

ECOLE DES MÉTIERS DE LAUSANNE
ECOLE SUPÉRIEURE

GÉNIE ÉLECTRIQUE

SYSTÈME D'ENREGISTREMENT DE TRAJECTOIRES DE VOL

Boîte noire miniaturisée

Auteur

Ali ZOUBIR

Superviseur

Juan José MORENO

Mandant

ASSOCIATION POUR LE MAINTIEN
DU PATRIMOINE AÉRONAUTIQUE

10 septembre 2023

Table des matières

Acronymes	5
Glossaire	5
1 Introduction	6
1.1 Aperçu	6
1.2 Tâches à réaliser	7
1.3 Schéma de principe	7
1.4 Planification	8
1.5 Livrable	8
2 Pré-étude	9
2.1 Fonctionnement du système	9
2.2 Schéma bloc	9
2.3 Description des blocs	9
2.4 Choix des composants et technologies	10
2.4.1 Microcontrôleur	10
2.4.2 Centrale inertielle	11
2.4.3 GPS / GNSS	12
2.4.4 Carte SD	13
2.4.4.1 Estimation de la capacité	13
2.5 Batterie, autonomie, charge et régulation	14
2.5.1 Technologies	14
2.5.2 Choix de la batterie	14
2.5.3 Régulateur de charge	15
2.6 Systèmes d'économie d'énergies	16
2.7 Diagramme des états du système	16
2.8 Estimation du coût du prototype	17
2.9 Conclusion et perspectives de la Pré-étude	17
3 Développement du schéma électronique	18
3.1 Blocs développés	18
3.2 Microcontrôleur	19
3.2.1 Connexion	19
3.2.2 Programmateur et reset	19
3.2.3 LED de vie	20
3.3 Périphériques	21
3.3.1 Carte SD	21
3.3.2 Centrale inertielle	21
3.3.3 GNSS	22
3.3.4 USB - FTI	22
3.4 Alimentations	23
3.4.1 Chargeur de batterie	23
3.4.2 Système d'enclenchement	24
3.5 Conclusion et perspectives de l'étude	24

4 Développement du circuit-imprimé	25
4.1 Mécanique du projet	26
4.1.1 Choix du boîtier	26
4.1.2 Placement des composants	26
4.1.3 Assemblage	27
4.2 Bill of materials	28
4.3 Routage	29
4.4 Conclusion et perspectives de la conception	30
5 Développement du firmware	31
5.1 Protocoles du GNSS	31
5.1.1 Messages NMEA	32
5.1.2 Interprétation des données NMEA	32
5.1.3 Code décodeur de trame NMEA	33
5.2 Configuration des périphériques	34
5.2.1 Timers	35
5.2.2 USARTs	37
5.2.3 Carte SD	37
5.3 Application principale	38
5.3.1 Commandes USB-UART	38
5.3.2 État de logging du système	40
5.4 Carte SD	41
5.4.1 Carte SD : initialisation et configuration	41
5.4.2 Carte SD : écriture des données	42
5.5 Centrale inertielle	43
5.5.1 Initialisation	43
5.5.2 Lecture des données	43
5.5.3 Système de détection de mouvements	44
5.5.3.1 Gestion de l'interruption	44
5.5.3.2 Mode low power	44
5.5.3.3 Configuration enclenchement automatique	45
5.6 Format des données	46
5.6.1 Données de l'IMU	46
5.6.2 Données du GNSS	47
5.7 Fonctions des fichiers	48
5.7.1 Centrale inertielle	48
5.7.2 Carte SD	49
5.7.3 Communication série avec FTDI	49
6 Développement application interface	50
6.1 Choix de l'environnement	50
6.2 Présentation de l'interface	50
7 Validation du design	52
7.1 Liste de matériel	52
7.2 Consommations	52
7.2.1 Méthode de mesure	52
7.2.2 Schéma de mesure	52
7.2.3 Mesures	52

7.2.3.1	Proposition de stratégies d'économie d'énergie	53
7.3	Bus de communications	53
7.3.1	Communication I2C	54
7.3.1.1	Méthode de mesure	54
7.3.1.2	Schéma de mesure	54
7.3.1.3	Mesures	54
7.3.2	Communication UART GNSS	56
7.3.2.1	Méthode de mesure	56
7.3.2.2	Schéma de mesure	56
7.3.2.3	Mesures	56
7.3.3	Communication UART2 FTDI	58
7.3.3.1	Méthode de mesure	58
7.3.3.2	Schéma de mesure	58
7.3.3.3	Mesures	58
7.3.4	Communication SPI, carte SD	59
7.3.4.1	Méthode de mesure	59
7.3.4.2	Schéma de mesure	59
7.3.4.3	Mesures	60
8	Caractéristiques du prototype final	61
8.1	Problème rencontré, modification et amélioration	61
9	Conclusion	62
10	Bibliographie	63
11	Annexes	64
11.1	Schéma	65
11.2	Dessin du boitier	69
11.3	Fichiers PCB	73
11.4	Code du firmware	75
11.5	Code python du script visualisation CSV	121
11.6	Code python de l'application BBX-Connect	124

Acronymes

MCU microcontrôleur.

PCB circuit imprimé.

IMU centrale inertielle.

RF radio-fréquence.

GPS global Positioning System.

GNSS global navigation satellite systems.

Glossaire

Centrale inertielle Instrument utilisé en navigation, capable d'intégrer les mouvements d'un mobile (accélération et vitesse angulaire) pour estimer son orientation (angles de roulis, de tangage et de cap), sa vitesse linéaire et sa position.

timestamp Enregistrement de l'heure et/ou la date d'un événement.

GPS Le système de positionnement global (GPS) est un service public américain qui fournit aux utilisateurs des services de positionnement, de navigation et de synchronisation (PNT). Ce système se compose de trois segments : le segment spatial, le segment de contrôle et le segment utilisateur. L'U.S. Space Force développe, entretient et exploite les segments spatial et de contrôle.

GNSS Le système mondial de navigation par satellite (GNSS) est un terme général décrivant toute constellation de satellites qui fournit des services de positionnement, de navigation et de synchronisation (PNT) à l'échelle mondiale ou régionale.

PIC32 Famille de microcontrôleur 32-bits de Microchip.

FTDI Composant Future Technology Devices International. Sert de convertisseur USB-UART.

harmony Configurateur graphique, générateur de code pour les microcontrôleurs Microchip.

datasheet Document du fabricant fournissant les spécifications d'un produit.



Boîte noire miniaturisée 2023, 1942B

1 Introduction

Les enregistreurs de données de vol, communément appelés "boîtes noires", sont essentiels dans l'aviation contemporaine. Ils jouent un rôle crucial dans la sécurité aérienne et la compréhension des phénomènes aéronautiques en capturant de manière inaltérable des informations vitales. Ces données sont précieuses pour l'investigation d'incidents ou d'accidents aériens, permettant une reconstitution précise des événements à bord des aéronefs. Les boîtes noires sont ainsi des outils indispensables pour détecter des anomalies, des dysfonctionnements, voire des défaillances techniques, contribuant ainsi à renforcer la sécurité dans le secteur aéronautique.

Une boîte noire comprend deux principaux enregistreurs conçus pour résister aux conditions extrêmes : l'enregistreur de la voix du cockpit (CVR), qui enregistre les communications du cockpit, et l'enregistreur de données de vol (FDR), chargé de collecter divers paramètres techniques de l'aéronef. Ces paramètres englobent la vitesse de l'aéronef, son altitude, son orientation, les données des moteurs, les paramètres aérodynamiques, les données de navigation, les commandes du pilote, les transmissions radio, les données environnementales, et bien d'autres éléments essentiels.

Dans ce cadre, le présent projet a pour objectif la collecte et le stockage des données de mesures et de localisation d'un aéronef au moyen d'une centrale inertuelle et d'un système de positionnement GPS/GNSS. Par le biais de la conjonction de ces technologies, il est envisageable d'enregistrer des données d'une précision remarquable concernant les paramètres du vol et la trajectoire empruntée par l'aéronef. En cas de survenue d'un incident, ces enregistrements jouent un rôle déterminant en permettant d'établir les causes potentielles. Pour une analyse approfondie des spécifications techniques, veuillez consulter le cahier des charges complet disponible en annexe.

1.1 Aperçu

- Sauvegarde des données inertielles chaque 500ms par défaut.
- Sauvegarde des données de localisation chaque 5'000ms par défaut.
- Possibilité de configurer les temps de sauvegarde.
- Résistance aux chocs.
- Bonne autonomie / Low power.
- [Global Positioning System. \(GPS\)](#)
- [Global navigation satellite systems. \(GNSS\)](#).
- [Timestamp](#) par satellite.
- [Centrale inertuelle](#) :
 - Accéléromètre 3-axes.
 - Gyroscope 3-axes.
- Charge, lecture et config. par USB-C.

1.2 Tâches à réaliser

Développement et intégration d'un PCB avec capteurs et logging sur carte SD dans un boîtier compact.

- Développement schématique
 - Fonctionnement MCU.
 - Pérophériques de mesures et de sauvegarde / Bus de communication.
 - Gestion batterie
- Routage pour intégration dans boîtier résistant aux chocs.
- Programmation mesure et sauvegarde des données.
 - Configuration MCU.
 - Configuration du périphérique de mesure pour **centrale inertuelle. (IMU)**.
 - Configuration du périphérique de sauvegarde (Carte SD).
 - Configuration du périphérique de localisation **GPS/GNSS**.
 - Configuration et communication avec l'interface.
 - Communication et traitement des données mesurées.

1.3 Schéma de principe

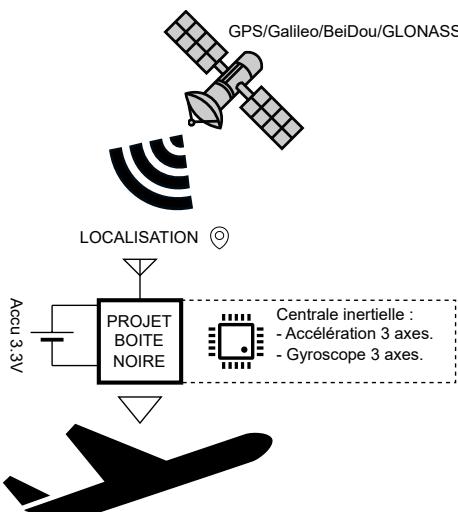


FIGURE 1 – Schéma de principe.

Source: Auteur

Ce système électronique de mini boîte noire pour avion, serait capable d'enregistrer des informations récentes sur les données inertielles et la position d'un vol, dans une mémoire non volatile (carte microSD). Le dispositif, abrité dans un boîtier plastique pour assurer une réception optimale des données **GPS** et une installation compacte. Celui-ci fournirait des données via un port USB pour l'extraction des mesures sur la carte microSD ou pour configurer des paramètres tels que les intervalles de mesures. Les mesures sur les trois axes d'accélération et de vitesse angulaire seraient recueillies par défaut toutes les 500 ms et les données de position **GPS** toutes les 5000 ms. Des intervalles d'enregistrement plus longs sont envisageables pour optimiser la durée de vie de la carte SD, selon la taille et l'organisation des données. Le dispositif sauvegarderait les données des 15 dernières minutes de vol (ou plus) dans un fichier CSV pour traitement ultérieur, selon le principe FIFO. L'objectif principal du prototype est de privilégier la compacité pour minimiser son encombrement à bord de l'avion.

1.4 Planification

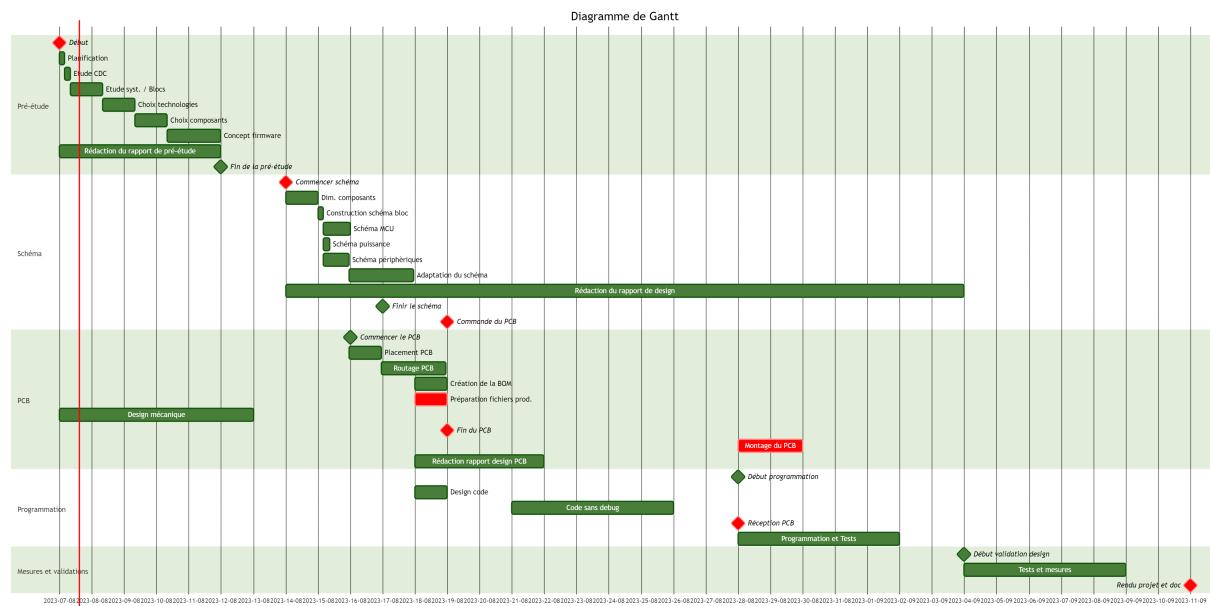


FIGURE 2 – Planification, Diagramme de gantt.

Source: Auteur

No jour de travail	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Date	lundi 7 août 2023	mardi 8 août 2023	mercredi 9 août 2023	vendredi 11 août 2023	samedi 12 août 2023	dimanche 13 août 2023	lundi 14 août 2023	mardi 15 août 2023	mercredi 16 août 2023	jeudi 17 août 2023	vendredi 18 août 2023	samedi 19 août 2023	dimanche 20 août 2023	lundi 21 août 2023	mardi 22 août 2023	mercredi 23 août 2023	jeudi 24 août 2023	vendredi 25 août 2023	samedi 26 août 2023	dimanche 27 août 2023	lundi 28 août 2023	mardi 29 août 2023	mercredi 30 août 2023	jeudi 31 août 2023	vendredi 1 septembre 2023	samedi 2 septembre 2023	dimanche 3 septembre 2023	lundi 4 septembre 2023	mardi 5 septembre 2023	mercredi 6 septembre 2023	vendredi 8 septembre 2023	samedi 9 septembre 2023	lundi 11 septembre 2023			
Cahier des charges																																				
Pré-étude																																				
Dimensionnement + design + schéma																																				
Design de parties mécaniques																																				
Montage PCB																																				
Design software																																				
Réalisation des softwares																																				
Mise en service et tests																																				
Caractéristiques et mesures																																				
Préparation et présentation																																				
Rédaction rapport																																				
Finalisation/corrections/documentation																																				

FIGURE 3 – Planification théorique.

Source: Auteur

1.5 Livrable

- Les fichiers sources de CAO électronique des PCB réalisés
- Tout le nécessaire à fabriquer un exemplaire hardware de chaque :
- fichiers de fabrication (GERBER) / liste de pièces avec références pour commande / implantation
- Prototype fonctionnel
- Modifications / dessins mécaniques, etc
- Les fichiers sources de programmation microcontrôleur (.c / .h)
- Tout le nécessaire pour programmer les microcontrôleurs (logiciel ou fichier .hex)
- Un calcul / estimation des coûts
- Un rapport contenant les calculs - dimensionnement de composants - structogramme, etc.

2 Pré-étude

2.1 Fonctionnement du système

Le microcontrôleur interagît avec 4 périphériques principaux : Avec le **GNSS**, il partage une communication qui lui permet d'obtenir les informations de localisation par le biais de plusieurs systèmes de satellites. Il y a ensuite, la centrale inertuelle qui lui donne accès de une multitude de mesures sur 9 axes, or, ici les mesures gyroscopiques et d'accélération sont exploitées. La carte SD, permet quant-à-elle, de stocker toutes ces données pour avoir minimum les information des 15 dernières minutes de vol. Le dernier périphérique principal **FTDI**, permet d'avoir une interface avec un ordinateur via connexion USB-C.

2.2 Schéma bloc



FIGURE 4 – Schéma bloc.

Source: Auteur

2.3 Description des blocs

Bloc	Description
GNSS.	Récepteur Radio-fréquence. (RF) avec antenne interne/externe et communication UART.
Microcontrôleur. (MCU).	Microcontrôleur PIC32, intelligence du système, basse consommation.
IMU.	Centrale inertuelle, accélération, gyroscope...
Carte SD	Stockage des données de vol.
FTDI.	Convertit la communication USB en série.
Régulateurs.	Le régulateur de charge gère la charge de l'accu. et un régulateur 3.3V le suit.
Batterie.	La batterie est un accu que l'on peut charger par USB et permet une bonne autonomie.

2.4 Choix des composants et technologies

L'objectif de la pré-étude consiste en grande partie à sélectionner méthodiquement les technologies et les composants du projet. Cette partie du travail est essentielle et critique.

2.4.1 Microcontrôleur

Le microcontrôleur nécessite au minimum les périphériques suivants :

2 UART **1 SPI** **1 I2C**

Il est préférable que le **MCU** dispose de différentes configurations de gestion de puissance, notamment des modes d'économie d'énergie, afin d'avoir une maîtrise de la consommation et de permettre une meilleure autonomie. Enfin, le standard de l'école veut que les familles de microcontrôleurs PIC32 (Microchip) sont préférées.

	Memory Flash/SRAM	Automotive	Connectivity	Functional Safety	Graphics	Motor Control	Security	Ultra-Low Power	Touch
PIC32MZ EF FPU MIPS32® M-Class, 252 MHz	512-2048 KB/ 128-512 KB	●	●	●	●		●		●
PIC32MZ DA MIPS32 microAptiv™, 200 MHz	1024-2048 KB/ 256-640 KB		●	●	●		●		●
PIC32CX SG Arm® Cortex®-M4F, 120 MHz	1024 KB/ 256 KB	●	●	●	●	●	●		●
PIC32MK MIPS32 microAptiv, 120 MHz	256-1024 KB/ 128-256 KB	●	●	●	●	●			●
PIC32MX 3/4 MIPS32 M4K, 80-120 MHz	32-512 KB/ 8-128 KB		●	●					
PIC32MX 5/6/7 MIPS32 M4K, 60 MHz	64-512 KB/ 16-128 KB	●	●						
PIC32MX 1/2 XLP MIPS32 M4K, 72 MHz	128-256 KB/ 32-64 KB		●	●				●	
PIC32MX 1/2/5 MIPS32 M4K, 50 MHz	16-512 KB/ 4-64 KB		●	●					
PIC32CM JH Arm® Cortex®-M0+, 48 MHz	128-512 KB/ 16-64 KB	●	●	●		●	●		●
PIC32CM Lx Arm® Cortex®-M23, 48 MHz	256-512 KB/ 32-16 KB		●			●	●		●
PIC32CM MC Arm® Cortex®-M0+, 48 MHz	64-128 KB/ 8-16 KB			●		●			
PIC32MM MIPS32 microAptiv UC, 25 MHz	16-256 KB/ 4-32 KB	●					●		

FIGURE 5 – Familles PIC32.

Source: [Microchip, familles](#)

Sur la figure 5 le **MCU** sélectionné appartient à la gamme MX (Baseline performance-mémoire) et à la famille XLP qui offre notamment la fonctionnalité "Ultra low power" qui est celle qui nous intéresse.

PIC32MX274F256D

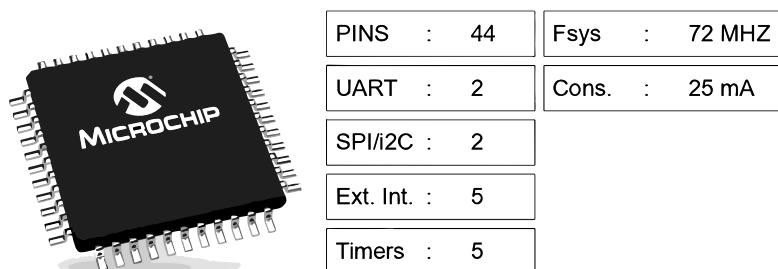


FIGURE 6 – Caractéristiques du PIC32 choisi.

Source: Auteur

2.4.2 Centrale inertielle

Pour la centrale inertielle, il existe un composant avec lequel j'ai déjà acquis une certaine expérience et eu l'occasion d'utiliser et de créer des librairies pour le firmware en C. Celui-ci est performant et très utilisé dans l'industrie. Il est hautement configurable et a le grand avantage de calculer déjà une fusion de capteurs ainsi qu'une compensation de la dérive par les mesures de température. Cela permet notamment d'accéder à des données plus poussées, telles que les quaternions et les angles d'Euler. Il s'agit du **BNO055** de BOSCH.

Caractéristiques importantes :

Résolution gyroscope	:	16	[bits]
Résolution accéléromètre	:	14	[bits]
Résolution magnétomètre	:	~0.3	[μ T]
I_{DD}	:	12.3	[mA]
Dérive de température	:	± 0.03	[%/K]
Dérive accéléromètre	:	0.2	[%/V]
Dérive gyroscope	:	<0.4	[%/V]

Afin de simplifier l'implémentation de ce composant dans le projet, sachant qu'il s'agit d'un boîtier 28-TFLGA difficile à souder ou à mettre au four, il est possible d'utiliser la carte miniature d'Adafruit, cette carte comprend tous les composants passifs requis. Cela facilite ainsi son montage sur le [circuit imprimé. \(PCB\)](#).



FIGURE 7 – Carte d'extension, centrale inertielle.

Source: [Digikey, 4646](#)

Données disponibles :

Température	
Vecteur gravité	XYZ
Orientation compensées, quaternion	WXYZ
Orientation compensée, angle de Euler	HPR
Données gyroscopiques	XYZ
Intensité du champ magnétique	XYZ
Accélération	XYZ

TABLE 1 – Liste des données accessibles.

2.4.3 GPS / GNSS

Pour le **GPS/GNSS**, différents critères entrent en jeu dans le cadre de ce projet : le prix, la facilité d'implémentation, la complexité (par complexité, nous entendons le nombre de fonctionnalités), la consommation et la performance.

Il existe un très grand nombre de récepteurs **RF** pour la navigation. Parmi les plus utilisés dans l'industrie, dont l'implémentation est la plus simple et la documentation la plus complète, il y a plusieurs gammes chez le fabricant **ublox**. Deux composants ont principalement été pris en considération : Le **CAM-M8C-0** (BeiDou, GLONASS, GNSS, GPS, QZSS) avec une antenne omnidirectionnelle interne au composant et différents modes de puissance, ainsi que le **MAX-M10M-00B** (BeiDou, Galileo, GLONASS, GNSS, GPS) sans antenne interne mais avec une consommation de base plus faible.

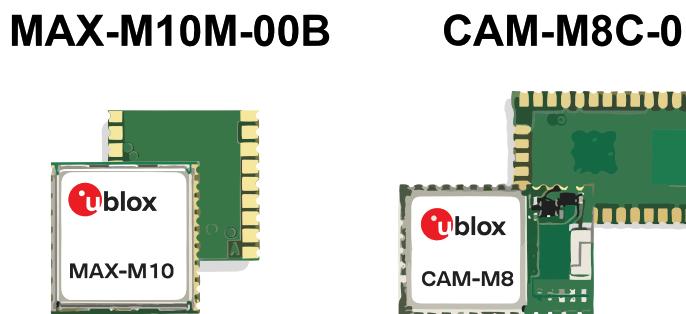


FIGURE 8 – Illustration des deux GNSS.

Source: Digikey, MAX-M10 et CAM-M8C

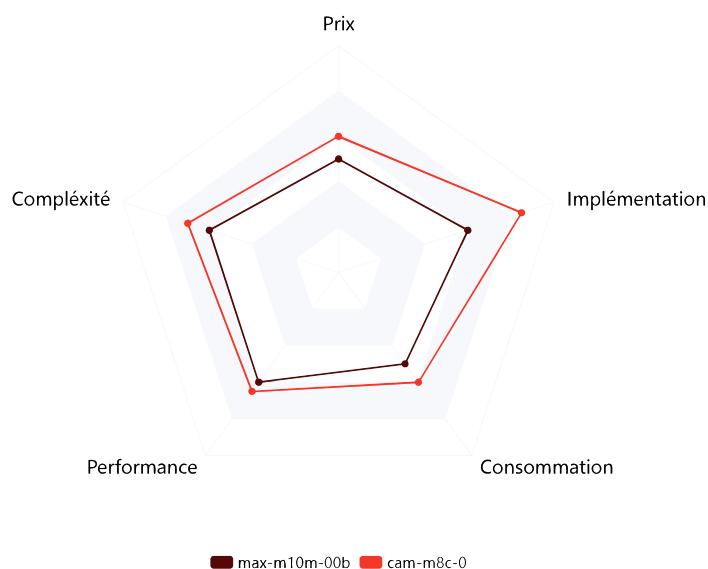


FIGURE 9 – Comparaison GNSS.

Source: Auteur

Malgré les différents avantages que présente le **MAX-M10M-00B** sur la figure 9, le choix s'est porté sur **CAM-M8C-0** grâce à sa facilité d'implémentation et à la garantie du fabricant sur son antenne omnidirectionnelle de qualité. Les détails concernant les protocoles du modules seront décrits lors de la section 5.1.

2.4.4 Carte SD

Les pilotes de carte SD disponibles dans [harmony](#) ne permettent pas une gestion des capacités de stockage trop importantes. Cela signifie, que nous ne pouvons pas avoir des cartes SD de trop grandes capacités, c'est pour cela que nous allons baser nos dimensionnements sur une carte de **256MB**.



FIGURE 10 – Illustration carte SD.

Source: <https://www.oempcworld.com/OEMPCworld-com/017045.html>

2.4.4.1 Estimation de la capacité Admettons les paramètres suivants où S représente une taille de données et T un temps :

S_{SD}	256	[MB]
S_{gyro}	16	[Bytes]
S_{accel}	16	[Bytes]
S_{gnss}	~ 100	[Bytes]
$T_{inertiel}$	0.5	[s]
T_{gnss}	5	[s]
T_{mesMin}	900	[s]

Nous pouvons déduire la taille mémoire que prendra 5 secondes d'enregistrement avec les paramètres par défaut du système :

$$S_{single} = \frac{T_{gnss}}{T_{inertiel}} S_{accel} + S_{gnss} = \frac{5}{0.5} 16 + 100 = 260 \text{ [Bytes]}$$

Nous pouvons enfin calculer à partir de cela, la taille mémoire que prendra 15 minutes d'enregistrement afin de valider le temps de logging minimum :

$$S_{mesures} = \frac{S_{single}}{T_{gnss}} * T_{mesMin} = \frac{260}{5} * 900 = 46'800 \text{ [Bytes]} = 49.8 \text{ [KB]}$$

Nous pouvons déduire avec cette estimation qu'une carte SD de 256MB est largement suffisante et permet de mesurer jusqu'à un temps calculable de cette façon :

$$T_{mesures} = \frac{S_{SD} * T_{gnss}}{S_{single}} = \frac{256 * 10^6 * 5}{260} = \sim 82'051 \text{ Minutes} = \sim 1368 \text{ Heures}$$

Nous estimons donc, que la capacité de stockage de la carte SD est largement suffisante pour le stockage de nombreuses données de vol.

2.5 Batterie, autonomie, charge et régulation

Afin de dimensionner une batterie pour le projet, il faut considérer les différentes consommation :

Liste des consommations principales

Microcontrôleur	24	[mA]	Typ.
Carte-SD	100	[mA]	Max.
Carte-SD	60	[mA]	Moyenne
IMU	12.3	[mA]	Typ.
GNSS	71	[mA]	Max.
GNSS	29	[mA]	Typ.
Totale max	<u>207.3</u>	[mA]	Max.
Totale moyennes	<u>125.3</u>	[mA]	Moyenne

TABLE 2 – Tableau des consommations de courant.

En examinant les consommations typiques et moyennes du tableau 2 à régime constant, et en visant une autonomie de **10 heures**, nous aurions besoin d'une batterie d'une capacité de **1253 mAh**.

2.5.1 Technologies

Concernant la technologie de la batterie, notre objectif, dans un souci d'ergonomie, est de permettre son chargement via un connecteur USB. La technologie offrant actuellement la meilleure densité d'énergie est le **Lithium Ion**¹. Elle présente cependant des inconvénients, énumérés dans le tableau 3.

Avantages	Inconvénient
Haute densité d'énergie.	Risque d'éclatement.
Poids léger.	Risque d'enflammement avec l'eau.
Haute durée de vie.	Sensible a la température.
Charge rapide.	Décharge complète altérante.

TABLE 3 – Tableau avantages/inconvénient LI-ION.

Les inconvénients listés dans le tableau 3 posent un risque pour l'intégrité des données de la carte-SD. C'est pourquoi il est essentiel de bien dimensionner le boîtier ainsi que l'intégration du circuit et de la batterie pour pallier ces dangers. Malgré les risques du Li-Ion, il s'agit d'une technologie très utilisée, y compris dans des domaines sensibles.

2.5.2 Choix de la batterie

La batterie doit avoir une capacité d'au moins **1260 mAh** et être suffisamment compacte pour être intégrée dans un boîtier de petite taille. Une batterie répond à ces critères : il s'agit de la **PICPAL36** de chez Farnell². Les dimensions et caractéristiques de cette batterie sont visibles sur les figures 11.

1. <https://www.epecetc.com/batteries/cell-comparison.html>

2. Distributeur de composants électroniques. [Lien de la batterie](#).

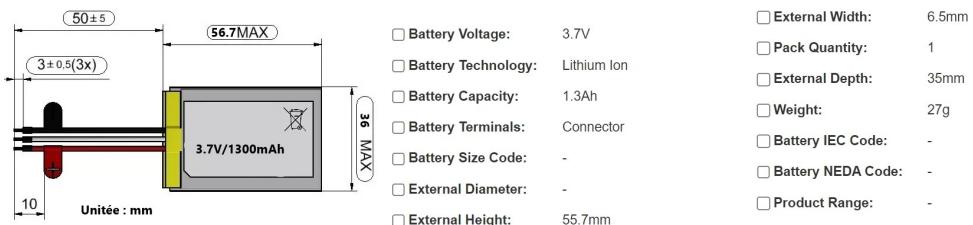


FIGURE 11 – Caractéristiques de la PICPAL36.

Source: PICPAL36 [Farnell](#)

En raison de problèmes avec les commandes, une batterie alternative a dû être trouvée. Il s'agit de la XCell **B520003** Li-Ion, d'une capacité de **1600mAh**. Ses caractéristiques sont présentées dans la figure 12.



FIGURE 12 – Caractéristiques de la XCell B520003

Source: B520003 [Conrad](#)

2.5.3 Régulateur de charge

Le choix du régulateur de charge s'est orienté vers un composant déjà utilisé par l'école, dont les caractéristiques et le montage sont bien connus. C'est un composant performant et fréquemment employé. De plus, il est disponible dans le stock de composants de l'école. Il s'agit du **MCP73871T-2CCI/ML**. Il est configurable et permet la charge des batteries Li-ion et Li-po.

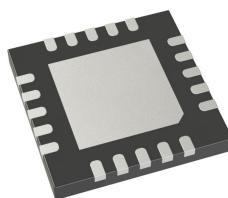


FIGURE 13 – MCP73871T-2CCI/ML.

Source: [Digikey IC BATT CHG LI-ION 1CELL 20QFN](#)

2.6 Systèmes d'économie d'énergies

Au-delà des 10 heures d'autonomie estimées, il est envisageable, pour maximiser le temps de logging, d'adopter des mécanismes d'économie d'énergie. Ces mécanismes pourraient être :

- **Modes de puissance du PIC32** Le microcontrôleur peut basculer entre différents modes de puissance où il réduit à la fois sa consommation et ses fonctionnalités. On pourrait envisager de passer en mode d'énergie restreinte entre les périodes d'enregistrement ou de limiter l'utilisation de certains périphériques.
- **Modes de puissance du GNSS** À l'instar du **MCU**, le composant de navigation dispose de plusieurs modes de fonctionnement qui permettent d'économiser de l'énergie. Un mode particulièrement utile permet de limiter sa cadence de données à 1Hz, étant donné qu'il n'est pas nécessaire qu'il fournisse des informations de localisation aussi fréquemment.
- **Alimentation du GNSS** On pourrait également envisager de ne pas alimenter le **GNSS** lorsqu'il n'est pas en service, afin de supprimer complètement sa consommation entre chaque mesure.
- **Modes de puissance IMU** La centrale inertuelle dispose également de différents modes. Ces modes permettent, par exemple, de limiter les mesures à certains axes ou de la mettre en veille. Elle se met d'ailleurs automatiquement en veille après un certain temps entre les mesures. Ces modes peuvent être utilisés efficacement.
 - **Alimentation de la carte SD** Il est envisageable d'éteindre entièrement la carte SD entre les écritures. Cependant, étant donné que les écritures seront fréquentes, l'intérêt de cette démarche est limité.
- **Mise en veille automatique du système** Si aucun mouvement n'est détecté après un certain délai, il est envisageable que le système s'éteigne entièrement.

2.7 Diagramme des états du système

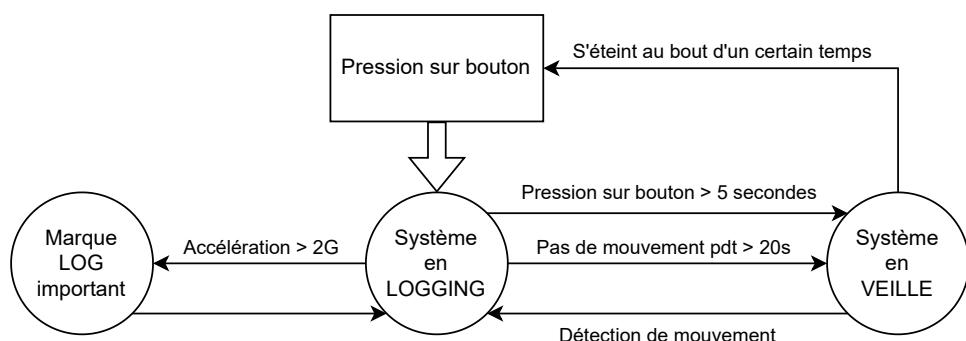


FIGURE 14 – Diagrammes des états (Sans USB).
Source: Auteur

Sur la figure 14 les états de communication par le port USB ne sont pas représentés, il s'agit des modes principaux.

2.8 Estimation du coût du prototype

À partir des composants et des technologies définis lors de la pré-étude, nous pouvons estimer le coût du projet. Cette estimation est présentée dans le tableau 4.

<u>Estimation des coût</u>	
Élément	Prix estimé
PCB	60.-
Batterie	36.-
IMU	26.-
GNSS	24.-
Boîtier	20.-
Composants divers	15.-
MCU	6.-
Régulateurs	6.-
USB & FTDI	3.-
Total	<u>196.-</u>

TABLE 4

Cette estimation est relativement élevée. Il est à noter que le coût du **PCB** pourrait être réduit en cas de commande groupée dans un panel³. Par ailleurs, les prix du boîtier et des composants semblent également être assez élevés.

2.9 Conclusion et perspectives de la Pré-étude

Fonctionnement du système : La conception du système semble possible, le microcontrôleur s'interface avec quatre périphériques essentiels : un **GNSS** pour la localisation, une centrale inertie pour des mesures sur 9 axes (priorisant les mesures gyroscopiques et d'accélération), une carte SD pour le stockage (conservant au moins 15 minutes de données de vol), et un périphérique **FTDI** pour l'interface avec un ordinateur via USB-C.

Choix des composants et technologies :

- *Microcontrôleur* : Requiert au moins 2 UART, 1 SPI, et 1 I2C. Le PIC32MX274F256D est privilégié car il offre divers modes d'économie d'énergie.
- *Centrale inertie* : Le BNO055 de BOSCH a été choisi, car il offre une variété de mesures avancées et une facilité d'implémentation grâce à la carte Adafruit.
- *GPS/GNSS* : Le CAM-M8C-0 d'ublox a été retenu pour sa facilité d'implémentation et son antenne interne omnidirectionnelle.
- *Carte SD* : Une capacité de 256MB a été choisie à cause des limites du pilote. Une telle carte est amplement suffisante pour enregistrer toutes les données de vol.
- *Batterie, charge, et régulation* : Une capacité de 1253 mAh est nécessaire pour atteindre l'autonomie souhaitée de 10 heures. Une batterie compact de 1600 mAh a été retenue.

Estimation des coûts : L'estimation totale pour le projet s'élève à environ 196 CHF. Néanmoins, il existe des possibilités d'économie, en particulier par une commande groupée du **PCB**.

En conclusion, cette pré-étude a décortiqué le projet de façon méthodique pour sélectionner les composants et technologies appropriés. Elle a également fourni des estimations de coûts et identifié des mécanismes d'économie d'énergie potentiels. La prochaine étape consistera à entamer la phase de conception en utilisant ces informations utiles.

3. Regroupement de plusieurs circuits imprimés en un seul, pour diminuer le coût.

3 Développement du schéma électronique

Dans cette section, nous décrirons la phase principale du développement ainsi que la démarche suivie pour élaborer le schéma électronique du projet.

3.1 Blocs développés

Pour faciliter le développement et la lecture du schéma, il est judicieux de diviser le système en plusieurs blocs. Une structure a ainsi été définie, divisant le circuit en trois blocs principaux : **Microcontrôleur 3.2** (Intelligence du système, connexion du programmeur et LED de vie.), **Périphériques 3.3** (GNSS, IMU, FTDI, connecteur USB, Carte SD.) et **Alimentations 3.4** (Connecteur batterie, gestion de charge, régulateurs de tension et système ON/OFF.).

Nous pouvons sur la figure 15 observer les différentes interactions entre les blocs, elles sont par la suite décrites dans le tableau 5.



FIGURE 15 – Blocs du système.

Source: Auteur

Connexion/s	Description
INT_IMU	Interruption de la centrale inertuelle, informe le MCU et peut allumer le système.
RST_IMU	Permet de réinitialiser la IMU .
PWR_HOLD	Le MCU peut se maintenir alimenté par cette connexion.
BAT_STAT	Fournit le statut de la batterie au MCU .
Button_MF	Fournit le niveau logique du bouton au MCU .
BAT_READ	Le MCU peut activer la lecture de la tension de la batterie.
BAT_LVL	Lecture analogique de la tension de batterie.
SDA1, SCL1	Communication I2C avec l' IMU .
SDA2, SCL2	Communication I2C optionnelle avec le GNSS .
UI1TX/RX...	Communication UART avec le FTDI pour l'USB.
U21TX/RX	Communication UART avec le GNSS .
RESET_N	Permet de réinitialiser le GNSS .
TIMEPULSE	Permet de mesurer le temps par un signal pulsé.
EXTINT	Permet de gérer le mode d'alimentation du GNSS .
SCK, MOSI...	Communication SPI avec la carte SD.
R1,2,3,4	Résistances de PULL-UP pour communication I2C.

TABLE 5 – Description des connexions

3.2 Microcontrôleur

3.2.1 Connexion

Pour utiliser le microcontrôleur, il est nécessaire de définir ses entrées/sorties en se référant à son [datasheet](#). Ce dernier permet de connaître les connexions dédiées à certains bus de communication ou à des entrées analogiques.

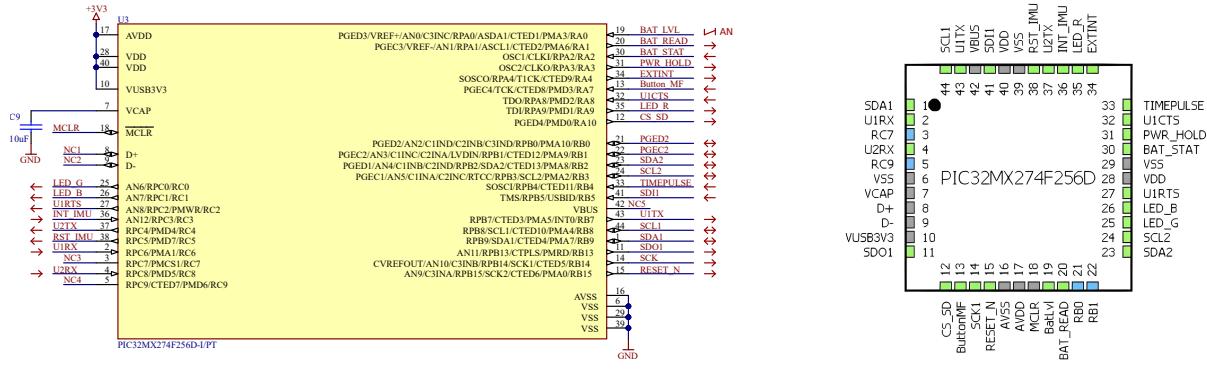


FIGURE 16 – Configuration des PINs du microcontrôleur.

