P.E. Station environnementale

Raphaël COMPS, Adrien KOMAROFF-KOURLOFF, Étienne GOZILLON,
Savinien THIEBAUT, Axel GUICHAOUA, Louis MAINIX-CHIRIO[1]

[1]École des Mines de Saint-Étienne

Mars 2023

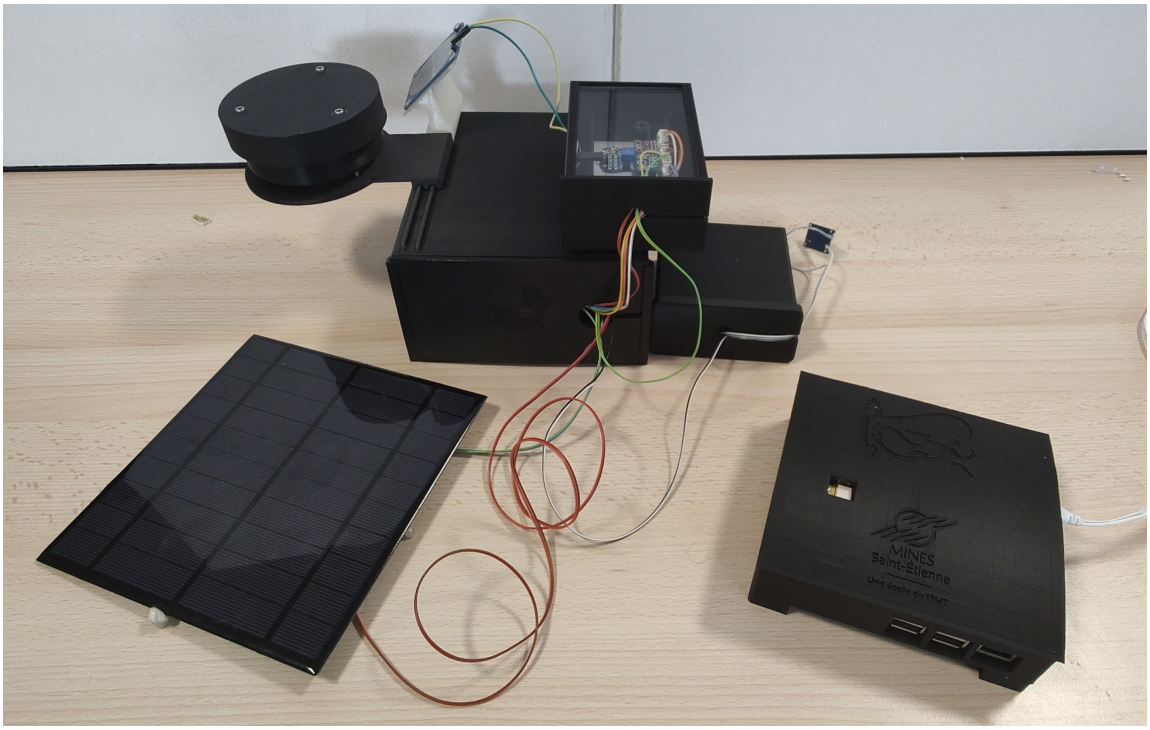


Table des matières

1 Introduction

Ce rapport technique présente le projet de fin d'étude "*Réalisation d'une station environnementale*". L'objectif est de réaliser une station comportant une multitude de capteurs environnementaux ainsi que le point d'accès Wi-Fi associé, l'ensemble pouvant être déplacé et déployé en fonction du besoin. Les données sont affichées sur un tableau de bord accessible en ligne. Cette station doit avoir une portée éducative et être facilement extensible. L'utilisateur final est autant celui qui accède au tableau de bord pour consulter les relevés que l'élève étudiant le projet pour se former. En tant qu'objet IoT, la partie embarquée de mesure placée en extérieur devra être autonome énergiquement grâce à un panneau solaire et une batterie. Les performances énergétiques atteintes fixent le délai d'acquisition des données, dans l'idéal toutes les 5 minutes. Le choix des différents capteurs a été réalisé par le client. Le projet s'inspire d'une première itération réalisée par le client présentée en figure 1.

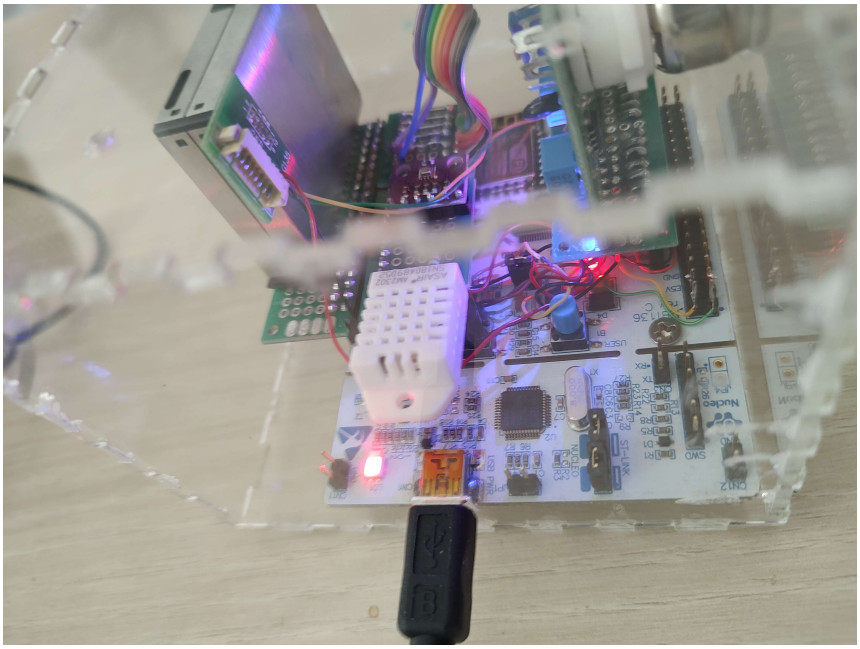


FIGURE 1 – Première itération réalisé par le client

Le client a également fait le choix du microcontrôleur STM32F767 qui possède de nombreux *GPIOs* pour la modularité. Conformément à sa demande, le microcontrôleur est programmé grâce au *framework* STM32duino.

Ce rapport fait également office de passation de savoirs à quiconque souhaite par la suite modifier le système ou se former à son utilisation. Il rend compte du travail effectué et des choix faits en considérant, le cas échéant, les contraintes conflictuelles d'autonomie en énergie, de conception d'un système modulaire (possibilité de rajouter des éléments avec les *GPIOs* libres), accessible au plus grand nombre, et de compatibilité matérielle ou logicielle des composants. Le code du projet et les ressources additionnelles sont sur github.

2 Vue d'ensemble

Le but de cette partie est de donner une idée de la forme et du fonctionnement général de la station météorologique, ainsi que des interactions et dépendances entre les différents blocs fonctionnels. Les principaux aspects y sont présentés et sont expliqués dans le détail dans la suite de ce document.

La figure 2 présente le synoptique global du système. Ce dernier est composé de deux parties distinctes. La première est la partie embarquée, dont le rôle est la mesure des données environnementales et météorologiques et leur remontée vers le tableau de bord par protocole Wi-Fi. Son autonomie énergétique est assurée par une cellule photovoltaïque et des batteries Li-On, lui apportant ainsi une contrainte d'équilibre énergétique entre production et consommation. La seconde partie, dite "fixe", constitue le point d'accès Wi-Fi par lequel les données relevées sont envoyées au tableau de bord. Elle inclus également deux capteurs environnementaux, mais cela ne constitue qu'une faible minorité des données. La partie fixe est alimentée sur secteur et sa consommation n'est pas importante dans le cadre du projet. Les données relevées par les deux parties sont envoyés par Internet à *ThingsBoard*, une plateforme IoT pour la collecte et le traitement de données. Enfin, une interface graphique *Graphana* présente les données sous la forme d'un tableau de bord accessible au plus grand nombre. La station exploite des capteurs achetés en ligne, c'est-à-dire livrés fonctionnels avec une fiche technique, et deux capteurs réalisés sur mesure par l'équipe du projet faute d'offre satisfaisante sur le marché : l'anémomètre et le détecteur de foudre.

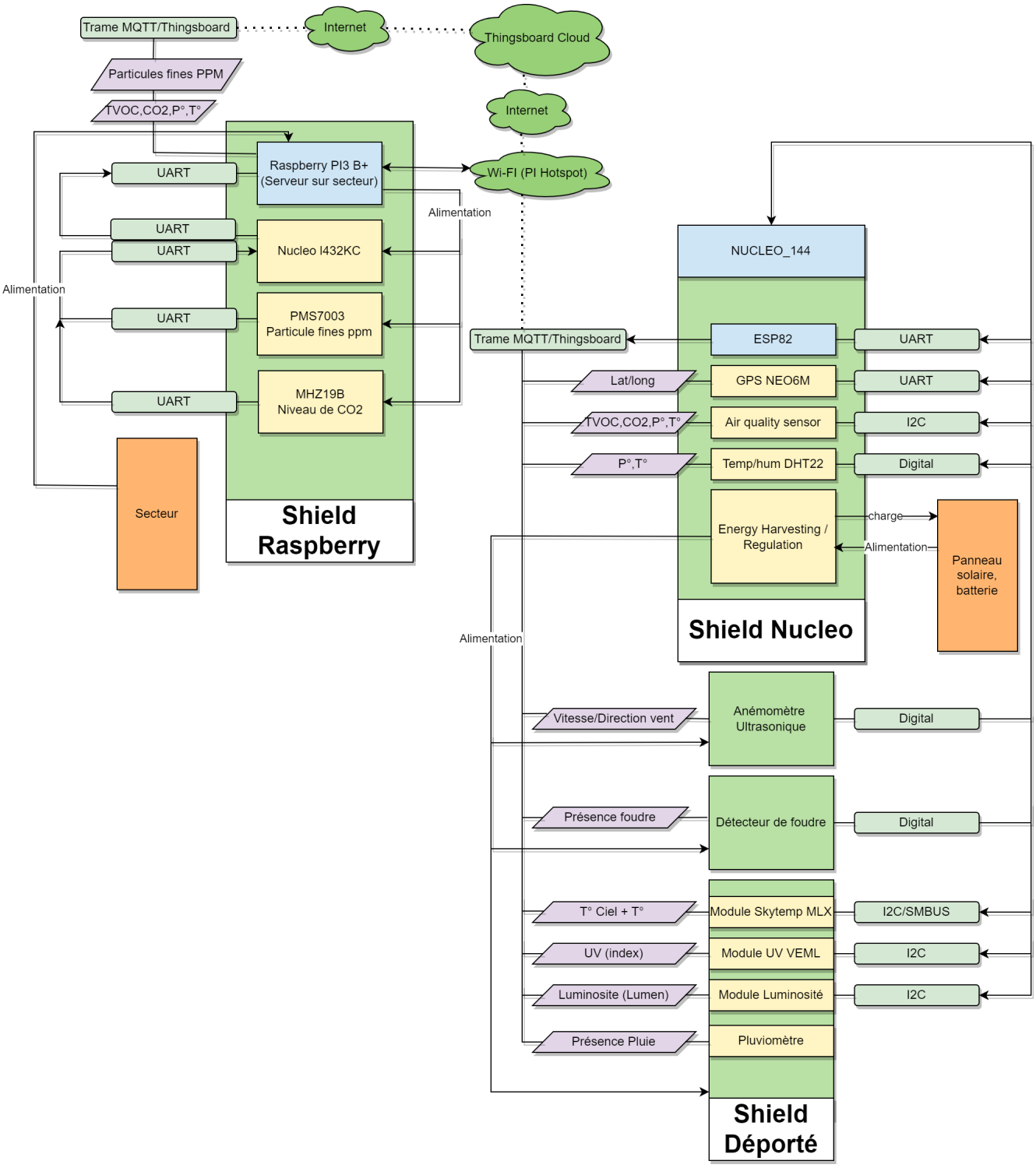


FIGURE 2 – Synoptique global de la station

La station entière est composée d'une multitude de composants, placés sur des circuits imprimés (ou PCB pour *Printed Circuit Board*) en fonction de leur utilisation. Les composants ayant les mêmes besoins ou les mêmes conditions d'utilisation sont regroupés sur le même

circuit. Ces circuits sont au nombre de 5 :

- le *Shield Raspberry*, qui relie l'ensemble de la partie fixe du système. Il est monté sur la carte Raspberry Pi et héberge deux capteurs. La Raspberry Pi fait remonter périodiquement les données de ces capteurs et agit comme point d'accès Wi-Fi pour que la partie embarquée puisse envoyé ses propres relevés à *ThingsBoard*.
- le *Shield Nucleo*, qui est le circuit principal de la partie embarquée. Il intègre la carte Nucleo-144, qui inclus le microcontrôleur principal STM32F767ZI, la récolte et conservation de l'énergie (panneau solaire et batterie), le module GPS, le module ESP8266 (pour communiquer par Wi-Fi avec le serveur) ainsi que tous les capteurs environnementaux qui n'ont pas besoin de conditions d'implémentation particulière.
- le *Shield déporté* sur lequel sont soudés les capteurs ayant besoin d'avoir une ligne de vue directe avec le ciel : capteurs de température de ciel, de luminosité, d'intensité lumineuse ultraviolette.
- le circuit constituant le détecteur de foudre. Il s'agit d'un des deux capteurs réalisés par l'équipe pour le besoin du projet : leur conception analogique impose un circuit qui leur est propre. De plus, le capteur de foudre doit être exposé à l'extérieur pour fonctionner d'où le fait qu'il soit déporté et non pas intégré sur le *Shield Nucleo*.
- le circuit constituant l'anémomètre. Il s'agit du deuxième capteur réalisé sur-mesure pour le projet. Tout comme le détecteur de foudre, il doit être exposé au vent pour fonctionner, d'où le choix de l'avoir déporté du *Shield Nucleo*.

