Académie Aix-Marseille Lycée Alphonse Benoît SESSION 2022

RAPPORT DE PROJET
REVUE 3

HighLine / SlackLine

MERCHAT RICARD DOYER SOUALMIA

Sommaire

Partie Commune

Contexte :

L'entreprise ROPE & WEB nous a contactés pour pouvoir les aider dans le développement d'un système qui permet la sécurisation de différents systèmes de secours sur corde qui peuvent être délicats à installer et qui nécessitent une grande précision pour éviter tout sur-accident.
Exemple d'installation :

Cette entreprise est aussi chargée de l'installation et de la sécurité de la slackline (une corde tendue entre deux points) pour que Nathan PAULIN puisse exercer sa discipline highline en toute sécurité.
Il est question en 2024 pour les jeux olympiques de tendre une slackline entre la Tour Eiffel et la Tour Montparnasse, soit une distance de 2700 m

Nous, le BTS SN du lycée BENOIT sommes en charge d'équiper cette slackline de capteurs afin d'aider le physicien de la société à modéliser la corde pendant le trajet de Nathan Paulin, de mettre en évidences les forces en jeu, de pouvoir prévenir d'éventuel dangers grâce à différentes alarmes en cas de dépassement de seuil définis en amont par l'entreprise.

Rôle de chaque étudiant :

IR :

MERCHAT Maxence **IR 2.1**	Liste des tâches assurées par l'étudiant -Cas d'utilisation 'Communiquer XBEE'	**Installation :** XCTU, PyCharm **Mise en œuvre :** Module digit 3 XBee. **Configuration :** EDI, XCTU, modules XBee **Réalisation :** Codage de la réception des informations par XBee. **Documentation :** Mise en forme du Cahier de recettes du système total.

EC :

RICARD Loïc **EC 2.2**	Liste des tâches assurées par l'étudiant **Acquisition de la vitesse du vent et de sa direction** -Se documenter sur le bus de communication Modbus RS485 RTU et sur le protocole NMEA. Mettre en service chacun des capteurs proposés. -Produire le schéma structurel d'un Hat RPI permettant de récupérer les données des capteurs (une carte polyvalente si possible). -Prévoir une alarme sonore en cas de dépassement de seuil. -En lien avec l'étudiant IR 2.1 prévoir sur le Hat RPI un support pour module XBEE. -En lien avec les étudiants EC 1.4 et 1.5 prévoir une connectique de liaison I2C. -En lien avec l'étudiant EC 2.3 prévoir un support pour un module GNSS. -Créer le schéma complet du Hat RPI. -Effectuer un routage de cette carte et produire les fichiers afin que la fabrication du PCB soit sous-traitée. - Câbler la carte et effectuer les essais. -	**Installation :** Mise en service (initialisation/configuration) d'un Raspberry Pi : librairie BCM2835, Qt Creator, autres si nécessaire. **Mise en œuvre :** Tester un anémomètre et une girouette, communicants par bus de type Modbus RTU RS485; dans un premier temps la liaison se fera avec un PC avec un adaptateur USB, puis sur Rpi. Tester un capteur intégrant un anémomètre et une girouette, communicant par protocole NMEA. Une comparaison des 2 solutions sera effectuée, en lien avec l'intégralité du projet. Le dépassement de seuils réglables devra déclencher une alarme sonore. Ces essais devront aboutir à la conception du schéma structurel d'un Hat Rpi permettant de récupérer la mesure des capteurs, de disposer d'un module de communication XBEE (en lien avec l'étudiant IR2.1), d'un module GNSS (en lien avec l'étudiant EC2.3) et d'une connectique de communication I2C. **Réalisation :** Après validation de la solution, concevoir un circuit imprimé devant être fabriqué industriellement. **Documentation :** - Schéma de câblage rapide (Fritzing) pour

	Documenter la mise en service de la carte finalisée. -Une comparaison sera effectuée entre les solutions MODBUS et NMEA.	documenter la phase d'essais. - Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. - Schéma structurel avec contours IBD. - Liste complète des composants avec leur source d'approvisionnement, code commande et prix. Programme en C/C++ de communication avec les capteurs, accompagné des commentaires et diagrammes nécessaires à sa compréhension. - Fiche de mise en service. - Fiche de dépannage.

DOYER Paul **EC 2.3**	Liste des tâches assurées par l'étudiant **Acquisition de la position du HighLiner et des ancrages de la highline par GNSS** Mettre en œuvre différents capteurs GNSS pour sélectionner le mieux adapté au projet. En commun avec l'étudiant IR 2.1 mettre en communication le capteur avec un module XBEE3. Évaluer la consommation de l'ensemble pour dimensionner et choisir la partie alimentation. Une régulation de tension sera probablement à prévoir. Produire le schéma structurel de la carte regroupant l'ensemble. Effectuer un routage de cette carte et produire les fichiers afin que la fabrication du PCB soit sous-traitée. Câbler la carte et effectuer les essais. Documenter la mise en service de la carte finalisée.	**Installation :** Mise en service (init./config.) des environnements de développement propres aux différents modules GNSS mis à disposition, et de XCTU pour modules XBEE. **Mise en œuvre :** Mettre en œuvre les différents modules seuls. Puis avec l'étudiant IR 2.1 effectuer une récupération des coordonnées sur le module XBEE. Choisir le capteur le mieux adapté au projet, en exposant les critères de choix. Effectuer des mesures de consommation de l'association capteur / module XBEE. Rechercher et tester une structure d'alimentation autonome de l'ensemble. Proposer un schéma structurel de l'ensemble. L'étudiant EC2.2 intégrera le module GNSS sur le Hat Rpi qu'il doit réaliser, il faudra lui communiquer toutes les informations nécessaires. **Réalisation :** Après validation de la solution retenue, concevoir un circuit imprimé devant être fabriqué industriellement. **Documentation :** - Schéma de câblage rapide (Fritzing) pour documenter la phase d'essais. - Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. - Schéma structurel avec contours IBD. - Liste complète des composants avec leur source d'approvisionnement, code commande et prix. - Programme utilisé pour récupérer les données lors de l'utilisation du capteur seul. Programme de communication avec le capteur, depuis le module XBEE. Accompagnés des commentaires et diagrammes nécessaires à leur compréhension. - Fiche de mise en service. - Fiche de dépannage.

SOUALMIA Jessim **EC 2.4**	Liste des tâches assurées par l'étudiant **Acquisition des données de la centrale inertielle** Mettre en œuvre la centrale inertielle mise à disposition. En lien avec l'étudiant IR 2.1 mettre en communication le capteur avec un module XBEE3 pour récupérer les données. En lien avec l'étudiant EC 2.3 un capteur GNSS sera prévu sur la carte, mais son câblage sera optionnel. Évaluer la consommation de l'ensemble pour dimensionner et choisir la partie alimentation. Une régulation de tension sera probablement à prévoir. Produire le schéma structurel de la carte regroupant l'ensemble. Effectuer un routage de cette carte et produire les fichiers afin que la fabrication du PCB soit sous-traitée. Câbler la carte et effectuer les essais. Documenter la mise en service de la carte finalisée.	**Installation :** Mise en service (init./config.) : -de l'IDE Arduino, et des librairies adaptées, pour prendre en main la centrale inertielle avec une carte Arduino Nano -de XCTU pour modules XBEE. **Mise en œuvre :** Mettre en œuvre le module IMU seul. Justifier le choix de ce modèle en exposant les critères pris en compte. Puis avec l'étudiant IR 2.1 effectuer une récupération des données sur le module XBEE. Puis avec l'étudiant EC 2.3 prévoir l'ajout possible d'un capteur GNSS, qui sera câblé ou non sur la carte selon la localisation de celle-ci. Effectuer des mesures de consommation de l'association. Rechercher et tester une structure d'alimentation autonome de l'ensemble. Proposer un schéma structurel de l'ensemble. **Réalisation :** - Après validation de la solution retenue, concevoir un circuit imprimé devant être fabriqué industriellement. **Documentation :** - Schéma de câblage rapide (Fritzing) pour documenter la phase d'essais. - Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. - Schéma structurel avec contours IBD. - Liste complète des composants avec leur source d'approvisionnement, code commande et prix. - Programme utilisé pour récupérer les données lors de l'utilisation du capteur seul. Programme de communication avec le capteur, depuis le module XBEE. Accompagnés des commentaires et diagrammes nécessaires à leur compréhension. - Fiche de mise en service. - Fiche de dépannage.

Tous les étudiants	Tâches à traiter par l'ensemble des étudiants de l'équipe projet pour le développement de la solution Documents de vie du projet : - Fiches de lecture croisée - Comptes rendus de réunion. Domaines de physique à traiter par l'ensemble des étudiants de l'équipe projet : Les capteurs, les antennes, les OEM, modulations numériques, WIFI. Puissance et énergie. Filtrage numérique	Intégration de la solution et livraison au client du matériel/logiciel/sources/manuels. Le logiciel sera installable facilement chez le client en suivant une procédure écrite.

Annexe *:*

Diagramme exigences :

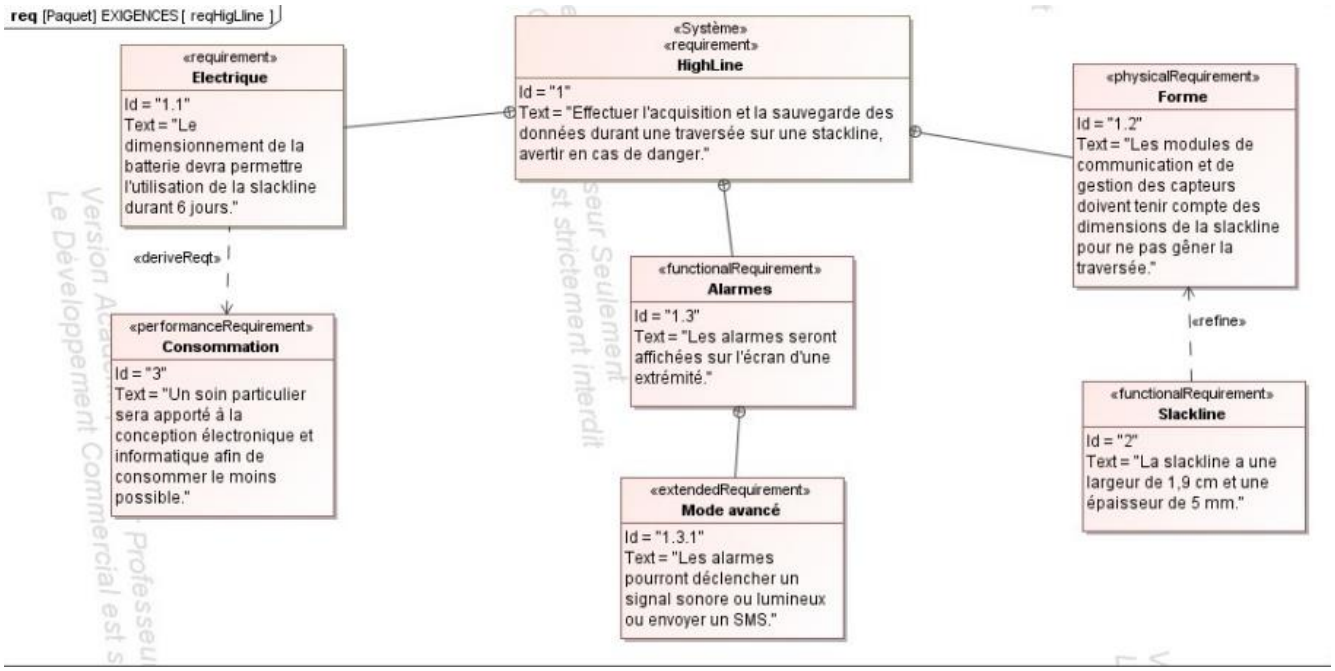


Diagramme de déploiement du système :

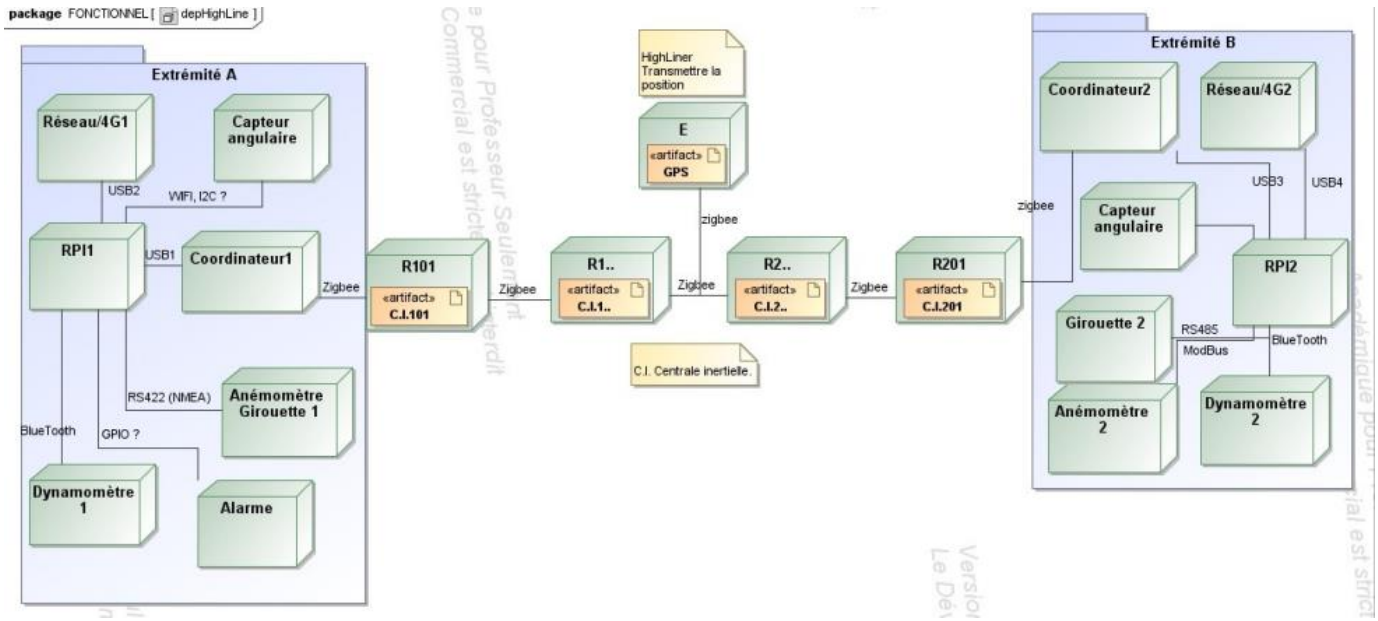


Diagramme de cas d'utilisations :

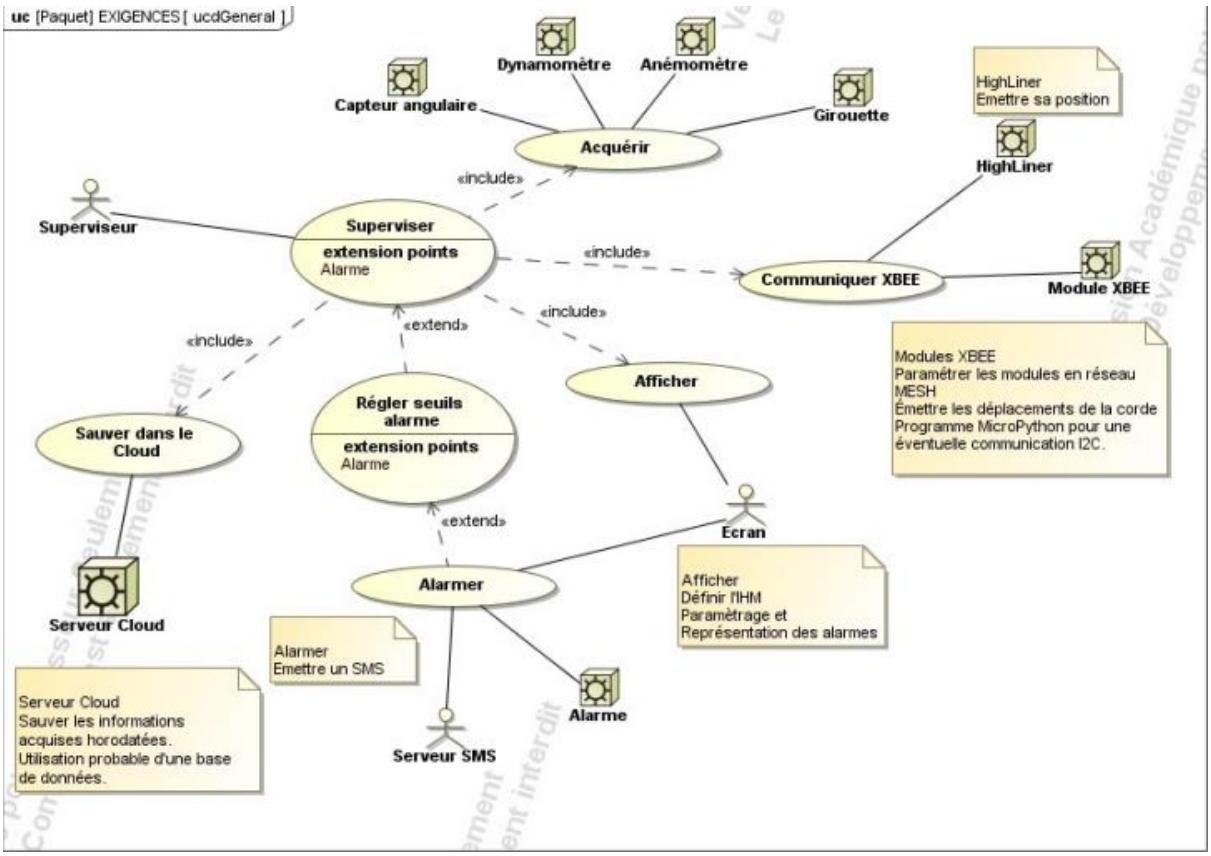
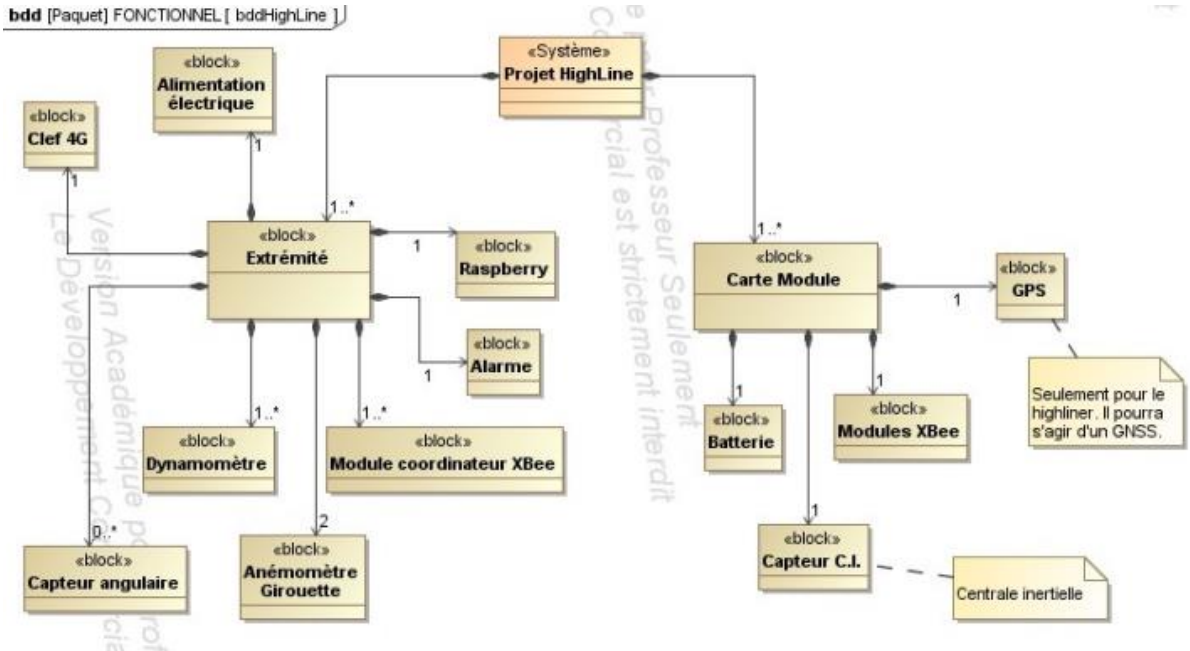


Diagramme de blocs :



Partie individuelle
Merchat Maxence

Pour une bonne communication et réception des données acquis grâce à mes camarades EC, et moi-même je suis en charge de la communication et de l'acquisition des données. Et pour cela j'utilise les modules XBEE et un capteur BNO055 afin que les informations soient acquises et puissent remonter jusqu'à la Raspberry au bout de la ligne.

Pour ce faire, je dispose du matériel suivant :
-3 module XBEE avec 3 câbles USB-Micro USB (pour pouvoir les brancher à l'ordinateur).
-un Capteur BNO055
-de nombreux logiciel telles que pycharm, xctu

Gestion du projet :

Voici le planning de Gantt réaliser avant de commencer le projet :

	Projet	**201 h**	**Mar 03/01/23**	**Mar 30/05/23**		
	Taches communes	**10 h**	**Mar 03/01/23**	**Mer 04/01/23**		
	Lecture du dossier	3 h	Mar 03/01/23	Mar 03/01/23		Maxence;MR; Thomas
	Réunion de début de projet	1 h	Mar 03/01/23	Mar 03/01/23	3	Maxence;MR; Thomas
	Compréhension du contrat, découverte du matériels et documentations	6 h	Mer 04/01/23	Mer 04/01/23	4	Maxence;MR; Thomas
	Planification prévisionnelle	2 h	Mar 10/01/23	Mar 10/01/23	5	Maxence
	Installation et paramètrage XCTU	10 h	Mar 10/01/23	Lun 16/01/23	6	Maxence
	Mise en oeuvre des modules XBEE	25 h	Lun 16/01/23	Lun 30/01/23	7	Maxence
	Rédaction du rapport de projet 1	4 h	Lun 30/01/23	Mar 31/01/23	8	Maxence
	analyse SysML du fonctionnement	10 h	Mar 31/01/23	Lun 06/02/23	9	Maxence
	REVUE 1	30 min	Mar 07/02/23	Mar 07/02/23		
	Conception d'un programme Qt/C++ de communication avec les modules XBEE sur RPI	25 h	Lun 06/02/23	Mer 08/03/23	10	Maxence
	Programme MicroPython de test	23 h	Mer 08/03/23	Mar 21/03/23	12	Maxence
	Intégration dans l'application générale	10 h	Mar 21/03/23	Lun 27/03/23	13	Maxence
	Rédaction du rapport de projet 2	4 h	Lun 27/03/23	Mar 28/03/23	14	Maxence
	Conception de la classe CZigbee	20 h	Mar 28/03/23	Mer 05/04/23	15	Maxence
	REVUE 2	30 min	Mar 11/04/23	Mar 11/04/23		
	Programme MicroPython de lecture d'un capteur I2C	30 h	Mar 11/04/23	Mer 10/05/23	16	Maxence
	Intégration de l'ensemble	8 h	Lun 15/05/23	Mer 17/05/23	18	Maxence

Voici la réalité :

Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédéce	Noms ressources
	Projet	**201 h**	**Mar 03/01/23**	**Mar 30/05/23**		
	Taches communes	**10 h**	**Mar 03/01/23**	**Mer 04/01/23**		
	Lecture du dossier	3 h	Mar 03/01/23	Mar 03/01/23		Maxence;MR; Thomas
	Réunion de début de projet	1 h	Mar 03/01/23	Mar 03/01/23	3	Maxence;MR; Thomas
	Compréhension du contrat, découverte du matériels et documentations	6 h	Mer 04/01/23	Mer 04/01/23	4	Maxence;MR; Thomas
	Planification prévisionnelle	2 h	Mar 10/01/23	Mar 10/01/23	5	Maxence
	Installation et paramètrage XCTU	10 h	Mar 10/01/23	Lun 16/01/23	6	Maxence
	Mise en oeuvre des modules XBEE	25 h	Lun 16/01/23	Lun 30/01/23	7	Maxence
	Rédaction du rapport de projet 1	4 h	Lun 30/01/23	Mar 31/01/23	8	Maxence
	analyse SysML du fonctionnement	10 h	Mar 31/01/23	Lun 06/02/23	9	Maxence
	REVUE 1	30 min	Mar 07/02/23	Mar 07/02/23		
	Programme MicroPython de test	23 h	Mer 08/03/23	Mar 21/03/23	18	Maxence
	Intégration dans l'application générale	10 h	Mar 21/03/23	Lun 27/03/23	12	Maxence
	Programme MicroPython de lecture d'un capteur I2C	30 h	Mar 11/04/23	Mer 10/05/23	17	Maxence
	Rédaction du rapport de projet 2	4 h	Lun 27/03/23	Mar 28/03/23	13	Maxence
	REVUE 2	30 min	Mar 11/04/23	Mar 11/04/23		
	Conception de la classe CZigbee	20 h	Mar 28/03/23	Mer 05/04/23	15	Maxence
	Conception d'un programme Qt/C++ de communication avec les modules XBEE sur RPI	25 h	Lun 06/02/23	Mer 08/03/23	10	Maxence
	Intégration de l'ensemble	8 h	Lun 15/05/23	Mer 17/05/23	14	Maxence

On peut voir que j'ai effectué le codage en micropython plus rapidement que prévu, et la partie QT/C++ je l'ai mise de côté pour la faire vers la fin. La partie micropython m'a pris plus de temps que prévu dû à certains bugs ou juste une incompréhension de ma part. Il y a aussi la prise en main du module xbee qui a étais un peu long comparé au capteur BNO055 qui a étais un peu plus facile, mais cela reste une découverte donc cela m'a pris quand même du temps.

