Rapport de stage

Implémentation communication radio sur système embarqué

tuteur en entreprise : Patrice MEDINA

Brice BARET

tuteur enseignant : Laurent FERAL

Entreprise :LAERO - Observatoire Pic du Midi - CNRS

Raphaël LOUIS LE DENMAT

BUT2 GEII ESE 2023-2024

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce stage.

Tout d'abord, je remercie chaleureusement, Monsieur Patrice MEDINA ingénieur électronicien, mon maître de stage, pour son accueil, ses conseils avisés et son encadrement tout au long de cette expérience. Sa disponibilité et son expertise m'ont été d'une aide précieuse.

Je voudrais également adresser mes remerciements sincères à Monsieur Brice BARET chercheur à l'OMP et initiateur du projet MicroMEGA, mon second maître de stage, pour son soutien et son encadrement complémentaire.

Je souhaite exprimer une reconnaissance particulière à Monsieur Giles ATHIET, ingénieur informatique à l'OMP, pour son aide apportée durant la conception de l'interface homme-machine (IHM). Son expertise ont été d'une grande valeur et ont significativement enrichi mon travail.

Je souhaite remercier également Monsieur Laurent FERAL, mon professeur référent, pour m'avoir soutenu et ma permis d'obtenir ce stage.

Sommaire

Table des matières

Introduction

Dans le cadre de mon cursus en Génie Électrique et Informatique Industrielle (GEII) à l'Université Toulouse III - Paul Sabatier, j'ai eu l'opportunité d'effectuer un stage au sein de l'Observatoire Midi-Pyrénées (OMP), un laboratoire de recherche environnemental. Ce stage s'inscrit parfaitement dans mon parcours académique et mes aspirations professionnelles, car j'ai toujours eu un intérêt marqué pour les sciences, en particulier celles liées à l'environnement.

L'OMP, reconnu pour ses recherches de pointe en aérologie et en études atmosphériques, m'a semblé être un cadre idéal pour mettre en pratique mes connaissances et développer de nouvelles compétences. L'intégration de ce laboratoire m'a offert la chance de travailler sur un projet innovant et aligné avec mes ambitions professionnelles dans le domaine de l'électronique et de l'informatique embarquée.

Le thème de mon stage portait sur la réalisation d'une communication entre un système embarqué (Micro MEGA), monté sur un drone, et un ordinateur. L'objectif était de récupérer les données de mesurer de pollution atmosphérique et les lire en temps réel au sol. Ce projet, à la croisée des chemins entre technologie de pointe et enjeux environnementaux, m'a permis d'explorer en profondeur les défis techniques et les solutions innovantes nécessaires pour mener à bien une telle mission.

Le présent rapport s'articule autour des grandes étapes de mon stage et de mes réalisations. La première partie présente l'Observatoire Midi-Pyrénées et le service dans lequel j'ai évolué. La seconde partie est consacrée à la description du projet MicroMEGA, les outils et moyens utilisés, ainsi que les différentes phases de la réalisation du projet. Enfin la troisième partie met en lumière les activités effectuées et les résultats obtenus, notamment en matière d'affichage de l'information.

Partie 1. Présentation de l'entreprise et du service dans lequel vous avez évolué

1. Observatoire Midi-Pyrénées

L'Observatoire Midi-Pyrénées (OMP) fait partie de l'Université Paul Sabatier (UPS, Toulouse III), et regroupe les huit laboratoires des Sciences de l'Univers de cette université ainsi qu'une Unité Mixte de Service (UMS).

L'OMP est aussi un Observatoire des Sciences de l'Univers (OSU) du Centre National de la Recherche Scientifique/Institut National des Sciences de l'Univers (CNRS/INSU) qui a sous sa responsabilité les services d'observation et tâches de services, ainsi que les services scientifiques communs.

L'OMP s'est installé sur plusieurs sites géographiques différents, dans le complexe scientifique de Rangueil à Toulouse, ainsi qu'à Tarbes, à Lannemezan-Campistrous et au Pic du Midi de Bigorre. Environ 600 permanents sont rattachés à l'OMP, dont une moitié est représentée par des chercheurs et l'autre par des ingénieurs, techniciens et personnels administratifs. Outre les deux tutelles principales que sont l'Université et le CNRS, certains de ses laboratoires peuvent également fonctionner sous la tutelle du Centre National d'Etudes Spatiales (CNES) et/ou de l'Institut de Recherche pour le Développement (IRD) selon les projets.

Figure 1: Logo différent laboratoire OMP

2. Laboratoire d'Aérologie

L'activité scientifique du Laboratoire d'Aérologie a pour objectif l'observation, la modélisation et la compréhension des processus dynamiques et physico-chimiques qui gouvernent l'évolution de l'atmosphère et de l'océan côtier.

Le Laboratoire d'Aérologie est réparti sur deux sites :

- site principal à l'Observatoire Midi-Pyrénées à Toulouse,
- site de Lannemezan (Centre de Recherches Atmosphériques de Campistrous) où se trouvent des instruments de mesure atmosphérique (stations sol, tour instrumentée, profileurs de vent, instrumentation pour l'électricité atmosphérique et analyseurs de gaz en trace).

Les axes de recherche :

- Les processus dynamiques, thermodynamiques et microphysiques : à petite échelle (convection nuageuse ou couche limite) à plus grande échelle (perturbations aux latitudes moyennes et tropicales).
- La physico-chimie de la troposphère et de la basse stratosphère : (production, transport, flux et impact d'espèces chimique et particulaires).
- L'océanographie côtière (dynamique, biogéochimie et changement climatique).

Le laboratoire compte 67 permanents (34 chercheurs et 28 Ingénieurs, techniciens et personnels administratifs) et 36 non permanent

Partie 2. MicroMEGA le renifleur de pollution

1. Présentation du projet

Mon maître de Stage Brice BARRET, chercheur à LAERO, étudie les nuages de pollution, notamment ceux des cheminées de Fairbank en Alaska.

Lors de la campagne d'hiver 2022, une équipe de recherche avait embarqué sur le même ballon que le système MicroMEGA, un système de communication permettant de connaître en temps réel la quantité de monoxyde de carbone, ceci permettant de détecter de nuit, si le ballon était dans un nuage de pollution.

La prochaine campagne, en janvier 2025, se faisant sans l'équipement de cette équipe, il est important que le système MicroMEGA soit équipé de cette fonctionnalité.

C'est de là que part le postula de mon stage : **mettre en place une communication en temps réel entre le système embarqué MicroMEGA et un ordinateur.**

Figure 2: Campagne de mesure a Fairbank

Le cahier des charges est le suivant :

- Le système doit acquérir les données suivantes : la température, la pression, l'humidité, la localisation GPS (3D) et les données des capteurs Alphasense.
- Le temps d'acquisition, traitement et enregistrement des données doit être d'une seconde.
- Les données doivent être enregistrées sur une carte SD.
- Les données doivent pouvoir être lues en temps réel sur un ordinateur à une distance minimale de 1Km en milieu ouvert.

Une première modification avait été faite entre la campagne de 2022 et le début du stage. Malheureusement celle-ci comportait des problèmes : le temps d'acquisition des données n'était pas contrôlé et la distance de communication n'était pas supérieure à 100m en milieu urbain. La première partie de mon stage s'est donc concentrée sur l'optimisation et la mise en place d'un nouveau module de communication.