3 Capteurs, données, points d'intérêt du code

3.1 Répartition et exploitation des capteurs

Au total, 10 capteurs sont utilisés sur la station. Ils permettent de relever la plupart des données environnementales nécessaire à l'établissement d'une prévision météorologique. Dans le cas de l'anémomètre et du détecteur de foudre, les composants proposés sur le marché laissent à désirer par leur fiabilité, leur taille ou leur consommation ; ainsi ces deux capteurs furent conçus sur circuit par oppositions aux autres qui sont "*off the shelf*". Certains composants mesurent les mêmes grandeurs (par exemple, la station comporte plusieurs thermomètres). En accord avec la portée éducative du projet, le choix d'offrir le plus de diversité a été retenu et toutes ces données sont accessibles ; leur finalité est laissée aux utilisateurs futurs du projet.

3.1.1 Capteurs côté embarqué

Capteurs	Mesures	Communication	Emplacement	Régulation de l'alimentation
DHT22	Température, humidité	Custom	Shield Nucleo	par transistor
BMP280	Pression de l'air, Température	I2C	Shield Nucleo	toujours alimenté
CCS811	TVOC, niveau de CO_2	I2C	Shield Nucleo	toujours alimenté
HDC1080	Température, Humidité	I2C	Shield Nucleo	toujours alimenté
NEO-6M GPS	Longitude, latitude	UART	Shield Nucleo	par transistor
Détecteur de foudre	Présence de foudre	Tension (analogique)	PCB indépendant	toujours alimenté
Anémomètre	Vitesse du vent	Tension (analogique)	PCB indépendant	par transistor
MLX90614	Température ambiante et du ciel	I2C	PCB déporté	par transistor
VEML6070	Indice UV	I2C	PCB déporté	mode veille par I2C
TCS34725	Luminosité	I2C	PCB déporté	mode veille par I2C
Pluviomètre	Quantité d'eau sur le capteur	Lecture de *GPIOs*	PCB déporté	par transistor

FIGURE 3 – Liste des capteurs

La consommation de tous les éléments côté embarqué est contrôlée afin de garantir l'autonomie du dispositif ; cette contrainte entre en conflit avec l'objectif de simplicité de prise en main du projet, car le circuit électrique et le code en sont quelque peu complexifiés. En effet, pour les capteurs, le module GPS, et l'ESP8266, usage est fait de transistors (pilotés par *GPIOs* du STM32F767ZI) coupant l'alimentation ou de modes ultra-basse consommation par configuration idoine des registres. Le choix a été fait d'équilibrer au mieux la nécessité de consommer le moins possible tout en libérant le plus de *GPIOs* possible sur la Nucleo-144 en vue d'éventuelles modifications de la station dans le futur. En conséquence de quoi, les composants qui le permettent sont mis en veille (courant de l'ordre du μA) par commande (*via* bus de communication ou pilotage de certains pins) ou, si cette option n'est pas disponible, ne sont alimentés que lorsque le STM32 effectue une ronde de collecte de données. De plus amples explications sur les optimisations de consommation, en *software* ou en *hardware*, sont disponibles dans cette partie. Le tableau 3 résume l'utilisation faite des composants de mesure du projet.

Concernant le fonctionnement dans le détail des composants de mesures, nous renvoyons à leur fiche technique. Toutefois, nous proposons ici des remarques composant par composant afin d'attirer l'attention sur les points d'intérêt relevés lors de la démarche de conception :

- **DHT22** : Capteur de température et d'humidité. Il s'utilise avec un protocole de communication dressé par le fabriquant, qui nécessite que le module soit branché sur des pins qui lui sont propres. Comme indiqué sur la datasheet, le capteur à besoin d'être allumé pendant 1 seconde avant de pouvoir l'utiliser cela a pour conséquence que ce capteur est allumé avant le TCS34725 qui a également une longue initialisation géra par la librairie du capteur.
- **BMP280**, **CCS811**, **HDC1080** : Ces trois capteurs sont présents sur le même module. Comme l'explique sa fiche technique, le CCS811 a un temps d'établissement d'une vingtaine de minutes avant de produire des mesures cohérentes. Par conséquent, il doit être alimenté en permanence ; or l'alimentation au module est commune aux trois capteurs ce qui n'est pas optimal, car les deux autres pourraient être exploités avec un contrôle plus fin de leur alimentation. Cependant, le CCS811 est initialisé dans son mode de consommation le moins gourmand (*DRIVE_MODE_60SEC*) et, de plus, nous exploitons son pin WAK (voir fiche technique) pour pousser l'économie. À noter que le CCS811 propose une interruption sur dépassement de seuil dont la valeur est paramétrable avec le pin INT, mais que ce pin n'est pas routé faute de place. Le pin ADR, routé à 5V, fixe l'adresse du CCS811 à $0x5A$. Le BMP280 et le HDC1080 proposent une mesure de température, mais celle-ci peut être faussée par l'émission de chaleur du CCS811, d'où la redondance de l'information grâce au DHT22 et au MLX90614.
- **NEO-6M GPS** : Ce module, dont les informations de latitude et de longitude ne sont remontés qu'à l'initialisation du microcontrôleur principal permet de localiser la station. Il fonctionne grâce à une liaison série "*SoftwareSerial*". Contrairement à une liaison HardwareSerial qui utilise un périphérique spécifique, SoftwareSerial émule le comportement de HardwareSerial ce qui permet d'utiliser n'importe quel pin *GPIO* de la carte. Cela permet de libérer plus de liaison série pour des questions de modula-

rité, puisque ce composant n'est utilisé qu'une fois. Le module utilisé dans la station comporte notamment une pile ce qui lui permet de garder en mémoire sa dernière position connue, ce qui accélère sa reconnexion aux satellites en cas de redémarrage (différence entre *cold start* et *hot start* dans la fiche technique). A l'initialisation de la carte, le GPS tente de contacter la station GPS en boucle pour obtenir la position (latitude, longitude). Cette routine est bloquante mais il est possible de bypass cette étape d'initialisation en appuyant sur le bouton bleu ciel de la Nucleo-144.

- **MLX90614** : il est utilisé comme capteur de température de ciel, c'est-à-dire qu'il doit être pointé vers le ciel pour fonctionner. Il propose également une mesure de température ambiante. Pour une raison inconnue, il entre en conflit avec le CCS811 lorsqu'ils sont sur le même bus I2C : il est impossible de communiquer avec eux en succession, alors que leur adresses sont pourtant bien différentes. Ainsi, le ML90614 est isolé sur le bus I2C2, tous les autres composants sont sur I2C1. Son adresse a été changée à $0x69$ (par défaut $0x5A$).
- **VEML6070** : il propose un mode ultra-basse consommation avec courant d'environ $1\mu A$, commandé par I2C. À noter que la mesure du capteur donne une valeur sur 16 bits en unités arbitraire et qu'il faut faire un travail de caractérisation et d'interprétation, laissé à des développements futurs, pour lui donner un sens. Ce capteur propose une interruption (pin ACK) si l'intensité détectée dépasse un seuil paramétrable, fonctionnalité que nous avons laissée disponible aux utilisateurs futurs en routant ce pin sur le circuit.
- **Le détecteur de foudre** : produit une tension dépendant des pics d'électricité statique venant des nuages. Cette grandeur analogique est scrutée par un pin en *digital input* qui peut déclencher une interruption sur front montant. Lorsque cette interruption est déclenchée, elle provoque le réveil du microcontrôleur STM32F767ZI qui procède à une récolte et à un envoi des données, en précisant que de la foudre vient d'être détectée. Ceci implique une irrégularité dans la période des mesures qui peut être facilement mis en évidence côté serveur. L'idée est que le nombre de pics d'électricité statique en temps normal est telle que la tension de sortie du détecteur constitue un "0" logique, mais qu'un nuage d'orage en implique tellement que cette tension augmente suffisamment pour constituer un "1" logique et déclencher l'interruption.
  Ce capteur est basé sur celui présenté dans ce projet.
- **Anémomètre** : L'anémomètre ultrasonique permet de mesurer la direction et la vitesse du vent sans partie mobile, permettant de réduire l'usure du capteur. Le capteur fonctionne en utilisant les différences de phases entre les différents récepteurs intégrés. Une explication plus précise du fonctionnement de celui-ci est donnée dans la partie 5.2.
- **Pluviomètre** : Le pluviomètre est un module qui mesure la quantité de pluie tombée. Il est composé d'un circuit amplificateur/comparateur et d'un panneau résistif. Le circuit détecte les changements de résistance du panneau lorsqu'il est mouillé par la pluie et envoie un signal à la Nucleo-144. La sortie analogique varie entre un seuil de tension bas (forte pluie) et un seuil de tension haut (pas de pluie). Un potentiomètre permet de régler le seuil de détection de la pluie. Lorsque ce seuil est atteint, un signal est envoyé pour indiquer la présence de pluie. Le pourcentage de pluie est calculé en

effectuant une régression linéaire à partir des seuils lus pour une inondation minimale et maximale.

- **TCS34725** : Ce module est originalement prévu pour effectuer des mesures de couleurs. Cependant en fusionnant les valeurs d'éclairement des photo-diodes polarisées RGB du module, il est possible d'obtenir la luminosité ambiante. Cette fusion est réalisé par la librairie fournie avec le module. Il faut noter que le module utilise (en interne) une intégration, signifié dans le constructeur de la classe du module par "*TCS34725_INTEGRATIONTIME_614MS*". Cela a pour effet de moyenner la valeur lue par le capteur mais cela introduit un délai bloquant. En l'occurrence environ $2.4 \times 614ms \simeq 1.5s$. On attire enfin l'attention sur le fait que la documentation est floue sur le fait que la valeur indicatrice de la luminosité est en unités arbitraire et doit faire l'objet d'une interprétation ou si elle est déjà en lux.

3.1.2 Capteurs côté fixe

La Raspberry Pi accueille 2 capteurs. Étant donné que la Raspberry est branchée sur secteur, il n'y a pas de problème de consommation d'énergie.

Capteurs	Mesures	Communication	Régulation de l'alimentation
PMS7003	Particules entre 10 et 0.3 µm	UART	N/A (côté fixe)
MH-Z19B	Niveau de CO_2	UART	N/A (côté fixe)

FIGURE 4 – Liste des capteurs sur la Raspberry Pi

La Raspberry n'a qu'un seul port Tx et Rx, ce qui pose un problème pour communiquer avec deux capteurs en UART. Il est donc nécessaire d'utiliser une carte *Nucleo-L432KC* entre les deux, car elle dispose de suffisamment de lignes en UART. Ainsi, la carte *Nucleo-L432KC* est branchée sur les deux capteurs, et communique ces données à la Raspberry Pi grâce à une troisième connexion. Les données sont sous forme d'une chaîne de caractère dont chaque valeur est séparée par une virgule.

