERBAND  868E6 // Frequency to send (868 Mhz)
#define RECEIVERBAND  868E6 // Frequency to send (868 Mhz)
```

```
///--- VARIABLE
time_t now;
char strftime_buf[64];

Adafruit_BME280 bme;  // I2C
// pass in a number for the sensor identifier (for your use later)
Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591);
LoRaClass LoRa0;
LoRaClass LoRa1;

unsigned long actualTime;                   // Actual time (in ms since the programme was launched
unsigned long TimeOfTheLastSend = 0;        // Time where the last message was sent
int           TimeBetweenTwoSend = 900000;  // Time between two sending (900000 ms -> 15 minutes)
unsigned long counter = 0;                  // Number of message sent
String        messageReceived = "";         // Message that the balance module will send to us
String        messageToSend = "";           // Message to send to the gateway

int nbInterrupt = 0;   //number of interruption & interruption per second
long measurementPeriod = 1000;           //sampling period in ms
unsigned long instant = 0;               //current time
unsigned long previous_loop = 0;         //time of the last loop
unsigned long previous_interrupt = 0;    //time of the last interruption
float Speed = 0, interruptSec = 0;       //wind speed

RTC_DATA_ATTR int nbRocker = 0;         //number of rockers (saved on RTC memory)
RTC_DATA_ATTR time_t lastTimeDataSent;
RTC_DATA_ATTR unsigned long timePassed;
float qtyRain = 0;                      //quantity of rain (mm)

esp_sleep_source_t wakeUpReason;        //reason of the waking up

///--- FUNCTIONS
void launchBmeProcess();
void launchTslProcess();
void launchRainProcess();
void launchAnemometerProcess();
```

```
void setup()
{
  //-- SETTING SERIAL CONNECTION
  Serial.begin(9600);
  Serial.println("");
  while (!Serial);
  Serial.println("Beginning all module settings...");

  //-- SETTING INTERRUPT PINS
  pinMode (PinP, INPUT_PULLDOWN);
  pinMode(PinA, INPUT);
  pinMode (sender_cs,   OUTPUT);
  pinMode (receiver_cs,  OUTPUT);
  pinMode (measure,     OUTPUT);

  SPI.begin();
  digitalWrite(measure, HIGH);

  //-- SETTING BME280 SENSOR
  while (!bme.begin(0x77, &Wire))
  {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
  }
  Serial.println("BME280 is founded and set");

  //-- SETTING TSL2591 SENSOR
  while (!tsl.begin())
  {
  Serial.println(F("No sensor found ... check your wiring?"));
  }
  Serial.println(F("TSL2591 sensor founded and set"));
  //-- CONFIGURING THE TSL2591
  tsl.setGain(TSL2591_GAIN_MED);  //Medium gain (LOW for bright light and HIGH for dim light)
  tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS); //Medium intergration time  (100MS for bright light to 600MS for dim light)

  //-- SETTING LORA SENDER MODULE
  Serial.println("LoRa sender");
  digitalWrite(sender_cs, LOW);
  digitalWrite(receiver_cs, HIGH);
  delay(1000);
  LoRa0.setPins(sender_ss, sender_rst, sender_dio0);
  LoRa0.setSPIFrequency(8E6);
  while (!LoRa0.begin(SENDERBAND))
  {
    Serial.println("Starting sender LoRa failed !");
    delay(1000);
  }
  Serial.println("Starting sender LoRa successfully !");
  digitalWrite(sender_cs, HIGH);
  delay(1000);

  //-- SETTING LORA RECEIVER MODULE
  Serial.println("LoRa1 receiver");
  digitalWrite(receiver_cs, LOW);
  LoRa1.setPins(receiver_ss, receiver_rst, receiver_dio0);
  LoRa1.setSPIFrequency(8E6);
  while (!LoRa1.begin(RECEIVERBAND))
  {
    Serial.println("Starting receiver LoRa failed !");
    delay(500);
  }
  Serial.println("Starting receiver LoRa successfully !");
  digitalWrite(receiver_cs, HIGH);
  delay(1000);

  //Enable waking up with external interruption
  esp_sleep_enable_ext0_wakeup(GPIO_NUM_34, DOWN);
  esp_sleep_enable_ext1_wakeup(GPIO_NUM_32, DOWN);
}
```

```
void loop()
{
  time(&now);

  wakeUpReason = esp_sleep_get_wakeup_cause();
  switch(wakeUpReason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 :
      nbRocker += 1;
      Serial.printf("Waked up by external interruption : %d \n",nbRocker);
      Serial.printf("Current time : %d s \n", now);
      timePassed = difftime(now, lastTimeDataSent) * 1000000; //timePassed need to be in us
      Serial.printf("Data passed since last send : %l us \n",timePassed);
      esp_sleep_enable_timer_wakeup(10000000ULL - timePassed);
    break;

    case ESP_SLEEP_WAKEUP_EXT1 :
      // if it is the first interruption or the next one after 5 ms (debouncing)
      if(previous_interrupt == 0 || instant-previous_interrupt >= 5)
      {
        previous_interrupt = instant;
        nbInterrupt = nbInterrupt + 1;  //increment of the number of interruptions
      }
    break;

    case ESP_SLEEP_WAKEUP_TIMER :
      Serial.println("Waked up by timer, display datas then send them to the gateway :");