2. Présentation du matériel et des outils utilisés

Système MicroMEGA

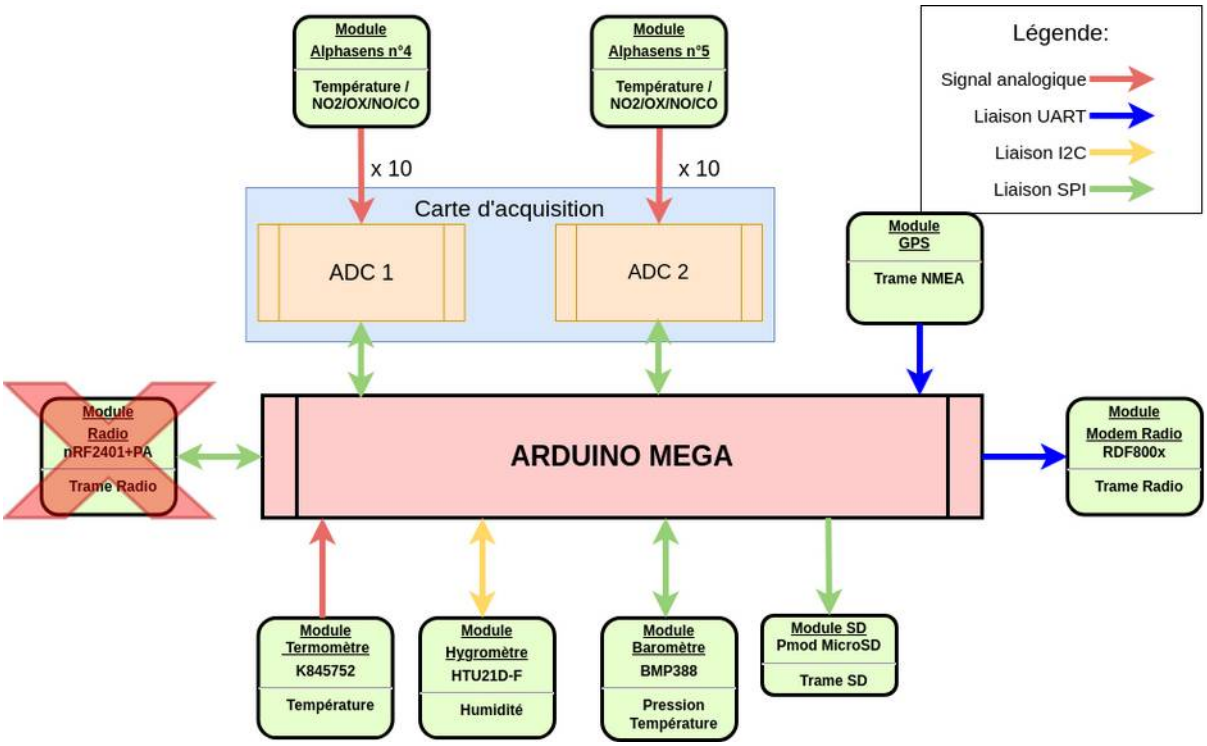


Figure 3: Diagramme de communication du système MicroMEGA

Le système MicroMEGA est un système d'acquisition de données aérologiques. Il est aussi appeler « Renifleur de Pollution » car il a pour but premier d'analyser les nuages de pollution de Fairbank en Alaska. Pour atteindre les nuages, il doit être embarqué sur un ballon ou un drone.

Le système microMEGA est composé de divers modules pour l'acquisition, l'enregistrement et la transmission de mesures physiques. Chaque module communique avec une Arduino MEGA (cf : Figure 3). Le module Radio nRF2401+PA était le module de communication d'origine, celui ci est toujours connecté à l'Arduino mais n'est plus utilisé.

Arduino UNO

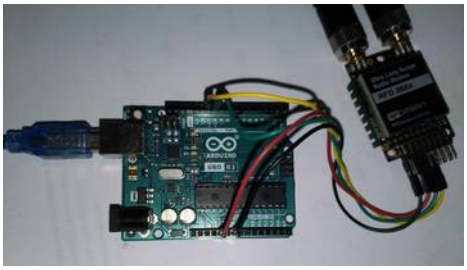


Figure 4: Arduino Uno avec modem RFD800x

On m'a confié en plus une Arduino UNO qui permet de faire la récupération des données sur l'ordinateur. Celle-ci est branchée au pc et a un module radio. Le module radio a été modifié au cours du stage pour équiper l'Arduino d'un modem le RFD800x

Outils de soudures :

J'ai eu des problèmes de soudures qui ont laché. J'ai donc dû utiliser du matériel pour refaire ces soudures pour le bon fonctionnement du système.

Analyseur de signaux :

Un analyseur de signaux en électronique est un instrument de mesure utilisé pour examiner et analyser les caractéristiques d'un signal électrique. Cet appareil est essentiel dans le diagnostic, le développement et la maintenance de systèmes électroniques.

Je m'en suis servis pour analyser les signaux de communication entre les Arduino et les modems radios.

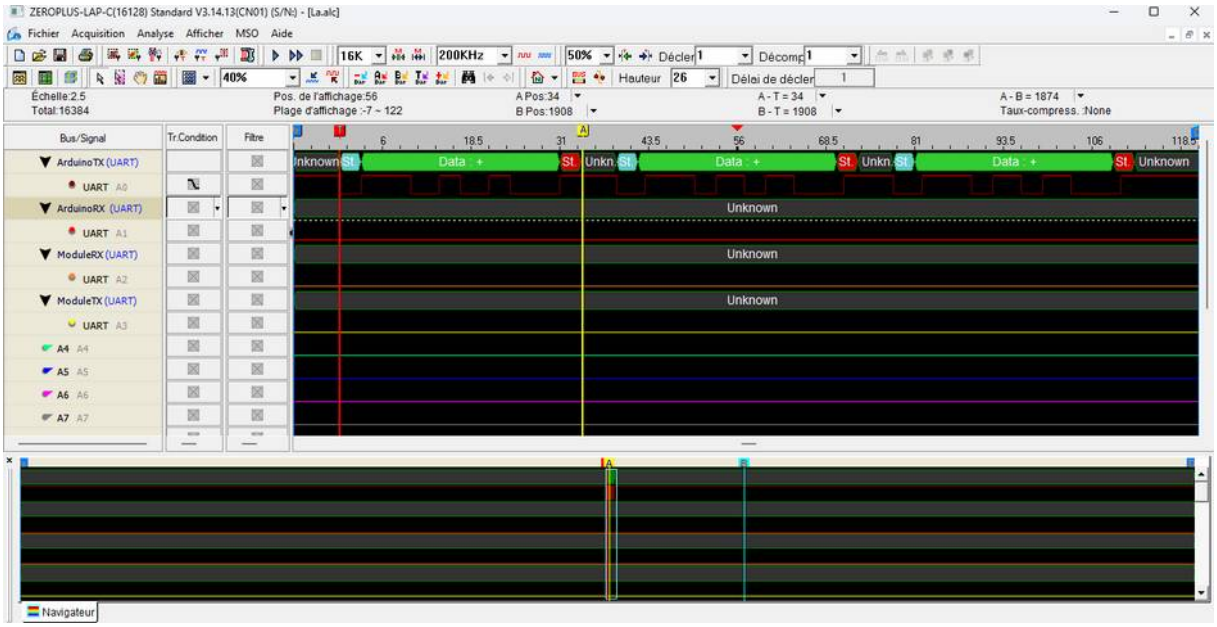


Figure 5: Interface PC analyser de signaux ZeroPLUS

3. Réalisation du projet

A) Première analyse du programme

Pour débuter mon stage j'ai commencé par lire et comprendre le programme de mon prédécesseur, celui-ci était « fonctionnel » c'est-à-dire qu'il pouvait acquérir les données, les enregistrer et les transmettre à une autre Arduino. J'en ai fait l'algorithme par bloc de fonctionnalité (Annexe Note pour algorithme programme initial MicroMEGA). C'est après cette première lecture que j'ai décelé trois problèmes à régler en priorité :

1 - Une utilisation massive de fonctions gourmandes en temps (millis() et Serial.print()) ;

2 - La gestion du temps pour effectuer une boucle n'était pas prise en compte : dès que la boucle était finie, le programme relançait directement une série de mesure ;

3 - Le programme n'était pas structuré : il n'y avait pas de fonction en dehors de setup() loop() et d'autres fonctions de setup de capteurs.

```
LOOP time : 1510ms
SetupRadio time : 12ms
Temperature time : 58ms
Humidity time : 51ms
Barometre time : 4ms
GPS time : 829ms
GPS TrameTrouve time : 1ms
GPS build TrameGGA time : 0ms
Split GPS time : 2ms
Acquisition_ADC1_ADC2 time : 3ms
Radio Emission All Data time : 480ms
Radio Cpt Arduino : 96ms
Radio ADC : 192ms
Radio GpS : 192ms
Sauvgarde SD time : 43ms
Programme time : 1027ms
```

Figure 6: temps différente étape du programme sans modification après une boucle

```
LOOP time : 719ms
SetupRadio time : 11ms
Temperature time : 60ms
Humidity time : 50ms
Barometre time : 4ms
GPS time : 42ms
GPS TrameTrouve time : 1ms
GPS build TrameGGA time : 0ms
Split GPS time : 2ms
Acquisition_ADC1_ADC2 time : 4ms
Radio Emission All Data time : 480ms
Radio Cpt Arduino : 96ms
Radio ADC : 192ms
Radio GpS : 192ms
Sauvgarde SD time : 40ms
Programme time : 273ms
```

Figure 7: temps différente étape du programme sans modification après une boucle

Comme on peut le voir sur les Figures 6 et 7 sur deux boucles différentes, on avait un temps d'exécution différent (voir valeur « LOOP time »), alors que l'on veut la même fréquence d'une seconde.