À l'instar de la partie précédente, voici quelques remarques sur le fonctionnement des capteurs sur la Raspberry Pi :

- **Capteur de qualité de l'air *PMS7003*** Communiquant en UART, le *PMS7003* est utilisé avec une librairie stockant les données reçues dans une structure prévue à cet effet. Cette structure regroupe le nombre de particules détectées par le capteur en

fonction de leur taille. On compte les particules de 10, 5, 2.5, 1, 0.5 et 0.3 µm. Ces nombres sont envoyés à la Raspberry dans l'ordre décroissant. Le capteur fonctionne sur 5V.

- **Capteur de CO_2 *MH-Z19B*** Le capteur de CO_2 envoie par l'UART le taux de dioxyde de carbone dans l'air en «partie par million» ou «ppm». Cette valeur est envoyée à la Raspberry à la fin de la chaîne de caractère, après les données du capteur *PMS7003*. Ce capteur fonctionne aussi sur 5V.

Un script Python sur la Raspberry Pi envoie le chiffre *"1"* à la Nucleo *via* l'UART, selon un délai choisi par l'utilisateur. La Nucleo envoie les données des capteurs à la Raspberry lors de la réception de ce *"1"*. Le script indexe ensuite ces données dans une variable au format JSON, qui est envoyée sur *ThingsBoard* grâce à un protocole appelé *MQTT*, présenté dans cette section.

Activation du relevé :

Lorsque la Raspberry Pi est allumée, la LED rouge de la Nucleo clignote, signifiant qu'elle n'est pas activée. Il faut alors brancher puis débrancher en USB la Nucleo. La LED ne clignotera plus et restera allumée, indiquant que la carte est prête. La Nucleo reste dans cet état jusqu'à ce que la Raspberry Pi soit débranchée.

Une fois la carte activée, lancer le script python *get_indoor_data.py*. Les données seront ainsi récoltée par *ThingsBoard*. Dans le script, le délai entre chaque relevé peut être modifié.

3.2 Algorithmique côté embarqué

Le cœur de la station est le STM32F767ZI qui contrôle l'ensemble de la partie embarquée. Nous présentons ici son algorithmique générale et les particularités de son fonctionnement pour compléter les commentaires rédigés dans le code du projet. Un algorigramme du flux d'exécution global du système est proposé en figure 5.

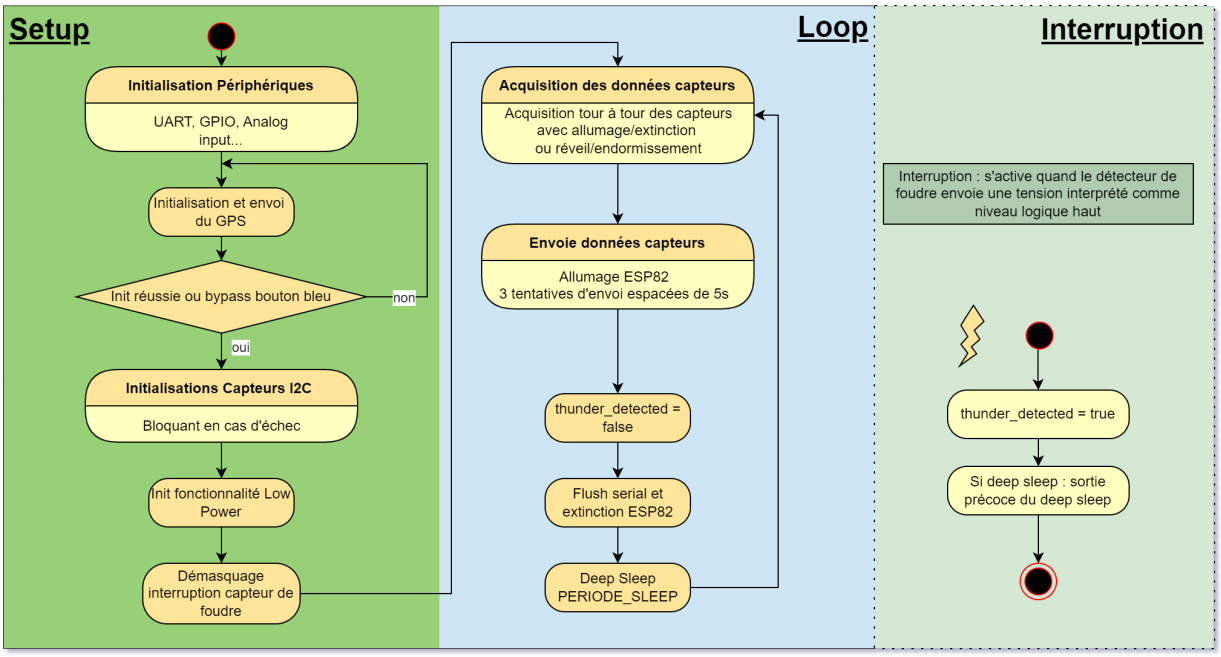


FIGURE 5 – Algorigramme global du code du STM32F767

Lors de la mise sous tension du système, tous les capteurs sont initialisés puis le module GPS est mis sous tension pour obtenir la position de la station. Une trame spécifique permet de mettre à jour l'affichage côté serveur. Le GPS n'est plus utilisé par la suite et reste éteint lors du fonctionnement nominal (boucle *loop*). La recherche de satellite est une opération coûteuse en temps et en énergie et l'utilisation périodique du module GPS est difficilement envisageable en terme de consommation. Lors de l'initialisation (fonction *setup*, pour les composants qui le permettent, nous bloquons par un *while(1)* en cas d'erreur (il s'agit certainement d'une erreur matérielle qui mérite maintenance). Par contre, en fonctionnement nominal (boucle *loop*) aucun test n'est fait pour vérifier que les capteurs se comportent correctement ; nous supposons qu'il est préférable que la station continue de fonctionner même si un composant devient défaillant. De plus il est très peu probable que le composant devienne défaillant si l'initialisation a été réussie.

En fonctionnement nominal (boucle *loop*) tout comme en phase de *setup*, chaque composant est mis sous tension ou sorti de veille, le STM32F767ZI interagit avec, puis le composant est éteint ou repasse en veille. Ainsi, chaque interaction avec un composant est entouré par une "balise" de code qui l'active en amont puis qui le désactive en aval.

Après l'initialisation, le fonctionnement est cyclique : le système alterne une courte phase active où tous les données sont récupérées auprès des capteurs avec une longue phase de veille où les composants de mesure sont éteints ou en veille et le microcontrôleur est en mode STOP (fonction *deepSleep*). Plus d'informations à ce sujet dans cette partie. Une interruption, déclenchée par le détecteur de foudre, peut réveiller le microcontrôleur pour signaler un éclair. Cela provoque alors une ronde de récolte de données et, en particulier, le

booléen qui signale que la foudre est tombée est mis à jour en accord.

L'affichage de messages d'état peut être activé en décommentant le *#define DEBUG* dans *msg_ESP.h* et *debug.h*. Dans ce cas, le port UART de *debug* du ST-LINK (sur UART3, instance nommée *Serial*) est initialisé et utilisé à ces fins. En fonctionnement normal de la station, cela doit être désactivé pour économiser de l'énergie (périphérique désactivé et moins de temps d'activité du microcontrôleur).

3.3 Envoi des données vers le serveur

3.3.1 Utilisation de l'ESP8266

Les données des capteurs récoltés par la station sont envoyées au *Cloud ThingsBoard*, en passant par la Raspberry Pi qui agit en point d'accès, *via* WiFi grâce au module ESP8266.

FIGURE 6 – Photo de l'ESP8266.

Les données sont envoyées grâce à un protocole appelé **Message Queuing Telemetry Transport**, ou **MQTT**. Le principe est de créer un serveur de message appelé *Broker* qui va recevoir les données envoyés par les appareils en tant que publicateurs. D'autres appareils peuvent se connecter au serveur *MQTT* en tant qu'abonnés afin de recevoir les données.

L'intérêt de ce protocole est d'abord la légèreté et la simplicité des transmissions. De plus, une connexion continue au serveur n'est pas nécessaire. L'ESP8266 peut s'y connecter temporairement et y envoyer ses données, puis s'éteindre sans problème, minimisant ainsi l'énergie consommée.

Lors du développement du système d'envoi des données, nous avons d'abord pensé à une solution utilisant les trames de données les plus courtes possible. Mais voyant que l'ajout d'un nouveau capteur serait difficile pour un novice, nous avons finalement décidé d'opter pour un mode opératoire plus simple, mais plus gourmand en ressources (trames plus longues).

Première solution : Envoi de données en binaire

Pour cette première solution, nous avons flashé l'ESP8266 afin de pouvoir programmer dessus. Le module fonctionne ainsi de manière autonome. Lorsque la carte Nucleo-144 veut envoyer des données au serveur, elle communique avec l'UART de l'ESP8266, et lui envoie en brut une *struct* (notion de langage C) contenant les données des capteurs. L'ESP8266 tente ensuite la connexion par Wi-Fi, et envoie par le protocole MQTT cette même structure au *Broker* sur la Raspberry Pi.

En envoyant une structure en binaire, on s'assure d'envoyer le moins d'octets possible et on réduit grandement le *time-on-air*. On diminue ainsi l'impact énergétique de l'ESP8266, qui consomme beaucoup d'énergie lors de la communication.

Par la suite, la structure en binaire est réceptionné et décodé en clair grâce à un script python.

Cette solution permet la plus grande maîtrise, mais implique de devoir reprogrammer l'ESP8266 et le script python sur la Raspberry lors de l'ajout d'un nouveau capteur. De plus, pour s'assurer que la trame communiquée de la Nucleo-144 vers l'ESP8266 est correcte, un système de contrôle en vérifiant la cohérence des données envoyées (*checksum CCRIT-16*) est mis en place. Ce système doit être revu à chaque fois que l'on ajoute un capteur. Il aurait été possible d'en faire une amélioration (à coût énergétique minime) pour ne pas avoir à reprogrammer l'ESP8266 mais la difficulté reste la même quand à la programmation du script python. Ces deux points ne sont pas en accord avec la volonté du client, pour qui le système doit aussi être éducatif et facile à programmer plutôt qu'un système optimisant le plus possible son énergie.

Deuxième solution : Utilisation de la librairie *ThingsBoard*

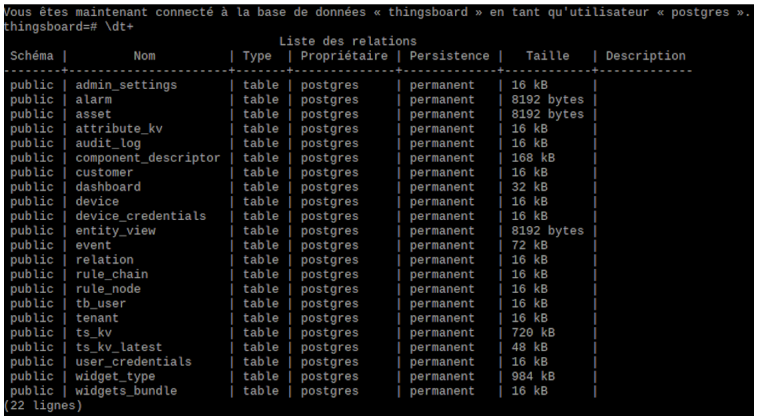
La deuxième solution finalement retenue est d'utiliser l'ESP8266 en tant que périphérique de la carte Nucleo-144 plutôt qu'en tant que microcontrôleur *standalone*. L'ESP8266 a été *flashé* avec un nouveau *firmware*, permettant de lui communiquer des commandes AT (des chaînes de caractères simples, exécutant diverses actions). Sur la Nucleo-144, on installe de nouvelles librairies profitant de ces commandes. Ainsi, c'est le STM32F767ZI qui gère les commandes de connexion et de communication Wi-Fi. On évite ainsi de devoir reprogrammer l'ESP8266 pour ajouter des capteurs, ce qui facilite la prise en main par l'utilisateur novice. Pour envoyer les données, on utilise *ThingsBoard Community Edition*, un service qui permet de faciliter la gestion des protocoles MQTT et de la base de données. Le système traite automatiquement les données envoyées sur MQTT, et les repartit dans la base données choisie mais aussi ajoute des informations comme l'identifiant de l'appareil mais aussi un /timestamp pour chaque données reçues .

