



Brevet de Technicien Supérieur
SN_{IR}
Session 2020
Lycée Polyvalent Touchard-Washington



P2020 : Ballon Sonde
Johan Le Cren

Dossier technique du projet – partie individuelle
[Git Hub](#)

Sommaire

1- SITUATION DANS LE PROJET.....	3
1.1- Description de la partie personnelle.....	3
1.2- Le synoptique de la structure interne de la nacelle.....	3
2- CONCEPTION DÉTAILLÉE.....	4
2.1- «EMETTRE TRAME SIGFOX».....	4
2.1.1- Câblage du modem.....	4
2.1.2- Exemples de trames.....	5
2.1.3- Description de la classe Sigfox.....	6
2.1.4- Mise en œuvre du module Sigfox.....	7
2.1.5- Algorithme «émission».....	7
2.1.6- Algorithme « testOK ».....	8
2.1.7- Algorithme « conversionbinairehexa ».....	8
2.1.7.1- Exemple de conversion.....	9
2.1.8- Test unitaire de la tâche Sigfox.....	9
2.2- «SAUVEGARDER MESURES ET POSITIONS».....	10
2.2.1- Mise en œuvre de la carte microSD.....	10
2.2.2- Schéma de principe du câblage de la carte Micro SD.....	11
2.2.3- Principe du BUS SPI.....	11
2.2.4- Description de la classe Msdcards.....	13
2.2.5- Algorithme Sauvegarde.....	14
2.2.6- Test unitaire Sauvegarde.....	15
2.3- SIMULATEUR.....	16
2.3.1- Description de la classe Simulation.....	16
2.3.2- Diagramme de classe du simulateur de vol.....	19
2.3.3- Description des méthodes.....	19
2.3.4- Algorithme méthode « gestion ».....	20
2.3.5- Algorithme « sensorTask ».....	21
2.4- FILES.....	22
2.4.1- Définition d'une file.....	22
2.4.2- Mise en œuvre des tâches et des files.....	22
2.4.3- Exemple pour la tâche sigfox.....	23

2.4.4- Algorithme des tâches et files.....	25
2.4.5- Test unitaire Taches files.....	26
2.5- SERVEUR WEB.....	27
2.5.1- Conception du serveur web.....	27
2.5.2- Description du serveur web.....	28
2.5.3- Réalisation du serveur WEB.....	31
2.5.4- IHM de la page web.....	32
2.6- Fiche de test unitaire global.....	33
2.7- PHYSIQUE.....	34
2.7.1- Analyse de l'antenne quart d'onde fournie par Sigfox.....	34
2.7.2- Antenne quart d'onde (brin rayonnant).....	34
2.7.3- Antenne J-Pole.....	35
2.7.4- Antenne Dipôle.....	35
2.7.5- Antenne Dipôle en ground plane.....	36
2.7.6- Antenne dans le ballon pour l'émission Sigfox.....	36
2.7.7- Diagramme de rayonnement.....	37
2.7.7.1- Tracé du diagramme de rayonnement.....	37
2.7.7.2- Essais préliminaires.....	38
3- CONCLUSION.....	39
4- ANNEXES.....	40
4.1- Définition d'une antenne.....	40
4.2- Pourquoi la fréquence 868Mhz du Réseau Sigfox ?.....	43
4.3- Normes de la fréquence ISM.....	43
4.4- Modulation D-BPSK.....	45

1- SITUATION DANS LE PROJET

1.1- Description de la partie personnelle

Le but de ma partie personnelle est de réaliser trois tâches du programme embarqué dans la nacelle :

Tâche 2 : Le ballon sonde doit pouvoir émettre une trame contenant des télémétries acquises par les différents capteurs présents dans le ballon tout au long du vol.

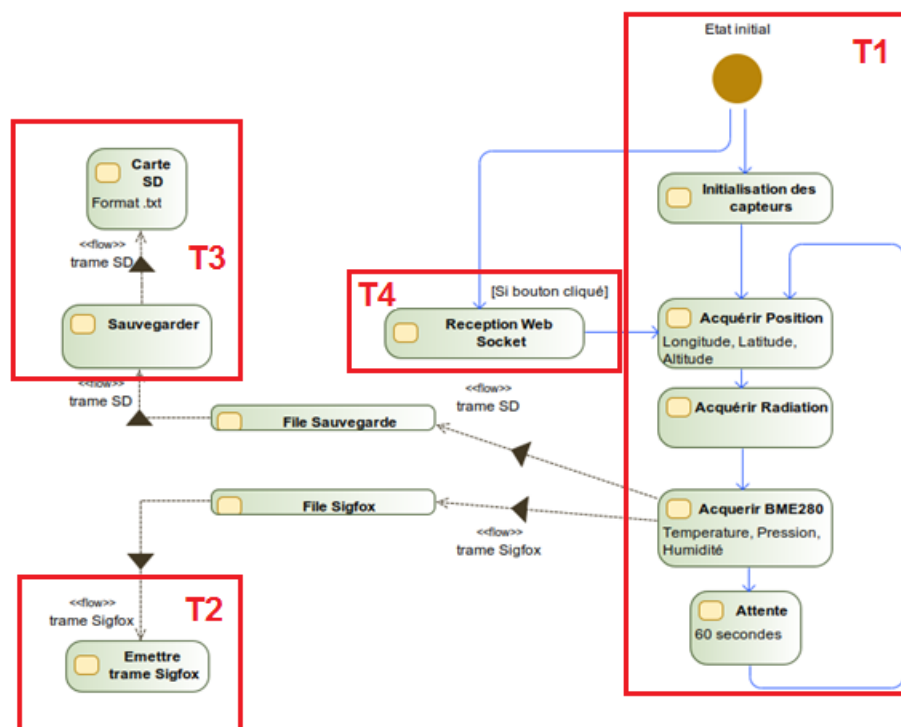
Tâche 3 : Toutes les télémétries sont sauvegardées dans une carte mSd.

Tâche 4 : L'utilisateur doit pouvoir contrôler le bon fonctionnement de l'ensemble avant le lancement. Pour cela, l'utilisateur consulte les télémétries à l'aide d'un téléphone portable connecté directement au point d'accès wifi du ballon sonde (ESP32) muni d'un navigateur (chrome). L'ordre de lancement est donné si tout est conforme. L'utilisateur peut également forcer l'émission d'une trame Sigfox. Cela évite d'attendre 2 minutes et ainsi perdre du temps lors de la séquence de pré lancement.

Tâche 1 : Cette tâche (acquisition des capteurs) est réalisée par Alex, mais pour des raisons pratiques dues au confinement, j'ai réalisé un simulateur de vol. Cela me permet de vérifier le bon fonctionnement des réceptions des files entre les différentes tâches.

Les tâches sont exécutées simultanément avec l'utilisation de FreeRTOS (système d'exploitation temps réel) disponible dans les bibliothèques de l'ESP32.

1.2- Le synoptique de la structure interne de la nacelle

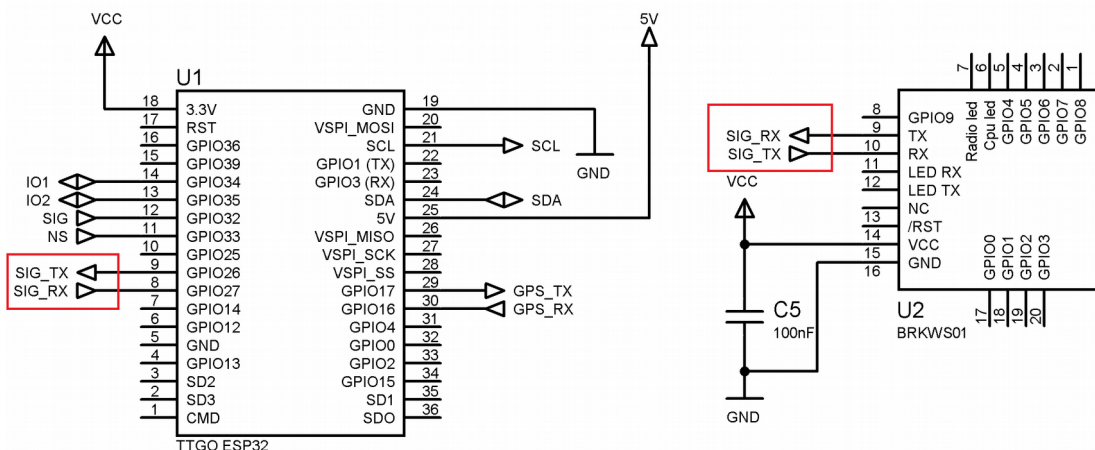


2- CONCEPTION DÉTAILLÉE

2.1- «EMETTRE TRAME SIGFOX»


2.1.1- Câblage du modem

Il faut commencer par repérer le numéro des broches TX et RX du module relié au microcontrôleur.