B) Modification du programme main

J'ai donc commencé par supprimer les fonctions superflues et créé une variable de débug pour commander l'affichage sur le sérial, ces modifications faisant gagner près de 300 ms. J'ai ensuite rangé chaque bloc fonctionnel dans des fichiers (cf : Figure 8) au nom du composant ou utilité de la fonction. Cette lisibilité de lecture gagnée, j'ai entamé la modification du programme.

Pour régler le problème de la gestion du temps, il y avait deux choix : soit utiliser le timer de l'Arduino ou utiliser le timer du GPS.

Le timer du GPS est le plus précis, mais le problème c'est qu'il fonctionne uniquement si on reçoit un signal GPS.

Dans l'hypothèse où le GPS ne capte pas de satellite, il est préférable de quand même faire les autres mesures et de récupérer les données GPS du drone.

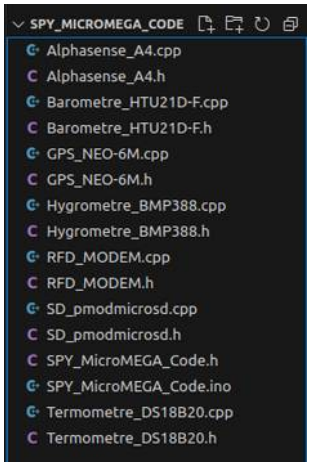


Figure 8: fichiers programme MicroMEGA (final)

La solution retenue est d'utiliser l'horloge du GPS, mais de sécuriser la boucle d'acquisition à l'aide d'une gestion du temps par la fonction millis() dans le cas où on ne recevrait pas de signal satellite.

Une autre solution a été imaginée : au lieu d'utiliser la fonction millis() c'était d'utiliser un timer harware. Malheureusement la difficulté de remettre en phase l'arrivée des informations du GPS et le timer m'a fait rebrousser chemin.

J'ai donc conçu l'algorithme en Figure 9 (ou annexe 1), les fonctions de lecture trame et trame GGA seront aborder en partie F .

En Orange, on a toutes les parties pour la récupération et la gestion de la trame GPS qui se fait toutes les secondes.

En Cyan on a la gestion du temps en cas de mauvaise réception GPS.

En Rouge c'est la partie du programme principal d'acquisition, enregistrement et transmission des données.

On peut noter que l'acquisition des données GPS se fait en dehors de la zone principale du programme. C'est parce que le module GPS nous envoie ses données sans qu'on l'interroge. C'est pour ça que la fonction de lecture est dans la partie qui boucle théoriquement le plus rapidement.

Cela a aussi permis de ne pas faire une fonction non bloquante comme c'était le cas à la base.

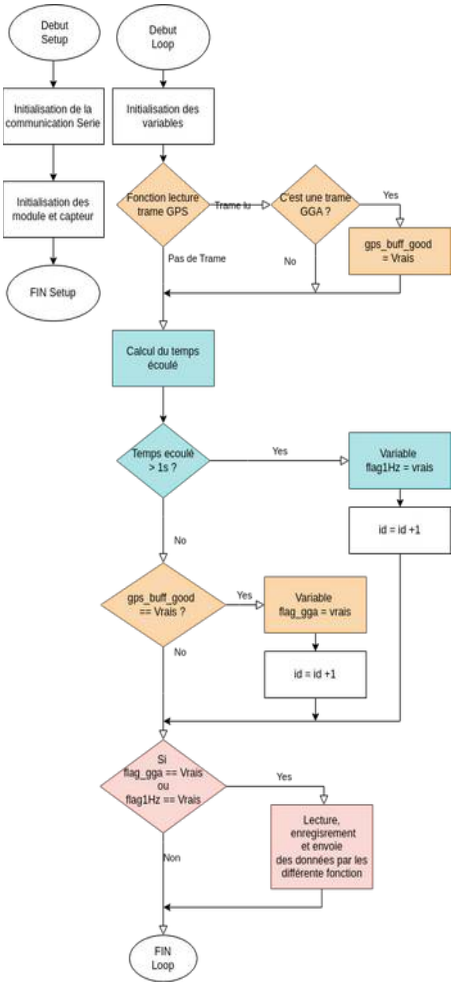


Figure 9: Algorithme Gestion du temps SYP_MicroMEGA

C) Modules Radio Communication :

Module NRF24L01

Dans un premier temps j'ai repris le module radio existant étant donné qu'il était vendu pour +1000 mètres en milieu ouvert.

C'est le NRF24L01, on communique en SPI (pin CE=7 CSN=8) à l'aide de la bibliothèque RF24.

Le programme de base envoye cinq trames de données. Etant donné que l'envoie d'une trame prenait 92 ms, il était impératif de réduire le nombre d'émission. Dans la documentation du produit, on peut lire qu'il envoie les données par paquet de 32 octets.

Pour gagner en place, on a limité les données que l'on a envoyé à : un indice de trame, la quantité du signal GPS, la température, la pression, l'altitude GPS et les valeurs des capteurs de gaz (à l'exception de ch0 la température du capteur Alphasence). L'indice et la qualité GPS sont mis sur le même octet pour gagner en place.

J'ai ainsi réussi à réduire le nombre de trame à deux (cf Figure 10), faisant gagner 250ms.

Trame 1																														
id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nom	I/Q	Température							Pression							Altitude GPS						ADC 1 gaz 1				ADC 1 gaz 2						

Trame 2																														
id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
nom	I/Q	ADC 1 gaz 3				ADC 1 gaz 4				ADC 2 gaz 1				ADC 2 gaz 2				ADC 2 gaz 3				ADC 2 gaz 4										

Figure 10: Positionnement des éléments dans les trames radio 1 et 2 (module NRF24L01)

Dans le programme de base, le module était initialisé de manière à avoir le maximum de portée : output 0 dBm, sensitivité 250kbps et on y avait rajouté un condensateur entre la masse et le VCC pour limiter les baisses de tensions durant la transmission de données.

- Transmitter
  - Programmable output power: 0, -6, -12 or -18dBm
  - 11.3mA at 0dBm output power
- Receiver
  - Fast AGC for improved dynamic range
  - Integrated channel filters
  - 13.5mA at 2Mbps
  - -82dBm sensitivity at 2Mbps
  - -85dBm sensitivity at 1Mbps
  - -94dBm sensitivity at 250kbps

Figure 11: Page 8 de la documentation nRF24L01Plus v1.0

$$P_r \leqslant P_e + G_e - 20\log\left(\frac{4\pi \cdot d \cdot f}{c}\right) + G_r \Leftrightarrow d \leqslant \frac{10^{\frac{P_e+G_e+G_r-P_r}{20}}}{4\pi \cdot f} \cdot c$$

Figure 12: Équation des télécommunications

A l'aide des données sur la figure 11, P_r = -94, P_e = 0, f = 2,4 Ghz, c = 300 000 km/s de l'équation en figure 12 et en prenant G_e et G_r égal a 2 dbi : on calcule une distance de réception inférieure ou égale à 758 m

Les résultats étant différents de ceux qu'annoncés par le constructeur (>1000m), j'ai préféré vérifier sur le terrain la distance en milieu urbain dégagé. La distance maximale mesurée lors de ce test était de 150m. Par rapport à ces informations et sur les conseils de Patrice MEDINA, il a été décidé d'acheter un modem fonctionnant sur une fréquence ISM à 868 Mhz.

Choix du Modem

Pour le choix du modem en plus de la distance, il fallait qu'il existe en version 915Mhz qui est la fréquence utilisable aux Etats Unis (contrairement en France où elle est interdite). La fréquence ISM 868 MHz étant limité en temps et puissance de transmission, selon l'ARCEP [1] les fréquences autour de 868MHz sont limitées à 10mW et 1 % de coefficient d'utilisation[2]

Figure 13: Modem transmission avec antenne en polarisation diverse

Le modem retenue fût le RFD800x et RFD900x (cf : figure 13) de chez RF DESIGN, en prenant en compte que l'on doit envoyer 53 caractères plus 4 caractères pour des bit de début et de fin de chaîne et que l'on soit limité à 1 % du temps de transmission, il faudra transmettre au minimum 57*100*8= 45 600 bit/s.

Ne connaissant pas le type de modulation du modem, j'ai pris en compte un facteur quatre pour envoyer les données soit 185 kbit/s, dans la documentation[3] on nous dit qu'il y a une sensitivité de -98 dBm a 200kbit/s.