Mise en œuvre et essaies des modules XBEE à l'aide de l'application X-CTU

Introduction à XBEE et X-CTU:

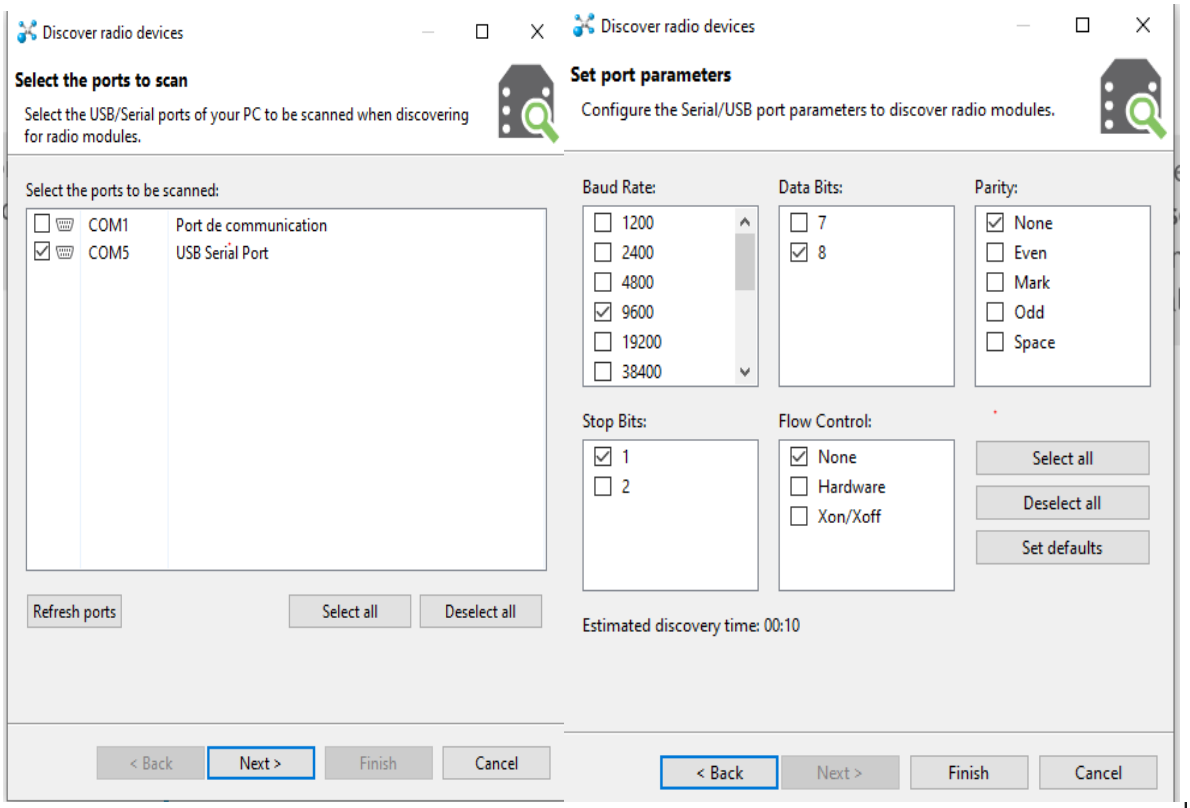
Il faut savoir que le module xbee et l'utilisation a été choisi au préalable par les professeurs et non un choix personnel

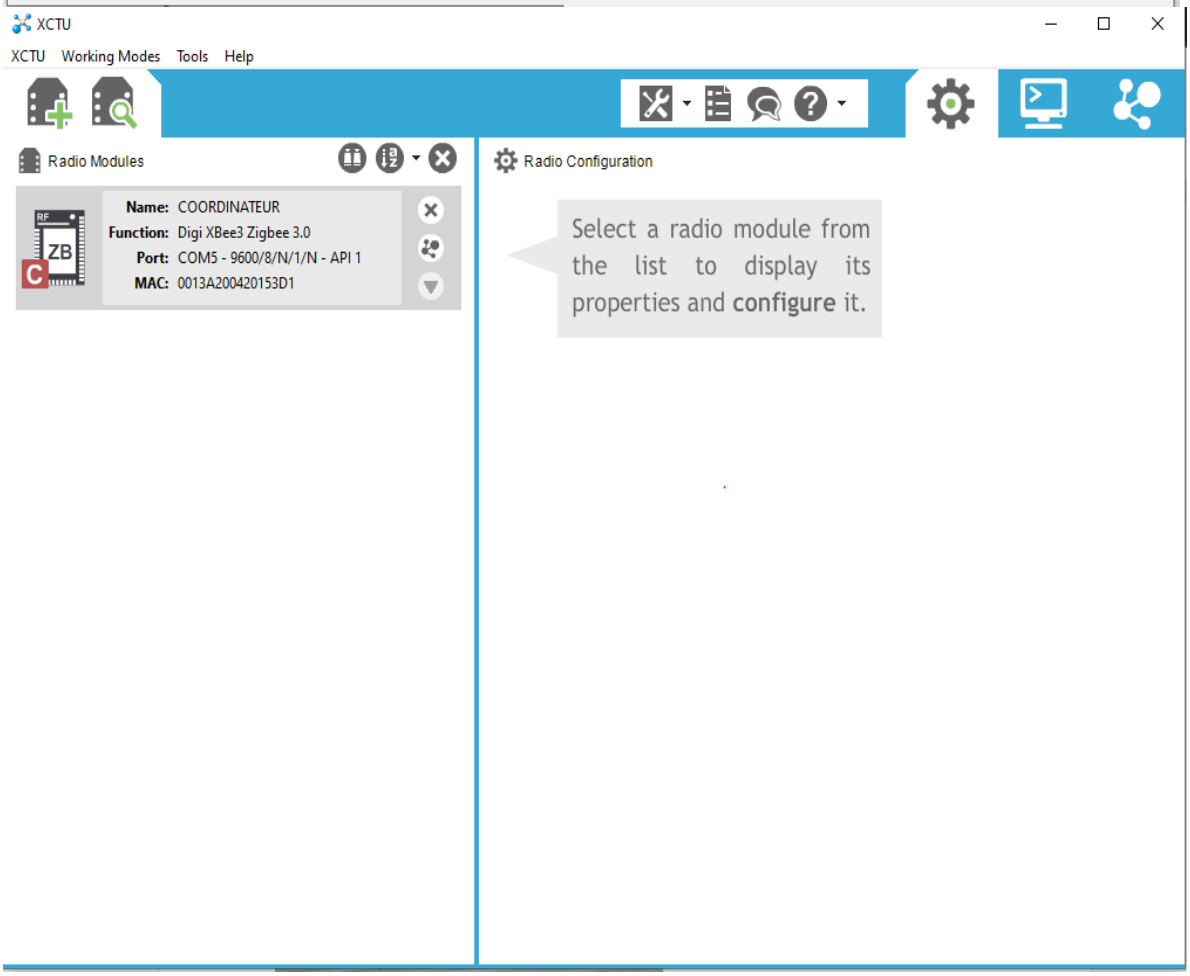
XBee est une technologie de communication sans fil conçue pour les réseaux de capteurs et les applications M2M (Le M2M, Machine to Machine, est le terme pour décrire les technologies utilisées par les machines afin de communiquer entre elles, sans intervention humaine directe).

Elle utilise le protocole de communication Zigbee pour permettre une communication sans fil fiable et économe en énergie entre différents périphériques. Les modules XBee sont largement utilisés dans de nombreux secteurs industriels, tels que l'agriculture, la surveillance environnementale, l'automatisation industrielle...

X-CTU est une application logicielle développée par Digi International pour configurer, tester et déboguer les modules XBee. L'interface graphique conviviale de X-CTU permet aux utilisateurs de configurer facilement les paramètres de leurs modules XBee. L'application offre également des outils de débogage pour résoudre les problèmes de communication.

Image d'un module xbee que j'utilise.





Voici comment on entre un module xbee grâce à l'application X-CTU

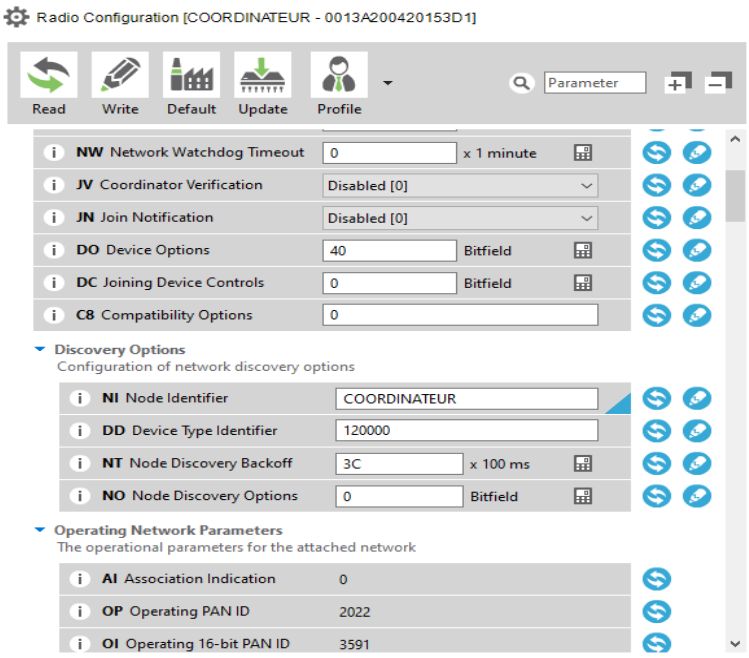
Comment fonctionne un module xbee :

Les modules XBee sont des modules de communication sans fil basés sur la norme ZigBee.

Voici comment fonctionne un module XBee :

1. Les modules XBee sont des modules émetteur-récepteur sans fil qui utilisent la bande de fréquence 2,4 GHz pour la communication sans fil.
2. Les modules XBee communiquent entre eux à l'aide de la norme ZigBee, un protocole de communication sans fil basé sur le standard IEEE 802.15.4.
3. Les modules XBee peuvent être configurés pour agir comme des émetteurs, des récepteurs ou des répéteurs de signal, permettant une communication sans fil à longue portée et à faible consommation d'énergie.
4. Les modules XBee peuvent être programmés pour communiquer avec d'autres appareils sans fil, tels que des microcontrôleurs, des ordinateurs ou d'autres modules XBee.
5. Les modules XBee peuvent être configurés en utilisant le logiciel XCTU (Xbee Configuration and Test Utility) pour définir les paramètres de communication, tels que la puissance de transmission, la fréquence, le taux de données, etc.
6. Les modules XBee peuvent être alimentés à partir d'une source d'alimentation externe, telle qu'une pile ou une alimentation électrique, ou à partir d'une source d'énergie intégrée, telle qu'une batterie,ou même le port USB d'un ordinateur.

En somme, les modules XBee sont des modules de communication sans fil flexibles et puissants qui peuvent être utilisés dans une grande variété d'applications, tels que les réseaux de capteurs sans fil, les systèmes de contrôle de processus industriels, les systèmes de surveillance et de sécurité...



Voici l'interface de paramétrage d'un module xbee

Le mode transparent avec le module XBEE

Il y a différents modes utilisables avec les modules xbee grâce à l'application X-CTU mais on va se pencher sur un en particulier :

Le mode Transparent (ou mode transparent de transmission) est un mode de fonctionnement des modules XBee qui permet une transmission de données sans fil de manière simple et directe. Dans ce mode, les modules XBee agissent comme des modems sans fil, transférant les données reçues d'un périphérique à l'autre sans les interpréter ou les modifier.

Lorsque les modules XBee sont configurés en mode transparent, les données reçues d'un module sont transmises immédiatement à l'autre module, sans être modifiées ou interprétées. Cela permet une communication sans fil simple et directe entre les deux modules.

Le mode Transparent est souvent utilisé pour la transmission de données en temps réel, tels que les données de capteurs, de systèmes de surveillance, ou pour la commande à distance de systèmes automatisés.

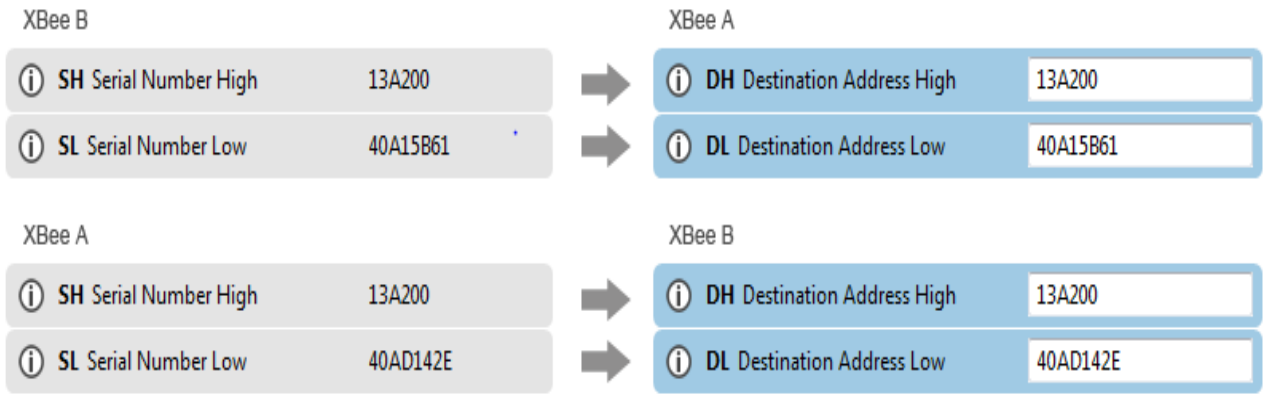
Pour configurer les modules XBee en mode Transparent, il suffit de sélectionner le mode Transparent dans l'onglet “Modem Configuration" de X-CTU. Les paramètres de configuration tels que la vitesse de transmission, le canal de communication et la qualité de service peuvent être ajustés pour optimiser la transmission de données.

Il convient de noter que le mode Transparent ne fournit pas de vérification d'erreur ni de correction d'erreur lors de la transmission de données. Par conséquent, il est important de prendre en compte la qualité de la communication sans fil et de mettre en place des mesures de sauvegarde en cas de perte de données.

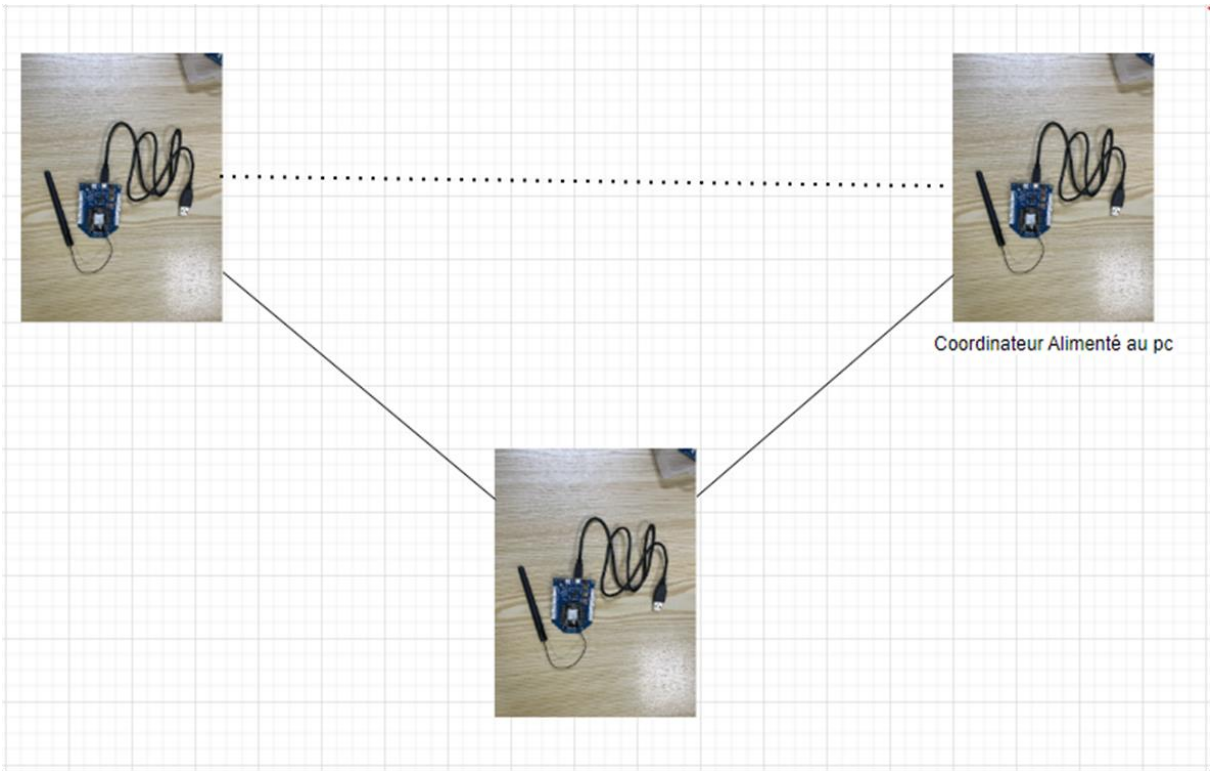
En résumé, le mode Transparent permet une transmission de données sans fil simple et directe entre les modules XBee, sans interpréter ni modifier les données reçues. Ce mode est souvent utilisé pour la transmission de données en temps réel et peut être configuré facilement à l'aide de l'application X-CTU.

Mon rôle est de réussir à faire communiquer les modules entre eux pour que les informations acquis puissent remonter jusqu' à la Raspberry au bout de la ligne donc l'utilisation du mode transparent est plus qu'envisageable.

Pour que deux modules XBee puissent communiquer, le module émetteur a besoin de l'adresse du destinataire. Lorsque vous travaillez en mode transparent, vous devez configurer cette adresse dans le module qui communique. Les modules XBee peuvent stocker l'adresse complète de 64 bits du module destinataire. Cette adresse doit être programmée dans deux paramètres : Adresse de destination haute (DH) et Adresse de destination basse (DL). Si vous voulez que les modules A et B communiquent, configurez l'adresse de destination (DH + DL) du XBee A comme l'adresse MAC (SH + SL) du XBee B, et vice versa.

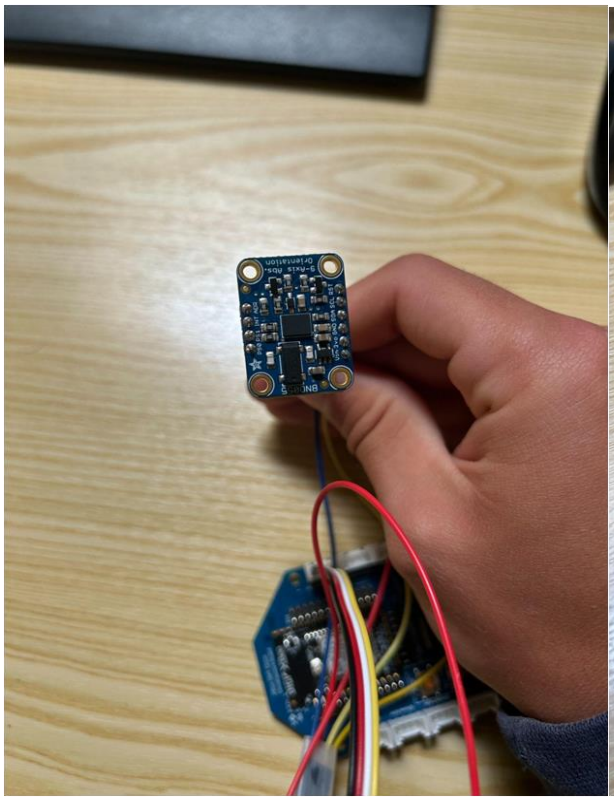
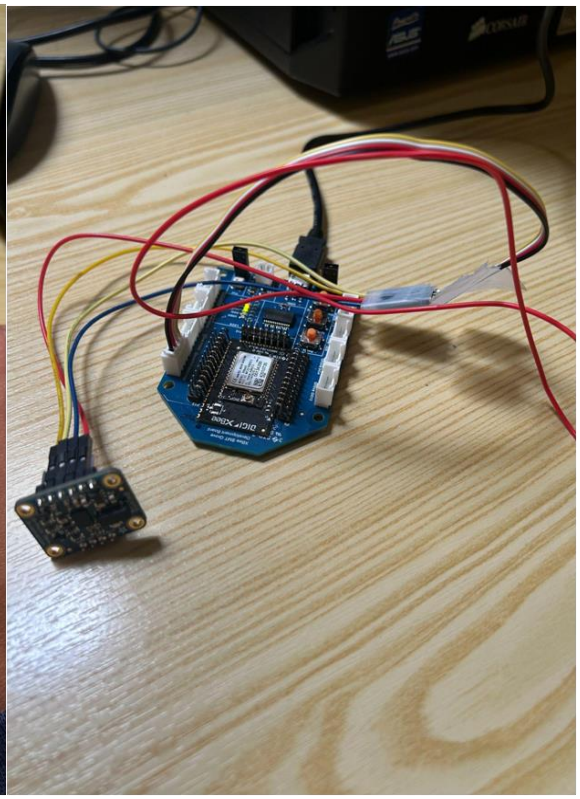


Test module :



-j'ai effectué un test pour voir si un module pouvait se servir d'un autre module comme passerelle pour atteindre le module coordinateur. Comme pour le jour final ou les modules (en dessous de la corde) auront comme adresse de destination le module au bout de la ligne et se serviront des modules à côté comme passerelle pour atteindre le bout de la ligne. Et ce test a été un succès, le module s'est bien servi d'un autre module pour atteindre le module coordinateur qui était éloigné.

Capteur BNO055 :

Le capteur BNO055 est un capteur d'inertie absolue qui combine plusieurs fonctionnalités telles que l'accéléromètre, le gyroscope et le magnétomètre. Voici quelque caractéristiques clés du capteur BNO055 :

-Mesures inertielles : Le capteur BNO055 fournit des mesures précises de l'accélération linéaire, de la vitesse angulaire(gyroscope) et du champ magnétique(magnétomètre).

-Interface I2C ou SPI : Le capteur BNO055 peut être utilisé avec une interface de communication I2C ou SPI, offrant une flexibilité d'intégration avec différents microcontrôleurs ou plates-formes, ce qui est utile car le module xbee peut utiliser l'interface I2C.

-précision : Le capteur BNO055 offre une précision élevée dans ses mesures, ce qui en fait un choix populaire pour les applications nécessitant des informations précises sur l'orientation et le mouvement.

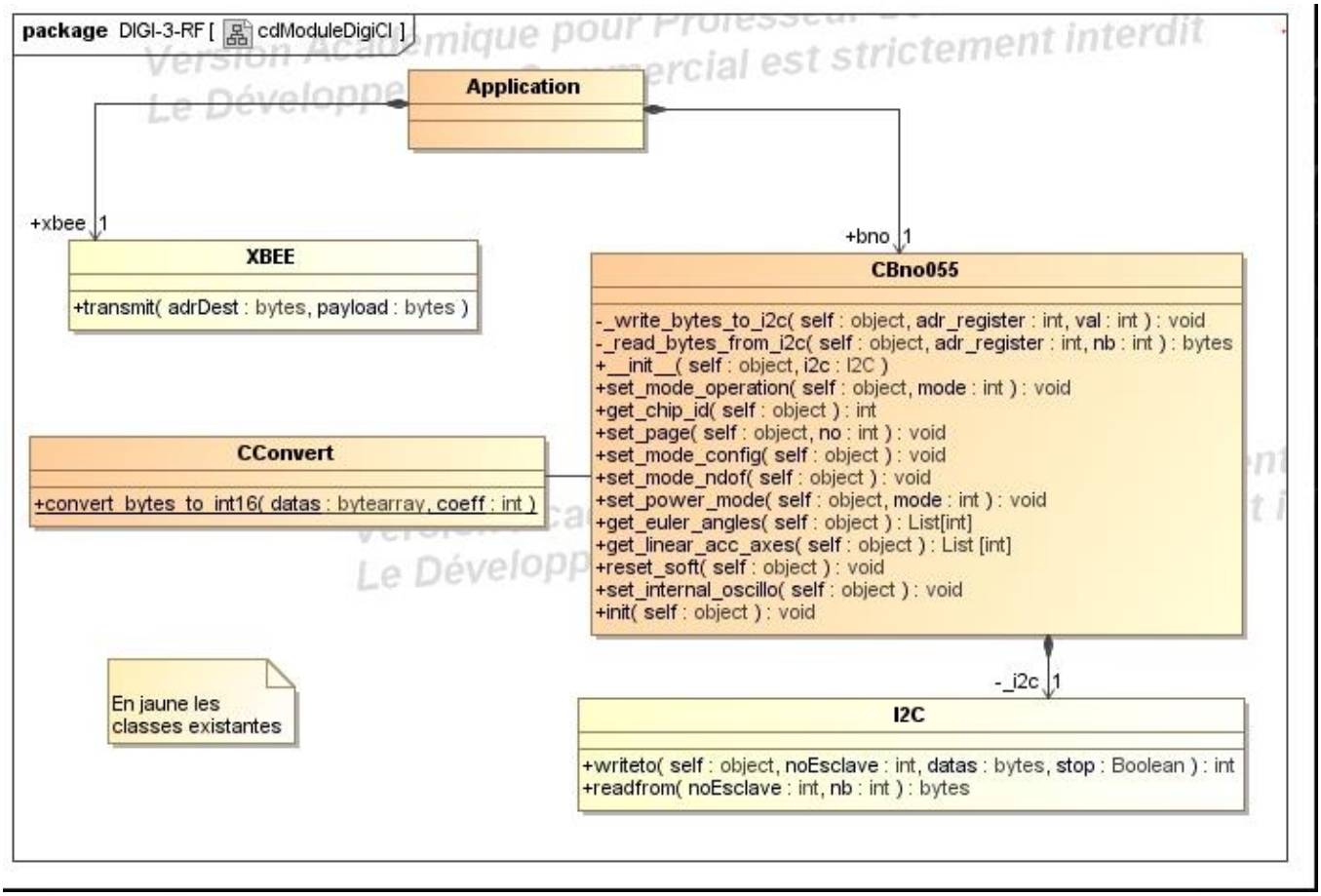
-Faible consommation d'énergie : Le capteur BNO055 est conçu pour une consommation d'énergie réduite, ce qui le rend adapté aux applications alimentées par batterie ou soucieuses de l'efficacité énergétique.

Le choix du capteur BNO055 n'est pas un choix personnel. Ce sont les professeurs qui m'ont imposé le choix du capteur qui s'avère être une bonne idée pour l'instant. Pour utiliser le capteur BNO055 j'ai dû le brancher en série grâce à un raccord faits maison comme on peut voir avec l'image à droite (juste au-dessus) car le capteur et le module peuvent communiquer en i2c, c'est pour cela qu'on les a choisies. On note aussi qu'il peut nous transmettre les angles d'euler qui sont le résultat de calcul fait par le capteur BNO055 (à partir du magnétomètre), ces angles sont importants à récolter pour pouvoir réaliser différents calculs pour la suite.

Importation de la librairie :

L'importation de la librairie ne pouvait pas se faire en ligne de commande alors on a dû créer une classe ou toutes les méthodes ont dû être initialisées.

```
def get_euler_angles(self):
    vals = CConvert.convert_bytes_to_int16(self._read_bytes_from_i2c(BNO055_EULER_H_LSB_ADDR, 6))
    fvals = [0] * 3
    for i in range(3):
        fvals[i] = (vals[i] >> 4)
    return fvals

def get_linear_acc_axes(self):
    vals = CConvert.convert_bytes_to_int16(self._read_bytes_from_i2c(BNO055_LINEAR_ACCEL_DATA_X_LSB_ADDR,
    fvals = [0.0] * 3
    for i in range(3):
        fvals[i] = float(vals[i]) / 100
    return fvals

def reset_soft(self):
    self._write_bytes_to_i2c(BNO055_SYS_TRIGGER_ADDR, 0x20)
    time.sleep_ms(700)

def set_internal_oscillo(self):
    self._write_bytes_to_i2c(BNO055_SYS_TRIGGER_ADDR, 0x00)

def init(self):
    self.set_mode_config()

def get_magnetometre(self):
    vals = CConvert.convert_bytes_to_int16(self._read_bytes_from_i2c(BNO055_MAG_DATA_X_LSB_ADDR, 6))
    fvals = [0.0] * 3
    for i in range(3):
        fvals[i] = float(vals[i]) / 16
    return fvals

def get_gyroscope(self):
    vals = CConvert.convert_bytes_to_int16(self._read_bytes_from_i2c(BNO055_GYRO_DATA_X_LSB_ADDR, 6))
    fvals = [0.0] * 3
    for i in range(3):
        fvals[i] = float(vals[i]) / 16
    return fvals
```

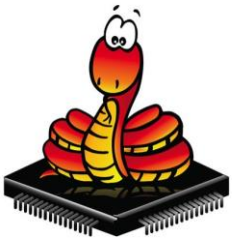


Voici un diagramme de classe CBno055 on peut voir les différentes méthodes utilisées les méthodes peuvent être modifiées ou alors certaines méthodes peuvent être ajoutées d'ici la fin du projet notamment la méthode pour avoir les valeurs du magnétomètre ou celle du gyroscope.

On peut voir que les méthodes ont étaient définies dans la classe CBno055 pour alléger le code (main), on peut voir qu'un calcule est effectué par exemple quand on divise par 16 ou par 100 on se base sur la documentation du BNO055 et en faisant ce calcule on obtient des données avec des unités que l'on connaît par exemple des angles en degrés et non des valeurs non cohérentes.

```
def get_magnetometre(self):
    vals = CConvert.convert_bytes_to_int16
    fvals = [0.0] * 3
    for i in range(3):
        fvals[i] = float(vals[i]) / 16
    return fvals
```

Analyse fonctionnement MicroPython



Le MicroPython est un langage de programmation léger et efficace qui permet d'exécuter du code Python sur des microcontrôleurs et des systèmes embarqués. Il permet de programmer facilement des cartes de développement telles que le Raspberry Pi Pico, le Pyboard, le ESP32 et d'autres, en utilisant le langage de programmation Python.

MicroPython offre une interface simple et intuitive pour contrôler les entrées/sorties numériques et analogiques, ainsi que pour interagir avec des capteurs, des actionneurs et d'autres périphériques. En plus de cela, MicroPython dispose d'un grand nombre de bibliothèques standard et tierces pour faciliter le développement de projets embarqués.

Les modules XBee peuvent être programmés avec différents langages de programmation,

Mais le MicroPython présente plusieurs avantages pour leur utilisation :

Simplification de la programmation : le MicroPython est un langage de programmation facile à apprendre et à utiliser, il permet de réduire la complexité du code, ainsi que le temps et les efforts nécessaires pour le développement de l'application.

Flexibilité : le MicroPython est un langage flexible et peut être utilisé sur différents microcontrôleurs, ce qui permet une grande variété d'applications pour les modules XBee.

Interactivité : le MicroPython permet l'interaction directe avec le module XBee via une console série, ce qui facilite le débogage et la mise au point du code.

Possibilité de personnalisation : en utilisant le MicroPython, il est possible de personnaliser les fonctionnalités des modules XBee selon les besoins spécifiques de l'application.

