SMART DEVICES

Capteur de gaz connecté

Date :

29/12/2017

Auteurs :

Jonathan PLEURON <pleuron@etud.insa-toulouse.fr>
Killian TESSIER <tessier@etud.insa-toulouse.fr>

I. Introduction

Ce document a pour but de présenter le travail effectué durant le projet de conception d'un capteur de gaz intelligent et connecté.

Dans un premier temps, nous verrons le contexte général du projet avec son cahier des charges (philosophie du projet). L'architecture globale est décrite juste après, du capteur à la mise sur le réseau des informations en passant par toute la chaine de mesure et de traitement.

Ensuite pour chaque couche, la problématique est posée et la solution choisie présentée.

Enfin, la dernière partie conclue le document en présentant les problèmes rencontrés et en ouvrant sur des pistes d'amélioration.

Ce document fait guise à la fois de documentation sur l'état final du projet mais a pour but également de servir de base pour une éventuelle reprise du projet par des étudiants. Ainsi dans une optique open-source les documents relatifs au projet (code, documentation etc…) sont mis à disposition librement sur Github.

II. Contexte et Cahier des charges

Dans le cadre de l'émergence des objets connectés, nous avons travaillé sur le développement d'un capteur de gaz connecté. Il s'agit ici de concevoir un capteur dit intelligent (smart device) c'est-à-dire qui doit :

- Mesurer la grandeur physique souhaitée (mesurande) : concentration de gaz
- Prendre en compte les grandeurs d'influences
- S'auto-étalonner voire s'autodiagnostiquer
- Consommer peu d'énergie afin de pouvoir être autonome
- Transmettre les données via un réseau sans fil

Ce projet est réalisé dans un contexte open source. Le matériel (hardware) et les logiciels (software) utilisés doivent être si possible open-source.

III. Chaine de mesure

Un smart device peut être décomposé en plusieurs blocs physiques ou logiques présentés ci-dessous

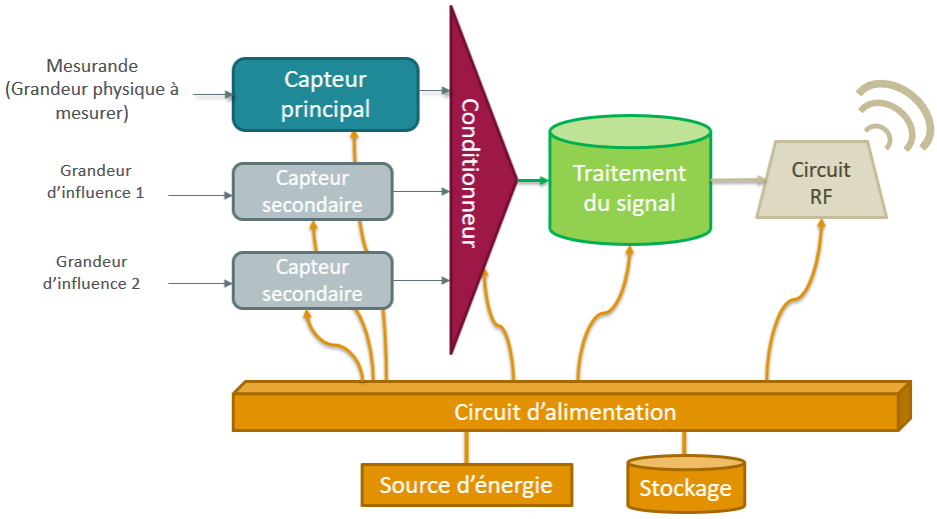

Fig 1 : Schéma bloc d'un smart device

Dans le cadre du projet le capteur utilisé est un capteur de gaz à base de nanoparticules de WO3 que nous avons réalisé en salle blanche (voir datasheet sur Github documentation/datasheets/GSWO3AIMe.pdf). Ce capteur mesure à la fois la concentration de gaz et la température (grandeur d'influence).