RX module Sigfox	Relié à	TX ESP32 broche 26
TX module Sigfox	Relié à	RX ESP32 broche 27

Ensuite, il faut vérifier que l'on puisse bien communiquer, pour cela j'utilise un programme de test qui consiste à rediriger les données série du module Sigfox vers le port série de programmation de l'ESP32.

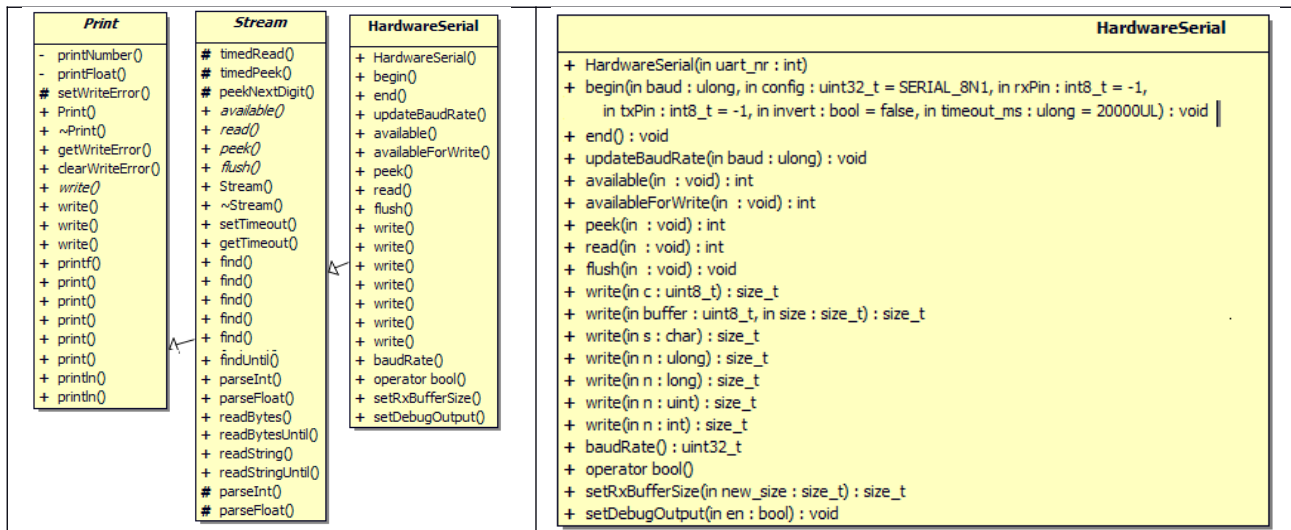
<pre>#include <HardwareSerial.h> HardwareSerial serialSigfox(1); void setup() { Serial.begin(9600); serialSigfox.begin(9600, SERIAL_8N1, 27, 26); } void loop() { char car; if (Serial.available() > 0) { car = Serial.read(); serialSigfox.write(car); } if (serialSigfox.available() > 0) { car = serialSigfox.read(); Serial.write(car); } }</pre>	<p>Résultat dans la console :</p> <p>AT OK</p> <p>AT\$SF= F7C6444258C9B33F62755902 OK</p> <p>Vérification de la trame dans le backend Sigfox</p> <p>F7c6444258c9b33f62755902 </p>
---	--

Seule la commande AT\$SF est utile pour émettre une trame.

La trame de 12 octets est une chaîne de caractères codée en Hexadécimale.

La liaison série se programme avec la classe `HardwareSerial` qui hérite de la classe `Stream` et de la classe `Print`.

Ci-dessous la synthèse de la classe réalisée avec l'utilitaire Bouml.



Dans la classe `HardwareSerial`, j'ai utilisé 4 méthodes pour programmer la classe Sigfox permettant de gérer le modem.

<code>void begin(unsigned long baud, SerialConfig config, SerialMode mode, uint8_t tx_pin)</code>	Initialise l'UART, en définissant le débit en bauds, la taille des données de 8 bits et aucune parité, la broche de réception et de transmission
<code>int available(void)</code>	Renvoie le nombre de caractères actuellement disponibles dans le tampon de réception.
<code>int read(void)</code>	Renvoie le caractère suivant du buffer de réception et le supprime du tampon, ou -1 si aucun caractère n'est disponible dans le buffer.
<code>size_t write(uint8_t c)</code>	Attend que l'émetteur soit inactif, puis transmet le caractère spécifié.

2.1.2- Exemples de trames

Trame position radiation	Structure trame position télémessures
latitude: 47.994961 longitude: 0.204593 altitude: 1000 impulsion: 1 D7,FA,3F,42,D4,80,51,3E,E8,03,02,00	latitude: 47.994961 longitude: 0.204593 température: 20 humidité: 70 pression: 1020 D7,FA,3F,42,D4,80,51,3E,14,46,F9,07

Nom	Latitude	Longitude	altitude	Impulsion+id
Valeur	47.994961	0.204593	1000	1
Hexadécimal	D7,FA,3F,42	D4,80,51,3E	E8,03	0200

Nom	Latitude	Longitude	Température	Humidité	Pression+id
Valeur	47.994961	0.204593	20	70	1020
Hexadécimal	D7,FA,3F,42	D4,80,51,3E	14	46	F907

Détail du champ impulsion+id et pression+id:

	impulsion+id	pression+id
Héxadécimal	0200	F907
Héxadécimale (inversion lsb/msb)	0002	07F9
Binaire	0000 0000 0000 0010	0000 0111 1111 1001

2.1.3- Description de la classe Sigfox

<div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center; margin: 0;">Sigfox</p> <ul style="list-style-type: none"> - rx : uint8_t - tx : uint8_t - serialSigfox : HardwareSerial - debug : bool + Sigfox(in rxPin : uint8_t, in txPin : uint8_t, in debugEn : bool) + emission(in trame : void, in size : uint8_t) : bool - testOK() : bool - conversionBinaireHexa(in octet : byte) : String </div>	<p>Description des attributs :</p> <p>rx : numéro du GPIO de la broche de réception série sur l'ESP32.</p> <p>tx : numéro du GPIO de la broche de transmission série sur l'ESP32.</p> <p>serialSigfox : objet pointeur sur la classe HardwareSerial</p> <p>debug : drapeau pour informer le programmeur d'un débogage des données envoyées sur le moniteur série.</p>
<p>Description des méthodes :</p> <p>emission : commande d'émission d'une trame du modem Sigfox, Les paramètres sont : trame : pointeur sur la mémoire des données à transmettre size : nombre d'octets à transmettre la méthode retourne vrai si la trame a été transmise.</p> <p>testOK : vérifie si le modem répond ok suite à l'envoi de la commande d'émission. La méthode retourne vrai si l'on reçoit bien OK sur la ligne Série en provenance du modem Sigfox.</p> <p>conversionBinaireHexa : conversion d'un octet en chaîne de caractère hexadécimal octet : valeur à convertir La méthode retourne une chaîne de caractère (octet en hexadécimal)</p>	

2.1.4- Mise en œuvre du module Sigfox

Format des télémesures :

Le réseau Sigfox impose 12 octets maximum lors de l'envoi des données. Afin de pouvoir émettre toutes les télémesures, deux trames sont émises en alternance en donnant la priorité à la position géographique dans les deux trames.