Source: Auteur

Pour valider les connexions de la figure 16a, une vérification a été réalisée à l'aide du configurateur graphique [harmony](#), comme le montre la figure 16b. Cette étape a confirmé la possibilité d'attribuer certaines fonctions à des PINs spécifiques. Les PINs non utilisés du microcontrôleur sont redirigées vers un connecteur. Dans le contexte du prototype, cela permet de les exploiter ailleurs sur la carte.

3.2.2 Programmateur et reset

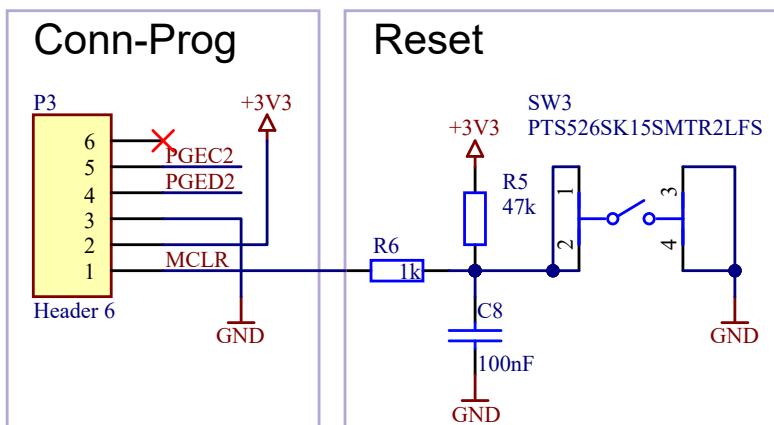


FIGURE 17 – Schéma programmateur et reset.

Source: Auteur

Sur la figure 17, nous pouvons observer le connecteur de programmation *P3*. La connexion **MCLR** (Master Clear), qui permet de réinitialiser le **MCU** lors de sa programmation, est suivie d'un bouton pour permettre une réinitialisation manuelle.

3.2.3 LED de vie

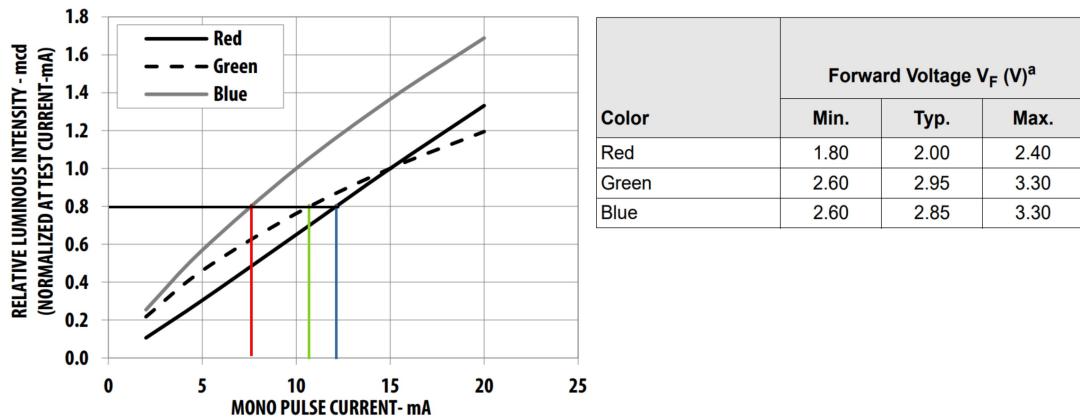


FIGURE 18 – Données de la LED
Source: [datasheet ASMB-KTF0-0A306](#)

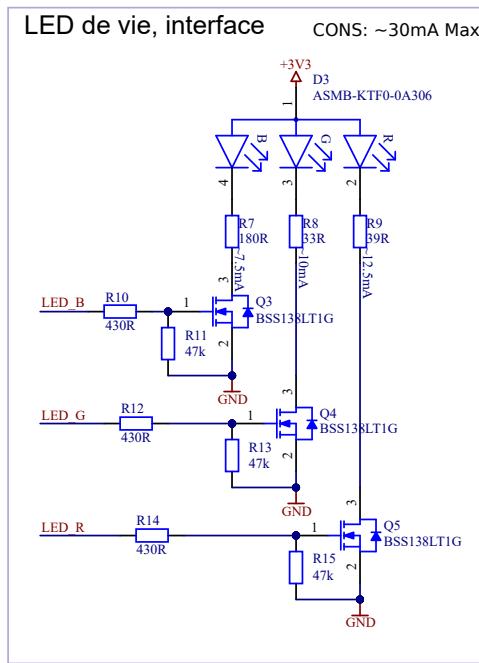


FIGURE 19 – Schéma de la LED de vie.
Source: Auteur

Les résistances de la figure 19 ont été dimensionnées en respectant les caractéristiques des LEDs pour chacune des couleurs (voir figure 18). Ces dernières éclairent à 80% de leur luminosité nominale dans un souci d'économie d'énergie, leur utilité étant réduite à ce stade de prototypage.

Exemple dimensionnement résistance LED bleue

$$R_{blue} = \frac{(V_{cc} - V_{blue})}{I_b} = \frac{(3.3 - 2.85)}{12 * 10^{-3}} = 37.5\Omega$$

3.3 Périphériques

Dans cette section, nous allons décrire les schématiques des périphériques du système.

3.3.1 Carte SD

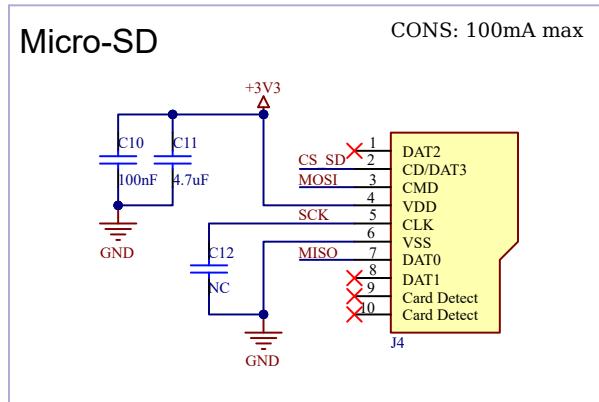


FIGURE 20 – Branchement carte SD.

Source: Auteur

La carte SD de la figure 20 est branchée en respectant la connectique pour une communication SPI simple. De plus, un condensateur est connecté entre **SCK** et **GND**. Toutefois, ce condensateur ne sera pas monté sauf si un filtrage sur la ligne de clock s'avère nécessaire.

3.3.2 Centrale inertie

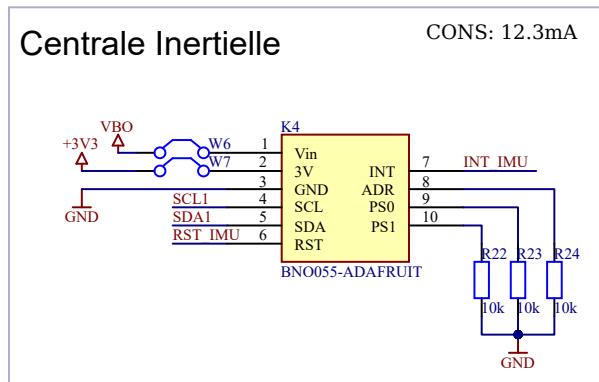


FIGURE 21 – Schéma centrale inertie.

Source: Auteur

La centrale inertie, étant déjà montée sur une carte d'extension, ne nécessite pas de composants passifs additionnels. Trois résistances de PULL-DOWN sont présentes, respectivement sur **ADR**, **PS0** et **PS1**. Elles permettent de configurer l'adresse et l'interface de communication de l'**IMU**. De plus, un jumper⁴ est présent sur les alimentations. Ceci permet de choisir si la centrale inertie est alimentée indépendamment du reste du système (pour pouvoir l'allumer séparément) ou avec la même alimentation que le reste du système.

4. Connecteur permettant de faire ou de rompre une connexion.

3.3.3 GNSS

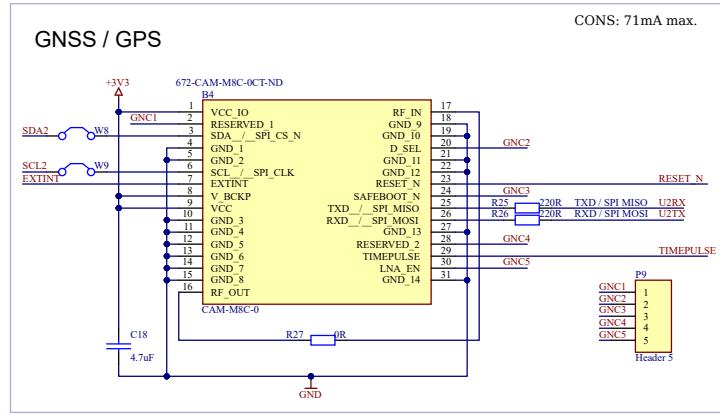


FIGURE 22 – Schéma du GNSS.

Source: Auteur

Le **GNSS** est connecté selon un schéma présenté dans son [Hardware Integration Manual](#), à la section 2.2. Ce schéma représente un design minimal pour recevoir des données via l’interface UART. Par rapport à la figure 22, les différences sont les suivantes : l’interface I₂C est également disponible, la PIN **RESET_N** est connectée au **MCU** et les PINS non-utilisées sont reliées à un connecteur afin de permettre leur utilisation éventuelle dans le cadre du prototype.

3.3.4 USB - FTDI

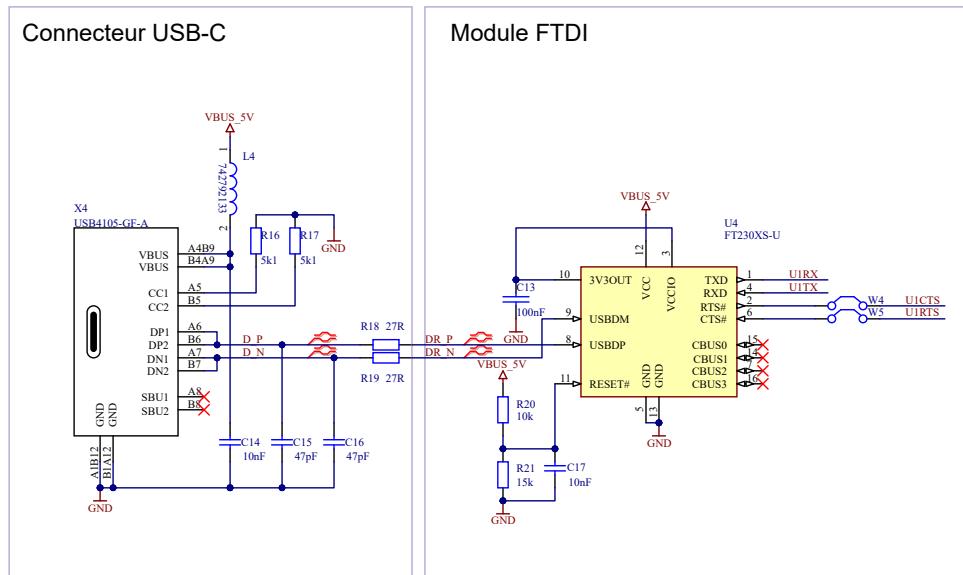


FIGURE 23 – Schéma connecteur USB et FTDI.

Source: Auteur

Le schéma du connecteur USB-C est conçu pour présenter une résistance aux interférences. Son signal 5V est filtré par une *ferrite bead*. Les pistes différentielles de l’USB sont ensuite reliées au module **FTDI**, qui est interfacé en UART et est alimenté par l’USB.

3.4 Alimentations

Dans cette section, nous allons décrire les fonctionnement et dimensionnements principaux du bloc alimentation comprenant les différents régulateurs et la batterie.

3.4.1 Chargeur de batterie

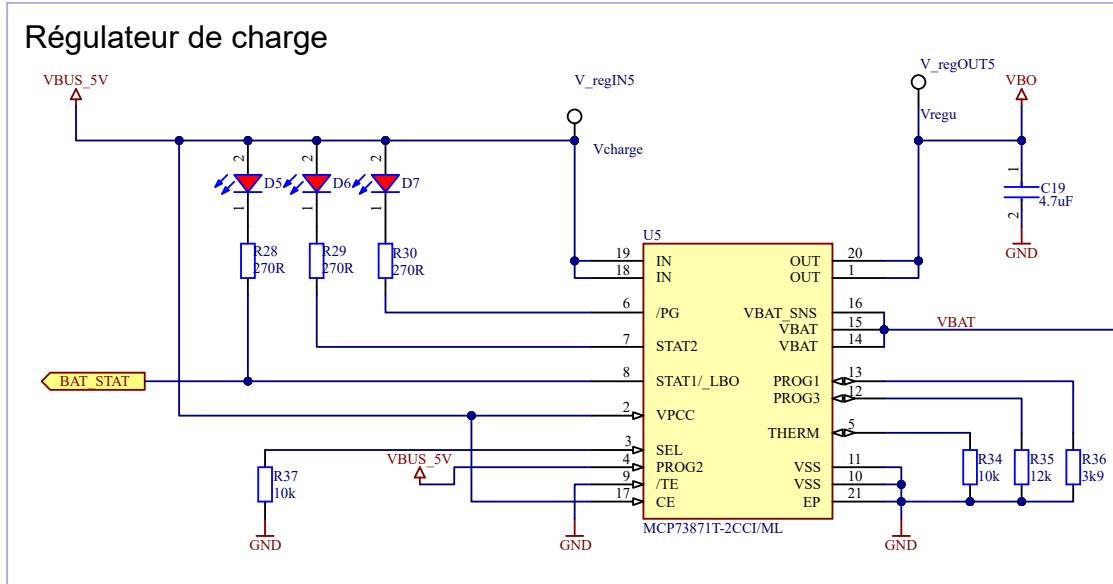


FIGURE 24 – Schéma chargeur de batterie.

Source: Auteur

Le composant **MCP73871T-2CCI/ML** est configurable pour délivrer un courant de charge programmable à la batterie. Ce courant se sépare en deux phases : la charge normale et la charge de terminaison. Cette dernière est activée lorsque la batterie approche de sa capacité maximale. Pour déterminer ces courants, il convient de se baser sur la capacité de la batterie et de certains ratios standards associés.

Où :

$C = 1600mAh$ est la capacité de la batterie.

$k_{term} = 0.05$ est le ratio de fin de charge.

$k_{chg} = 0.15$ est le ratio de charge.

Le courant de terminaison peut être calculé ainsi :

$$I_{term} = C * k_{term} = 1600 * 0.05 = 80 mA$$

Par conséquent, d'après la [datasheet](#) ([Equation 4-2](#)) la résistance de programmation de fin de charge vaudrait :

$$R_{prog3} = \frac{1000V}{I_{term}} = \frac{1000}{65 * 10^{-3}} = 12.5 k\Omega \Rightarrow 12 k\Omega$$

Nous pouvons de la même façon cacluler la résistance de charge **Rprog1** :

$$R_{prog1} = \frac{1000V}{C * k_{chg}} = \frac{1000}{1600 * 10^{-3} * 0.15} = 4.166 k\Omega \Rightarrow 3.9 k\Omega$$

3.4.2 Système d'enclenchement

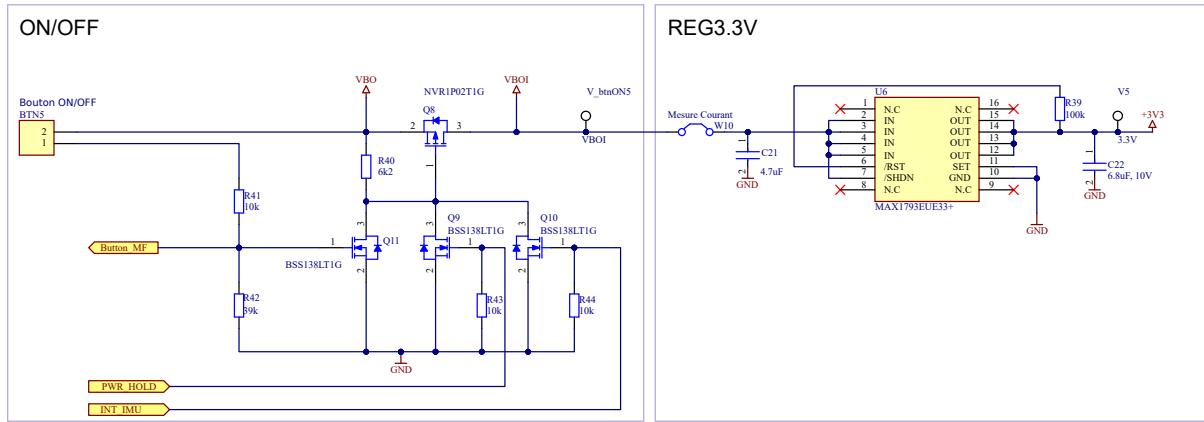


FIGURE 25 – Schéma allumage du système.

Source: Auteur

Sur la figure 25, nous observons les différentes manières de connecter la tension régulée de la batterie au régulateur linéaire 3.3V. Quand cette connexion n'est pas réalisée, le système n'est pas alimenté, à l'exception de l'**IMU** qui peut être alimenté de manière autonome. Pour mettre en marche le système, trois options sont possibles :

- Le microcontrôleur peut maintenir le système alimenté avec **PWR_HOLD**.
- L'**IMU** peut 'allumer le système avec son interruption **INT_IMU**.
- Un bouton peut être pressé pour enclencher la carte **Button_MF**.

3.5 Conclusion et perspectives de l'étude

Tout au long de cette étude, nous avons pu décomposer et analyser le processus de développement du schéma électronique. La démarche a priorisé la modularité, avec une division du système en trois blocs : **Microcontrôleur 3.2**, **Périphériques 3.3** et **Alimentations 3.4**. Cette approche a permis une meilleure lisibilité et une facilité d'adaptation quant au prototypage. Les différentes interactions et connexions entre les composants ont été détaillées.

Par la suite, le circuit imprimé sera développé, les composants placés et routés le tout dans un design petit et compact. La modularité du projet, permettra des tests empiriques sur les différents blocs du système, lors de la mise en service.

4 Développement du circuit-imprimé

Dans cette section, nous nous attarderons sur le processus de conception et de développement du **PCB**. Nous aborderons également les différents aspects mécanique d'intégration du projet, en essayant d'analyser les différentes contraintes des composants et la structure globale. Cette analyse combinée des aspects électroniques et mécaniques est cruciale pour assurer la réussite du projet.

4.1 Mécanique du projet

L'un des objectifs majeurs du projet est de concevoir un produit qui, par sa petite taille et son design compact, se prête à une installation qui se veut à la fois discrète et non encombrante.

4.1.1 Choix du boitier

Pour réduire la taille au maximum, un processus itératif a eu lieu, consistant à trouver un boîtier plastique le plus petit possible et d'essayer en respectant les contraintes de taille, de placer les composants, puis, d'observer et déduire si la taille est suffisante. Grâce à cette procédure itérative, un boîtier a été déterminé : le **SIC5-9-3B** de chez **TAKACHI**.

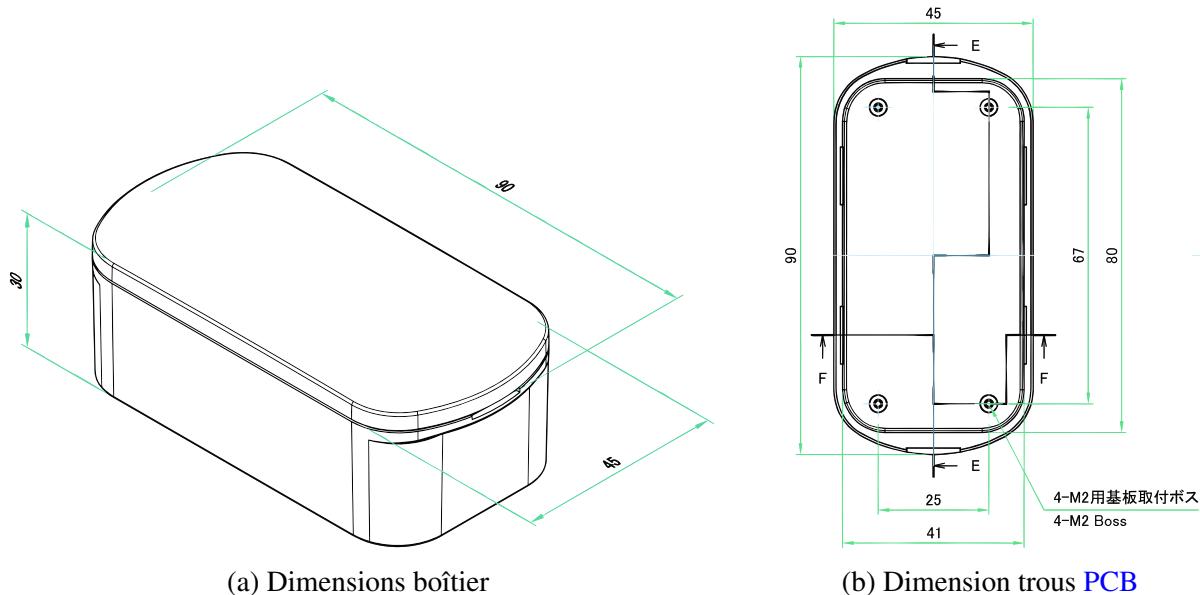


FIGURE 26 – Dimensions du SIC5-9-3B.

Source: [Datasheet du SIC5-9-3B, TAKACHI](#)

4.1.2 Placement des composants

Désormais, sachant que nous connaissons les dimensions du **PCB** grâce à la figure 26b, nous pouvons positionner les composants de manière optimisée pour le routage et la mécanique. En prenant différents éléments en compte :

Élément	Contraintes, règles et dispositions particulières
> Connecteur USB	: Placer proche du bord, usinage du boîtier pour passer chargeur.
> Connecteur µSD	: Proche du bord, facile d'accès.
> GNSS	: Isolé des autres composants, faciliter réception.
> Carte BNO055	: Espace requis, permettre de le braser sur le PCB .
> LED de vie	: Dégagée, peut être déportée avec un guide-lumière.
> Connecteur batterie	: Accessible et permettre de passer les fils de l'autre côté du PCB .
> Connecteur bouton ON/OFF	: Bouton déporté sur le boîtier.
> Bouton reset	: Accessible pendant le développement et en cas de bug.

TABLE 6 – Tableau des contraintes de placement.

Source: Auteur

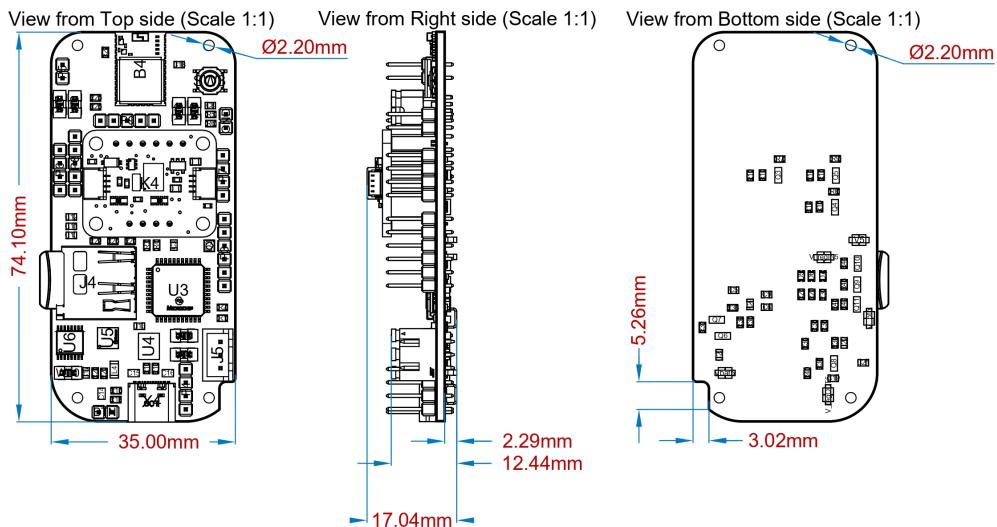


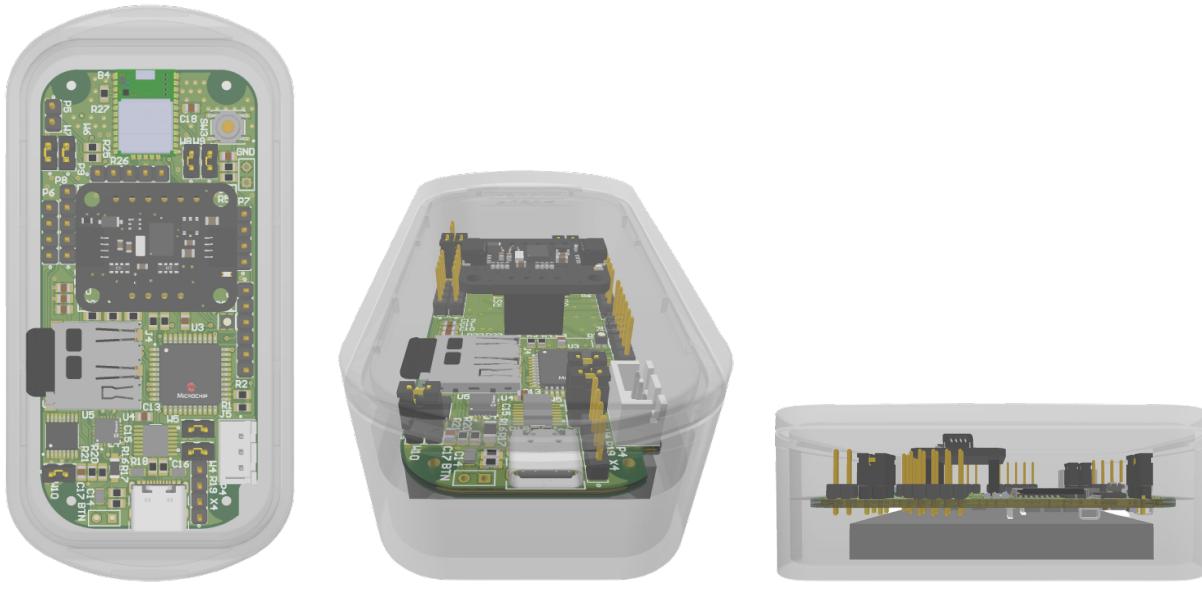
FIGURE 27 – Placement des composant et dimensions de la carte.

Source: Auteur

En considérant les contraintes des différents éléments de la table 6, nous obtenons un design tel que celui de la figure 27. Nous y observons plusieurs solutions apportées telles que : un trou (En bas à droite du PCB) permettant de passer les fils de la batterie, une carte SD et un connecteur USB placés proche du bord, un bouton de reset accessible et un GNSS isolé du reste pour garantir l'intégrité de l'antenne.

4.1.3 Assemblage

Lors de cette section, nous allons assembler le circuit imprimé avec ses composants et la batterie, dans le modèle 3D du boîtier **SIC5-9-3B**.



(c) *Vacuum*, 1

– Vues 3d de l'

Chap. 11: The First World War and its Aftermath 221

4.2 Bill of materials

Avec le schéma électronique et la mécanique désormais finalisés, nous pouvons dresser une liste des composants nécessaires. La table 7 répertorie les composants **hors-stock** qui ont été commandés auprès de différents fournisseurs. La table 8, quant à elle, liste les composants **en stock** disponibles dans les locaux de l'école supérieure, avec leurs coûts respectifs. Il est à noter que ces derniers n'ont pas été commandés. Les tableaux 9 et 10 détaillent respectivement les valeurs des résistances et des condensateurs utilisés pour le projet.

Composant	Fournisseur	Référence	Quantité	Prix [CHF]
Batterie PICPAL36	Farnell	PICPAL36	1	36,54
BNO055-Adafruit	Digikey	1528-4646-ND	1	26,06
GNSS	Digikey	672-CAM-M8C-0CT-ND	1	23,49
Boitier boîte noire	Farnell	CHH9840BK	1	6,84
Guide lumière	Digikey	LFB075CTP-ND 492-1980-ND	4	6,13
PIC32MX274F256D	Digikey	PIC32MX274F256DT-I/PTCT-ND	1	6,07
Bouton poussoir ext.	Digikey	EG5949-ND	1	4,54
Connecteur uSD	Digikey	732-3819-1-ND	1	2,71
LEDS	Digikey	516-3906-1-ND 732-4985-1-ND	4	2,05
Connecteur USB	Digikey	2073-USB4105-GF-ACT-ND	1	1,97
Mosfet P	Digikey	NVR1P02T1GOSCT-ND	2	0,7
Bouton tactile reset	Digikey	CKN12221-1-ND	1	0,35
Ferrite Bead 600 Ohm	Digikey	732-4655-1-ND	1	0,22
Connecteur batterie	Farnell	B3B-XH-A (LF)(SN)	1	0,11
TOTAL		117.78		

TABLE 7 – Liste des composants hors-stocks.

Composant	Fournisseur	Référence	Quantité	Prix [CHF]
Régulateur linéaire 3.3V	Digikey	MAX1793EUE33+-ND	1	3.85
IC chargeur de batterie lithium	Digikey	MCP73871T-2CCI/MLCT-ND	1	2.1
Mosfet Canal N	Digikey	BSS138LCT-ND	7	2.1
FTDI, USB-UART	Digikey	768-1154-5-ND	1	1.97
TOTAL		10.02		

TABLE 8 – Liste des composants en stock.

Résistances														
Valeur	27R	220R	270R	430R	1k	5k1	6k2	8k2	10k	15k	39k	47k	51k	100k
Quantité	2	3	4	6	1	2	1	1	12	2	4	1	4	1

TABLE 9 – Liste des valeurs des résistances.

Condensateurs						
Valeur	47pF	10nF	100nF	4.7uF	6.8uF	10uF
Quantité	2	2	7	4	1	2

TABLE 10 – Liste des valeurs des condensateurs.

4.3 Routage

Une fois les composants placés, les pistes et les plan ont pu être

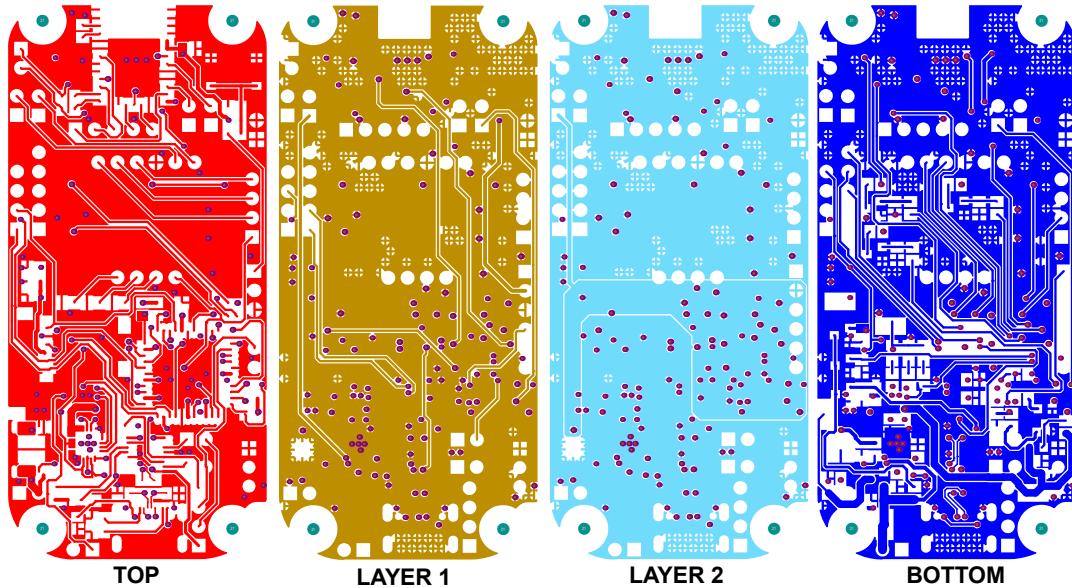


FIGURE 29 – Routage des différentes couches.

Source: Auteur

Les justifications du routage des différentes couche sont présentées dans la table 11.

TOP	<ul style="list-style-type: none"> > Composants à braser au four. > Signaux proches du MCU. > Signaux sortant du GNSS. > Pistes horizontales. > Plan de masse. > Connecteurs pour boutons et batterie.
LAYER 1	<ul style="list-style-type: none"> > Pistes de signaux verticaux. > Plan de masse.
LAYER 2	<ul style="list-style-type: none"> > Plan de masse homogène. > Plan de 3.3V.
BOTTOM	<ul style="list-style-type: none"> > Composants des régulateurs. > Mosfets et composants passifs. > Pistes de signaux verticaux. > Plan de masse.

TABLE 11 – Description des éléments du routage.

Le routage étant complété et contrôlé, il a pu être commandé sous forme de panel avec un pochoir chez [Eurocircuit](#). Celui-ci a eu des retards de production et de livraison par UPS.

4.4 Conclusion et perspectives de la conception

Durant cette section, nous avons conçu le circuit imprimé de la boîte noire, défini son implantation dans un petit boîtier plastique, vérifié l'intégration du circuit et de la batterie à l'intérieur, et avons décrit les différentes justifications de développement. Par la suite, la carte sera montée, brasée, et les phases de programmation sur la carte auront lieu.

5 Développement du firmware

Lors de cette section, nous décrirons le processus de développement du firmware du PIC32. Les décisions prises seront expliquées et les différents algorithmes seront illustrés et décrits.

En premier lieu nous allons analyser lors de la section 5.1 comment traiter les données du **GNSS**, il s'agit d'un élément critique.

5.1 Protocoles du GNSS

Il existe différents protocoles pour le format de données de localisation. Le **CAM-M8C-0** supporte plusieurs protocoles, visibles sur la figure 30 :

1.15 Protocols and interfaces

Protocol	Type
NMEA	Input/output, ASCII, 0183, version 4.0 (Configurable to V 2.1, V 2.3 or V4.1)
UBX	Input/output, binary, u-blox proprietary
RTCM	Input, message 1, 2, 3, 9

FIGURE 30 – Protocoles disponibles.

Source: [Datasheet du CAM-M8C-0](#)

NMEA : Norme établie par la *National Marine Electronics Association*. Elle suit un format **ASCII**.

UBX : Format propriétaire de u-blox avec des données **binaires**. Il permet d'envoyer des trames de configuration.

RTCM : Protocole pour des données GPS différentielles. Établi par la *Radio Technical Commission for Maritime Service*.

Le **CAM-M8C-0** est configuré, par défaut, pour envoyer des messages au format NMEA, comme illustré sur la figure 31.

8 Default messages

Interface	Settings
UART Output	9600 Baud, 8 bits, no parity bit, 1 stop bit Configured to transmit both NMEA and UBX protocols, but no UBX messages and only the following NMEA have been activated at start-up: GGA, GLL, GSA, GSV, RMC, VTG, TXT
UART Input	9600 Baud, 8 bits, no parity bit, 1 stop bit, Autobauding disabled Automatically accepts following protocols without need of explicit configuration: UBX, NMEA, RTCM The GPS receiver supports interleaved UBX and NMEA messages.
DDC	Fully compatible with the I ² C industry standard, available for communication with an external host CPU or u-blox cellular modules, operated in slave mode only. NMEA and UBX are enabled as input messages, only NMEA as output messages Maximum bit rate 400 kb/s.
TIMEPULSE (1 Hz Nav)	1 pulse per second, synchronized at rising edge, pulse length 100 ms

FIGURE 31 – Protocole par défaut.

Source: [Datasheet du CAM-M8C-0](#)

5.1.1 Messages NMEA

Comme nous avons pu constater sur la figure 31, il y a plusieurs messages **NMEA** :

GGA, GLL, GSA, GSV, RMC, VTG, TXT

Ceux-ci sont décrits dans le [manuel de référence du protocole NMEA](#) sur la figure 32

1. Output Messages.....	1-1
GGA —Global Positioning System Fixed Data.....	1-2
GLL—Geographic Position - Latitude/Longitude.....	1-3
GSA—GNSS DOP and Active Satellites.....	1-3
GSV—GNSS Satellites in View	1-4
MSS—MSK Receiver Signal.....	1-4
RMC—Recommended Minimum Specific GNSS Data.....	1-5
VTG—Course Over Ground and Ground Speed	1-5
ZDA—SiRF Timing Message	1-6
150—OkToSend	1-6

FIGURE 32 – Messages NMEA.

Source: [Manuel du protocole NMEA](#)

Les différents messages de la figure 32 présentent des différentes données sous divers formats. Les messages peuvent être activés ou désactivés en configurant le module u-blox.

5.1.2 Interprétation des données NMEA

Avant d'avoir le **PCB** monté et exploitable, sachant que nous connaissons le protocole par défaut du module, nous pouvons analyser des données **NMEA** pour mieux les comprendre et les traiter par la suite. Pour ce faire, le site <https://www.nmeagen.org/> permet de générer des coordonnées avec le protocole **NMEA**.



FIGURE 33 – Application d'une localisation NMEA.

Source: nmeagen.org, aéroport de La Blécherette, Lausanne

Sur la figure 34, les messages de la figure 33 sont décodés via un [analyseur NMEA en ligne](#).

Date	Time UTC	Latitude	Longitude	Altitude	Fix	Fix quality	PDOP	HDOP	VDOP	Inview Sats	Active sats	Active GPS
N/A	08:06:14	46.54516667	6.6169	0.0	N/A	GPS	N/A	1.0	N/A	N/A	N/A	N/A
N/A	08:06:14	46.54516667	6.6169	0.0	F	GPS	1.0	1.0	1.0	N/A	12	12
21/08/2023	08:06:14	46.54516667	6.6169	0.0	F	GPS	1.0	1.0	1.0	N/A	12	12

FIGURE 34 – Messages NMEA décodés.

Source: [NMEA online analyser](#)

5.1.3 Code décodeur de trame NMEA

Afin de développer le code du décodeur **NMEA**, la librairie **minmea** de licence libre, a pu être utilisée, modifiée et adaptée. Un algorithme a ensuite été mis en place :

- 1 Lecture du type de message (**GBS**, **GGA**, **GLL**, **GSA**, **GST**, **GSV**, **RMC**, **VTG**, **ZDA**) ;
- 2 Parsing des valeurs correspondantes à l'ID du message ;
- 3 Sauvegarde des valeurs dans une structure.

Code Source 1 – gnss_posGet_nmea(...), décodage **NMEA**.

```
// Get message ID, strict
*msg_id = minmea_sentence_id(message, true);
// Parse message depending on ID
if (*msg_id == MINMEA_SENTENCE_GBS)
    minmea_parse_gbs(&sentences->gbs, message);
else if (*msg_id == MINMEA_SENTENCE_GGA)
    minmea_parse_gga(&sentences->gga, message);
else if ...
```

Dans le code 1, on fait appel à une fonction qui va masquer les caractères du type du message. Ensuite, selon le message dont il s'agit, on le décode de différentes façons.

Code Source 2 – minmea_parse_gbs(...), décodage message **GBS**.

```
bool minmea_parse_gbs(struct minmea_sentence_gbs *frame, const char
    ↵ *sentence)
{
    // $GNGBS,170556.00,3.0,2.9,8.3,,,,*5C
    char type[6];
    if (!minmea_scan(sentence, "tTffff", ,
                      type,
                      &frame->time,
                      &frame->err_latitude,
                      &frame->err_longitude,
                      &frame->err_altitude,
                      &frame->svid,
                      &frame->prob,
                      &frame->bias,
                      &frame->stddev
                      ))
        return false;
    if (strcmp(type+2, "GBS"))
        return false;
    return true;
}
```

Dans le code 2, on masque les éléments du message selon son format, d'une manière similaire à la fonction standard **scanf(...)**.

Les données spécifiques au message **GBS** sont ensuite stockées dans une structure, que l'on peut observer dans le code 3.

Code Source 3 – Structure **GBS**.

```
struct minmea_sentence_gbs {
    struct minmea_time time;
    struct minmea_float err_latitude;
    struct minmea_float err_longitude;
    struct minmea_float err_altitude;
    int svid;
    struct minmea_float prob;
    struct minmea_float bias;
    struct minmea_float stddev;
};
```

Sachant que plusieurs types de messages peuvent être interceptés, chacune des structures des différents formats de messages est elle-même stockée dans une structure que l'on peut observer dans le code 4.