Les gains d'antennes ont été déteminés à l'aide de la documentation « RFD900 DataSheet.pdf » page 13 (en Annexe) pour la « Half Wave Dipole Antenna »(3dBi) et la page 12 de la documentation « RFD900 DataSheet V1.2.pdf » pour la « 2 dBi right angle monopole antenna » j'ai considéré que l'autre antenne avait le même gain.

En reprenant l'équation en figure 12, Pe = 10 dBm, Ge = 2 dBi,Gr = 3dBi , Pr = -98 dBm et f = 915 Mhz : on détermine la distance de transmission inférieure ou égale à 11,6 km, sur ces conclusions et avec l'accord du responsable du projet Brice BARRET, nous avons effectué l'achat d'un kit du modem (sur le site mybotshop.de)

1 Autorité de régulation des communications électroniques, des postes et de la distribution de la presse

2 Source: page 19: https://www.arcep.fr/uploads/tx_gsavis/21-1589.pdf

3 Tableau en page 4 du document « RFD900x DataSheet V1.2.pdf »

Implémentation du RFD800x/900x

Le modem RFD doit être alimenté en 5v et se configure en UART avec le paramétrage suivant 57600 baud rate, No parity, 8 data bits, 1 stop bit.

La position des pins est celle donnée en figure 14 avec :

- pin 1 pour GND
- pin 3 pour l'allimentation 5v
- pin 7 pour RX
- pin 9 pour TX

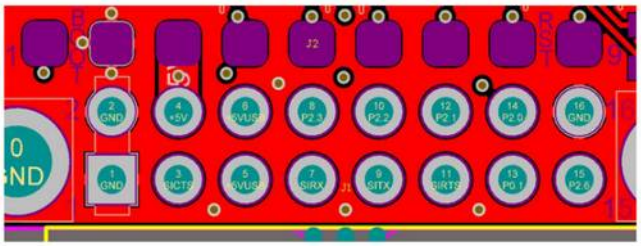


Figure 14: Disposition physique des broches du modem radio RFD900

Le modem qui transmet les données est connecté sur l'Arduino méga, celle-ci possédant 4 hardware serial et n'en utilisant que 2 (le 0 pour la communication usb et le 1 pour la communication GPS) le choix s'est porté sur le Serial 3 avec comme pin Rx : 13 et comme pin Tx : 14

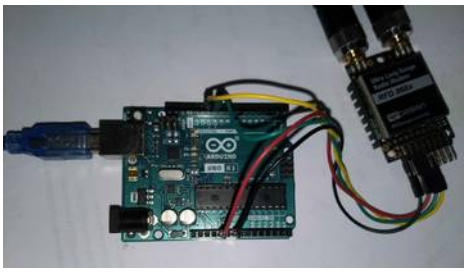


Figure 15: Arduino Uno branchement du modem

Le modem de réception est connecté à une Arduino Uno (cf figure 15), celle-ci ne possédant qu'un seul Hardware Serial, on a utilisé la bibliothèque « Software Serial » et avec comme configuration de pin RX = 11 et TX = 12

On branche respectivement chaque modem à leurs cartes avec le TX du modem au RX d'une Arduino et le RX de l'Arduino en question avec le TX du modem. L'alimentation est aussi prise directement sur l'Arduino (le courant fourni par l'Arduino était suffisant durant les tests)

Pour la première configuration du module RFD800x, j'ai utilisé le logiciel fourni par la RF DESIGN.

Je l'ai configuré de façon à être conforme à la réglementation européenne :

Duty Cycle = 1 %

Tx Power = 10 dBm

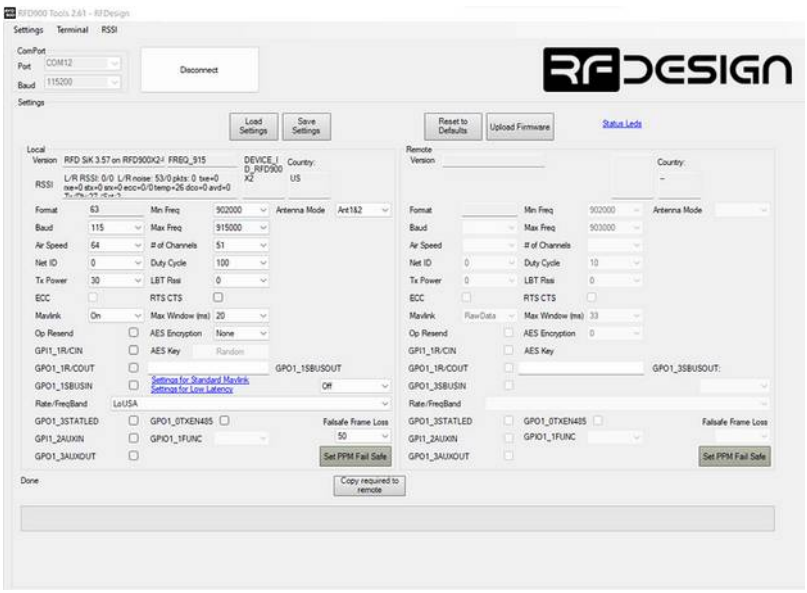


Figure 16: Logiciel configuration RFD900x et RFD800x

J'ai ensuite voulu configurer automatiquement le modem RFD900x par les cartes Arduino à leur mise sous tension. Dans la documentation « RFD900x Peer-to-peer User Manual V1.1 » on apprend à la page 3 que le système fonctionne avec des commandes AT. Ce sont des commandes à envoyer par le serial pour reconfigurer le modem.

- Passer en mode configuration : « +++ »

- Modifier la vitesse de transmission : (AIR_SPEED) « ATS2=188 » pour une vitesse de 188 Kbit/s

- Modifier l'ID de connection (NETID) : « ATS3=169 » l'ID a été choisi au hasard.

- Modifier la puissance de transmission : « ATS4=10 » pour 10dBm

- Modifier la fréquence d'utilisation de la bande passante (DUTY_CYCLE) : « ATS11=1 »

- Enregistrer les modifications : « AT&W »

- Redémarrer le modem : « ATZ »

```
void setupRadio(void) {
    radio.begin(57600,SERIAL_8N1);
    radio.write(START_COMMANDE_AT,3);
    _attente();
    // delay(1000);
    sendATCommand("ATS2=188");//188 vi

    sendATCommand("ATS3=169");//NETID

    sendATCommand("ATS4=10");//10mW

    sendATCommand("ATS11=1");//1% duty
    sendATCommand("AT&W");
    sendATCommand(REBOOT);//stop comma

    delay(1000);
}
```

Figure 17: Programme initialisation RDF_MODEM.cpp

En figure 17, on peut lire le programme d'initialisation du modem implémenter dans l'Arduino Mega. La commande « sendATCommand » permet d'envoyer une commande et d'attendre la réponse du modem. Cette réponse est « OK », si on est en mode DEBUG. Si au bout de 3s on n'a pas de réponse on reçoit le message sur le serial « Pas de Reponse »

L'un des problèmes que j'ai rencontré a été que j'avais du mal à rentrer en mode configuration. Grâce à l'analyseur de signal, j'ai pu regarder la trame qu'envoyait l'Arduino au modem. Je me suis aperçu que j'envoyais « + + + <CR> » . Le caractère ascii « <CR> » est en fait celui du retour chariot qui donne la fin d'une chaîne de caractère. C'est pour cela que dans la configuration finale, on utilise Serial.write pour contrôler précisément l'envoie de 3 caractères (« + ++ »). Et donc on rentre bien en mode configuration.

Pour tester si la configuration était bien appliquée, le protocole d'expérimentation que j'ai utiliser est :

- noter grâce au logiciel de configuration les paramètres internes du modem,

- appliquer une configuration différente par le programme Arduino,

- vérifier que la configuration soit bien appliquée à l'aide du logiciel de configuration.

Je n'ai pas directement vérifier si le composant était fiable, c'est à dire que le paramétrage était conforme aux signaux physiques. J'ai pu quand même noter durant mes tests que lorsque j'avais un bit rate faible par rapport au duty cycle que l'envoi de données était modifié. C'est comme si le modem gardait les données en mémoire le temps qu'il puisse à nouveau en transmettre, et dès qu'il peut envoyer des données, il envoie plusieurs trames en même temps.