Le conditionnement du signal se fait de manière analogique et le traitement via une microcontrôleur ATmega328P (celui présent sur l'Arduino UNO).

La transmission Radio Fréquence (sans fil) se fait en utilisant la technologie LoRa (Low Range Wide Area) via un module Microship RN2483A qui connecte le capteur au réseau TTN (https://www.thethingsnetwork.org/).

Enfin le capteur est autonome et puise son énergie d'un accumulateur Li-ion qui est rechargé par un panneau solaire.

Le schéma global du capteur de gaz intelligent est présenté ci-dessous :

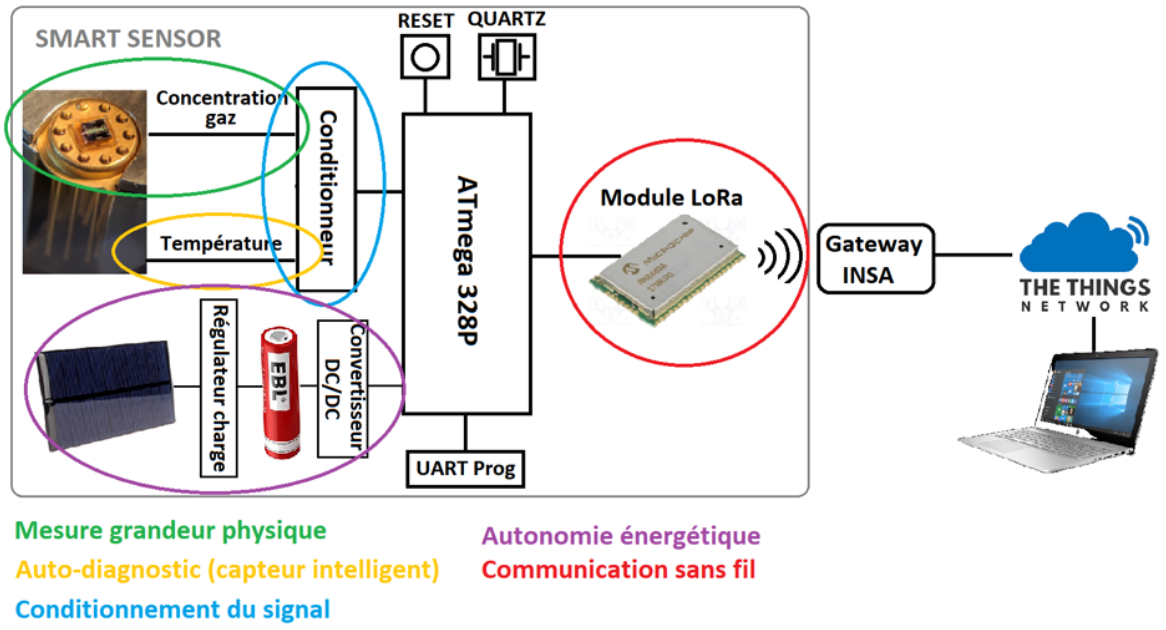

Fig 2 : Schéma bloc du capteur de gaz connecté

IV. Acquisition des grandeurs physiques et conditionnement du signal

L'acquisition des grandeurs physiques (concentration de gaz et température) est réalisée par un seul capteur. Ce-dernier peut être vu comme deux impédances résistives :

- R*sensor* = résistance proportionnelle à la concentration de gaz -> 10kΩ à 1GΩ
- R*temp* = résistance aluminium proportionnelle à la température du capteur
  -> 100-200Ω

Les caractéristiques courant/tension de ces impédances sont détaillées dans la datasheet du capteur. Les valeurs approximatives suffisent à concevoir le circuit de conditionnement du signal.

Une troisième impédance, purement résistive (piste en poly-silicone), est présente sur le capteur et sert à chauffer celui-ci pour le mettre à sa bonne température d'utilisation. La tension nécessaire pour faire chauffer cette résistance est de minimum 10V. C'est pourquoi un circuit de type booster de tension devra être ajouté au circuit qui repose sur une alimentation par accu de 3.7V.