<pre><<struct>> positionRadiation + latitude : float + longitude : float + altitude : ushort + id : uint + impulsions : ushort</pre>	<pre>typedef struct { float latitude; /*!< latitude en degrés décimaux*/ float longitude; /*!< longitude en degrés décimaux*/ unsigned short altitude; /*!< altitude en metres*/ unsigned id : 1; /*!< identifiant de la trame toujours égal à 0*/ unsigned short impulsions : 15; /*!< impulsions image de la radiation en cpm (coups par minutes)*/ } positionRadiation;</pre>
<pre><<struct>> positionMesures + latitude : float + longitude : float + temperature : char + humidite : byte + id : uint + pression : ushort</pre>	<pre>typedef struct { float latitude; /*!< latitude en degrés décimaux*/ float longitude; /*!< longitude en degrés décimaux*/ char temperature; /*!< température en °C*/ unsigned char humidite; /*!< humidité en %HR*/ unsigned id : 1; /*!< identifiant de la trame toujours égal à 1*/ unsigned short pression : 15; /*!< pression en hPa*/ } positionMesures;</pre>

2.1.5- Algorithme «emission»

Projet : Ballon sonde		Auteur : Johan Le Cren		
Méthode : emission				
Rôle : Commande d'émission d'une trame du modem Sigfox,				
Environnement :				
Paramètre d'entrée :	trame : pointeur sur la mémoire des données à transmettre,			
Paramètre d'entrée :	size :nombre d'octets à transmettre			
Paramètre de retour :	retourne vrai si la trame a été transmise			
Schéma algorithmique : Debut (pointeur trame, size taille du buffer à envoyer) pointeur bytes ← pointeur trame envoie les caracteres « AT\$SF= » pour i de 0 à size-1 par pas de 1 faire octet ← bytes[i] envoie le caractere sur le port série (conversionbinairehexa(octet)) fin pour envoie le caractere 0xa (\n) retourner testok() fin			Lexiques :	
			Constantes : Aucune	
			Variables locales :	
			I : entier	
			Fonctions :	
testOK bool conversionBinaireHexa string				

2.1.6- Algorithme « testOK »

Projet : Ballon sonde	Auteur : Johan Le Cren
Méthode : testOK	
Rôle : vérifie si le modem répond ok suite à l'envoi d'une commande,	
Environnement :	
Paramètre de retour :	Retourne vrai si la trame a été transmise
Schéma algorithmique : debut retour ← faux data ← chaîne de caractère vide compteur ← 0 tant que (buffer serie vide et compteur<100) faire attendre 100ms compteur =compteur +1 fin tant que tant que (donnees dans le buffer serie) faire caractere ← lire buffer si (caractere !=0xa et caractere !=0xd) alors acq =acq+caractere fin si attendre 10ms fin tant que si « ok » se trouve dans ack alors retour ← faux fin si retourner retour fin	Lexiques : Constantes : Aucune Variables locales : Retour : booléen acq : String caractere : char compteur : entier

2.1.7- Algorithme « conversionbinairehexa »

Projet : Ballon sonde	Auteur : Johan Le Cren
Méthode : conversionbinairehexa	
Rôle : vérifie si le modem répond ok suite à l'envoi d'une commande,	
Environnement :	
Paramètre d'entrée :	octet à convertir
Paramètre de retour :	retourne une chaîne de caractère (octet en hexadécimal)
Schéma algorithmique : debut initialise table = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'} msb ← octet décalé de 4 bits vers la gauche lsb ← octet & 0x0f resultat ← chaîne(table[msb]) + chaîne(table[lsb]); si debug=1 alors affiche (resultat) fin si retourne resultat fin	Lexiques : Constantes : Aucune Variables locales : Resultat : String Msb, lsb : Byte table : char[]

2.1.7.1- Exemple de conversion

octet 0xf1	
quartet msb = f soit 15 en décimal (octet >> 4)	quartet lsb = 1 (octet & 0xf)
table[msb] est égal à 'F' soit 70 en décimal	table[lsb] est égal à '1' soit 49 en décimal
Concaténation des deux caractères : f1 en chaîne de caractères	

2.1.8- Test unitaire de la tâche Sigfox

Fiche de tests			
Nature :	Fonctionnel	Référence :	F2.
Module :	Classe Sigfox		
Objectif :	Vérifier l’envoi de la trame de données du module Sigfox au backend Sigfox		
Condition du test			
État initial du module		Environnement du test	
Programme	Projet Ballon Sonde	PC avec Arduino / Netbeans	
Conditions initiales			
Le modem Sigfox est sous tension.			
Procédure de test			
Récupération des données dans la file sigfox, le début de l'émission			
Repère	Opérations	Résultats attendus	
1	Avec modem Sigfox connecté	aucun	
2	Récupère les données dans la file sigfox	Affichage dans le moniteur série d'un message de vérification si les données sont dans la file sigfox	
3	Convertit les données récupérées de la file en hexadécimal	aucun	
4	Forge la trame avec les données converties en hexadécimal	aucun	
5	Émet la trame en hexadécimal	Affichage dans le moniteur série des données en hexadécimal	
6	Attend la réception du OK en provenance du module Sigfox	Affichage 'Message envoyé avec succès'	
7	Vérifie la réponse du backend sigfox	2020-03-19 16:49:34 211 b75241424f640a3f0221a900 Les données s'affichent bien dans le backend Sigfox	
8	Émet pendant 4 heures 30 minutes	Les données s'affichent bien dans le backend Sigfox	

2.2- «SAUVEGARDER MESURES ET POSITIONS»

2.2.1- Mise en œuvre de la carte microSD

Format d'écriture dans la carte mSd

Pendant toute la durée du vol, le système embarqué dans le ballon sonde enregistre les télémessures dans une carte mSd. Le fichier « telemesure.csv » enregistré dans la carte au format FAT32 est lisible avec un tableur. Les données sont alors facilement exploitables par l'utilisateur pour tracer des courbes ou pour des calculs de statistiques entre différents vols.

Ci-dessous un exemple de résultat dans un tableur

	A	B	C	D	E	F	G	H	I
1	Date	Heure	Latitude	Longitude	Altitude	Pression	Humidité	Température	Impulsions
2	30/04/2020	13:59:15	47,531349	0,20135	155	1017	69	20,1	1
3	30/04/2020	13:59:30	47,5327	0,2027	230	1015	69	20,1	2
4	30/04/2020	13:59:45	47,53405	0,20405	305	1012	69	20	3
5	30/04/2020	14:00:00	47,5354	0,2054	380	1010	69	19,9	3
6	30/04/2020	14:00:15	47,536751	0,206749	455	1008	69	19,8	4
7	30/04/2020	14:00:30	47,538101	0,208099	530	1005	69	19,8	5
8	30/04/2020	14:00:45	47,539452	0,209449	605	1003	68	19,7	5

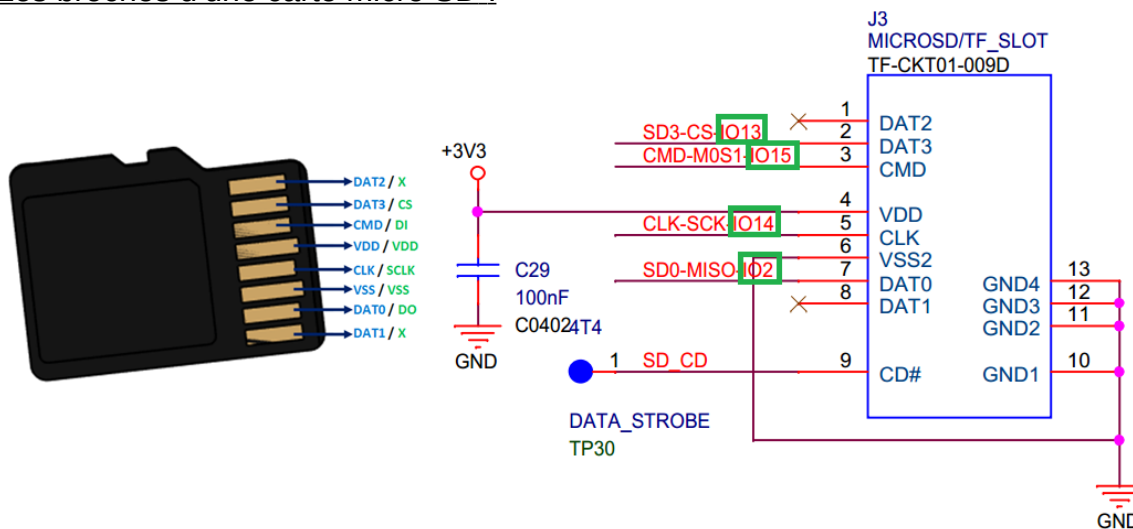
Par convention le caractère « ; » est adopté comme séparateur entre les différents champs. Le caractère de virgule ne peut pas être utilisé car on le retrouve dans les coordonnées géographiques.