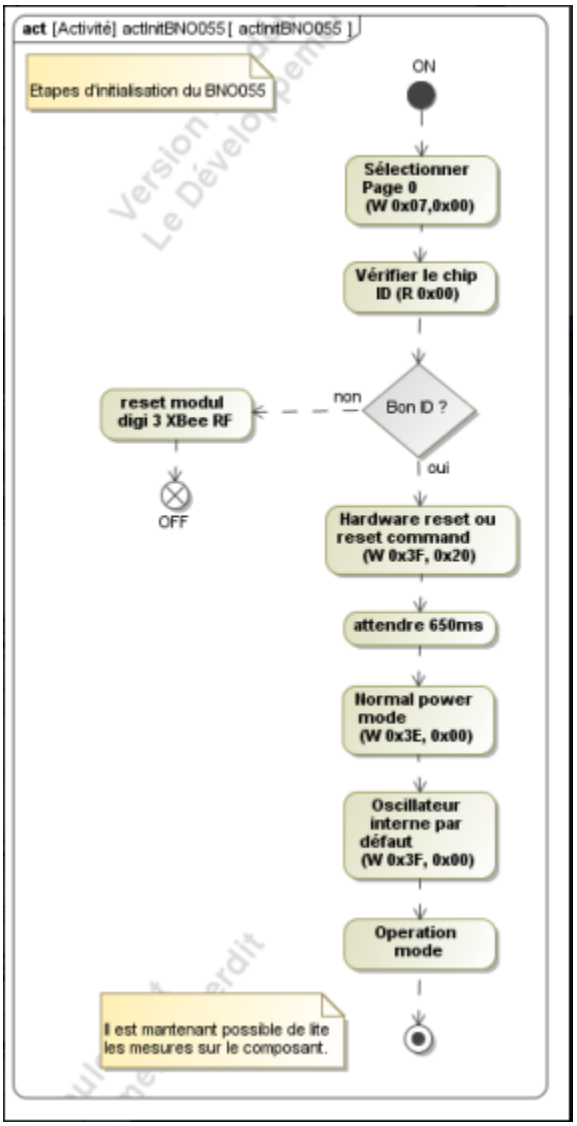
En somme, le MicroPython est un langage de programmation qui permet de faciliter

L’utilisation des modules XBee en offrant une plus grande flexibilité, une interactivité directe et une facilité de développement.

-Voici un code en micropython qui permet de récolter 12 données grâce au capteur BNO055. Il y a les angles d'euler, accéléromètre, magnétomètre et le gyroscope. Si je devais faire un résumer du code je dirais qu'au début du code j'initialise le capteur BNO055 et ensuite j'effectue une boucle (for i in range(10):) ou je récupère les 12 valeurs

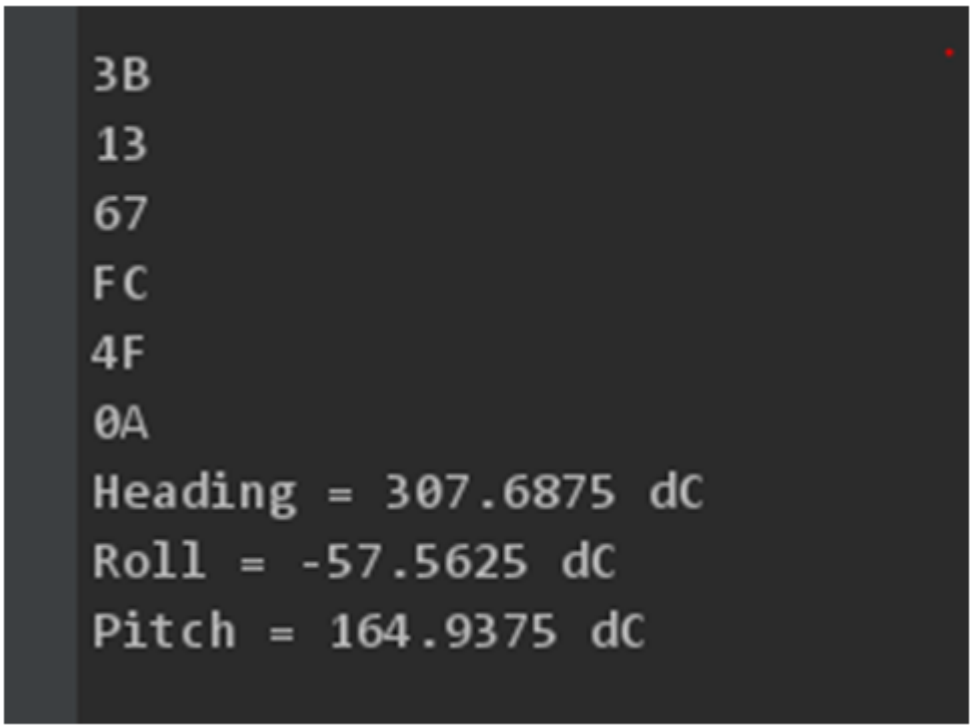
```
import xbee
import time
import machine
from bno055 import *

# PROGRAMME PRINCIPAL
bno = CBno055()
bno.init()
bno.set_page(0)
bno_id = bno.get_chip_id()
print('CHIP ID = {0:02X}'.format(bno_id))
if bno_id != 0xA0:
    exit(-1)
bno.reset_soft()
bno.set_power_mode(POWER_MODE_NORMAL)
bno.set_internal_oscillo()
bno.set_mode_operation(OPERATION_MODE_NDOF)
time.sleep_ms(10)

for i in range(10):
    datas = bno.get_euler_angles()
    print('Heading = {0} dC'.format(datas[0]), end='')
    print(' Roll = {0} dC'.format(datas[1]), end='')
    print(' Pitch = {0} dC'.format(datas[2]))
    print('  ')
    accelerations = bno.get_linear_acc_axes()
    print('accX = {0:0.2f} m/s2 '.format(accelerations[0]), end='')
    print('accY = {0:0.2f} m/s2 '.format(accelerations[1]), end='')
    print('accZ = {0:0.2f} m/s2 '.format(accelerations[2]))
    print('  ')
    magnetometre = bno.get_magnetometre()
    print('magnX = {0:0.2f} uT '.format(magnetometre[0]), end='')
    print('magnY = {0:0.2f} uT '.format(magnetometre[1]), end='')
    print('magnZ = {0:0.2f} uT '.format(magnetometre[2]))
    print('  ')
    gyroscope = bno.get_gyroscope()
    print('gyrX = {0:0.2f} Dps '.format(gyroscope[0]), end='')
    print('gyrY = {0:0.2f} Dps '.format(gyroscope[1]), end='')
    print('gyrZ = {0:0.2f} Dps '.format(gyroscope[2]))
    print('  ')
```

-Voici un diagramme de bloc de l'initialisation du capteur BNO055 ce qui équivaut au début du code (au-dessus). Ce qui facilite la compréhension de code de au-dessus c'est à partir de ce diagramme que j'ai pu effectuer le début du code. Certaines partie et étapes on était obligatoire car en lisant la documentation du capteur BNO055 on a pu voir que par exemple à un moment on était obligé d'attendre 650ms après un reset sinon il risquait d'y avoir un problème.



Lecture trame I2C :

Pour visualiser la trame envoyer j'ai utilisé un analogue Discovery on a pu voir ce qui était envoyé. Et on peut voir que les données étaient cohérentes avec celle reçue.



De base l'idée de la lecture de trame avec un analogue était pour voir d'où le problème venait car avec une interface graphique du logiciel pycharm, lorsqu'on mettait le code en route il bugger alors avec l'analogue on a pu voir ou précisément le code planter. Finalement on a découvert que l'ordinateur avait un problème au niveau de l'USB, que l'alimentation n'était pas assez forte du coup le module(alimenter) planter. Ce bug m'a fait perdre pas mal de temps lors de mon projet car je pense que cela venait du code et non du pc.

Calcul de la distance :

Une chose était déterminante pour le calcul de la distance : Intervalle de temps entre 2 acquisitions. Voici le calcul à effectuer pour avoir la distance parcourue et on peut voir que la valeur que l'on ne possède pas est l'intervalle de temps.

D = v0 * t + (1/2) * a * t^2

Où :

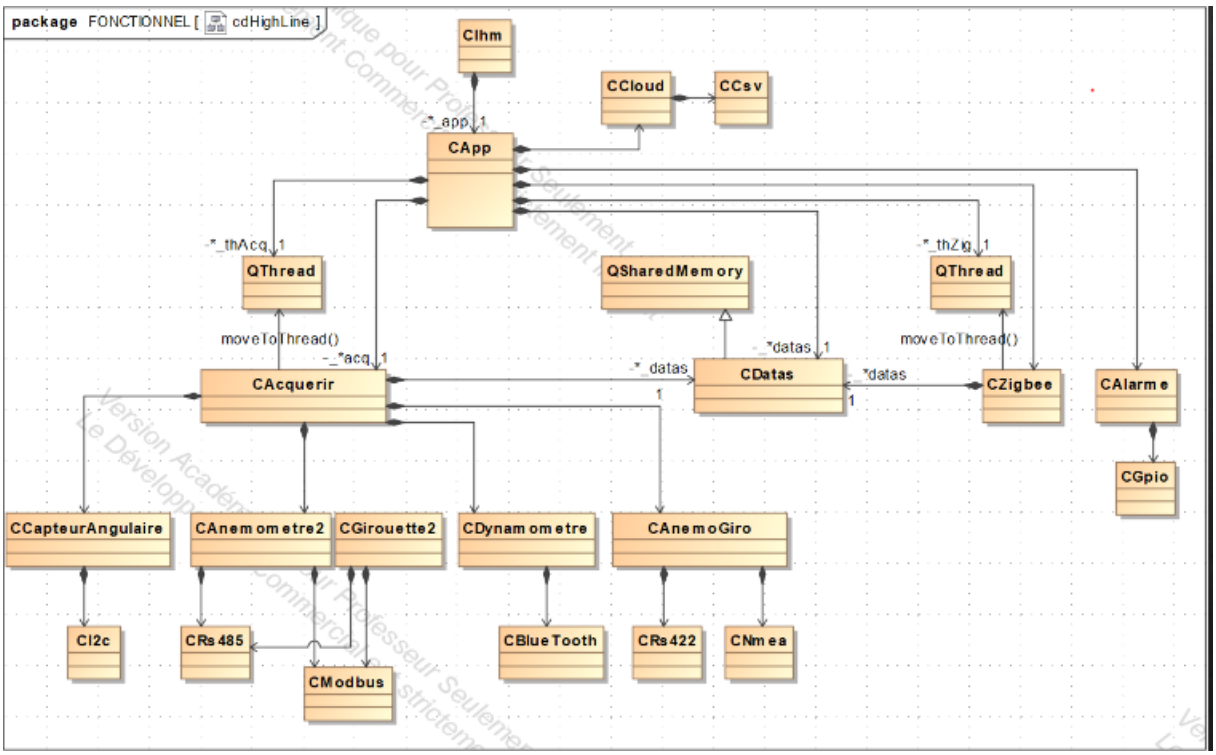
-d est la distance parcourue
-v0 est la vitesse initiale (supposée être nulle dans ce cas)
-t est l'intervalle de temps entre deux mesures
-a est l'accélération

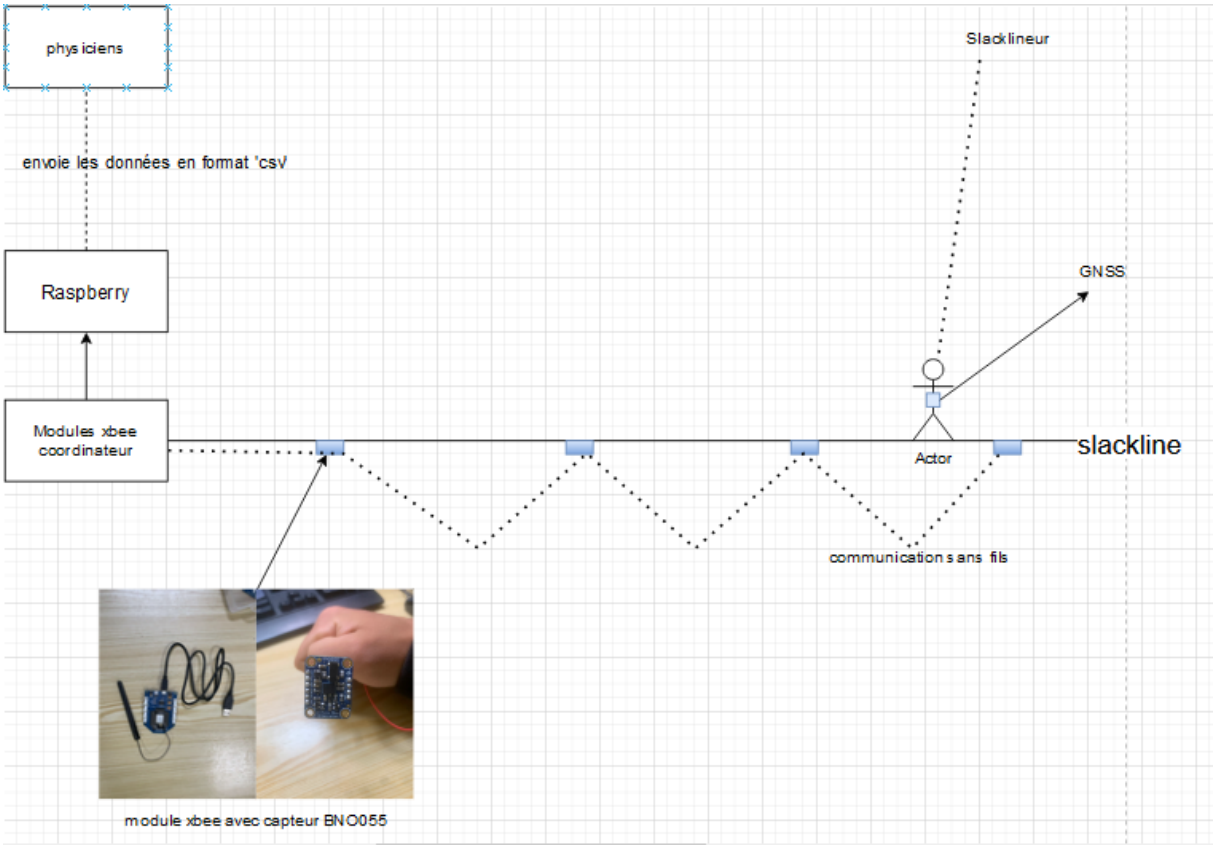
Et voici un code en micropython pour pouvoir le calcul de l'intervalle de temp entre deux mesure, on utilise la méthode "time.ticks_cpu" qui nous renvoie le nombre de cycles d'horloge du processeur depuis le démarrage du microcontrôleur:
Ce code nous permet d'avoir un dT qui mesure le temp que met le module à communiquer avec le capteur
Voici un exemple :

```
import time

t1 = time.ticks_cpu()

time.sleep(10)

t2 = time.ticks_cpu()

print ('dT={0}'.format(t2-t1))
```

QT/C++ :

Grâce à ce diagramme de classe on peut voir toutes les classes que l'on doit créer grâce à qt creator ce diagramme n'est pas finalisé il peut être modifié si on rajoute des classes par exemple. Ce diagramme comporte toutes les classes y compris celle de mes camarades ir qui travaille sur le projet aussi, et donc ma partie se concentre plus sur la droite car je travaille sur la classe CZigbee, aussi sur une partie de la classe Cdatas, CApp et CIhm qu'on se partage avec un camarade. C'est-à- dire quand dans cette classe il y aura ma partie mais aussi celle de mon camarade. Je suis entrain de travailler dessus et je n'ai pas fini c'est pour cela que je ne vais pas trop en parler lors de cette revue, mais on peut voir les différentes classes que je vais devoir créer ou ajouter ma partie. La partie QT se fera sur un Raspberry et lors du projet final la Raspberry sera connecté au module au bout de la ligne, elle servira de stockage et de passerelle entre les capteurs, module qu'il y aura sur la ligne et le physicien qui veut récolter les données acquises. C'est avec la Raspberry que la transformation dans données en format csv se fera normalement.





Voici par un schéma simplifié toutes les activités que je dois faire excepter pour le GNSS. On peut en conclure que les modules sont importants pour la récolte des données. Il y aura à peu près trente modules le long de la ligne ils seront placés à 100 mètres distance pour assurer une communication sans fils efficace. Après que les données soient remontées au coordinateur il y aura une communication par voie série à la Raspberry pour toutes les données seront envoyées aux physiciens en format csv ce qui est utile pour mettre les données dans un tableau Excel par la suite.

Partie individuelle
Ricard Loïc

Pour assurer la sécurité du highliner, je suis chargé de gérer la capture de la vitesse du vent ainsi que sa direction pour émettre un signal d'alarme si un certain seuil de déclenchement est dépassé.
Pour ce faire, je dispose du matériel suivant :
-Un anémomètre **DF Robot 7-24V RS485 Interface Modbus**
-D'une girouette **DF Robot 7-24V RS485 Interface Modbus**
-D'un **Capteurs Girouette, Anémomètre, Température sur bus NMEA**
-De différentes interfaces pour la conversion USB -> RS485 ou RS422 ->RS485

Je devrais en dernière partie de projet concevoir un hat-Rpi pour pouvoir inclure un module XBEE ainsi qu'un module GNSS, le hat-Rpi devra inclure quelques fonctionnalités comme le support du bus I2C.

Gestion du projet :

Voici le planning de Gantt réalisé avant de commencer le projet.

Projet	Mar 03/01/23	Ven 16/06/23			257 h?
Specifications générales	Mar 03/01/23	Mar 03/01/23			3 h
Documentation sur la transmission MODBUS RTU	Lun 09/01/23	Lun 09/01/23	11		3 h
Mise en oeuvre capteurs anémomètre et girouette PC	Mar 10/01/23	Lun 16/01/23	12		10 h
Mise en oeuvre capteurs anémomètre et girouette RPI	Mar 17/01/23	Mar 24/01/23	13		13 h
Documentation sur la transmission NMEA	Mar 24/01/23	Lun 30/01/23	14		6 h
Mise en œuvre le capteur aérien NMEA	Lun 30/01/23	Lun 06/02/23	15		10 h
Prototypage rapide et routage	Mar 07/03/23	Mer 12/04/23	8		60 h
Fabrication et essais	Mar 02/05/23	Ven 09/06/23	9		60 h
Fichier de fabrication PCB finalisés	Mer 12/04/23	Mer 12/04/23			0,43 jr?
Rédaction du dossier	Mer 30/11/22	Jeu 25/05/23			257 h

Voici la réalité :

Projet	257 h?	Mar 03/01/23	Ven 16/06/23	257 h?
Specifications générales	12 h	Mar 03/01/23	Mer 04/01/23	12 h
Documentation sur la transmission MODBUS RTU	4 h	Lun 09/01/23	Mar 10/01/23	4 h
Installation carteSD RPI	2 h	Mar 10/01/23	Mar 10/01/23	2 h
essaies anémométre sur RPI (logiciel constructeur)	4 h	Mar 10/01/23	Mer 11/01/23	4 h
mise en oeuvre girouette RPI	2 h	Lun 16/01/23	Lun 16/01/23	2 h
Essaie girouette avec commix	1 h	Mar 17/01/23	Mar 17/01/23	1 h
Essaies girouette avec commix	4 h	Mar 17/01/23	Mer 18/01/23	4 h
Mise à jour RPI et installation des logiciels	3 h	Mer 18/01/23	Lun 23/01/23	3 h
Mise en oeuvre QT Creator	16 h	Mar 24/01/23	Mer 01/02/23	16 h
mise en oeuvre RS485 -> UART	70 h	Lun 06/02/23	Mer 22/03/23	70 h
Prototypage rapide et routage	30 h	Lun 27/03/23	Mer 12/04/23	30 h
Fabrication et essais	60 h	Lun 01/05/23	Mer 07/06/23	60 h
Fichier de fabrication PCB finalisés	3 h	Mer 12/04/23	Mer 12/04/23	3 h
Rédaction du dossier	257 h	Mer 30/11/22	Jeu 25/05/23	257 h

La mise en œuvre des capteurs Modbus m'aura pris plus de temps que prévu pour la première partie mais par la suite j'ai quand même pu respecter les délais notamment la réalisation du schéma du hat rpi ainsi que du routage.

Mise en œuvre et essais des capteurs DF Robot

Présentation des capteurs :

Il faut savoir que les capteurs ont un fonctionnement et des caractéristiques identiques donc la présentation ci-dessous est la même pour les deux capteurs.

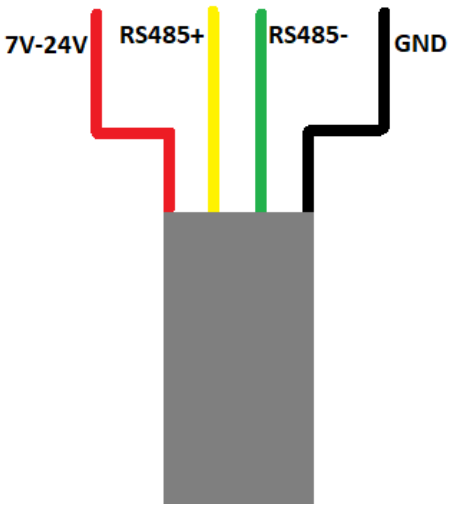
Les capteurs ont une plage d'alimentation entre 7V et 24V, ce qui les rend polyvalents et compatibles pour plusieurs systèmes notamment le nôtre qui sera alimenté en 12V pour une faible consommation d'environ 1 à 2 mA.

Spécificité de la Girouette :
Elle permet de mesurer jusqu'à 16 directions sur 360° et peut fonctionner avec un vent minimum de 1.08 km/h.

Spécificité de l'Anémomètre :
L'anémomètre peut mesurer le vent de 1.08 à 116.64 km/h avec une précision de ± 1.08 km/h.

Branchement des capteurs :

Les capteurs possèdent 4 fils de différentes couleurs :



Les capteurs fonctionnent en RS485 avec une interface modbus RTU.

A propos de la transmissions RS485 et modbus RTU :

RS485 :

Le RS485 est une liaison Half Duplex communicant en ASCII ou Binaire qui est souvent utilisée en industrie.
Par rapport au RS232 cette liaison permet la mise en service de jusqu'à 32 récepteurs sur une seule paire de fils ce qui simplifie grandement la communication entre plusieurs appareils. Dans notre cas, 4 modules Modbus seront mis en service sur la même ligne.

Le RS485 supporte une grande distance de communication qui est d'environ de 1200m et un débit jusqu'à 10 Mbauds sur une courte distance.
Cette liaison semble pertinente pour le projet highline grâce à sa simplicité de mise en service et à ses caractéristiques.

Modbus :

Le protocole Modbus est un système demande / réponses, en effet les esclaves vont seulement envoyer des données lorsque le maître leur demande.
Ce protocole permet donc de choisir quel esclave sera concerné par la demande.
La demande devra donc être composée de :

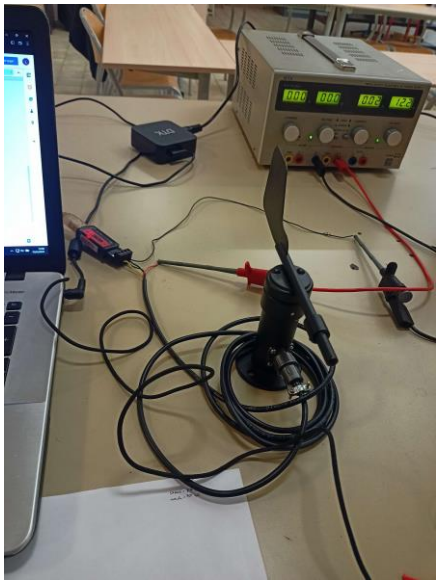
-L'adresse de l'esclave.
-Le code de fonction qui détermine l'action que le maître veut effectuer sur l'esclave (lire, écrire etc.)
-Deux octets d'adresse de départ qui spécifient l'adresse du premier registre de l'emplacement mémoire dans l'esclave que le maître veut lire ou écrire.
-Deux octets qui servent à spécifier le nombre de registres que le maître souhaite lire ou écrire.
-Un code CRC de deux octets pour la détection d'éventuelles erreurs.

La réponse sera composée de :

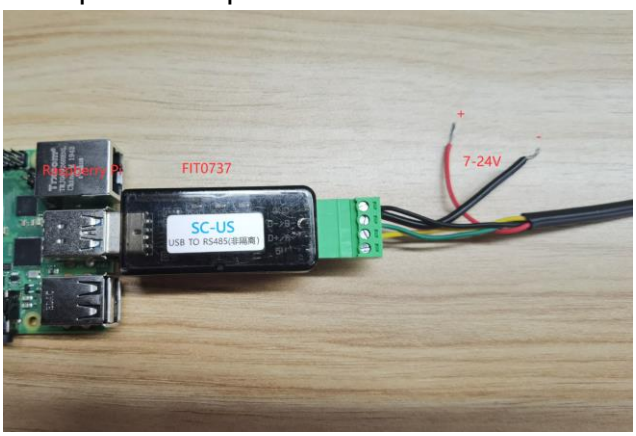
-L'adresse de l'esclave.
-Le code de fonction.
-Données transmises...
-Un code CRC.

Mise en œuvre des capteurs :

Après m'être renseigné sur la communication RS485 Modbus j'ai pu faire des essais en utilisant un adaptateur USB -> RS485 pour pouvoir faire une demande et voir la réponse du capteur.



J'ai donc branché la girouette à l'interface USB, j'ai dû alimenter le capteur avec une alimentation externe car l'interface USB fournie seulement une tension de 5V.
Schéma de branchement préconisé par le constructeur :



Pour les essais, je me suis inspiré de la documentation des capteurs sur le wiki de DFRobot.

Liens des wikis :

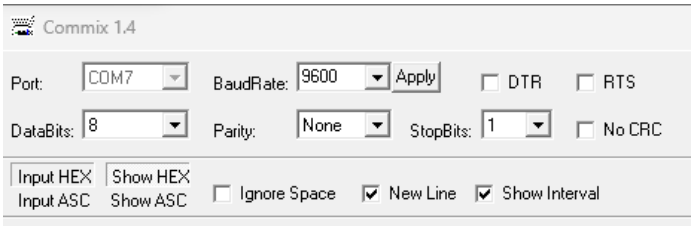
Girouette : https://wiki.dfrobot.com/SKU_SEN0482_RS485_Wind_Direction_Transmitter_V2

Anémomètre : https://wiki.dfrobot.com/RS485_Wind_Speed_Transmitter_SKU_SEN0483

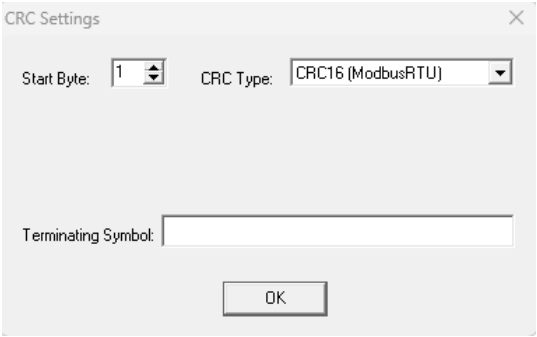
Essais des capteurs avec commix :

En utilisant un logiciel gratuit nommé “Commix” j’ai pu établir la communication entre la girouette et mon ordinateur.
Voici le lien de téléhargement de Commix : https://www.polier.fr/pages/question-reponse/modbus/logiciel.html

Configuration de commix :

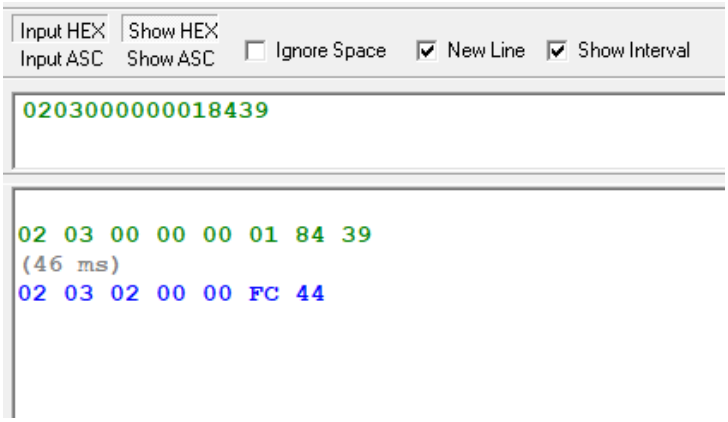


La transmission préconisée par le constructeur est une liaison en 9600 bit/sec, avec un bit de start, 8 bits de data, pas de parité et un bit de stop. J’ai donc configuré commix pour qu’il puisse respecter la norme définie par DFRobot. Le port COM7 correspond à l'interface USB.



Il ne faut pas oublier de configurer le CRC pour que logiciel le génère automatiquement et donc éviter toutes erreurs de communications.