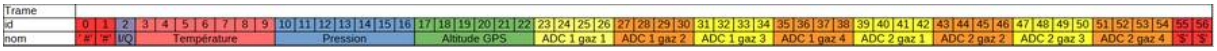


Figure 18: Trame de transmission radio modem RDF900x

Comme le modem pouvait envoyer une trame plus longue, j'ai crée une nouvelle fonction démission basée sur l'algorithme du module radio NRF24L01, mais avec une seule trame de 57 caractères (cf : figure 18)

L'algorithme qui permet d'envoyer les données est donné en figure 19 (ou annexe 2). Le principe étant de concaténer les valeurs (en bleu dans l'algorithme) à transmettre dans un tableau et d'envoyer ce tableau au modem. On peut noter que j'ai rajouter deux signes au début (##) et deux à la fin ($$) de la trame dans le but qu'à la réception je puisse vérifier quand est ce qu'une trame débute et quand est ce qu'elle s'arrête.

J'ai fait le choix d'envoyer deux signes et pas qu'un seul, car les gaz des ADC étant constitués de deux entiers non signés sur 16 bits, il ne fallait pas que je détecte un signe dans l'octet de poids faible de ces entiers. Ces entiers sont l'image de la lecture de tension du capteur alphasens et ont des valeurs entre 0 et 4095, soit en hexadécimal de 0x00 0x00 à 0x0F 0xFF. Les signes « # » et « $ » sont dans la table Ascii de valeurs hexadécimal respectivement 0x23 et 0x24, donc supérieures à 0x0F. De ce postula, j'en ai conclu qu'il ne devrait jamais y avoir deux fois d'affilé les signes « # » et « $ » dans la trame principale et que je pouvais les utiliser comme signe de début et fin de trame.

Pour vérifier que la trame était envoyée correctement avec les valeurs voulues, j'ai utilisé l'analyseur de signal que j'ai branché sur le TX de la carte Arduino Mega. J'ai branché la carte Arduino en USB au PC pour afficher le tableau de transmission sur l'interface Serial d'Arduino. Lorsqu'une trame était émise, j'ai vérifié que j'avais bien les mêmes valeurs entre la trame sur l'interface serial et la trame lue sur l'analyseur ; j'ai vérifié aussi que la traille était bien de 57 caractères.

J'ai fait de même en lisant le TX du module de réception pour vérifier que je recevais bien la même trame qu'à l'émission

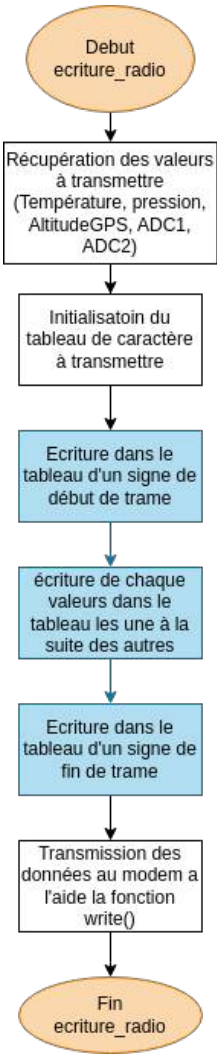


Figure 19: Algorithme écriture radio

Réception de la trame

Après avoir fait toute la partie récupération, traitement et envoi des données par une trame du côté du boîtier embarqué MICROMega, je devais m'occuper de la partie réception de cette trame sur l'Arduino Uno connecté sur l'ordinateur.

Le modem fonctionnant de la même manière à l'émission et à la réception, j'ai utilisé le même programme d'initialisation que dans la partie précédente, à l'exception que j'utilise un software serial au lieu d'un hardware. Il ne devait pas y avoir de différence entre les deux méthodes même si durant mes tests je n'arrivais pas à émettre de données avec la méthode software. N'étant pas une fonctionnalité utile à la réception, ce problème ne nous impacte pas.

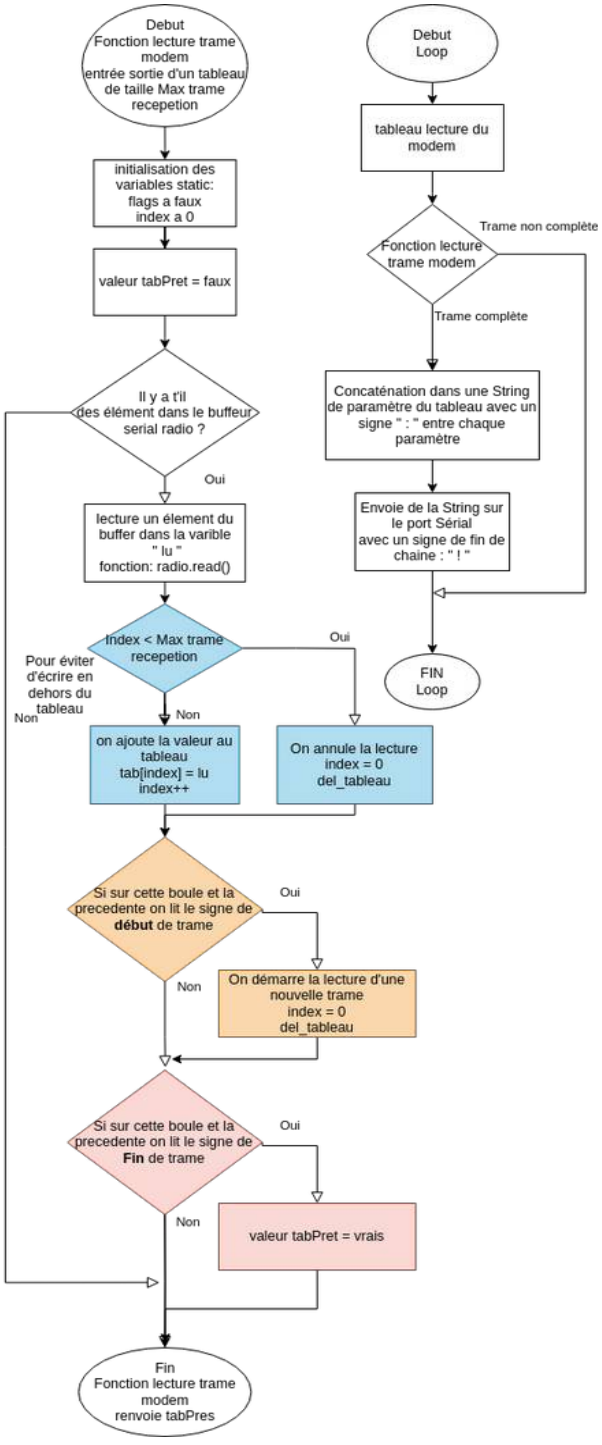


Figure 20: Algorigramme réception

J'ai totalement modifier le programme de réception pour l'adapter à la lecture de ma trame radio.

J'ai développé un programme de réception de la trame venant du modem (cf : algorithme de gauche figure 20).

Lorsqu'on repère que le bluffeur du software sérial n'est pas vide, on lit la valeur inscrite et on l'enregistre dans un tableau de la taille de notre trame.

Pour dé-risquer et éviter un dépassement de mémoire : la partie en bleu vérifie que l'on ne sort pas des limites, si c'est le cas il y a eu un problème dans l'acquisition de la trame, donc on abandonne le tableau qui est en train d'être lu. Sinon on ajoute la valeur lue au tableau

Ensuite vérifier si on est en début de trame (en jaune), si c'est le cas on abandonne notre tableau actuel pour en commencer un nouveau.

Enfin on vérifie si on est en fin de trame (double « $ ») : on fait passer la valeur qui dit à la suite du programme que le tableau est prêt.

L'algorithme du loop main (cf : algorithme de droite figure 20) est une boucle infinie qui lit avec la fonction « lecture trame modem » (expliquée précédemment), Lorsque le tableau est prêt, c'est-à-dire lorsqu'il est rempli d'une trame complète : le programme récupère les informations et les ajoute dans une variable de type chaîne de caractère (String).

Le format de la chaîne de caractère est en figure 21. Elle commence par un « # » et finit par un « ! » et chaque valeur est séparée par « : »

La particularité avec les ADC c'est que l'on doit avant de les enregistrer dans la chaîne, les convertir de leur forme 4 octets brut en 2 entiers non signé de 16 bits.

$$\#<id>:<Time\ loop>:<Qlté\ GPS>:<Temp>:<Pres>:<Alt>:<ADC1_{Gaz1}>:<ADC1_{Gaz2}>:\ldots:<ADC2_{Gaz4}>!$$

Figure 21: Format communication Arduino Uno - PC

Vérification de l'ensemble de la communication

Pour finir avec cette fonction, j'ai fait un test de fonctionnement global. Pour cela, on branche l'Arduino Méga en USB au PC et on fait de même avec l'Arduino UNO. On vérifie aussi que les modems sont bien connectés aux Arduino. On ouvre les communications serial, respectivement le COM11 pour la Méga et le COM3 pour la UNO (cette configuration peut changer en fonction des PC utilisés). On active l'horodatage des messages.