ThingsBoard attribue un *token* unique à chaque station environnementale. Celui-ci est écrit en dur dans le code exécuté sur la Nucleo-144. Il permet d'identifier les données envoyées et de

les regrouper. Sur la Nucleo-144, on installe la librairie *ThingsBoard* qui s'occupe de l'envoi de valeurs. Désormais, cet envoi se fait grâce à la fonction *sendTelemetry* de la librairie. Chaque donnée est associé à une chaîne de caractère. On reçoit ainsi sur le *dashboard* du serveur la valeur et son nom. L'ajout d'un capteur est désormais plus simple. Pour envoyer ses données, il suffit d'utiliser la fonction *sendTelemetry*. Il n'est plus question de modifier les structures ou d'envoyer des données en binaire. L'ESP8266 est ainsi transparent pour l'utilisateur. Cependant, on peut émettre une critique sur ce nouveau système, qui en envoyant des longues chaînes de caractères allonge le temps *on-air* de l'ESP8266. Le système consomme alors plus d'énergie. Une réflexion plus poussée à ce sujet est proposée ici.

3.3.2 Le serveur *ThingsBoard*

L'objectif du projet est d'abord de mettre le serveur *ThingsBoard* sur la Raspberry Pi. Pour cela nous suivons le tutoriel fourni par la plateforme. Cette installation comprends principalement 3 étapes. La première étape est d'installer OpenJDK sur la Raspberry Pi, en faisant attention à la version de celui-ci car les dernières versions de Java sont trop lourdes et ne sont pas optimisées pour une Raspberry Pi 3 avec 1Gb de RAM. La seconde étape est de créer une base de données Postgres (en commencant par installer PostgreSQL s'il n'est pas présent de base sur la machine), on appelle cette base de données "*thingsBoard*" et on renseignera les informations sur cette base dans le fichier de configuration de *ThingsBoard*. Enfin on télécharge, installe et lance *ThingsBoard* sur la Raspberry Pi. Une fois que *ThingsBoard* est lancé on accède à l'interface utilisateur depuis un navigateur à l'adresse http ://127.0.0.1 :8080, 127.0.0.1 étant l'adresse sur le réseau (*localhost*) et 8080 le port de connexion. Après avoir créer deux *devices* (un pour chaque carte qui envoie des données) on peut récupérer un *token* pour chaque machine. Le *token* est utilisé dans le code de la Raspberry Pi et du STM32F767ZI pour s'identifier. On peut alors maintenant recevoir et lire les données sur le site ou alors directement dans la base de donnée créée plus tôt, *ThingsBoard* se charge de créer chaque table dont il a besoin et de répartir les informations dans chaque table de la base. Par exemple toute les données reçues sont dans la table *ts-kv*, dans laquelle chaque télémétrie reçue est associé à un *device*, un *timestamp*, etc. Il exite même une table *ts-kv-latest* qui regroupe les dernières données reçues pour chaque appareil, ce qui est pratique pour créer un *dashboard* avec des données en direct, il n'est pas nécessaire de parcourir la base entière pour trouver la dernière donnée reçue.



```
Vous êtes maintenant connecté à la base de données « thingsboard » en tant qu'utilisateur « postgres ».
thingsboard=# \dt+
                                    Liste des relations
 Schéma |         Nom          | Type  | Propriétaire | Persistence |   Taille   | Description
--------+----------------------+-------+--------------+-------------+------------+-------------
 public | admin_settings       | table | postgres     | permanent   | 16 kB      |
 public | alarm                | table | postgres     | permanent   | 8192 bytes |
 public | asset                | table | postgres     | permanent   | 8192 bytes |
 public | attribute_kv         | table | postgres     | permanent   | 16 kB      |
 public | audit_log            | table | postgres     | permanent   | 16 kB      |
 public | component_descriptor | table | postgres     | permanent   | 168 kB     |
 public | customer             | table | postgres     | permanent   | 16 kB      |
 public | dashboard            | table | postgres     | permanent   | 32 kB      |
 public | device               | table | postgres     | permanent   | 16 kB      |
 public | device_credentials   | table | postgres     | permanent   | 16 kB      |
 public | entity_view          | table | postgres     | permanent   | 8192 bytes |
 public | event                | table | postgres     | permanent   | 72 kB      |
 public | relation             | table | postgres     | permanent   | 16 kB      |
 public | rule_chain           | table | postgres     | permanent   | 16 kB      |
 public | rule_node            | table | postgres     | permanent   | 16 kB      |
 public | tb_user              | table | postgres     | permanent   | 16 kB      |
 public | tenant               | table | postgres     | permanent   | 16 kB      |
 public | ts_kv                | table | postgres     | permanent   | 720 kB     |
 public | ts_kv_latest         | table | postgres     | permanent   | 48 kB      |
 public | user_credentials     | table | postgres     | permanent   | 16 kB      |
 public | widget_type          | table | postgres     | permanent   | 984 kB     |
 public | widgets_bundle       | table | postgres     | permanent   | 16 kB      |
(22 lignes)
```

FIGURE 7 – Tables créées automatiquement dans la base de données

ThingsBoard Community Edition permet de créer une visualisation des données mais nous avons utilisé Grafana qui permet de faire des *dashboard* plus complets. De même que pour les autres applications utilisées précédemment, on installe directement Grafana sur la Raspberry Pi, et on accède au service à l'adresse http ://127.0.0.1 :3000. Une fois que l'on a renseigné la base de donnée à utiliser on peut travailler directement sur les tables de celle-ci et en extraire des visualistion tels que des grapiques, des jauges, des alertes, etc. Grafana permet de simplifier la recherche des données, leur mise en forme et traitement et bien sûr, leur affichage.

En résumé, les données sont envoyé à la Raspberry PI en MQTT et sont traité par le serveur *ThingsBoard* qui sait identifier les différentes télémétries reçues mais aussi les différents appareils grâce à leur *token*. *ThingsBoard* s'occupe du traitement de ces données et de leur répartition dans une base Postgre en local sur la Raspberry Pi. Enfin on connecte l'outil Grafana sur cette base de données et nous affichons les données mises en forme.

Remarque : La Raspberry Pi 3 utilisée n'était pas la version idéale de Raspberry Pi pour cette utilisation, le manque de RAM et son CPU pas assez puissant rendent difficile l'installation et l'utilisation des logiciels (freezes, crashs, etc). Nous avons dû contourner le problème en installant des versions antérieures des logiciels mais avons quand même rencontré des problèmes de rapidité. Nous recommandons donc l'utilisation du Raspberry Pi 4 disposant d'au moins 2Gb de RAM pour les futurs projets reprenant ce concept.

4 Aspect énergétique

Un des objectifs à atteindre pour le succès du projet est l'autonomie de la partie embarquée. Il convient par conséquent d'équilibrer production et consommation d'énergie ; mécanismes détaillés dans la partie présente.

4.1 Production d'énergie

La station environnementale est composée d'une partie fixe sur secteur et d'une partie embarquée qui se doit d'être autonome pour être fonctionnelle. Pour ce faire, elle est équipée d'une panneau solaire dont la puissance électrique produite est rendue utilisable par notre système à l'aide d'un circuit de conditionnement et d'un batterie Li-ion. On détaillera dans cette partie les choix, le principe de fonctionnement et les performances associées au circuit d'extraction d'énergie.

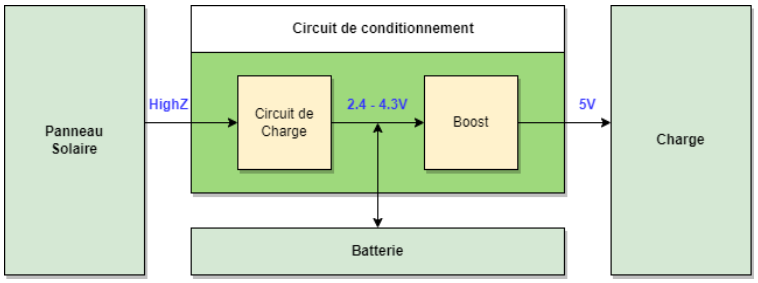


FIGURE 8 – Schéma bloc simplifié du circuit d'alimentation de la station embarquée

Dans le schéma bloc simplifié du circuit d'alimentation représenté ci-dessus on observe les contraintes associées à notre système imposées par les composants utilisés. En effet, le panneau solaire est une source d'énergie haute impédance, la charge (représentant la *Nucleo-144* et les capteurs embarqués) est alimentée avec une tension de $5V$ et on utilise une batterie Li-ion dont la tension au sein de notre système sera toujours comprise en $2.4-4.3V$. Le circuit de conditionnement a pour but de rendre compatible les différents composants utilisés et ce de la manière la plus optimale possible et d'autre part de protéger la batterie dont la technologie (Li-ion) peut présenter d'importants risques de fuite chimique, de combustion voir d'explosion.

4.1.1 Circuit de conditionnement

Le circuit de conditionnement représenté figure 8 peut être séparé en deux partie distinctes :

- **Le circuit de charge**, visant à maintenir des conditions optimale de courants et de tension aux bornes de la batterie pour préserver sa durée de vie.

- **Le Boost**, visant à stabiliser la tension d'alimentation de la charge à $5V$ indépendamment de la tension en amont.

Circuit de charge Le circuit de charge est une adaptation du très répandu module de charge TP4056 5V/1A dont on a retiré les leds de l'interface utilisateur pour des raisons de consommation. On notera sur la figure 9 qu'on utilise trois puces :

- le composant **TP4056**, chargé d'imposer l'allure de charge de la batterie. En effet, il limite la tension de batterie maximale à $4.2V$. De plus, il a été programmé à l'aide d'une résistance externe de $1,2k\Omega$ afin d'obtenir un courant de charge maximum de $1A$. Il est à noter que le TP4056 est uniquement capable d'abaisser la tension d'entrée mais pas de la hausser.

  Ce composant permet également de bloquer les courants allant de la batterie vers le panneau solaire qui constituerai une perte importante d'énergie susceptible d'arriver durant les périodes de faible intensités lumineuses.
- le composant **8205A** sert de commutateur, composé de deux NMOS en série et de diodes tête-bêche en parallèle de chacun d'eux. Il permet de bloquer le courant de GND à $BATTERIE\_-$ dans un sens ou dans l'autre et donc de déconnecter la batterie de la charge ou de la source d'énergie.
- le composant **DW01A**, quant à lui, permet de contrôler le commutateur. Il l'actionne en fonction de la tension de la batterie (détectée par lecture de tension entre les broches VCC et GND) pour prévenir une surcharge ou une décharge trop importante mais également en cas de surintensité (détectée par lecture de tension entre les broches CS et GND).

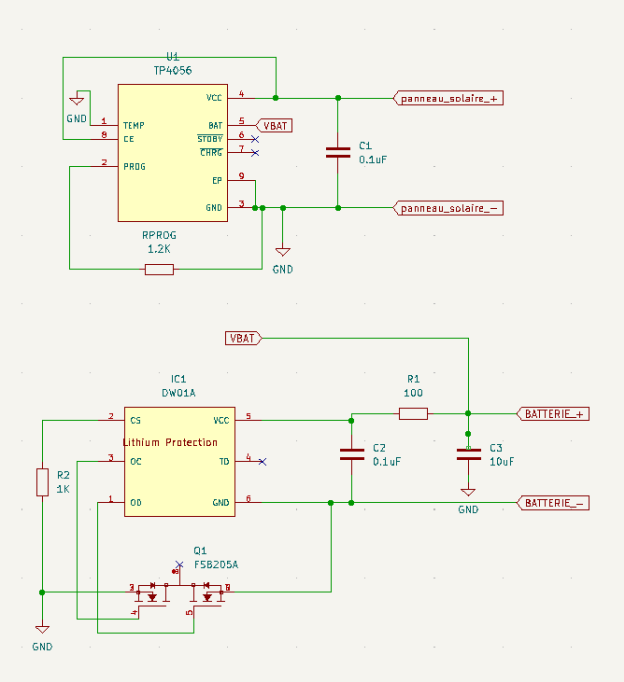


FIGURE 9 – Schéma électrique du circuit de charge

Il est à noter que le TP4056 protège la batterie uniquement contre la surcharge et sa décharge éventuelle dans le panneau solaire. De plus, il alimente la batterie et sa charge en parallèle et est donc incapable de réaliser une protection complète de celle-ci. C'est pourquoi on utilise le DW01A dont l'usage pourrait dans un premier temps sembler redondant. De plus, ce dernier permet également de protéger la batterie indépendemment de sa charge de manière exhaustive (surcharge ET décharge).