Les deux résistances vont être converties en tension via des montages de type potentiométrique afin d'être reliées à deux entrées ADC (Analog to Digital Converter) du microcontrôleur. La résistance de température étant relativement faible elle sera juste insérée dans un pont diviseur avec une tension Vcc égale à la tension de référence de l'ADC. La résistance de mesure du gaz (Rsensor) elle est très grande et va poser des problèmes d'adaptation d'impédance si elle est directement connectée à l'ADC (Rmax = 10k Ω à l'entrée de l'ADC du ATmega328). C'est pourquoi on attaque plutôt un AOP (via un pont diviseur)

avec une impédance d'entrée tolérée bien plus grande et qui va amplifier la tension. On réalise un montage amplificateur non inverseur :

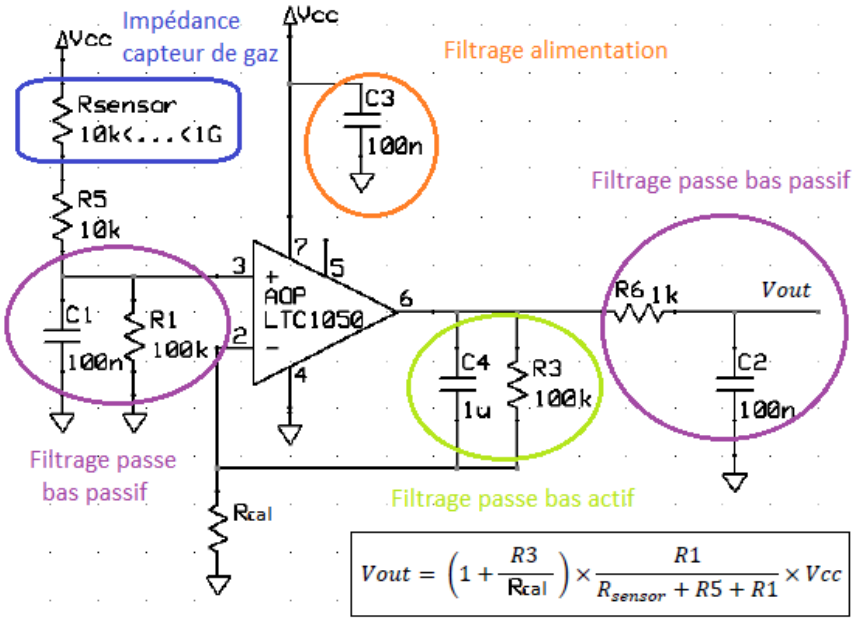

Fig 3 : Circuit de conditionnement pour la mesure de la concentration de gaz

On constate sur le schéma la présence de plusieurs filtres passe bas qui permettent notamment de couper le 50Hz très perturbant et d'éviter le repliement lors de l'échantillonnage.

Remarque : L'AOP utilisé doit être de type « low rail » (ex : LTC1050).

V. Traitement numérique (Microcontrôleur) et communication sans fil

Nous l'avons déjà évoqué, le traitement numérique se fait à l'aide d'un microcontrôleur ATmega328P, celui présent sur l'Arduino UNO. Attention à bien choisir le modèle avec le bootloader. Nous utilisons principalement deux fonctions du µC qui sont les convertisseurs analogiques numérique (CAN ou ADC) et les UARTs pour la liaison série avec le programmeur et pour la communication avec le module LoRa. Pour fonctionner le µC a aussi besoin d'un résonnateur et d'un bouton de reset.