Le format des données enregistrées dans la carte mSd a la forme suivante :

```
Date;Heure;Latitude;Longitude;Altitude;Pression;Humidité;Température;Impulsions
30/04/2020;13:59:15;47,531349;0,20135;155;1017;69;20,1;1
30/04/2020;13:59:30;47,532700;0,202700;230;1015;69;20,1;2
30/04/2020;13:59:45;47,534050;0,204050;305;1012;69;20,0;3
30/04/2020;14:00:00;47,535400;0,205400;380;1010;69;19,9;3
30/04/2020;14:00:15;47,536751;0,206749;455;1008;69;19,8;4
30/04/2020;14:00:30;47,538101;0,208099;530;1005;69;19,8;5
30/04/2020;14:00:45;47,539452;0,209449;605;1003;68;19,7;5
```

2.2.2- Schéma de principe du câblage de la carte Micro SD

Les broches d'une carte micro SD :

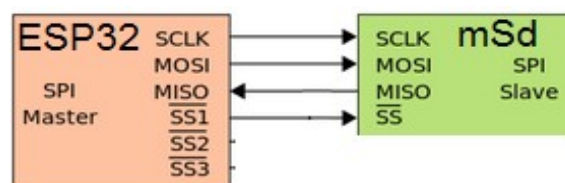


Le schéma montre le numéro des broches utilisées par la carte SD. (Encadré en VERT). Au vu du nom des broches SCK, MOSI, MISO, le bus de communication entre l'ESP32 et la carte mSD est SPI. La bibliothèque est SPI.h.

Nom du signal	Broche
SCK	14
MOSI	15
MISO	2
CS	13

2.2.3- Principe du BUS SPI

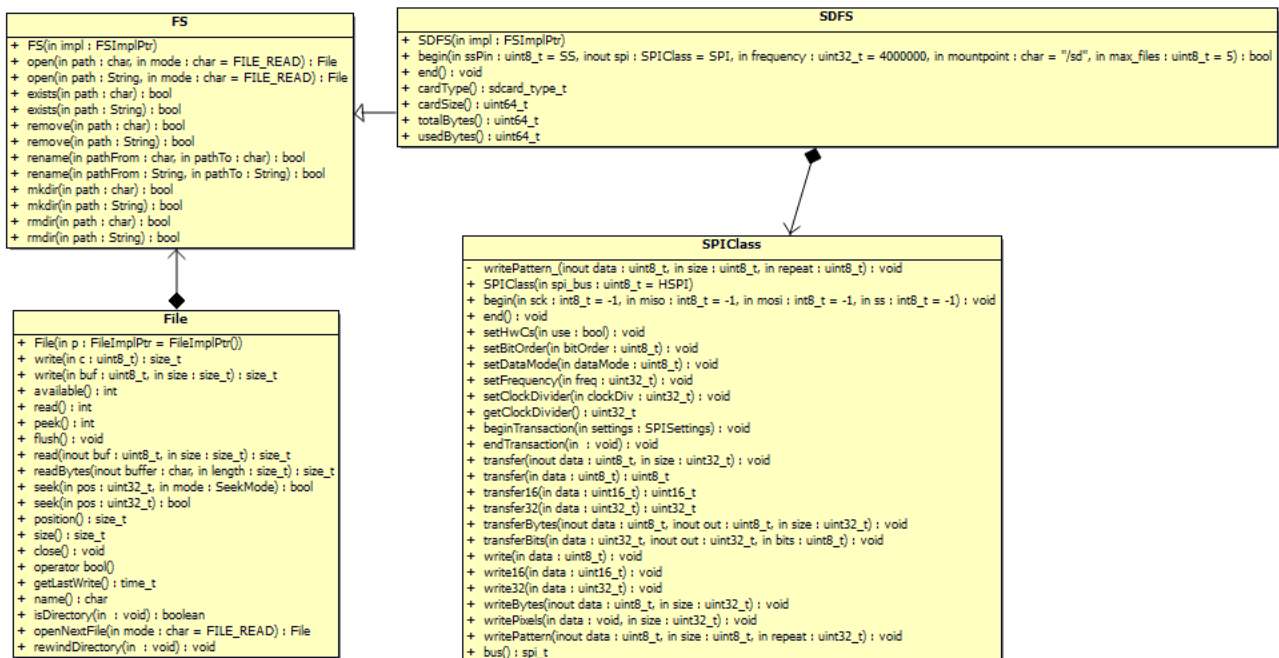
Une liaison SPI (pour Serial Peripheral Interface) est un bus de données séries synchrones qui fonctionne en mode full-duplex (échange bidirectionnel en même temps). Les circuits communiquent selon un schéma maître-esclave, où le maître (ESP32) contrôle la carte mémoire mSd.



Le bus SPI utilise quatre signaux logiques :

- **SCLK** — Serial Clock, Horloge (généré par le maître)
- **MOSI** — Master Output, Slave Input (généré par le maître)
- **MISO** — Master Input, Slave Output (généré par l'esclave)
- **SS** — Slave Select, Actif à l'état bas (généré par le maître)

La fréquence du bus SPI sur en ESP32 peut aller jusqu'à 40Mhz. Le contrôle du bus SPI est géré automatiquement via les bibliothèques de gestion de la carte mSd fournie dans l'environnement de programmation. Pour accéder à la carte mSd, j'ai utilisé les quatre classes décrites ci-dessous et disponibles dans l'environnement de programmation de l'ESP32.



On remarque des méthodes très similaires à la gestion des fichiers en c sur le PC. (open, close, write, etc.)

2.2.4- Description de la classe Msdcard

Msdcard	Description des attributs :
<pre> - SD_CS : uint8_t + Msdcard(in sckPin : uint8_t, in misoPin : uint8_t, in mosiPin : uint8_t, in ssPin : uint8_t) + begin() : bool + appendFile(inout fs : fs::FS, in path : char, in message : char) : bool + writeFile(inout fs : fs::FS, in path : char, in message : char) : bool + fichierPresent(inout fs : fs::FS, in path : char) : bool + replaceCaractere(in ligneCsv : char, in longueur : int, in carSource : char, in carDest : char) : void </pre>	<p>SD_CS : numéro du GPIO de la broche du chip select de la carte mSd</p>
<p>Description des méthodes :</p> <p>begin : initialise les paramètres de la liaison SPI, vérifie la présence de la carte mSD, Affiche le type de carte mSD présente, Affiche la taille de la carte mémoire retourne un booleen à vrai si erreur (pas de carte mSD)</p> <p>appendFile : ajoute des données dans le fichier à la suite sans écraser les données précédentes &fs adresse de la classe file system *path chemin et nom du fichier en ajout de données *message chaine de caractères à ajouter dans le fichier retourne un booleen à vrai si erreur d'écriture.</p> <p>writeFile : Ecrit des données dans un fichier. Si le fichier existe, il est remplacé. &fs adresse de la classe file system *path chemin et nom du fichier en ajout de données *message chaine de caractères à ajouter dans le fichier retourne un booleen à vrai si erreur d'écriture.</p> <p>fichierPresent : vérifie si le fichier de sauvegarde est présent ou non &fs adresse de la classe file system *path chemin et nom du fichier à tester retourne un booleen à vrai si le fichier existe sur la carte SD</p> <p>replaceCaractere : remplace un caractère par un autre (point par une virgule) ligneCsv[] chaine de caractères à traiter longueur longueur de la chaine carSource caractère à rechercher dans la chaine carDest caractère à remplacer dans la chaine</p>	