Code Source 4 – Structures messages.

```
typedef struct minmea_messages{
    struct minmea_sentence_gbs gbs;
    struct minmea_sentence_rmc rmc;
    struct minmea_sentence_gga gga;
    struct minmea_sentence_gll gll;
    struct minmea_sentence_gst gst;
    struct minmea_sentence_gsa gsa;
    struct minmea_sentence_gsv gsv;
    struct minmea_sentence_vtg vtg;
    struct minmea_sentence_zda zda;
}minmea_messages;
```

Les données sont ensuite traitées dans l'application. Pour paramétriser le **CAM-M8C-0**, des messages **UBX** peuvent être envoyés. Pour ce faire, des éléments de la librairie *ubxlib* peuvent être repris, notamment les fonctions d'encodage et de décodage des messages **UBX**.

5.2 Configuration des périphériques

Les périphériques utilisés par le microcontrôleur doivent être paramétrés et initialisés. Le configurateur [harmony](#) permet de le faire simplement.



5.2.1 Timers

Lors de cette section, 2 timers vont être configurés, ceux-ci ont des utilités différentes :

Timer 1 sert pour les attentes bloquantes précises. 1 [ms]

Timer 2 gère les délais entre les mesures et l'affichage des LEDs. 10 [ms]

Dimensionnement timer 1

$$f_{sys} = 72 \text{ MHz}$$

$$f_{per} = 1 \text{ kHz}$$

$$presc = 8 [-]$$

$$C_{T1} = \frac{f_{sys}}{f_{per} * presc} - 1 = \frac{72 * 10^6}{1 * 10^3 * 8} - 1 = 8'999 [-]$$

Dimensionnement timer 2

$$f_{sys} = 72 \text{ MHz}$$

$$f_{per} = 100 \text{ Hz}$$

$$presc = 16 [-]$$

$$C_{T1} = \frac{f_{sys}}{f_{per} * presc} - 1 = \frac{72 * 10^6}{100 * 16} - 1 = 44'999 [-] \quad (1)$$

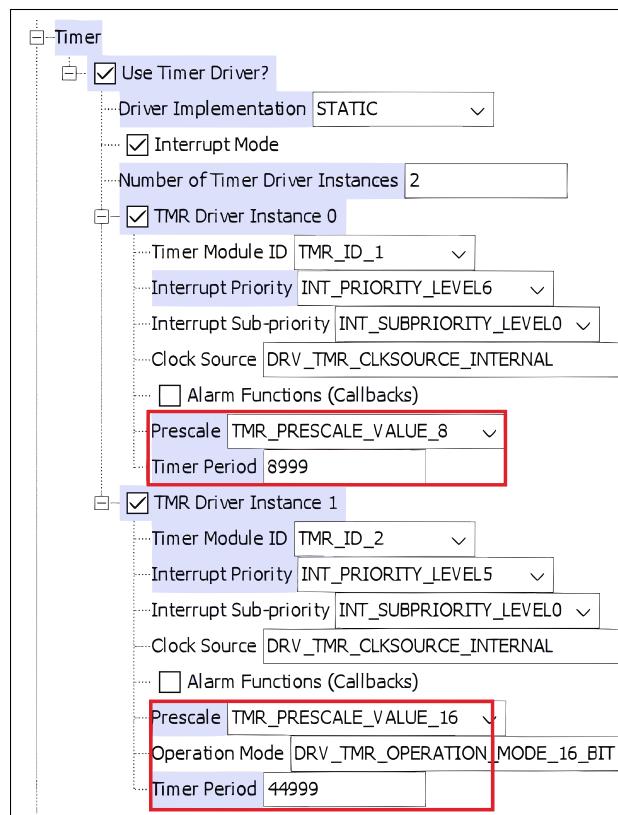


FIGURE 35 – Configuration des timers.

Source: Auteur

Code Source 5 – Interruption **timer 1**, 1 [ms]

```
void delayTimer_callback(){
    /* Increment delay timer */
    timeData.delayCnt++;
}
```

Code Source 6 – Interruption **timer 2**, 10 [ms]

```
void stateTimer_callback()
{
    /* Increment all counters */
    timeData.ledCnt++;
    timeData.measCnt[BN0055_idx]++;
    timeData.measCnt[GNSS_idx]++;
    timeData.tmrTickFlag = true;
    /* When the button is pressed, the hold time is counted. */
    if(timeData.flagCntBtnPressed){
        timeData.cntBtnPressed++;
    }
    /* Do debounce on button every 10 ms */
    DoDebounce(&switchDescr, ButtonMFStateGet());
    /* Start a measure set each IMU period */
    if( ( timeData.measCnt[BN0055_idx] %
        → (timeData.measPeriod[BN0055_idx]/10) ) == 0)
        timeData.measTodo[BN0055_idx] = true;

    /* Start a measure set each GNSS period */
    if( ( timeData.measCnt[GNSS_idx] %
        → (timeData.measPeriod[GNSS_idx]/10) ) == 0)
        timeData.measTodo[GNSS_idx] = true;
    /* Manage LED if enabled */
    if((timeData.ledCnt >= LED_PERIOD) && (appData.ledState ==
        → true))
        LED_B0ff();
}
```

Dans le code 6, nous pouvons voir le code de la routine d'interruption du timer 2. Lors de cette interruption, les différents compteurs de mesures sont incrémentés et différentes conditions vont tester si la période de mesure est atteinte. Une gestion du bouton a également lieu, notamment une mesure du temps d'appui, et une limitation des effets de rebonds mécaniques sur les signaux électriques observés par le MCU par un échantillonnage diminué. Enfin, la LED de vie est gérée, notamment son temps allumé.

5.2.2 USARTs

Deux bus de communication UART ont dû être configurés dans le cadre de ce projet ; ceux-ci ont des utilités différentes.

Configuration des USARTs

ID du BUS	Utilité	Baudrate	Interruption	Trame	Parité
UART ID1	Réceptions commandes USB	9600	Priorité 1	8bits + 1stop	Non
UART ID2	Communication avec le GNSS	9600	Non	8bits + 1stop	Non

TABLE 12

L'USART ID1, mentionné dans le tableau 12, est configuré avec une interruption. Celle-ci a été mise en place afin d'implémenter un mécanisme de FIFO associé à un tampon de réception. Ceci facilite le traitement des données et évite leur perte lors de réceptions consécutives.

5.2.3 Carte SD

Harmony permet de configurer directement un périphérique carte-SD avec le bus SPI.

(a) Configuration Carte SD.

(b) Configuration SPI.

FIGURE 36 – Harmony, driver carte-SD.

Source: Auteur

Sur la figure 36a, on peut observer la configuration de la carte SD. La broche de *Chip Select* est attribuée à **RA10**, qui correspond à **CS_SD** sur le schéma présenté à la figure 16a. La connexion est établie avec une fréquence de **12MHz**.

Concernant la figure 36b, les spécificités du bus SPI y sont configurées, incluant le mode de l'horloge et la phase de l'échantillonnage. Ces configurations ont été adaptées en adéquation avec les protocoles des cartes SD⁵.

5. SD Memory Card Interface Using SPI.

5.3 Application principale

Lors de cette section nous allons décrire le fonctionnement principale du système sous la forme d'un diagramme d'état sur la figure 37.

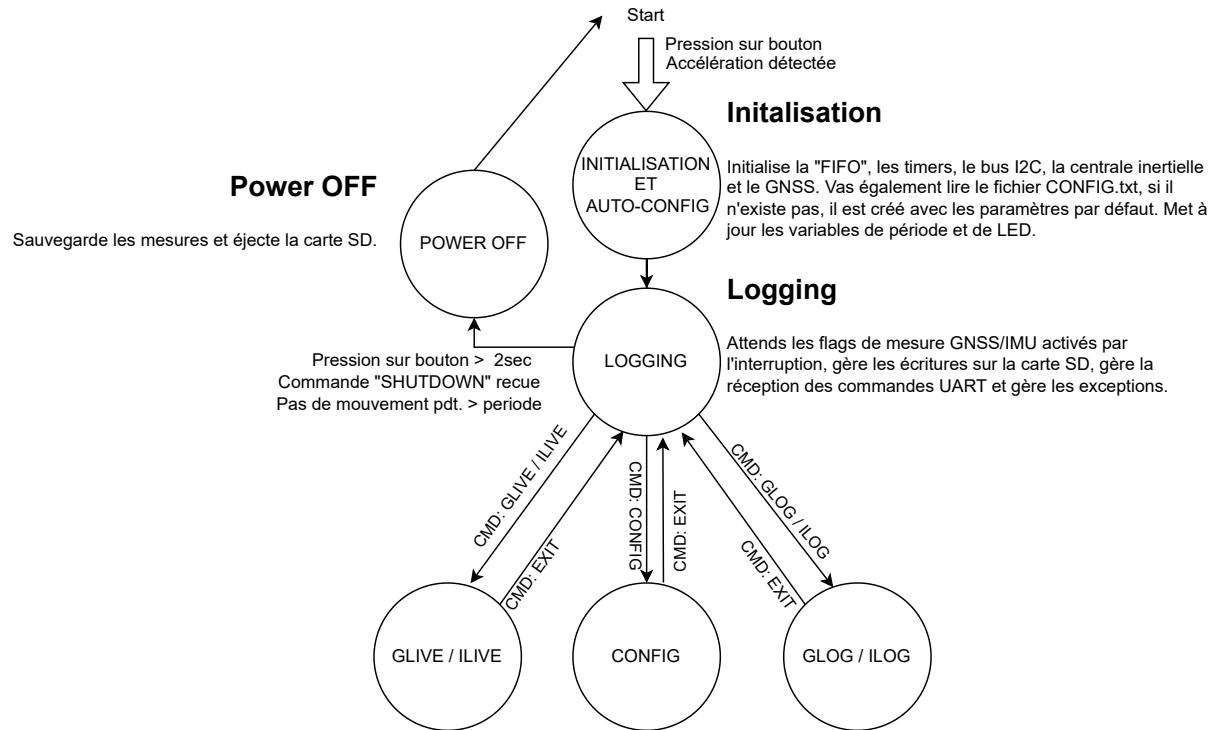


FIGURE 37 – Diagramme d'état principal.

Source: Auteur

5.3.1 Commandes USB-UART

Les commandes permettent d'interagir avec le système et de manipuler les différentes données de la boîte noire.

Liste des commandes

GLIVE	-LVG	: Permet d'afficher les données du GNSS en live sur le terminal sans les enregistrer.
ILIVE	-LVI	: Lance l'envoi des données live de l' IMU sans les enregistrer.
GLOG	-GL	: Envoie les données de localisation de la carte-SD au terminal.
ILOG	-IL	: Envoie les données inertie de la carte-SD au terminal.
GCLR	-GC	: Supprime les logs du GNSS .
ICLR	-IC	: Supprime les logs de l' IMU .
EXIT	X	: Quitte le mode en cours.
SHUTDOWN	-OFF	: Sauvegarde et éteint le système.
CONFIG	-CFG	: Démarrer le mode de configuration du système.
>	INTG :XXXX	: Dans mode config. permet de régler la période de mesure du GNSS .
>	INTI :XXXX	: Dans mode config. permet de régler la période de mesure de l' IMU .
>	LEDV :X	: Dans mode config. permet d'activer ou non la LED de vie.
>	TOFF :XX	: Dans mode config. permet de fixer le temps d'inactivité (éteint le système).

TABLE 13 – Commandes disponibles

Les commandes de la table 13 peuvent également être envoyées en caractères minuscules.

Sur la figure 38, la séquence de la boucle principale de l'application est représentée. Celle-ci interagit avec les différents périphériques en utilisant diverses bibliothèques C.

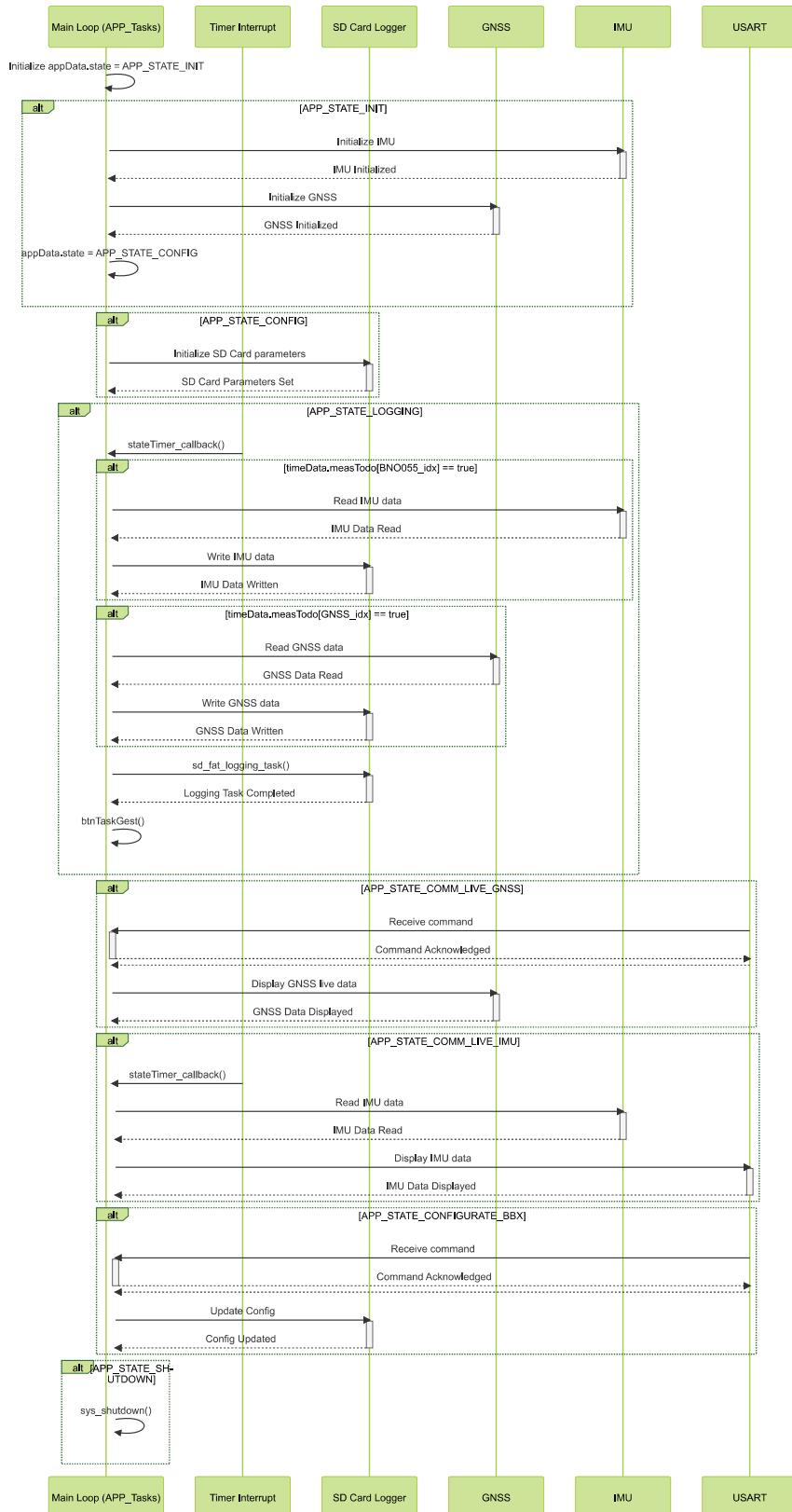


FIGURE 38 – Diagramme de séquence principal.

Source: Auteur

5.3.2 État de logging du système

L'état de logging est représenté par un flowchart sur la figure 39.

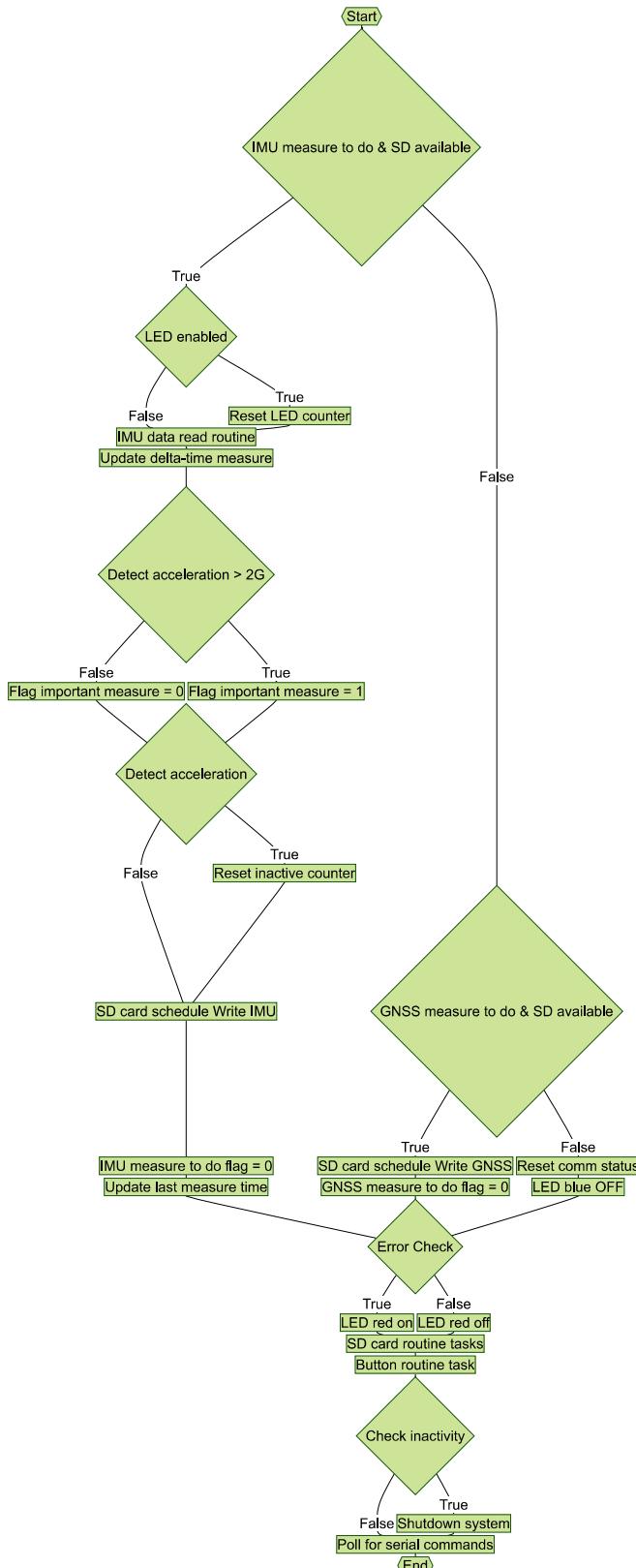


FIGURE 39 – Flowchart de l'état de logging.

Source: Auteur

5.4 Carte SD

Le code de gestion de fichier de la carte SD est divisé en deux machines d'états. La première est dédiée à l'initialisation, à la lecture et à la modification du fichier de configuration. La seconde s'occupe du logging et de l'enregistrement des mesures dans des fichiers distincts. Ces machines d'états sont non-bloquantes et doivent être appelées périodiquement pour permettre une routine de gestion de la carte SD.

5.4.1 Carte SD : initialisation et configuration

La machine d'état initiale est appelée lorsqu'elle se trouve dans l'état **APP_STATE_CONFIG**, comme illustré sur la figure 38. Elle continuera d'être appelée jusqu'à ce qu'elle atteigne l'état **IDLE**. Notons également que cette machine d'état est sollicitée en mode configuration (*init = false*) durant l'état **APP_STATE_CONFIGURATE_BBX**, visible sur la même figure 38. Ce processus est représenté sous forme de diagramme d'état à la figure 40

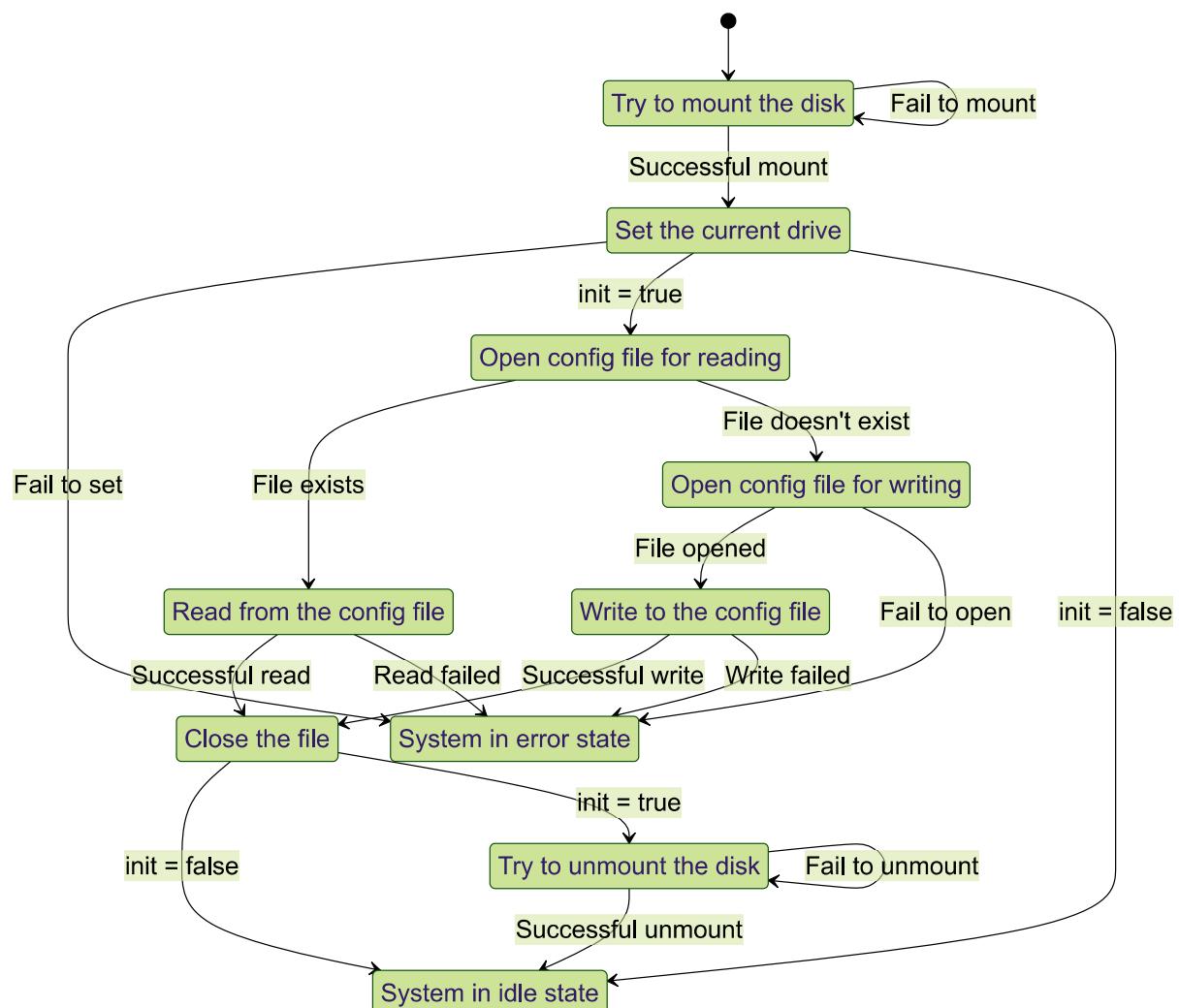


FIGURE 40 – Application carte SD initialisation/config.

Source: Auteur

5.4.2 Carte SD : écriture des données

Cette machine d'état est systématiquement appelée lorsque le système se trouve dans l'état **APP_STATE_LOGGING**, comme illustré sur la figure 38. Ceci est réalisé via la fonction `sd_fat_logging_task()`. Pour programmer des écritures et interagir avec cette machine d'état, deux fonctions d'interface sont utilisées : `sd_IMU_scheduleWrite()` qui permet de planifier l'écriture de données de l'**IMU**, et `sd_GNSS_scheduleWrite()` pour l'écriture des données du **GNSS**.

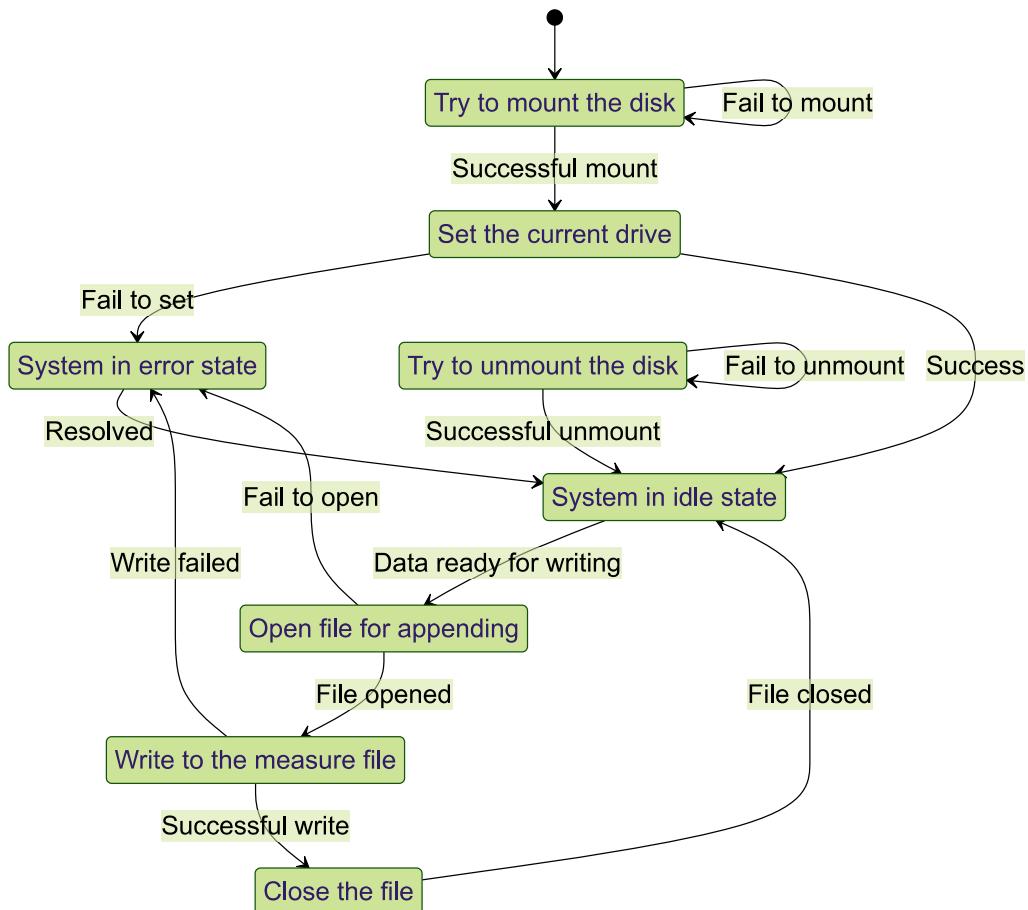


FIGURE 41 – Machine d'état logging.

Source: Auteur

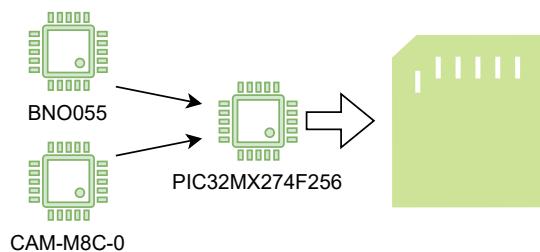


FIGURE 42 – Illustration des interactions.

Source: Auteur

5.5 Centrale inertie

Lors de cette section, nous allons décrire les différents aspects de la mise en place de la centrale inertie **BNO055**.

Interfaçage de l'IMU Bosch propose une librairie⁶ d'APIs pour communiquer avec la centrale inertie. Cette librairie est accompagnée d'un fichier *support* qui sert à lier les fonctions de protocoles aux fonctions bas niveau du bus de communication I2C.

5.5.1 Initialisation

Procédure d'initialisation

- * 1 Réinitialisation hardware par la pin RESET pendant 100ms.
- * 2 Désactivation de l'interruption⁷ de l'**IMU**.
- * 3 Attribution des fonctions bas niveau dans les pointeurs de la librairie.
- * 4 Obtention des informations du BNO055 (IDs, Version...).
- * 5 Attribution du mode d'alimentation "Normal".
- * 6 Choix du mode d'opération "AMG" (pour lecture simple Accel. magnitude et gyro.).
- * 7 Lecture initiale à vide, d'initialisation (Accel. magnitude gyroscope.).
- * 6 Choix du mode d'opération "NDOF" (pour lecture des données de fusions complexes.).
- * 7 Lecture initiale à vide, d'initialisation (Euler, Quaternion, gravité et accel. linéaire).
- * 8 Programmation de l'interruption du BNO055 pour la détection d'une certaine accélération.

5.5.2 Lecture des données

La phase cruciale concernait l'initialisation du BNO055. La lecture des diverses valeurs est assurée par une fonction spécifiquement créée, nommée `bno055_read_routine()`. Cette fonction récupère les données suivantes : Quaternion WXYZ, magnitude XYZ, gyroscope XYZ, angles d'Euler HPR, accélération linéaire XYZ, et gravité XYZ. Ces données sont stockées dans une structure locale au fichier *app.c* et possède la structure suivante :

```
typedef struct {
    s32 comres;
    bool flagMeasReady;
    uint8_t flagImportantMeas;
    struct bno055_gravity_double_t gravity;
    struct bno055_linear_accel_double_t linear_accel;
    struct bno055_euler_double_t euler;
    struct bno055_gyro_double_t gyro;
    struct bno055_mag_double_t mag;
    struct bno055_quaternion_t quaternion;
    unsigned long time;
    unsigned long l_time;
    uint16_t d_time;
}s_bno055_data;
```

6. https://github.com/BoschSensortec/BNO055_driver

7. Reset toutes les sources d'interruptions de l'**IMU** dans le registre 0x3F bit 6.

5.5.3 Système de détection de mouvements

La centrale inertuelle *BNO055* permet de configurer une interruption pour un certain événement et une certaine durée. Dans cette section, nous décrirons la configuration de cette interruption pour l'enclenchement du système ainsi que la possibilité d'activer/désactiver cette option.

Code Source 7 – Configuration de l'interruption

```
/* BNO055 motion interrupt mode */
1) bno055_set_accel_any_motion_no_motion_axis_enable
   ← (BNO055_ACCEL_ANY_MOTION_NO_MOTION_X_AXIS, 1);
2) bno055_set_accel_any_motion_no_motion_axis_enable
   ← (BNO055_ACCEL_ANY_MOTION_NO_MOTION_Y_AXIS, 1);
3) bno055_set_accel_any_motion_no_motion_axis_enable
   ← (BNO055_ACCEL_ANY_MOTION_NO_MOTION_Z_AXIS, 1);

4) bno055_set_accel_any_motion_thres(10);
5) bno055_set_accel_any_motion_durn(20);
6) bno055_set_intr_accel_any_motion(1);
7) bno055_set_intr_mask_accel_any_motion(1);
```

Les valeurs de configuration peuvent être définies en tant que constantes pour être plus facilement configurable lors de mise-à-jour firmware.

Description des lignes du code [7](#)

- * 1 Activation de l'axe X pour le mode *anymotion*.
- * 2 Activation de l'axe Y pour le mode *anymotion*.
- * 3 Activation de l'axe Z pour le mode *anymotion*.
- * 4 Attribution de la valeur de déclenchement à $78.3mg = 0.768m/s^2$.
- * 5 Configuration de la durée à 20ms.
- * 6 Active l'interruption, accélération.
- * 7 Masque l'interruption.

5.5.3.1 Gestion de l'interruption L'interruption est réinitialisée au démarrage du système ainsi que lors de sa mise en veille.

5.5.3.2 Mode low power lorsque le système se met en veille, la *BNO055* est configurée pour fonctionner uniquement avec son accéléromètre et son mode de puissance passe en *low power*.

Code Source 8 – Configuration de l'interruption

```
/* Set acceleration only operation, power saving */
bno055_set_operation_mode(BNO055_OPERATION_MODE_ACONLY);
/* set the power mode to LOW POWER */
bno055_set_power_mode(BNO055_POWER_MODE_LOWPOWER);
/* Reset interrupt pin */
bno055_set_intr_RST(1);
```

5.5.3.3 Configuration enclenchement automatique Le système propose deux modes d'opération : le premier permet un enclenchement automatique lors de la détection d'un mouvement, tandis que le second active le système suite à une impulsion sur le bouton. Il est à noter que pour activer le mode automatique, une première impulsion sur le bouton est nécessaire et le système doit avoir transité au moins une fois par l'état "shutdown" du code. Dans les deux modes, le système peut entrer en veille si aucun mouvement n'est détecté sur une durée configurée.

Lorsque le système est en mode d'enclenchement automatique, le circuit reste partiellement actif, ce qui engendre une consommation d'énergie détaillée ultérieurement dans la section "Validation du design". Même si la batterie est en mesure de supporter cette consommation, il reste possible de désactiver ce mode pour activer le système uniquement par le biais du bouton.

Configuration La centrale inertuelle peut être alimentée soit de manière indépendante, soit avec le reste du système, influençant ainsi les modes décrits précédemment. Pour configurer ces modes, deux options sont disponibles :

W6 connecté	IMU alimentation autonome.	Mode accélération auto-enclenchement.
W7 connecté	Alimentation dépendant du système.	Mode enclenchement bouton uniquement.
W6 & W7	Connexion à éviter !	

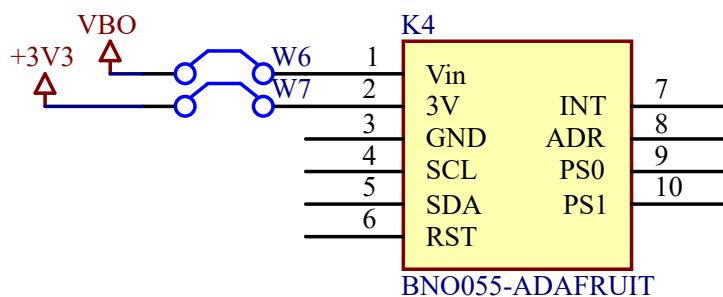


FIGURE 43 – Connexions des jumber du BNO055.

Source: Auteur

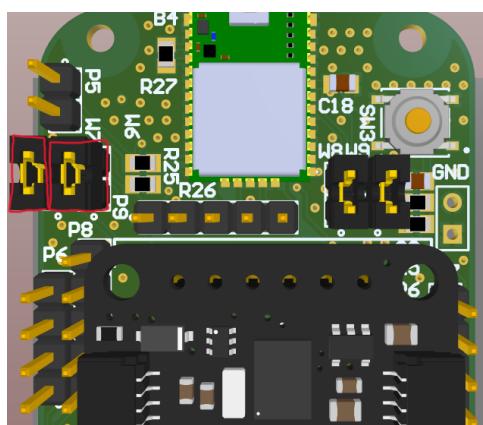


FIGURE 44 – Emplacement des jumpers sur le **PCB**.

Source: Auteur

Sur la figure 44, les jumpers sont entourés en **rouge**. Le jumper **W6** se situe à droite tandis que **W7** se situe à gauche.

5.6 Format des données

Dans cette section, nous décrirons les formats des données enregistrées sur la carte SD.

5.6.1 Données de l'IMU

Les données de la centrale inertielles sont enregistrer dans un fichier CSV et suivent un format fixe :

Flag accélération;Delta time;Gravity X;Y;Z;Gyroscope X;Y;Z;Magnitude X;Y;ZAccél. linéaire X;Y;Z;Angle Euler H;P;R,Quaternion W;X;Y;Z

```
0;500;-3.5900;-3.9900;8.2000;17.3750;141.6875;20.0625;-0.8750;28.3750;-17.8750;0.9700;-0.0300;-0.4700;19.4375;25.9375;-21.4375;15475;-2902;3674;-2655;
0;500;-9.1700;-3.4500;-0.2200;6.0625;184.1250;-153.9375;21.0000;26.7500;-9.3750;0.0200;-1.0300;-0.3300;35.1875;93.6875;-69.3125;11140;-521;11538;-3308;
0;500;-0.7300;-2.2000;-9.5200;-26.1875;26.6250;1.5625;12.2500;26.7500;19.5000;0.3500;0.5500;0.0000;21.0625;166.9375;4.3125;272.2977;15992;-1936;
0;500;-9.1600;-2.9400;1.8500;21.7500;-80.5000;-52.6875;22.0000;24.1875;-14.5625;1.6800;-0.9300;-0.2700;45.2500;57.7500;-69.1875;11613;936;10478;-4787;
0;500;2.4100;-2.6900;9.1100;-32.1250;-58.6875;-29.3750;-14.3750;28.2500;-19.5625;1.2500;0.6500;-1.1100;24.1250;16.1875;13.6250;15752;-2619;-1450;-3367;
0;500;9.6700;-0.1800;-1.5900;18.1875;-87.3125;-61.8750;-26.8750;26.1875;6.5000;-0.5400;0.2900;-0.3300;10.6250;173.3125;80.5000;10648;-1472;-12328;-958;
0;500;5.2100;-2.8900;7.7800;-2.2500;129.8125;-21.5625;-24.0625;25.2500;-7.7500;0.5000;0.5100;-0.8700;26.8125;20.3750;32.0625;15093;-3554;-3876;-3605;
```

FIGURE 45 – Données CSV.

Source: Auteur

Exemple d'un set de données CSV Ayant précédemment développé un script Python (disponible en annexe), nous pouvons visualiser simplement et rapidement les données de logs inertielles sous forme de courbes, comme sur la figure 46.

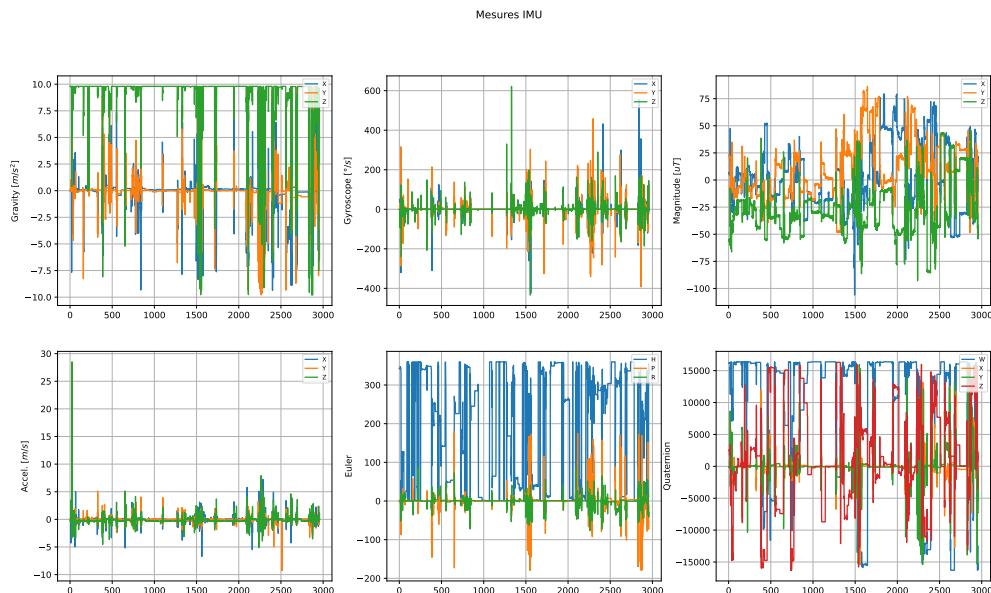


FIGURE 46 – Visualisation des données inertielles.

Source: Auteur

Il est important de souligner que, lors de la prise des mesures présentées à la figure 46, le module était en déplacement. Nous constatons que 3000 mesures ont été effectuées et sur la figure 45 nous pouvons observer que l'intervalle entre les mesures était de 500ms, ce qui indique un enregistrement d'une durée de 25 minutes. Le prototype en l'état actuel ne permet pas de localiser suffisamment de sa

5.6.2 Données du GNSS

Les données du **GNSS** ont été sauvegardées en format NMEA dans un fichier texte sans modifications. Cela a été fait afin de permettre des analyses plus approfondies lors du développement en important les données sur ordinateur, plutôt que de les analyser directement sur le microcontrôleur. Toutefois, il serait tout à fait possible d'analyser les messages NMEA sur le microcontrôleur et d'enregistrer uniquement les données pertinentes sur la carte SD. Une telle approche permettrait d'économiser de l'espace de stockage sur la carte SD. Néanmoins, comme nous l'avons observé dans la section [2.4.4](#), une telle économie ne serait pas nécessaire étant donné l'importante marge de capacité.