Exemple de communication :



Étant donné de nous communiquons en HEX, il ne faut pas oublier de sélectionner “Input HEX” et “Show HEX”

Pour faire une requête, il faudra toujours envoyer la trame suivante :

Girouette :

0x02 0x03 0x00 0x00 0x00 0x01 0x84 0x39

Les caractères les plus importants sont le 0x02 qui correspond à l'adresse du capteur qui peut prendre une valeur de 0 jusqu'à 255, l'octet 0x03 est le code de fonction qui correspond à la lecture seul du capteur et les octets 0x00 0x00 0x00 0x01 qui sont les octets qui correspondent au registre et pour finir les deux derniers octets 0x84 0x39 qui composent le CRC.
Voici un exemple de réponse :

0x02 0x03 0x02 0x00 0x03 0xBC 0x45

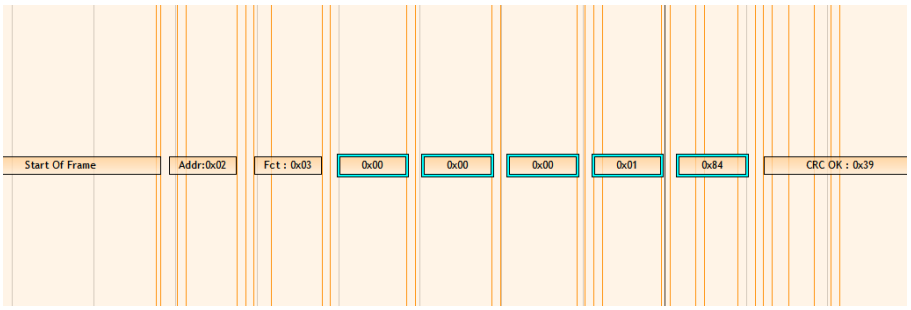
Nous retrouvons bien l'adresse du capteur 0x02 avec le code de fonction qui est identique à la demande. Le 0x00 0x03 sont les octets qui nous intéressent car ils correspondent à la valeur du capteur. Nous retrouvons ensuite les deux octets de CRC.

Anémomètre :

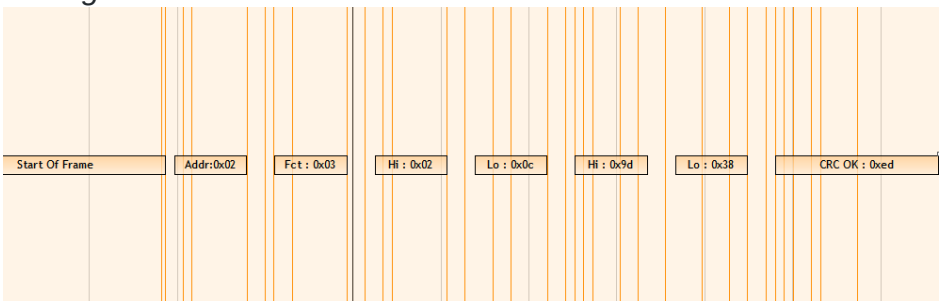
0x02 0x03 0x00 0x00 0x00 0x01 0x84 0x39

Nous pouvons constater que le message envoyé est identique à la girouette seulement l'adresse change.

Exemple de communications avec la girouette à l'adresse 2 :



Réponse de la girouette :



Pour les premiers essais, les capteurs étaient à la même adresse, dans l'application ils auront bien des adresses différentes.

Mise en œuvre sur RPI :

Maintenant que j'ai réussi à communiquer avec les capteurs via commix, il me reste à comprendre comment le faire sur une carte RPI sans utiliser le programme fait par le constructeur car celui-ci ne fonctionnait pas sur la version de OS de ma carte Raspberry 3 qui est la version Debian 11 bullseye.

.

Programme sous QT creator 4.14 :

Configuration de la liaison série :

```
if(USBModbus_is_available) {

    USBModbus = new QSerialPort();

    // configurer le port
    USBModbus->setPortName(USBModbus_port_name);
    USBModbus->setBaudRate(QSerialPort::Baud9600);
    USBModbus->setDataBits(QSerialPort::Data8);
    USBModbus->setParity(QSerialPort::NoParity);
    USBModbus->setStopBits(QSerialPort::OneStop);
    USBModbus->setFlowControl(QSerialPort::NoFlowControl);

    // Mettre en place le slot pour la réception
    connect(USBModbus, &QSerialPort::readyRead, this, &Dialog::onReadyRead);

    // Ouvrir le port
    USBModbus->open(QSerialPort::ReadWrite);



} else {
    // Message d'erreur si port non disponible
    QMessageBox::warning(this, "Erreur Port", "USB Modbus non trouvé");
    QCoreApplication::exit();
}
}
```

Cette partie du programme permet d'initialiser la communication, en réglant la vitesse à 9600 bit/s, le bit de parité etc.

Analyse de la trame (girouette) :

```
64  void Dialog::onReadyRead()
65  {
66      QByteArray infosGirouettes = USBModbus->readAll();
67
68      qDebug() << "car. recus : " << QString(infosGirouettes);
69
70      int rawAngle = (infosGirouettes[ 3 ] << 8) | infosGirouettes[ 4 ];
71      float angle = rawAngle / 10.0;
72
73      int portion = (int)(round(angle / 22.6));
74      if((portion < 0) || (portion > 15)) {
75          portion  = 0;
76      }
77
78      qDebug() << "portion : " << portion;
79
80      int direction = infosGirouettes[ 3 ];
81      const QString directions[] {
82        "N"
83          , "NNE"
84          , "NE"
85          , "ENE"
86          , "E"
87          , "ESE"
88          , "SE"
89          , "SSE"
90          , "S"
91          , "SSW"
92          , "SW"
93          , "WSW"
94          , "W"
95          , "WNW"
96          , "NW"
97          , "NNW"
98      };
99
100     qDebug() << "angle : " << angle << " / direction : " << directions[ direction ];
101     qDebug() << "angle : " << angle << " / portion : " << directions[ portion ];
102 }
103
104
```

Ici, il faut bien penser à diviser la valeur du capteur “rawAngle” par 10 car celle-ci est initialement donnée en dixième de degré.

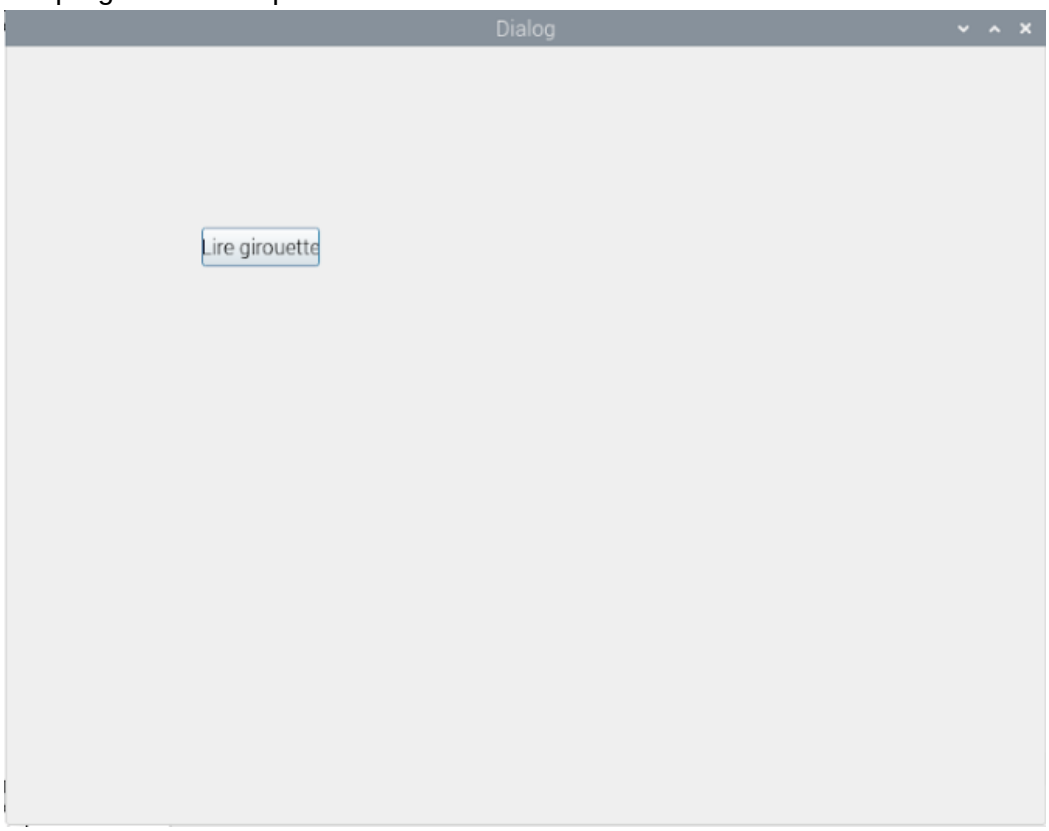
Envoie du message :

```
void Dialog::on_pushButton_clicked()
{
    if (USBModbus->isWritable()){

        static const char emission[]={'\x03','\x03','\x00','\x00','\x00','\x01','\x85','\xE8'};
        QByteArray emission1 = QByteArray::fromRawData(emission, sizeof (emission));

        USBModbus->write(emission1);
    }
    else{
        qDebug()<<"Impossible d'écrire sur interface USB Modbus";
    }
}
```

Nous retrouvons bien la trame que nous souhaitons envoyer dans la fonction “static const char emission[]”.

Grâce à ce programme, nous pouvons donc lire l'angle de la girouette quand nous le souhaitons et autant de fois que nécessaire grâce au bouton "Lire girouette".

Rendu du programme sur rpi :



Console du programme :

```
09:23:32: Starting /home/pi/build-girouette-Desktop-Debug/girouette ...
Port :  "ttyAMA0"  / Description :  ""
Port :  "ttyUSB0"  / Description :  "1a86"
libEGL warning: DRI2: failed to authenticate
car. reçus :  "\u0003\u0003\u0002"
portion :  1
angle :  15.9  / direction :  "N"
angle :  15.9  / portion :  "NNE"
```

Nous pouvons donc avoir les informations sur les ports de communication disponibles sur la RPI, l'angle de la girouette ainsi que sa direction en respectant le tableau fourni par le constructeur.

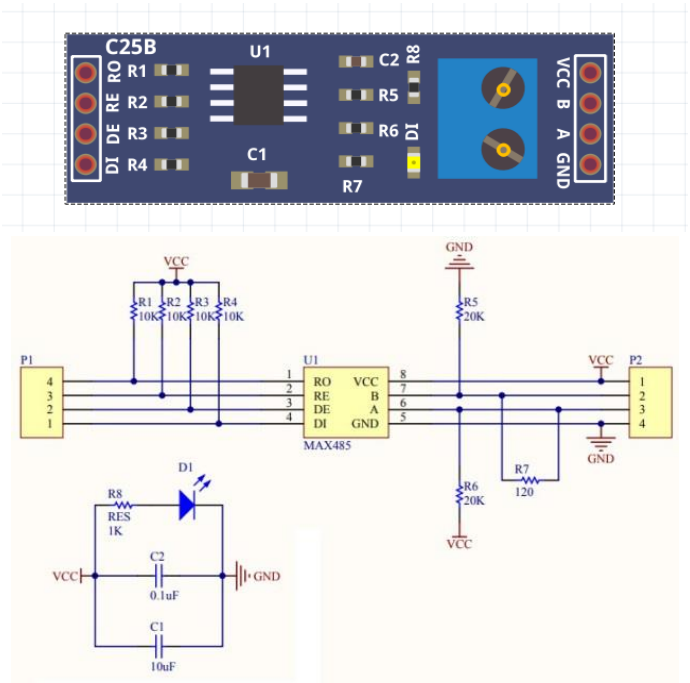
Il faut savoir que pour le moment, l'interface graphique permet également d'envoyer une trame sur demande, pour lire le résultat obtenu, il faudra forcément regarder la console du programme.

Ce programme a été conçu pour fonctionner avec la girouette, mais l'adapter pour l'anémomètre sera plutôt simple.

Utilisation du module MAX485 TTL to RS485 :

Comme nous utilisons la liaison UART de la Raspberry pi, il me faut maintenant utiliser un module pour adapter le signal UART en RS485 TRU.

Ce module fonctionne en 5V et permet de travailler avec des signaux allant jusqu'à 2.5Mps ce qui est largement suffisant pour notre utilisation.



Le module est divisé en 2 parties, la partie TTL et la partie RS485 :

Partie TTL :

La broche RO correspond à la broche RX du Raspberry, la broche DI à la broche TX du Raspberry.
Enfin les broches RE et DE sont les broches de contrôle du module, en effet il ne peut pas lire et envoyer des données simultanément, il faut donc lui dire quoi faire.

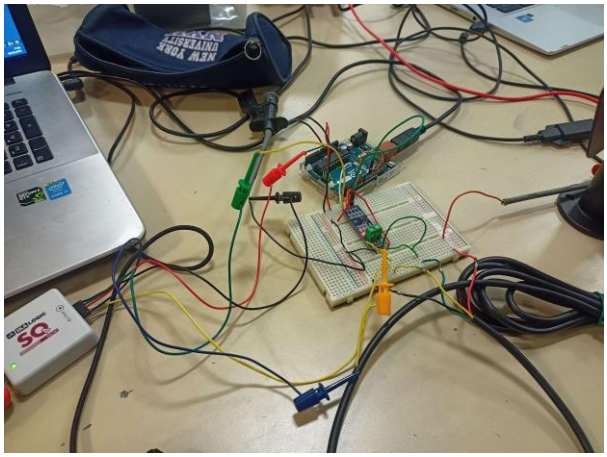
Si les broches RE et DE sont à 1 (5V) : Le module va être en mode transmission
Si les broches RE et DE sont à 0 (0V) : Le module va être en mode réception

Partie RS485 :

Cette partie est plutôt simple puisqu'il y a seulement les broches A et B du signal RS485 ainsi que l'alimentation en 5V (VCC) accompagnée de la masse (GND).

Pour les premiers essais, j'ai utilisé une carte Arduino en réalisant le montage suivant :

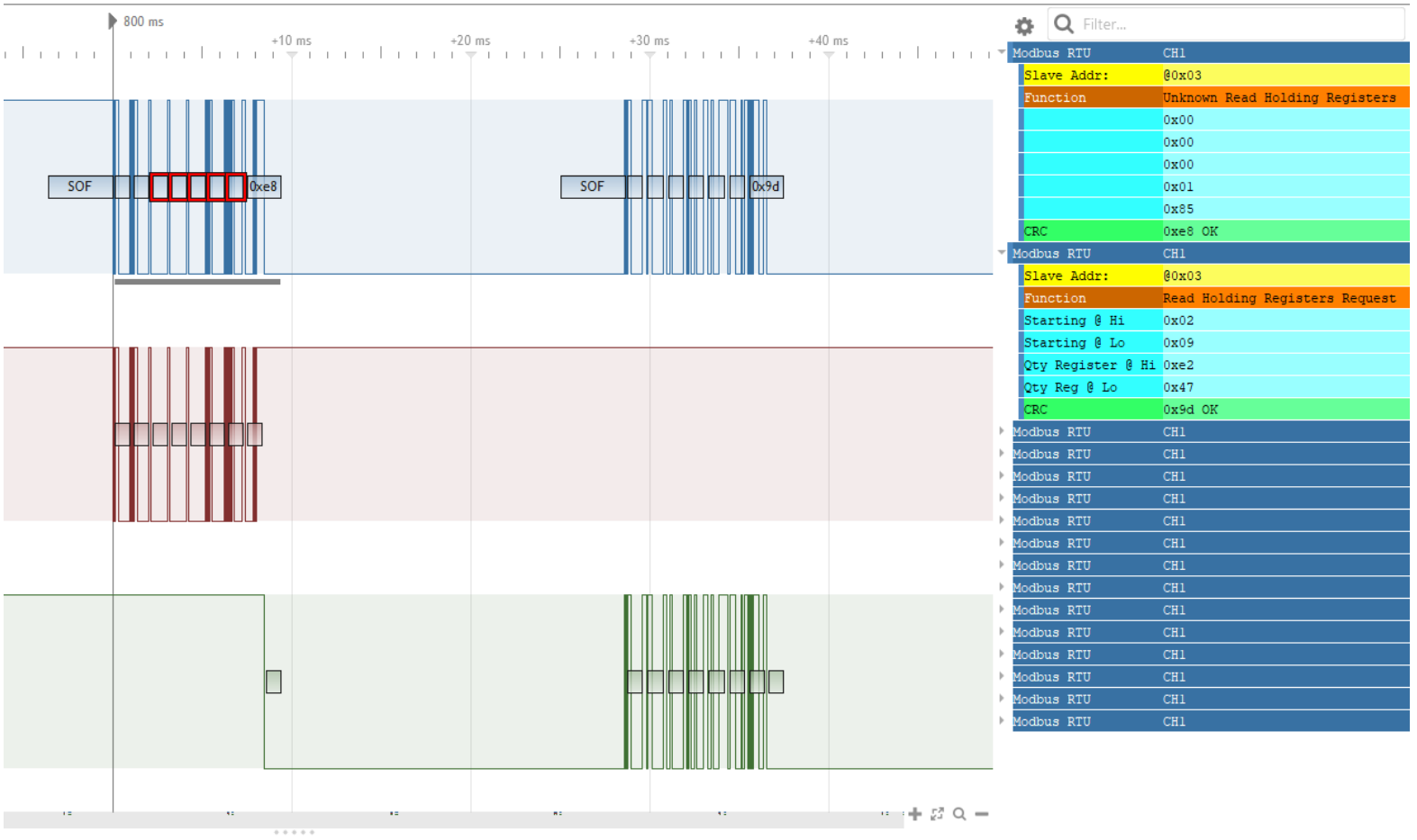
Photo du montage :



Programme sur Arduino :

```
#include <SoftwareSerial.h>

#define RS_RO   10
#define RS_DI   11
#define RS_DE_RE   12

SoftwareSerial RS_Master (RS_RO,RS_DI); //RX,TX
void setup() {
//configuration de la communication série
Serial.begin(9600);
RS_Master.begin(9600);
pinMode (RS_DE_RE, OUTPUT);
}

void loop() {
const byte val [] = {0x03, 0x03, 0x00, 0x00, 0x00, 0x01, 0x85, 0xE8}; //donnée à envoyer

if (Serial.available()) {
  while(true){
digitalWrite (RS_DE_RE,HIGH); // Mise des broches DR_RE à l'état haut = transmission
RS_Master.write(val, sizeof val);// Envoie de la donnée
digitalWrite (RS_DE_RE,LOW); // Mise des broches DR_RE à l'état bas = réception
delay(1000);
  }
}
if(RS_Master.available())
Serial.write(RS_Master.read()); // Lecture de la trame reçu
}
```

Analyse de trames :

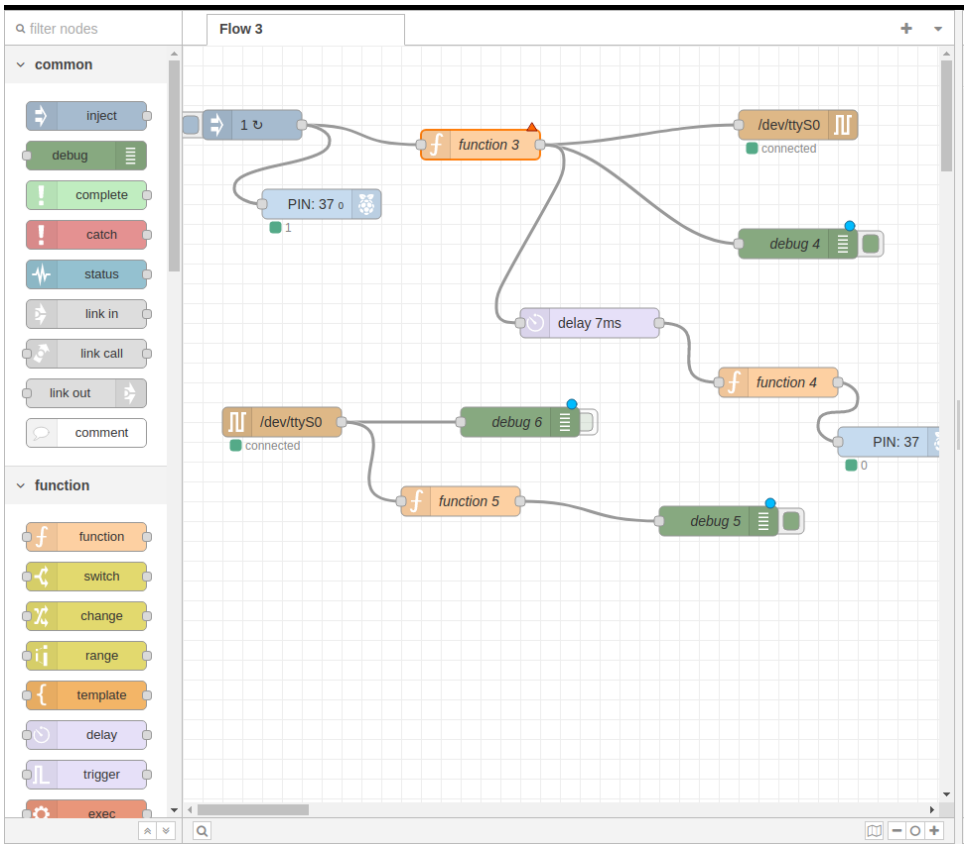


Nous pouvons voir que la transmission s'est bien réalisée avec en premier temps l'envoie de la trame 0x03 0x03 0x00 0x00 0x00 0x01 0x85 0xE8 puis quelques millisecondes plus tard, la réponse de la girouette.

Mise en œuvre du module MAX485 TTL to RS485 sur Raspberry :

Maintenant que je connais le fonctionnement du module, j'ai dû adapter le programme de l'Arduino sur le Raspberry pi, pour ce faire j'ai décidé d'utiliser Node Red, un outil de développement gratuit et plus intuitif que Qt Creator, il sera amplement suffisant pour les essais que je vais réaliser dans le futur .

Voici le programme :



Le programme est divisé en 2 parties, la partie envoie de données et la partie réception.

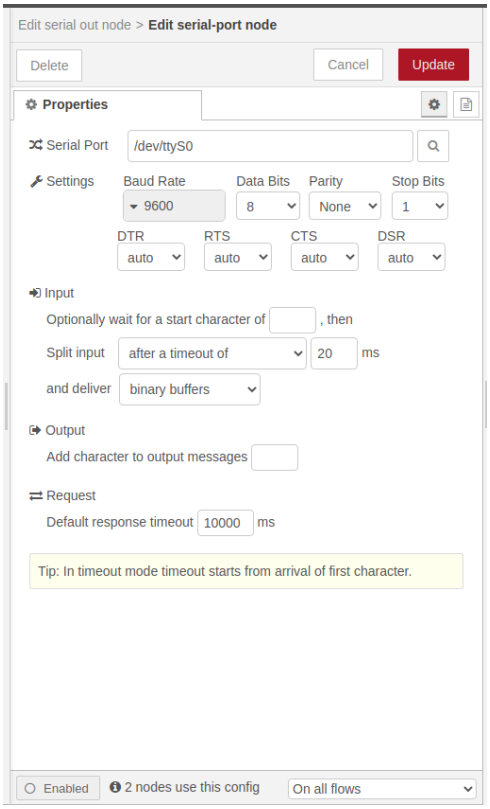
Partie envoi de données :

La fonction 3 permet d'envoyer la trame que nous voulons sur le port série de la Raspberry /dev/ttyS0.

Voici le code de la fonction :

```
frame = [0x03, 0x03, 0x00, 0x00, 0x00, 0x01, 0x85, 0xe8];
msg.payload = Buffer.from(frame);

return msg;
```

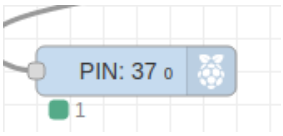
Le port série /dev/ttyS0 est configuré de la manière suivante :

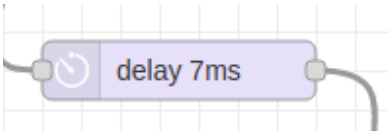


Nous pouvons y retrouver les paramètres de base comme la vitesse de communication, les bits de datas etc.

Enfin nous contrôlons l'état des broches RE et DE grâce à un module qui permet le changement d'état d'une PIN, j'ai utilisé la PIN37 mais en théorie toutes les PIN peuvent être utilisées.

Voici les fonctions correspondantes :





Ce sont des fonctions programmables intégrées à nodered, il n'y a donc aucun code.

Au début du programme, la PIN37 est à l'état haut, puis 7 ms après l'envoi de la trame elle passe à l'état bas. Les 7 ms de délai correspondent au temps que met la trame pour être envoyée sur le bus, comme le capteur met une vingtaine de millisecondes à répondre, nous avons laissé une marge de quelques ms pour être sûr que la trame à bien été envoyée.

Partie réception des données :

Cette partie est composée d'une seule fonction qui permet la lecture et l'analyse des données réceptionnées par le port série /dev/tty0.

Les debugs servent à afficher les valeurs envoyées et réceptionnées dans la console :

Ici, le débug 4 (celui du haut) correspond à la trame envoyer par le Raspberry pi, puis le debug 5 (celui du bas) analyse la réponse donnée par la girouette en donnant la direction et l'angle.

Maintenant que j'ai un programme fonctionnel sur la carte Raspberry pi, il ne me manque plus qu'à développer un hat-rpi pouvant accueillir le MAX 485 ainsi que les autres éléments de mon contrat comme le support du module GNSS ainsi que du module XBEE avec un buzzer qui se déclenchera en cas du dépassement d'un seuil fixé au préalable.

A propos du capteur NMEA :

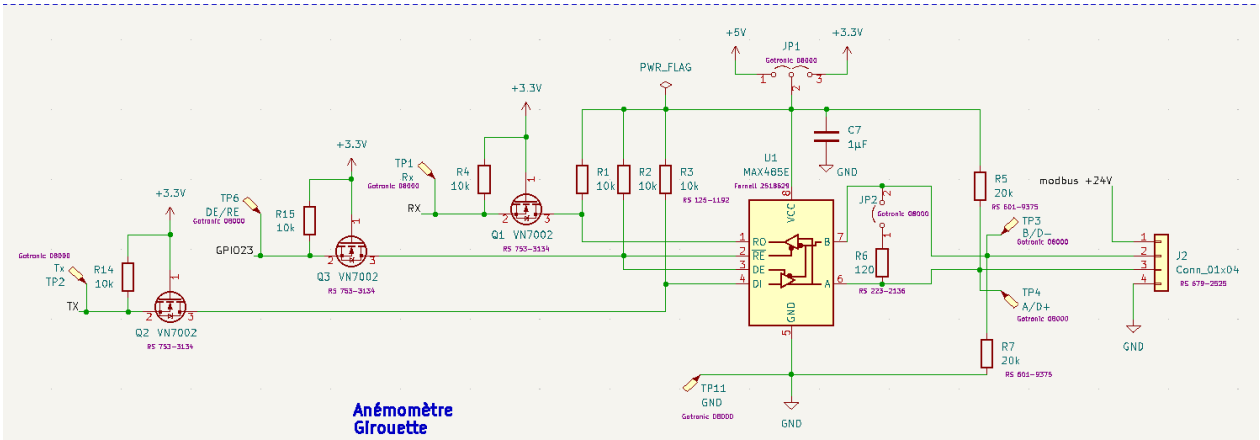
Par faute de temps et de matériel, nous avons décidé pour cette année de ne pas mettre en œuvre le capteur NMEA, l'ajout de ce capteur sera surement possible si le projet est amélioré l'année prochaine.

Schéma et routage sur KiCad :

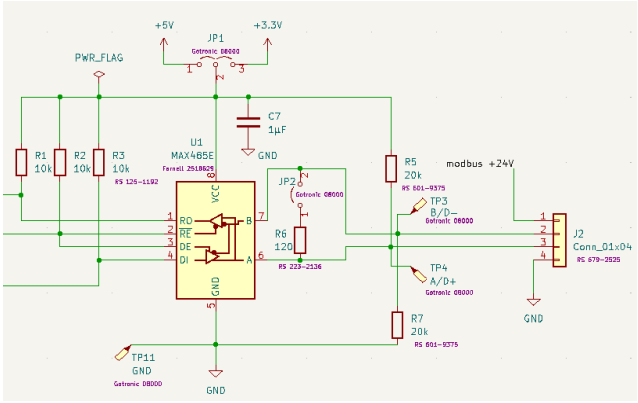
Pour le développement du Hat-Rpi et pour intégrer les différents modules, nous avons dû suivre et adapter les schémas proposés par les constructeurs.