Le but du test est de vérifier que l'on reçoive bien les mêmes valeurs entre l'Arduino méga et l'Arduino UNO. Il devrait aussi y avoir normalement un écart de moins de 500 ms entre l'émission et la réception.

Le test étant concluant, j'ai pu m'attaquer à l'affichage des valeurs expliquées en partie 3.

D) Programme Baromètre, Hygromètre, Thermomètre

Ces trois fonctions n'ont pas été grandement modifiées. Chaque modules communique de manière différente, analogique pour le thermomètre, SPI pour l' hygromètre et I2C pour le baromètre. Malgré le système de communication différent leurs programmes sont sensiblement les mêmes dans leur fonctionnement :

- elles utilisent toutes un tableau de caractères pour enregistrer leurs valeurs ;

- leur algorithme respectif est le même :

  - en entrée les fonctions prennent leur tableau respectif de valeurs
  - on interroge valeur du module par sa bibliothèque dédiée,
  - on vérifie que la valeur n'est pas aberrante ou en dehors des limites du composant,
  - si c'est le cas, on écrit « nan » dans le tableau de retours,
  - sinon on écrit sa valeur dans ce même tableau.

Il y une exception pour le baromètre qui possède un thermomètre. J'ai rajouté le fait de récupérer sa valeur de température et de l'enregistrer de la même manière.

Figure 23: Module capteur d'humidité

Figure 24: Module capteur de pression

Figure 22: Module capteur de température

E) Programme Alphasense :

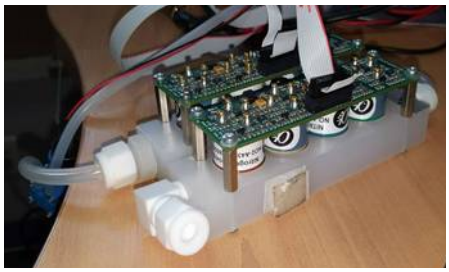


Figure 25: 2 Module Alphasense

La première chose notable avec le module Alphasense c'est qu'il est connecté à une carte d'acquisition et pour lire les valeurs on communique avec les ADC de cette carte. La communication se fait par liaison SPI. J'ai cherché un moment la documentation pour communiquer avec les modules Alphasense, parce que je n'avais pas compris que pour lire les valeurs du module Alphasense on discutait avec des ADC.

Une fois la documentation des ADC en main, j'ai vérifier qu'on lisait bien les valeurs que l'on avait besoin. J'ai testé la liaison entre les sorties des modules Alphasense et les entrées des ADC pour vérifier que les appels avec la fonction SPI.transfer soient les bons. J'ai eu un problème d'interprétation du code parce qu'en liaison SPI l'envoi d'une commande et la réception sont différées. Quand on demande la valeur à l'entrée d'un pin, on reçoit la valeur de la précédente commande. Autrement dit, quand je demande la valeur du pin 2, je reçois la valeur du pin 1, parce que la commande précédente est : donne moi la valeur du pin 1. Une fois cette logique comprise, j'en ai conclu que je n'avais rien à modifier sur cette fonction .

Même si cette conclusion était à prévoir, il m'était important, durant le projet de bien comprendre le fonctionnement de chaque partie.

Les valeurs brutes lues sont ensuite recomposées pour passer de 2 octet en 1 entier sur 16 bits. J'ai créé un type particulier pour faciliter la lisibilité du programme. Les données sur 16 bits sont donc stockées dans ce type particulier pour pouvoir les enregistrer dans la SD par la suite.

J'ai rajouté le fait de stocker également les valeurs brutes dans un tableau pour ne pas à avoir à repasser de 1 entier sur 16 bits à 2 octets.

F) Programme GPS_NEO :

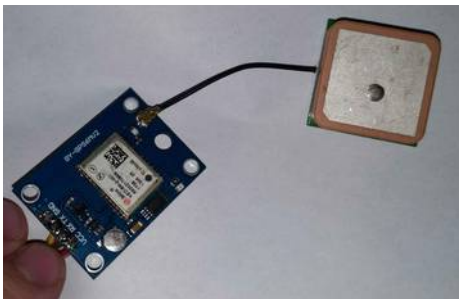


Figure 26: Module GPS avec son antenne

Comme dit dans la partie A) « l'acquisition des données GPS se fait en dehors de la zone principale du programme ». C'est parce qu'à la base, l'acquisition se faisait dans la boucle de lecture des capteurs à l'aide d'une boucle While rendant la fonction bloquante. J'ai modifié la fonction de lecture pour l'utiliser dans la boucle du loop, limitant sa propriété de bloquer le programme.

J'utilise encore une boucle while, mais je n'y rentre uniquement que si on reçoit une trame venant du GPS. Je ne me bloque pas à attendre dans cette boucle si je ne reçois pas de trame.

L'acquisition des données ce fait sur le Serial 1 en UART (115200 bauds 8 bit data, no parity, 1 bit stop)

Lorsque le module gps fonctionne il nous envoie des trames NMEA, je cite le document[4] «trames NMEA183.pdf »

- Chaque trame commence par le caractère $.

- Suivi par un groupe de 2 lettres pour l'identifiant du récepteur.

4 Document en quetion: http://www.cedricaoun.net/eie/trames%20NMEA183.pdf

- Puis un groupe de 3 lettres pour l'identifiant de la trame. Suivent ensuite un certain nombre de champs séparés par une "virgule".
- Et enfin un champs optionnel dit checksum précédé du signe *
- Suit la fermeture de la séquence avec un [CR][LF] »

Le sous programme de lecture récupère une ligne commençant par un $ et finissant par une fin de ligne ([LF] ou '\n' en C). Elle renvoie au programme principal un tableau contenant la ligne avec un drapeau pour prévenir que la lecture s'est bien effectuée.

Un autre sous programme permet de vérifier si la trame commence par « GPGGA ». Il permet de dire si la trame est une trame GGA ou pas. La trame GGA (exemple Figure 27) possède les éléments intéressants suivants : le temps UTC, la longitude, la latitude et l'altitude.

Le dernier sous programme, Split GGA, permet de récupérer les informations importantes de la trame. Elle les stocke dans un type dédié. Ce sous programme est une version adaptée du programme de base.

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

Figure 27: Exemple de trame GGA

G) Programme Enregistrement SD

Toutes les données récupérées dans les fonctions précédentes sont acheminées dans la fonction « ecriture_SD ». Cette fonction est la même que d'origine, les modifications majeures sont l'initialisation et la trame d'enregistrement. La communication se fait en SPI.

Figure 28: Module SD

L'un des problèmes du programme de base était que les données s'enregistraient dans le même fichier entre deux mises sous tension de la carte. Pour régler ce problème, le nom du fichier de la dernière session d'enregistrement est le nom avec le plus haut numéro. Lorsqu'on initialise la fonction SD, on vérifie s'il existe un fichier avec un numéro, en commençant par 0, si le fichier existe on passe au numéro supérieur. On fait ce test jusqu'à ce qu'on trouve un fichier inexistant, le nom de ce fichier sera celui de la session d'enregistrement.

Dans les pistes d'amélioration, je pourrai récupèrer depuis le GPS une trame RMC, pour donner au fichier un nom avec la date et l'heure de sa création.

La trame d'enregistrement est la suivante :

```
[ id | hh | mm | ss | long | lat | alt | adc 1 | adc 2 | temp | hum | pres | temp_baro | tmp_pro ]
```

Le fonctionnement du sous programme est le suivant :

- on ouvre le fichier de sauvegarde créé durant le setup
- on écrit chaque partie de la trame d'enregistrement avec un espace entre chaque valeur.
- on revient à la ligne
- on ferme le fichier

On note que lorsqu'on ne détecte pas de SD, on bloque volontairement le programme tant qu'on ne retrouve pas la SD.

4. Résultats

Pour les résultats on va reprendre les 2 points que je devais améliorer pour ce stage :

- les mesures doivent se faire toutes les secondes
- la portée de réception des données doit être supérieure à 1km

Pour la partie fréquences des mesures, j'ai fais fonctionner le système durant 2 fois 2 jours

- Une fois avec le GPS connecté
- Une fois sans le GPS

les résultats avec le GPS activé me donnent une moyenne de mesure toute les 999 ms avec un écart de 25ms

Les résultats sans le GPS me donnent une moyenne de mesure toute les 1 seconde avec un écart de 2ms

En vue du temps de latence des capteurs et du temps pour faire les mesures, cet écart de maximum 50ms est négligeable. Je peux donc valider ce point du cahier des charges.