2.2.5- Algorithme Sauvegarde

Projet : Ballon sonde		Auteur : Johan Le Cren	
Fonction sauvegarde des données			
Rôle : Ecrit les télémesures dans le fichier texte de la carte SD			
Environnement :			
En entrée :	Télémesures présentes dans la file		
En sortie :	Télémesures enregistrées dans la carte mSd ("/telemesures.csv")		
Paramètre d'entrée :	Aucun		
Paramètre de sortie :	Aucun		
Paramètre d'E/S :	Aucun		
Paramètre de retour :	Aucun		
Schéma algorithmique :		Lexiques :	
		Constantes : Aucune	
		Variables locales :	
		Booléens :	
		chaîne de caractères	
<div>Debut</div> <div>erreurCarte ← initialise carte mSd</div> <div>si erreurCarte = false alors</div> <div> si la présence du fichier = faux alors</div> <div> erreurWrite ← ecriture de l'entête dans le</div> <div>fichier</div> <div> si erreurWrite = false alors</div> <div> affiche ("données écrites")</div> <div> fin si</div> <div> fin si</div> <div>sinon affiche ("Pas de carte présente")</div> <div>tant que que vrai faire</div> <div> defile les données dans la file queueMSD</div> <div> horaire ← formatage de l'heure</div> <div> ligneCsv ← formatage de la ligne csv</div> <div> si erreurCarte = faux alors</div> <div> erreurAppend ← ajoute ligneCsv au fichier</div> <div> si erreurAppend = faux alors</div> <div> affiche (" données ajoutées");</div> <div> fin si</div> <div> sinon affiche ("Pas de carte présente")</div> <div> fin si</div> <div>temporisation de 50ms</div> <div>fin</div>		erreurCarte	
		erreurWrite;	
		erreurAppend	
		ligneCsv	
		horaire	

2.2.6- Test unitaire Sauvegarde

Fiche de tests			
Nature :	Fonctionnel	Référence :	
Module :	Programme d'écriture dans la carte mSD		
Objectif :	Vérifier l'écriture et l'ajout de données au format CSV dans la carte mSD		
Condition du test			
État initial du module		Environnement du test	
Programme	Test_msdcard.ino	PC avec Moniteur série	
Conditions initiales			
La carte msd est formatée en FAT32 et insérée sous le module ESP32			
Procédure de test			
Constater dans le moniteur série la gestion de l'écriture des données sur la carte SD			
Repère	Opérations	Résultats attendus	
1	La carte est connectée	Affichage du type de carte SDHC	
2	Détection de la taille de la carte SD	Taille SD Card : 7695MB	
3	Création du fichier et écriture de l'entête du fichier csv	Ecriture du fichier : /telemesures.csv Succès	
4	Ajout des données de télémesures dans le fichier csv	Données ajoutées dans : /telemesures.csv Succès	
5	Couper l'alimentation de l'esp32 et retirer la carte sd	Aucun	
6	Lire le contenu le carte avec un lecteur USB	Aucun	
7	Ouvrir le fichier CSV avec un tableur (Calc).	Les données de télémesures sont bien dans différentes cases du tableur	

2.3- SIMULATEUR

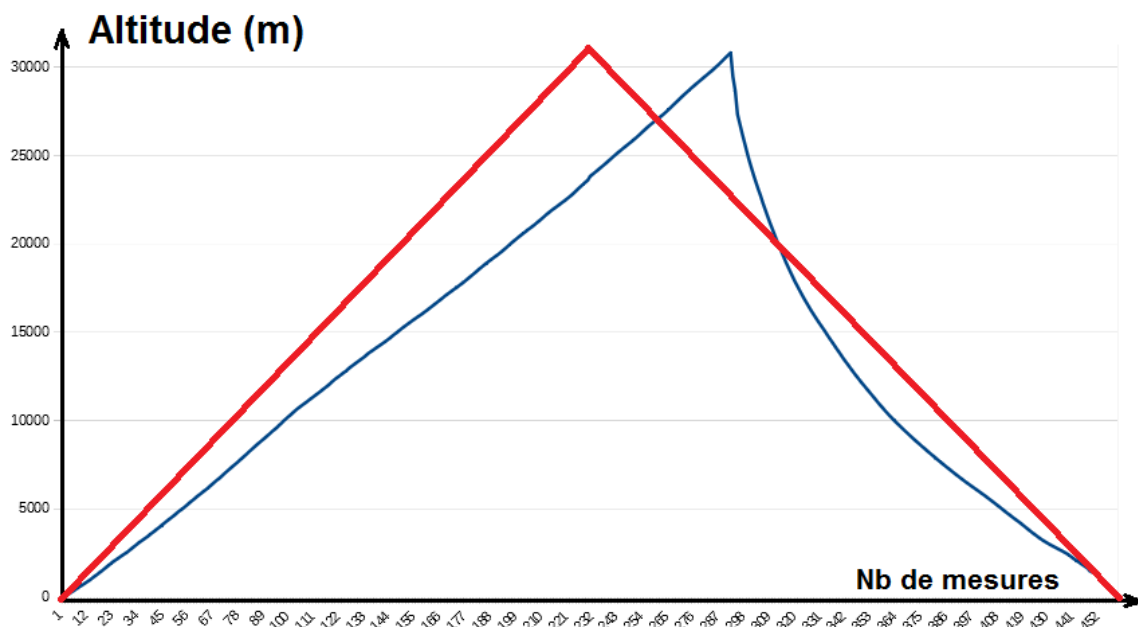
2.3.1- Description de la classe Simulation

Afin de tester la gestion des files avec freeRTOS, j'ai programmé un simulateur de vol de ballon.

Cela évite d'envoyer par onde radio toujours les mêmes valeurs de position, température, humidité, pression et radiations.

De plus cette classe me permet de tester aussi la mise hors service du WIFI lorsque le ballon est supérieur à 2000 mètres d'altitude et d'obtenir un fichier CSV dans la carte mSD aussi réaliste que possible.

Si je prends comme exemple le tracé de l'altitude en fonction du nombre de mesures dans le temps (tracé en bleu), on constate que la descente n'est pas linéaire. Pour des raisons de temps, je n'ai pas implanté le modèle réel dans le simulateur de vol. La montée et la descente du ballon sont basées sur un calcul proportionnel sous forme de produit en croix, comme le montre la figure ci-dessous (tracé en rouge)



Il en va de même pour les autres paramètres : Température, humidité, pression et radiations.

La date du vol est toujours la même, seule l'heure change.

L'utilisateur doit paramétrer les valeurs indiquant les conditions initiales au sol avant lancement

```
#define ANNEE 2020 //date de lancement
#define MOIS 04
#define JOUR 30
#define HEURE 13 //heure de lancement
#define MINUTE 59
#define LATITUDE 47.53 //position de départ
#define LONGITUDE 0.20 //En degrés décimaux
#define ALTITUDE 80 //Altitude de départ (mètres)
#define PRESSION 1020 //Pression au sol (hpa)
#define TEMPERATURE 20.2 //Température au sol (°C)
#define HUMIDITE 70 //Humidité au sol (%HR)
#define IMPULSIONS 1 //Nb Impulsions au sol (CPM)
```

Ensuite l'utilisateur paramètre l'altitude d'éclatement, la vitesse ascensionnelle en (m/s), la vitesse de déplacement par rapport au sol, l'intervalle de temps entre chaque sauvegarde dans la carte mSD, ainsi que le nombre de sauvegardes avant la commande d'une émission radio Sigfox.

```
//condition du vol
#define ECLATEMENT 32000 //Altitude d'éclatement (mètres)
#define VITESSE_ASC 5 // vitesse ascensionnelle (m/s)
#define VITESSE_SOL 10 //vitesse de déplacement par rapport au sol (m/s)
#define INTERVALLE 15 // Intervalle entre chaque mesure (secondes) doit être un
diviseur entier de 60
#define RAPPORT_SD_SIGFOX 8 // 8 mémorisations dans la carte mSd pour 1 envoi
télémessures Sigfox (2 minutes = 8*15 secondes)
```

Le constructeur de la classe Simulation va calculer les différents pas d'incrémentations des mesures et du déplacement.