On note également que le panneau solaire est une source d'énergie haute impédance et que le TP4056 n'est pas capable d'effectuer un *MPPT* (Maximum Power Point Tracking). Cette réflexion sera développée dans la partie caractérisation de notre étude.

Circuit de boost Le circuit de boost a été réalisé au moyen du **MIC2288YD5-TR**. Il s'agit d'un module régulateur boost DC/DC qui fonctionne par switch et à l'aide d'un PWM de fréquence *1.2MHz* généré grâce à un oscillateur interne. Le circuit réalisé, représenté figure 10 est celui préconisé dans la fiche technique pour les valeurs de tension de notre système. On obtient ainsi un boost pour une tension d'entrée nominale comprise entre $3V$ et $4.2V$ et avec une tension de sortie de $5V$.

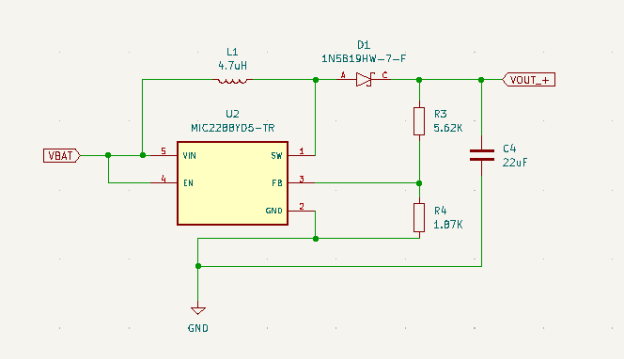


FIGURE 10 – Schéma électrique du boost

Les valeurs de tension d'entrée nominales du boost correspondent aux valeurs de tensions aux bornes de la batterie à l'exception de la plage de valeur $2.4 - 3V$. Sur cette plage de valeur de tension le rendement du boost chute. Cependant, ces valeurs de tension de batterie sont atteintes pour une batterie à la limite de la décharge ce qui correspond à un cas limite de notre utilisation. De plus, le but du dimensionnement final est d'obtenir un bilan énergétique positif. Dans ce cas de figure, une augmentation de la capacité de la batterie pouvant être réalisée par mise en parallèle de plusieurs batteries permet d'amortir la consommation en aval de celle-ci et donc de ne pas atteindre la plage valeurs de tensions de $2.4 - 3V$.

4.1.2 Caractérisation

On cherche a établir le bilan énergétique de notre système pour évaluer l'autonomie de celui-ci et éventuellement effectuer un redimensionnement du système d'acquisition d'énergie. Pour ce faire, on cherche dans un premier temps à caractériser chaque élément du système présenté précédemment afin de déterminer l'énergie disponible pour alimenter sa charge à savoir la carte *Nucleo-144* en fonctionnement et les capteurs interfacés avec celle-ci.

Caractérisation du panneau solaire La documentation technique disponible pour le panneau est inexistante, ce qui rend presque impossible toute démarche d'évaluation sérieuse du budget énergétique théorique. Nous proposons donc la meilleure méthodologie envisageable avec les moyens à notre disposition. Nous cherchons à déterminer empiriquement la puissance électrique extractible par notre panneau solaire, et son rendement. Le panneau est une source haute impédance, on le caractérise donc à son point de fonctionnement dans notre système. Pour ce faire, on réalise un montage composé uniquement du panneau solaire, du circuit de charge et de la batterie que l'on maintient à un état de charge moyen ($V_{bat} = 3.7V$).

On rappelle que le rendement du panneau est :

$$\eta_{PV} = \frac{V_{PV} \times I_{PV}}{Irr \times S}$$

On ne dispose pas materiel de mesure d'irradiance mais uniquement des données moyennes d'irradiance à Gardanne, représentées pour le mois de mars en figure 11.

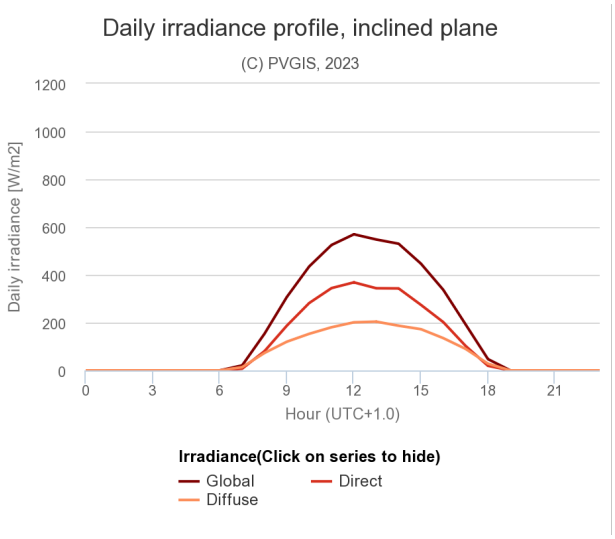


FIGURE 11 – Données moyennes d'irrandiance au mois de mars à Gardanne

On réalise donc des mesures de tension et de courant aux bornes du panneau solaire à des heures et des jours différents afin de pouvoir mettre en relation nos mesures avec les données moyennes dont nous disposons. On établit finalement un rendement moyen de notre panneau solaire sur le mois de mars que l'on étendra à un rendement moyen sur toute l'année par manque d'une meilleure approximation.

On rappelle que le TP4056 n'est pas capable de hausser la tension d'entrée afin de charger la batterie. Si la tension aux bornes du panneau solaire est inférieure à $4V$ le composant se retrouve donc dans l'incapacité d'extraire de la puissance électrique du panneau. De ce fait, il est possible d'obtenir des erreurs sur le bilan énergétique en appliquant naïvement le rendement précédemment établi sur des plage de faible ensoleillement durant lesquelles aucune puissance n'est extraite du panneau.

On cherche à établir l'irradiance limite Irr_{min} au dessus de laquelle on peut réellement extraire une puissance électrique du panneau solaire.

On rappelle que $I_{PV} \geq 0$ (imposé par le TP4056) et donc que $V_{PV} \leq V_{OC}$. Le panneau de solaire équilibre son point de fonctionnement à l'interface avec le TP4056 de telle sorte que si $V_{OC} > 4V$ il existe un point de fonctionnement V_{PV} tel que $4V \leq V_{PV} < V_{OC}$ d'où $I_{PV} > 0$ et finalement $P_{PV} > 0$.

Inversement, si $V_{OC} < 4V$ alors $V_{PV} \leq V_{OC} < 4V$ donc le TP4056 ne permet pas l'extraction de puissance électrique.

On a donc cherché l'irradiance seuil pour laquelle $V_{OC} = 4V$. Cette mesure a été en réalité une mesure d'heure pour les raisons précédemment expliquées. On a obtenu une passage de V_{OC} en dessous de $4V$ à 19h pour une irradiance nulle selon nos données. Il est tout de même à noter que le TP4056 n'est pas capable de réaliser un MPPt et donc que la puissance tirée dans les cas limites d'irradiance est extrèmement faible. Le modèle d'irradiance nulle sur cette plages horaires nous convient finalement. On a donc choisi de modéliser la puissance électrique extraite de la manière suivante :

$$P_{PV,moy} = \eta_{PV,moy} \times Irr_{moy} \times S$$

On applique cette formule à l'irradiance moyenne au mois de décembre qui constitue notre pire cas d'ensoleillement. On obtient alors :

$$P_{PV,moy} = 171mW$$

Prise en compte des courants de fuite On a précédemment caractérisé le panneau solaire à son point de fonctionnement dans notre système. Il faut cependant tenir compte de la consommation du circuit de charge en lui-même. Pour ce faire on se réfère aux fiche techniques associées car la mesure de ces valeurs sur les composants CMS utilisés s'avère

compliquée sans multiplier les PCBs utilisés. On utilisera les valeurs maximales afin d'obtenir un bilan pour le pire cas.

- La fiche technique du **TP4056** nous indique une courant maximal consommé par la puce de $I_{CC,max} = 500\mu A$ (mode charge).
  On obtient :
  $$P_{TP4056} = I_{CC,max} \times 4.2 = 2.1mW$$

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
V_{CC}	Input Supply Voltage		●	4.0	5	8.0	V
I_{CC}	Input Supply Current	Charge Mode, R_{PROG} = 1.2k	●		150	500	µA
		Standby Mode (Charge Terminated)	●		55	100	µA
		Shutdown Mode (R_{PROG} Not Connected, V_{CC} < V_{BAT}, or V_{CC} < V_{UV})	●		55	100	µA

FIGURE 12 – Extrait de la section "ELECTRICAL CARACTERISTICS" de la fiche technique du TP4056

- La fiche technique du **DW01A** indique un courant de fuite maximal de $I_{CC,max} = 6\mu A$ à la tension nominale de la batterie de $V_{CC} = 3.6V$. On utilisera cette valeur en supposant que la tension de la batterie se situe en moyenne autour de cette valeur.
  On obtient :
  $$P_{DW01A} = V_{CC} \times I_{CC,max} = 21.6\mu W$$

PARAMETER	TEST CONDITIONS	SYMBOL	Min	Typ	Max	UNIT
Supply Current	Vcc=3.6V	Icc		3.0	6.0	µA
Power-Down Current	Vcc=1.8V	Ipd			4	µA
OD Pin Output "H" Voltage		Vdh	Vcc-0.1	Vcc-0.02		V
OD Pin Output "L" Voltage		Vdl		0. 1	0.5	V
OC Pin Output "H" Voltage		Vch	Vcc-0.1	Vcc-0.02		V
OC Pin Output "L" Voltage		Vcl		0.1	0.5	V

FIGURE 13 – Extrait de la section "Electrical Characteristics" de la fiche technique du DW01A

- La fiche technique du **8205A** présente la courbe $R_{DS(ON)}(V_{GS})$ représentée ci-dessous. Le cas du switch entièrement bloqué est une sécurité et ne fait pas partie du fonctionnement normal de notre système. On suppose donc que le switch est toujours passant. Le courant ne peut passer que dans un sens dans le switch. Un unique transistor participe à la dissipation de puissance à chaque instant. L'état haut de la tension de grille V_{GS} (commandée par le DW01A) est de $V_{CC,DW01A} - 0.02 \approx V_{Bat}$ comme indiqué figure 13.

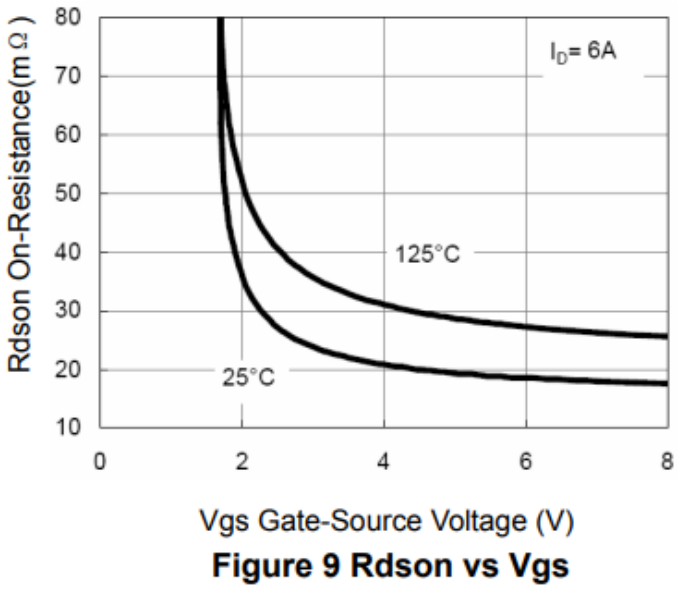


FIGURE 14 – Courbe $R_{DS(ON)}(V_{GS})$ extraite de la fiche technique du 8205A

On suppose qu'on a $V_{GS} = V_{bat,moy} = 3.7V$ et donc que $R_{DS(ON)} = 25m\Omega$. On a :

$$P_{8205A} = R_{DS(ON)} \times |I_{BAT}|^2$$

Le circuit de charge peut alimenter la batterie mais également la charge directement. De ce fait, on a $|I_{BAT}| = \frac{|P_{PV} - P_{LOAD}|}{V_{BAT}}$ et on obtient :

$$P_{8205A} = R_{DS(ON)} \times \frac{|P_{PV} - P_{LOAD}|^2}{V_{BAT}^2}$$

On ne peut pas raisonner directement sur les puissances moyennes mises en jeu au sein de notre système du fait de la non linéarité de l'expression de P_{8205A} établie ci-dessus. Une réflexion plus poussée sur P_{8205A} sera donc menée dans la partie dédiée au bilan énergétique global en prenant en compte la consommation de la charge de notre système.