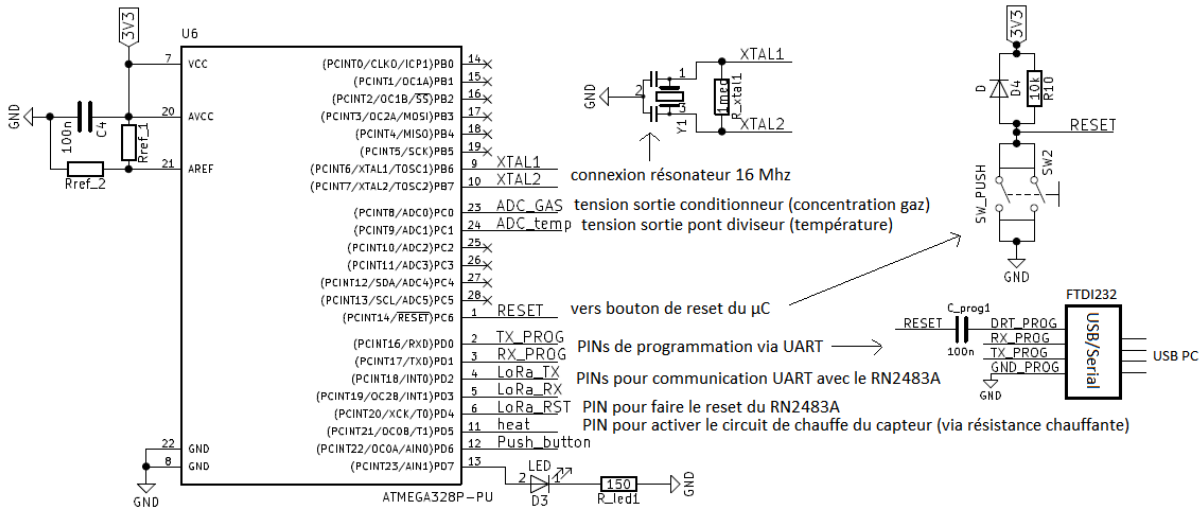

Fig 4 : Schéma électrique de la partie microcontrôleur

La programmation de l'ATmega peut se faire depuis l'IDE Arduino en utilisant un adaptateur USB/Série de type FTDI232. Les broches TX et RX du FTDI doivent être impérativement branchées aux pins D0 et D1 (UART physique) de l'ATMega et la broche DTR doit être relié au reset via un condensateur de 100pF.

La partie communication sans fil se fait à l'aide du module LoRa RN2483A de Microship qui inclue le circuit RF et un microcontrôleur gérant le stack réseau (LoRaWAN). La datasheet du module LoRa se trouve sur le Github (documentation/datasheets/Microship _RN2483A.pdf). Celui-ci communique via un port série à 57600 baud (par défaut) et se pilote via des commandes de trois types : mac, radio et système (détail des commandes dans l'Application Note sur Github documentation/datasheets/RN2483A_AN.pdf). Ces commandes sont en fait une suite de caractères envoyés sur le port UART (ex : `mac tx uncf 1 5` ,pour envoyer un '5' sur le port 1 sans confirmation).