Pour la distance de réception, le test n'a pas pu être encore fait, mais en vue de la distance théorique du module, je suis confiant sur les résultats.

Pour la suite de mon travail, je compte faire ce test de distance mais aussi faire la documentation de mes programmes et un manuel de mise en place du système complet.

Partie 3. Affichage de l'information

1. Activités réalisées

En plus d'acheminer les informations sur l'ordinateur, il faut que l'utilisateur puisse les interpréter facilement et rapidement. En effet lorsque Brice BARRET sera en Alaska, il est important qu'il puisse au mieux informer le pilote de l'aéronef pour faire en sorte de rester le plus longtemps dans les nuages de pollutions. Plus longtemps le système MicroMEGA est dans un nuage meilleur seront les relevés.

C'est pour cela que une IHM étais initialement développée sous mathLab, elle permettait d'afficher les concentrations de chaque gaz en fonction d'une configuration de capteur choisi. Malheureusement il y avait un bug qui faisait planter l'IHM au bout de quelques minutes de fonctionnement.

On m'a laissé le choix : soit je réparais l'IHM sous mathLab, soit j'en développais une autre. Ne connaissant pas suffisamment le langage de matLab et préférant utiliser un logiciel libre, j'ai pris la décision d'en développer une autre sous Python3.

J'ai donc réalisé une IHM sous python dans le but d'afficher en temps réel les mesures des capteurs transmits par signal radio.

J'ai dû aussi modifier le programme d'émission des données de la carte Arduino Uno vers l'ordinateur.

2. Vos réalisations

J'ai d'abord cherché une bibliothèque qui permettait d'afficher des graphiques en temps réel, la bibliothèque mathPlot n'étant pas adaptée pour ce genre de tâche, j'ai utilise la bibliothèque PyQTgraph avec sa sous classe PySide6

Figure 29: Logo PySide

Dans un premier temps pour me familiariser avec la bibliothèque, j'ai fais quelque fonctions : récupération des données de l'Arduino Uno, d'affichage d'un page avec des graphiques et des valeurs en temps réel. Création d'une trame de communication entre l'Arduino Uno et le programme Python.

Quand j'ai commencé a avoir une interface fonctionnelle, je suis allé voir Brice BARRET, qui allait être utilisateur de cette interface, pour savoir comment il voulait quelle soit. Avec mes connaissances de la bibliothèque et ses demandes, on a conçu le cahier des charges de l'IHM :

- L'IHM devra être modulaire

- Elle devra afficher les valeurs en temps réel et les valeurs passées sous forme de graphiques.

- Calculer les différentes concentrations de gaz en fonction de l'ADC et de valeur donnée par le constructeur Alphasens

Ce cahier des charges mis en place, j'ai recommencé mon code de façon a ne pas le rendre illisible ou inutilisable par les modifications que j'aurais du faire. J'en ai profité pour m'auto-former sur la programmation orienté objet.

Mon programmé est séparé en 3 parties.

* Une partie qui s'occupe de la gestion des données (classe Data) :

- Récupérer les données depuis une trame émise par l'Arduino

- Attribuer le type de gaz aux données en fonction d'une configuration pré-établie par l'utilisateur.

- Faire les calculs des concentrations de chaque gaz et les enregistrer dans un tableau

* Une partie qui s'occupe de la création et l'affichage des fenêtres :

- j'ai 3 type de fenêtre :

- une page « simple » avec une valeur et un graphique.

- une page « double » avec 2 valeurs et un graphique a double courbe.

- une page « info » avec un texte modulaire pour afficher des données.

* Une partie principale qui s'occupe de diriger les 2 classes.

Avec une fenêtre principale avec des boutons pour sélectionner les pages que l'on veut afficher (cf Figure 30)

Une partie importante que je devais prendre en compte est le fait que l'utilisateur puisse changer les capteurs Alphasens. Comme il n'y a aucun moyen de savoir informatiquement quels sont les capteurs branchés, il est demandé à l'utilisateur de donner manuellement la configuration de ses capteurs. C'est à dire a partir des données fournies par Alphasens de créer un fichier de config pour ces capteurs comme ceux déjà en place.

Le premier problème était que lorsque la connexion avec l'Arduino étais rompue, l'IHM se bloquait. Pour résoudre ce problème j'ai obtenu les conseils de Giles ATHIET, qui m'a conseillé d'utiliser la fonction python « try except » pour contrôler le programme en cas d'erreur.

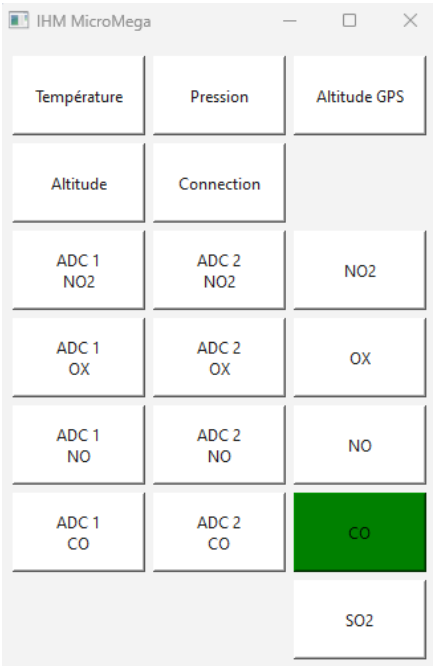


Figure 30: fenêtre de selection de page

Un autre problème rencontré était de l'ordre d'UX design, il fallait que mon IHM soit facile d'utilisation et intuitive. Pour exemple dans une version antérieure, lorsqu'on voulait fermer une page, on pouvait appuyer sur la croix mais le programme ne le prenait pas en compte. Du coup pour rouvrir la page, il fallait appuyer deux fois sur l'ouverture (une fois pour la refermer convenablement et une fois pour l'ouvrir).

Pour résoudre ce problème j'ai du refaire l'algorithme d'ouverture de page. J'ai fait une fonction update qui vérifie a chaque boucle si une page est fermée. Si cette page est fermée le prendre en compte et permettre de la ré-ouvrir d'un clic.

3. Résultats

L'IHM est suffisamment avancé pour être fonctionnel, pour la suite du stage, je ne pense pas faire de modification importante. Mais pour les prochaines personnes qui pourraient modifier l'IHM je compte rajouter des commentaires au programmes.
Je compte aussi faire un manuel d'utilisation et un manuel de création de fichier config.

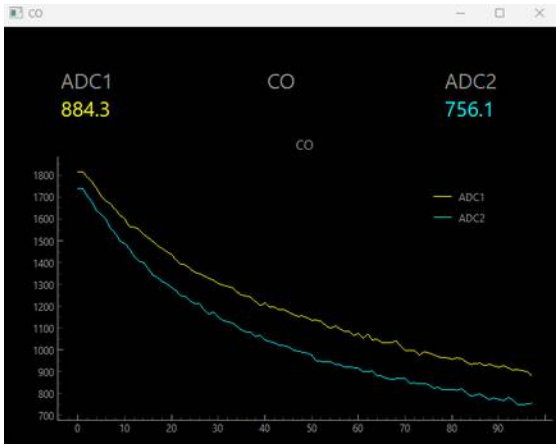


Figure 32: Page double CO

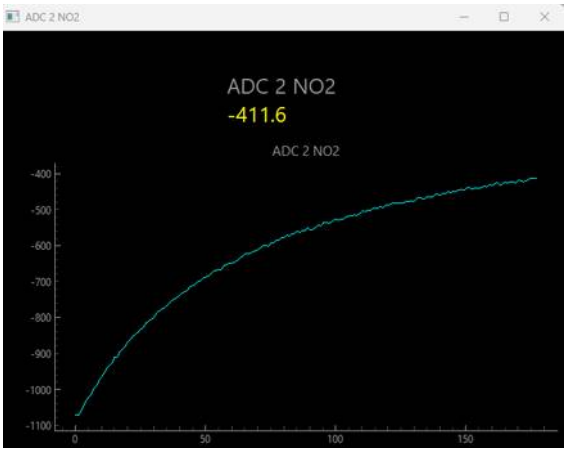


Figure 31: Page simple ADC 2 NO2

Figure 33: page texte pour affiché l'états des signaux

Bilan :

Après les différentes améliorations sur le programme principal de l'Arduino MEGA, l'exécution d'une boucle prend entre 150 - 200 ms. À cette vitesse de lecture on pourraiy théoriquement faire 4 à 5 mesures par seconde. On constate ainsi une amélioration en temps de cinq à dix fois supérieure. On limite quand même l'échantillonnage à une fois par seconde. Plus exactement une fois par seconde à ± 25ms.