Boost Le circuit régulateur de tension boost nous permet d'alimenter la charge de notre système d'acquisition d'énergie à une tension stabilisée de $5V$. Pour mesurer la consommation de la carte *Nucleo-144* et de ses différents périphériques on utilise une carte ***STM32L562E-DK Discovery kit*** en conjonction avec le logiciel ***STM32CubeMonitorPwr*** afin d'obtenir des courbes de courant consommé sur un cycle complet d'exécution du code. Cependant, la tension délivrée le dispositif de mesure est fixée à 3.3V alors que celle d'alimentation de la partie embarquée est de 5V. On utilisera donc le boost pour l'interfacer avec la charge à mesurer. De ce fait, **la consommation du boost est en réalité comprise dans les mesures de consommation de la *Nucleo-144* et de ses périphériques**. On caractérise cependant le circuit régulateur de tension boost isolé afin de pouvoir raisonner sur la cohérence de nos mesures et de mettre en lumière de potentielles pistes d'améliorations.

Il est à noter que la fiche technique du MIC2288YD5-TR indique un courant de fuite maximal de $5mA$ que l'on observe en effet dans nos mesures. Ce courant de fuite représente une consommation de l'énergie disponible à notre système et est donc pris en compte dans le rendement du boost défini comme :

$$\eta_{boost} = \frac{V_{out} \times I_{out}}{V_{in} \times I_{in}}$$

On applique en entrée une tension $V_{in} = 3.7$ (tension nominale d'une batterie Li-ion) et on interface une résistance R en sortie du circuit. On rappelle qu'on a $V_{out} = 5V$ et donc $I_{out} = \frac{V_{out}}{R}$. En mesurant I_{in} on peut tracer la courbe expérimental $\eta_{boost}(I_{out})$ représentée ci-dessous.

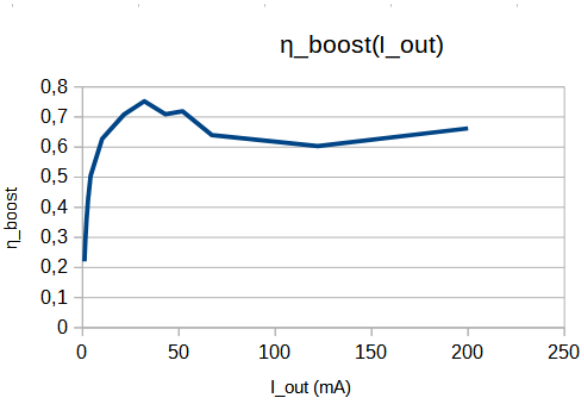


FIGURE 15 – Courbe expérimentale de $\eta_{boost}(I_{out})$

On note que la courbe expérimentale de $\eta_{boost}(I_{out})$ est cohérente avec celle présentée dans la fiche technique du MIC2288YD5-TR. Il est également à noter que le rendement est très mauvais pour des valeurs de I_{out} de l'ordre de $10mA$ qui correspond à la consommation de la charge lorsque la carte *Nucleo-144* est en mode économie d'énergie. Cependant, on obtient un rendement correct au alentour de 70% pour des valeurs de courant de charge plus élevées.

4.1.3 Bilan

Le module d'acquisition d'énergie de la station embarqué fournit une puissance $P_{disp} = \eta_{boost}(P_{PV} - P_{TP4056} - P_{DW01A} - P_{8205A})$. En réalité, les conditions de mesures de la consommation de la charge exigent de définir plutôt $P_{disp} = P_{PV} - P_{TP4056} - P_{DW01A} - P_{8205A}$. Comme expliqué précédemment on ne peut pas raisonnablement définir P_{8205A} sans s'intéresser à P_{LOAD}. On peut tout de même conclure :

$$P_{moy,disp} < P_{PV,moy} - P_{TP4056,moy} - P_{DW01A,moy} \approx 168.9mW$$

4.2 Optimisations de consommation

4.2.1 Optimisations sur le *software*

Plusieurs méthodes sont mises en oeuvre afin de diminuer la consommation de la partie embarquée tout en préservant sa fonctionnalité. D'un point de vue *software*, nous avons tenté de diminuer au maximum les fréquences de fonctionnement car la consommation est proportionnelle au carré de cette dernière. Nous attirons toutefois l'attention sur le fait que la carte et le microcontrôleur ne sont pas fait pour les applications *low-power* et que de meilleurs résultats seraient obtenus en passant à du matériel plus adapté.

L'utilisation d'un mode STOP est un des choix dont l'impact est le plus important. Dans ce mode, la plupart des horloges et périphériques sont désactivés et le microcontrôleur n'exécute plus d'instruction (le *program counter* ne bouge plus). Le microcontrôleur alterne des périodes courtes d'activité en mode RUN, durant lesquelles il récolte les données et les envoie au serveur, avec de longues périodes d'attente. Le microcontrôleur passe en mode STOP (fonction *deepSleep*) avec réveil par RTC lors de ces dernières. Le mode STOP est le mode le plus économe en énergie qui conserve la mémoire vive ; l'utilisation d'un microcontrôleur plus adapté permettrait d'accéder à un mode STOP0 encore plus économe. Nous avons vérifié que la librairie configure correctement le système pour consommer le moins possible en mode STOP : régulateur de tension en mode low-power, horloge LSI utilisée pour la RTC. Ainsi, le système passe la plupart de son temps en mode basse consommation.

Afin de diminuer la consommation lors des phases de mesure (mode RUN), nous avons diminué le plus possible les fréquences de l'arbre d'horloge. De notre expérience, il est impossible de réduire les fréquences au-delà de ce qui est proposé sans briser le fonctionnement de certaines librairies. Comme nous ne recherchons pas la vitesse, nous pouvons désactiver le mode *overdrive* (qui permet d'augmenter les fréquences en consommant plus) et nous pouvons passer le *Power Regulator* de *Voltage Scale 1* (plus énergivore et plus rapide) à *Voltage Scale 3* (moins énergivore et moins rapide). Nous renvoyons ici à la fiche technique du microcontrôleur pour plus de détails. Nous avons utilisé l'interface graphique de *Cube IDE* pour paramétrer notre arbre d'horloge comme nous venons de le décrire, puis nous avons copié la fonction *SystemClock_ Config* générée par le logiciel dans notre *main.cpp* côté *PlatformIO*. Nous remplaçons ainsi la fonction disponible par défaut dans le projet, trop énergivore puisqu'elle cherche à avoir les plus hautes fréquences possibles. Notre définition supplante la fonction par défaut car elle est déclarée ailleurs avec le mot-clé *WEAK*. Le seul abaissement des fréquences de l'arbre d'horloge permet, en mode RUN, de passer pour le microcontrôleur seul d'une consommation de $44.5mA$ à $14.8mA$, ce qui est cohérent avec les valeurs annoncées dans sa fiche technique (figure 16, notre configuration fait passer f_{HCLK} de $216MHz$ à $24MHz$).

Parameter	f_{HCLK} (MHz)	Typ	Max(1)			Unit
			T_A = 25 °C	T_A = 85 °C	T_A = 105 °C	
Supply current in RUN mode	216	92	104	150	-	mA
	200	86	97	143	170	
	180	76	85	119	140	
	168	67	75	107	126	
	144	52	58	84	101	
	60	23	28	54	71	
	25	11	15	42	56	

FIGURE 16 – Extrait de la table 25, page 116 de la fiche technique du STM32F767ZI

Une mention particulière doit être faite de l'ESP8266 et de la communication par Wi-Fi. Le module est éteint lorsqu'il n'est pas utilisé et rallumé à chaque envoi de données. Comme cela nécessite de redémarrer sa machine d'états et de se reconnecter au point d'accès Wi-Fi, cela implique un temps d'activité et donc une consommation importante en préambule à chaque émission. Cependant, laisser le module en veille en permanence consomme autour de $15mA$ ce qui est inenvisageable pour l'autonomie du système. Ceci explique le mode de fonctionnement choisi. Par ailleurs, la consommation est typiquement maximale lors des pics correspondant à l'émission de données (utilisation de l'antenne). Lors de communications radiofréquence, diminuer la fréquence revient à diminuer le *time on air* et donc la taille des trames échangées. Dans un souci de pédagogie, nous avons utilisé un protocole et des librairies permettant une prise en main simple de l'ESP8266. L'inconvénient est que les trames sont plus longues que nécessaire. Par exemple, chaque donnée est envoyées sous la forme d'une chaîne de caractère identifiant la donnée, puis de la donnée elle-même. La chaîne de caractère constitue la clé unique côté *ThingsBoard*. Passer d'appels de fonction du type

```
tb.sendTelemetryFloat("temperature_ciel", temp_ambiant_celsius_sky);
```

à

```
tb.sendTelemetryFloat("1", temp_ambiant_celsius_sky);
```

en choisissant une convention pour faire correspondre des numéros à chaque donnée, est un exemple de piste pour diminuer la taille des trames. Il serait encore plus efficace (mais complexe) de travailler sans la bibliothèque ThingsBoard, de définir ses propres trames, et de programmer directement l'ESP8266 pour plus de contrôle. Cette démarche est laissée aux utilisateurs futurs en accord avec la portée éducative du projet.

4.2.2 Optimisations sur le *hardware*

Dans la mesure du possible, les composants embarqués ne sont alimentés que lorsque qu'ils mesurent et communiquent avec le microcontrôleur. Nous contrôlons leur alimentation à l'aide de *GPIOs* du STM32F767ZI reliés à des transistors commandant les pins V_{dd}. C'est la manière la efficace pour maîtriser la consommation des capteurs ; cependant elle n'est possible que pour les capteurs qui n'ont pas de temps d'établissement et pour lesquels il ne faut pas appeler de fonction d'initialisation trop longue. C'est le cas pour tous les capteurs saufs un. En effet, le CCS811 doit être alimentés en permanence pour produire des données correctes. Néanmoins, nous le paramétrons dans son mode le plus économe en énergie (actualisation toutes les minutes) et nous baissons sa consommation avec un pin *WAKE* disponible sur le composant (nous renvoyons à sa fiche technique). Cependant, comme le BMP280 et le HDC1080 sont alimentés à partir de la même piste que le CCS811, la contrainte sur ce dernier nous force à alimenter les deux premiers en permanence. L'utilisation de composants indépendants permettrait de différencier leur alimentation. Malheureusement, la plupart des composants dont l'alimentation est contrôlée par transistors nécessitent un temps d'établissement pour produire une mesure cohérente. Il s'agit d'un temps où le microcontrôleur est forcé d'attendre *delay*) ce qui constitue une perte sèche de puissance. Nous avons tenté au mieux de minimiser cet état de fait, d'une part en utilisant les délais les plus courts possible, et d'autre part en réarrangeant l'ordre de lecture des capteurs pour que le microcontrôleur communique avec un capteur le temps qu'un autre se stabilise. Dans le cas du *VEML6070* et du *TCS34725*, il est possible de passer en mode économie d'énergie par une commande I2C, faisant descendre la consommation autour du micro-Ampère ce qui est négligeable. Afin de simplifier le tracé du PCB (moins de vias) et de libérer des *GPIOs*, c'est la solution retenue pour ces deux éléments. Enfin, l'alimentation au module GPS et à l'ESP8266 est coupée par transistors lorsqu'il ne sont pas utilisés. Ces informations sont résumées dans le tableau 3. **Une piste non explorée faute de temps est l'exploitation des modes de fonctionnement *low-power* du *BMP280* et du *HDC1080*, mentionnés dans les fiches techniques. Il faudrait pour cela compléter les bibliothèques utilisées ou en trouver d'autres.**

Certains composants comportent des LEDs allumées en permanence pour indiquer leur bon fonctionnement, ce qui est utile dans une démarche de prototypage et de développement, mais constitue une perte inutile de puissance lors de l'exploitation de la station. Les composants concernés sont le pluviomètre, l'ESP8266 et le module GPS. Nous avons fait le choix de laisser ces LEDs, le gain en courant nous semblant moins conséquent que l'intérêt de cette signalisation pour le prototypage et le *debug*.