Exemple de logs NMEA Les données de la figures [47](#) contiennent différents messages NMEA (colorés en bleus) loggés lors d'un essai.

```
$GNGSA,A,1,,,,,,,,,,99.99,99.99,99.99*2E
$GPGSV,1,1,00*79
$GLGSV,1,1,00*65
$GNGLL,,,,,,V,N*7A
$GNRMC,,V,,,,,,,N*4D
$GNVTG,,,,,,N*2E
$GNGGA,,,,,0,00,99.99,,,,,*56
```

FIGURE 47 – Données NMEA loggées.

Source: Auteur

Sur la figure [47](#), nous constatons qu'au moment de la prise de ces logs, aucun satellite n'a été détecté (le message GSV indique **00**) et que les valeurs reçues n'étaient pas valides (comme le montrent les **V** colorés en rouge). Il est probable que ce problème de réception soit dû à un plan de masse autour de l'antenne insuffisamment étendu, empêchant un blindage adéquat pour la réception du signal. Pour remédier à ce problème, des tests avec des antennes externes de différents types sont actuellement en cours. Toutefois, dans son état actuel, le prototype est capable de recevoir des données d'heure et de date, et peut se connecter à jusqu'à 4 satellites (maximum observé lors de tests extérieurs. Le prototype pourrait être sujet à de meilleures conditions dans des zones à signaux forts).

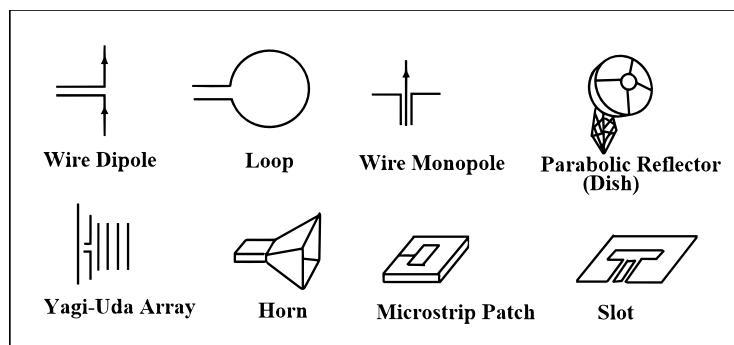


FIGURE 48 – Types d'antennes.

Source: <https://www.fiberglassantennas.com/antenna-types.html>

5.7 Fonctions des fichiers

Dans cette section, nous examinerons les différentes fonctions et variables mises à disposition et utilisées par les différents fichiers des bibliothèques C créées. Pour illustrer cela de manière pratique, nous utiliserons une représentation sous forme de classes.

5.7.1 Centrale inertie

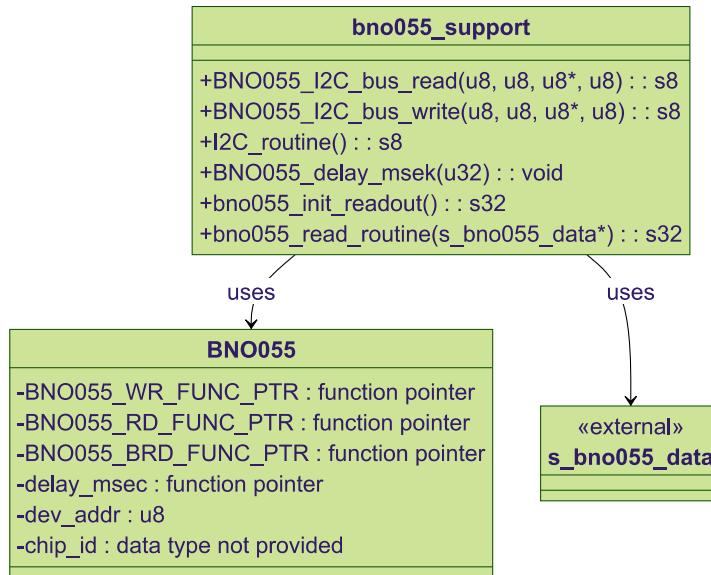


FIGURE 49 – Classe de la librairie de l’IMU.

Source: Auteur

5.7.2 Carte SD

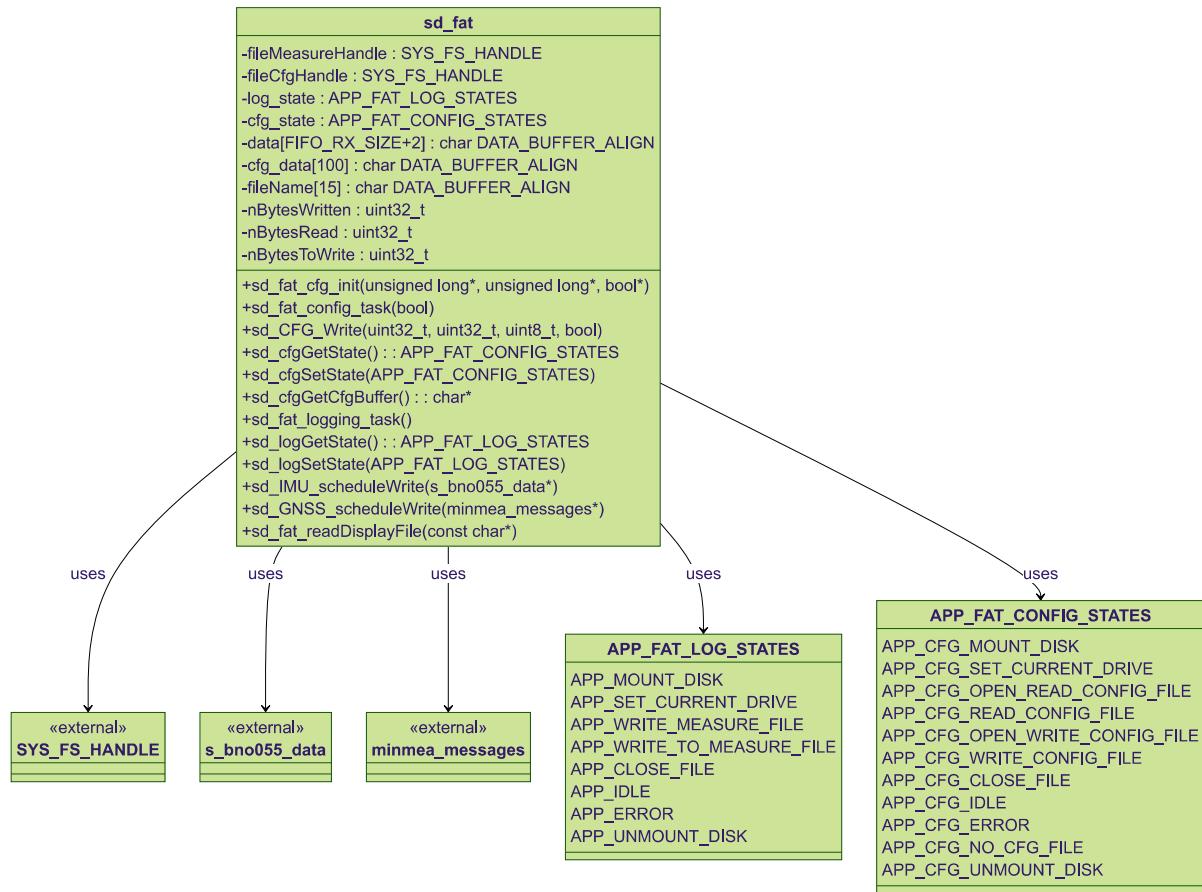


FIGURE 50 – Classe de la librairie de la carte SD.

Source: Auteur

5.7.3 Communication série avec FTDI

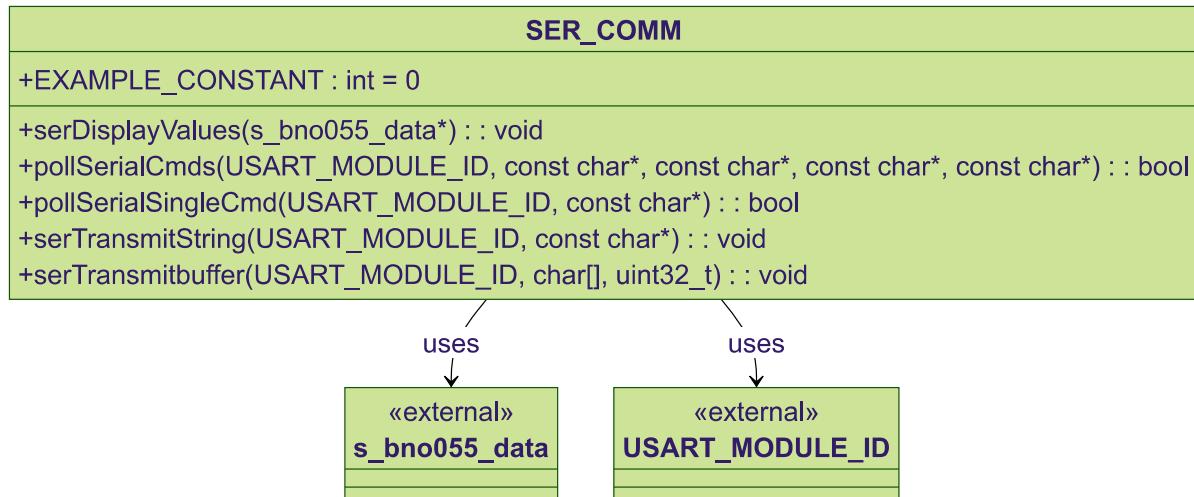


FIGURE 51 – Classe de la librairie communication USART.

Source: Auteur

6 Développement application interface

Étant donné que la boîte noire développée offre plusieurs commandes et permet une communication USB-C avec divers appareils, la création d'une interface graphique est particulièrement adaptée. Cette application facilite l'accès aux données en temps réel et aux logs, et permet également de configurer aisément la boîte noire. Dans cette section, nous décrirons la démarche suivie pour créer cette application.

6.1 Choix de l'environnement

Pour développer cette application, le langage de programmation Python a été choisi. Ce langage est particulièrement adapté pour concevoir des applications graphiques grâce à la bibliothèque Tkinter. Il s'agit d'une bibliothèque graphique libre pour Python qui facilite la création d'applications compatibles avec les systèmes d'exploitation les plus populaires auprès du grand public.

Tkinter permet d'utiliser des objets/widgets pour facilement mettre en place une interface graphique de façon très intuitive.

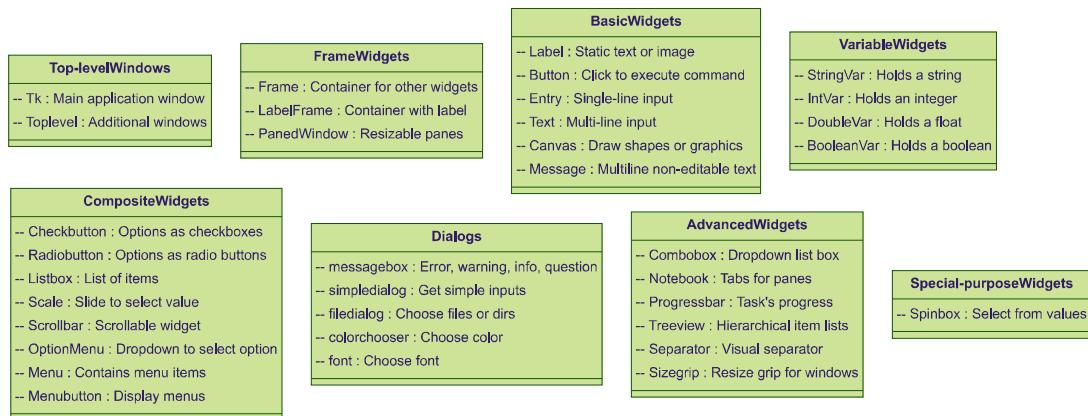


FIGURE 52 – Widgets Tkinter.

Source: Auteur

Le code python complet de l'application est disponible en annexe de ce rapport.

6.2 Présentation de l'interface

Code Source 9 – Extrait de code, création de la forme.

```
# Setup main window
gRoot = Tk()
gRoot.config(bg="white")
gRoot.geometry("1080x640")
gRoot.title("Black Box Connect")
# Style configuration
sty = ThemedStyle(gRoot)
sty.set_theme('radiance')
```

Comme illustré sur la figure 53, différentes fonctions de l'application sont visibles. Le port de communication série ainsi que le baud rate sont configurables. À la fermeture de l'application, le port se ferme automatiquement. L'application permet d'afficher les données des périphériques en temps réel *LIVE* ou sous forme de *LOGS*. Elle offre également la possibilité de supprimer les fichiers de logs stockés sur la carte SD et de sauvegarder les données reçues dans divers formats tels que "CSV, TXT...", comme le démontre la figure 55. Enfin, la configuration de la boîte noire est accessible via le menu présenté sur la figure 54.

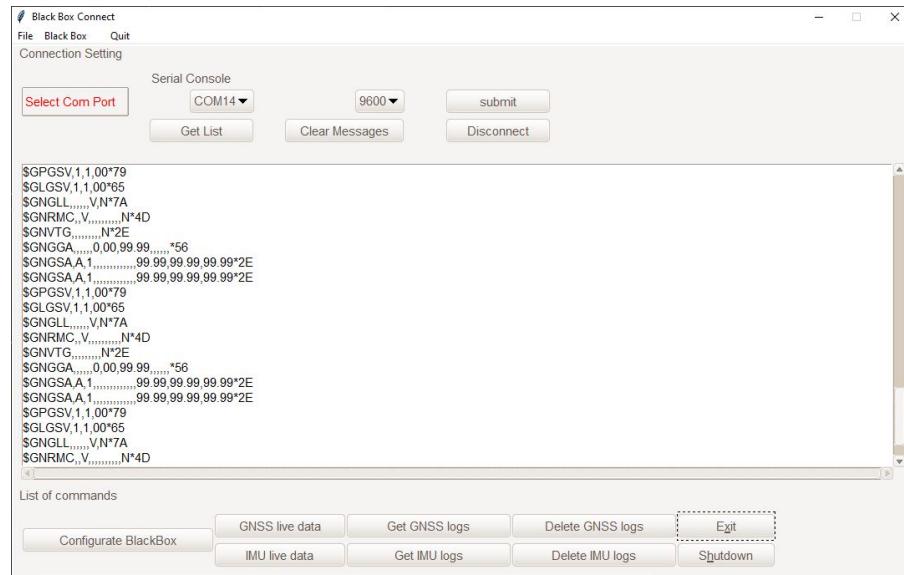


FIGURE 53 – Application, logs GNSS obtenus.

Source: Auteur



FIGURE 54 – Menu configuration.

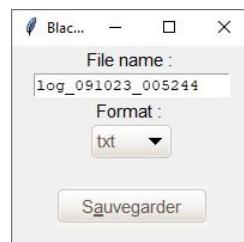


FIGURE 55 – Menu de sauvegarde.

Source: Auteur

7 Validation du design

Dans cette section, nous décrirons la procédure de vérification des caractéristiques du projet ainsi que sa validation.

7.1 Liste de matériel

- **P1** : Oscilloscope Tektronix RTB2004 ES.SLO2.05.01.11
- **P2** : Multimètre GwInsteek GDM-396 ES.SLO2.00.00.94
- Carte Mini-Boite-Noire 1924B

7.2 Consommations

Dans cette section, nous mesurerons les différentes consommations du système. Cette étape est importante pour caractériser le système et déterminer son autonomie.

7.2.1 Méthode de mesure

L'objectif est de basculer entre les différents modes (veille, logging...) de la carte et d'en mesurer la consommation. À cet effet, un ampèremètre a été placé en série avec la batterie, comme illustré à la figure 56.

7.2.2 Schéma de mesure

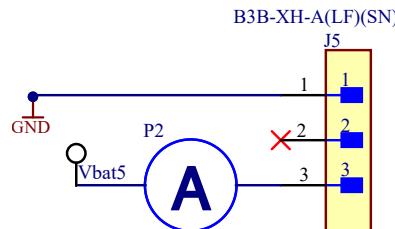


FIGURE 56 – Schéma de mesure, courant
Source: Auteur

7.2.3 Mesures

Index	Etat du système	Condition	Courant [mA]	Symbol
[1]	Eteint.	Mode auto-enclenchement OFF.	0.02	I_{off}
[2]	Veille.	Est passé par l'état SHUTDOWN.	4.4	I_{sleep}
[3]	Veille brutale. ⁸	Pas passé par l'état SHUTDOWN.	9.56	I_{ws}
[4]	Initialisation.	-	90	I_{init}
[5]	Logging.	-	100	I_{log}
[6]	Shutdown.	-	83	I_{sh}
[7]	Communication.	USB branché.	0	I_{usb}

TABLE 14 – Mesure des consommations
Source: Auteur

8. Lorsque le système a été éteint de façon non-contrôlée (Batterie débranchée).

Nous pouvons par les mesure de la table 14 déduire les éléments suivants :

Où :

Capacité de la batterie $C = 1600 \text{ mAh}$

- Temps de logging (Table 14-[5]) : $T_l = \frac{C}{I_{log}} = 16h.$
- Temps épuisement batterie en veille (Table 14-[2]) : $T_l = \frac{C}{I_{sleep}} = 364h = 15J.$
- Temps épuisement en veille brutale (Table 14-[3]) : $T_l = \frac{C}{I_{ws}} = 167h = 7J.$

Par conséquent, les caractéristiques d'autonomie de la batterie sont suffisantes pour notre application.

7.2.3.1 Proposition de stratégies d'économie d'énergie Afin de diminuer la consommation lors du mode veille il existe différentes possibilités :

Débraser la LED de la carte BNO0555 d'adafruit

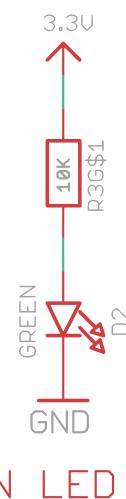


FIGURE 57 – LED de vie de la carte d'adafruit (2.2V).

Source: Schéma de la [carte BNO055 d'adafruit](#)

Comme le montre la figure 57, une LED reste constamment allumée lors de la mise sous tension de la carte de l'**IMU**. Elle présente une différence de potentiel de 2.2V à ses bornes, entraînant une consommation constante de 0.11mA. Bien que cette consommation soit faible, elle est inutile. Dans le cadre de ce prototype, la LED n'a pas été débrasée, mais un adhésif a été collé dessus afin de limiter la pollution lumineuse sur la LED d'état qui passe par un guide-lumière.

Permettre de changer de mode plus facilement Permettre à l'utilisateur de facilement basculer entre le mode "auto-enclenchement" et "enclenchement manuel" permettrait d'éviter des consommations inutiles.

7.3 Bus de communications

Avec le code implémenté dans le microcontrôleur, les différents périphériques fonctionnent correctement et la communication avec les différents bus est opérationnelle. C'est pourquoi, dans cette section, l'objectif principal est de mesurer la qualité et l'intégrité des signaux plutôt que d'analyser les différents protocoles.

7.3.1 Communication I2C

La communication I2C s'effectue entre la centrale inertuelle BNO055 et le microcontrôleur. Par ce bus, les configurations du registre de l'**IMU** ainsi que les données inertielles mesurées, en fonction de ces configurations, sont transmises. Comme nous avons pu le voir précédemment, cette communication est effective et les données sont cohérentes. Dans ce contexte, nous examinerons si l'intervalle entre les mesures est respecté, la durée de ces mesures, ainsi que la qualité des signaux I2C.

7.3.1.1 Méthode de mesure La boîte noire a été configurée pour transmettre et enregistrer des données inertielles à intervalles de 2 secondes. De plus, les mesures ont été prises pendant la transmission des données inertielles, et non lors de la configuration. Enfin, les trames ont été décodées automatiquement par l'oscilloscope grâce à sa fonction "Protocol".

7.3.1.2 Schéma de mesure Le schéma de mesure est présenté sur la figure 58.

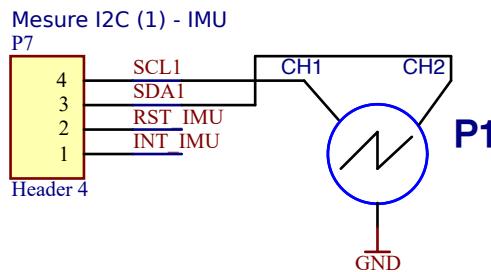


FIGURE 58 – Schéma de mesure, mesures I2C.

Source: Auteur

7.3.1.3 Mesures Comme le montre la figure 59, il y a un délai de 1.98 secondes entre l'envoi de deux sets de données de la centrale inertuelle.

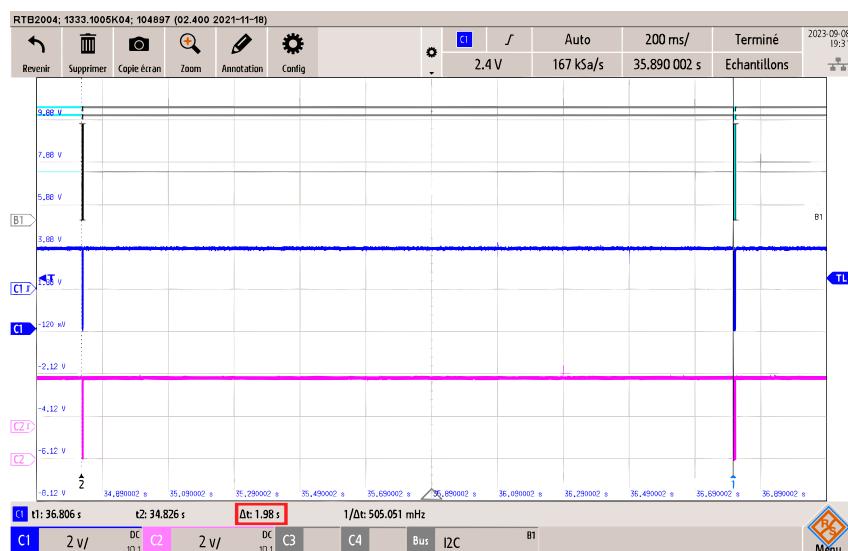


FIGURE 59 – Intervalle entre deux mesures, config. 2 secondes.

Source: Auteur

Malgré la légère différence sur la figure 59 de $20ms$, probablement due à une imprécision de mesure, nous pouvons conclure que l'intervalle entre les mesures respecte bien la configuration établie, même après modifications de celle-ci.

Sur la figure 60, nous observons la durée d'une trame de mesures qui englobe toutes les données mentionnées dans la section 5.6.1. La durée de transmission d'un set de mesure est de $2.274ms$. Compte tenu que le bus est configuré en mode "fast" à $400kbit/s$, cela suggère qu'environ ~ 910 bits ont été transmis, ce qui explique la densité des données rendant la figure peu lisible.

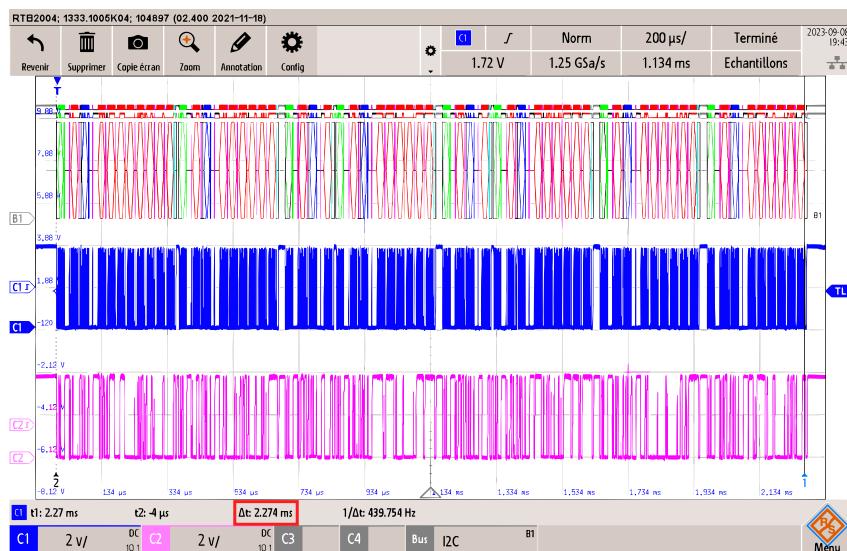


FIGURE 60 – Durée d'une mesure.

Source: Auteur

Sur la figure 61, nous observons le début d'une communication avec l'**IMU**. Le premier byte (**0x28**) correspond à l'adresse du BNO055. Ensuite, le registre **0x28** est adressé : il s'agit du registre contenant les données **LSB** de l'accélération linéaire. Les flancs des bits transmis semblent suffisamment nettes et peu parasités pour une bonne communication.

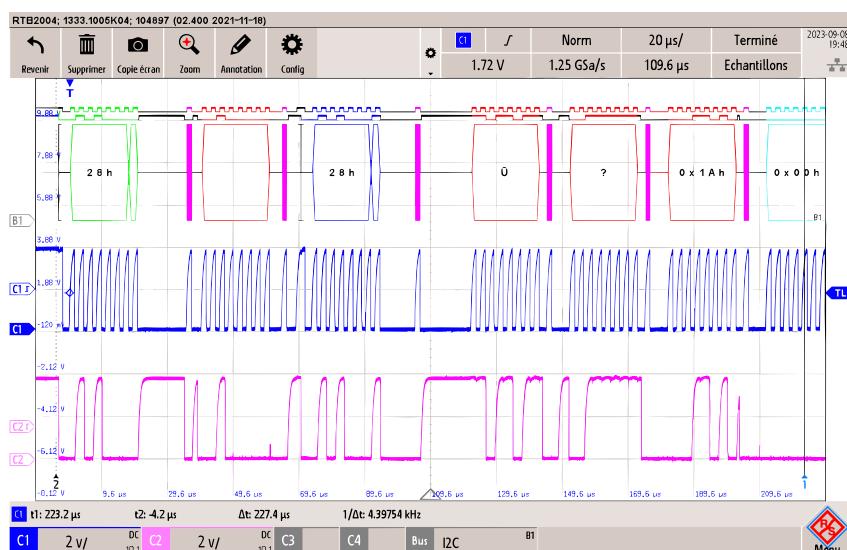


FIGURE 61 – Début de trame d'une mesure.

Source: Auteur

7.3.2 Communication UART GNSS

Dans cette section, nous analyserons les différentes caractéristiques de la communication UART entre le microcontrôleur et le module GNSS.

7.3.2.1 Méthode de mesure Pour réaliser ces mesures, le système a été activé et le GNSS a été configuré par défaut à une fréquence de **1Hz**.

7.3.2.2 Schéma de mesure Le schéma de mesure est présenté sur la figure 62, nous nous intéressants ici, qu'au message envoyés par le GNSS dans sa configuration par défaut.

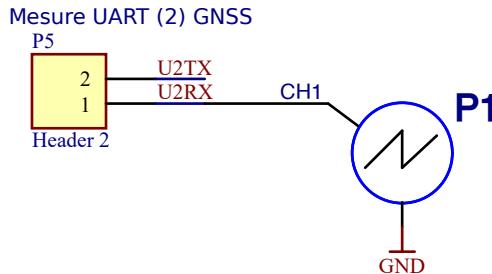


FIGURE 62 – Schéma de mesure GNSS.

Source: Auteur

7.3.2.3 Mesures Sur la figure 63, nous observons l'intervalle entre les messages GNSS. Effectivement, cet intervalle est de 1s, offrant ainsi suffisamment de temps au **MCU** pour traiter ces données, que ce soit pour les afficher sur un terminal ou les enregistrer sur la carte SD.

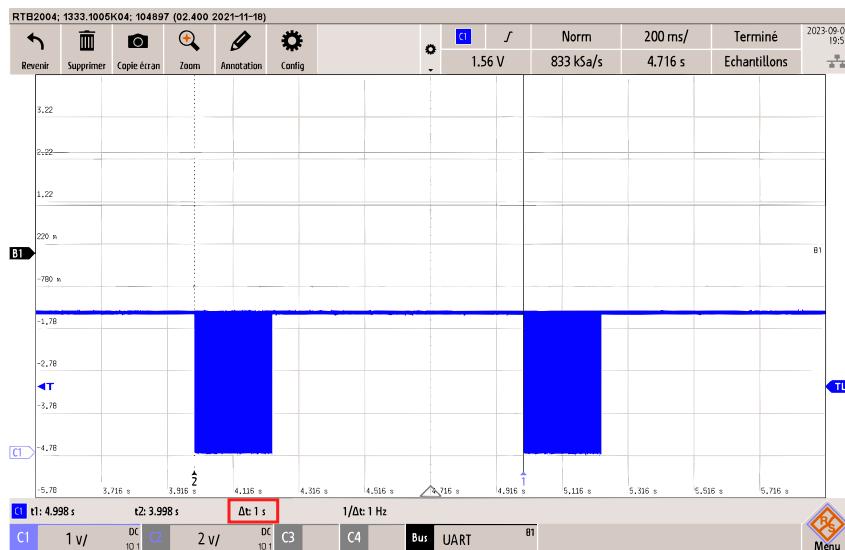
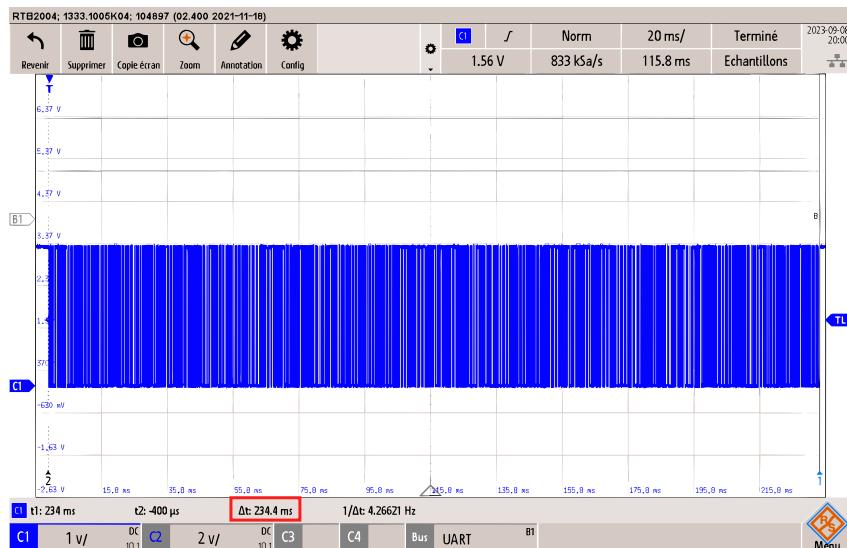


FIGURE 63 – Intervalle entre les données NMEA.

Source: Auteur

Sur la figure 64, nous constatons que la durée d'envoi des messages **NMEA** est de 234.4ms. Sachant que l'intervalle entre les transmissions est de 1s, cela laisse une fenêtre de 756ms au **MCU** pour traiter ces données. Cette période est largement suffisante pour la transmission des données via le second UART (opérant au même baud rate) ou vers la carte SD, dont la fréquence

est nettement supérieure ($10MHz$). Si nous considérons la durée des transmissions de l'**IMU**, qui est de $2.274ms$, nous estimons avec une marge qu'environ $\sim 400ms$ des $756ms$ disponibles sont utilisés pour traiter les données. Ce temps de traitement pourrait être encore diminué si la fréquence d'enregistrement du **GNSS** était baissée. Ainsi, nous pourrions envisager d'obtenir jusqu'à ~ 155 mesures d'**IMU** dans l'intervalle donné.



7.3.3 Communication UART2 FTDI

Dans cette section, nous examinerons les trames de communication sur l'UART 2, établies entre le **MCU** et le **FTDI**, qui sert de bus d'interface entre la boîte noire et un appareil externe.

7.3.3.1 Méthode de mesure Ces mesures ont été effectuées pendant une communication entre la boîte noire et un ordinateur fixe, en utilisant l'application développée dans la section [6](#).

7.3.3.2 Schéma de mesure Le schéma de mesure est présenté sur figure [66](#) :

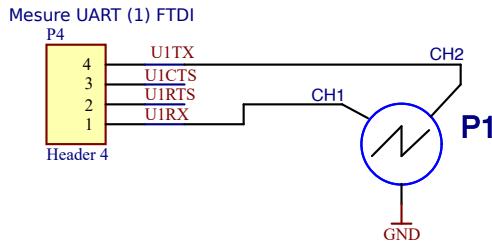


FIGURE 66 – Schéma de mesure UART2.

Source: Auteur

7.3.3.3 Mesures Sur la figure [67](#), une communication est visible. Il s'agit d'une commande de configuration envoyée par l'application pour modifier le temps entre l'écriture des mesures **GNSS** à 5 secondes. Nous pouvons distinguer l'envoi de la commande "**INTG :5000**" et, 120µs après, la réception de la réponse de la boîte noire. Cela démontre une réactivité rapide du système dans le mode **CONFIG**.

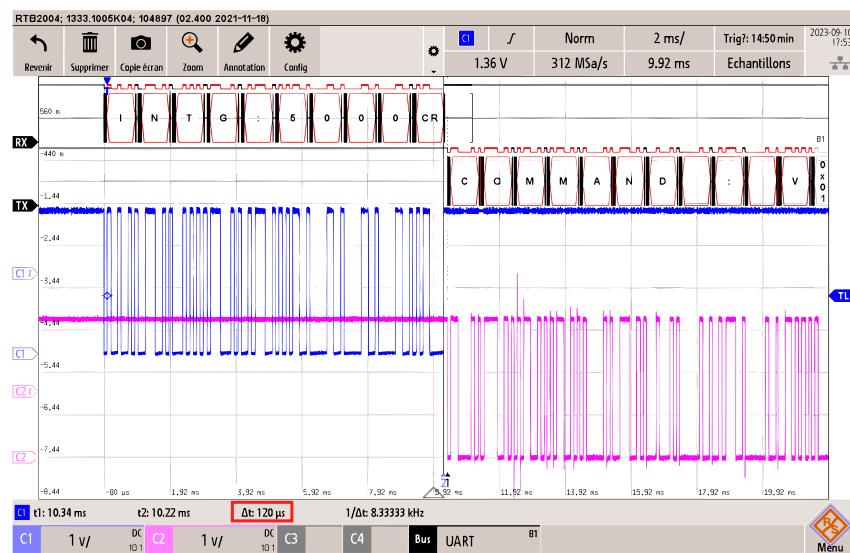


FIGURE 67 – Trames configuration.

Source: Auteur

Sur la figure 68, nous observons le temps de réaction du système suite à une commande demandant l'obtention des LOGS de l'**IMU**. L'intervalle entre cette demande et la réponse du système en mode **LOGGING** est de **2.7ms**, ce qui est plus élevé que la figure 67, sachant que le système à plus d'éléments à gérer lors de ce mode.

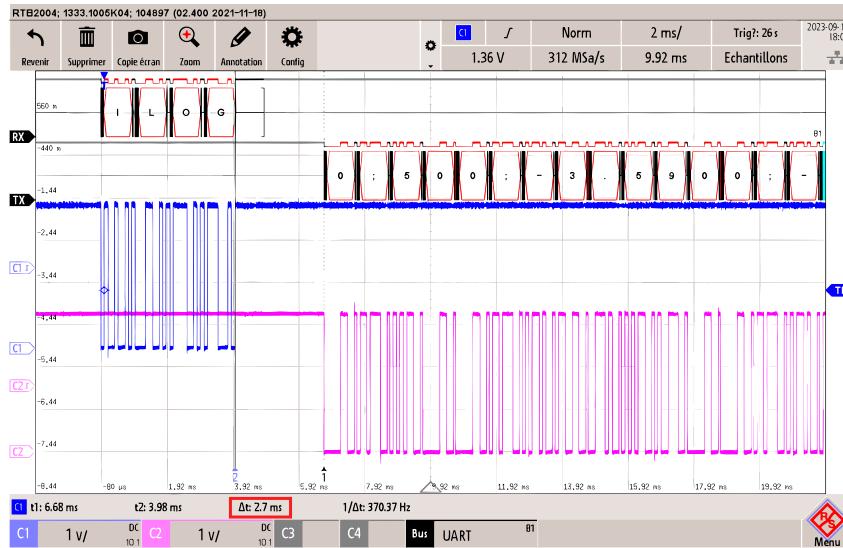


FIGURE 68 – Réaction commande durant **LOGGING**.

Source: Auteur

7.3.4 Communication SPI, carte SD

Dans cette section, nous examinerons la communication SPI entre le **MCU** et la carte SD.

7.3.4.1 Méthode de mesure Étant donné la complexité de la communication SPI pour la gestion du système de fichiers FAT, notre objectif ici n'est pas d'analyser le protocole en lui-même, mais plutôt d'observer les signaux électriques lors du logging.

7.3.4.2 Schéma de mesure Le schéma de mesure est visible sur la figure 69 :

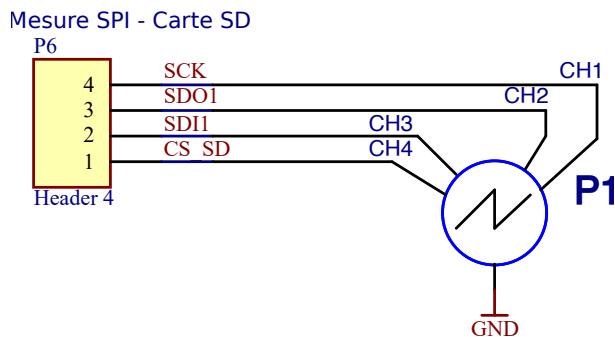


FIGURE 69 – Schéma de mesure SPI.

Source: Auteur

7.3.4.3 Mesures Sur la figure 70, nous observons que la communication SPI est dense et maintenue de manière constante pendant la phase de logging. De cette manière, le périphérique de stockage demeure associé au **MCU** et, lors des écritures, les transmissions se densifient.

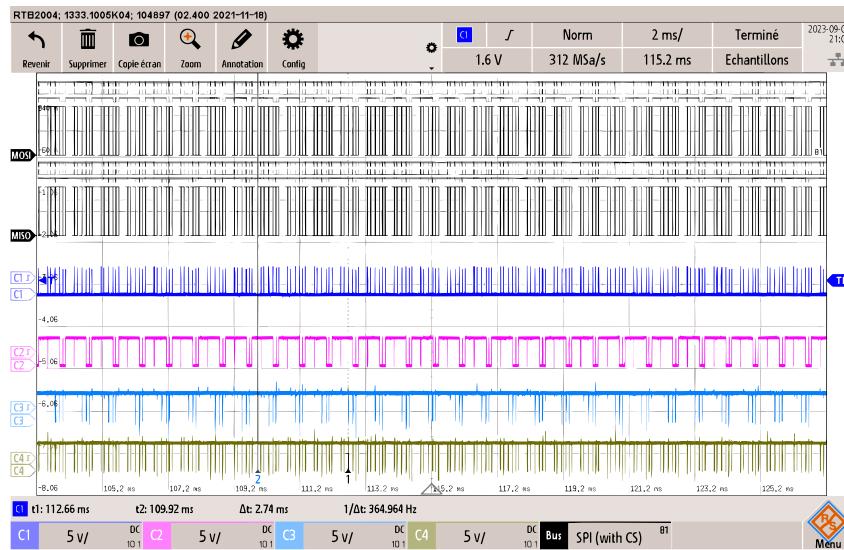


FIGURE 70 – Densité de la communication SPI.

Source: Auteur

Sur la figure 71, nous observons le signal d'horloge du bus SPI avec une échelle de temps plus petite. En mesurant le signal **SCK** sur le canal **CH1**, nous déterminons une fréquence de 12.0192MHz, ce qui correspond à la fréquence configurée dans [harmony](#).

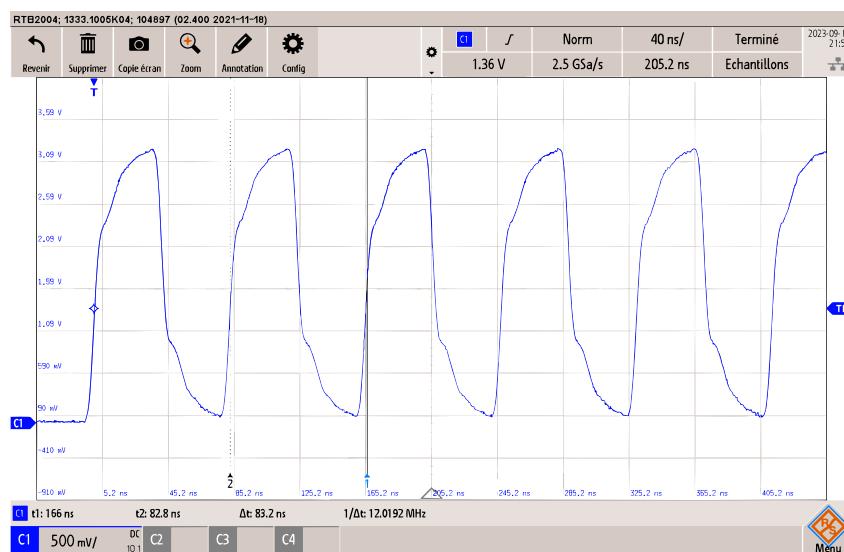


FIGURE 71 – Mesure fréquence clock SPI.