Le hat se divise donc en 6 parties :

La première partie concerne le module MAX485 qui permet donc l'adaptation bidirectionnelle des signaux UART en RS485.



Voici la structure globale de cette partie, mais je vais détailler certains éléments ci-dessous :

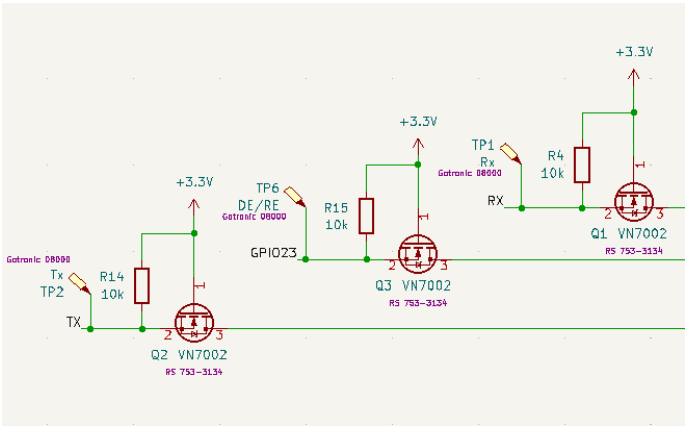


Pour cette partie, je me suis inspiré du schéma du module MAX485 TTL to RS485, nous avons cependant laissé la possibilité du choix d'alimentation de la puce car durant les essais j'ai remarqué qu'elle fonctionnait bien en 3.3V ou 5V (même si la tension d'alimentation préconisée par le constructeur est 5V).

Nous avons aussi décidé de mettre le MAX485 sur support pour pouvoir mettre un autre modèle, par exemple le MAX3485 si nous devons passer sur une tension de 3.3V

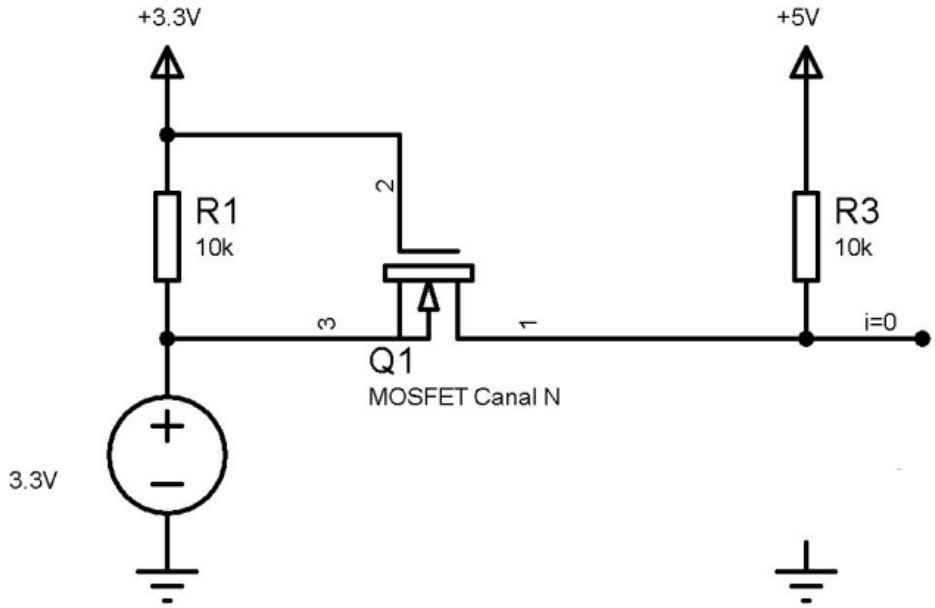
Il y a aussi un jumper pour pouvoir si nous le souhaitons, mettre la résistance de 120 Ohms pour adapter l'impédance de la ligne. Nous avons décidé d'utiliser un câble DMX 512 XLR 5 car ce sont des câbles fiables même sur de longues distances et certains câbles comme le nôtre sont prévus pour véhiculer des signaux numériques.

Nous avons ici utilisé un câble de 30m, la communication est bien effectuée sans aucun problème.

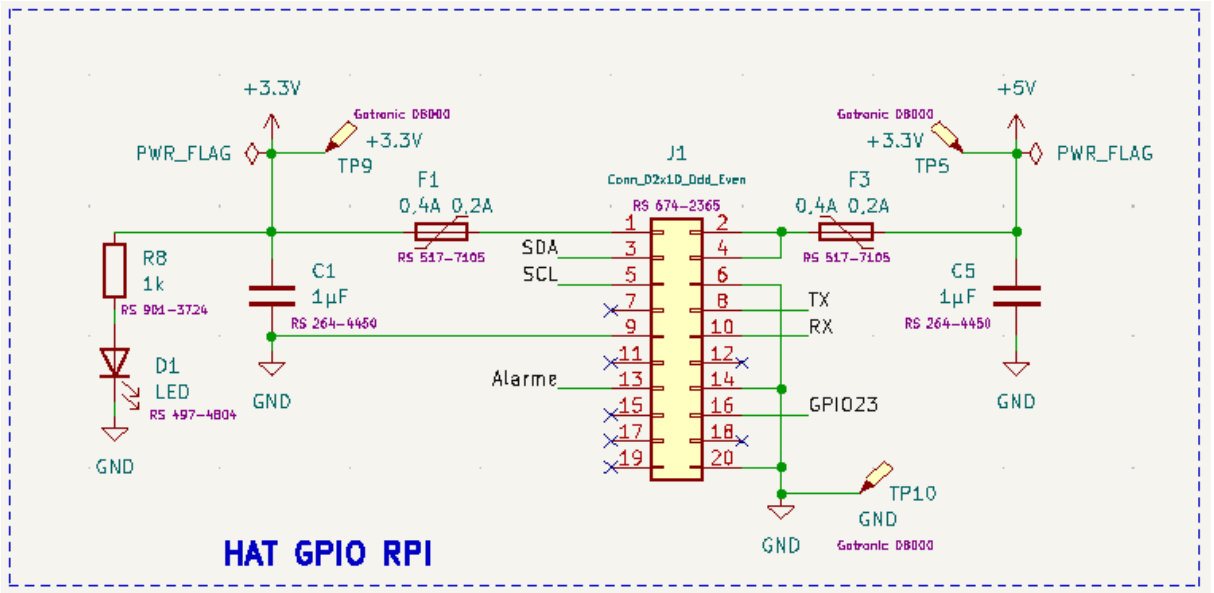


J'ai dû intégrer un level shifter pour pouvoir adapter la tension des broches du RPI qui sont en 3.3V avec celle du module MAX485 qui elles sont en 5V.

Voici le schéma du level shifter :

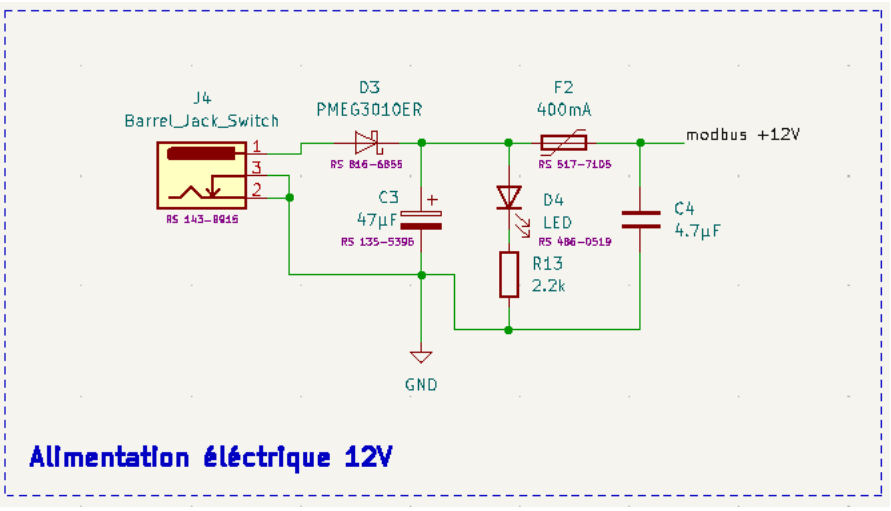


La deuxième partie concerne le connecteur GPIO 20 broches du hat rpi :

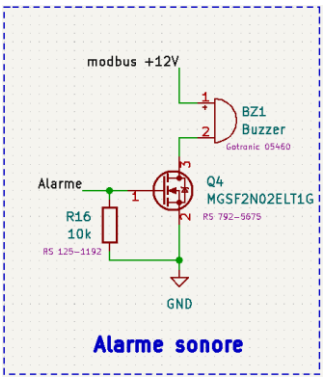


Nous retrouvons donc les alimentations 3.3V et 5V avec un PolyFuse pour protéger la carte en cas de surconsommation.
Il y a bien évidemment toutes les connexions nécessaires pour le fonctionnement des modules comme les liaisons UART ou I2C et une led pour indiquer que la carte est bien alimentée ainsi qu'une sortie logique "alarme" reliée à un buzzer qui pourra être déclenché en cas d'une valeur anormale de la vitesse du vent par exemple.

Pour finir avec les alimentations, il y a aussi une partie qui permet l'alimentation des capteurs Modbus :

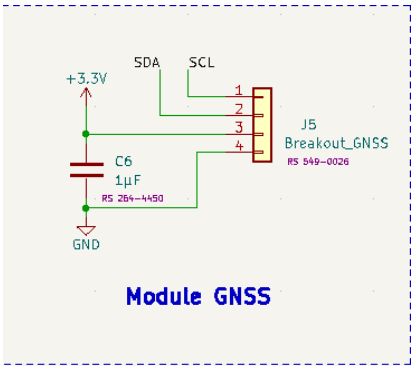


Il y a bien évidemment tous les composants nécessaires pour la protection du Hat-Rpi.

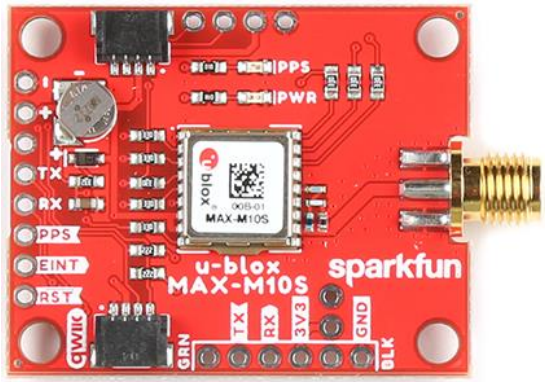


J'ai décidé d'alimenter les capteurs en 12V pour être sûr de bien alimenter les capteurs avec n'importe quelle ligne, et aussi pour être compatible avec le buzzer utilisé.

La quatrième partie concerne le module GNSS :

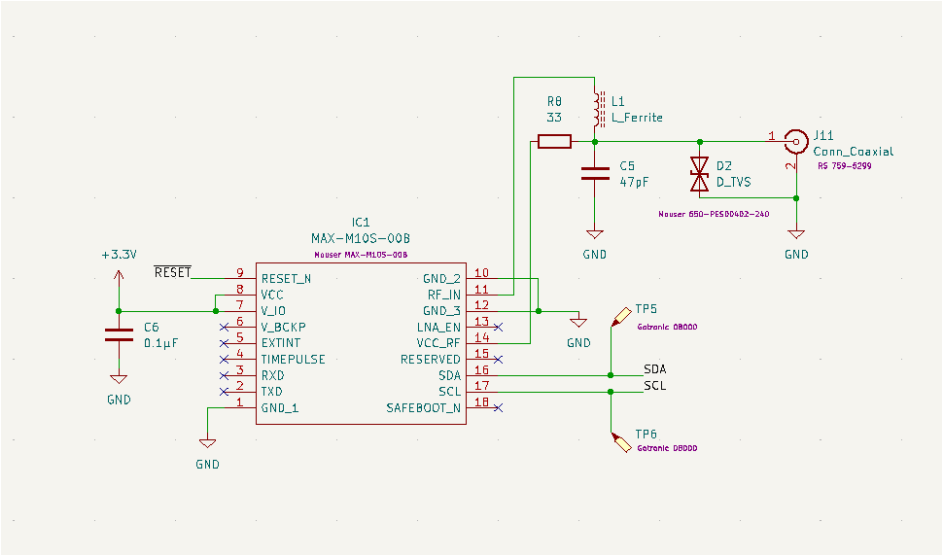


Pour simplifier la carte nous avons décidé de ne pas directement intégrer le module GNSS sur le Hat-Rpi. Nous avons juste prévu l'emplacement pour pouvoir y brancher la carte GNSS.

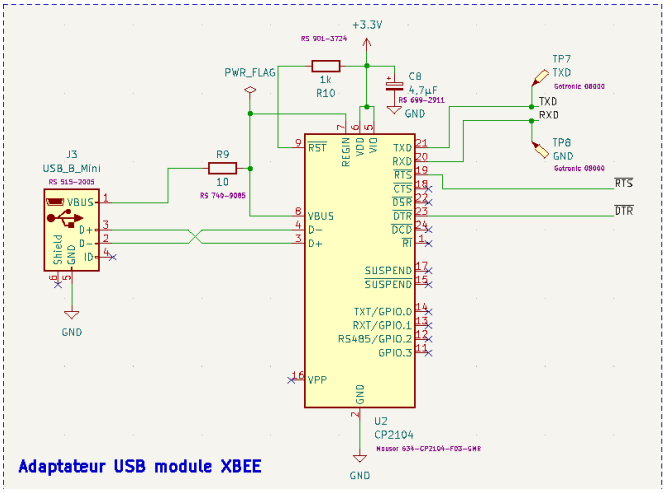


Voici une photo du module GNSS, l'analyse de ce break out ne fait pas partie de mon contrat, je dois juste intégrer sur mon Hat-Rpi et vérifier que la liaison I2C est bien fonctionnelle.

Voici ci-dessous le schéma initialement prévu pour l'intégration du module GNSS sur le Hat-Rpi :

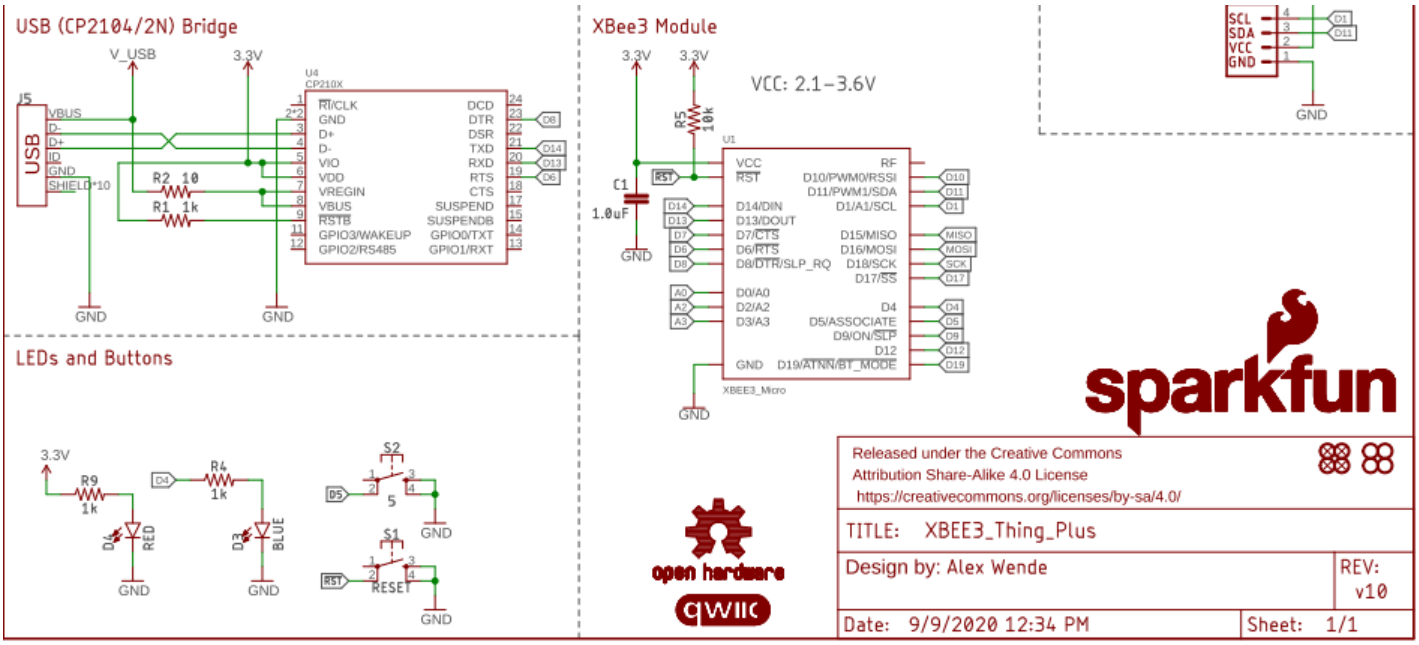


Enfin les deux dernières parties concernent l'intégration du module XBEE sur la carte :

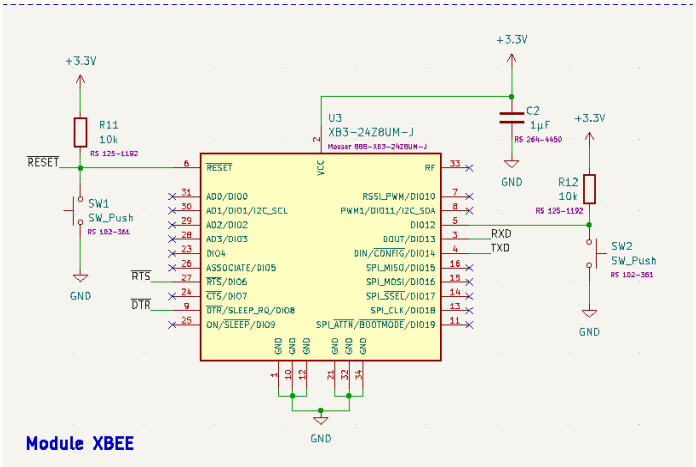


Comme le module XBEE utilise la liaison UART et que la Raspberry pi ne possède qu'un seul bus de communication, nous avons dû utiliser un adaptateur USB vers UART, nous nous sommes donc inspirés d'une carte disposant du module XBEE de chez Sparkfun.

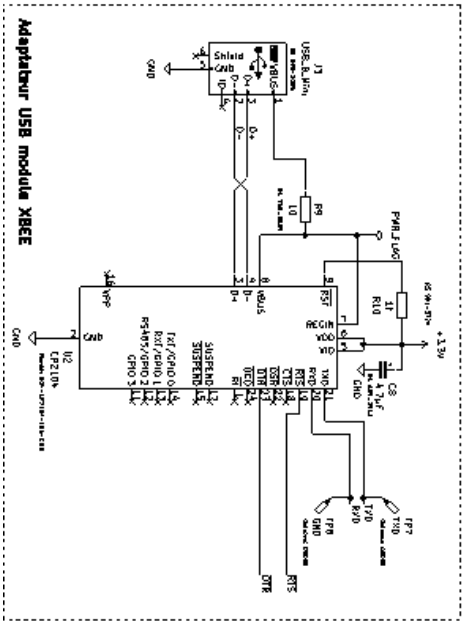
Voici les schémas en question :

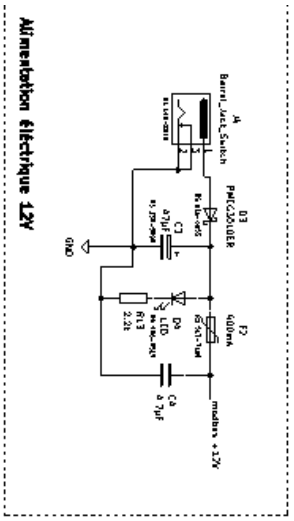


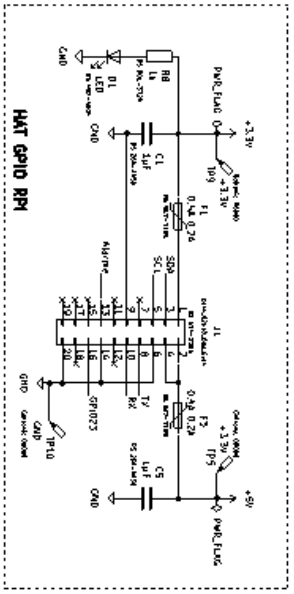
Pour finir avec les schémas voici l'intégration du module XBEE :

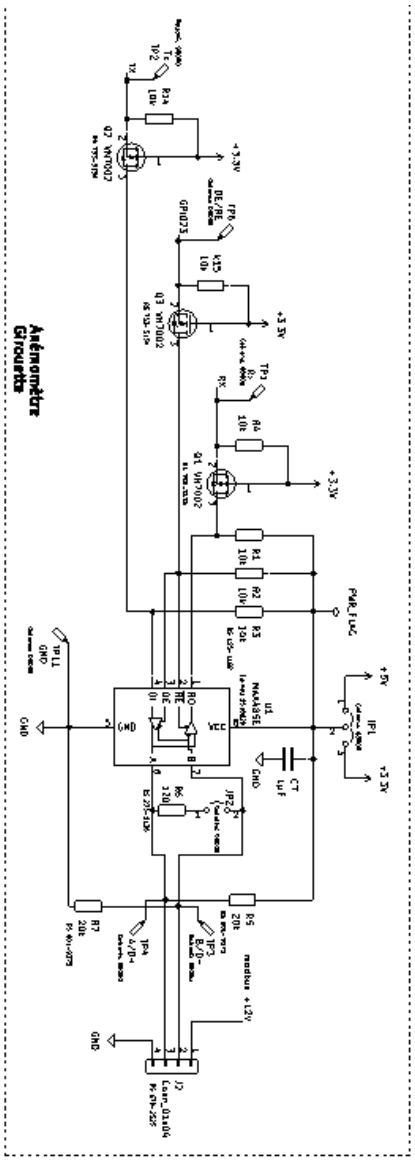


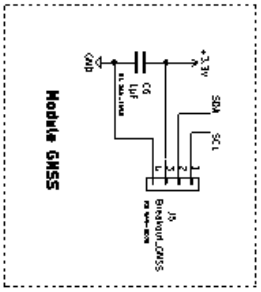
L'analyse du module XBEE ne fait pas partie de mon contrat.

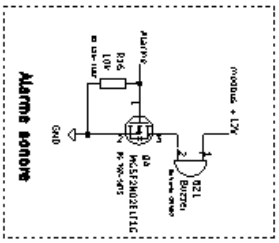

Adaptateur USB module XBEE
Shield
GND
D+
D-
VBUS
PWR_FLAG
RST
REGIN
VCC
VIO
TXD
RXD
RTS
CTS
DSR
DTR
RI
GND
SUSPEND
SUSPENDN
TXT/GPIO 0
RXT/GPIO 1
RS485/GPIO 2
GPIO 3
TP6
GND
TP7
TXD
RXD
RTS
DTR

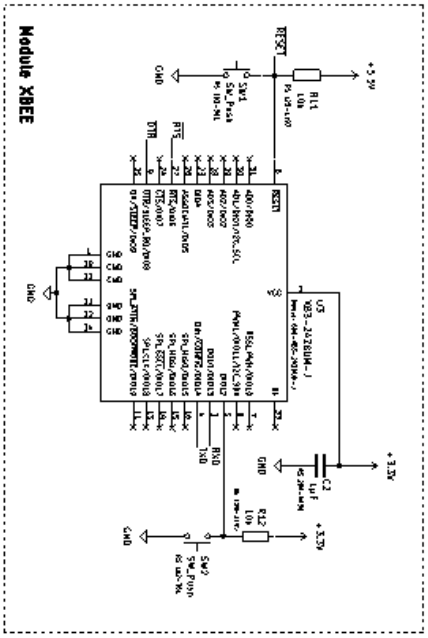

Alimentation électrique 12V
Barrel_Jack_Switch
PWR_FLAG
GND
F2
400mA
modbus +12V
C4
4.7µF


HAT GPIO RPi
PWR_FLAG
+3.3V
GND
D1
LED
R8
C1
1µF
F1
0.5A 0.2V
SDA
SCL
Alarme
J1
GPIO23
TX
RX
F3
0.5A 0.2V
C5
1µF
TP5
+5V
PWR_FLAG
TP10
GND

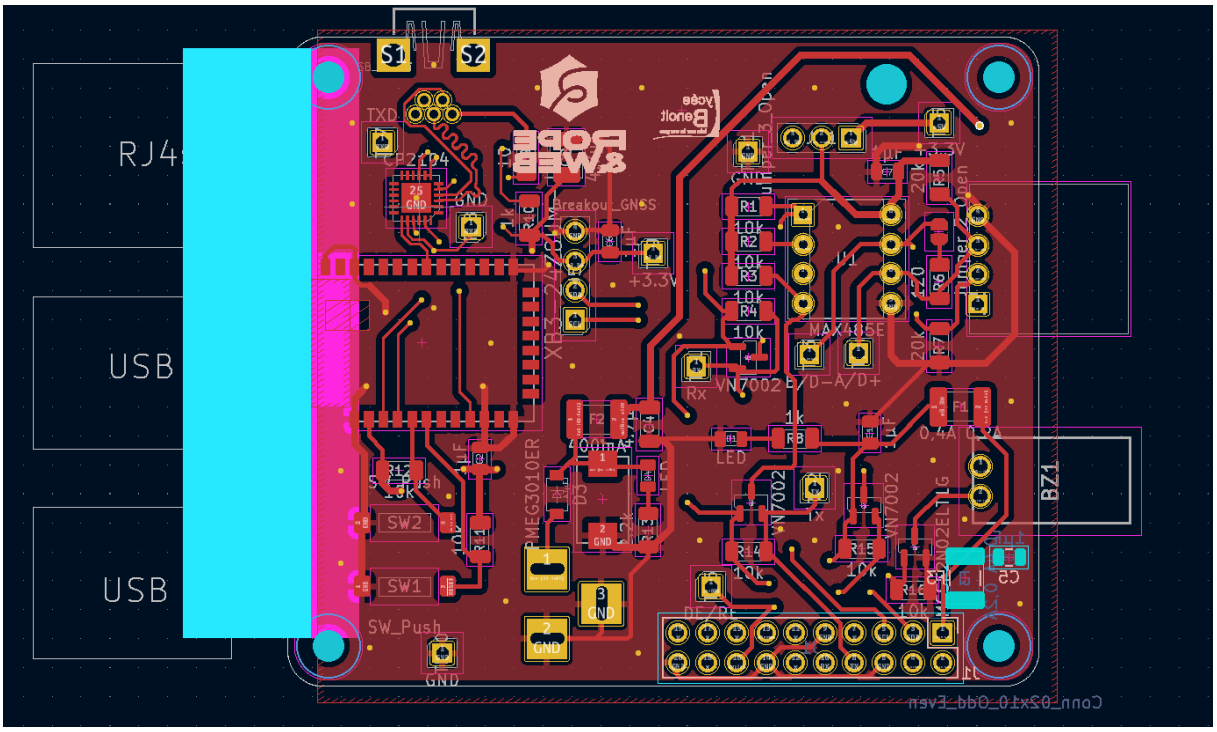

Anémomètre
Girouette
+3.3V
GPIO23
R4
10k
Q2
2N7002
R5
10k
Q3
2N7002
R6
10k
Q1
2N7002
R1
10k
R2
10k
R3
10k
PWR_FLAG
U1
MAX485E
RO
RE
DE
DI
GND
VCC
A
B
C7
1uF
R7
20k
R8
120
JP2
R5
20k
TP4
4.70
TP3
8.20
modbus +12V
J2
Conn_01x04
TP11
GND
IP1
+5V
+3.3V


Module GNSS
+3.3V
GND
C6
1µF
SDA
SCL
J5
Breakout_GNSS

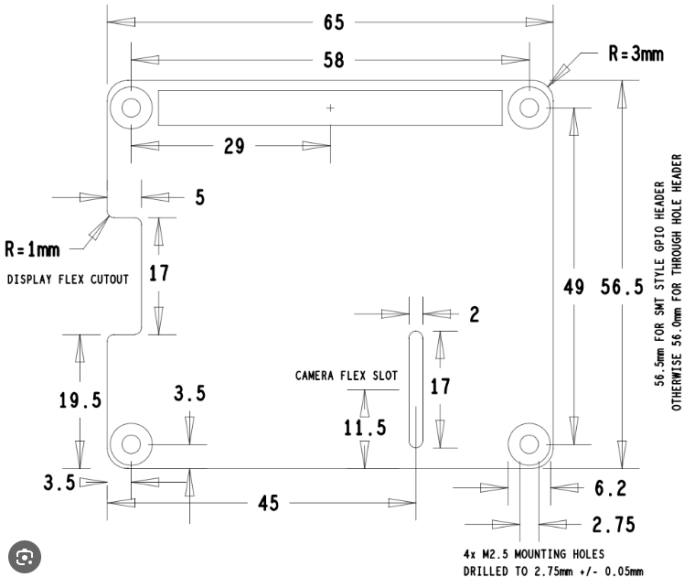

Alarme sonore
modbus +12V
Alarme
R16
10k
Q4
BUZ21
D21
Buzzer


Module XBEE
RESET
+3.3V
R11
10k
SW1
SW_Push
GND
DTR
RTS
RESET
ASSOCIATE/DIO5
GPIO
RTS/DIO7
CTS/DIO8
DIO10/PWM0
PWM1/DIO11/RSSI
DIO12
SPI_MOSI/DIO14
SPI_MISO/DIO15
SPI_SCK/DIO16
SPI_SS/DIO17
DIO18/SPI_ATTN
DIO19/SPI_CLK
DIN/CONFIG/DIO13
TXD
RXD
R12
10k
C2
1µF
SW2
SW_Push
U3
XB3-24ZB8UM-J

Voici le routage de la carte :

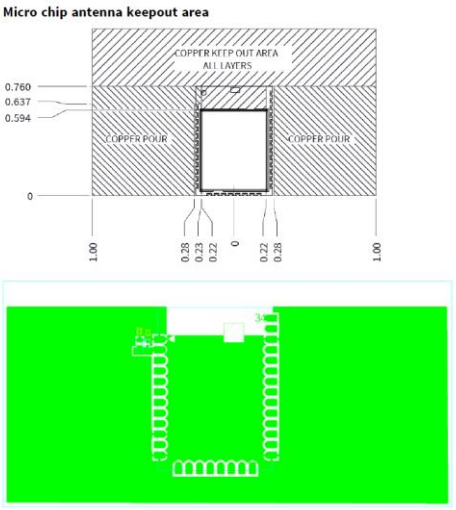


J’ai dû respecter certaines contraintes comme la taille de la carte :



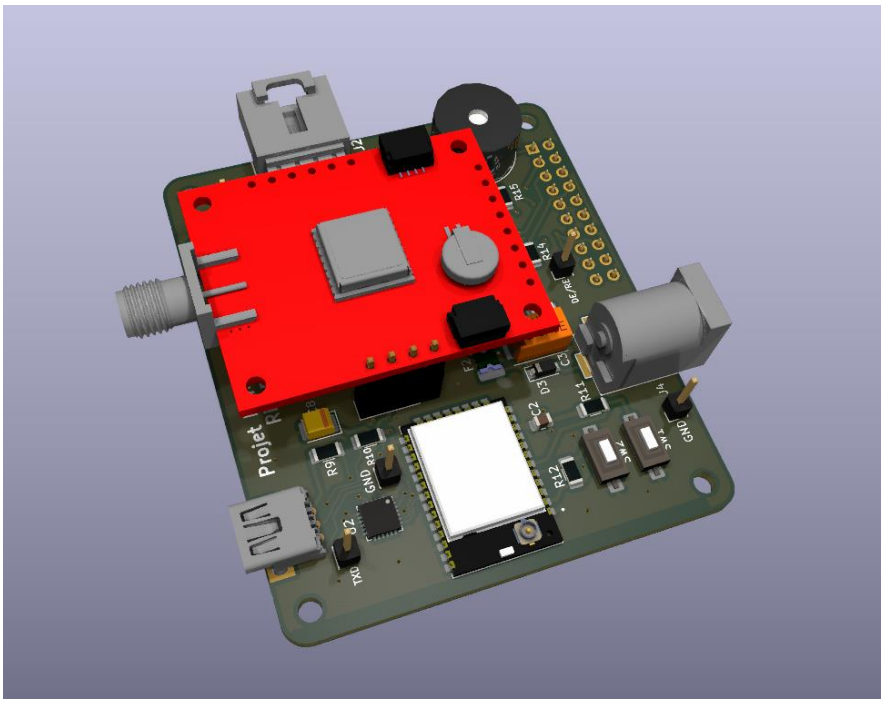
Il y a aussi un plan de masse en TOP et BOTTOM, j’ai dû aussi faire attention à l'accessibilité des connecteurs en les disposant sur les côtés de la carte.