      //-- LAUNCH ALL SENSOR PROCESS
      launchBmeProcess();
      launchTslProcess();
      launchRainProcess();
      launchAnemometerProcess();

      //-- SEND THE DATA
      launchLoRaProcess();

      timePassed = 0;
      time(&lastTimeDataSent);
      esp_sleep_enable_timer_wakeup(10000000ULL - timePassed);
    break;

    default :
      Serial.printf("Wake up was not caused by deepsleep : %d\n", wakeUpReason);
      esp_sleep_enable_timer_wakeup(10000000ULL - timePassed);
    break;
  }
  Serial.println("=== goto sleep ======");

  esp_deep_sleep_start();
}
```

```
void launchBmeProcess()
{
  bme.takeForcedMeasurement();

  //-- DISPLAY PRESSURE
  Serial.print("Pressure : ");
  Serial.print(bme.readPressure());
  Serial.println(" hPa");

  //-- DISPLAY HUMIDITY
  Serial.print("Humidity : ");
  Serial.println(bme.readHumidity());
  Serial.println(" %");

  //-- DISPLAY TEMPERATURE
  Serial.print("Temperature : ");
  Serial.println(bme.readTemperature());
  Serial.println(" °C");

  //-- DISPLAY ALTITUDE
  Serial.print("Altitude = ");
  Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");
}

void launchTslProcess()
{
  uint32_t lum = tsl.getFullLuminosity();
  uint16_t ir, full;
  ir = lum >> 16;
  full = lum & 0xFFFF;
  Serial.print(F("IR: ")); Serial.print(ir);  Serial.print(F("  "));
  Serial.print(F("Full: ")); Serial.print(full); Serial.print(F("  "));
  Serial.print(F("Visible: ")); Serial.print(full - ir); Serial.print(F("  "));
  Serial.print(F("Lux: ")); Serial.println(tsl.calculateLux(full, ir), 6);
}
```

```
void launchRainProcess()
{
  //-- Calculate the quantity of rain on the period
  qtyRain = 0.2794*nbRocker;

  //-- Display of the number of rockers & the rain quantity
  Serial.printf("Number of rockers : %d\n", nbRocker);
  Serial.printf("Quantity of rain : %f L/m²\n", qtyRain);
  nbRocker = 0;
}

void launchLoRaProcess()
{
  //-- SENDING DATA TO THE GATEWAY
  Serial.print("Sending packet to the gateway : ");

  LoRa0.begin(LoraSenderBand);
  delay(500);
  LoRa0.beginPacket();
  LoRa0.print("W");
  LoRa0.print("MAC::ADDRESSE");
  //-- EACH DATA NEED TO BE SEND ON ONE BYTE
  //-- TWO BYTE FOR THE TEMP
  LoRa0.endPacket(true);
  delay(1000);
}

void launchAnemometerProcess()
{
  //A wind speed of 2.4 km/h causes the switch to close once per second
  interruptSec = nbInterrupt / (measurementPeriod * 0.001);
  // measurementPeriod * 0.001 is used to define interrupt/sec depending on the periode
  Speed = interruptSec*2.4;

  //Display of the speed
  //Serial.println(nbInterrupt);
  //Serial.println(interruptSec);
  Serial.print("wind speed: ");
  Serial.print(Speed);
  Serial.println(" km/h");

  previous_loop = instant;     //resets the time and counter for the next measurement period
  nbInterrupt = 0;
}
```

Annexe n°7 : Code incomplet de la girouette

girouette

```
#include <Wire.h>
#include <SPI.h>
#include <driver/adc.h>

///--- Définitions des PINS
# define PinG 34// ->32-39

///--- VARIABLES
long measurementPeriod = 1000;           //sampling period in ms
unsigned long instant = 0;               //current time
unsigned long previous_loop = 0;         //time of the last loop
float tension = 0, tension_alim = 0;       //tensions
float coef_ajustement = 0;
long sum;

///--- FUNCTIONS

// Initiate mod with pins management and communication speed

void setup()
{

  pinMode(PinG, INPUT);
  Serial.begin(9600);
  while (!Serial);
  Serial.println();

}



void loop()
{
  instant = millis();

  if ((instant - previous_loop) >= measurementPeriod)     //if the measurement period has passed
  {
    //adc_power_on();
    //adc1_config_width(ADC_WIDTH_12Bit);
    //adc1_config_channel_atten(ADC1_CHANNEL_6,ADC_ATTEN_DB_11);

    sum = 0;
    for (int i = 0; i<20; i++) {
      //tension = adc1_get_raw(ADC1_CHANNEL_6);
      tension = analogRead(PinG);
      sum += tension;
    }

    tension = sum / 20;

    Serial.println(float((tension * 3.9)/4095));


    previous_loop = instant;     //resets the time and counter for the next measurement period
   }

}
```