Source: Auteur

8 Caractéristiques du prototype final

La table 15 présente les caractéristiques finales du prototype basées sur le hardware actuel.

Caractéristique	Attribut	
Axes de mesures	9	[axes]
Mesures IMU	Temps, Gravité, Gyro, Magnitude, Accél. linaire, Angle de Euler, Quaternions	
Intervalles mesure GNSS (par défaut)	5000	[ms]
Intervalles mesure IMU (par défaut)	500	[ms]
Capacité carte SD	256	[MB]
Nombre de mesures possibles	$1.11 * 10^6$	
Vitesse MCU	72	[MHz]
Capacité batterie	1600	[mAh]
Autonomie logging	16	[h]
Autonomie veille	2	[semaines]
Interface	USB et LED RGB	
Communications	I2C, SPI, UART(2), USB	
Vitesse SPI	12	[MHz]
Vitesse UART 1 et 2	9600	[Bd]
Détection accélération > 2G	Oui	
Données satellites	Date et heure	
Compensation température (Mesures IMU)	Oui	
Enclenchement automatique	Oui	
Système d'économie d'énergie	Oui	
Application software développée	Oui, "BBX-Connect"	

TABLE 15 – Caractéristiques finales.

Source: Auteur

8.1 Problème rencontré, modification et amélioration

Dans l'état actuel, tel que décrit dans la section 5.6.2, le **GNSS** présente un problème de réception satellite. Pour pallier ce problème, un connecteur **MHF1** a été ajouté sur le circuit imprimé afin de permettre le raccordement d'antennes externes. Des tests sont en cours et un autre type d'antenne est en attente de livraison. Pour résoudre ce souci dans les versions antérieures, il est conseillé d'étendre le plan de masse autour du CAM-M8C-0 ou d'implémenter correctement un connecteur pour une antenne externe.

9 Conclusion

Ce rapport a présenté en détail le développement d'une mini boîte noire destinée à collecter et stocker des données de vol d'aéronefs en utilisant une centrale inertielle et un système de positionnement **GPS/GNSS**. Le projet a débuté par une phase de pré-étude visant à comprendre le fonctionnement du système et à choisir les composants et les technologies appropriés, notamment le microcontrôleur, la centrale inertielle, le **GNSS**, et la carte SD. Ensuite, le développement du schéma électronique a été abordé, détaillant la connectivité du microcontrôleur, des périphériques tels que la carte SD, la centrale inertielle, et le module USB-FTDI, ainsi que les alimentations et le système de charge de la batterie. La conception du circuit-imprimé a été décrite, mettant l'accent sur les aspects mécaniques, le placement des composants, l'assemblage, et la liste des matériaux.

La phase de développement du firmware a été exposée, couvrant les protocoles du **GNSS**, la configuration des périphériques, l'application principale gérant les commandes USB-UART et l'état de logging, ainsi que la gestion des données de la carte SD, de la centrale inertielle, et des communications série avec le **FTDI**. L'interface utilisateur de l'application a été brièvement décrite, ainsi que les critères de validation du design, incluant la consommation d'énergie, les bus de communication I2C-**IMU**, UART-**GNSS**, UART-**FTDI** et SPI pour la carte SD.

Ce projet témoigne de l'importance des boîtes noires dans l'aviation, représentant un exemple concret de l'application de technologies avancées pour améliorer la sécurité aérienne et la compréhension des incidents aéronautiques. L'objectif de stocker des données de mesures a été atteint avec succès, et ce projet ouvre des perspectives passionnantes pour le développement futur de systèmes similaires.

Je tiens à adresser mes sincères remerciements à M. J. J. Moreno pour son expertise et son aide précieuse tout au long du développement de ce projet. Mes remerciements s'étendent également à l'AMPA pour avoir mandaté ce projet passionnant et pour la confiance qu'ils m'ont accordée.

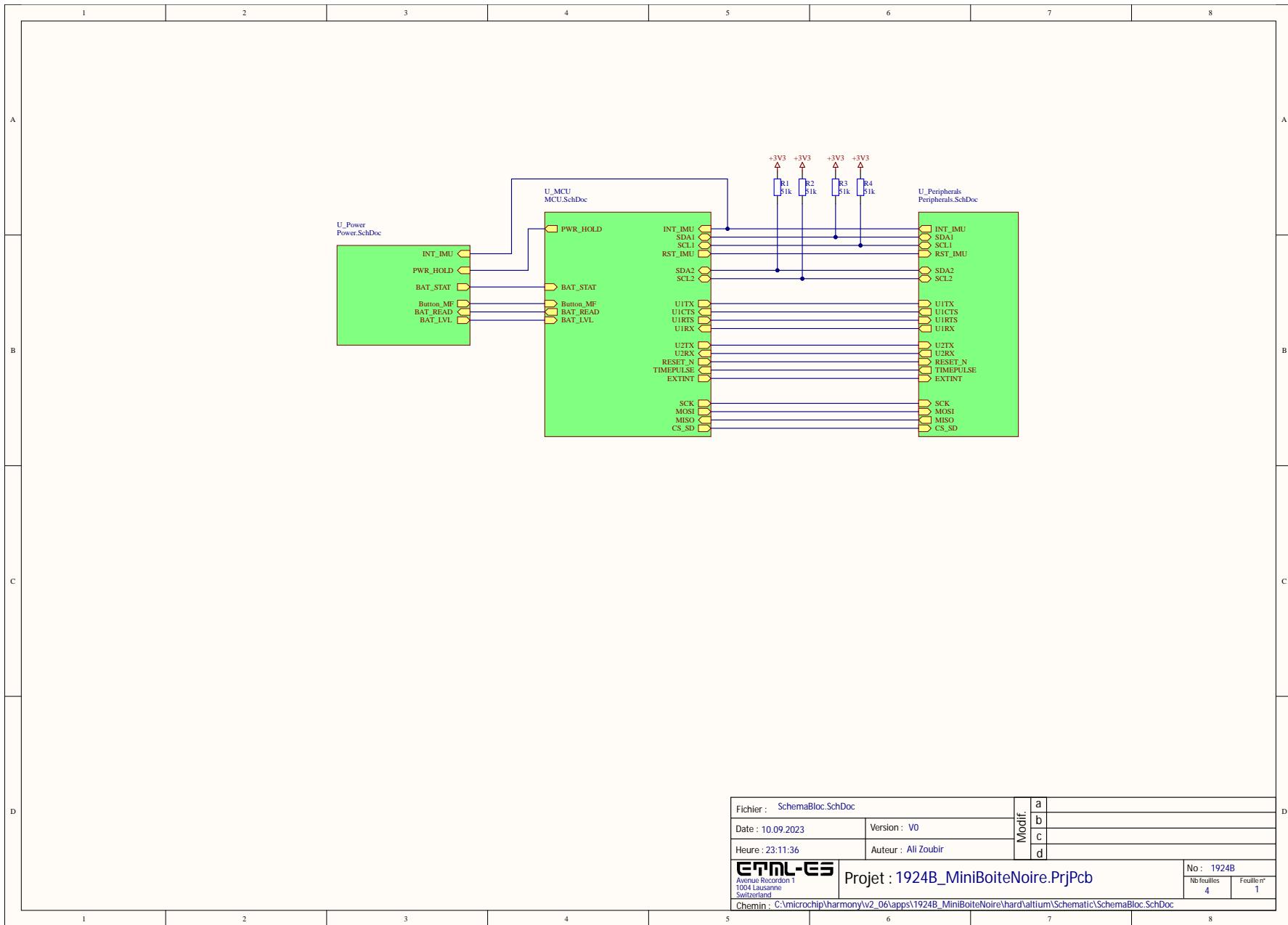
10 Bibliographie

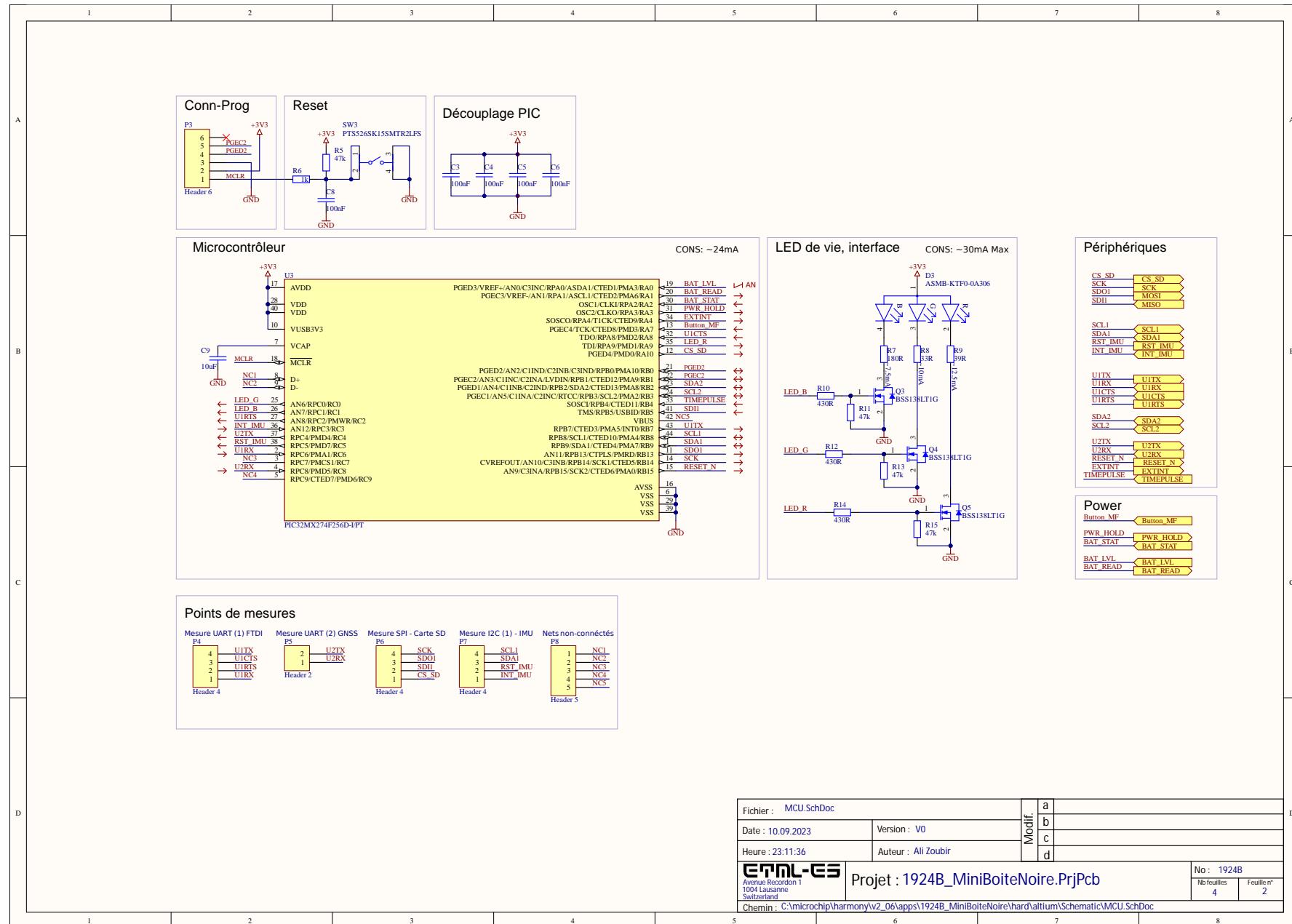
Références

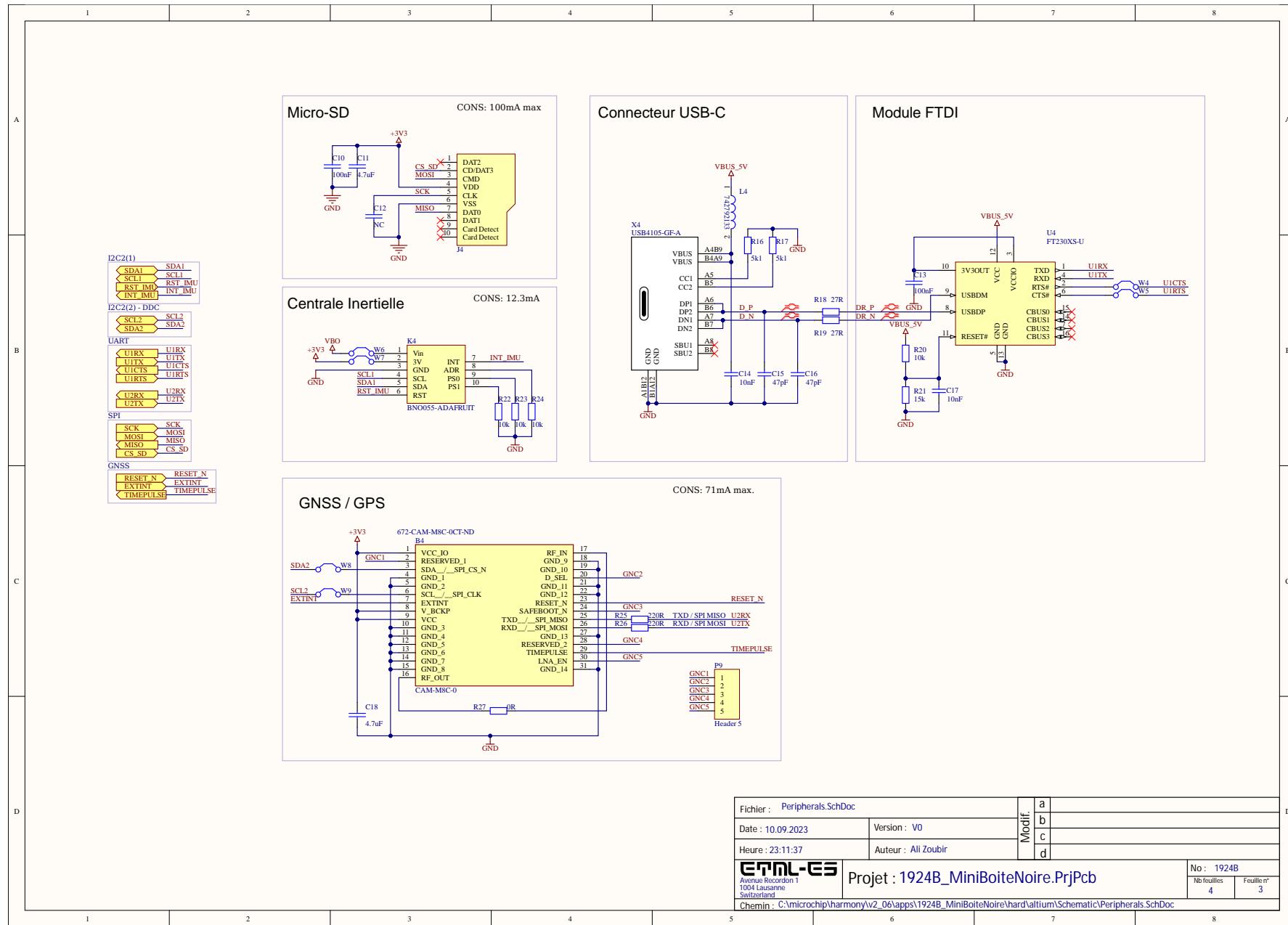
- [1] P. Kordowski, Z. Jakielaszek, M. Nowakowski, and A. Panas, “Miniaturized flight data recorder for unmanned aerial vehicles and ultralight aircrafts,” in *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pp. 484–488, 2018.

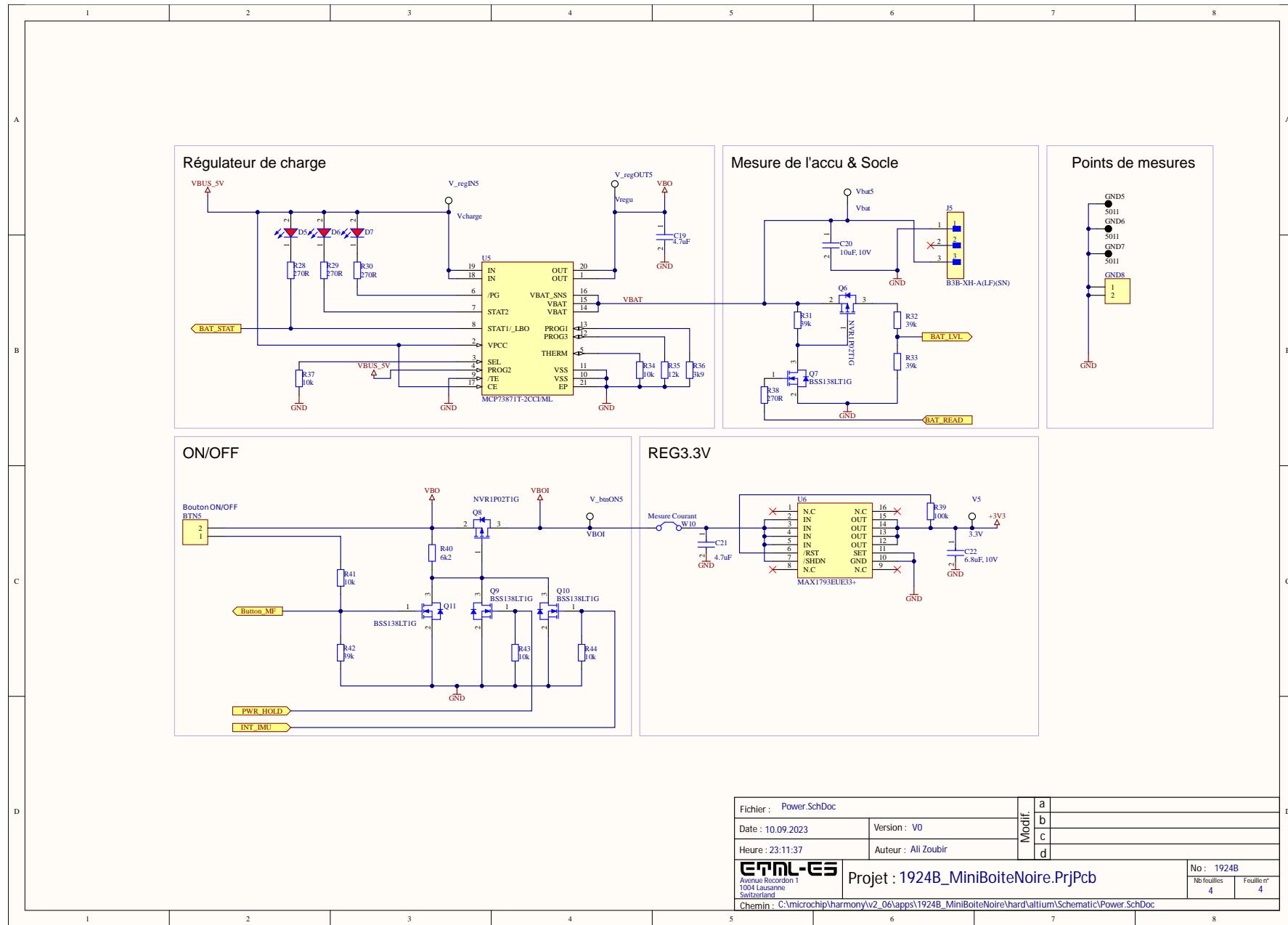
11 Annexes

11.1 Schéma









11.2 Dessin du boîtier

69

構成内容 / Components			
No.	名称 / Part name	数量 / Pcs	材質 / Material
1	カバー / Cover	1	ASA樹脂 UL94HB / ASA UL94HB ABS樹脂 UL94HB / ABS UL94HB
2	ボディー / Base	1	ASA樹脂 UL94HB / ASA UL94HB ABS樹脂 UL94HB / ABS UL94HB

■テクニカルデータ / Technical Data ■
使用温度範囲 / Operating temperature ASA : -30°C ~ +60°C
ABS : -10°C ~ +60°C

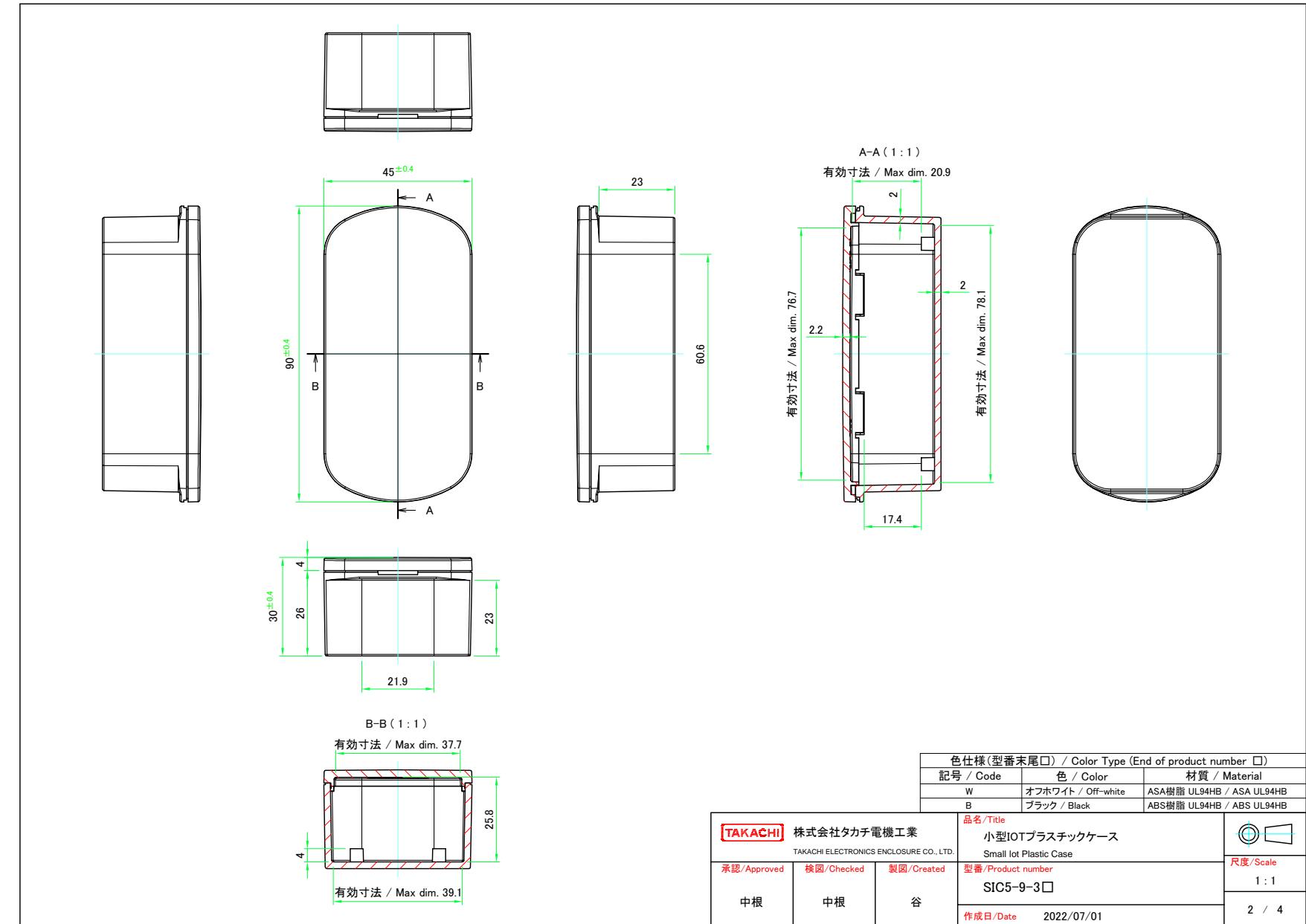
※ プラスチックは低温環境になるほど脆化します。
低温環境でご使用の場合、衝撃が加わらないよう設置して下さい。
It may become brittle and break under impact
when temperature is lowered close to -30° C.

色仕様(型番末尾□) / Color Type (End of product number □)		
記号 / Code	色 / Color	材質 / Material
W	オフホワイト / Off-white	ASA樹脂 UL94HB / ASA UL94HB
B	ブラック / Black	ABS樹脂 UL94HB / ABS UL94HB

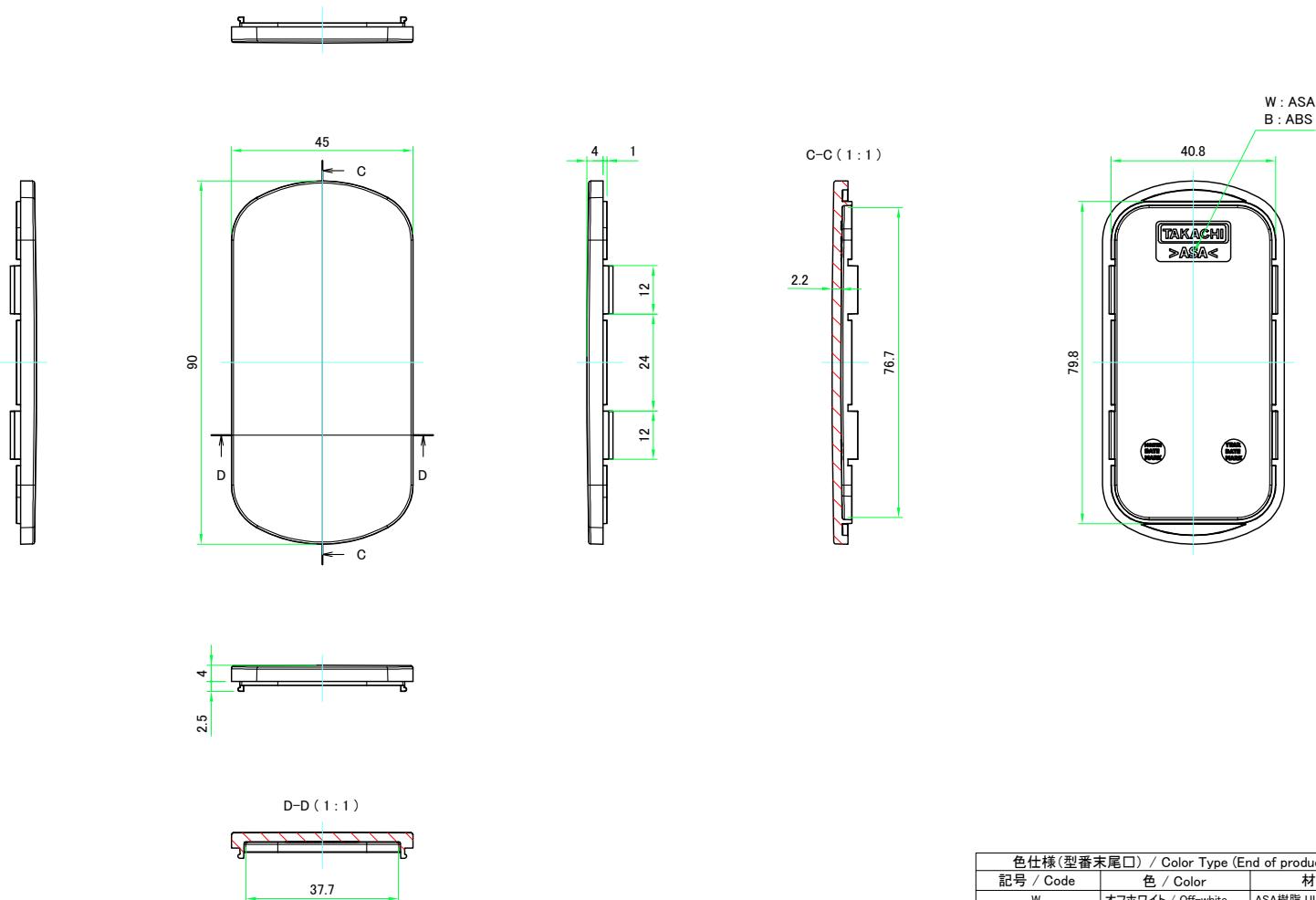
本図面および付帯する部品図の著作権は株式会社タカチ電機工業に帰属します。/ Copyright (C) TAKACHI ELECTRONICS ENCLOSURE CO., LTD. All Rights Reserved.

TAKACHI	株式会社タカチ電機工業			品名/Title	
			小型IoTプラスチックケース		
			Small IoT Plastic Case		
承認/Approved	検査/Checked	製図/Created	型番/Product number		
中根	中根	谷	SIC5-9-3□		
			作成日/Date 2022/11/24		

寸度/Scale
1.5 : 1
1 / 4

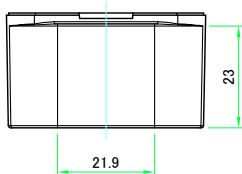
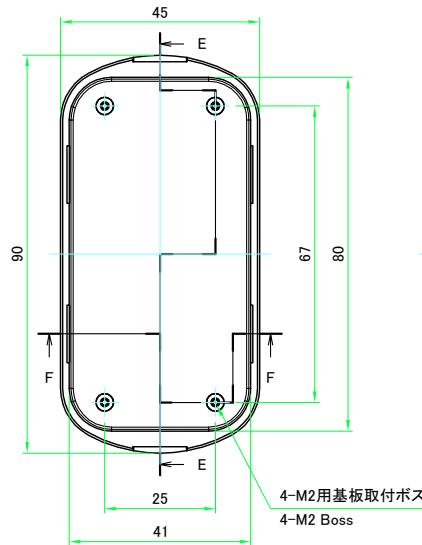
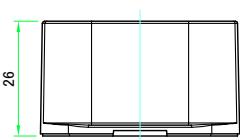


カバー / Cover

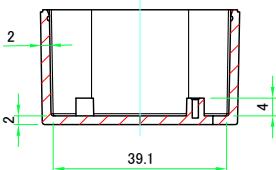


色仕様(型番末尾□) / Color Type (End of product number □)		
記号 / Code	色 / Color	材質 / Material
W	オフホワイト / Off-white	ASA樹脂 UL94HB / ASA UL94HB
B	ブラック / Black	ABS樹脂 UL94HB / ABS UL94HB
TAKACHI 株式会社タカチ電機工業 TAKACHI ELECTRONICS ENCLOSURE CO., LTD.		
承認/Approved	検査/Checked	製図/Created
中根	中根	谷
品名/Title 小型IoTプラスチックケース Small IoT Plastic Case		
型番/Product number SIC5-9-3□		
作成日/Date 2022/07/01		
尺度/Scale 1 : 1		
3 / 4		

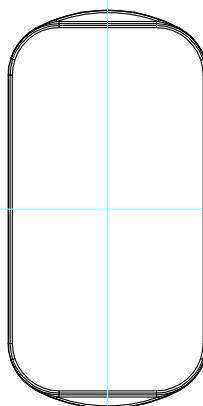
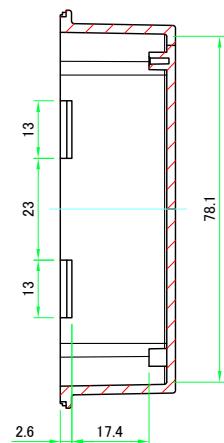
ボディー / Base



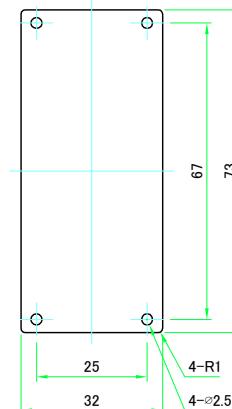
F-F (1:1)



E-E (1:1)

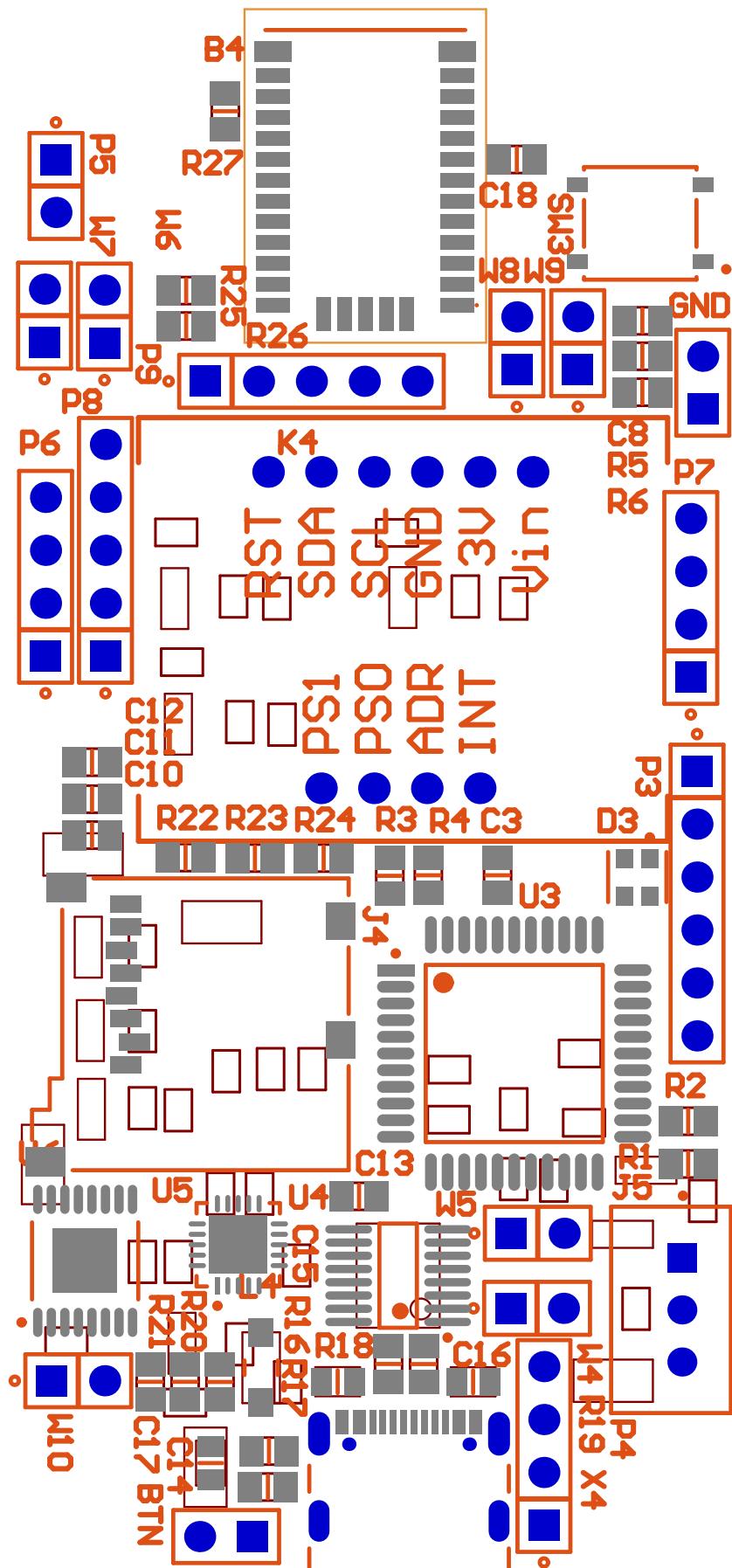


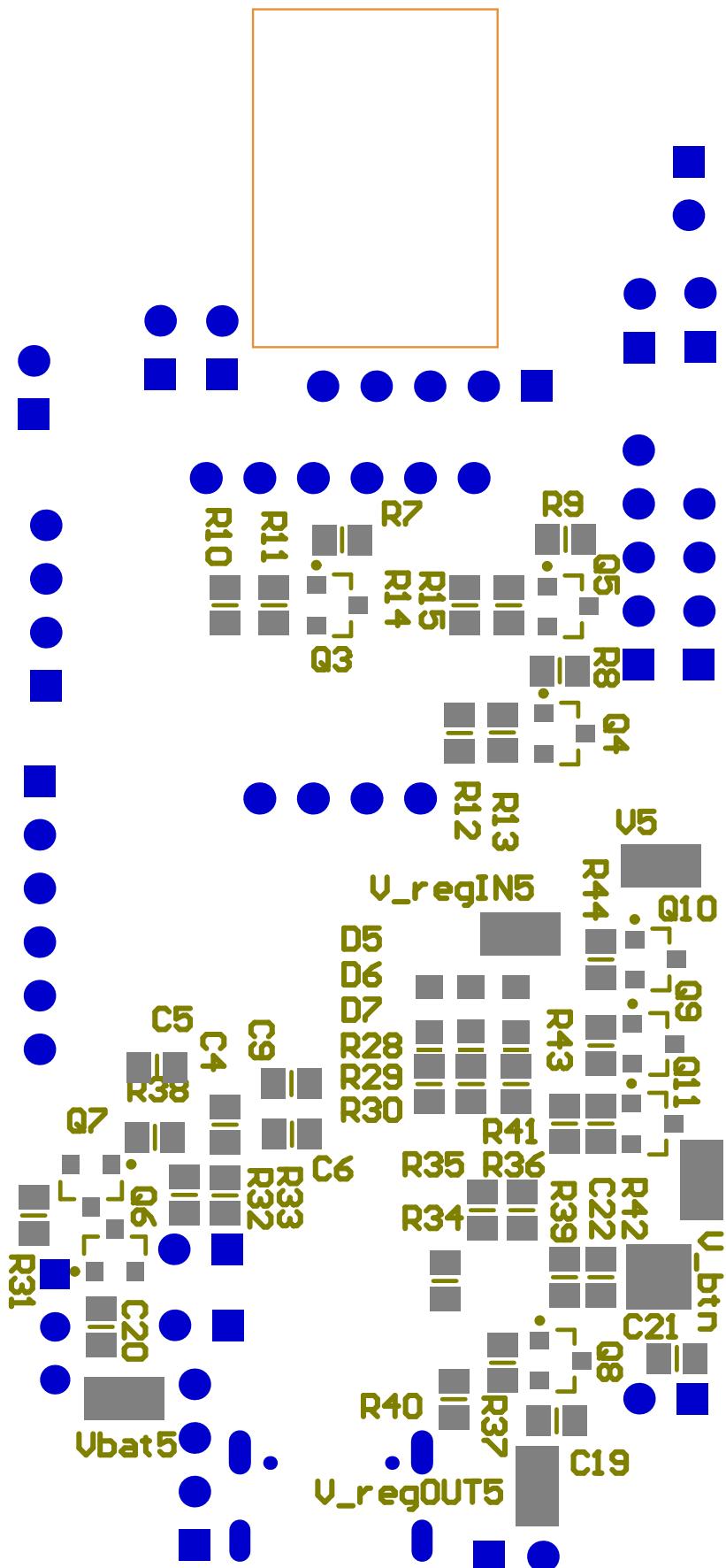
推奨基板形状 *販売していません。
Recommended PCB dimension * NOT SUPPLIED



色仕様(型番末尾□) / Color Type (End of product number □)		
記号 / Code	色 / Color	材質 / Material
W	オフホワイト / Off-white	ASA樹脂 UL94HB / ASA UL94HB
B	ブラック / Black	ABS樹脂 UL94HB / ABS UL94HB
TAKACHI 株式会社タカチ電機工業 TAKACHI ELECTRONICS ENCLOSURE CO., LTD.		
承認 / Approved	検査 / Checked	製図 / Created
中根	中根	谷
品名 / Title 小型IOTプラスチックケース Small IoT Plastic Case		
型番 / Product number SIC5-9-3□		
作成日 / Date 2022/07/01		
尺度 / Scale 1 : 1		
4 / 4		