J’ai aussi décider de mettre le module XBEE sur le bord de la carte pour éviter de perdre beaucoup de place car les constructeurs demandent de respecter des zones sans cuivre (zone en bleu) au niveau du module pour éviter des perturbations :



J’ai aussi ajouté différents points de test pour pouvoir analyser les différents signaux et les différentes tensions

Voici la prévisualisation de la carte :



Voici la liste des composants avec le prix estimés de ma carte, sans compter les frais de fabrication du PCB et des transports :

Référence	Valeur	Code Commande	Quantité	Prix
BZ1	KPEG242	Gotronic 05460	1	5,9
C3	47µF	RS 135-5396	1	0,61
C4	4.7µF	RS 691-1224	1	1,309
C1, C2, C5, C6, C7	1µF	RS 264-4450	5	0,28
C8	4,7µF	RS 699-2911	1	0,4
D1	LED	RS 497-4804	1	0,389
D4	LED	RS 486-0519	1	0,293
D3	PMEG3010ER	RS 816-6855	1	0,464
F2	400mA	RS 517-7105	1	0,41
> F1, F3	0,4A 0,2A	RS 517-7105	2	0,82
J1	Conn_02x10_Odd_Even	RS 674-2365	1	2,08
J2	Conn_01x04	RS 679-2525	1	1,348
J3	USB_B_Mini	RS 515-2005	1	2,6
J4	Barrel_Jack_Switch	RS 143-8915	1	3,667
J5	Breakout_GNSS	RS 549-0026	1	4,76
JP1	Jumper_3_Open	Gotronic 08000	1	-
JP2	Jumper_2_Open	Gotronic 08000	1	-
> Q1-Q3	VN7002	RS 753-3134	3	INCONNU
R6	120	RS 223-2136	1	0,103
> R5, R7	20k	RS 601-9375	2	0,07
> R8, R10	1k	RS 901-3724	2	0,08
R9	10	RS 740-9085	1	0,066
R13	2.2k		1	
> R1-R4, R11, R12, R14, R15	10k	RS 125-1192	8	0,28
> SW1, SW2	SW_Push	RS 102-361	2	1,364
TP1	Rx	Gotronic 08000	1	0,6
TP2	Tx	Gotronic 08000	1	-
TP3	B/D-	Gotronic 08000	1	-
TP4	A/D+	Gotronic 08000	1	-
> TP5, TP9	+3.3V	Gotronic 08000	2	-
TP6	DE/RE	Gotronic 08000	1	-
TP7	TXD	Gotronic 08000	1	-
> TP8, TP10, TP11	GND	Gotronic 08000	3	-
U1	MAX485E	Farnell 2518629	1	8,088
U2	CP2104	Mouser 634-CP2104-F03-GMR	1	4,1
U3	XB3-24Z8UM-J	Mouser 888-XB3-24Z8UM-J	1	20,03
				60,111

Une fois les composants et le PCB réceptionnés, il me reste à souder tous les composants puis à effectuer tous les tests nécessaires dans différentes configurations pour être sûr du bon fonctionnement de la carte.

Procédure de câblage :

Je vais commencer par souder les composants CMS du côté BOTTOM car ils sont peu nombreux donc plus facile à souder.

Après le passage au four à refusion, je souderai les composants CMS du côté TOP en commençant par placer les plus petits composants comme les résistances, condensateurs, transistors et le module XBEE en partant du centre de la carte jusqu'aux extrémités. Pour finir je souderai le CP2104 car c'est le plus délicat à mettre en place.

Une fois les premiers composants CMS souder, je pourrai commencer à placer les composants traversants en commençant par le support du MAX485et le GPIO 20 car ce sont les plus délicat à souder puis je pourrais terminer ma carte en soudant les connecteurs en extrémité comme l'alimentation JACK, le port USB etc.

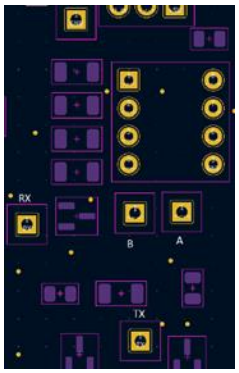
Une fois cela fait, et en vérifiant bien qu'il n'y a aucun composant manquant, je pourrais effectuer la première mise en service de la carte.

Procédure de première mise en service :

Pour la première mise en service, je devrai m'assurer grâce aux points de tests que les bonnes tensions arrivent aux bons endroits comme :

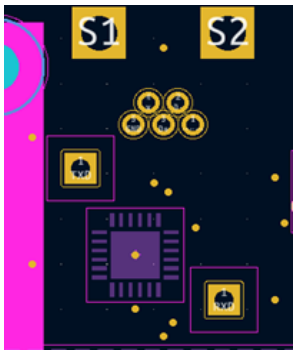
L'alimentation 12V, l'alimentation en 5V ou 3.3V du MAX485 et la tension des levels shifter. Il est aussi impératif que je vérifie qu'il n'y ait pas d'échauffement anormal qui pourrait être synonyme de courts-circuits.

Si toutes les alimentations sont correctes je pourrais alors commencer mes essais en testant la liaison I2C avec un i2cdetect pour vérifier que le module GNSS communique correctement,



Par la suite je pourrais commencer à tester les liaisons UART et Modbus en répétant la même procédure que j'ai pu faire précédemment avec le break out MAX485 TTL to RS485.

Je pourrais aussi tester que le module CP2104 convertie bien les signaux USB en UART grâce aux points de tests à proximité :



Si les modules sont bien alimentés et fonctionnent lors des essais basiques, alors nous pourrons commencer à faire des programmes plus complexes pour vraiment répondre aux attentes de ROPE & WEB.

Partie individuelle
Doyer Paul

Pour ma part je m'occupe de la réception GNSS ce qui consiste à récupérer la localisation de l'highliner et de la transmettre au module XBEE donc j'ai fait :

- La sélection de module GNSS le plus adapté.
- La conception du schéma fonctionnel qui me permettra ceci.
- Le routage de la carte qui intègre une partie alimentation, une partie pour le module XBEE et une partie avec le module GNSS.

Planning donné avant le projet :

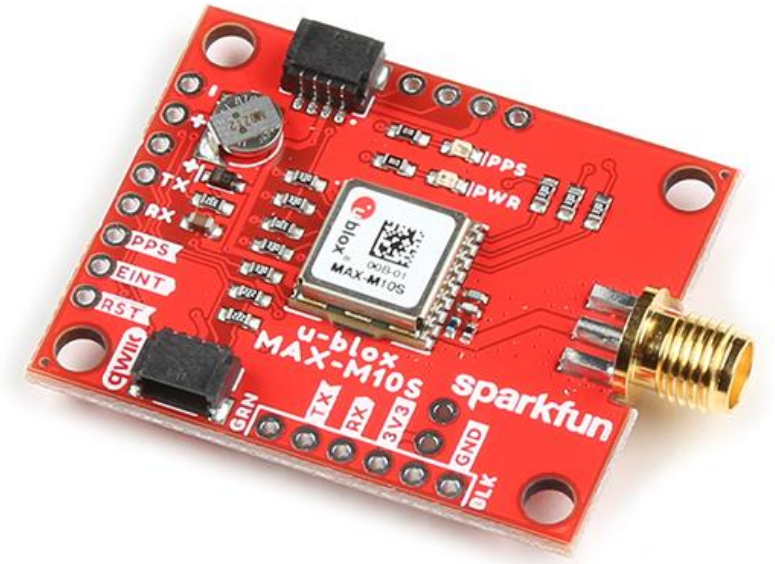
Projet	Mar 03/01/23	Ven 16/06/23		
Specifications générales	Mar 03/01/23	Mar 03/01/23		
Essais et validation des structures (prototypage rapide)	Mar 03/01/23	Ven 10/02/23	11	
Prototypage rapide et routage	Lun 06/03/23	Mer 12/04/23	8	
Fabrication et essais	Mar 02/05/23	Ven 09/06/23	9	
Fichier de fabrication PCB finalisés	Mer 12/04/23	Mer 12/04/23		
Rédaction du dossier	Lun 02/01/23	Jeu 25/05/23		

Voici le planning réel :

Projet	257	Mar 03/01/23	Ven 16/06/23	
Se documenter sur ce qu'est un module GNSS	1 h	Mar 03/01/23	Mar 03/01/23	
Prendre connaissance de la documentation des modules proposés et les mettre en oeuvre.	62 h	Mar 03/01/23	Mer 08/02/23	11
Concevoir la partie alimentation de l'association	13 h	Lun 13/02/23	Lun 20/02/23	
Schéma structurel + liste des composants avec code commande.	43 h	Lun 20/02/23	Lun 20/03/23	
Router un circuit imprimé et fichier de fabrication PCB finalisés	54 h	Lun 20/03/23	Mer 19/04/23	

Je me renseigne sur les différents modules d'acquisition GNSS pour définir le plus adapté à notre projet

GNSS est synonyme de Global Navigation Satellite System, un terme générique qui décrit à la fois le GPS des États-Unis, les systèmes de positionnement mondiaux GLONASS en Russie et le système européen Galileo.
Le premier le Max-M10S de chez sparkfun.



Le deuxième le GNSS 7 click de chez mikroe

Pour finir le troisième le Grove Air530 de chez Seeed Studio.

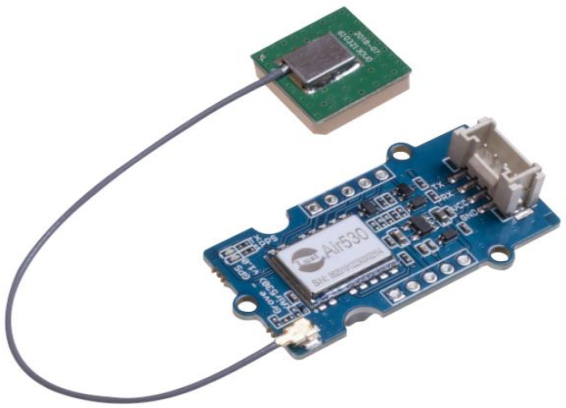


Tableau comparatif des trois modules gnss sur différentes caractéristique:

	Grove Air530	**MAX-M10S**	**Module GNSS 7**
Fabricant	Seeed Studio	Sparkfun	Mikroe
Référence	Grove Air530	MAX-M10S	Modules GNSS 7
Constellation	GPS,GALILEO, GLONASS,BEIDOU	GPS,GALILEO, GLONASS,BEIDOU	GPS,GLONASS
Prix	13 euros	40 euros	40 euros
Consommation	Ultra low : 31µA Low : 0.85µA	En fonctionnement : 25mA Veille : 46µA	50mA consommation unique car pas de mode veille
Tension d'alimentation	3.3/5V	max 3.6V typical : 3V min : 2.7V	max 3.6V typical : 3V min : 2.7V
Librairies arduino à installer pour les tests		SparkFun_u-blox_GNSS_v3 LIEN	SparkFun_u-blox_GNSS_Arduino_Library LIEN
Distributeur	Mouser Electronics	Sparkfun	Lextronic
Types de communication	UART	12C,UART	GPIO,I2C,SPI,UART,USB
Disponibilité	Disponible	Disponible	Disponible
Précision	10m	2-4m	2-4m

Après avoir analyser les caractéristiques des 3 capteurs je définis que parmi les trois proposé le MAX_M10S de chez sparkfun sera le plus adapté à notre projet

Qwiic and I²C (a.k.a. DDC)

There are two PTHs labeled SDA and SCL which indicates the I²C data lines. Similarly, you can just use the Qwiic connector to provide power and connect to the I²C pins. The Qwiic ecosystem is made for fast prototyping by removing the need for soldering. All you need to do is plug a Qwiic cable into the Qwiic connector and voila!

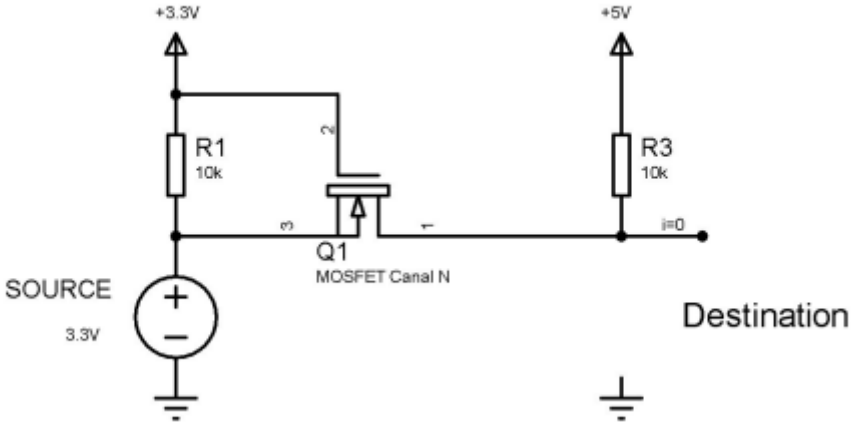
Top View | *Bottom View*

Voici le code proposé dans les exemples de la librairie que le revendeur nous demande de télécharger.

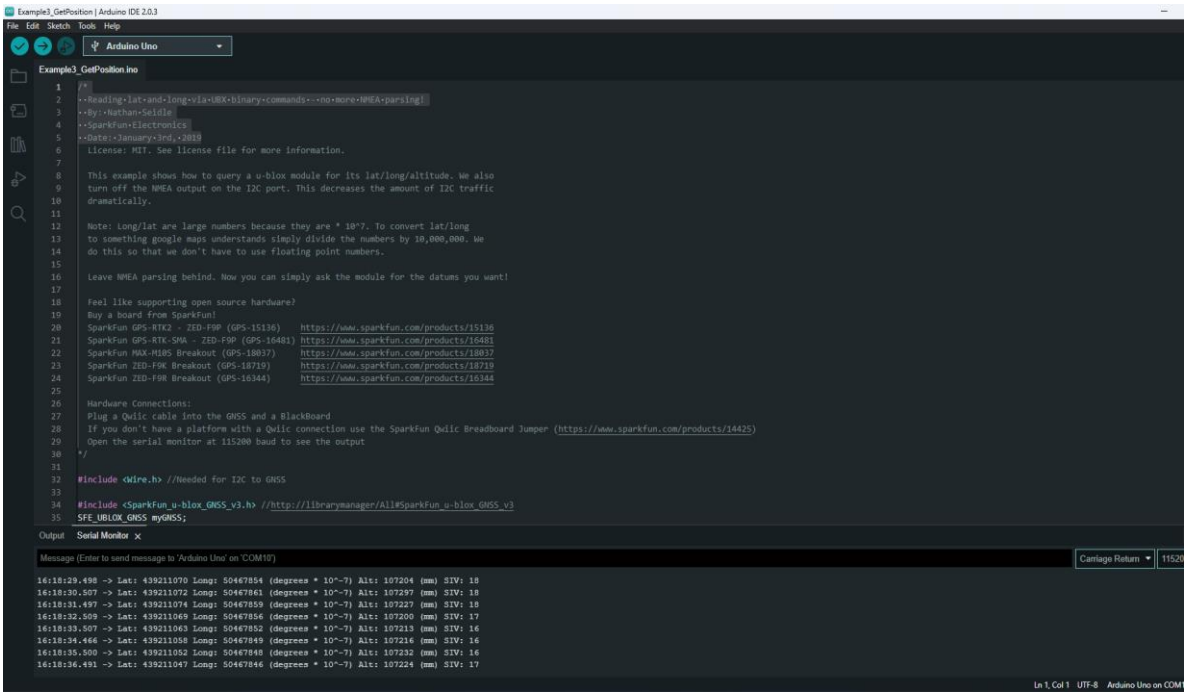
```
#include <Wire.h> //Needed for I2C to GNSS

#include <SparkFun_u-blox_GNSS_Arduino_Library.h> //http://librarymanager/All#SparkFun_u-blox_GNSS
SFE_UBLOX_GNSS myGNSS;

long lastTime = 0; //Simple local timer. Limits amount if I2C traffic to u-blox module.

void setup()
{
  Serial.begin(115200);
  while (!Serial); //Wait for user to open terminal
  Serial.println("SparkFun u-blox Example");

  Wire.begin();

  //myGNSS.enableDebugging(); // Uncomment this line to enable helpful debug messages on Serial

  if (myGNSS.begin() == false) //Connect to the u-blox module using Wire port
  {
    Serial.println(F("u-blox GNSS not detected at default I2C address. Please check wiring. Freezing."));
    while (1);
  }

  myGNSS.setI2COutput(COM_TYPE_UBX); //Set the I2C port to output UBX only (turn off NMEA noise)
  myGNSS.saveConfigSelective(VAL_CFG_SUBSEC_IOPORT); //Save (only) the communications port settings to flash and BBR
```

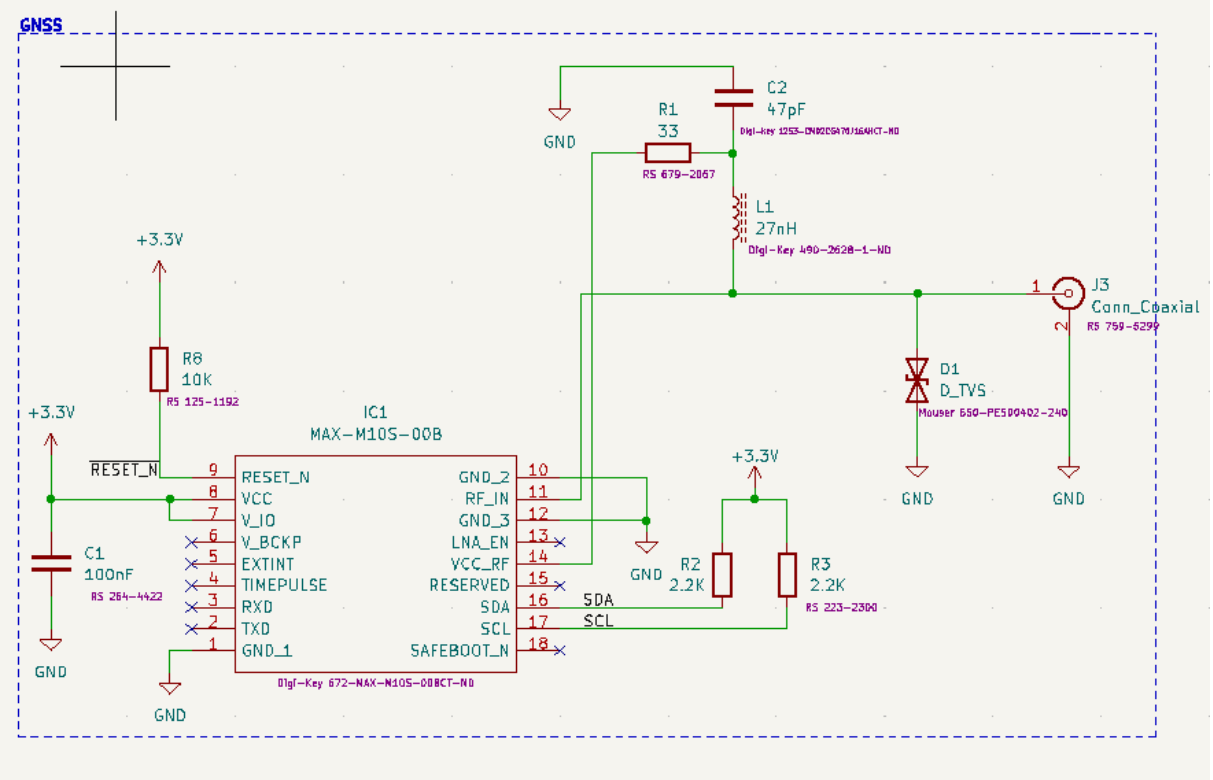
```
}

void loop()
{
  //Query module only every second. Doing it more often will just cause I2C traffic.
  //The module only responds when a new position is available
  if (millis() - lastTime > 1000)
  {
    lastTime = millis(); //Update the timer

    long latitude = myGNSS.getLatitude();
    Serial.print(F("Lat: "));
    Serial.print(latitude);

    long longitude = myGNSS.getLongitude();
    Serial.print(F(" Long: "));
    Serial.print(longitude);
    Serial.print(F(" (degrees * 10^-7)"));

    long altitude = myGNSS.getAltitude();
    Serial.print(F(" Alt: "));
    Serial.print(altitude);
    Serial.print(F(" (mm)"));

    byte SIV = myGNSS.getSIV();
    Serial.print(F(" SIV: "));
    Serial.print(SIV);

    Serial.println();
  }
}
```

Je dois faire un level shiffter car je sors du gnss avec du 3,3V mais la communication via la Raspberry pi se fait en 5V.

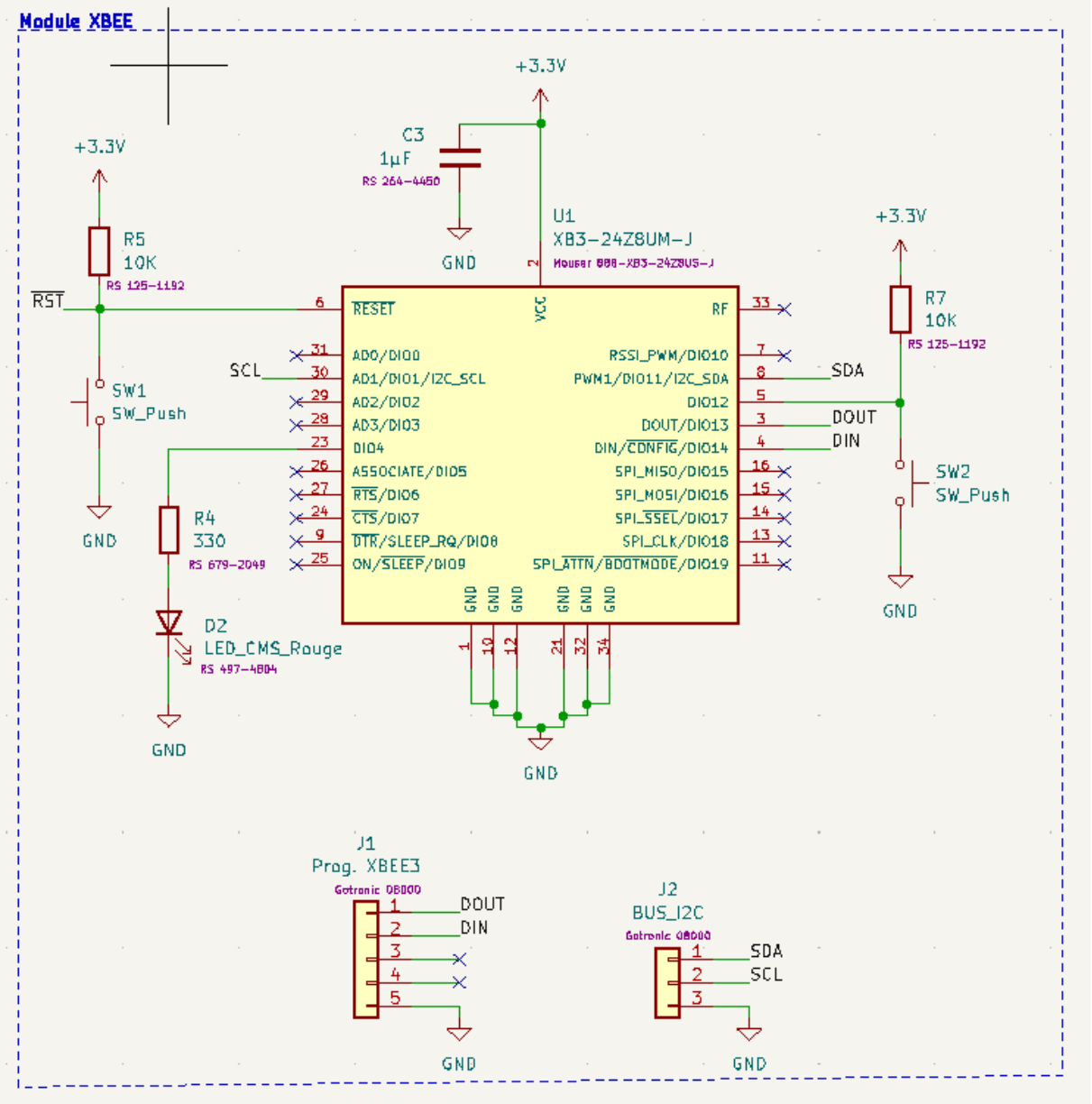


Et donc voilà le résultat avec les MAX-M10S et le montage avec level shiffter j'ai mes coordonnée qui concorde avec ma position lors du test effectué sur la capture d'écran :

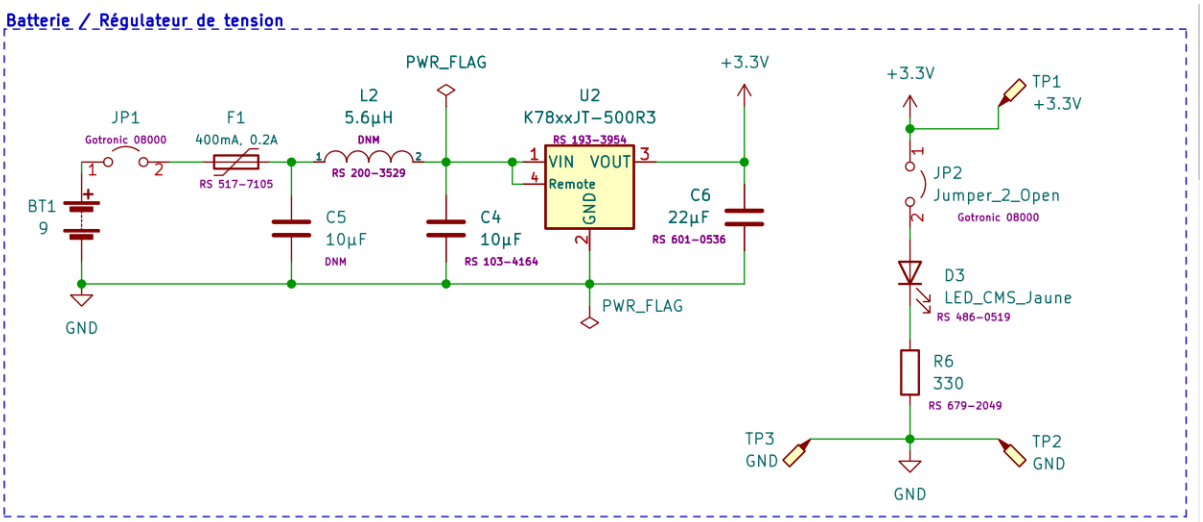




Ci-dessus se trouve le module GNSS qui récupère (les 3 informations voulu longitude latitude et hauteur) via l'antenne.