Enfin, certains ponts de soudure ont été modifiés par rapport à la configuration d'usine de la carte Nucleo-144. Il s'agit ici encore de désactiver des composants inutiles au fonctionnement de la station. Afin de comprendre les implications de ces modifications, nous attirons l'attention sur le fait que le connecteur JP5 est connecté à l'alimentation isolée du microcontrôleur V_{dd} ; si on y branche un ampèremètre, on peut différencier la consommation de ce dernier de celle de la carte Nucleo-144 entière. De plus, si ce cavalier est enlevé, le microcontrôleur n'est plus alimenté, ainsi la consommation de la carte Nucleo-144 est uniquement

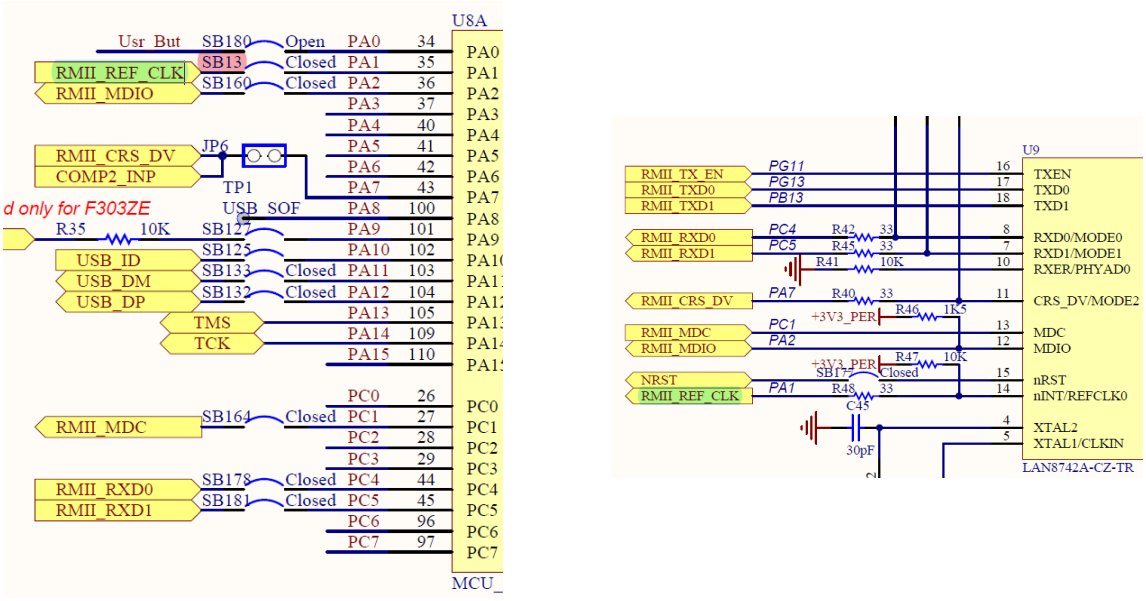
celle des composants périphériques.

Élément	État initial	État final	Gain	Perte
ST-Link sécable	Solidaire de la Nucleo-144	Détaché	$\approx 50mA$ économisés	Protocole de programmation/*debug* nécessitant plus d'efforts
SB13	ON	OFF	$\approx 50mA$ économisés	Déconnecte la connexion d'horloge entre le STM32F767ZI et le module Ethernet, ainsi rendu inutilisable
SB2	ON	OFF	$\approx 60mA$ économisés	Déconnecte l'alimentation en 3V3 de tous les périphériques (en particulier ST-Link et module Ethernet) ainsi que d'autres composants mineurs (LED1, etc.)
SB8	OFF	OFF	Extinction de HSE et sa déconnexion de PF0/PF1	Aucun gain observé, possiblement parce que la HSE doit être manuellement activée côté *software*
SB9	OFF	OFF		
SB112	ON	OFF		
SB148	OFF	ON		
SB149	ON	OFF		
SB163	ON	ON		

FIGURE 17 – Récapitulatif des modifications de la carte Nucleo-144 par rapport à la configuration d'usine

Avec ce cavalier retiré, la consommation de la carte Nucleo-144 est d'environ $160mA$. Conformément au manuel de la carte Nucleo-144, nous avons retiré le pont SB13 afin de

désactiver le module Ethernet, ce qui permet d'économiser environ $50mA$. Cependant, une étude poussée du circuit électrique de la carte Nucleo-144 révèle que ce module est en fait toujours alimenté même après cette modification. Comme on peut le voir en figure 18, le pont SB13 ne fait que briser la piste reliant l'horloge du STM32F767ZI au module Ethernet, ce qui, nous le présumons, le désactive en partie. Cependant (figure 19) il est toujours alimenté par la piste *3V3_PER*. Le problème est que la carte Nucleo-144 n'offre pas d'autre possibilité pour couper l'alimentation au module Ethernet que de couper SB2, ce qui déconnecte alors l'alimentation *3V3_PER* de toute la piste (figure 20). En d'autres termes, tous les périphériques alimentés en 3V3, c'est-à-dire le module Ethernet, le ST-Link, et d'autres composants mineurs tels que la *USER LED 1* ne sont plus alimentés. Cependant, le gain en consommation justifie ce choix. Le ST-Link est sséparé de la carte afin de pouvoir continuer à programmer le microcontrôleur, ce qui économise environ $50mA$, et le passage de SB2 en OFF économise les $60mA$ restant. Ainsi, la carte Nucleo proposée est configurée pour tirer le moins de courant possible tout en préservant ses fonctionnalités utiles. Les seuls éléments encore alimentés sont les convertisseurs de tension et les éléments reliés directement à V_{dd} : le microcontrôleur STM32F767ZI et quelques LEDs et boutons.



(a) Côté STM32 - SB13 est indiqué en rouge et la piste d'horloge en vert.

(b) Côté module Ethernet - On retrouve la piste d'horloge en vert.

FIGURE 18 – Influence de SB13 sur le circuit électrique de la carte Nucleo-144

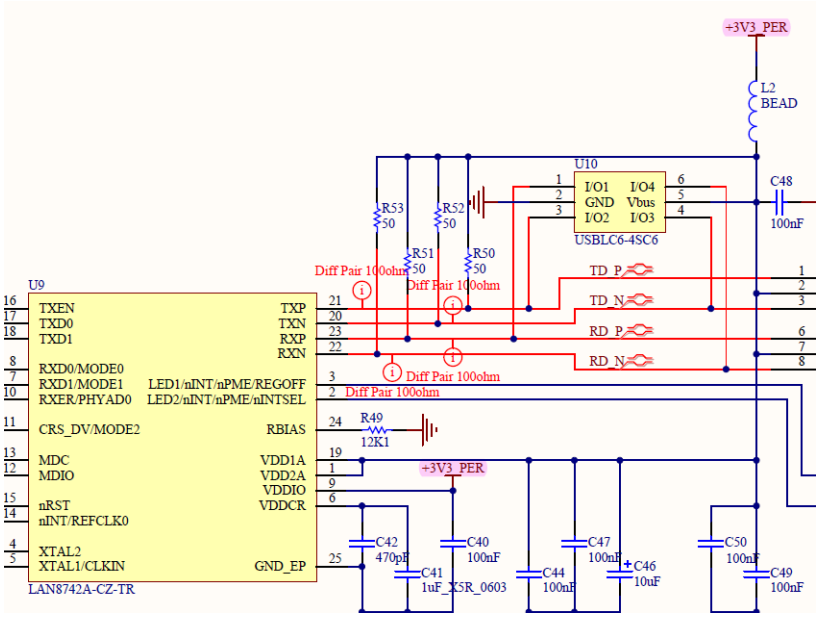


FIGURE 19 – Partie du circuit du module Ethernet. On voit la piste *3V3_PER* en rose.

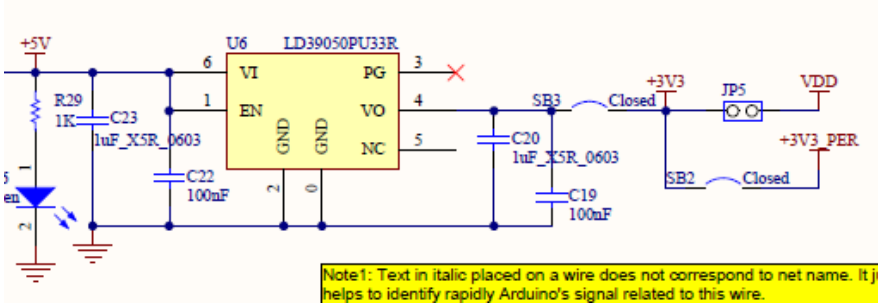


FIGURE 20 – Séparation de la sortie $3V3$ de l'abaisseur de tension entre V_{dd} pour le STM32 et *3V3_PER* pour les autres éléments

4.3 Bilan énergétique

L'étude de l'autonomie énergétique de la partie embarquée de notre système est fondée sur la comparaison de la puissance disponible et de la puissance consommée. L'utilisation de batteries nous permet d'adopter une approche basée sur l'étude des puissances moyennes dans le cas car celle-ci permet le stockage de l'énergie en surplus pour un usage ultérieur.

L'étude de la puissance disponible à notre système a été présentée précédemment. Cependant, l'étude de la consommation à l'aide de la carte *STM32L562E-DK Discovery kit* s'est avérée impossible de fait des trop fort courants mis en jeu au sein de notre système (phase de démarrage et de communication sans fil) vis-à-vis du courant limite délivrable par la carte. De plus, une tentative de mesure en temps réel du courant consommé à l'aide d'un montage de conversion courant-tension et d'un oscilloscope a été réalisée sans succès.

S'il avait été possible de mesurer la consommation du système, la méthodologie que nous aimerions avoir suivi est la suivante : la mesure du courant consommée par l'ensemble du dispositif embarquée est mesurée à l'aide d'un *STM32L562E-DK Discovery kit* branché en entrée du convertisseur boost (à la place de la batterie). Un cycle de fonctionnement complet, c'est-à-dire mesure de tous les capteurs suivie d'envoi au serveur puis de passage en mode STOP est mesuré. Ce n'est pas l'initialisation (*setup*) qui nous intéresse mais le fonctionnement nominal (*loop*).

La tension délivrée par la carte d'alimentation est de 3.3V, et le graphe de mesure donne l'intensité I tirée en fonction du temps. Ainsi, la puissance moyenne consommée en mode STOP est de

$$P_{moy,STOP} = 3.3 \times I_{moy,STOP}$$

et la puissance moyenne consommée pendant la phase d'activité (dont la durée est T_{mesure}) est

$$P_{moy,mesure} = 3.3 \times I_{moy,mesure}$$

dont le logiciel de mesure permet de calculer la valeur sur une fenêtre temporelle donnée. On rappelle que la puissance fournie par le panneau solaire est estimée à $P_{moy,disp} = 168.9mW$. Il en résulte que la plus petite période T_{STOP} de veille viable pour le système embarqué, c'est-à-dire telle que

$$\frac{T_{mesure} \times P_{moy,mesure} + T_{STOP} \times P_{moy,STOP}}{T_{mesure} + T_{STOP}} \leq P_{moy,disp}$$

se déduit des autres paramètres. On peut réécrire l'équation comme suit :

$$\frac{T_{mesure}(P_{moy,mesure} - P_{moy,disp})}{P_{moy,disp} - P_{moy,STOP}} \leq T_{STOP}$$

d'où on déduit une valeur minimale de T_{STOP}. Une fois T_{STOP} choisie, il suffit de modifier le code de la partie embarquée pour que le *deepSleep* dure T_{STOP} (l'argument de la fonction est un temps de millisecondes). Enfin, la donnée de T_{STOP} et de T_{mesure} qui elle est fixée par la vitesse de calcul du microcontrôleur donne la période totale $T = T_{STOP} + T_{mesure}$ qui définie la fréquence d'acquisition de la partie embarquée. On aurait également pu raisonner avec une marge de sécurité, par exemple de 10%, sur la puissance nécessaire $P_{moy,disp}$.