Tous les programmes développés durant ce stage sont trouvable à ce lien : https://github.com/Raphael-Nyphalem/OMP-LAERO-Micro-MEGA

Conclusion

Ce stage a constitué une étape intéressante dans mon parcours académique et professionnel. Travailler sur un projet concret, m'a permis de mettre en pratique les connaissances acquises durant mon cursus en Génie Électrique et Informatique Industrielle (GEII). Ce projet a été particulièrement enrichissant car il m'a offert une grande liberté dans l'approche et la réalisation des tâches, ce qui a stimulé ma créativité et ma capacité à résoudre des problèmes aux abords complexes.

L'un des apports majeurs de ce stage a été l'opportunité de travailler dans un environnement professionnel structuré. J'ai confirmé que j'apprécie particulièrement le travail en bureau où je me sens plus motivé et productif que lorsque je travaille chez moi. Ce cadre m'aide à me concentrer et à gérer efficacement mon temps, tout en me permettant de dissocier clairement les espaces de travail et de détente. Ne pas avoir à ramener le stress lié aux délais et aux travaux à rendre chez moi a également été un aspect très appréciable de cette expérience.

Sur le plan personnel, ce stage m'a permis de mieux comprendre mes préférences et mon fonctionnement en milieu professionnel. J'ai découvert que j'aime réellement travailler, surtout lorsque les projets sont stimulants et offrent une liberté créative. Cette expérience a renforcé mon désir de poursuivre dans cette voie et m'a apporté une meilleure compréhension de mes besoins en termes de cadre et d'organisation pour être pleinement efficace.

Pour l'an prochain, je prévois de poursuivre mes études à l'INSA Toulouse en formation Automatique et Électronique. Cette orientation me permettra de développer davantage mes compétences en système embarqué et de me préparer encore mieux à une carrière dans ce domaine.

En conclusion, ce stage a été une expérience formatrice. Il m'a offert l'opportunité de travailler sur un projet concret et significatif, tout en me permettant de mieux comprendre mes aspirations et mes besoins en milieu professionnel. Je suis reconnaissant envers toutes les personnes qui ont rendu cette expérience possible et je me réjouis des perspectives qui s'ouvrent à moi grâce à ce stage.

Bibliographie / Sitographie

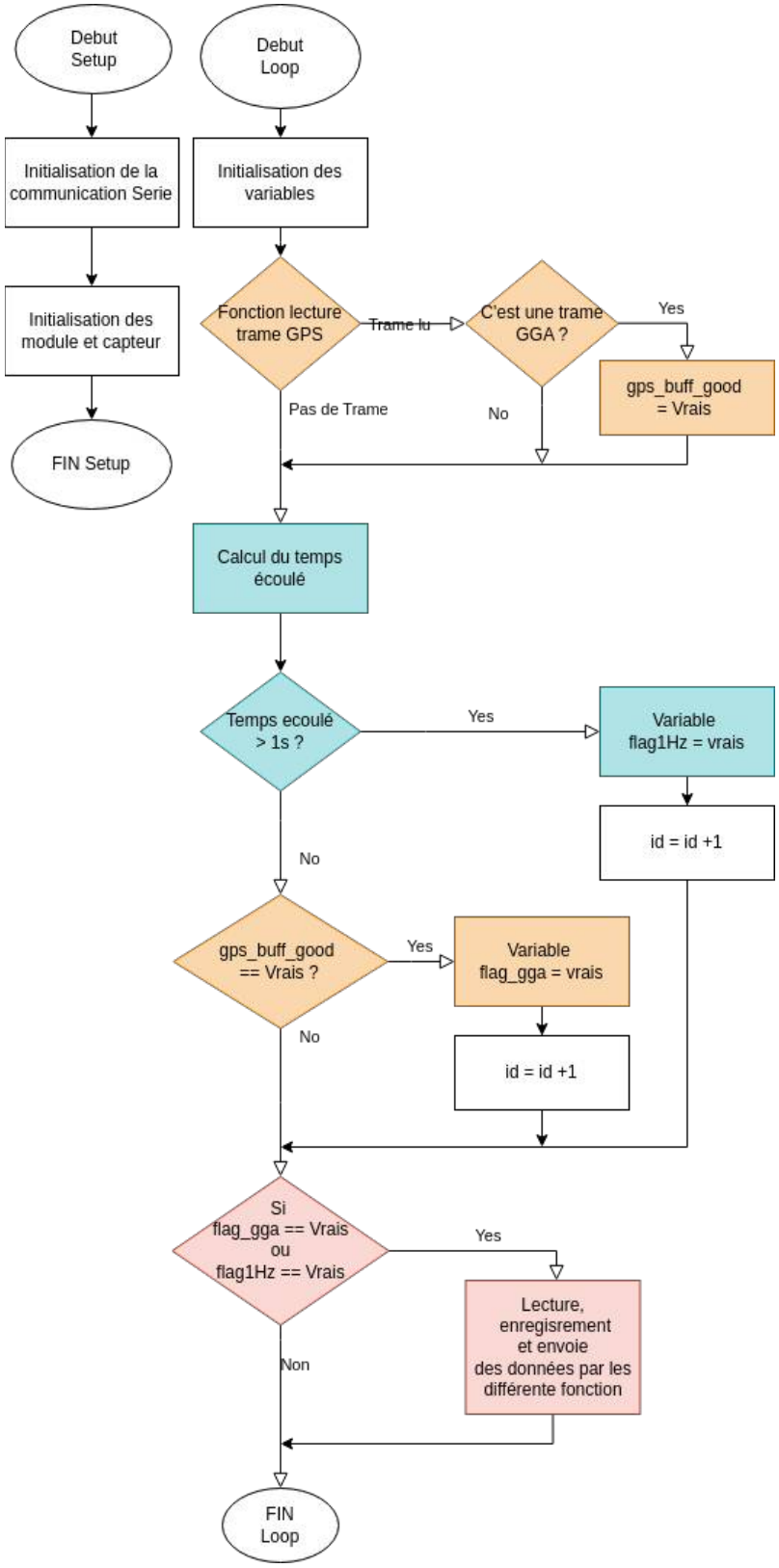
Site de l'Observatoire Pic du Midi : https://www.omp.eu/

Réglementation radio ARCEP : https://www.arcep.fr/uploads/tx_gsavis/21-1589.pdf

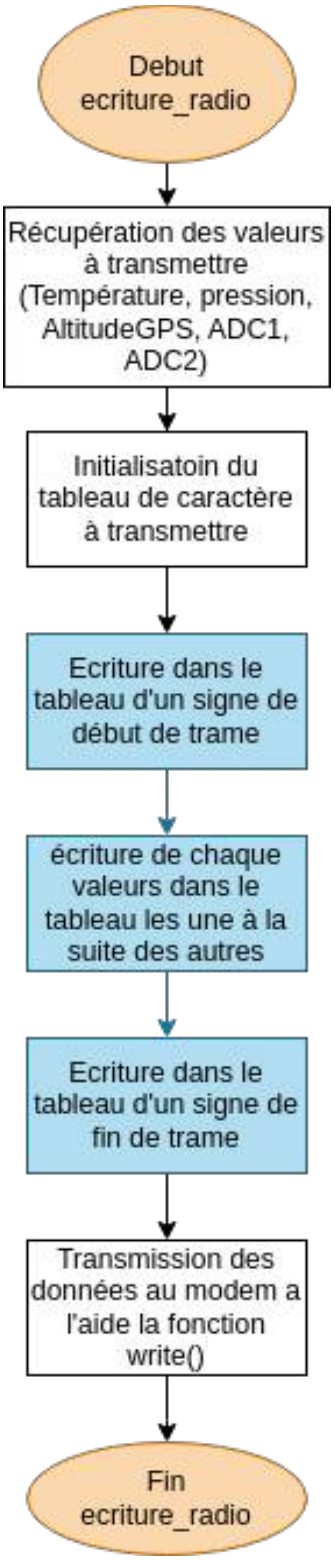
Norme NMEA : http://www.cedricaoun.net/eie/trames%20NMEA183.pdf

Annexes :

Annexe 1 Algorigramme :



Annexe 2 :


Debut ecriture_radio
Récupération des valeurs à transmettre (Température, pression, AltitudeGPS, ADC1, ADC2)
Initialisatoin du tableau de caractère à transmettre
Ecriture dans le tableau d'un signe de début de trame
écriture de chaque valeurs dans le tableau les une à la suite des autres
Ecriture dans le tableau d'un signe de fin de trame
Transmission des données au modem a l'aide la fonction write()
Fin ecriture_radio

Annexe 3 Algorigramme :