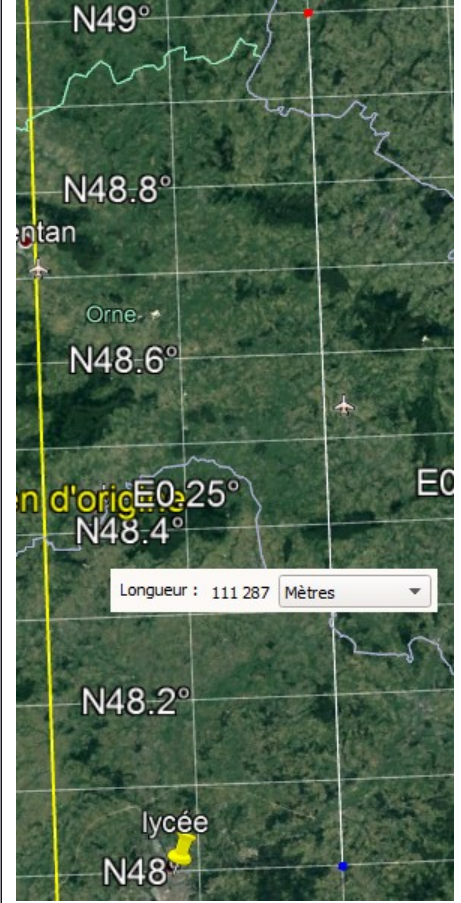
Pour calculer les différents pas d'incrémentations, je me base sur le calcul du nombre de mesures (nbMesuresMontee) que l'on va mémoriser dans la carte mSD pendant la phase de montée.

```
nbMesuresMontee = (eclatement - altitudeMin) / (vitesse_asc *
intervalle); //calcul du nombre de mesures
```

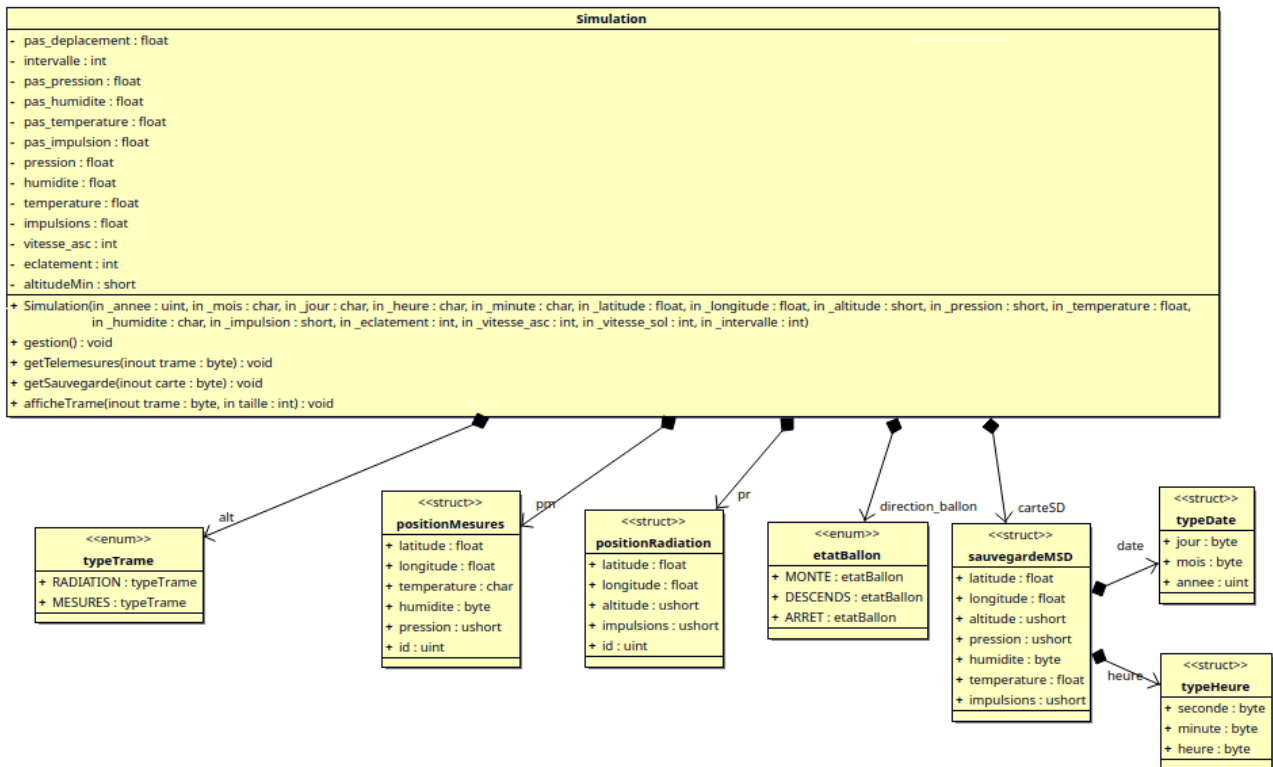
C'est à partir de la variable nbMesuresMontee que je vais calculer les différents pas d'incrémentation comme par exemple la pression :

```
pas_pression = (float) pression/ nbMesuresMontee; //calcul du pas de la pression
```

Il en va de même pour les autres mesures physiques.

	<p>Calcul du pas de déplacement :</p> <p>Le calcul du déplacement est un peu particulier puisqu'il faut modifier les coordonnées de longitude et de latitude en degrés décimaux. Le calcul est basé sur le fait que, sur la latitude, une minute d'arc correspond à un mille nautique soit 1852 mètres. Donc un déplacement d'un degré correspondra à $60 \times 1852 = 111120$ mètres. (valeur vérifiée approximativement dans google earth)</p> <p>Sachant que l'on connaît la distance de déplacement entre chaque mesure, on en déduit la fraction de degré à utiliser pour l'incrémentation.</p> <pre>pas_deplacement = (float) (_vitesse_sol * intervalle) / (60 * 1852); //</pre> <p>La même valeur de pas est utilisée pour le calcul de la longitude même si elle devrait être légèrement différente à mesure que l'on se rapproche des pôles.</p> <p>L'essentiel est de pouvoir déplacer le ballon de manière plausible.</p> <p>Par défaut le ballon partira toujours au nord-est. Je n'ai pas implanté de possibilité de choix du cap.</p>
--	--

2.3.2- Diagramme de classe du simulateur de vol



La classe Simulation utilise les structures décrites précédemment positionMesures, positionRadiation et sauvegardeMsd. Il est nécessaire d'ajouter deux types énumérés :

typeTrame : Indique le type de la trame qui va être envoyée :

- RADIATION : correspondant à la structure positionRadiation ;
- MESURES : correspondant à la structure positionMesures.

etatBallon : Indique quelle est la situation du ballon représentant les 3 phases du vol :

- MONTE : le ballon est en phase de montée ;
- DESCENDS : le ballon a éclaté et est en phase de descente ;
- ARRET : le ballon est posé au sol, donc à l'arrêt.