11.3 Fichiers PCB





11.4 Code du firmware

```

1  ****
2  MPLAB Harmony Application Source File
3
4  Company:
5      Microchip Technology Inc.
6
7  File Name:
8      app.c
9
10 Summary:
11     This file contains the source code for the MPLAB Harmony application.
12
13 Description:
14     This file contains the source code for the MPLAB Harmony application. It
15     implements the logic of the application's state machine and it may call
16     API routines of other MPLAB Harmony modules in the system, such as drivers,
17     system services, and middleware. However, it does not call any of the
18     system interfaces (such as the "Initialize" and "Tasks" functions) of any of
19     the modules in the system or make any assumptions about when those functions
20     are called. That is the responsibility of the configuration-specific system
21     files.
22 ****
23
24 // DOM-IGNORE-BEGIN
25 ****
26 Copyright (c) 2013-2014 released Microchip Technology Inc. All rights reserved.
27
28 Microchip licenses to you the right to use, modify, copy and distribute
29 Software only when embedded on a Microchip microcontroller or digital signal
30 controller that is integrated into your product or third party product
31 (pursuant to the sublicense terms in the accompanying license agreement).
32
33 You should refer to the license agreement accompanying this Software for
34 additional information regarding your rights and obligations.
35
36 SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
37 EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
38 MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
39 IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
40 CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
41 OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
42 INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
43 CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
44 SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
45 (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
46 ****
47 // DOM-IGNORE-END
48
49
50 // ****
51 // ****
52 // Section: Included Files
53 // ****
54 // ****
55
56 #include "app.h"
57 #include "bno055.h"
58 #include "bno055_support.h"
59 #include "GNSS/u_gnss_pos.h"
60 #include "Mc32_I2cUtiICCS.h"
61 #include "Mc32_serComm.h"
62 #include "Mc32_sdFatGest.h"
63 #include "Mc32Debounce.h"
64 #include "uart_FIFO.h"
65 #include "GNSS/u_ubx_protocol.h"
66 #include <stdio.h>
67
68 // ****
69 // ****
70 // Section: Global Data Definitions
71 // ****
72 // ****
73 /* Switch descriptor */

```

```

74     S_SwitchDescriptor switchDescr;
75     // **** Application Data
76
77     Summary:
78         Holds application data
79
80     Description:
81         This structure holds the application's data.
82
83     Remarks:
84         This structure should be initialized by the APP_Initialize function.
85
86     Application strings and buffers are be defined outside this structure.
87 */
88
89
90 APP_DATA appData;
91 TIMER_DATA timeData;
92
93
94 // **** Application Callback Functions
95 // **** Application Local Functions
96 // **** Application Initialization and State Machine Functions
97 // ****
98 // ****
99
100 void delayTimer_callback(){
101     /* Increment delay timer */
102     timeData.delayCnt++;
103 }
104
105 void stateTimer_callback()
106 {
107     /* Increment all counters */
108     timeData.ledCnt++;
109     timeData.measCnt[BNO055_idx]++;
110     timeData.measCnt[GNSS_idx]++;
111     timeData.inactiveCnt++;
112     timeData.tmrTickFlag = true;
113     /* When the button is pressed, the hold time is counted. */
114     if(timeData.flagCntBtnPressed){
115         timeData.cntBtnPressed++;
116     }
117     /* Do debounce on button every 10 ms */
118     DoDebounce(&switchDescr, ButtonMFStateGet());
119     /* Start a measure set each IMU period */
120     if( ( timeData.measCnt[BNO055_idx] % (timeData.measPeriod[BNO055_idx]/10) ) == 0 )
121         timeData.measTodo[BNO055_idx] = true;
122
123     /* Start a measure set each GNSS period */
124     if( ( timeData.measCnt[GNSS_idx] % (timeData.measPeriod[GNSS_idx]/10) ) == 0 )
125         timeData.measTodo[GNSS_idx] = true;
126     /* Manage LED if enabled */
127     if((timeData.ledCnt % LED_PERIOD == 0) && (appData.ledState == true))
128         LED_Boff();
129 }
130
131 // **** Application Local Functions
132 // ****
133 // **** Application Initialization and State Machine Functions
134 // ****
135 // ****
136 static void stopLogging( void );
137 static void btnTaskGest( void );
138 static void sys_shutdown( void );
139 static void startLogging( void );
140
141 // ****
142 // **** Application Initialization and State Machine Functions
143 // ****
144 // ****
145 // ****
146 // ****

```

```

147     Function:
148         void APP_Initialize ( void )
149
150     Remarks:
151         See prototype in app.h.
152     */
153
154 void APP_Initialize ( void )
155 {
156     /* Keep the device ON */
157     PWR_HOLDOn();
158     LED_GOn();
159
160     // Start GNSS
161     //char gnssMessage[4+U_UBX_PROTOCOL_OVERHEAD_LENGTH_BYTES];
162     //char msgBody[4] = {0xFF, 0xFF, 0x09, 0x00};
163
164     // GNSS initialisation data
165     /*char gnssMessage2[4+U_UBX_PROTOCOL_OVERHEAD_LENGTH_BYTES];
166     char msgBody2[13] = {0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00,
167     0x00, 0x01};
168     char msgBody3[4] = {0xFF, 0xFF, 0x09, 0x00};*/
169
170     // Initialization of the USART FIFOs
171     initFifo(&uartFifoRx, FIFO_RX_SIZE, a_fifoRx, 0);
172     initFifo(&uartFifoTx, FIFO_TX_SIZE, a_fifoTx, 0);
173
174     /* Start timers*/
175     DRV_TMR0_Start();
176     /* Init i2c bus */
177     i2c_init(1);
178
179     /* Reset GNSS*/
180     RESET_NOff();
181     BNO055_delay_msek(100);
182     /* Unreset GNSS */
183     RESET_NOn();
184     BNO055_delay_msek(300);
185
186     // Start GNSS
187     //uUbxProtocolEncode(0x06, 0x04, msgBody, 4, gnssMessage);
188     //serTransmitbuffer(USART_ID_2, gnssMessage, sizeof(gnssMessage));
189
190     /* Reset IMU */
191     RST_IMUOff();
192     BNO055_delay_msek(100);
193     RST_IMUOn();
194     BNO055_delay_msek(100);
195
196     // Reset interrupt pin
197     bno055_set_intr_rst(1);
198
199     /* Place the App state machine in its initial state. */
200     appData.state = APP_STATE_INIT;
201 }
202
203
204
205 /***** Function:
206     void APP_Tasks ( void )
207
208     Remarks:
209         See prototype in app.h.
210     */
211
212
213 void APP_Tasks ( void )
214 {
215     /* Local bno055 data */
216     s_bno055_data bno055_local_data;
217     /*s_gnssData gnss_ubx_local_data;
218     minmea_messages gnss_nmea_local_data;

```

```

219     //enum minmea_sentence_id gnss_nmea_msgId = MINMEA_UNKNOWN;
220     /* CONFIGURATION */
221     static char charRead[CHAR_READ_BUFFER_SIZE] = {0};
222     static uint32_t readCnt = 0;
223     static unsigned long oldIntG = 0;
224     static unsigned long oldIntI = 0;
225     static uint32_t oldInaPer = 0;
226     static bool oldLed = 0;
227     static int ledStateTemp = 0;
228
229     // Character to send through USART
230     static char charToSend = 0;
231
232     /* Check the application's current state. */
233     switch (appData.state)
234     {
235         /* Application's initial state. */
236         case APP_STATE_INIT:
237         {
238             // Init delay
239             BNO055_delay_msek(500);
240             // Init and Measure set
241             bno055_init_readout();
242             BNO055_delay_msek(10);
243
244             /* BNO055 motion interrupt mode */
245             bno055_set_accel_any_motion_no_motion_axis_enable(
246                 BNO055_ACCEL_ANY_MOTION_NO_MOTION_X_AXIS, BNO055_BIT_ENABLE);
247             bno055_set_accel_any_motion_no_motion_axis_enable(
248                 BNO055_ACCEL_ANY_MOTION_NO_MOTION_Y_AXIS, BNO055_BIT_ENABLE);
249             bno055_set_accel_any_motion_no_motion_axis_enable(
250                 BNO055_ACCEL_ANY_MOTION_NO_MOTION_Z_AXIS, BNO055_BIT_ENABLE);
251
252             bno055_set_accel_any_motion_durn(1);
253             bno055_set_accel_any_motion_thres(25);
254
255             bno055_set_intr_accel_any_motion(BNO055_BIT_ENABLE);
256             bno055_set_intr_mask_accel_any_motion(BNO055_BIT_ENABLE);
257             bno055_set_intr_accel_no_motion(BNO055_BIT_DISABLE);
258
259             /*bno055_set_accel_slow_no_motion_enable(0);
260             bno055_set_intr_accel_no_motion(BNO055_BIT_DISABLE);
261             bno055_set_intr_mask_accel_no_motion(BNO055_BIT_DISABLE);*/
262
263             /* go to service task */
264             appData.state = APP_STATE_CONFIG;
265             /* Init ltime_BNO055 counter */
266             timeData.ltime[BNO055_idx] = 0;
267             break;
268         }
269         case APP_STATE_CONFIG:
270         {
271             // Reset interrupt pin
272             bno055_set_intr_rst(1);
273             /* Init sd_card parameters and read/create config File */
274             sd_fat_cfg_init(&timeData.measPeriod[GNSS_idx], &timeData.measPeriod[
275                 BNO055_idx], &appData.ledState, &timeData.inactivePeriod);
276
277             LED_GOff();
278
279             /* --- Unmount timeout --- */
280             if (ButtonMFStateGet())
281                 appData.state = APP_STATE_SHUTDOWN;
282             break;
283         }
284         case APP_STATE_LOGGING:
285         {
286             // BNO055 Measure routine
287             if((timeData.measTodo[BNO055_idx] == true )&&(sd_logGetState() == APP_IDLE
288             ))
289             {
290                 // If LED enabled

```

```

287         if(appData.ledState == true){
288             timeData.ledCnt = 0;
289             LED_BOn();
290         }
291         /* BNO055 Read all important info routine */
292         bno055_local_data.comres = bno055_read_routine(&bno055_local_data);
293         /* Delta time */
294         bno055_local_data.d_time = timeData.measCnt[BNO055_idx] - timeData.
295         ltime[BNO055_idx];
296         /* Flag measure if acceleration detected */
297         if((bno055_local_data.linear_accel.x >= 2*G) || (bno055_local_data.
298         linear_accel.y >= 2*G) || (bno055_local_data.linear_accel.z >= 2*G))
299             bno055_local_data.flagImportantMeas = 1;
300         else
301             bno055_local_data.flagImportantMeas = 0;
302
303         /* Detect activity */
304         if((bno055_local_data.linear_accel.x >= ACCEL_ACTIV_DETECT_msq)
305             || (bno055_local_data.linear_accel.y >= ACCEL_ACTIV_DETECT_msq)
306             || (bno055_local_data.linear_accel.z >= ACCEL_ACTIV_DETECT_msq))
307             timeData.inactiveCnt = 0;
308
309         /* Write value to sdCard */
310         sd_IMU_scheduleWrite(&bno055_local_data);
311         /* Reset measure flag */
312         timeData.measTodo[BNO055_idx] = false;
313         /* Update last time counter */
314         timeData.ltime[BNO055_idx] = timeData.measCnt[BNO055_idx];
315     }
316     // GNSS Measure routine
317     else if((timeData.measTodo[GNSS_idx] == true )&&(sd_logGetState() ==
318     APP_IDLE))
319     {
320         /* Read GNSS position measure */
321         //gnss_posGet_nmea(&gnss_nmea_local_data, &gnss_nmea_msgId);
322         /* Write value to sdCard */
323         sd_GNSS_scheduleWrite(&gnss_nmea_local_data);
324         /* Reset measure flag */
325         timeData.measTodo[GNSS_idx] = false;
326     }
327     else
328     {
329         /* No comm, so no error */
330         bno055_local_data.comres = 0;
331         //LED_BOff();
332     }
333
334     /* If error detected : error LED */
335     if((bno055_local_data.comres != 0)|| (sd_logGetState() == APP_MOUNT_DISK))
336         LED_ROn();
337     else
338         LED_ROff();
339
340     /* --- SD FAT routine --- */
341     sd_fat_logging_task();
342     /* --- Button routine --- */
343     btnTaskGest();
344     /* --- Inactivity shutdown --- */
345     if (timeData.inactiveCnt >= (timeData.inactivePeriod*100))
346         appData.state = APP_STATE_SHUTDOWN;
347
348     /* --- LIVE GNSS COMMAND --- */
349     if(pollSerialCmds(USART_ID_1, "glive", "GLIVE", "-lvg", "-LVG")){
350         /* Stop SD card logging */
351         stopLogging();
352         /* USB communication states */
353         appData.state = APP_STATE_COMM_LIVE_GNSS;
354         LED_BOn();
355     }
356     /* --- LIVE IMU COMMAND --- */
357     if(pollSerialCmds(USART_ID_1, "ilive", "ILIVE", "-lvi", "-LVI")){
358         /* Stop SD card logging */
359     }

```

```

357         stopLogging();
358         /* USB communication states */
359         appData.state = APP_STATE_COMM_LIVE_IMU;
360         LED_GOn();
361         /* Deactivate USART2 (not used) */
362         PLIB_USART_Disable(USART_ID_2);
363         /* Reset measure flags and stop timer */
364         DRV_TMR1_Start();
365     }
366
367     /* --- SHUTDOWN SYSTEM COMMAND --- */
368     if(pollSerialCmds(USART_ID_1, "shutdown", "SHUTDOWN", "-off", "-OFF")){
369
370         /* Turn off state */
371         appData.state = APP_STATE_SHUTDOWN;
372     }
373
374     /* --- CONFIG BLACKBOX --- */
375     if(pollSerialCmds(USART_ID_1, "config", "CONFIG", "-cfg", "-CFG")){
376
377         // Stop SD card logging
378         stopLogging();
379         /* Deactivate USART2 (not used) */
380         PLIB_USART_Disable(USART_ID_2);
381         serTransmitString(USART_ID_1, "CONFIGURATION MODE \r\n");
382         // Set config state to idle
383         sd_cfgSetState(APP_CFG_IDLE);
384         // Update configuration variables
385         oldIntG = timeData.measPeriod[GNSS_idx];
386         oldIntI = timeData.measPeriod[BNO055_idx];
387         oldLed = appData.ledState;
388         ledStateTemp = appData.ledState;
389         // Turn off state
390         appData.state = APP_STATE_CONFIGURATE_BBX;
391         LED_GOn();
392     }
393
394     /* --- GET GNSS LOGS --- */
395     if(pollSerialCmds(USART_ID_1, "glog", "GLOG", "-gl", "-GL")){
396
397         // Display GNSS logs
398         sd_fat_readDisplayFile("LOG_GNSS.txt");
399     }
400
401     /* --- GEST IMU LOGS --- */
402     if(pollSerialCmds(USART_ID_1, "ilog", "ILOG", "-il", "-IL")){
403
404         // Display IMU logs
405         sd_fat_readDisplayFile("LOG_IMU.csv");
406     }
407
408     /* --- DELETE COMMAND --- */
409     if(pollSerialCmds(USART_ID_1, "gclr", "GCLR", "-gc", "-GC")){
410
411         // Delete file
412         SYS_FS_FileDirectoryRemove("LOG_GNSS.txt");
413         serTransmitString(USART_ID_1, "GNSS LOG DELETED \r\n");
414     }
415
416     /* --- DELETE COMMAND --- */
417     if(pollSerialCmds(USART_ID_1, "iclr", "ICLR", "-ic", "-IC")){
418
419         // Delete file
420         SYS_FS_FileDirectoryRemove("LOG_IMU.csv");
421         serTransmitString(USART_ID_1, "IMU LOG DELETED \r\n");
422     }
423
424     break;
425 }
426 case APP_STATE_COMM_LIVE_GNSS:
427
428     /* No inactivity during this mode */
429     timeData.inactiveCnt = 0;
430     // Display GNSS live data trough USART 1
431     if(getReadSize(&uartFifoRx) > 0){
432
433         getCharFromFifo(&uartFifoRx, &charToSend);
434         PLIB_USART_TransmitterByteSend(USART_ID_1, charToSend);
435     }
436
437     // If exit command detected, return to logging

```

```

429         if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x", "x"))
430             startLogging();
431         break;
432     case APP_STATE_COMM_LIVE_IMU:
433         /* No inactivity during this mode */
434         timeData.inactiveCnt = 0;
435         // BNO055 Measure routine
436         if(timeData.measTodo[BNO055_idx] == true )
437         {
438             // If LED enabled
439             if(appData.ledState > 0){
440                 timeData.ledCnt = 0;
441                 LED_BOn();
442             }
443             /* BNO055 Read all important info routine */
444             bno055_local_data.comres = bno055_read_routine(&bno055_local_data);
445             /* Delta time */
446             bno055_local_data.d_time = timeData.measCnt[BNO055_idx] - timeData.
447             ltime[BNO055_idx];
448             /* Display readed values */
449             serDisplayValues(&bno055_local_data);
450
451             /* Reset measure flag */
452             timeData.measTodo[BNO055_idx] = false;
453             /* Update last time counter */
454             timeData.ltime[BNO055_idx] = timeData.measCnt[BNO055_idx];
455         }
456         // If exit command detected, return to logging
457         if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x", "x")){
458             startLogging();
459             /* Reactivate USART2 (used) */
460             PLIB_USART_Enable(USART_ID_2);
461         }
462
463         break;
464
465     case APP_STATE_CONFIGURATE_BBX:
466         /* No inactivity during this mode */
467         timeData.inactiveCnt = 0;
468         // Get command's characters
469         while(!DRV_USART0_ReceiverBufferIsEmpty())&&(readCnt <
470             CHAR_READ_BUFFER_SIZE)){
471             charRead[readCnt] = PLIB_USART_ReceiverByteReceive(USART_ID_1);
472             readCnt++;
473         }
474         // Command
475         if(readCnt >= CHAR_READ_BUFFER_SIZE)
476         {
477             /* Reset read counter */
478             readCnt = 0;
479             /* Clear read buffer */
480             memset(charRead,0,CHAR_READ_BUFFER_SIZE);
481         }
482
483         // Detect ENTER (End of command)
484         if(strstr(charRead, "\r") != NULL){
485             // Scan command data
486             sscanf(charRead, "INTG:%5lu", &timeData.measPeriod[GNSS_idx]);
487             sscanf(charRead, "INTI:%5lu", &timeData.measPeriod[BNO055_idx]);
488             sscanf(charRead, "LEDV:%2d", &ledStateTemp);
489             sscanf(charRead, "TOFF:%5d", &timeData.inactivePeriod);
490             // Cast int into boolean
491             if (ledStateTemp > 0)
492                 appData.ledState = true;
493             else
494                 appData.ledState = false;
495
496             /* Reset read counter */
497             readCnt = 0;
498             /* Clear read buffer */
499             memset(charRead,0,CHAR_READ_BUFFER_SIZE);
500         }

```

```

500     // If config value changed
501     if((timeData.measPeriod[GNSS_idx] != oldIntG) || (timeData.measPeriod[BNO055_idx] != oldIntI) || (appData.ledState != oldLed)
502         || (timeData.inactivePeriod != oldInaPer)){
503
504         serTransmitString(USART_ID_1, "COMMAND : VALUE CHANGED \r\n");
505         // If data is not valid, keep the previous one
506         if(timeData.measPeriod[GNSS_idx] <= 0){
507             timeData.measPeriod[GNSS_idx] = oldIntG;
508             serTransmitString(USART_ID_1, "ERROR GNSS VALUE <= 0 \r\n");
509         }
510         // If data is not valid, keep the previous one
511         if(timeData.measPeriod[BNO055_idx] <= 0){
512             timeData.measPeriod[BNO055_idx] = oldIntI;
513             serTransmitString(USART_ID_1, "ERROR IMU VALUE <= 0 \r\n");
514         }
515         // If data is not valid, keep the previous one
516         if(timeData.inactivePeriod <= 10){
517             timeData.inactivePeriod = oldInaPer;
518             serTransmitString(USART_ID_1, "ERROR INACTIVE PERIOD VALUE <= 10
519             \r\n");
520         }
521         /* Clear read buffer */
522         memset(charRead,0,CHAR_READ_BUFFER_SIZE);
523         // Write new config file
524         sd_CFG_Write (timeData.measPeriod[GNSS_idx], timeData.measPeriod[BNO055_idx], appData.ledState, timeData.inactivePeriod, true);
525
526         // Update polling config parameter
527         oldIntG = timeData.measPeriod[GNSS_idx];
528         oldIntI = timeData.measPeriod[BNO055_idx];
529         oldLed = appData.ledState;
530         oldInaPer = timeData.inactivePeriod;
531
532         // Check occurrence with commands
533         if((strstr(charRead, "exit") != NULL) || (strstr(charRead, "EXIT") != NULL)
534             || (strstr(charRead, "x") != NULL) || (strstr(charRead, "X") != NULL))
535         {
536             /* Command detected */
537             startLogging();
538             /* Clear read buffer */
539             memset(charRead,0,CHAR_READ_BUFFER_SIZE);
540             /* Reset read counter */
541             readCnt = 0;
542             /* Reactivate USART2 (used) */
543             PLIB_USART_Enable(USART_ID_2);
544             break;
545         }
546         // Manipulate config file
547         sd_fat_config_task ( false );
548         break;
549
550     case APP_STATE_SHUTDOWN:
551     {
552         /* Save and shutdown system */
553         sys_shutdown();
554         break;
555     }
556
557     /* The default state should never be executed. */
558     default:
559     {
560         /* TODO: Handle error in application's state machine. */
561         break;
562     }
563 }
564 void appStateSet( APP_STATES newState ){
565     appData.state = newState;
566 }
567 static void btnTaskGest( void ){

```

```

569     static bool Hold = false;
570     /* Button management : if rising edge detected */
571     if((ButtonMFStateGet())||(Hold == true))
572     {
573         /* Hold until falling edge */
574         Hold = true;
575         /* Start counting pressed time */
576         timeData.flagCntBtnPressed = true;
577         /* If falling edge detected */
578         if (ButtonMFStateGet() == 0)
579         {
580             /* Reset flag and switchdescr */
581             timeData.flagCntBtnPressed = false;
582             DebounceClearReleased(&switchDescr);
583             /* If pressed more time than power off */
584             if(timeData.cntBtnPressed >= BTN_HOLD_SHUTDOWN_x10ms){
585                 /* Power off the system */
586                 appData.state = APP_STATE_SHUTDOWN;
587             }
588             timeData.cntBtnPressed = 0;
589             Hold = false;
590         }
591     }
592 }
593
594 static void sys_shutdown( void ) {
595     /* Display shutting off mode */
596     LED_BOff();
597     LED_GOff();
598     LED_ROn();
599
600     /* If and SD card is mounted */
601     if(sd_logGetState() != APP_MOUNT_DISK){
602         /* Wait until SD available */
603         while(sd_logGetState() != APP_IDLE){
604             /* SD FAT routine */
605             sd_fat_logging_task();
606         }
607         /* Unmount disk */
608         sd_logSetState(APP_UNMOUNT_DISK);
609         /* Wait until unmounted*/
610         while(sd_logGetState() != APP_IDLE){
611             sd_fat_logging_task();
612         }
613     }
614     /* Set acceleration only operation to save power */
615     bno055_set_operation_mode(BNO055_OPERATION_MODE_ACONLY);
616     /* set the power mode as LOW POWER*/
617     bno055_set_power_mode(BNO055_POWER_MODE_LOWPOWER);
618     bno055_set_intr_accel_no_motion(BNO055_BIT_DISABLE);
619     // Reset interrupt pin
620     bno055_set_intr_rst(1);
621     do{
622         /* turn off the device */
623         PWR_HOLDOff();
624     }while(ButtonMFStateGet() == 0);
625 }
626
627 static void stopLogging (void)
628 {
629     /* Reset measure flags and stop timer */
630     DRV_TMR1_Stop();
631     timeData.measTodo[GNSS_idx] = false;
632     timeData.measTodo[BNO055_idx] = false;
633
634     /* Finish config */
635     while(sd_cfgGetState() != APP_CFG_IDLE){
636         sd_fat_cfg_init(&timeData.measPeriod[GNSS_idx], &timeData.measPeriod[BNO055_idx], &appData.ledState, &timeData.inactivePeriod);
637     }
638
639     /* Finish logging */
640     while(sd_logGetState() != APP_IDLE){

```

```
641     sd_fat_logging_task();
642 }
643
644 /* Reset Leds states */
645 LED_ROff();
646 LED_ROff();
647 LED_GOff();
648 }
649
650 static void startLogging (void)
651 {
652     // Logging state
653     appData.state = APP_STATE_LOGGING;
654     // Restart timer 1
655     DRV_TMR1_Start();
656     /* Reset Leds states */
657     LED_ROff();
658     LED_ROff();
659     LED_GOff();
660 }
661
662 ****
663 End of File
664 */
665
```

```

1  ****
2  MPLAB Harmony Application Header File
3
4  Company:
5      Microchip Technology Inc.
6
7  File Name:
8      app.h
9
10 ****
11
12 //DOM-IGNORE-BEGIN
13 ****
14 Copyright (c) 2013-2014 released Microchip Technology Inc. All rights reserved.
15
16 Microchip licenses to you the right to use, modify, copy and distribute
17 Software only when embedded on a Microchip microcontroller or digital signal
18 controller that is integrated into your product or third party product
19 (pursuant to the sublicense terms in the accompanying license agreement).
20
21 You should refer to the license agreement accompanying this Software for
22 additional information regarding your rights and obligations.
23
24 SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
25 EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
26 MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
27 IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
28 CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
29 OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
30 INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
31 CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
32 SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
33 (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
34 ****
35 //DOM-IGNORE-END
36
37 #ifndef _APP_H
38 #define _APP_H
39
40 // ****
41 // ****
42 // Section: Included Files
43 // ****
44 // ****
45
46 #include <stdint.h>
47 #include <stdbool.h>
48 #include <stddef.h>
49 #include <stdlib.h>
50 #include "system_config.h"
51 #include "system_definitions.h"
52 #include "bno055.h"
53
54 // DOM-IGNORE-BEGIN
55 #ifdef __cplusplus // Provide C++ Compatibility
56
57 extern "C" {
58
59 #endif
60 // DOM-IGNORE-END
61
62
63 #define TIME_OUT          80000000U
64 #define BTN_HOLD_SHUTDOWN_x10ms 200
65 #define NB_MEASURES        2
66
67 #define ACCEL_ACTIV_DETECT_msq 0.3
68 #define T_CONFIG_TIMEOUT    20
69 #define T_INACTIVE_PERIOD_DEFAULT 20UL
70 #define T_INTERVAL_GNSS_DEFAULT 5000UL
71 #define T_INTERVAL_IMU_DEFAULT 500UL
72 #define LED_STATE_DEFAULT   (uint8_t)1
73

```

```

74 #define LED_PERIOD           5
75
76 #define CHAR_READ_BUFFER_SIZE 30
77
78 #define G                   9.81
79
80
81 // ****
82 // ****
83 // Section: Type Definitions
84 // ****
85 // ****
86 typedef struct {
87     s32 comres;
88     bool flagMeasReady;
89     uint8_t flagImportantMeas;
90     struct bno055_gravity_double_t gravity;
91     struct bno055_linear_accel_double_t linear_accel;
92     struct bno055_euler_double_t euler;
93     struct bno055_gyro_double_t gyro;
94     struct bno055_mag_double_t mag;
95     struct bno055_quaternion_t quaternion;
96     unsigned long time;
97     unsigned long l_time;
98     uint16_t d_time;
99 }s_bno055_data;
100 // ****
101 /* Application states
102
103 Summary:
104     Application states enumeration
105
106 Description:
107     This enumeration defines the valid application states. These states
108     determine the behavior of the application at various times.
109 */
110
111 typedef enum
112 {
113     /* Application's state machine's initial state. */
114     APP_STATE_INIT=0,
115     APP_STATE_CONFIG,
116     APP_STATE_LOGGING,
117     APP_STATE_FLAG_MEAS,
118     APP_STATE_COMM_LIVE_GNSS,
119     APP_STATE_COMM_LIVE_IMU,
120     APP_STATE_CONFIGURE_BBX,
121     APP_STATE_SHUTDOWN
122     /* TODO: Define states used by the application state machine. */
123 }
124 APP_STATES;
125
126
127 // ****
128 /* Application Data
129
130 Summary:
131     Holds application data
132
133 Description:
134     This structure holds the application's data.
135
136 Remarks:
137     Application strings and buffers are be defined outside this structure.
138 */
139
140 typedef struct
141 {
142     /* The application's current state */
143     APP_STATES state;
144     bool ledState;
145
146

```

```

147     /* TODO: Define any additional data used by the application. */
148
149 } APP_DATA;
150
151 typedef struct
152 {
153     /* DELAY DATA */
154     bool tmrTickFlag;
155     unsigned long delayCnt;
156
157     /* MEASURES DATA */
158     unsigned long measCnt[NB_MEASURES];
159     unsigned long ltime[NB_MEASURES];
160     bool measTodo[NB_MEASURES];
161     unsigned long measPeriod[NB_MEASURES];
162
163     unsigned long inactiveCnt;
164     uint32_t inactivePeriod;
165
166     /* DISPLAY DATA */
167     uint32_t ledCnt;
168
169     /* BUTTON DATA */
170     bool flagCntBtnPressed;
171     uint32_t cntBtnPressed;
172 } TIMER_DATA;
173
174 /* Measures index */
175 enum measure{BNO055_idx, GNSS_idx};
176
177 // ****
178 // ****
179 // Section: Application Callback Routines
180 // ****
181 // ****
182 /* These routines are called by drivers when certain events occur.
183 */
184
185 // ****
186 // ****
187 // Section: Application Initialization and State Machine Functions
188 // ****
189 // ****
190
191 // ****
192 Function:
193     void APP_Initialize ( void )
194
195 Summary:
196     MPLAB Harmony application initialization routine.
197
198 Description:
199     This function initializes the Harmony application. It places the
200     application in its initial state and prepares it to run so that its
201     APP_Tasks function can be called.
202
203 Precondition:
204     All other system initialization routines should be called before calling
205     this routine (in "SYS_Initialize").
206
207 Parameters:
208     None.
209
210 Returns:
211     None.
212
213 Example:
214     <code>
215         APP_Initialize();
216     </code>
217
218 Remarks:
219     This routine must be called from the SYS_Initialize function.

```

```
220  */
221
222 void APP_Initialize ( void );
223
224
225 //*****
226 Function:
227     void APP_Tasks ( void )
228
229 Summary:
230     MPLAB Harmony Demo application tasks function
231
232 Description:
233     This routine is the Harmony Demo application's tasks function. It
234     defines the application's state machine and core logic.
235
236 Precondition:
237     The system and application initialization ("SYS_Initialize") should be
238     called before calling this.
239
240 Parameters:
241     None.
242
243 Returns:
244     None.
245
246 Example:
247     <code>
248         APP_Tasks();
249     </code>
250
251 Remarks:
252     This routine must be called from SYS_Tasks() routine.
253 */
254
255 void APP_Tasks( void );
256
257 // CALLBACKS
258 void delayTimer_callback( void );
259 void stateTimer_callback( void );
260
261 void appStateSet( APP_STATES newState );
262
263 #endif /* _APP_H */
264
265 //DOM-IGNORE-BEGIN
266 #ifdef __cplusplus
267 }
268#endif
269 //DOM-IGNORE-END
270
271 //*****
272 End of File
273 */
274
275
```

```
1  ****
2  System Interrupts File
3
4  File Name:
5      system_interrupt.c
6
7  Summary:
8      Raw ISR definitions.
9
10 Description:
11     This file contains a definitions of the raw ISRs required to support the
12     interrupt sub-system.
13
14 Summary:
15     This file contains source code for the interrupt vector functions in the
16     system.
17
18 Description:
19     This file contains source code for the interrupt vector functions in the
20     system. It implements the system and part specific vector "stub" functions
21     from which the individual "Tasks" functions are called for any modules
22     executing interrupt-driven in the MPLAB Harmony system.
23
24 Remarks:
25     This file requires access to the systemObjects global data structure that
26     contains the object handles to all MPLAB Harmony module objects executing
27     interrupt-driven in the system. These handles are passed into the individual
28     module "Tasks" functions to identify the instance of the module to maintain.
29 ****
30
31 // DOM-IGNORE-BEGIN
32 ****
33 Copyright (c) 2011-2014 released Microchip Technology Inc. All rights reserved.
34
35 Microchip licenses to you the right to use, modify, copy and distribute
36 Software only when embedded on a Microchip microcontroller or digital signal
37 controller that is integrated into your product or third party product
38 (pursuant to the sublicense terms in the accompanying license agreement).
39
40 You should refer to the license agreement accompanying this Software for
41 additional information regarding your rights and obligations.
42
43 SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
44 EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
45 MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
46 IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
47 CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
48 OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
49 INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
50 CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
51 SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
52 (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
53 ****
54 // DOM-IGNORE-END
55
56 // ****
57 // ****
58 // Section: Included Files
59 // ****
60 // ****
61
62 #include "system/common/sys_common.h"
63 #include "app.h"
64 #include "system_definitions.h"
65 #include "uart_FIFO.h"
66
67 // ****
68 // ****
69 // Section: System Interrupt Vector Functions
70 // ****
71 // ****
72 void __ISR(_UART_1_VECTOR, ipl0AUTO) _IntHandlerDrvUsartInstance0(void)
73 {
```

```

74     DRV_USART_TasksTransmit(sysObj.drvUsart0);
75     DRV_USART_TasksError(sysObj.drvUsart0);
76     DRV_USART_TasksReceive(sysObj.drvUsart0);
77 }
78
79
80
81
82 void __ISR(_UART_2_VECTOR, ipl1AUTO) _IntHandlerDrvUsartInstance1(void)
83 {
84     USART_ERROR usartStatus;
85     bool isTxBuffFull;
86     char charReceived;
87     char charToSend;
88     char TXsize;
89
90     //-----// RX
91     interrupt
92     if(PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_RECEIVE) &&
93         PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_RECEIVE)) {
94
95         // Parity error or overrun
96         usartStatus = PLIB_USART_ErrorsGet(USART_ID_2);
97
98         if ((usartStatus & (USART_ERROR_PARITY | USART_ERROR_FRAMING |
99             USART_ERROR_RECEIVER_OVERRUN)) == 0) {
100
101            // All char received are transferred to the FIFO
102            // 1 if ONE_CHAR, 4 if HALF_FULL and 6 3B4FULL
103            while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_2)) {
104
105                charReceived = PLIB_USART_ReceiverByteReceive(USART_ID_2);
106                putCharInFifo(&usartFifoRx, charReceived);
107            }
108            // Buffer is empty, clear interrupt flag
109            PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_RECEIVE);
110
111        } else{
112            // Deleting errors
113            // Reading errors clears them except for overrun
114            if((usartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
115                USART_ERROR_RECEIVER_OVERRUN) {
116
117                PLIB_USART_ReceiverOverrunErrorClear(USART_ID_2);
118            }
119        }
120
121
122     //-----// TX
123     interrupt
124     if (PLIB_INT_SourceFlagGet(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT) &&
125         PLIB_INT_SourceIsEnabled(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT)) {
126
127         TXsize = getReadSize(&usartFifoTx);
128         // i_cts = input(RS232_CTS);
129
130         isTxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_2);
131
132         if /*(i_cts == 0) && */(TXsize > 0) && (isTxBuffFull == false)) {
133             do{
134                 getCharFromFifo(&usartFifoTx, &charToSend);
135                 if(charToSend != '\0') PLIB_USART_TransmitterByteSend(USART_ID_2,
136                     charToSend);
137                 /*i_cts = RS232_CTS;*/
138                 TXsize = getReadSize(&usartFifoTx);
139                 isTxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_2);
140             }while(/*(i_cts == 0) && */( TXsize > 0 ) && isTxBuffFull == false);
141
142             // Disables TX interrupt (to avoid unnecessary interruptions if there's
143             // nothing left to transmit)
144             if(TXsize == 0){
145

```

```
144         PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT) ;
145     }
146     // Clears the TX interrupt Flag
147     PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_2_TRANSMIT) ;
148 }
149 }
150 }
151
152 void __ISR(_TIMER_1_VECTOR, ipl6AUTO) IntHandlerDrvTmrInstance0(void)
153 {
154     PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1) ;
155     delayTimer_callback() ;
156 }
157 void __ISR(_TIMER_2_VECTOR, ipl5AUTO) IntHandlerDrvTmrInstance1(void)
158 {
159     PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_2) ;
160     stateTimer_callback() ;
161 }
162
163 void __ISR(_SPI_1_VECTOR, ipl1AUTO) _IntHandlerSPIInstance0(void)
164 {
165     DRV_SPI_Tasks(sysObj.spiObjectIdx0) ;
166 }
167 /***** End of File ****/
168 */
169
170
```

```
1  /* **** */
2  /** Descriptive File Name
3
4  @Company
5      ETML-ES
6
7  @File Name
8      sd_fat_gest.c
9
10 @Summary
11     SD card fat system management
12
13 @Description
14     SD card fat system management
15 */
16 /* **** */
17
18 /* **** */
19 /* **** */
20 /* Section: Included Files */
21 /* **** */
22 /* **** */
23
24 /* This section lists the other files that are included in this file.
25 */
26
27 #include "Mc32_sdFatGest.h"
28 #include <stdio.h>
29 #include "app.h"
30 #include "bno055_support.h"
31 #include "GNSS/u_gnss_pos.h"
32 #include <stdio.h>
33 #include "uart_FIFO.h"
34 #include "MC32_serComm.h"
35
36 /* **** */
37 /* **** */
38 /* Section: File Scope or Global Data */
39 /* **** */
40 /* **** */
41
42 APP_FAT_DATA COHERENT_ALIGNED appFatData;
43 /* **** */
44 /** Descriptive Data Item Name
45
46 @Summary
47     Brief one-line summary of the data item.
48
49 @Description
50     Full description, explaining the purpose and usage of data item.
51     <p>
52         Additional description in consecutive paragraphs separated by HTML
53         paragraph breaks, as necessary.
54     <p>
55         Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.
56
57 @Remarks
58     Any additional remarks
59 */
60
61
62 /* **** */
63 /* **** */
64 // Section: Local Functions
65 /* **** */
66 /* **** */
67 // Function prototype
68 static uint8_t parseConfig(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
69     unsigned long *tInactive);
70 /* **** */
71 /* **** */
72 /* **** */
```

```

73 // Section: Interface Functions
74 /* **** Section: Interface Functions **** */
75 /* **** Section: Interface Functions **** */
76
77 void sd_fat_config_task ( bool init )
78 {
79     /* The application task cfg_state machine */
80     switch(appFatData.cfg_state)
81     {
82         case APP_CFG_MOUNT_DISK:
83             if(SYS_FS_Mount("/dev/mmcblk0", "/mnt/myDrive", FAT, 0, NULL) != 0)
84             {
85                 /* The disk could not be mounted. Try
86                  * mounting again until success. */
87                 LED_ROn();
88                 appFatData.cfg_state = APP_CFG_MOUNT_DISK;
89             }
89         else
90         {
91             /* Mount was successful. Unmount the disk, for testing. */
92             LED_ROff();
93             appFatData.cfg_state = APP_CFG_SET_CURRENT_DRIVE;
94         }
95         break;
96
97         case APP_CFG_SET_CURRENT_DRIVE:
98             if(SYS_FS_CurrentDriveSet("/mnt/myDrive") == SYS_FS_RES_FAILURE)
99             {
100                 /* Error while setting current drive */
101                 appFatData.cfg_state = APP_CFG_ERROR;
102             }
103         else
104         {
105             if(init == true)
106                 /* Open config file for reading. */
107                 appFatData.cfg_state = APP_CFG_OPEN_READ_CONFIG_FILE;
108             else
109                 /* Wait for further commands. */
110                 appFatData.cfg_state = APP_CFG_IDLE;
111         }
112         break;
113
114         case APP_CFG_OPEN_READ_CONFIG_FILE:
115             appFatData.fileCfgHandle = SYS_FS_FileOpen("CONFIG.txt",
116                                         (SYS_FS_FILE_OPEN_READ));
117             if(appFatData.fileCfgHandle == SYS_FS_HANDLE_INVALID)
118             {
119                 /* No config file, write default config file */
120                 sd_CFG_Write(T_INTERVAL_GNSS_DEFAULT, T_INTERVAL_IMU_DEFAULT,
121                               LED_STATE_DEFAULT, T_INACTIVE_PERIOD_DEFAULT, true);
122
123                 /* Re-try to open file as read */
124                 //appFatData.cfg_state = APP_CFG_OPEN_READ_CONFIG_FILE;
125
126             }
127         else
128         {
129             /* Create a directory. */
130             appFatData.cfg_state = APP_CFG_READ_CONFIG_FILE;
131         }
132         break;
133
134         case APP_CFG_READ_CONFIG_FILE:
135             /* If read was success, try writing to the new file */
136             if(SYS_FS_FileRead(appFatData.fileCfgHandle, appFatData.cfg_data,
137                               SYS_FS_FileSize(appFatData.fileCfgHandle)) == -1)
138             {
139                 /* Write was not successful. Close the file
140                  * and error out.*/
141                 SYS_FS_FileClose(appFatData.fileCfgHandle);
142                 appFatData.cfg_state = APP_CFG_ERROR;
143             }
144         else