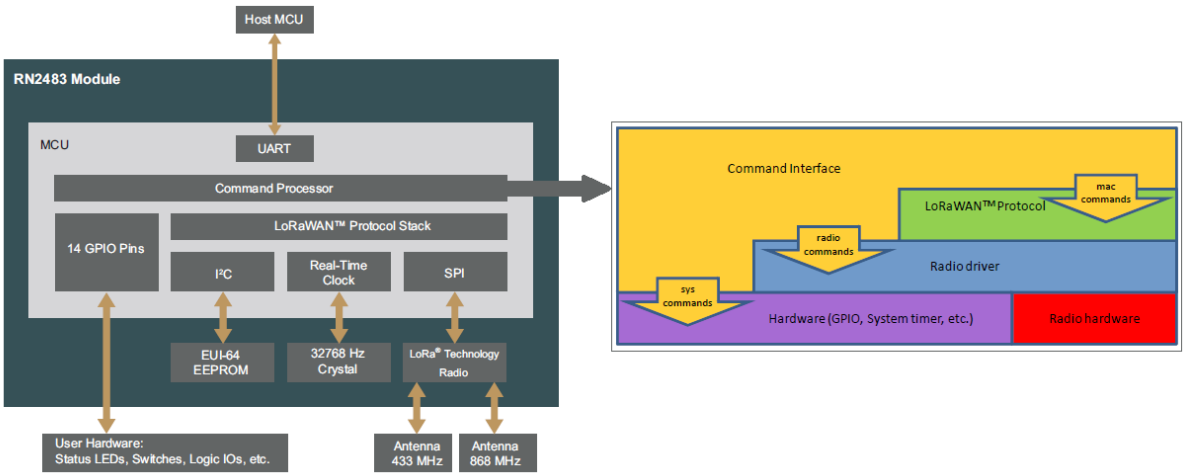

Fig 5 : Architecture logique du module LoRa RN2483A

Pour rappel la technologie LoRa utilise une architecture basée sur des devices émettant vers une gateway qui fait le relai vers un cloud (dans notre cas vers le réseau TheThingsNetwork). Afin de tester le module, on peut effectuer une communication pair-à-pair entre deux modules sans passer par la gateway. Pour cela on utilise directement les commandes de type radio. Pour effectuer ces tests et visualiser les réponses des modules on peut utiliser un terminal série comme Putty sous Windows ou Gtkterm sous Linux.

Coté émetteur

```
RN2483 1.0.3 Mar 22 2017 06:00:42
4294967245
ok
radio_tx_ok
```

Coté récepteur

```
RN2483 1.0.3 Mar 22 2017 06:00:42
4294967245
ok
radio_rx  48656C6C6F
```

Fig 6 : Envoi d'un « hello » (0x48656C6C6F) en Peer-To-Peer

Pour une communication avec le réseau TTN, nous passons par la gateway de l'INSA de Toulouse et utilisons la librairie TheThingsNetwork pour Arduino. Cette librairie masque la partie protocolaire du réseau LoRaWAN et permet de faire des envois vers TTN avec des commandes simples de type sendBytes().

Avant d'envoyer des commandes, pour lier le code tournant sur l'ATmega et le réseau TTN il faut passer par la création d'une application en s'enregistrant sur le réseau. Pour cela, il faut suivre des tutoriels sur le site de TTN. Une fois enregistré sur le réseau TTN, le code est lié à l'application via une adresse d'appareil (devAddr), une clé de session réseau (nwkSKey) et une clé de session applicative (appSKey). Ce triplet est valable pour une connexion de type ABP (Activation by personalization), la connexion en OTAA () est également possible (https://www.thethingsnetwork.org/forum/t/what-is-the-difference-between-otaa-and-abp-devices/2723 pour voir la différence entre les deux modes de connexion).

Le code source arduino se trouve sur le Github sous /Arduino.

VI. Traitement logiciel et réseau

Pour récupérer les données du capteur présentes sur TTN, nous avons choisi d'utiliser java avec MQQT, une des APIs conseillées par TTN pour récupérer nos données, et qui est très utilisée dans le domaine de l'IoT. MQTT est un protocole M2M très léger qui fonctionne à partir de publication et de souscriptions à des topics.

Pour cela nous avons utilisé la librairie fournie par TTN qui fournit un client qui implémente la librairie paho mqtt. Avec elle, on peut facilement se connecter à notre application TTN, à l'aide de notre AppID et de notre Access Key. Ainsi on peut initier une connexion à notre TTN et définir les actions à mettre en place lors de la réception de messages sur nos topics, l'activation, la déconnection, etc.