2.3.3- Description des méthodes

gestion	Met à jour la position, les mesures physiques au rythme de l'intervalle de temps passé en paramètre dans le constructeur.
getTelemesures	Reçoit en paramètre un pointeur sur une mémoire. La méthode copie les données des télémessures dans cette mémoire. Le contenu des télémessures est alterné à chaque appel (position et mesure ou position et radiation)
getSauvegarde	Reçoit en paramètre un pointeur sur une mémoire. La méthode copie toutes données à l'instant actuel du vol conformément à la structure de données sauvegardeMSD
afficheTrame	Reçoit en paramètre un pointeur sur une mémoire et le nombre d'octets à afficher en hexadécimal. Cette méthode est utile pour une comparaison avec les données affichées dans le backend Sigfox.

2.3.4- Algorithme méthode « gestion »

Méthode gestion	
<p>Schéma algorithmique</p> <p>début</p> <p>temporisation d'une durée intervalle en secondes</p> <p>carteSD.heure.seconde \leftarrow carteSD.heure.seconde + intervalle</p> <p>si carteSD.heure.seconde = 60 alors</p> <p> carteSD.heure.minute = carteSD.heure.minute + 1</p> <p> carteSD.heure.seconde \leftarrow 0</p> <p>si carteSD.heure.minute = 60 alors</p> <p> carteSD.heure.heure \leftarrow carteSD.heure.heure + 1</p> <p> carteSD.heure.minute \leftarrow 0;</p> <p>si carteSD.heure.heure = 24 alors</p> <p> carteSD.heure.heure \leftarrow 0</p> <p> finsi</p> <p> finsi</p> <p>finsi</p> <p>si direction_ballon = MONTE alors</p> <p> carteSD.latitude \leftarrow carteSD.latitude + pas_deplacement</p> <p> carteSD.longitude \leftarrow carteSD.longitude + pas_deplacement</p> <p> carteSD.altitude \leftarrow carteSD.altitude + (vitesse_asc * intervalle)</p> <p> pression \leftarrow pression - pas_pression</p> <p> temperature \leftarrow temperature - pas_temperature</p> <p> humidite \leftarrow humidite - pas_humidite</p> <p> impulsions \leftarrow impulsions + pas_impulsion</p> <p>si carteSD.altitude > eclatement alors</p> <p> direction_ballon \leftarrow DESCENDS;</p> <p> finsi</p> <p>sinon si direction_ballon = DESCENDS alors</p> <p> carteSD.latitude \leftarrow carteSD.latitude + pas_deplacement</p> <p> carteSD.longitude \leftarrow carteSD.longitude + pas_deplacement</p> <p> carteSD.altitude \leftarrow carteSD.altitude - (vitesse_asc * intervalle)</p> <p> pression \leftarrow pression + pas_pression</p> <p> temperature \leftarrow temperature + pas_temperature</p> <p> humidite \leftarrow humidite + pas_humidite</p> <p> impulsions \leftarrow impulsions - pas_impulsion</p> <p>si carteSD.altitude < altitudeMin alors</p> <p> direction_ballon \leftarrow ARRET</p> <p> finsi</p> <p>fin</p>	<p>Attributs :</p> <p>carteSD de type sauvegardeMsd</p> <p>variables décimales</p> <p>pas_deplacement</p> <p>pas_pression</p> <p>pas_humidite</p> <p>pas_temperature</p> <p>pas_impulsion</p> <p>pression</p> <p>humidite</p> <p>temperature</p> <p>impulsions</p> <p>variables entières</p> <p>intervalle</p> <p>vitesse_asc</p> <p>eclatement</p> <p>altitudeMin</p> <p>type enum :</p> <p>direction_ballon :</p> <p>etatBallon</p>

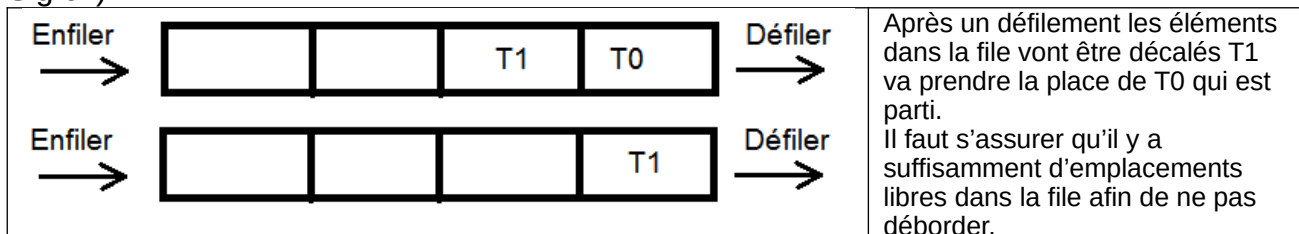
2.3.5- Algorithme « sensorTask »

Description de la tache sensorTask : La tache sensorTask utilise la classe Simulation et gère l'envoi des données vers les tâches sigfoxTask et msdTask à l'aide des 2 files queueMSD et queueSigfox. SensorTask reçoit par l'intermédiaire d'une file queueEnvoiBouton la demande de l'utilisateur (connecté sur le point d'accès WIFI de l'ESP32) pour l'envoi d'une donnée radio Sigfox.	
Schéma algorithmique début commande ← faux *trame ← allocation mémoire de taille positionRadiation *carte ← allocation mémoire de taille sauvegardeMSD Tanque (1) faire Pour nbSauvCarte de 0 à RAPPORT_SD_SIGFOX -1 par pas de 1 faire simul.gestion() simul.getSauvegarde(carte) xQueueSend(queueMSD, carte, portMAX_DELAY) xQueueReceive(queueEnvoiBouton, &commande, 0) si commande = true alors simul.getTelemesures(trame) simul.afficheTrame(trame, de taille positionRadiation) xQueueSend(queueSigfox, trame, portMAX_DELAY) commande ← faux; finsi finpour temporisation de 100ms simul.getTelemesures(trame) simul.afficheTrame(trame, de taille positionRadiation) xQueueSend(queueSigfox, trame, portMAX_DELAY) fintant que fin	Variables locales : variable booléenne : commande pointeurs (octet) : trame carte variable entière nbSauvCarte
	Fonctions freeRTOS xQueueSend xQueueReceive
	Instance de la classe Simulation Simul
	Directive de compilation RAPPORT_SD_SIGFOX
	Pointeurs de file queueMSD queueEnvoiBouton queueSigfox

2.4- FILES

2.4.1- Définition d'une file

Une file est appelée FIFO (First In First Out), en français premier entré premier sorti. Cela est similaire à une file d'attente où chaque personne correspond à un élément à traiter. (Dans le projet un élément correspond à un message radio à envoyer vers le réseau Sigfox).



2.4.2- Mise en œuvre des tâches et des files

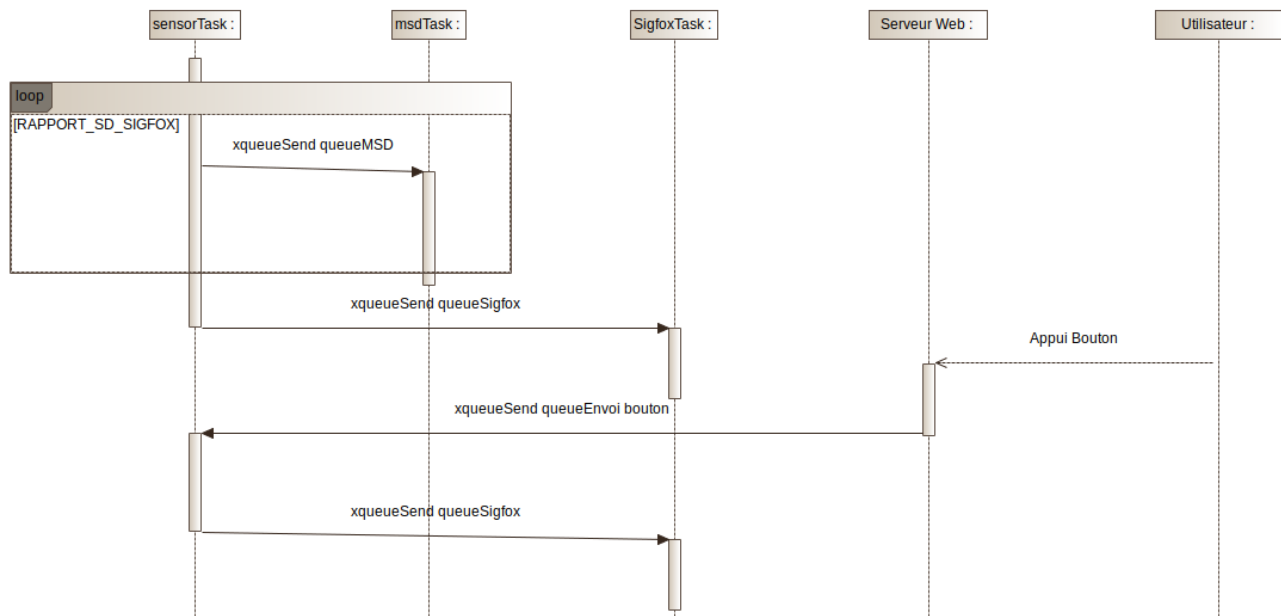
L'objectif est d'utiliser freeRTOS afin de gérer les 4 tâches suivantes :