```

```

145
146     {
147         appFatData.cfg_state = APP_CFG_CLOSE_FILE;
148     }
149     break;
150 case APP_CFG_OPEN_WRITE_CONFIG_FILE:
151     appFatData.fileCfgHandle = SYS_FS_FileOpen("CONFIG.txt",
152                                                 (SYS_FS_FILE_OPEN_WRITE));
153     if(appFatData.fileCfgHandle == SYS_FS_HANDLE_INVALID)
154     {
155         /* Could not open the file. Error out*/
156         appFatData.cfg_state = APP_CFG_ERROR;
157     }
158     else
159     {
160         /* Create a directory. */
161         appFatData.cfg_state = APP_CFG_WRITE_CONFIG_FILE;
162     }
163     break;
164
165 case APP_CFG_WRITE_CONFIG_FILE:
166     /* If read was success, try writing to the new file */
167     if(SYS_FS_FileStringPut(appFatData.fileCfgHandle, appFatData.cfg_data) == -1)
168     {
169         /* Write was not successful. Close the file
170          * and error out.*/
171         SYS_FS_FileClose(appFatData.fileCfgHandle);
172         appFatData.cfg_state = APP_CFG_ERROR;
173     }
174     else
175     {
176         appFatData.cfg_state = APP_CFG_CLOSE_FILE;
177     }
178     break;
179 case APP_CFG_CLOSE_FILE:
180     /* Close the file */
181     SYS_FS_FileClose(appFatData.fileCfgHandle);
182     /* The test was successful. Lets idle. */
183     if(init == true)
184         appFatData.cfg_state = APP_CFG_UNMOUNT_DISK;
185     else
186         appFatData.cfg_state = APP_CFG_IDLE;
187     break;
188
189 case APP_CFG_IDLE:
190     /* The application comes here when the demo
191      * has completed successfully. Switch on
192      * green LED. */
193     //BSP_LEDOn(APP_SUCCESS_LED);
194     LED_ROff();
195     break;
196 case APP_CFG_ERROR:
197     /* The application comes here when the demo
198      * has failed. Switch on the red LED.*/
199     //BSP_LEDOn(APP_FAILURE_LED);
200     LED_ROn();
201     break;
202 default:
203     break;
204
205 case APP_CFG_UNMOUNT_DISK:
206     if(SYS_FS_Unmount("/mnt/myDrive") != 0)
207     {
208         /* The disk could not be un mounted. Try
209          * un mounting again until success. */
210
211         appFatData.cfg_state = APP_CFG_UNMOUNT_DISK;
212     }
213     else
214     {
215         /* UnMount was successful. Mount the disk again */
216         appFatData.cfg_state = APP_CFG_IDLE;
217     }

```

```

217         break;
218     }
219 }
220
221 // Logging task
222 void sd_fat_logging_task ( void )
223 {
224     /* The application task log_state machine */
225     switch(appFatData.log_state)
226     {
227         case APP_MOUNT_DISK:
228             if(SYS_FS_Mount("/dev/mmcblk0", "/mnt/myDrive", FAT, 0, NULL) != 0)
229             {
230                 /* The disk could not be mounted. Try
231                  * mounting again until success. */
232
233                 appFatData.log_state = APP_MOUNT_DISK;
234             }
235             else
236             {
237                 /* Mount was successful. Unmount the disk, for testing. */
238
239                 appFatData.log_state = APP_SET_CURRENT_DRIVE;
240             }
241             break;
242
243         case APP_SET_CURRENT_DRIVE:
244             if(SYS_FS_CurrentDriveSet("/mnt/myDrive") == SYS_FS_RES_FAILURE)
245             {
246                 /* Error while setting current drive */
247                 appFatData.log_state = APP_ERROR;
248             }
249             else
250             {
251                 /* Open a file for reading. */
252                 appFatData.log_state = APP_IDLE;
253             }
254             break;
255
256         case APP_WRITE_MEASURE_FILE:
257             appFatData.fileMeasureHandle = SYS_FS_FileOpen(appFatData.fileName,
258                                         (SYS_FS_FILE_OPEN_APPEND_PLUS));
259             if(appFatData.fileMeasureHandle == SYS_FS_HANDLE_INVALID)
260             {
261                 /* Could not open the file. Error out*/
262                 appFatData.log_state = APP_ERROR;
263             }
264             else
265             {
266                 /* Create a directory. */
267                 appFatData.log_state = APP_WRITE_TO_MEASURE_FILE;
268             }
269             break;
270
271         case APP_WRITE_TO_MEASURE_FILE:
272             /* If read was success, try writing to the new file */
273             if(SYS_FS_FileStringPut(appFatData.fileMeasureHandle, appFatData.data) ==
274 -1)
275             {
276                 /* Write was not successful. Close the file
277                  * and error out.*/
278                 SYS_FS_FileClose(appFatData.fileMeasureHandle);
279                 appFatData.log_state = APP_ERROR;
280             }
281             else
282             {
283                 appFatData.log_state = APP_CLOSE_FILE;
284             }
285             break;
286
287         case APP_CLOSE_FILE:
288             /* Close both files */
289             SYS_FS_FileClose(appFatData.fileMeasureHandle);

```



```

354         // Prepare file name
355         sprintf(appFatData.fileName, "LOG_GNSS.txt");
356         /* Next log_state : write to file */
357         appFatData.log_state = APP_WRITE_MEASURE_FILE;
358         /* Write the buffer */
359         getFifoToLastReturn(&usartFifoRx, fifoBuffer);
360
361         sprintf(appFatData.data, "%s", fifoBuffer);
362
363         //sprintf(appFatData.data, "%s", );
364         /* Compute the number of bytes to send */
365         appFatData.nBytesToWrite = strlen(appFatData.data);
366     }
367 }
368
369 void sd_CFG_Write (uint32_t tLogGNSS_ms, uint32_t tLogIMU_ms, uint8_t ledState,
370 uint32_t tInactiveP, bool skipMount)
371 {
372     /* If sd Card available */
373     if((appFatData.cfg_state == APP_CFG_IDLE) || (appFatData.cfg_state ==
374 APP_CFG_OPEN_READ_CONFIG_FILE))
375     {
376         /* Close the file */
377         SYS_FS_FileClose(appFatData.fileCfgHandle);
378
379         if(skipMount == false)
380             /* Next config : mount disk */
381             appFatData.cfg_state = APP_CFG_MOUNT_DISK;
382         else if(skipMount == true)
383             /* Next config : write to file */
384             appFatData.cfg_state = APP_CFG_OPEN_WRITE_CONFIG_FILE;
385
386         /* Write the buffer */
387         sprintf(appFatData.cfg_data, "$LOG INTERVAL GNSS [ms] : %u\r\n$LOG INTERVAL
388 IMU [ms] : %u\r\n$LED ENABLE [1/0] : %u\r\n$INACTIVE PERIOD [s] : %u\r\n",
389             tLogGNSS_ms, tLogIMU_ms, ledState, tInactiveP);
390         /* Compute the number of bytes to send */
391         appFatData.nBytesToWrite = strlen(appFatData.cfg_data);
392     }
393 }
394
395 APP_FAT_LOG_STATES sd_logGetState( void )
396 {
397     return appFatData.log_state;
398 }
399
400 void sd_logSetState( APP_FAT_LOG_STATES newState )
401 {
402     appFatData.log_state = newState;
403 }
404
405 // CONFIG FUNCTIONS
406
407 APP_FAT_CONFIG_STATES sd_cfgGetState( void )
408 {
409     return appFatData.cfg_state;
410 }
411 void sd_cfgSetState( APP_FAT_CONFIG_STATES newState )
412 {
413     appFatData.cfg_state = newState;
414 }
415
416 char* sd_cfgGetCfgBuffer( void )
417 {
418     return appFatData.cfg_data;
419 }
420
421 void sd_fat_cfg_init(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
422 uint32_t *tInactivePeriod)
423 {
424     // Config parser error

```

```

423     uint8_t parseError = 0;
424     unsigned long tGnssLocal = 0;
425     unsigned long tImuLocal = 0;
426     unsigned long tInactive = 0;
427     bool ledStateLocal = 0;
428
429     //appFatData.nBytesRead = 0;
430     //appFatData.nBytesToWrite = 0;
431
432     //appFatData.log_state = APP_MOUNT_DISK;
433     //appFatData.cfg_state = APP_CFG_MOUNT_DISK;
434
435     // Read config routine, until error or success
436     sd_fat_config_task(true);
437
438     // If read config routine was a success
439     if(sd_cfgGetState() == APP_CFG_IDLE)
440         // Parse config buffer to get parameters
441         parseError = parseConfig(&tGnssLocal, &tImuLocal, &ledStateLocal, &tInactive);
442     // If the parsing failed or the read config routine failed
443     if((parseError > 0) || (sd_cfgGetState() == APP_CFG_ERROR))
444     {
445         // Set default system parameters
446         *tGnss = T_INTERVAL_GNSS_DEFAULT;
447         *tImu = T_INTERVAL_IMU_DEFAULT;
448         *ledState = LED_STATE_DEFAULT;
449         *tInactivePeriod = T_INACTIVE_PERIOD_DEFAULT;
450         appStateSet(APP_STATE_LOGGING);
451         // Start measure timer
452         DRV_TMR1_Start();
453     }
454     else if ((sd_cfgGetState() == APP_CFG_IDLE))
455     {
456         *tGnss = tGnssLocal;
457         *tImu = tImuLocal;
458         *ledState = ledStateLocal;
459         *tInactivePeriod = tInactive;
460         appStateSet(APP_STATE_LOGGING);
461         // Start measure timer
462         DRV_TMR1_Start();
463     }
464 }
465
466 static uint8_t parseConfig(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
467 unsigned long *tInactive)
468 {
469     char *ptBufferHead;
470     char *ptBufferTail;
471     char ptTrame[10];
472     uint8_t error = 0;
473
474     // Locate the head and tail of the first data
475     ptBufferHead = strstr(appFatData.cfg_data, " :");
476     ptBufferTail = strstr(appFatData.cfg_data, "\r\n");
477     // Check if the pointers are corrects
478     if((ptBufferHead != NULL) && (ptBufferTail != NULL) && (ptBufferHead < ptBufferTail)){
479         // Copy the data between the head and the tail in a sub-pointer
480         strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
481         // Convert the character to value
482         *tGnss = (uint32_t) atoi(ptTrame);
483     }
484     else
485         error++;
486
487     // Locate the head and tail of the first data
488     ptBufferHead = strstr(ptBufferTail, " :");
489     ptBufferTail = strstr(ptBufferHead, "\r\n");
490     // Check if the pointers are corrects
491     if((ptBufferHead != NULL) && (ptBufferTail != NULL) && (ptBufferHead < ptBufferTail)){
492         // Copy the data between the head and the tail in a sub-pointer
493         strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
494         // Convert the character to value
495         *tImu = (uint32_t) atoi(ptTrame);

```

```

495     }
496     else
497         error++;
498
499     // Locate the head and tail of the first data
500     ptBufferHead = strstr(ptBufferTail, " :");
501     ptBufferTail = strstr(ptBufferHead, "\r\n");
502     // Check if the pointers are corrects
503     if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
504         // Copy the data between the head and the tail in a sub-pointer
505         strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
506         // Convert the character to value
507         *ledState = (bool) atoi(ptTrame);
508     }
509     else
510         error++;
511
512     // Locate the head and tail of the first data
513     ptBufferHead = strstr(ptBufferTail, " :");
514     ptBufferTail = strstr(ptBufferHead, "\r\n");
515     // Check if the pointers are corrects
516     if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
517         // Copy the data between the head and the tail in a sub-pointer
518         strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
519         // Convert the character to value
520         *tInactive = (uint32_t) atoi(ptTrame);
521     }
522     else
523         error++;
524
525     return error;
526 }
527
528
529 void sd_fat_readDisplayFile(const char * fileName)
530 {
531     const uint16_t READSIZE = 256;
532     uint32_t i = 0;
533     char stringRead[READSIZE];
534     unsigned long cntTimeaout = 0;
535
536     /* Close both files */
537     SYS_FS_FileClose(appFatData.fileMeasureHandle);
538     // Read config file
539     appFatData.fileCfgHandle = SYS_FS_FileOpen(fileName, (SYS_FS_FILE_OPEN_READ));
540
541     do{
542
543         SYS_FS_FileStringGet(appFatData.fileCfgHandle, stringRead, READSIZE);
544
545         do{
546             if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
547             {
548                 PLIB_USART_TransmitterByteSend(USART_ID_1, stringRead[i]);
549                 i++;
550             }
551             cntTimeaout++;
552         }while((i < strlen(stringRead))&&(cntTimeaout<TIME_OUT));
553
554         i = 0;
555         cntTimeaout = 0;
556
557         if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x", "X"))
558             break;
559
560     }while(!SYS_FS_FileEOF(appFatData.fileCfgHandle));
561
562     /* Close both files */
563     SYS_FS_FileClose(appFatData.fileMeasureHandle);
564 }
565
566 //bool sd_fat_readFile(const char * fileName, char readBuffer[])
567 //{

```

```
568 //     uint32_t fileSize = 0;
569 //     static bool fullyRead = false;
570 //     /* Close both files */
571 //     SYS_FS_FileClose(appFatData.fileMeasureHandle);
572 //     /* Read config file
573 //     appFatData.fileCfgHandle = SYS_FS_FileOpen(fileName, (SYS_FS_FILE_OPEN_READ));
574 //
575 //     fileSize = SYS_FS_FileSize(appFatData.fileCfgHandle);
576 //
577 //     if (fileSize <= sizeof(readBuffer))
578 //         SYS_FS_FileRead(appFatData.fileCfgHandle, readBuffer, fileSize);
579 //     else{
580 //
581 //     }
582 //     /* Close both files */
583 //     SYS_FS_FileClose(appFatData.fileMeasureHandle);
584 //}
585
586 /* ****
587 End of File
588 */
589
```

```
1  ****MPLAB Harmony Application Header File
2
3  Company:
4      ETML-ES
5
6  File Name:
7      MC32_sdFatGest.h
8
9  ****
10 //DOM-IGNORE-BEGIN
11
12 //DOM-IGNORE-END
13
14 #ifndef _SD_FAT_GEST_H
15 #define _SD_FAT_GEST_H
16
17 // ****
18 // **** Section: Included Files
19 // ****
20 // ****
21 // ****
22 // **** Section: Type Definitions
23 // ****
24 // ****
25
26 #include "app.h"
27 #include "GNSS/minmea.h"
28 #include "uart_FIFO.h"
29 // ****
30 // ****
31 // **** Section: Type Definitions
32 // ****
33 // ****
34
35 #ifdef DRV_SDHC_USE_DMA
36 #define DATA_BUFFER_ALIGN           __attribute__((coherent, aligned(32)))
37 #else
38 #define DATA_BUFFER_ALIGN           __attribute__((aligned(32)))
39 #endif
40
41 // ****
42 /* Application States
43
44     Summary:
45         Application states enumeration
46
47     Description:
48         This enumeration defines the valid application states. These states
49         determine the behavior of the application at various times.
50 */
51
52 typedef enum
53 {
54     /* Application's state machine's initial state. */
55     /* The app mounts the disk */
56     APP_MOUNT_DISK = 0,
57
58     /* Set the current drive */
59     APP_SET_CURRENT_DRIVE,
60
61     /* The app opens the file to read */
62     APP_WRITE_MEASURE_FILE,
63
64     /* The app reads from a file and writes to another file */
65     APP_WRITE_TO_MEASURE_FILE,
66
67     /* The app closes the file*/
68     APP_CLOSE_FILE,
69
70     /* The app closes the file and idles */
71     APP_IDLE,
72
73     /* An app error has occurred */
74 }
```

```

74     APP_ERROR,
75
76     /* Unmount disk */
77     APP_UNMOUNT_DISK
78
79 } APP_FAT_LOG_STATES;
80
81 typedef enum
82 {
83     /* Application's state machine's initial state. */
84     /* The app mounts the disk */
85     APP_CFG_MOUNT_DISK = 0,
86
87     /* Set the current drive */
88     APP_CFG_SET_CURRENT_DRIVE,
89
90     /* The app opens the file to read */
91     APP_CFG_OPEN_READ_CONFIG_FILE,
92
93     /* The app opens the file to read */
94     APP_CFG_READ_CONFIG_FILE,
95
96     /* The app opens the file to write */
97     APP_CFG_OPEN_WRITE_CONFIG_FILE,
98
99     /* Execute write */
100    APP_CFG_WRITE_CONFIG_FILE,
101
102    /* The app closes the file*/
103    APP_CFG_CLOSE_FILE,
104
105    /* The app closes the file and idles */
106    APP_CFG_IDLE,
107
108    /* An app error has occurred */
109    APP_CFG_ERROR,
110
111    /* Couldnt find config file */
112    APP_CFG_NO_CFG_FILE,
113
114    /* Unmount disk */
115    APP_CFG_UNMOUNT_DISK
116
117 } APP_FAT_CONFIG_STATES;
118
119
120 // ****
121 /* Application Data
122
123 Summary:
124     Holds application data
125
126 Description:
127     This structure holds the application's data.
128
129 Remarks:
130     Application strings and buffers are be defined outside this structure.
131 */
132
133 typedef struct
134 {
135     /* SYS_FS File handle for 1st file */
136     SYS_FS_HANDLE     fileMeasureHandle;
137
138     /* SYS_FS File handle for 2nd file */
139     SYS_FS_HANDLE     fileCfgHandle;
140
141     /* Application's current state */
142     APP_FAT_LOG_STATES      log_state;
143     APP_FAT_CONFIG_STATES    cfg_state;
144
145     /* Application data buffer */
146     char              data[FIFO_RX_SIZE+2] DATA_BUFFER_ALIGN;

```

```

147     /* Application config file */
148     char             cfg_data[200] DATA_BUFFER_ALIGN;
149
150     /* Filename variable */
151     char             fileName[15] DATA_BUFFER_ALIGN;
152
153     uint32_t         nBytesWritten;
154
155     uint32_t         nBytesRead;
156
157     uint32_t         nBytesToWrite;
158 } APP_FAT_DATA;
159
160
161 // ****
162 // ****
163 // Section: Application Callback Routines
164 // ****
165 // ****
166 /* These routines are called by drivers when certain events occur.
167 */
168
169
170 // ****
171 // ****
172 // Section: Application Initialization and State Machine Functions
173 // ****
174 // ****
175
176 /*****
177
178 Function:
179     void APP_Tasks ( void )
180
181 Summary:
182     MPLAB Harmony Demo application tasks function
183
184 Description:
185     This routine is the Harmony Demo application's tasks function. It
186     defines the application's state machine and core logic.
187
188 Precondition:
189     The system and application initialization ("SYS_Initialize") should be
190     called before calling this.
191
192 Parameters:
193     None.
194
195 Returns:
196     None.
197
198 Example:
199     <code>
200     APP_Tasks();
201     </code>
202
203 Remarks:
204     This routine must be called from SYS_Tasks() routine.
205 */
206
207
208 void sd_fat_cfg_init(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
209     uint32_t *tInactivePeriod);
210
211 void sd_fat_config_task ( bool init );
212 void sd_CFG_Write (uint32_t tLogGNSS_ms, uint32_t tLogIMU_ms, uint8_t ledState,
213     uint32_t tInactiveP, bool skipMount);
214 APP_FAT_CONFIG_STATES sd_cfgGetState( void );
215 void sd_cfgSetState( APP_FAT_CONFIG_STATES newState );
216 char* sd_cfgGetCfgBuffer( void );
217
218 void sd_fat_logging_task ( void );
219 APP_FAT_LOG_STATES sd_logGetState( void );

```

```
218 void sd_logSetState( APP_FAT_LOG_STATES newState );
219
220 void sd_IMU_scheduleWrite (s_bno055_data * data);
221
222 void sd_GNSS_scheduleWrite (minmea_messages * pGnssData);
223
224 void sd_fat_readDisplayFile(const char * fileName);
225
226
227 #endif /* _APP_H */
228 /***** End of File ****/
229
230
231
232
```

```

1  /**
2  * @file bno055_support.c
3  *
4  */
5
6  *-----*
7  * Includes
8  *-----*/
9 #include "app.h"
10 #include "bno055.h"
11 #include "bno055_support.h"
12 #include "Mc32_I2cUtilCCS.h"
13 #include "driver/tmr/drv_tmr_static.h"
14
15 // Global variable
16 TIMER_DATA timeData;
17
18 #ifdef BNO055_API
19
20 s32 bno055_read_routine(s_bno055_data *data)
21 {
22     /* Variable used to return value of
23      * communication routine*/
24     s32 comres = BNO055_ERROR;
25
26     /* variable used to set the power mode of the sensor*/
27     //u8 power_mode = BNO055_INIT_VALUE;
28
29     /* For initializing the BNO sensor it is required to the operation mode
30      * of the sensor as NORMAL
31      * Normal mode can set from the register
32      * Page - page0
33      * register - 0x3E
34      * bit positions - 0 and 1*/
35     //power_mode = BNO055_POWER_MODE_NORMAL;
36
37     /* set the power mode as NORMAL*/
38     //comres += bno055_set_power_mode(power_mode);
39
40     /*-----*
41     ***** END INITIALIZATION *****/
42     *-----*/
43
44     /****** START READ RAW FUSION DATA *****/
45     /* For reading fusion data it is required to set the
46      * operation modes of the sensor
47      * operation mode can set from the register
48      * page - page0
49      * register - 0x3D
50      * bit - 0 to 3
51      * for sensor data read following operation mode have to set
52      * FUSION MODE
53      * 0x08 - BNO055_OPERATION_MODE_IMUPLUS
54      * 0x09 - BNO055_OPERATION_MODE_COMPASS
55      * 0x0A - BNO055_OPERATION_MODE_M4G
56      * 0x0B - BNO055_OPERATION_MODE_NDOF_FMC_OFF
57      * 0x0C - BNO055_OPERATION_MODE_NDOF
58      * based on the user need configure the operation mode*/
59     //comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
60
61     /* Raw Quaternion W, X, Y and Z data can read from the register
62      * page - page 0
63      * register - 0x20 to 0x27 */
64     comres += bno055_read_quaternion_wxyz(&data->quaternion);
65     /****** END READ RAW FUSION DATA *****/
66     /******START READ CONVERTED SENSOR DATA******/
67     /* API used to read mag data output as double - uT(micro Tesla)
68      * float functions also available in the BNO055 API */
69     comres += bno055_convert_double_mag_xyz_uT(&data->mag);
70     /* API used to read gyro data output as double - dps and rps
71      * float functions also available in the BNO055 API */
72     comres += bno055_convert_double_gyro_xyz_dps(&data->gyro);
73     /* API used to read Euler data output as double - degree and radians

```

```

74     * float functions also available in the BNO055 API */
75     comres += bno055_convert_double_euler_hpr_deg(&data->euler);
76     /* API used to read Linear acceleration data output as m/s2
77     * float functions also available in the BNO055 API */
78     comres += bno055_convert_double_linear_accel_xyz_msq(&data->linear_accel);
79     comres += bno055_convert_double_gravity_xyz_msq(&data->gravity);
80
81     /*-----*
82     ***** START DE-INITIALIZATION *****
83     -----*/
84
85     /* For de - initializing the BNO sensor it is required
86     * to the operation mode of the sensor as SUSPEND
87     * Suspend mode can set from the register
88     * Page - page0
89     * register - 0x3E
90     * bit positions - 0 and 1*/
91     //power_mode = BNO055_POWER_MODE_SUSPEND;
92
93     /* set the power mode as SUSPEND*/
94     //comres += bno055_set_power_mode(power_mode);
95
96     /* Flag measure ready */
97     data->flagMeasReady = true;
98
99     /*-----*
100    ***** END DE-INITIALIZATION *****
101   -----*/
102   return (comres+1);
103 }
104
105 /*-----*
106  * The following API is used to map the I2C bus read, write, delay and
107  * device address with global structure bno055_t
108  *-----*/
109
110 /*-----*
111  * By using bno055 the following structure parameter can be accessed
112  * Bus write function pointer: BNO055_WR_FUNC_PTR
113  * Bus read function pointer: BNO055_RD_FUNC_PTR
114  * Delay function pointer: delay_msec
115  * I2C address: dev_addr
116  *-----*/
117 s8 I2C_routine(void)
118 {
119     bno055.bus_write = BNO055_I2C_bus_write;
120     bno055.bus_read = BNO055_I2C_bus_read;
121     bno055.delay_msec = BNO055_delay_msek;
122     bno055.dev_addr = BNO055_I2C_ADDR1;
123     return BNO055_INIT_VALUE;
124 }
125
126 /****** I2C buffer length*****/
127
128 #define I2C_BUFFER_LEN 8
129 #define I2C0 5
130
131 /*-----*
132  *
133  * This is a sample code for read and write the data by using I2C
134  * Use either I2C based on your need
135  * The device address defined in the bno055.h file
136  *
137  *-----*/
138
139 /* \Brief: The API is used as I2C bus write
140 * \Return : Status of the I2C write
141 * \param dev_addr : The device address of the sensor
142 * \param reg_addr : Address of the first register,
143 *   will data is going to be written
144 * \param reg_data : It is a value hold in the array,
145 *   will be used for write the value into the register
146 * \param cnt : The no of byte of data to be write

```

```

147  */
148 s8 BNO055_I2C_bus_write(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
149 {
150     s8 BNO055_iERROR = BNO055_INIT_VALUE;
151     u8 array[I2C_BUFFER_LEN];
152     u8 stringpos = BNO055_INIT_VALUE;
153
154     array[BNO055_INIT_VALUE] = reg_addr;
155
156     i2c_start();
157     BNO055_iERROR = i2c_write(dev_addr<<1);
158
159     for (stringpos = BNO055_INIT_VALUE; stringpos < (cnt+
160         BNO055_I2C_BUS_WRITE_ARRAY_INDEX); stringpos++)
161     {
162         BNO055_iERROR = i2c_write(array[stringpos]);
163         array[stringpos + BNO055_I2C_BUS_WRITE_ARRAY_INDEX] = *(reg_data + stringpos);
164     }
165
166     i2c_stop();
167
168 /*
169  * Please take the below APIs as your reference for
170  * write the data using I2C communication
171  * "BNO055_iERROR = I2C_WRITE_STRING(DEV_ADDR, ARRAY, CNT+1)"
172  * add your I2C write APIs here
173  * BNO055_iERROR is an return value of I2C read API
174  * Please select your valid return value
175  * In the driver BNO055_SUCCESS defined as 0
176  * and FAILURE defined as -1
177  * Note :
178  * This is a full duplex operation,
179  * The first read data is discarded, for that extra write operation
180  * have to be initiated. For that cnt+1 operation done
181  * in the I2C write string function
182  * For more information please refer data sheet SPI communication:
183  */
184
185 /*if(BNO055_iERROR)
186     BNO055_iERROR = -1;
187 else
188     BNO055_iERROR = 0;
189
190     return (s8)(BNO055_iERROR);*/
191 // Error comm return
192
193 if(BNO055_iERROR-1 != 0)
194     BNO055_iERROR = -1;
195 else
196     BNO055_iERROR = 0;
197
198     return (s8)(BNO055_iERROR);
199 }
200
201 /* \Brief: The API is used as I2C bus read
202  * \Return : Status of the I2C read
203  * \param dev_addr : The device address of the sensor
204  * \param reg_addr : Address of the first register,
205  * will data is going to be read
206  * \param reg_data : This data read from the sensor,
207  * which is hold in an array
208  * \param cnt : The no of byte of data to be read
209 */
210 s8 BNO055_I2C_bus_read(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt)
211 {
212     s8 BNO055_iERROR = BNO055_INIT_VALUE;
213     u8 array[I2C_BUFFER_LEN] = { BNO055_INIT_VALUE };
214     u8 stringpos = BNO055_INIT_VALUE;
215
216     array[BNO055_INIT_VALUE] = reg_addr;
217
218     i2c_start();

```

```

219     // Write asked register
220     BNO055_iERROR = i2c_write(dev_addr<<1);
221     BNO055_iERROR = i2c_write(reg_addr);
222     // Send read address
223     i2c_reStart();
224     dev_addr = (dev_addr<<1) | 0b00000001;
225     BNO055_iERROR = i2c_write(dev_addr);
226
227     /* Please take the below API as your reference
228      * for read the data using I2C communication
229      * add your I2C read API here.
230      * "BNO055_iERROR = I2C_WRITE_READ_STRING(DEV_ADDR,
231      * ARRAY, ARRAY, 1, CNT)"
232      * BNO055_iERROR is an return value of SPI write API
233      * Please select your valid return value
234      * In the driver BNO055_SUCCESS defined as 0
235      * and FAILURE defined as -1
236      */
237     for (stringpos = BNO055_INIT_VALUE; stringpos < cnt; stringpos++)
238     {
239
240         if(((stringpos+1) < cnt)&&(cnt > BNO055_I2C_BUS_WRITE_ARRAY_INDEX))
241             array[stringpos] = i2c_read(1);
242         else
243             array[stringpos] = i2c_read(0);
244
245         *(reg_data + stringpos) = array[stringpos];
246
247     }
248
249     i2c_stop();
250
251     // Error comm return
252     if(BNO055_iERROR!=0)
253         BNO055_iERROR = -1;
254     else
255         BNO055_iERROR = 0;
256
257     return (s8)(BNO055_iERROR);
258 }
259
260 /* Brief : The delay routine
261  * \param : delay in ms
262  */
263 void BNO055_delay_msek(u32 msek)
264 {
265     /*Delay routine*/
266     DRV_TMR0_Stop();
267     DRV_TMR0_CounterClear();
268     timeData.delayCnt = 0;
269     DRV_TMR0_Start();
270     while (timeData.delayCnt < msek)
271     {
272         DRV_TMR0_Stop();
273     }
274
275 #endif
276
277
278 s32 bno055_init_readout(void)
279 {
280     /* Variable used to return value of
281      * communication routine*/
282     s32 comres = BNO055_ERROR;
283
284     /* variable used to set the power mode of the sensor*/
285     u8 power_mode = BNO055_INIT_VALUE;
286
287
288     /* variable used to read the accel xyz data */
289     struct bno055_accel_t accel_xyz;
290
291     /*****read raw mag data*****/

```

```

292     /* structure used to read the mag xyz data */
293     struct bno055_mag_t mag_xyz;
294
295     /*****read raw gyro data*****/
296     /* structure used to read the gyro xyz data */
297     struct bno055_gyro_t gyro_xyz;
298
299     /*****read raw Euler data*****/
300     /* structure used to read the euler hrp data */
301     struct bno055_euler_t euler_hrp;
302
303     /*****read raw quaternion data*****/
304     /* structure used to read the quaternion wxyz data */
305     struct bno055_quaternion_t quaternion_wxyz;
306
307     /*****read raw linear acceleration data*****/
308     /* structure used to read the linear accel xyz data */
309     struct bno055_linear_accel_t linear_acce_xyz;
310
311     /*****read raw gravity sensor data*****/
312     /* structure used to read the gravity xyz data */
313     struct bno055_gravity_t gravity_xyz;
314
315     /*****read accel converted data*****/
316     /* structure used to read the accel xyz data output as m/s2 or mg */
317     struct bno055_accel_double_t d_accel_xyz;
318
319     /*****read mag converted data*****/
320     /* structure used to read the mag xyz data output as uT*/
321     struct bno055_mag_double_t d_mag_xyz;
322
323     /*****read gyro converted data*****/
324     /* structure used to read the gyro xyz data output as dps or rps */
325     struct bno055_gyro_double_t d_gyro_xyz;
326
327     /*****read euler converted data*****/
328     /* variable used to read the euler h data output
329      * as degree or radians*/
330     double d_euler_data_h = BNO055_INIT_VALUE;
331     /* variable used to read the euler r data output
332      * as degree or radians*/
333     double d_euler_data_r = BNO055_INIT_VALUE;
334     /* variable used to read the euler p data output
335      * as degree or radians*/
336     double d_euler_data_p = BNO055_INIT_VALUE;
337     /* structure used to read the euler hrp data output
338      * as as degree or radians */
339     struct bno055_euler_double_t d_euler_hpr;
340
341     /*****read linear acceleration converted data*****/
342     /* structure used to read the linear accel xyz data output as m/s2*/
343     struct bno055_linear_accel_double_t d_linear_accel_xyz;
344
345     /*****Gravity converted data*****/
346     /* structure used to read the gravity xyz data output as m/s2*/
347     struct bno055_gravity_double_t d_gravity_xyz;
348
349     /*****
350      ***** START INITIALIZATION *****
351     *****/
352 #ifdef BNO055_API
353
354     /* Based on the user need configure I2C interface.
355      * It is example code to explain how to use the bno055 API*/
356     I2C_routine();
357 #endif
358
359     /*****
360      * This API used to assign the value/reference of
361      * the following parameters
362      * I2C address
363      * Bus Write
364      * Bus read

```

```

365      * Chip id
366      * Page id
367      * Accel revision id
368      * Mag revision id
369      * Gyro revision id
370      * Boot loader revision id
371      * Software revision id
372      *-----*/
373 comres = bno055_init(&bno055);
374
375 /* For initializing the BNO sensor it is required to the operation mode
376 * of the sensor as NORMAL
377 * Normal mode can set from the register
378 * Page - page0
379 * register - 0x3E
380 * bit positions - 0 and 1*/
381 power_mode = BNO055_POWER_MODE_NORMAL;
382
383 /* set the power mode as NORMAL*/
384 comres += bno055_set_power_mode(power_mode);
385
386 /*-----*
387 ***** END INITIALIZATION *****
388 *-----*/
389
390 /***** START READ RAW SENSOR DATA*****/
391
392 /* Using BNO055 sensor we can read the following sensor data and
393 * virtual sensor data
394 * Sensor data:
395 * Accel
396 * Mag
397 * Gyro
398 * Virtual sensor data
399 * Euler
400 * Quaternion
401 * Linear acceleration
402 * Gravity sensor */
403
404 /* For reading sensor raw data it is required to set the
405 * operation modes of the sensor
406 * operation mode can set from the register
407 * page - page0
408 * register - 0x3D
409 * bit - 0 to 3
410 * for sensor data read following operation mode have to set
411 * SENSOR MODE
412 * 0x01 - BNO055_OPERATION_MODE_ACONLY
413 * 0x02 - BNO055_OPERATION_MODE_MAGONLY
414 * 0x03 - BNO055_OPERATION_MODE_GYRONLY
415 * 0x04 - BNO055_OPERATION_MODE_ACCMAG
416 * 0x05 - BNO055_OPERATION_MODE_ACCGYRO
417 * 0x06 - BNO055_OPERATION_MODE_MAGGYRO
418 * 0x07 - BNO055_OPERATION_MODE_AMG
419 * based on the user need configure the operation mode*/
420 comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_AMG);
421
422 /* Raw accel X, Y and Z data can read from the register
423 * page - page 0
424 * register - 0x08 to 0x0D*/
425 comres += bno055_read_accel_xyz(&accel_xyz);
426
427 /* Raw mag X, Y and Z data can read from the register
428 * page - page 0
429 * register - 0x0E to 0x13*/
430 comres += bno055_read_mag_xyz(&mag_xyz);
431
432 /* Raw gyro X, Y and Z data can read from the register
433 * page - page 0
434 * register - 0x14 to 0x19*/
435 comres += bno055_read_gyro_xyz(&gyro_xyz);
436
437 /***** END READ RAW SENSOR DATA*****/

```

```

438
439     **** START READ RAW FUSION DATA ****
440     * For reading fusion data it is required to set the
441     * operation modes of the sensor
442     * operation mode can set from the register
443     * page - page0
444     * register - 0x3D
445     * bit - 0 to 3
446     * for sensor data read following operation mode have to set
447     * FUSION MODE
448     * 0x08 - BNO055_OPERATION_MODE_IMUPLUS
449     * 0x09 - BNO055_OPERATION_MODE_COMPASS
450     * 0x0A - BNO055_OPERATION_MODE_M4G
451     * 0x0B - BNO055_OPERATION_MODE_NDOF_FMC_OFF
452     * 0x0C - BNO055_OPERATION_MODE_NDOF
453     * based on the user need configure the operation mode*/
454 comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
455
456     /* Raw Euler H, R and P data can read from the register
457     * page - page 0
458     * register - 0x1A to 0x1E */
459 //comres += bno055_read_euler_h(&euler_data_h);
460 //comres += bno055_read_euler_r(&euler_data_r);
461 //comres += bno055_read_euler_p(&euler_data_p);
462 comres += bno055_read_euler_hrp(&euler_hrp);
463
464     /* Raw Quaternion W, X, Y and Z data can read from the register
465     * page - page 0
466     * register - 0x20 to 0x27 */
467 //comres += bno055_read_quaternion_w(&quaternion_data_w);
468 //comres += bno055_read_quaternion_x(&quaternion_data_x);
469 //comres += bno055_read_quaternion_y(&quaternion_data_y);
470 //comres += bno055_read_quaternion_z(&quaternion_data_z);
471 comres += bno055_read_quaternion_wxyz(&quaternion_wxyz);
472
473     /* Raw Linear accel X, Y and Z data can read from the register
474     * page - page 0
475     * register - 0x28 to 0x2D */
476 //comres += bno055_read_linear_accel_x(&linear_accel_data_x);
477 //comres += bno055_read_linear_accel_y(&linear_accel_data_y);
478 //comres += bno055_read_linear_accel_z(&linear_accel_data_z);
479 comres += bno055_read_linear_accel_xyz(&linear_acce_xyz);
480
481     /* Raw Gravity sensor X, Y and Z data can read from the register
482     * page - page 0
483     * register - 0x2E to 0x33 */
484 //comres += bno055_read_gravity_x(&gravity_data_x);
485 //comres += bno055_read_gravity_y(&gravity_data_y);
486 //comres += bno055_read_gravity_z(&gravity_data_z);
487 comres += bno055_read_gravity_xyz(&gravity_xyz);
488
489     **** END READ RAW FUSION DATA ****
490     ****START READ CONVERTED SENSOR DATA****/
491
492     /* API used to read accel data output as double - m/s2 and mg
493     * float functions also available in the BNO055 API */
494 //comres += bno055_convert_double_accel_x_msq(&d_accel_datax);
495 //comres += bno055_convert_double_accel_x_mg(&d_accel_datax);
496 //comres += bno055_convert_double_accel_y_msq(&d_accel_datay);
497 //comres += bno055_convert_double_accel_y_mg(&d_accel_datay);
498 //comres += bno055_convert_double_accel_z_msq(&d_accel_dataz);
499 //comres += bno055_convert_double_accel_z_mg(&d_accel_dataz);
500 comres += bno055_convert_double_accel_xyz_msq(&d_accel_xyz);
501 comres += bno055_convert_double_accel_xyz_mg(&d_accel_xyz);
502
503     /* API used to read mag data output as double - uT(micro Tesla)
504     * float functions also available in the BNO055 API */
505 //comres += bno055_convert_double_mag_x_uT(&d_mag_datax);
506 //comres += bno055_convert_double_mag_y_uT(&d_mag_datay);
507 //comres += bno055_convert_double_mag_z_uT(&d_mag_dataz);
508 comres += bno055_convert_double_mag_xyz_uT(&d_mag_xyz);
509
510     /* API used to read gyro data output as double - dps and rps