On peut cependant observer l'autonomie de notre système de manière qualitative. En effet, la phase de démarrage et d'initialisation est la plus consommatrice d'énergie car elle comporte la localisation de notre station à l'aide du module GPS très gourmant en énergie. Malgré cette consommation très fortement supérieure à celle attendue en moyenne on arrive tout de même à alimenter notre système durant cette période à l'aide du panneau solaire uniquement et sans batterie. Cela semble prometteur quant à l'autonomie du système, qui, à défaut d'être complète, peut raisonnablement être estimée à hauteur de plusieurs jours voire plusieurs semaines sans recharge de la batterie (acte de maintenance).

5 Circuits

Afin de donner forme à la station, il faut créer un PCB pour chaque circuit qui a été traité dans les parties précédentes. Pour cela dans un premier temps il est important de créer un *PinList* afin de faciliter le routage. Cela permet aussi de s'assurer qu'il n'y a pas de conflit *hardware* sur les microcontrôleurs.

5.1 *Shield* de la Nucleo-144

Des connecteurs ont été placés sur chaque PCB afin d'augmenter la modularité de la station, et de faciliter le dépannage. La figure 21 décrit les fonctionnalités de chaque connecteurs placés sur la carte.

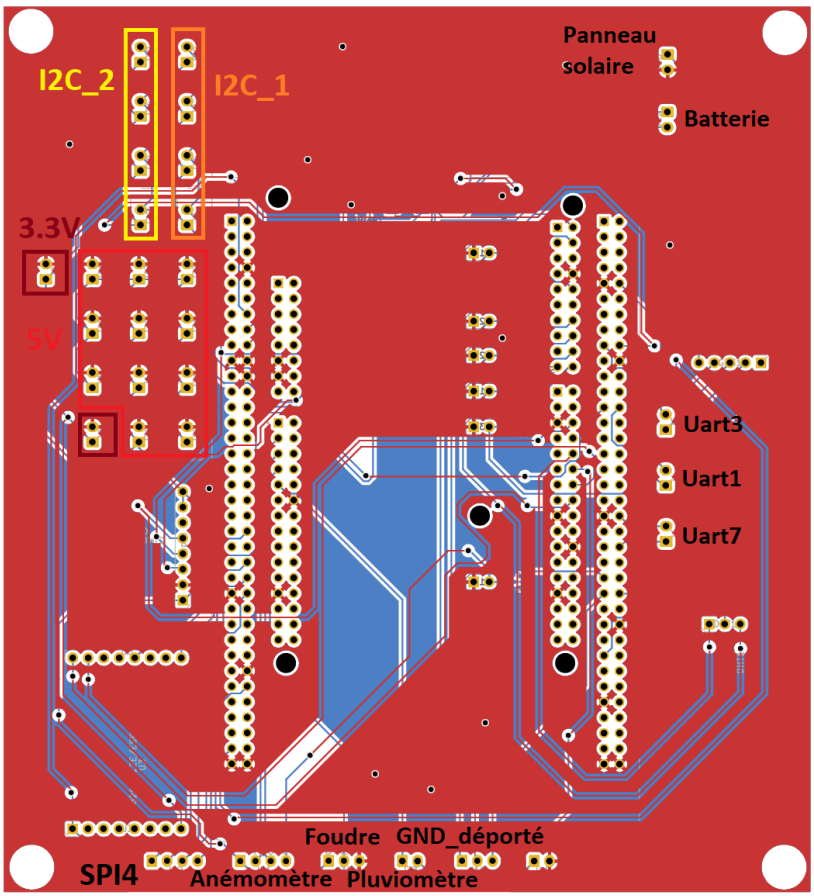


FIGURE 21 – Connecteurs JST du *shield* principal

Il est aussi possible d'utiliser les cavaliers au milieu du PCB afin d'outrepasser les transis-

tors d'activation des capteurs et de pouvoir tester le circuit sans avoir à se soucier de leurs activation ou non.

Pour ce qui est de la connexion entre le *Shield Nucléo* et le PCB déporté, se reporter à la figure 22. Si les connecteurs ne sont pas déjà sertis (utilisation des détrompeurs), il faut utiliser les fichiers KiCAD fournis afin de déterminer le sens de connexion entre chaque bornier.

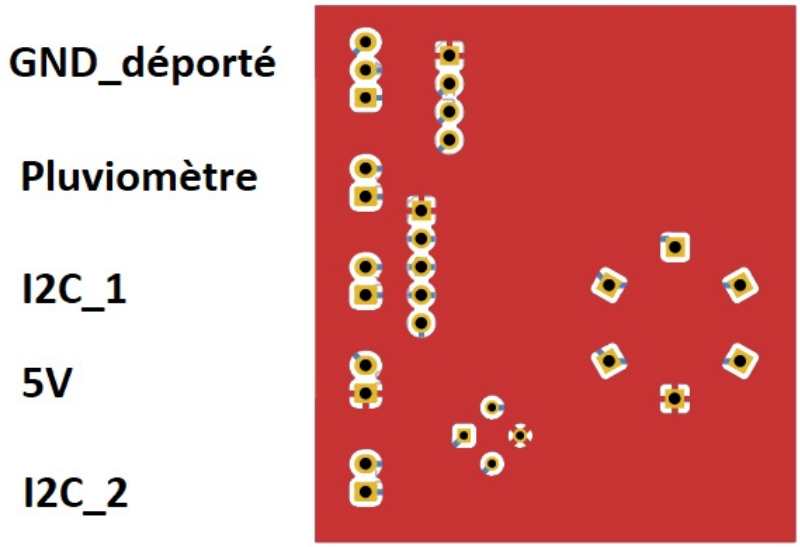


FIGURE 22 – Connecteurs JST du PCB déporté

Il est important de noter que les PCBs de test réalisé sont fonctionnels, mais ont nécessité un temps de *debug* non négligeable. Venant à la fois d'erreurs de *design* qui sont rectifiées dans les fichiers livrés, et aussi de la mauvaise qualité des vias dans le cadre de l'I2C (la ligne du *MLX90614* était tellement bruitée qu'il était impossible de faire fonctionner le protocole dessus). Les nombreux courts-circuits (cheveux de cuivres) à éliminer à la main après livraison des PCBs réalisés avec le matériel de l'école constituent également un point d'intérêt majeur.
La fabrication de ces PCBs par une entreprise spécialisée dans le domaine semble donc importante afin de limiter le temps passé à comprendre et rectifier les différents problèmes uniquement dûs uniquement au processus de fabrication des circuits.

5.2 Anémomètre ultrasonique

Cette partie se base sur cet article trouvé en ligne, le but est de donner une explication plus approfondie du fonctionnement du capteur (plus particulièrement du fonctionnement de la partie électronique). La description du fonctionnement général de ce capteur est disponible dans la partie 3.1.1.

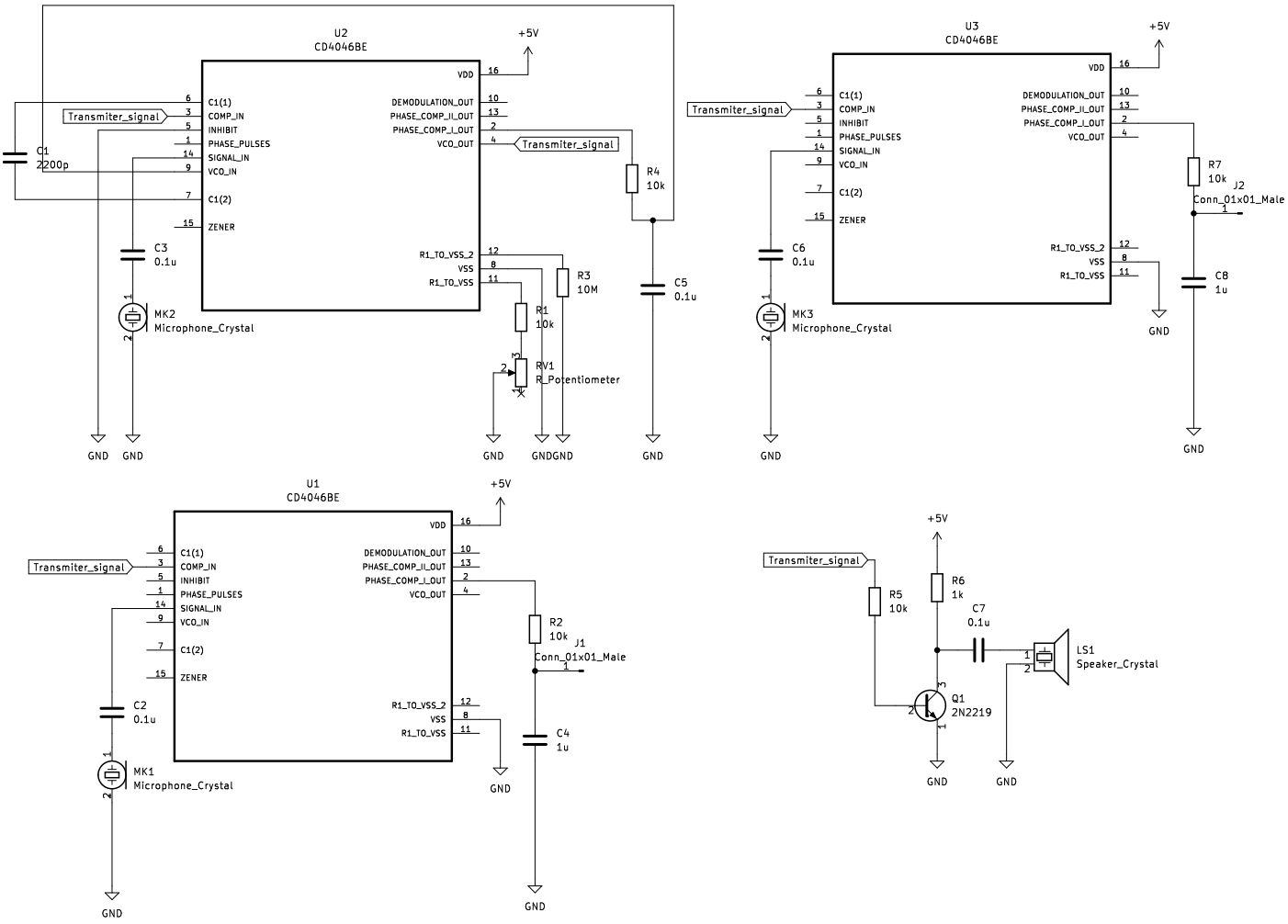


FIGURE 23 – Circuit de l'anémomètre

Dans un premier temps il est important de souligner que le prototype de l'anémomètre n'a pas été entièrement finalisé. Ainsi, il n'a donc pas été testé, cette partie à pour vocation de

donner une explication théorique du fonctionnement du capteur.
Afin de finir le capteur, il reste à finir le poteau, le canal d'air et à calibrer l'anémomètre dans le code fourni (toute la procédure est décrite dans l'article ...).

Le circuit fonctionne à l'aide de 3 circuits intégrés 4046 (plus exactement une PLL et 2 portes XOR).
Le premier 4046 est réglé pour fournir une PWM de 40kHz (la fréquence de raisonnance des récepteurs ultrasoniques). Il utilise la sortie du récepteur Nord en tant que référence avec un déphasage de zéro. Ensuite une porte XOR est utilisée pour comparer le signal du VCO (*voltage-controlled oscillator*) et les signaux Est et Ouest. Cela donne deux sorties dont le rapport cyclique est lié au déphasage des sortie Est et Ouest respectivement, on connecte ensuite ces signaux de sortie dans un filtre passe bas, coupant toutes les fréquences afin de ne garder que la composante continue. Il est possible d'extrapoler de celle-ci le rapport cyclique. De ce dernier, on peut calculer les tensions de sortie puis le déphasage des récepteur Est et Ouest.

Il est possible d'en déduire la vitesse et la direction du vent en utilisant le théorème de Pythagore et de la trigonométrie.