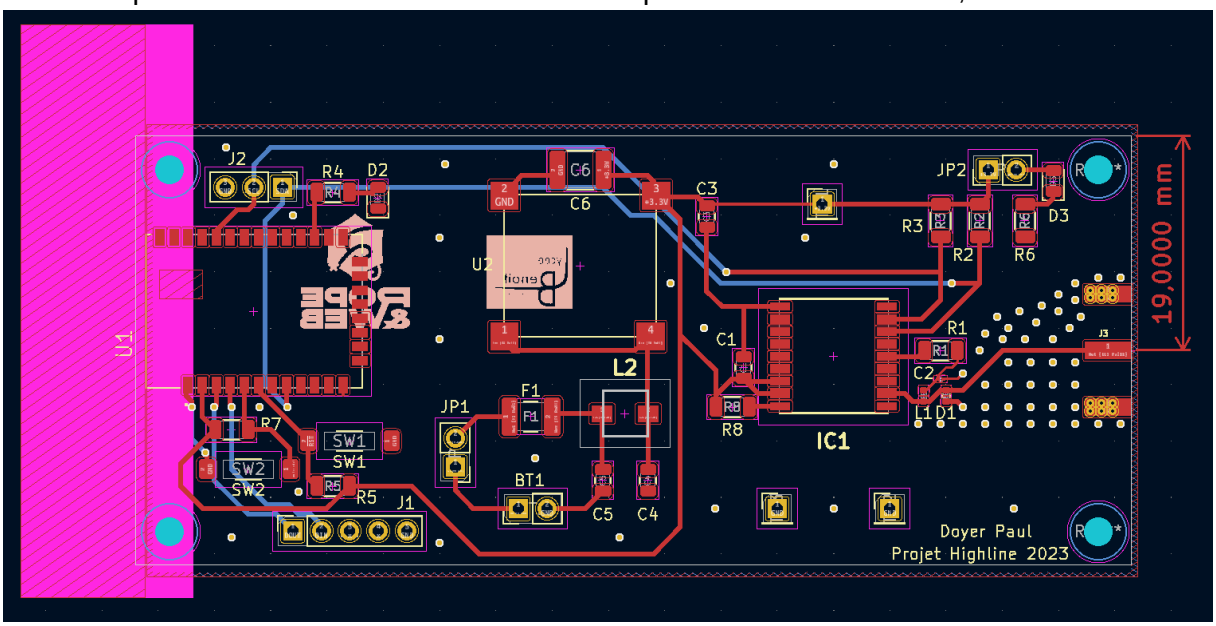
Le module xbee disposée sur la carte pour relayer les informations récupérées et les transmettre vers les autres cartes xbee dispose sur la slackline.



Voici la partie alimentation de la carte, une batterie 9V et un régulateur de tension pour passer de 9V à 3,3V car le module GNSS comme le le module XBEE ont besoin d'être alimenter en 3,3V.

Il y a aussi disposé sur la droite de la capture d'écran la partie point de test ou on pourra se mettre pour tester la carte.

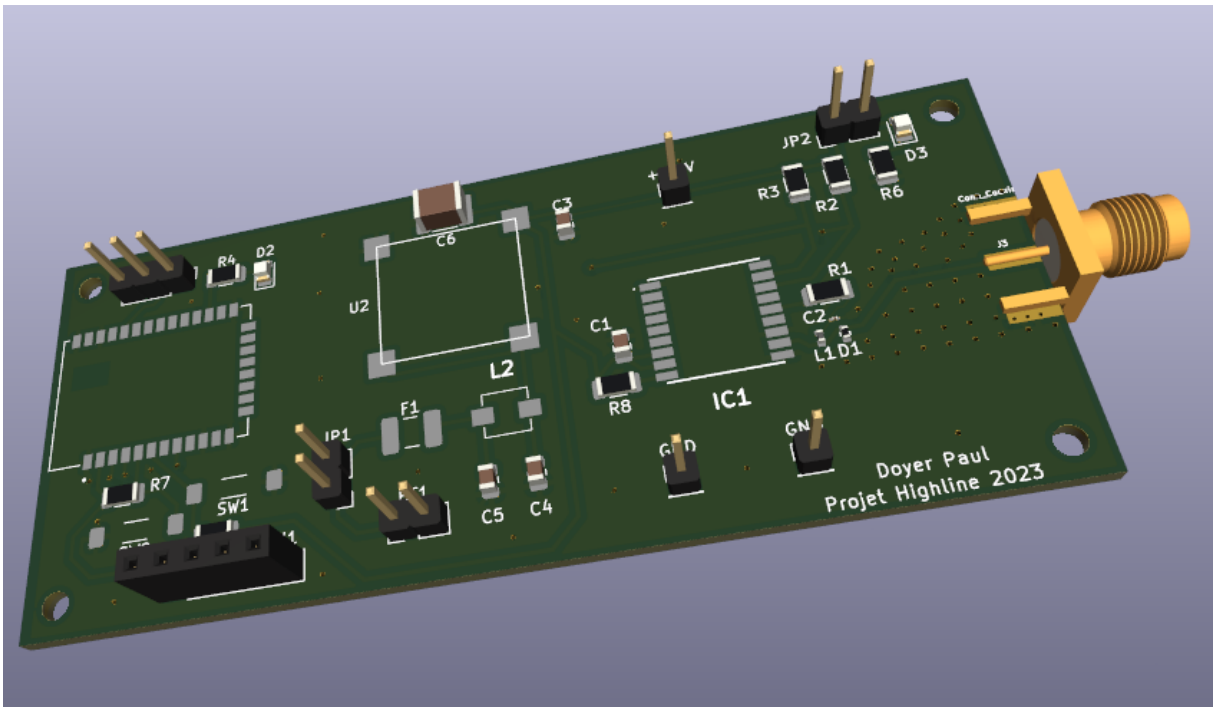
Le level shifter n'est plus car maintenant je ne passe plus par une Arduino uno mais je communique directement avec le module XBBE qui fonctionne aussi en 3,3V



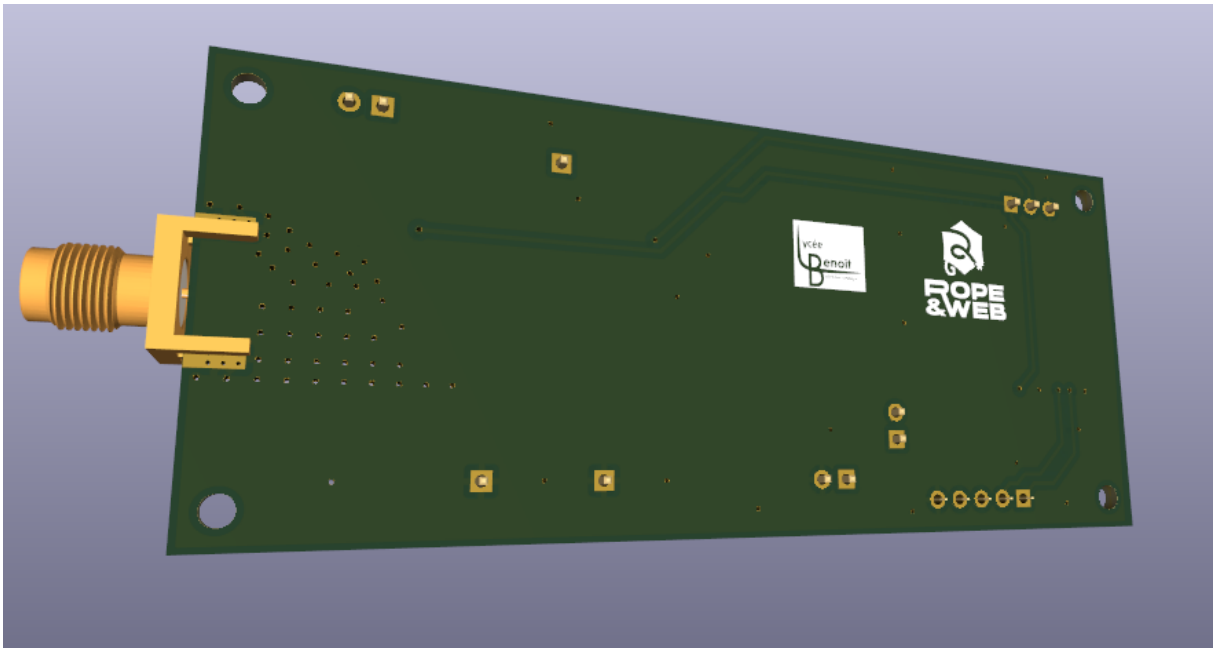
Voici le module une fois routé, mais ce routage c'est avant tout un échange avec Mr. Hortolland pour à force du temps optimiser la carte et l'adapter à nos besoins.

Enfin voici le rendu 3D :

Verso :



Recto :



Listes des composants présents sur la carte avec leurs codes commande pour pouvoir faire un devis :

	Valeurs :	Codes Commandes :
BT1	9V	
C1	100nF	RS 264-4422
C2	47pF	Digi-Key 1253-CM02CG470J16AHCT-ND
C3	1µF	RS 264-4450
C4, C5	10µF	RS 103-4164
C6	22µF	RS 601-0536
D1	D_TVS	Mouser 650-PESD0402-240
D2	LED_CMS_Rouge	RS 497-4804
D3	LED_CMS_Jaune	RS 486-0519
F1	400mA, 0.2A	RS 517-7105
IC1	MAX-M10S-00B	Digi-Key 672-MAX-M10S-00BCT-ND
J1	Prog. XBEE3	Gotronic 08000
J2	BUS_I2C	Gotronic 08000
J3	Conn_Coaxial	RS 759-5299
JP1, JP2	Jumper_2_Open	Gotronic 08000
L1	27nH	Digi-Key 490-2628-1-ND
L2	5.6µH	RS 200-3529
R1	33	RS 679-2067
R2, R3	2.2K	RS 223-2300
R4, R6	330	RS 679-2049
> R5, R7, R8	10K	RS 125-1192
> SW1, SW2	SW_Push	RS 102-361
TP1	+3.3V	Gotronic 08000
> TP2, TP3	GND	Gotronic 08000
U1	XB3-24Z8UM-J	Mouser 888-XB3-24Z8US-J
U2	K78xxJT-500R3	RS 193-3954

Partie individuelle
Soualmia Jessim

Présentation de ma partie :

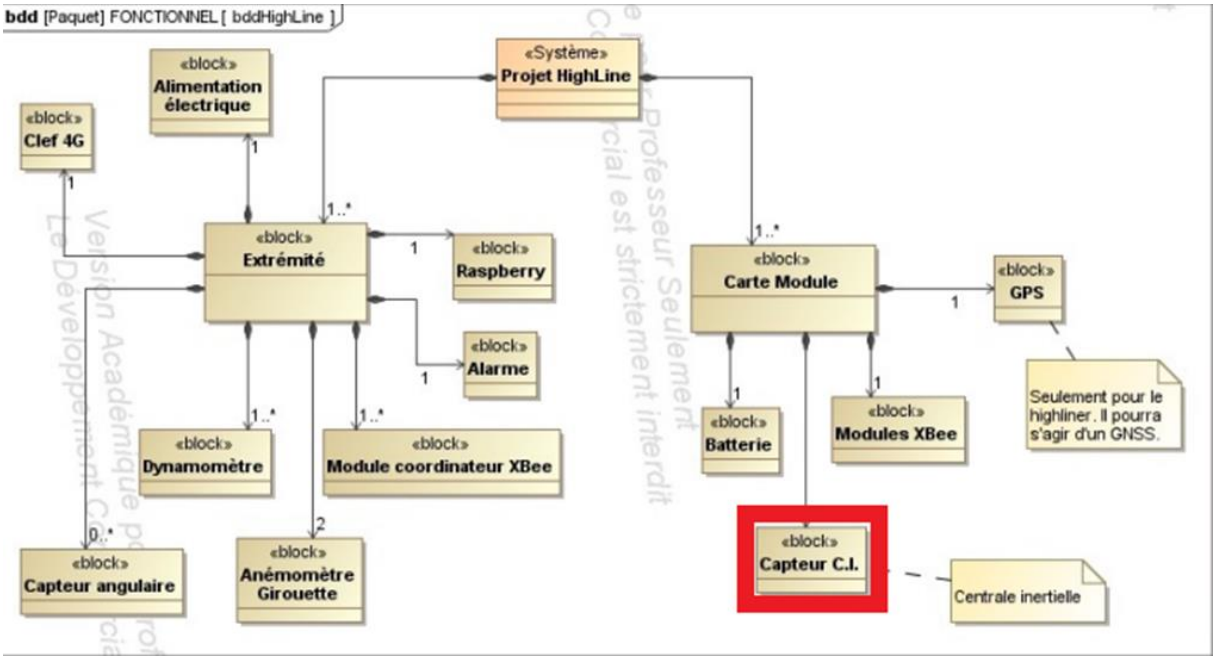
Ma partie concernant le projet consiste à programmer un central inertiel à partir d'un BNO055 pour assurer le suivi de la highline permettant d'afficher les relevés suivants :

-La vitesse Angulaire

-L'accélération

Pour finir, je mettrai en œuvre un programme qui sera associé à une liaison XBEE et serait reporté sur une carte d'une largeur inférieur à celle de la highline.

Diagramme de blocs :



Avant de commencer à travailler le projet j'ai d'abord analysé tous les documents qui m'ont été fournis par l'enseignant puis j'ai effectué un diagramme de gantt afin de terminer toutes les étapes qui j'ai effectué durant le projet.

Voici le diagramme de gant :

			Projet	257 h?	Mar 03/01/23	Ven 16/06/23	257 h?		
11			Specifications générales	12 h	Mar 03/01/23	Mer 04/01/23	12 h		
12			Documentation sur la central inertielle+ video tuto	4 h	Lun 09/01/23	Mar 10/01/23	4 h	11	
13			étude des composant+installation des logiciels	2 h	Mar 10/01/23	Mar 10/01/23	2 h	12	
14			réalisation du schéma fritzing	3 h	Mar 10/01/23	Mer 11/01/23	3 h	13	
15			instalation des librairie arduino+ cabalge du breakout	3 h	Lun 16/01/23	Lun 16/01/23	3 h	14	
16			réalisation du programme + video tuto	2 h	Mar 17/01/23	Mar 17/01/23	2 h	15	
17			test capteur avec programme	4 h	Mar 17/01/23	Mer 18/01/23	4 h	16	
18			cablage de la carte avec un analyseur logique + documentation	3 h	Mer 18/01/23	Lun 23/01/23	3 h	17	
19			modifiaction du programme arduino+ analyse de tram+ installation kicad (symbole, empreintes etc..)	14 h	Mar 24/01/23	Mer 01/02/23	14 h	18	
20			étude de la tram	5 h	Mar 07/02/23	Mer 08/02/23	5 h	19	
21			réalisation schéma kicad	6 h	Mer 08/03/23	Lun 13/03/23	6 h	20	
22			modification du schéma kicad	7 h	Mar 14/03/23	Mer 15/03/23	7 h	21	
23			affectuation d'empreintes + liste des comosant+ routage	30 h	Lun 27/03/23	Mer 12/04/23	30 h	22	
24			Fabrication et essais	60 h	Lun 01/05/23	Mer 07/06/23	60 h	23	

Liste de composants :

Pour cela je serais fourni du matériel et des documents suivants :

Centrale Inertielle BNO055 :

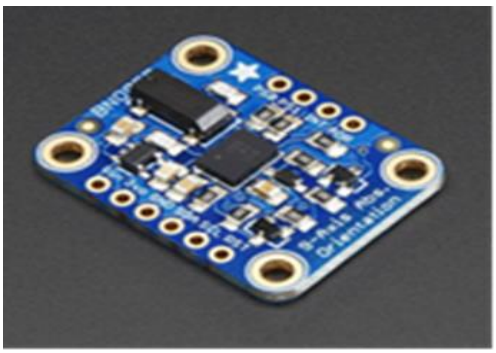
Plage de mesure :

Acceleration ranges ±2g/±4g/±8g/±16g

Ranges switchable from ±125°/s to ±2000°/s

Magnetic field range typical ±1300µT (x-, y-axis);
±2500µT (z-axis)
Magnetic field resolution of ~0.3µT

-Breakoutn BNO055



Power Pins

- VIN: 3.3-5.0V power supply input
- 3VO: 3.3V output from the on-board linear voltage regulator, you can grab up to about 50mA as necessary
- GND: The common/GND pin for power and logic

I2C Pins

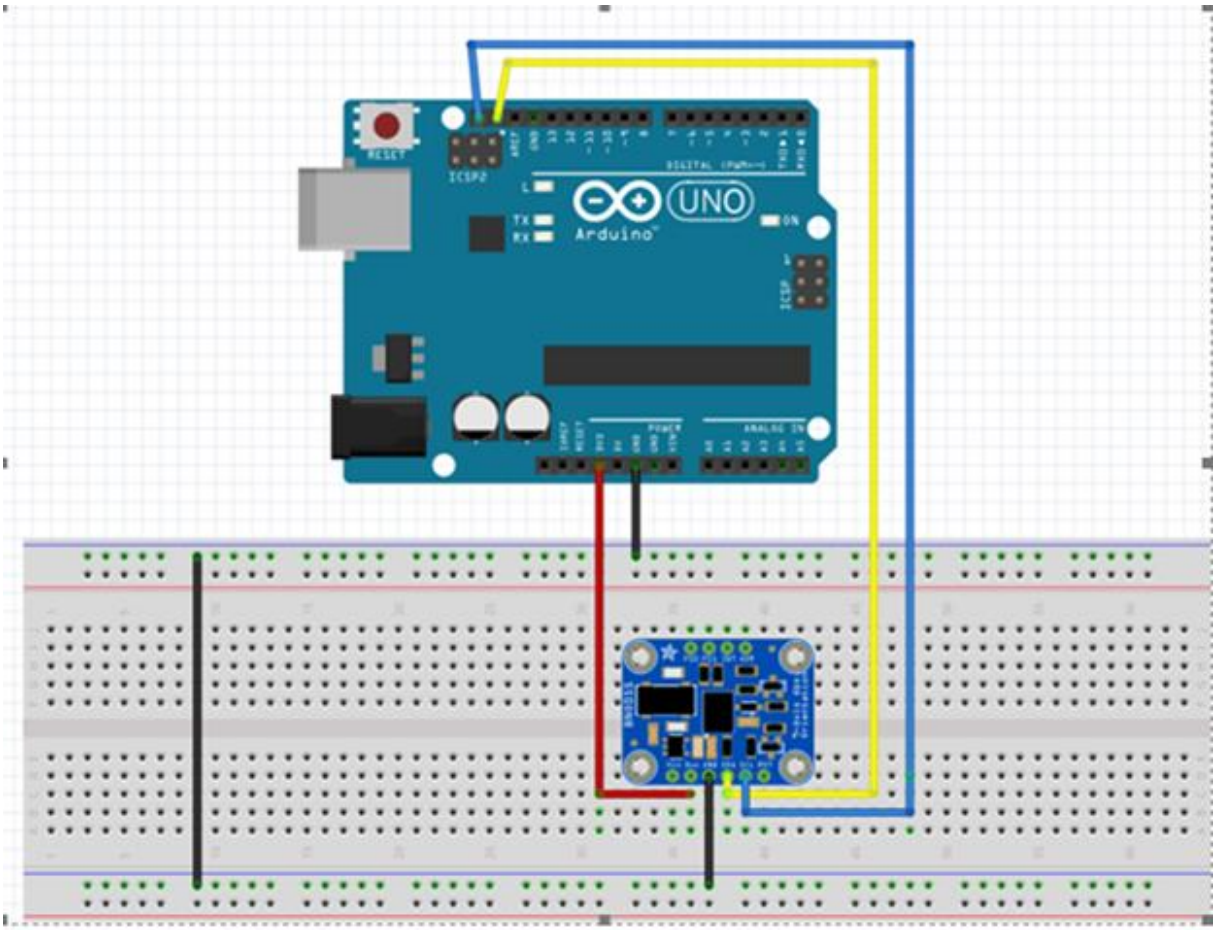
- SCL - I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.
- SDA - I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.

Arduino uno :

Microcontrôleur	ATmega328P
Tension de fonctionnement	5V
Tension d'entrée (recommandée)	7-12V
Tension d'entrée (limite)	6-20V
Broches d'E/S numériques	14 (dont 6 fournissent une sortie PWM)
Broches d'E/S numériques PWM	6
Broches d'entrée analogiques	6
Courant CC par broche d'E/S	20mA
Courant continu pour broche 3.3V	50 mA

Mise en œuvre avec Fritzing :

Avant de pouvoir faire le programme j'ai effectué le câblage suivant sur Fritzing :



Voici les informations qui m'ont permis de faire le câblage :

-La breakout BNO055 est alimentée sous 3.3V

-Relier au SDA et SCL de la carte Arduino que nous utilisons en I2c

C'est sur le logiciel Arduino Ide 2.0 IDE que nous allons travailler en installant d'abord les librairies qui sont :

- « Arduino AVR Boards By Arduino »

- « Adafruit_BNO055 »

De base je devais utiliser une carte Arduino nano pour le programme mais j'ai rencontré des difficultés pour la mettre en marche puis j'en ai discuté avec mon professeur d'électronique puis il m'a donné une carte Arduino uno.

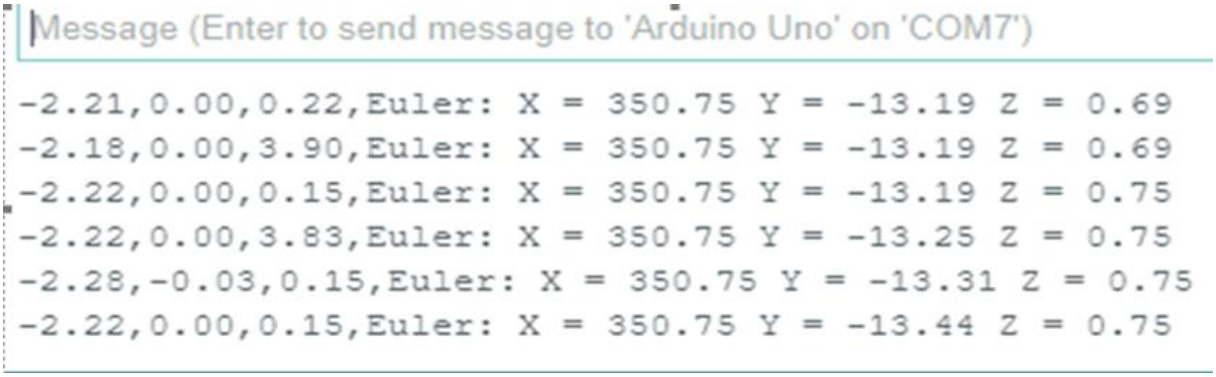
Pour donner suite à la réalisation du schéma fritzing j'ai ensuite effectué un programme pour la breakout BNO055 sur Arduino en suivant le tutoriel de Paul McWhorter. Ce programme va nous permettre de relever l'accélération et les angles X, Y, Z.

Voici le résultat et l'essai du Programme :

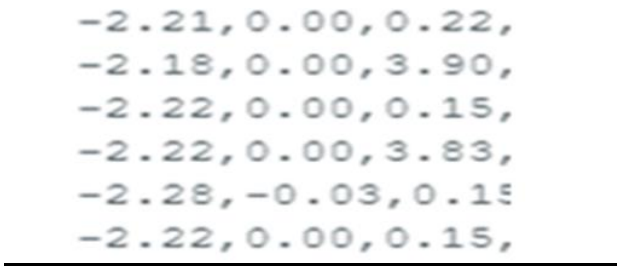
```
#include <Wire.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

Adafruit_BNO055 bno = Adafruit_BNO055(); // Crée une instance de la classe Adafruit_BNO055
void setup() {
  Serial.begin(9600); // Initialise la communication série à une vitesse de 9600 bauds

  if (!bno.begin()) { // Vérifie si l'initialisation du BNO055 a échoué
    Serial.println("Erreur lors de l'initialisation du BNO055");
    while (1); // Boucle infinie en cas d'erreur
  }
  Serial.println("BNO055 initialisé"); // Affiche un message indiquant que le BNO055 est initialisé
}
void loop() {
  imu::Vector<3> acc= bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER); // Lit les valeurs de l'accéléromètre et les stocke dans le vecteur 'acc'

  Serial.print(acc.x()); // Affiche la valeur de l'accélération sur l'axe X
  Serial.print(",");
  Serial.print(acc.y()); // Affiche la valeur de l'accélération sur l'axe Y
  Serial.print(",");
  Serial.print(acc.z()); // Affiche la valeur de l'accélération sur l'axe Z
  Serial.print(",");
  delay(100); // Attend pendant 100 millisecondes
```

```
  imu::Vector<3> euler = bno.getVector(Adafruit_BNO055::VECTOR_EULER); // Lit les angles d'Euler et les stocke dans le vecteur 'euler'

  Serial.print(" X = ");
  Serial.print(euler.x()); // Affiche l'angle d'Euler sur l'axe X
  Serial.print(" Y = ");
  Serial.print(euler.y()); // Affiche l'angle d'Euler sur l'axe Y
  Serial.print(" Z = ");
  Serial.println(euler.z()); // Affiche l'angle d'Euler sur l'axe Z

  delay(100); // Attend pendant 100 millisecondes
```

Voici Résultats reçu :

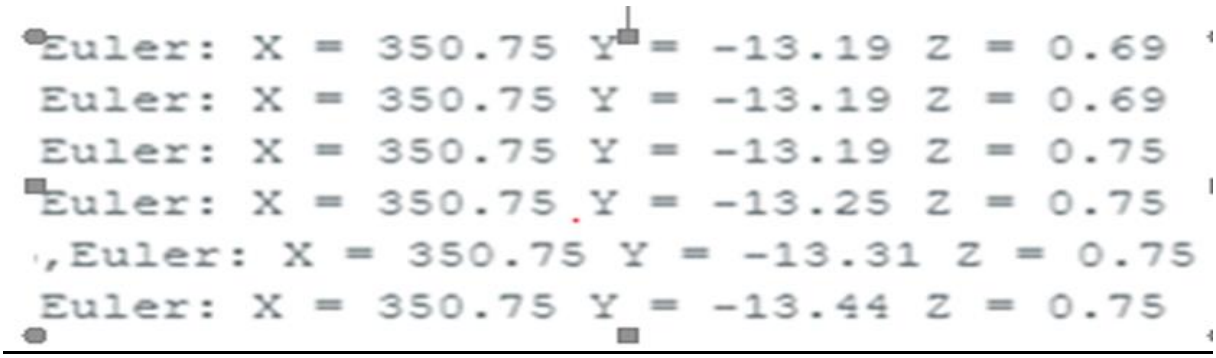


Résultat obtenue de l'accélération :



Les données d'accélération sont lues à partir du capteur en utilisant la méthode getVector()avec le paramètre Adafruit_BNO055::VECTOR_ACCELEROMETER. Les valeurs d'accélération dans les trois axes (x, y, z) sont ensuite affichées sur le port série.