Les payloads reçus sont sous forme d'objets Json, facilement découplables à l'aide la librairie de manipulation de Json pour java : Jackson. Le capteur envoie à TTN 4 octets de données correspondant aux valeurs des ADC pour la mesure de gaz et de température. Une fois sur TTN la payload est formatée en JSON et on peut la décoder :

```
function Decoder(bytes, port) {

  var decoded = {};
  var length=bytes.length
  var gas_sensor={};
  var temperature={};

  /**** Frame format (4 Bytes):
   *         B0       |       B1        |          B2       |         B3
   *  gas_sensor MSB | gas_sensor LSB | temperature MSB | temperature LSB
   */
```

Fig 7 : Format d'un payload sur TTN

La payload est décodée (reconstruction des données) puis validée (elimination des valeurs erronées). Cela se fait via les fonctions decoder() et validator() de TTN (voir github dans /TTN).

Nous avons créé une simple application qui se connecte à notre application TTN et qui affiche de façon lisible les informations récoltées par le capteur en récupérant les valeurs depuis le Json. À partir de là, on peut facilement étendre notre application, stocker les données reçues dans des objets java pour y effectuer des traitements, des statistiques et y ajouter une IHM.

```
String region = "eu";
String appId = "iot_gaz_sensor";
String accessKey = "ttn-account-v2.TzRdcT4kec91CdT2rC035chPC5xlzyVSzo6yzsSwdjQ";

Client client =  new Client(region, appId, accessKey);
client.start();
```

Fig 8 : Code de connexion à l'application TTN

Le code complet est disponible sur Github dans /JAVA.

Remarque importante :

Pour ce projet les données recueillies par les capteurs, à savoir les valeurs numériques sur 10 bits correspondantes aux valeurs analogiques des capteurs de température et gaz, sont envoyées directement sur TTN. Cela limite le calcul au niveau du device et donc la consommation d'énergie mais celui-ci n'est plus réellement « intelligent ». Il ne fait aucun traitement. Pour la suite il faudra que ces valeurs soit traitées (conversions valeur numérique->concentration gaz/température, élimination des valeurs aberrantes, effet de la température sur le calcul de la concentration gaz, …). Cette partie n'a pas été faite par manque de temps.

VII. Consommation et Source d'énergie

Dans le but de rendre notre appareil autonome en énergie, nous avons décidé de le mettre sur batterie. Le dispositif est alimenté par une accumulateur 3.7V Li-ion qui offre une grande autonomie avec une capacité de 3000mAh. Nous utilisons un modèle de chez EBL qui a pour avantage d'être protégé contre les courts-circuits.

Les batteries Li-ion doivent suivre un cycle particulier de charge, c'est pourquoi il faut utiliser des circuits de charge dédiés comme le TP4056 présent sur le module XCSOURCE.

L'énergie provient d'un panneau solaire 5-6V de 1W.

Enfin la tension nécessaire à chauffer le capteur est générée par un convertisseur DC-DC (WINGONEER XL6009) de type boost qui peut produire une tension entre jusqu'à 30V.