```

```

511     * float functions also available in the BNO055 API */
512     //comres += bno055_convert_double_gyro_x_dps(&d_gyro_datax);
513     //comres += bno055_convert_double_gyro_y_dps(&d_gyro_datay);
514     //comres += bno055_convert_double_gyro_z_dps(&d_gyro_dataz);
515     //comres += bno055_convert_double_gyro_x_rps(&d_gyro_datax);
516     //comres += bno055_convert_double_gyro_y_rps(&d_gyro_datay);
517     //comres += bno055_convert_double_gyro_z_rps(&d_gyro_dataz);
518     comres += bno055_convert_double_gyro_xyz_dps(&d_gyro_xyz);
519     //comres += bno055_convert_double_gyro_xyz_rps(&d_gyro_xyz);
520
521     /* API used to read Euler data output as double - degree and radians
522      * float functions also available in the BNO055 API */
523     comres += bno055_convert_double_euler_h_deg(&d_euler_data_h);
524     comres += bno055_convert_double_euler_r_deg(&d_euler_data_r);
525     comres += bno055_convert_double_euler_p_deg(&d_euler_data_p);
526     //comres += bno055_convert_double_euler_h_rad(&d_euler_data_h);
527     //comres += bno055_convert_double_euler_r_rad(&d_euler_data_r);
528     //comres += bno055_convert_double_euler_p_rad(&d_euler_data_p);
529     comres += bno055_convert_double_euler_hpr_deg(&d_euler_hpr);
530     //comres += bno055_convert_double_euler_hpr_rad(&d_euler_hpr);
531
532     /* API used to read Linear acceleration data output as m/s2
533      * float functions also available in the BNO055 API */
534     //comres += bno055_convert_double_linear_accel_x_msq(&d_linear_accel_datax);
535     //comres += bno055_convert_double_linear_accel_y_msq(&d_linear_accel_datay);
536     //comres += bno055_convert_double_linear_accel_z_msq(&d_linear_accel_dataz);
537     comres += bno055_convert_double_linear_accel_xyz_msq(&d_linear_accel_xyz);
538
539     /* API used to read Gravity sensor data output as m/s2
540      * float functions also available in the BNO055 API */
541     //comres += bno055_convert_gravity_double_x_msq(&d_gravity_data_x);
542     //comres += bno055_convert_gravity_double_y_msq(&d_gravity_data_y);
543     //comres += bno055_convert_gravity_double_z_msq(&d_gravity_data_z);
544     comres += bno055_convert_double_gravity_xyz_msq(&d_gravity_xyz);
545
546     /*-----*
547     ***** START DE-INITIALIZATION *****
548     *-----*/
549
550     /* For de - initializing the BNO sensor it is required
551      * to the operation mode of the sensor as SUSPEND
552      * Suspend mode can set from the register
553      * Page - page0
554      * register - 0x3E
555      * bit positions - 0 and 1*/
556     //power_mode = BNO055_POWER_MODE_SUSPEND;
557
558     /* set the power mode as SUSPEND*/
559     //comres += bno055_set_power_mode(power_mode);
560     comres += bno055_set_operation_mode(BNO055_OPERATION_MODE_NDOF);
561     /*-----*
562     ***** END DE-INITIALIZATION *****
563     *-----*/
564
565     return comres;
566 }

```

```

1  /**
2  * @file bno055_support.h
3  *
4  */
5
6  *-----*
7  * Includes
8  *-----*/
9 #include "bno055.h"
10
11 #define BNO055_API
12
13 #define FLAG_MEAS_ON    1
14 #define FLAG_MEAS_OFF   0
15 *-----*
16 * The following APIs are used for reading and writing of
17 * sensor data using I2C communication
18 *-----*/
19 #ifdef BNO055_API
20 #define BNO055_I2C_BUS_WRITE_ARRAY_INDEX ((u8)1)
21
22 /* \Brief: The API is used as I2C bus read
23 * \Return : Status of the I2C read
24 * \param dev_addr : The device address of the sensor
25 * \param reg_addr : Address of the first register,
26 * will data is going to be read
27 * \param reg_data : This data read from the sensor,
28 * which is hold in an array
29 * \param cnt : The no of byte of data to be read
30 */
31 s8 BNO055_I2C_bus_read(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt);
32
33 /* \Brief: The API is used as SPI bus write
34 * \Return : Status of the SPI write
35 * \param dev_addr : The device address of the sensor
36 * \param reg_addr : Address of the first register,
37 * will data is going to be written
38 * \param reg_data : It is a value hold in the array,
39 * will be used for write the value into the register
40 * \param cnt : The no of byte of data to be write
41 */
42 s8 BNO055_I2C_bus_write(u8 dev_addr, u8 reg_addr, u8 *reg_data, u8 cnt);
43
44 /*
45 * \Brief: I2C init routine
46 */
47 s8 I2C_routine(void);
48
49 /* Brief : The delay routine
50 * \param : delay in ms
51 */
52 void BNO055_delay_msek(u32 msek);
53
54 #endif
55
56 /*****End of I2C APIs declarations*****/
57
58 /* This API is an example for reading sensor data
59 * \param: None
60 * \return: communication result
61 */
62 s32 bno055_init_readout(void);
63
64 s32 bno055_read_routine(s_bno055_data *data);
65
66 *-----*
67 * struct bno055_t parameters can be accessed by using BNO055
68 * BNO055_t having the following parameters
69 * Bus write function pointer: BNO055_WR_FUNC_PTR
70 * Bus read function pointer: BNO055_RD_FUNC_PTR
71 * Burst read function pointer: BNO055_BRD_FUNC_PTR
72 * Delay function pointer: delay_msec
73 * I2C address: dev_addr

```

```
74     * Chip id of the sensor: chip_id
75     *-----*/
76 struct bno055_t bno055;
77
```

```
1  /* **** */
2  /** Descriptive File Name
3
4  @Company
5      ETML-ES
6
7  @File Name
8      mc32_serComm.c
9
10 */
11 /* **** */
12
13 /* **** */
14 /* Section: Included Files */
15 */
16 /* **** */
17 /* **** */
18 #include "Mc32_serComm.h"
19 #include <stdio.h>
20
21 /* This section lists the other files that are included in this file.
22 */
23
24 /* TODO: Include other files here if needed. */
25
26
27 /* **** */
28 /* **** */
29 /* Section: File Scope or Global Data */
30 */
31 /* **** */
32
33 /* A brief description of a section can be given directly below the section
34     banner.
35 */
36
37 /* **** */
38 /** Descriptive Data Item Name
39
40 @Summary
41     Brief one-line summary of the data item.
42
43 @Description
44     Full description, explaining the purpose and usage of data item.
45     <p>
46         Additional description in consecutive paragraphs separated by HTML
47         paragraph breaks, as necessary.
48     <p>
49         Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.
50
51 @Remarks
52     Any additional remarks
53 */
54
55
56 /* **** */
57 /* **** */
58 // Section: Local Functions
59 */
60 /* **** */
61
62
63
64 /* **** */
65 /* **** */
66 // Section: Interface Functions
67 */
68 /* **** */
69
70 /* A brief description of a section can be given directly below the section
71     banner.
72 */
73
```

```

74 // ****
75
76
77 void serDisplayValues ( s_bno055_data *bno055_data )
78 {
79     char sendBuffer[66] = {0};
80     uint8_t i = 0;
81     static uint32_t ctnTimeout = 0;
82
83     /* Preapare Gravity string */
84     sprintf(sendBuffer, "DT: %d0 ms\tGravity : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r", (bno055_data->d_time), bno055_data->gravity.x, bno055_data->gravity.y, bno055_data->gravity.z);
85     /* Transmit Gravity string */
86     do{
87         if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
88         {
89             PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
90             i++;
91         }
92         ctnTimeout++;
93     }while((sendBuffer[i-1] != '\r') &&(ctnTimeout<TIME_OUT));
94     i = 0;
95
96     /* Preapare gyroscope string */
97     sprintf(sendBuffer, "Gyro : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r", bno055_data->gyro.x, bno055_data->gyro.y, bno055_data->gyro.z);
98     /* Transmit Gravity string */
99     do{
100        if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
101        {
102            PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
103            i++;
104        }
105        ctnTimeout++;
106    }while((sendBuffer[i-1] != '\r') &&(ctnTimeout<TIME_OUT));
107    i = 0;
108
109    /* Preapare magnitude string */
110    sprintf(sendBuffer, "Mag : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r", bno055_data->mag.x, bno055_data->mag.y, bno055_data->mag.z);
111    /* Transmit Gravity string */
112    do{
113        if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
114        {
115            PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
116            i++;
117        }
118        ctnTimeout++;
119    }while((sendBuffer[i-1] != '\r') &&(ctnTimeout<TIME_OUT));
120    i = 0;
121
122    /* Preapare linear acceleration string */
123    sprintf(sendBuffer, "Accel : X = %04.03lf\tY = %04.03lf\tZ = %04.03lf \n\r", bno055_data->linear_accel.x, bno055_data->linear_accel.y, bno055_data->linear_accel.z);
124    /* Transmit Gravity string */
125    do{
126        if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
127        {
128            PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
129            i++;
130        }
131        ctnTimeout++;
132    }while((sendBuffer[i-1] != '\r') &&(ctnTimeout<TIME_OUT));
133    i = 0;
134
135    /* Preapare euler string */
136    sprintf(sendBuffer, "Euler : H = %04.03lf\tP = %04.03lf\tR = %04.03lf \n\r", bno055_data->euler.h, bno055_data->euler.p, bno055_data->euler.r);
137    /* Transmit Gravity string */
138    do{
139        if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))

```

```

140         {
141             PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
142             i++;
143         }
144         ctnTimeout++;
145     }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
146     i = 0;
147
148     /* Preapare quaternion string */
149     sprintf(sendBuffer, "Quater. : W = %05d\tx = %05d\ty = %05d\tz = %05d \n\n\r",
150     bno055_data->quaternion.w, bno055_data->quaternion.x, bno055_data->quaternion.y,
151     bno055_data->quaternion.z);
152     /* Transmit Gravity string */
153     do{
154         if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
155         {
156             PLIB_USART_TransmitterByteSend(USART_ID_1, sendBuffer[i]);
157             i++;
158         }
159         ctnTimeout++;
160     }while((sendBuffer[i-1] != '\r')&&(ctnTimeout<TIME_OUT));
161     i = 0;
162 }
163 void serTransmitString ( USART_MODULE_ID usartId, const char * msg )
164 {
165     char bufferMsg[60] = {0};
166     static uint32_t i = 0;
167     static uint32_t ctnTimeout = 0;
168
169     strncpy(bufferMsg, msg, strlen(msg));
170
171     /* Transmit string */
172     do{
173         if(!PLIB_USART_TransmitterBufferIsFull(usartId))
174         {
175             PLIB_USART_TransmitterByteSend(usartId, bufferMsg[i]);
176             i++;
177         }
178         ctnTimeout++;
179     }while((bufferMsg[i-1] != '\0')&&(ctnTimeout<TIME_OUT));
180     i = 0;
181 }
182
183 void serTransmitbuffer ( USART_MODULE_ID usartId, char msg[], uint32_t lenght )
184 {
185     uint32_t i = 0;
186     uint32_t ctnTimeout = 0;
187
188     /* Transmit string */
189     do{
190         if(!PLIB_USART_TransmitterBufferIsFull(usartId))
191         {
192             PLIB_USART_TransmitterByteSend(usartId, msg[i]);
193             i++;
194         }
195         ctnTimeout++;
196     }while((i < lenght)&&(ctnTimeout<TIME_OUT));
197     i = 0;
198 }
199
200 bool pollSerialSingleCmd(USART_MODULE_ID usartID, const char * command1)
201 {
202     static char charRead[30] = {0};
203     static uint32_t readCnt = 0;
204
205     // Get command's characters
206     while((PLIB_USART_ReceiverDataIsAvailable(usartID))&&(readCnt < 30)){
207         charRead[readCnt] = PLIB_USART_ReceiverByteReceive(usartID);
208         readCnt++;
209     }
210     // Command

```

```

211     if(readCnt >= 30)
212     {
213         /* Reset read counter */
214         readCnt = 0;
215         /* Clear read buffer */
216         memset(charRead,0,strlen(charRead));
217     }
218     // Check occurence with commands
219     if(strstr(charRead, command1) != NULL) {
220         /* Reset read counter */
221         readCnt = 0;
222         /* Clear read buffer */
223         memset(charRead,0,strlen(charRead));
224         /* Command detected */
225         return true;
226     }
227     else{
228         return false;
229     }
230 }
231
232 bool pollSerialCmds(USART_MODULE_ID usartID, const char * command1, const char *
233 command2, const char * command3,
234 const char * command4)
235 {
236     static char charRead[CHAR_READ_BUFFER_SIZE] = {0};
237     static uint32_t readCnt = 0;
238
239     // Get command's characters
240     while((PLIB_USART_ReceiverDataIsAvailable(usartID))&&(readCnt <
241 CHAR_READ_BUFFER_SIZE)){
242         charRead[readCnt] = PLIB_USART_ReceiverByteReceive(usartID);
243         readCnt++;
244     }
245     // Command
246     if(readCnt >= CHAR_READ_BUFFER_SIZE)
247     {
248         /* Reset read counter */
249         readCnt = 0;
250         /* Clear read buffer */
251         memset(charRead,0,CHAR_READ_BUFFER_SIZE);
252     }
253     // Check occurence with commands
254     if((strstr(charRead, command1) != NULL) || (strstr(charRead, command2) != NULL)
255     || (strstr(charRead, command3) != NULL) || (strstr(charRead, command4) != NULL
256     )) {
257         /* Reset read counter */
258         readCnt = 0;
259         /* Clear read buffer */
260         memset(charRead,0,CHAR_READ_BUFFER_SIZE);
261         /* Command detected */
262         return true;
263     }
264     else{
265         return false;
266     }
267 }
268 /* **** End of File ****
269 */

```

```
1  /* **** */
2  /** Descriptive File Name
3
4  @Company
5      ETML-ES
6
7  @File Name
8      MC32_serComm.h
9
10 */
11 /* **** */
12
13 #ifndef _SER_COMM_H      /* Guard against multiple inclusion */
14 #define _SER_COMM_H
15
16
17 /* **** */
18 /* **** */
19 /* Section: Included Files
20 */
21 /* **** */
22
23 #include "app.h"
24
25
26 /* Provide C++ Compatibility */
27 #ifdef __cplusplus
28 extern "C" {
29 #endif
30
31
32 /* **** */
33 /* **** */
34 /* Section: Constants
35 */
36 /* **** */
37
38 /* A brief description of a section can be given directly below the section
39    banner.
40 */
41
42
43 /* **** */
44 /** Descriptive Constant Name
45
46     @Summary
47         Brief one-line summary of the constant.
48
49     @Description
50         Full description, explaining the purpose and usage of the constant.
51         <p>
52             Additional description in consecutive paragraphs separated by HTML
53             paragraph breaks, as necessary.
54         <p>
55             Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.
56
57     @Remarks
58         Any additional remarks
59
60 #define EXAMPLE_CONSTANT 0
61
62
63 // ****
64 // ****
65 // Section: Data Types
66 // ****
67 // ****
68
69 /* A brief description of a section can be given directly below the section
70    banner.
71 */
72
73
```

```
74 // ****
75 // ****
76 // Section: Interface Functions
77 // ****
78 // ****
79
80
81 void serDisplayValues ( s_bno055_data *bno055_data );
82
83 bool pollSerialCmds(USART_MODULE_ID usartID, const char * command1, const char *
84 command2, const char * command3,const char * command4);
85
86 bool pollSerialSingleCmd(USART_MODULE_ID usartID, const char * command1);
87
88 void serTransmitString ( USART_MODULE_ID usartId, const char * msg );
89 void serTransmitbuffer ( USART_MODULE_ID usartId, char msg[], uint32_t lenght );
90 /* Provide C++ Compatibility */
91 #ifdef __cplusplus
92 }
93 #endif
94
95 #endif /* _EXAMPLE_FILE_NAME_H */
96
97 /* ****
98 End of File
99 */
100
```

11.5 Code python du script visualisation CSV

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import os
4
5 if not os.path.exists('./Output'):
6     os.mkdir('Output')
7
8 if not os.path.exists('./Output/csv'):
9     os.mkdir('./Output/csv')
10
11 if os.path.exists('MESURES.csv'):
12
13     headerName = ["flag", "dt", "gravity x", "gravity y", "gravity z", "gyro x",
14     "gyro_y", "gyro_z", "mag_x", "mag_y", "mag_z", "accel_x", "accel_y", "accel_z",
15     "euler_h", "euler_p", "euler_r", "quater_w", "quater_x", "quater_y", "quater_z"]
16
17     df = pd.read_csv('MESURES.csv', sep=';', index_col=False, names=headerName,
18     lineterminator='\n')
19
20     df.head()
21     df.to_excel(r'./Output/Mesures.xlsx', index = False, header=True)
22
23     df_t = df.transpose()
24
25     flg = pd.DataFrame(df_t.loc['flag'].to_numpy().reshape(1,-1))
26
27     dt = pd.DataFrame(df_t.loc['dt'].to_numpy().reshape(1,-1))
28
29     gravity_x = pd.DataFrame(df_t.loc['gravity_x'].to_numpy().reshape(1,-1))
30     gravity_y = pd.DataFrame(df_t.loc['gravity_y'].to_numpy().reshape(1,-1))
31     gravity_z = pd.DataFrame(df_t.loc['gravity_z'].to_numpy().reshape(1,-1))
32
33     gyro_x = pd.DataFrame(df_t.loc['gyro_x'].to_numpy().reshape(1,-1))
34     gyro_y = pd.DataFrame(df_t.loc['gyro_y'].to_numpy().reshape(1,-1))
35     gyro_z = pd.DataFrame(df_t.loc['gyro_z'].to_numpy().reshape(1,-1))
36
37     mag_x = pd.DataFrame(df_t.loc['mag_x'].to_numpy().reshape(1,-1))
38     mag_y = pd.DataFrame(df_t.loc['mag_y'].to_numpy().reshape(1,-1))
39     mag_z = pd.DataFrame(df_t.loc['mag_z'].to_numpy().reshape(1,-1))
40
41     accel_x = pd.DataFrame(df_t.loc['accel_x'].to_numpy().reshape(1,-1))
42     accel_y = pd.DataFrame(df_t.loc['accel_y'].to_numpy().reshape(1,-1))
43     accel_z = pd.DataFrame(df_t.loc['accel_z'].to_numpy().reshape(1,-1))
44
45     euler_h = pd.DataFrame(df_t.loc['euler_h'].to_numpy().reshape(1,-1))
46     euler_p = pd.DataFrame(df_t.loc['euler_p'].to_numpy().reshape(1,-1))
47     euler_r = pd.DataFrame(df_t.loc['euler_r'].to_numpy().reshape(1,-1))
48
49     quater_w = pd.DataFrame(df_t.loc['quater_w'].to_numpy().reshape(1,-1))
50     quater_x = pd.DataFrame(df_t.loc['quater_x'].to_numpy().reshape(1,-1))
51     quater_y = pd.DataFrame(df_t.loc['quater_y'].to_numpy().reshape(1,-1))
52     quater_z = pd.DataFrame(df_t.loc['quater_z'].to_numpy().reshape(1,-1))
53
54     flg.to_csv(r'./Output/csv/flag.csv', sep=";", index=False, header=False,
55     line_terminator=None)
56     dt.to_csv(r'./Output/csv/dt.csv', sep=";", index=False, header=False,
57     line_terminator=None)
58
59     gravity_x.to_csv(r'./Output/csv/gravity_x.csv', sep=";", index=False, header=False,
60     line_terminator=None)
61     gravity_y.to_csv(r'./Output/csv/gravity_y.csv', sep=";", index=False, header=False,
62     line_terminator=None)
63     gravity_z.to_csv(r'./Output/csv/gravity_z.csv', sep=";", index=False, header=False,
64     line_terminator=None)
65
66     gyro_x.to_csv(r'./Output/csv/gyro_x.csv', sep=";", index=False, header=False,
67     line_terminator=None)
68     gyro_y.to_csv(r'./Output/csv/gyro_y.csv', sep=";", index=False, header=False,
69     line_terminator=None)

```

```

63     gyro_z.to_csv(r'./Output/csv/gyro_z.csv', sep=";", index=False, header=False,
64     line_terminator=None)
65     mag_x.to_csv(r'./Output/csv/mag_x.csv', sep=";", index=False, header=False,
66     line_terminator=None)
66     mag_y.to_csv(r'./Output/csv/mag_y.csv', sep=";", index=False, header=False,
67     line_terminator=None)
67     mag_z.to_csv(r'./Output/csv/mag_z.csv', sep=";", index=False, header=False,
68     line_terminator=None)
68
69     accel_x.to_csv(r'./Output/csv/accel_x.csv', sep=";", index=False, header=False,
69     line_terminator=None)
70     accel_y.to_csv(r'./Output/csv/accel_y.csv', sep=";", index=False, header=False,
70     line_terminator=None)
71     accel_z.to_csv(r'./Output/csv/accel_z.csv', sep=";", index=False, header=False,
71     line_terminator=None)
72
73     euler_h.to_csv(r'./Output/csv/euler_h.csv', sep=";", index=False, header=False,
73     line_terminator=None)
74     euler_p.to_csv(r'./Output/csv/euler_p.csv', sep=";", index=False, header=False,
74     line_terminator=None)
75     euler_r.to_csv(r'./Output/csv/euler_r.csv', sep=";", index=False, header=False,
75     line_terminator=None)
76
77     quater_w.to_csv(r'./Output/csv/quater_w.csv', sep=";", index=False, header=False,
77     line_terminator=None)
78     quater_x.to_csv(r'./Output/csv/quater_x.csv', sep=";", index=False, header=False,
78     line_terminator=None)
79     quater_y.to_csv(r'./Output/csv/quater_y.csv', sep=";", index=False, header=False,
79     line_terminator=None)
80     quater_z.to_csv(r'./Output/csv/quater_z.csv', sep=";", index=False, header=False,
80     line_terminator=None)
81
82
83     fig, ((grav, gyro, mag), (acc, euler, quater)) = plt.subplots(2, 3)
84     fig.suptitle('Mesures IMU')
85
86     #flag.grid()
87     #flag.plot(df['flag'])
88     #flag.set_ylabel(r'Flag mesure')
89
90
91     grav.grid()
92     grav.plot(df['gravity_x'], label="X")
93     grav.plot(df['gravity_y'], label="Y")
94     grav.plot(df['gravity_z'], label="Z")
95     grav.legend(loc=1, prop={'size': 7})
96     grav.set_ylabel(r'Gravity $[m/s^2]$')
97
98     gyro.grid()
99     gyro.plot(df['gyro_x'], label="X")
100    gyro.plot(df['gyro_y'], label="Y")
101    gyro.plot(df['gyro_z'], label="Z")
102    gyro.legend(loc=1, prop={'size': 7})
103    gyro.set_ylabel('Gyroscope $[^{\circ}/s]$')
104
105    mag.grid()
106    mag.plot(df['mag_x'], label="X")
107    mag.plot(df['mag_y'], label="Y")
108    mag.plot(df['mag_z'], label="Z")
109    mag.legend(loc=1, prop={'size': 7})
110    mag.set_ylabel('Magnitude $[uT]$')
111
112    acc.grid()
113    acc.plot(df['accel_x'], label="X")
114    acc.plot(df['accel_y'], label="Y")
115    acc.plot(df['accel_z'], label="Z")
116    acc.legend(loc=1, prop={'size': 7})
117    acc.set_ylabel('Accel. $[m/s]^2$')
118
119    euler.grid()
120    euler.plot(df['euler_h'], label="H")

```

```
121     euler.plot(df['euler_p'], label="P")
122     euler.plot(df['euler_r'], label="R")
123     euler.legend(loc=1, prop={'size': 7})
124     euler.set_ylabel('Euler')
125
126     quater.grid()
127     quater.plot(df['quater_w'], label="W")
128     quater.plot(df['quater_x'], label="X")
129     quater.plot(df['quater_y'], label="Y")
130     quater.plot(df['quater_z'], label="Z")
131     quater.legend(loc=1, prop={'size': 7})
132     quater.set_ylabel('Quaternion')
133
134
135     manager = plt.get_current_fig_manager()
136     manager.full_screen_toggle()
137     plt.savefig("./Output/MESURES_GRAPH.svg")
138     plt.savefig("./Output/MESURES_GRAPH.png", dpi=300)
139     plt.show()
140
141 else :
142     print("Pas de fichier MESURES.csv")
143
144
```

11.6 Code python de l'application BBX-Connect

```

1  from tkinter import *
2  from tkinter import ttk
3  from tkinter import filedialog
4  from tkinter import messagebox
5  from ttkthemes import ThemedStyle
6  import serial
7  import serial.tools.list_ports
8  import datetime
9  import threading
10 import multiprocessing
11 import os
12 import csv
13
14 #for printing debugging messages in console
15 dbg = 0
16
17 gRoot = Tk()
18 gRoot.config(bg="white")
19 gRoot.geometry("1080x640")
20 gRoot.title("Black Box Connect")
21
22 # sty = ttk.Style()
23 # sty.theme_use("clam")
24 sty = ThemedStyle(gRoot)
25 sty.set_theme('radiance')
26
27
28 gRoot.columnconfigure(0,weight=1)
29 gRoot.rowconfigure(0,weight=1)
30 #sty.configure("gframe.TFrame",background="white")
31 gFrame = ttk.LabelFrame(gRoot, text="Connection Setting", padding=10, style='TFrame')
32 gFrame.grid(column=1, row=1, sticky=(W,E))
33
34 # Frame for commands
35
36 gFrameCmd = ttk.LabelFrame(gRoot, text="List of commands", padding=10, width=130 , style='TFrame')
37 gFrameCmd.grid(column=1, row=3, sticky=(N, S, E, W))
38
39
40 #Frame for COM messages
41
42 gFrame21 = ttk.Frame(gRoot, padding=10, style='TFrame')
43 gFrame21.grid(column=1, row=2, sticky=(W, E, N))
44 gRoot.resizable(0,0)
45
46
47 for x in range(10):
48     gFrame.columnconfigure(x, weight = x)
49     gFrame.rowconfigure(x, weight = x)
50
51 labell=ttk.Label(gFrame, text = "Serial Console")
52 labell.grid(column=2, row=0)
53 gFrame.rowconfigure(0, weight=2)
54
55 sty.configure("label2 TLabel", borderwidth=4, relief="ridge", foreground="red", ipadx=10)
56 label2=ttk.Label(gFrame, sty="label2 TLabel", text = "Select Com Port")
57 label2.grid(column=1, row=1, sticky = (N,E,W,S))
58
59 """
60 Com Port List
61 """
62 #Start
63 ports = serial.tools.list_ports.comports()
64 com_port_list = [com[0] for com in ports]
65 com_port_list.insert(0,"Select an Option")
66 if dbg == 1:
67     print(com_port_list)
68 #END
69 com_value_inside = StringVar()
70 baud_value_inside = StringVar()
71 baud_menu = ttk.OptionMenu(gFrame,baud_value_inside,"select baud rate","9600",

```

```

72             '19200','28800','38400','57600','76800')
73     baud_menu.grid(column=3, row=1, sticky = (E))
74 def com_port_list_update():
75     global ports
76     global com_port_list
77     ports = serial.tools.list_ports.comports()
78     com_port_list = [com[0] for com in ports]
79     com_port_list.insert(0,"Select an Option")
80     if dbg == 1:
81         print(com_port_list)
82     com_menu = ttk.OptionMenu(gFrame,com_value_inside,*com_port_list)
83     com_menu.grid(column=2, row=1, sticky = (E))
84     #Frame for the COM LIST
85     gRoot_com_list = Toplevel(gRoot)
86     x = gRoot.winfo_x()
87     y = gRoot.winfo_y()
88     gRoot_com_list.geometry("%d+%d" %(x+200,y+200))
89     gFrame01 = ttk.Frame(gRoot_com_list,padding=10)
90     gFrame01.grid(column=0,row=1, sticky=(W))
91     #Create a horizontal scrollbar
92     scrollbar = ttk.Scrollbar(gFrame01, orient= 'horizontal')
93     scrollbar.grid(column=1,row=2, sticky=W+E)
94
95     Lb1 = Listbox(gFrame01, xscrollcommand = 1, width = 50, font= ('Helvetica 8 bold'))
96     counter = 0;
97     for x in ports:
98         Lb1.insert(counter, str(x))
99     #print (counter)
100    counter += 1
101    Lb1.grid(column=1,row=1, sticky=W+E)
102    Lb1.config(xscrollcommand= scrollbar.set)
103
104    #Configure the scrollbar
105    scrollbar.config(command= Lb1.xview)
106
107
108 def serial_print():
109     global serFlag
110     global ser
111     global counter1
112     x =""
113     #print("Task 1 assigned to thread: {}".format(threading.current_thread().name))
114     #print("ID of process running task 1: {}".format(os.getpid()))
115     if(serFlag):
116         if(ser.in_waiting>0):
117             #
118             try:
119                 #x = ser.read(ser.in_waiting)
120                 x = ser.readline(ser.in_waiting)
121                 #x = ser.read_until(expected='\n', size=ser.in_waiting)
122                 #print(x)
123                 y = str(x.decode())
124                 Lb2.insert(counter1, str(y))
125                 Lb2.see("end")
126                 #print (counter1)
127                 counter1 += 1
128                 #gFrame.after(100,serial_print)
129             except:
130                 pass
131             ser.flush()
132             gFrame.after(100,serial_print)
133
134 def CONFIG():
135     ser.write("CONFIG".encode())
136
137 def write_intg(intg : str):
138     CINTG = ("INTG:" + intg).replace("\n","\r")
139     ser.write(CINTG.encode())
140
141 def write_inti(inti : str):
142     CINTI = ("INTI:" + inti).replace("\n","\r")

```

```

143     ser.write(CINTI.encode())
144
145     def write_toff(toff : str):
146         CTOFF = ("TOFF:" + toff).replace("\n","\r")
147         ser.write(CTOFF.encode())
148
149     def write_leds(leds : str):
150         if (leds == "ON"):
151             CLEDS = "LEDV:1\r"
152         elif (leds == "OFF"):
153             CLEDS = "LEDV:0\r"
154         else:
155             CLEDS = "LEDV:1\r"
156
157         ser.write(CLEDS.encode())
158
159     def write_configs(tgnss : str, timu : str, toff: str, ledst : str):
160
161         CINTG = ("INTG:" + tgnss).replace("\n","\r")
162         ser.write(CINTG.encode())
163         CINTI = ("INTI:" + timu).replace("\n","\r")
164         ser.write(CINTI.encode())
165         CTOFF = ("TOFF:" + toff).replace("\n","\r")
166         ser.write(CTOFF.encode())
167
168     def EXIT():
169         ser.write("EXIT".encode())
170
171     def config_mode():
172         filewin2 = Toplevel(gRoot)
173         filewin2.geometry("450x210")
174
175         Label(filewin2, text = "GNSS measure interval : ", anchor='w').grid(column=1, row =
176             1)
176         txt_tgnss = Text(filewin2, height=1, width=10)
177         txt_tgnss.grid(column=2, row = 1)
178         txt_tgnss.insert(END, "5000")
179         button_intg = ttk.Button(filewin2, text="Send", command=lambda:[write_intg(
179             txt_tgnss.get(1.0, END))]).grid(column=3, row = 1)
180
181         Label(filewin2, text = "IMU measure interval : ", anchor='w').grid(column=1, row =
182             2)
182         txt_timu = Text(filewin2, height=1, width=10)
183         txt_timu.grid(column=2, row = 2)
184         txt_timu.insert(END, "500")
185         button_inti = ttk.Button(filewin2, text="Send", command=lambda:[write_inti(
185             txt_timu.get(1.0, END))]).grid(column=3, row = 2)
186
187         Label(filewin2, text = "Inactive delay : ", anchor='w').grid(column=1, row = 3)
188         txt_toff = Text(filewin2, height=1, width=10)
189         txt_toff.grid(column=2, row = 3)
190         txt_toff.insert(END, "60")
191         button_toff = ttk.Button(filewin2, text="Send", command=lambda:[write_toff(
191             txt_toff.get(1.0, END))]).grid(column=3, row = 3)
192
193         ledList = StringVar(filewin2)
194         ledList.set("txt")
195         Label(filewin2, text = "Estat LED de Vie : ", anchor='w').grid(column=1, row = 4)
196         led_menu = ttk.OptionMenu(filewin2, ledList, "ON", "ON", "OFF")
197         led_menu.config(width=5)
198         led_menu.grid(column=2, row = 4)
199         button_toff = ttk.Button(filewin2, text="Send", command=lambda:[write_leds(ledList
199             .get())]).grid(column=3, row = 4)
200
201
202         Label(filewin2, text = "").grid(column=1, row = 5)
203         button2 = ttk.Button(filewin2, text="Exit", command=lambda:[EXIT(), filewin2.
203             destroy()], underline=TRUE).grid(column=1, row = 7, columnspan=2)
204         #filewin2.protocol("WM_DELETE_WINDOW", EXIT())
205
206     ser = serial.Serial()
207     serFlag = 0

```

```
208 def serial_connect(com_port,baud_rate):
209     global ser
210     ser.baudrate = baud_rate
211     ser.port = com_port
212     ser.timeout = 1
213     ser._xonxoff=1
214     ser.bytesize=serial.EIGHTBITS
215     ser.parity=serial.PARITY_NONE
216     ser.stopbits=serial.STOPBITS_ONE
217     ser.open()
218     global serFlag
219     serFlag = 1
220
221     t1 = threading.Thread(target = serial_print, args = (), daemon=1)
222     t1.start()
223     #t1.join()
224     """
225     P1 = multiprocessing.Process(target = serial_print, args=())
226     P1.start()
227     P1.join()
228     """
229     #serial_print()
230     counter1 = 0;
231
232     def SHUTDOWN():
233         ser.write("SHUTDOWN".encode())
234     def GCLR():
235         ser.write("GCLR".encode())
236     def ICLR():
237         ser.write("ICLR".encode())
238     def GLIVE():
239         ser.write("GLIVE".encode())
240     def ILIVE():
241         ser.write("ILIVE".encode())
242     def GLOG():
243         ser.write("GLOG".encode())
244     def ILOG():
245         ser.write("ILOG".encode())
246
247     def serial_close():
248         global ser
249         global serFlag
250         serFlag = 0
251         ser.close()
252
253     def power_off():
254         global ser
255         global serFlag
256         serFlag = 0
257         ser.close()
258         if messagebox.askokcancel("Quit", "Do you want to quit?"):
259             gRoot.destroy()
260
261
262     def submit_value():
263         if dbg == 1:
264             print("selected option: {}".format(com_value_inside.get()))
265             print(" Baud Rate {}".format(baud_value_inside.get()))
266             serial_connect(com_value_inside.get(),baud_value_inside.get())
267
268
269     Lb2 = Listbox(gFrame21, width = 130, height=20, xscrollcommand = 1)
270     Lb2.grid(column=1, row = 1, sticky = W+E)
271     Sb2 = ttk.Scrollbar(gFrame21,orient = 'vertical')
272     Sb2.config(command=Lb2.yview)
273     Sb2.grid(column = 2, row =1, sticky=N+S)
274     Sb2v = ttk.Scrollbar(gFrame21,orient = 'horizontal')
275     Sb2v.grid(column = 1, row =2, sticky=W+E)
276     Sb2v.config(command = Lb2.xview)
277     Lb2.configure(xscrollcommand = Sb2v.set, yscrollcommand = Sb2.set)
278
279     def clear_listbox():
```

```

280     Lb2.delete(0,END)
281
282
283
284
285     subBtn = ttk.Button(gFrameCmd, text="GNSS live data", command = GLIVE, width=15)
286     subBtn.grid(column=1, row=1, sticky = (E))
287     subBtn = ttk.Button(gFrameCmd, text="IMU live data", command = ILIVE, width=15)
288     subBtn.grid(column=1, row=2, sticky = (E))
289
290     subBtn = ttk.Button(gFrameCmd, text="Get GNSS logs", command = GLOG, width=20)
291     subBtn.grid(column=2, row=1, sticky = (E))
292     subBtn = ttk.Button(gFrameCmd, text="Get IMU logs", command = ILOG, width=20)
293     subBtn.grid(column=2, row=2, sticky = (E))
294
295     subBtn = ttk.Button(gFrameCmd, text="Delete GNSS logs", command = GCLR, width=20)
296     subBtn.grid(column=3, row=1, sticky = (E))
297     subBtn = ttk.Button(gFrameCmd, text="Delete IMU logs", command = ICLR, width=20)
298     subBtn.grid(column=3, row=2, rowspan=2, sticky = (E))
299
300     subBtn = ttk.Button(gFrameCmd, text="Configurate BlackBox", command = lambda:[CONFIG(), config_mode()], width=24)
301     subBtn.grid(column=0, row=1, rowspan=2, sticky = (E))
302
303
304     subBtn = ttk.Button(gFrameCmd, text="Exit", command = EXIT, style = "W.TButton",
305     underline=TRUE, width=10)
306     subBtn.grid(column=4, row=1, sticky = (E))
307
308     subBtn = ttk.Button(gFrameCmd, text="Shutdown", command = SHUTDOWN, style = "W.TButton",
309     underline=TRUE, width=10)
310     subBtn.grid(column=4, row=2, sticky = (E))
311
312     subBtn = ttk.Button(gFrame, text="submit", command = submit_value)
313     subBtn.grid(column=4, row=1, sticky = (E))
314
315     RefreshBtn = ttk.Button(gFrame, text="Get List", command = com_port_list_update)
316     RefreshBtn.grid(column=2, row=2, sticky = (E))
317
318
319
320     closeBtn = ttk.Button(gFrame, text="Disconnect", command = serial_close)
321     closeBtn.grid(column=4, row=2, sticky = (E))
322
323     clearBtn = ttk.Button(gFrame, text="Clear Messages", command = clear_listbox)
324     clearBtn.grid(column=3, row=2, sticky = (E))
325
326
327
328 """
329 #Add a Listbox Widget
330 listbox = Listbox(win, width= 350, font= ('Helvetica 15 bold'))
331 listbox.pack(side= LEFT, fill= BOTH)
332
333 #Add values to the Listbox
334 for values in range(1,101):
335     listbox.insert(END, values)
336 """
337 def donothing():
338     filewin = Toplevel(gRoot)
339     button = Button(filewin, text="Do nothing button")
340     button.pack()
341
342 def writeFile(fileName : str, extension : str):
343
344     extension = extension.replace("\n", "")
345     fileName = fileName.replace("\n", "")
346     filePath = str("./" + fileName + "." + extension)
347     txtToSave = "".join(str(el) for el in Lb2.get(0, END))
348

```

```
349     if(extension == "csv"):
350         with open(filePath, 'w', newline='', encoding='utf-8') as f:
351             writer = csv.writer(f)
352             writer.writerow(Lb2.get(0, END))
353
354     else:
355         file = open(filePath,"w")
356         file.write(txtToSave)
357
358
359
360 def save():
361     filewin = Toplevel(gRoot)
362     filewin.geometry("200x170")
363     Label(filewin, text = "File name : ").pack()
364
365     today = datetime.datetime.now()
366
367     display_text = StringVar()
368     txt_save = Text(filewin, height=1, width=20)
369     txt_save.pack()
370     txt_save.insert(END, "log_"+today.strftime("%m%d%y_%H%M%S"))
371
372     Label(filewin, text = "Format : ").pack()
373
374     varList = StringVar(filewin)
375     varList.set("txt")
376     format_menu = ttk.OptionMenu(filewin, varList, "txt", "txt", "csv", "xls", "docx",
377     , "odf")
378     format_menu.config(width=5)
379     format_menu.pack()
380
381     Label(filewin, text = "").pack()
382
383     button = ttk.Button(filewin, text="Sauvegarder", command=lambda:[writeFile(
384         txt_save.get(1.0, END), varList.get()), filewin.destroy()], underline=True).pack()
385
386     #Lb2.text
387
388 def About_me():
389     filewin = Toplevel(gRoot)
390     Label1 = Label(filewin, text = "https://github.com/Ali-Z0/1924B_MiniBoiteNoire").pack()
391     button = Button(filewin, text="Quit", command = filewin.destroy).pack()
392
393     menubar = Menu(gRoot)
394     filemenu = Menu(menubar, tearoff=0)
395     #filemenu.add_command(label="New", command=donothing)
396     #filemenu.add_command(label="Open", command=donothing)
397     filemenu.add_command(label="Save", command=save)
398     #filemenu.add_command(label="Save as...", command=donothing)
399     filemenu.add_command(label="Close", command=power_off)
400
401     #filemenu.add_separator()
402
403     #filemenu.add_command(label="Exit", command=gRoot.quit)
404     menubar.add_cascade(label="File", menu=filemenu)
405     editmenu = Menu(menubar, tearoff=0)
406     editmenu.add_command(label="Undo", command=donothing)
407
408     editmenu.add_separator()
409
410     """
411     editmenu.add_command(label="Cut", command=donothing)
412     editmenu.add_command(label="Copy", command=donothing)
413     editmenu.add_command(label="Paste", command=donothing)
414     editmenu.add_command(label="Delete", command=donothing)
415     editmenu.add_command(label="Select All", command=donothing)
416     """
```

```
417 #menubar.add_cascade(label="Edit", menu=editmenu)
418 helpmenu = Menu(menubar, tearoff=0)
419 #helpmenu.add_command(label="Help Index", command=donothing)
420 #helpmenu.add_command(label="About...", command=donothing)
421 #menubar.add_cascade(label="Help", menu=helpmenu)
422 menubar.add_command(label = "Black Box", command = About_me)
423 menubar.add_separator()
424 menubar.add_command(label = "Quit", command = power_off)
425
426 gRoot.protocol("WM_DELETE_WINDOW", power_off)
427 gRoot.config(menu=menubar)
428 gRoot.mainloop()
```