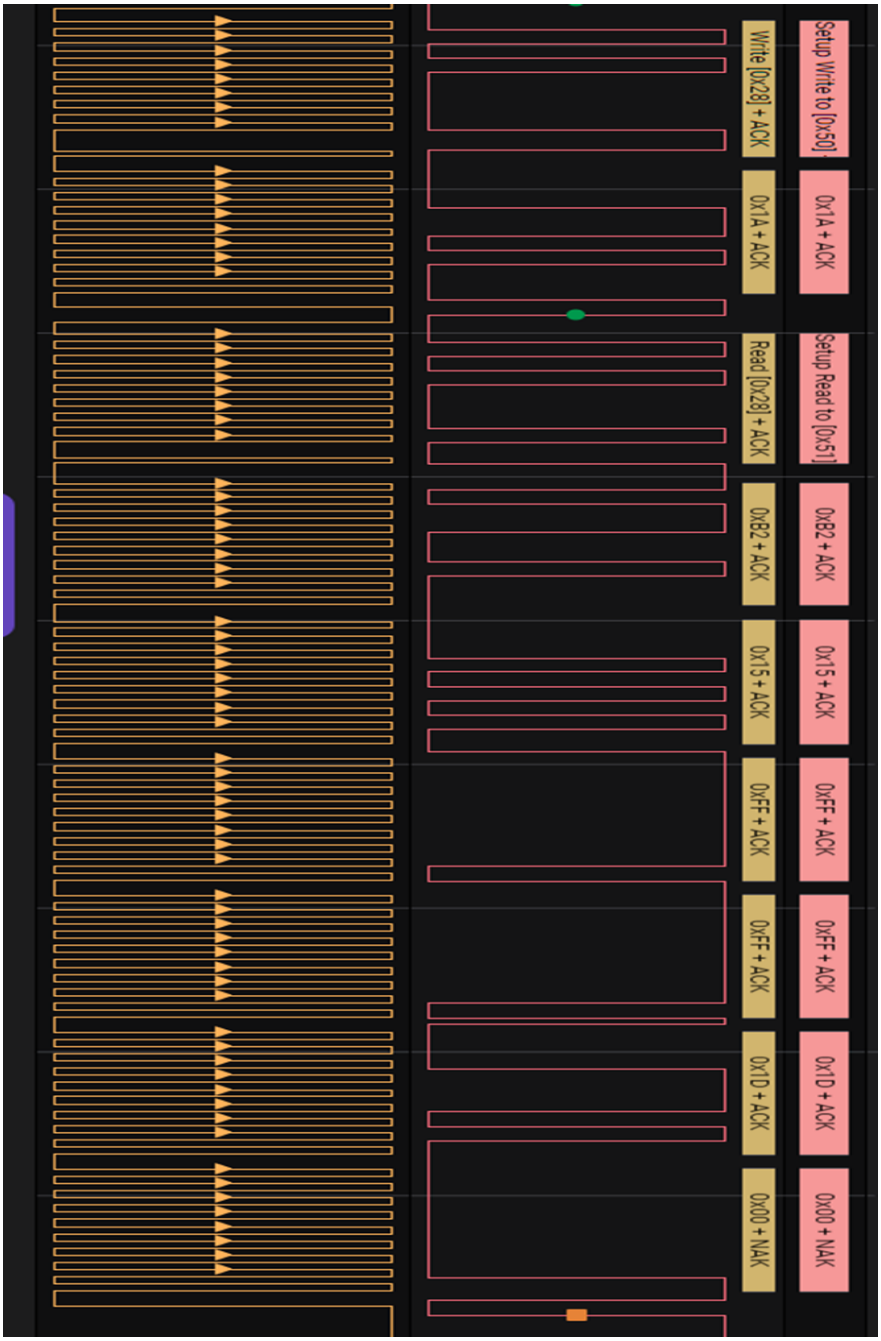
Résultat obtenue Pour les angles :



Les angles d'Euler sont lus à partir du capteur en utilisant la méthode getVector()avec le paramètre Adafruit_BNO055::VECTOR_EULER. Les valeurs des angles d'Euler sont ensuite affichées sur le port série.

Pour donner suite au résultat obtenu du programme j'ai utilisé un analyseur logique pour relever un tram avec des valeurs en hexadécimal.

Voici la trame suivante :

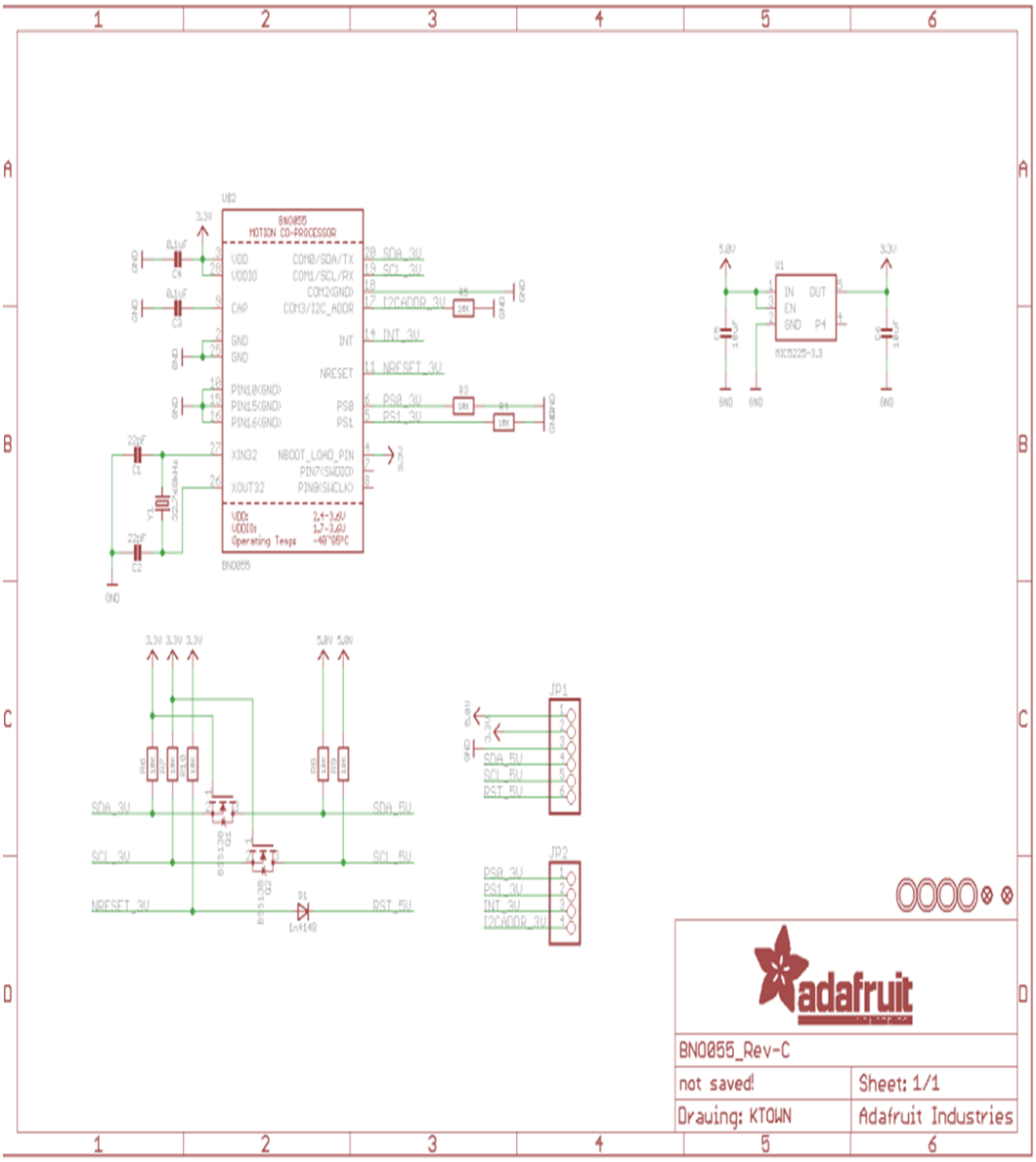


I2C configuration	COM3_state	I2C address
Slave	HIGH	0x29
Slave	LOW	0x28
HID-I2C	X	0x40

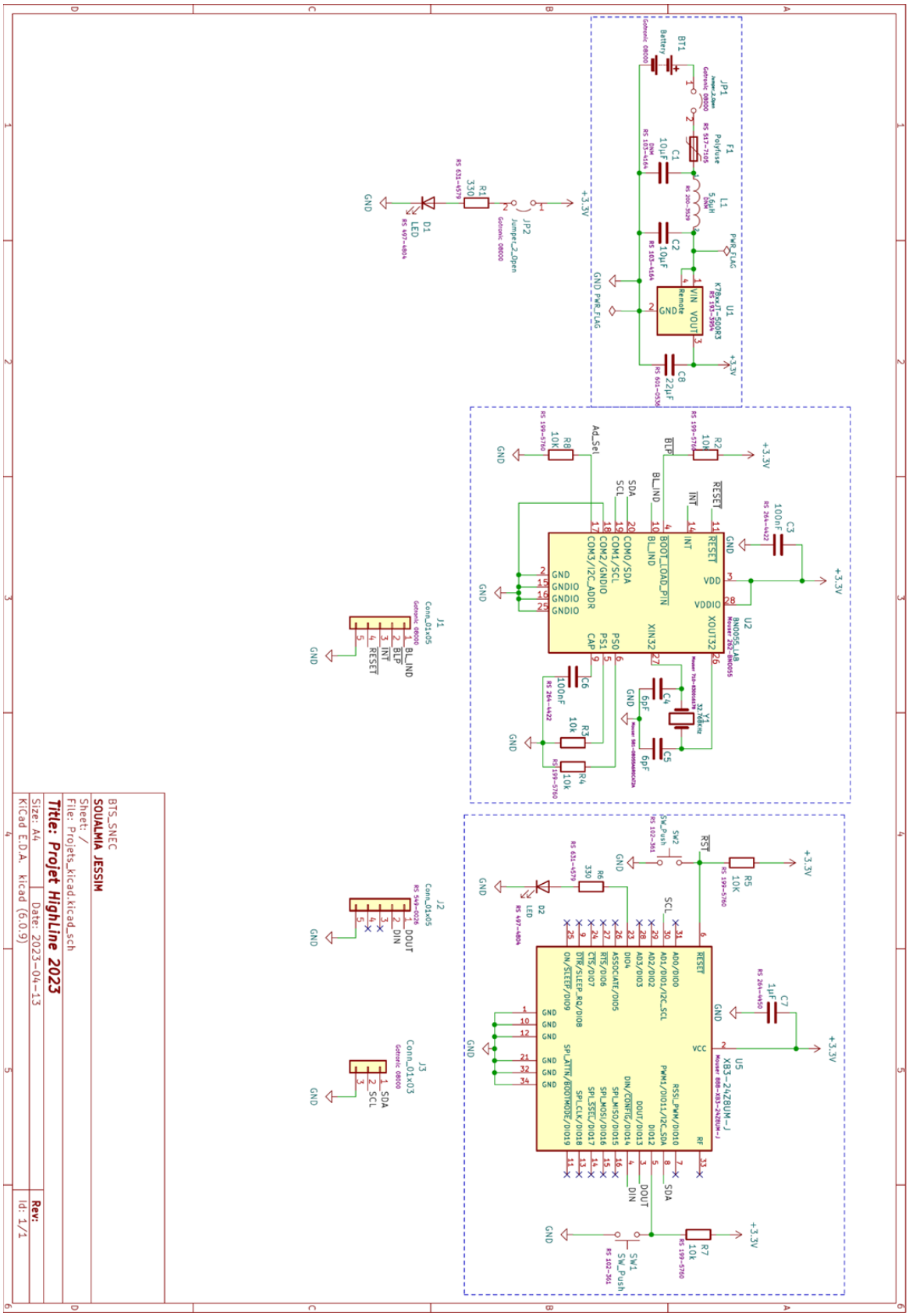
Le « write [Ox28] » représente l'adresse I2C du capteur BNO055 en écriture et le « Read [0x28] » en lecture

Avant de faire schéma final de la carte j'ai examiné le schéma de la breakout BNO055

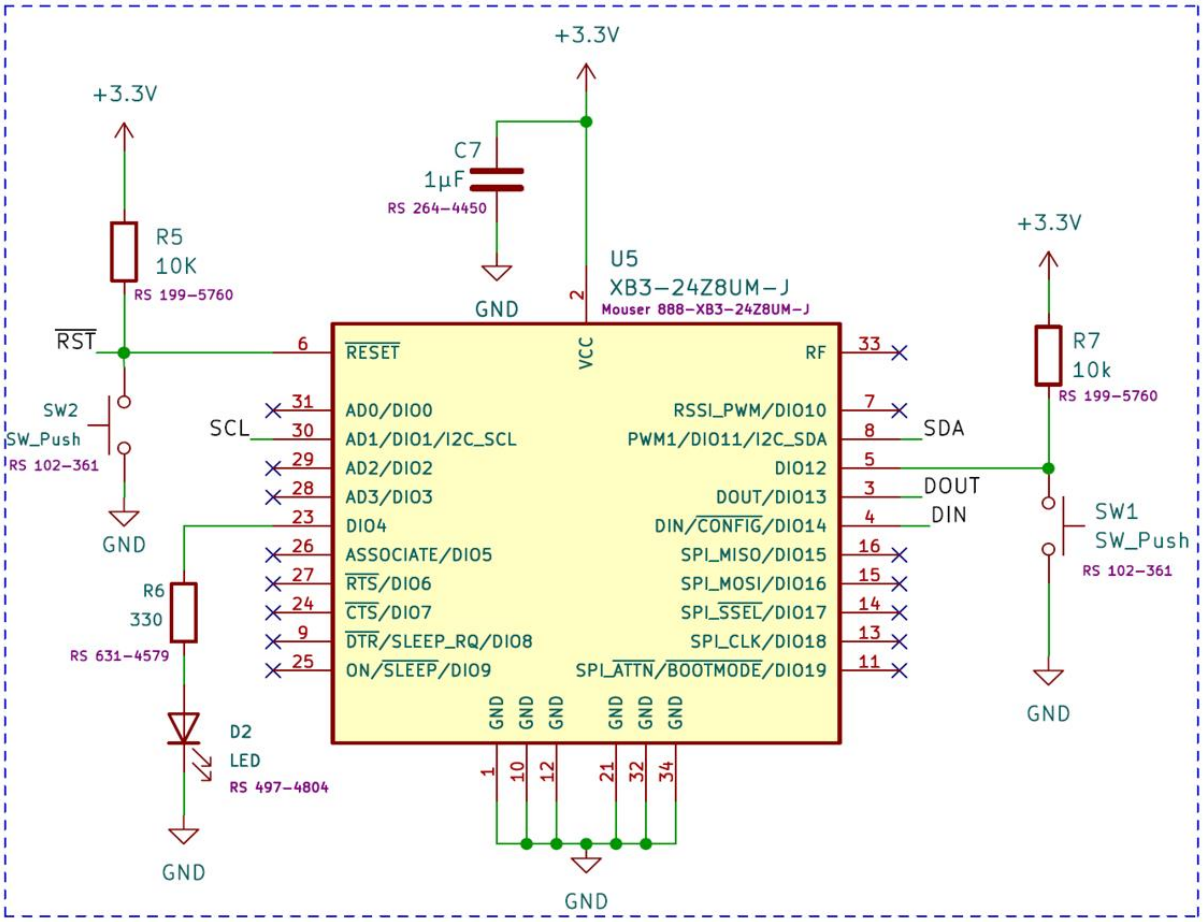
Voici le schéma du break out BNO055 de chez Adafruit :



Voici le schéma de la carte pour la centrale inertielle qui ira en dessous de la highline :



Module Xbee :

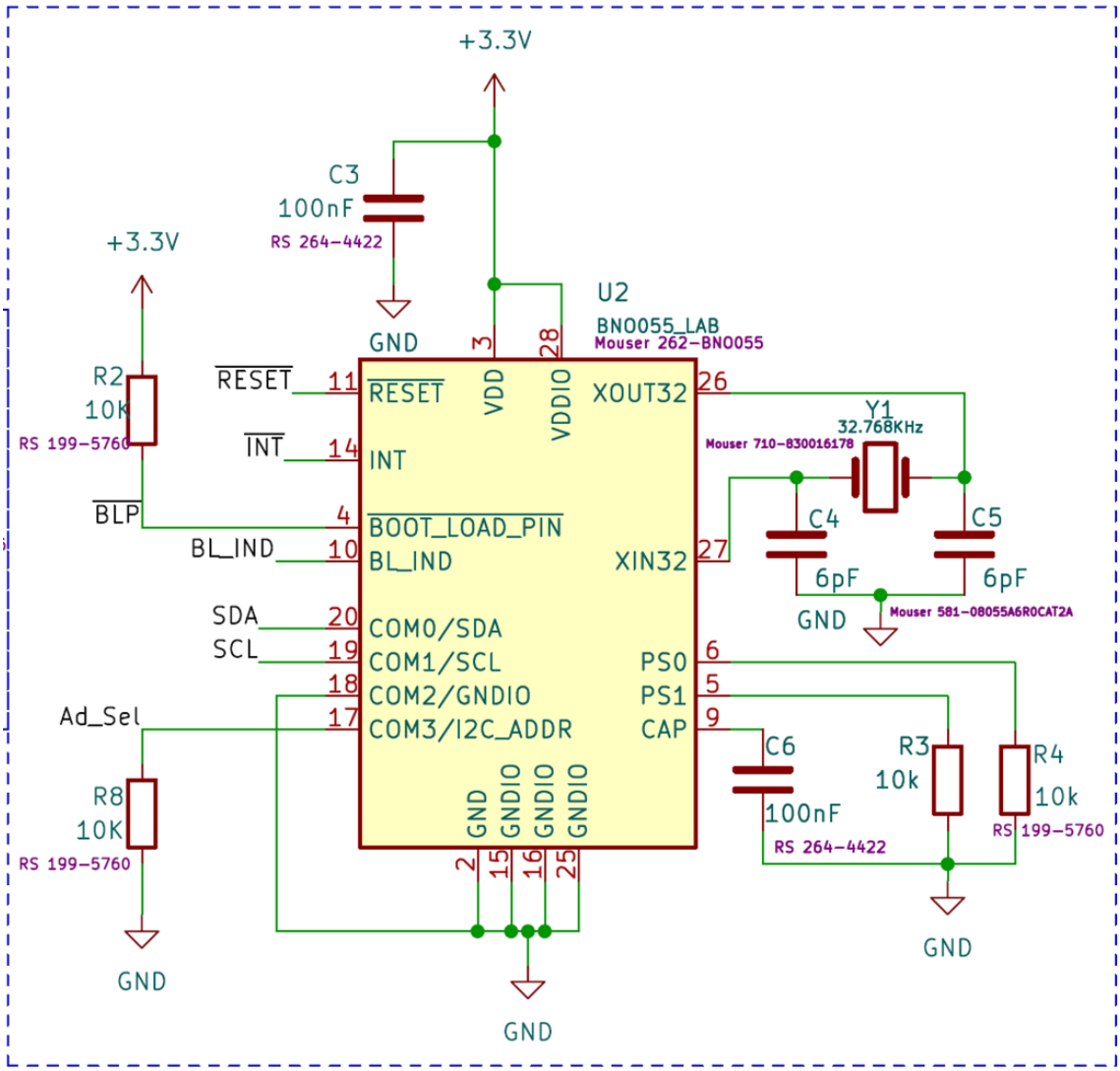


Voici le module XBEE qui servira à relayer les informations récupérées et les transmettre vers les autres cartes xbee disposées sur la slackline.

Un module Xbee 3 est un type de module de communication sans fil conçu pour faciliter la transmission de données sur de courtes distances. Il est principalement utilisé dans des applications d'Internet des objets, de domotique, de capteurs sans fil et de réseaux de capteurs.

Elle dispose de 2 résistances de Pull-up qui vont servir à fixer clairement un état électrique.

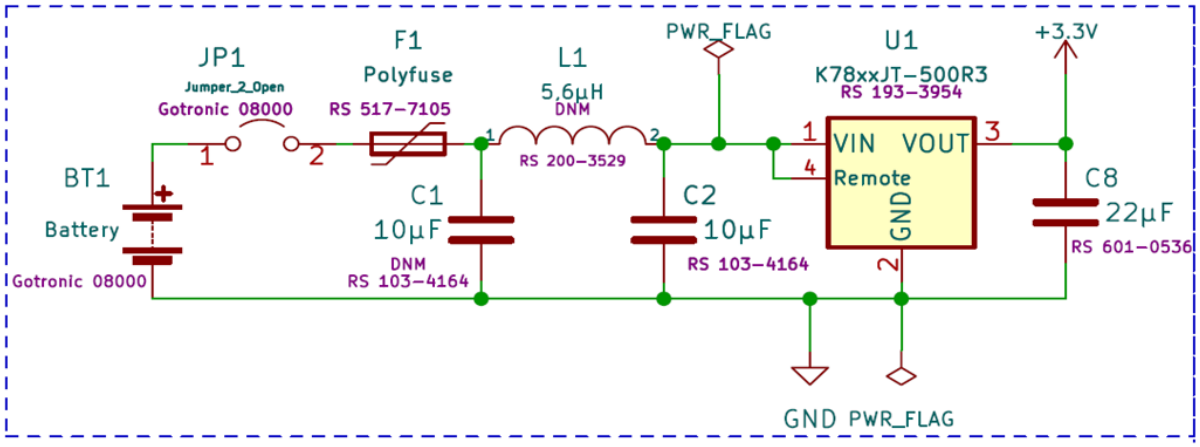
Capteur BNO055 :



Le BNO055 qui va servir à relever l'accélération et les angles X, Y, Z, et qui sera transmis vers le module XBEE.

Le capteur BNO055 est un capteur d'orientation à neuf axes qui combine plusieurs fonctionnalités pour fournir des informations précises sur l'orientation, l'accélération et le champ magnétique dans un seul composant.

Régulateur de découplage :



Un régulateur de découplage est un composant électronique utilisé dans les circuits électroniques pour maintenir une tension stable et constante, en particulier lorsqu'il y a des variations de la tension d'alimentation.

Le rôle principal d'un régulateur de découplage est de réduire les fluctuations indésirables de la tension d'alimentation qui peuvent se produire en raison de variations de charge, de bruit électrique ou d'autres perturbations.

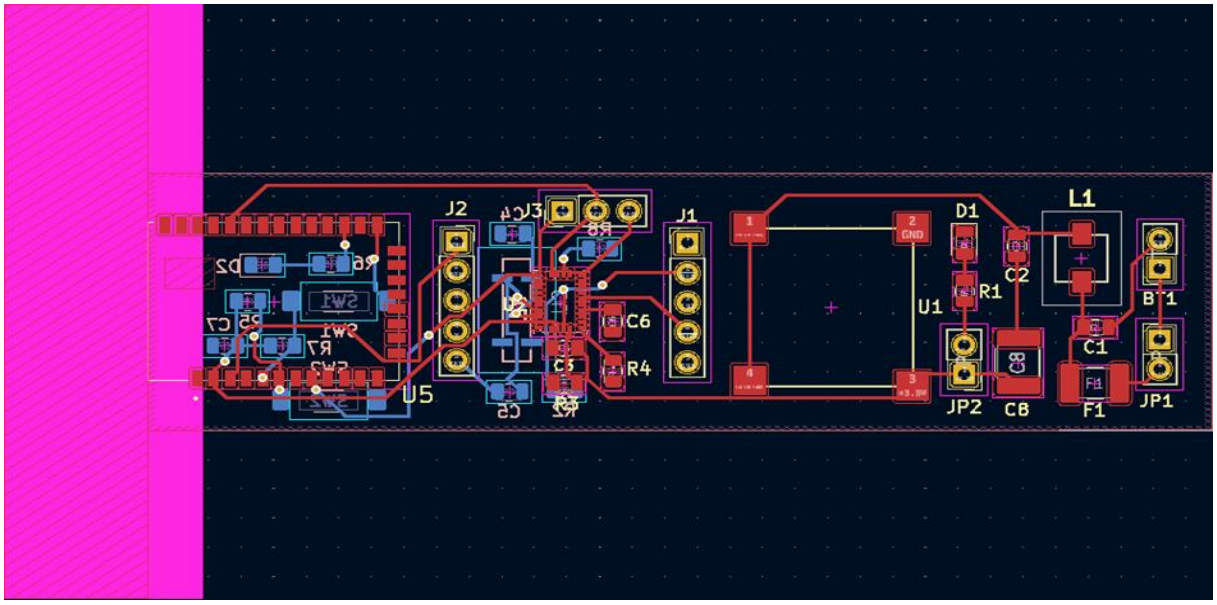
Suite à réalisation au schéma et à l'affectation des numéros de commande de chaque composant j'ai effectué un bon de commande :

Liste de matériel

SOUALMIA Jessim

Référence de stock	valeur	Code Commande	Prix unitaire	Quantité	Total:
BT1	Battery	Gotronic 08000		1	
> C1, C2	10µF	RS 103-4164		2	
> C4, C5	6pF	Mouser 581-08055A6R0CAT2A		2	
> C3, C6	100nF	RS 264-4422		2	
C7	1µF	RS 264-4450		1	
C8	22µF	RS 601-0536		1	
> D1, D2	LED	RS 497-4804		2	
F1	Polyfuse	RS 517-7105		1	
J1	Conn_01x05	Gotronic 08000		1	
J2	Conn_01x05	RS 549-0026		1	
J3	Conn_01x03	Gotronic 08000		1	
> JP1, JP2	Jumper_2_Open	Gotronic 08000		2	
L1	5,6µH	RS 200-3529		1	
> R1, R6	330	RS 631-4579		2	
> R3, R4, R7	10k	RS 199-5760		3	
> R2, R5, R8	10K	RS 199-5760		3	
> SW1, SW2	SW_Push	RS 102-361		2	
U1	K78xxJT-500R3	RS 193-3954		1	
U2	BNO055_LAB	Mouser 262-BNO055		1	
U5	XB3-24Z8UM-J	Mouser 888-XB3-24Z8UM-J		1	
Y1	32.768KHz	Mouser 710-830016178		1	

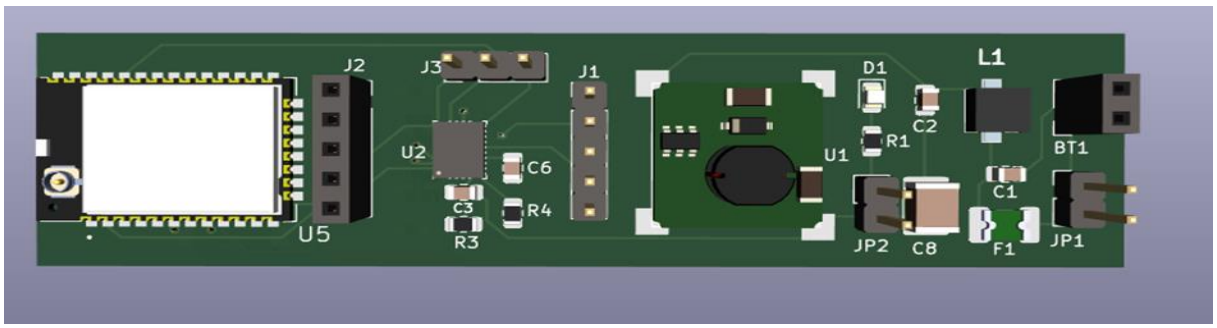
Voici le résultat après avoir fait le routage :



Pour le routage de la carte j'ai dû respecter certaines conditions :

- Comme la largeur de la carte pour qu'elle soit plus petite que la slackline
- Le plan de masse en top et bottom
- Mettre le module XBEE sur le bord de la carte pour respecter les zones sans cuivre en rose et afin d'éviter des perturbations

Voici maintenant sa visualisation en 3D:



Procédure de câblage :

Je vais commencer par souder les composants cms du côté BOTTOM car ils sont peu nombreux et donc plus simples pour les souder.
Suite au passage au four à refusion, je souderai le reste des composants du côté top comme le module XBEE, les résistances, condensateurs etc.…
Une fois les composants soudés je pourrais mettre en place les composants traversants et terminer ma carte en plaçant le BNO055.