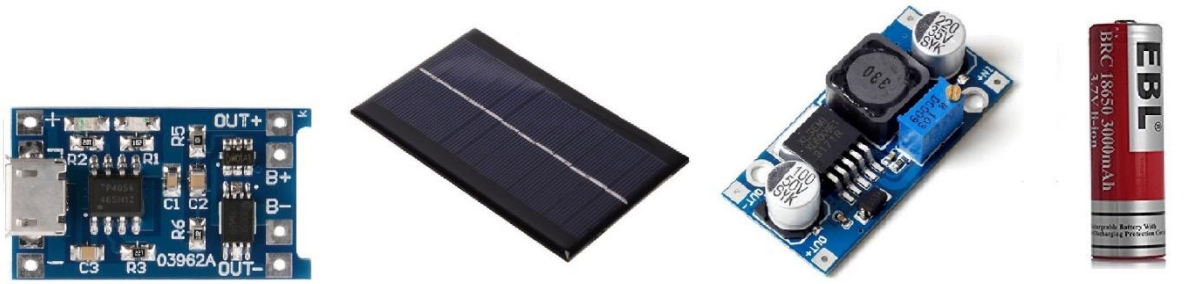

Fig 9 : Composants et kits utilisés pour alimenter le capteur connecté

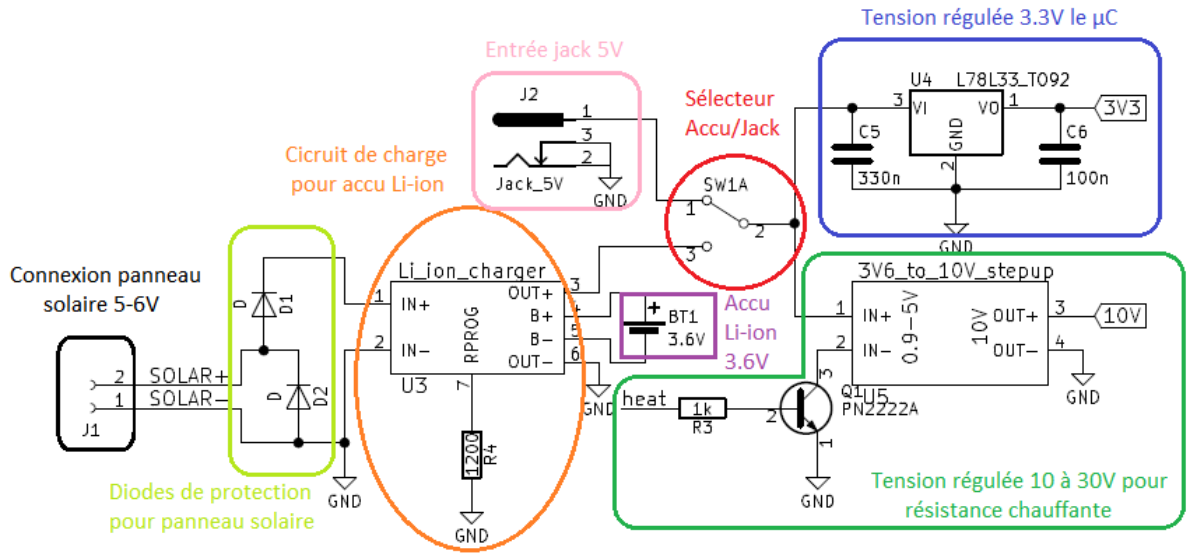

Fig 10 : Circuit électronique d'alimentation

VIII. Réalisation du PCB

Pour ce projet nous avons réalisé un PCB sur le logiciel libre Kicad. Une présentation (PowerPoint) du PCB ainsi que les fichiers sources sont disponibles sur le Github (/PCB).

IX. Problèmes rencontrés et améliorations possibles

Durant ce projet nous avons rencontrés quelques problèmes pouvant être résolus.

- La connexion en mode OTAA n'a pas fonctionné.
- Le compteur sur TTN doit être réinitialisé à chaque fois sinon les données ne sont pas reçues (comportement un peu aléatoire). Il faut identifier la source du problème ou faire en sorte d'automatiser la remise à zéro.
- Le panneau solaire semble délivrer trop peu de courant. Envisager un circuit d'adaptation/contrôle d'impédance ou prendre un panneau plus puissant.
- Le PCB peut être amélioré. En utilisant des composants directement à la place des kits la taille peut être réduite. Le module RN2483A peut être également souder directement sans passer par le shield.

Quelques autres améliorations peuvent être effectuées :

- Les PINs choisis peuvent être modifiés notamment en changeant les pin 2 et 3 qui servent aux interruptions de l'ATmega.
- Le code Arduino peut être amélioré en ajoutant des systèmes de détection d'erreur ou d'exceptions. Egalement, dans une logique de programmation orientée objet, les méthodes utilisées dans le loop() peuvent être déplacées dans des classes C++.
- On peut rajouter une IHM à notre application, avec des graphiques par exemple pour suivre l'évolution des données récoltées au cours du temps.