- Tâche gérant les mesures (sensorTask : T1) et envoyant des mesures dans 2 files (queueSigfox et queueMSD)
- Tâche gérant la réception de la file (queueSigfox) et l'émission des données Radio (sigfoxTask T2)
- Tâche gérant la réception de la file (queueMSD) et la sauvegarde des données sur carte SD (msdTask T3)
- La quatrième tâche est réalisée par la boucle principale du programme pour la gestion de la page HTML.

L'intérêt d'utiliser freeRTOS est de pouvoir gérer plusieurs tâches en parallèle. Le nombre de tâches exécutées simultanément et leur priorité ne sont limités que par l'ESP32. L'ordonnancement (le programme qui gère les différentes tâches) est basé sur le modèle Round-Robin avec gestion des priorités. Le round-robin est une sorte de tourniquet où chaque processus ou tâche qui est sur le tourniquet ne fait que passer devant le processeur, à son tour et pendant un temps fixe.



Le diagramme de séquence suivante montre les 4 tâches ainsi que la communication des trois permettant de gérer le système embarqué dans la nacelle.



Pour comprendre l'utilisation de freeRTOS, je me suis inspiré des exemples fournis sur [github](#) et la [documentation](#) de l'ESP32.

2.4.3- Exemple pour la tâche sigfox

TaskHandle_t Tasksigfox;	Déclaration de la tâche
<pre> xTaskCreatePinnedToCore(sigfoxTask, /* fonction d'entrée de la tâche */ "sigfoxTask", /* nom de la tâche. */ 10000, /* taille mémoire de la pile */ NULL, /* paramètres de la tâche */ 1, /* priorité de la tâche */ &tasksigfox, /* Task handle */ 1); /* tâche sur le coeur 1 */ </pre>	Paramètres de la tâche
<pre> void sigfoxTask(void * pvParameters) { //déclaration des variables locales while(1){ //programme de gestion Emission Sigfox } } </pre>	La fonction liée à la tâche ne doit jamais sortir. Il y a obligatoirement une boucle infinie dans cette fonction, sinon l'ordonnanceur va générer une erreur et le microcontrôleur va redémarrer.

Exemple pour la [file](#) sigfox

(Normalement, un élément dans la file doit suffire, mais par mesure de sécurité la taille est de 10)

<code>#define queueSize 10</code>	La structure <code>positionRadiation</code> est de 12 octets
<code>QueueHandle_t queueMSD;</code>	Déclaration de la file
<code>queueSigfox = xQueueCreate(queueSize, sizeof(positionRadiation));</code>	Création de la file Ici la taille mémoire allouée sera de <code>queueSize * sizeof(positionRadiation)</code> Soit $10 \times 12 = 120$ octets
<code>xQueueSend(queueSigfox, trame, portMAX_DELAY);</code>	Commande effectuée dans la tâche Sensor : On enfile le contenu de la mémoire <code>trame</code> dans <code>queueSigfox</code>
<code>xQueueReceive(queueSigfox, trame, portMAX_DELAY);</code>	Commande effectuée dans la tâche Sigfox : On défile le 1 ^{er} élément entré dans la file <code>queueSigfox</code> (le contenu est transféré dans la mémoire <code>trame</code>)

`portMAX_DELAY` correspond à un temps d'attente infini. Donc la fonction `xQueueReceive` sera bloquante tant qu'il n'y a aucun élément dans la file.

2.4.4- Algorithme des tâches et files

Projet : Ballon sonde		Auteur : Johan Le Cren	
Programme principal			
Rôle : Déclare et gère les 4 tâches			
Environnement :			
En entrée :		Lectures des mesures physiques et de la position GPS	
En sortie :		Mémorisation régulière des télémessures dans une carte mSd et émission radio Sigfox	
Schéma algorithmique :		Lexiques :	
Initialisation :		Constante : ALTITUDE_MAX_WIFI	
Déclarer les 3 tâches Tasksensor, Tasksigfox, Taskmsd Déclarer les 2 files queueSigfox, queueMSD Créer la file queueSigfox de taille de la structure positionRadiation Créer la file queueMSD de taille de la structure sauvegardeMSD Créer la tâche Tasksensor dans le core 0 de priorité 1 Créer la tâche Tasksigfox dans le core 1 de priorité 1 Créer la tâche msdTask dans le core 1 de priorité 2		Variables globales: TaskHandle_t : Tasksensor, Tasksigfox, Taskmsd QueueHandle_t : queueSigfox queueMSD queueEnvoiBouton	
Tasksensor :		Variables locales:	
algorithme page 21		trame : pointeur octet	
Algorithme de la tache msdTask :		carteSd de structure sauvegardeMSD	
algorithme page 14			
Algorithme de la tâche sigfoxTask :			
Alloue une mémoire trame de taille de la structure positionRadiation tantque(1) faire defile queueSigfox dans trame affiche le contenu de trame émission de la trame vers le modem Sigfox			
Algorithme de la boucle principale loop :			
Gestion des requêtes web (handler) si carteSd.altitude > ALTITUDE_MAX_WIFI alors arrêt du serveur arrêt du WIFI fin si			

2.4.5- Test unitaire Taches files

Fiche de tests zzz			
Nature :	Fonctionnel	Référence :	
Module :	Programme principal Taches Files		
Objectif :	Vérifier l’envoi de données de la tâche sensorTask (simulateur) vers les 2 autres tâches (sigfoxTask et msdTask) via 2 files		
Condition du test			
État initial du module		Environnement du test	
Programme	TotalV4.ino	PC avec Moniteur série	
Conditions initiales			
Les données issues du simulateur sont envoyées dans les 2 files.			
Procédure de test			
Constater dans le moniteur série le bon envoi et réception des données dans les 2 tâches respectives			
Repère	Opérations	Résultats attendus	
1	Les données carteSD de la file queueMSD sont enfilées	Envoi file pour carteSD	
2	Les télémesures sont reçues de file queueMSD dans carteSD	télémesures reçues 18/06/2020 13:45:34 47.994961 0.204593 50 1020 70 25.00 1	
3	Les repères 1 et 2 sont effectués 8 fois avec un délai de 15 sec	Affichage des télémesures dans le moniteur série	
4	Les données radio de la file queueSigfox sont enfilées	Envoi file pour Sigfox données position radiation et position temperature/humidite/pression en alternance	
5	La trame Sigfox est reçue de file queueSigfox	Tâche sigfox trame recue D7FA3F42CC3DFC3F32010	
6	Les repères 1, 2, 3, 4 et 5 sont effectués en boucle durant tout le vol	Les valeurs sont différentes de la trame précédente	
7	Consultation du backend Sigfox	Les données sont présentes dans le backend Sigfox	
8	Lecture de la carte mSD	Les données sont présentes dans le fichier telemesures.csv	

2.5- SERVEUR WEB

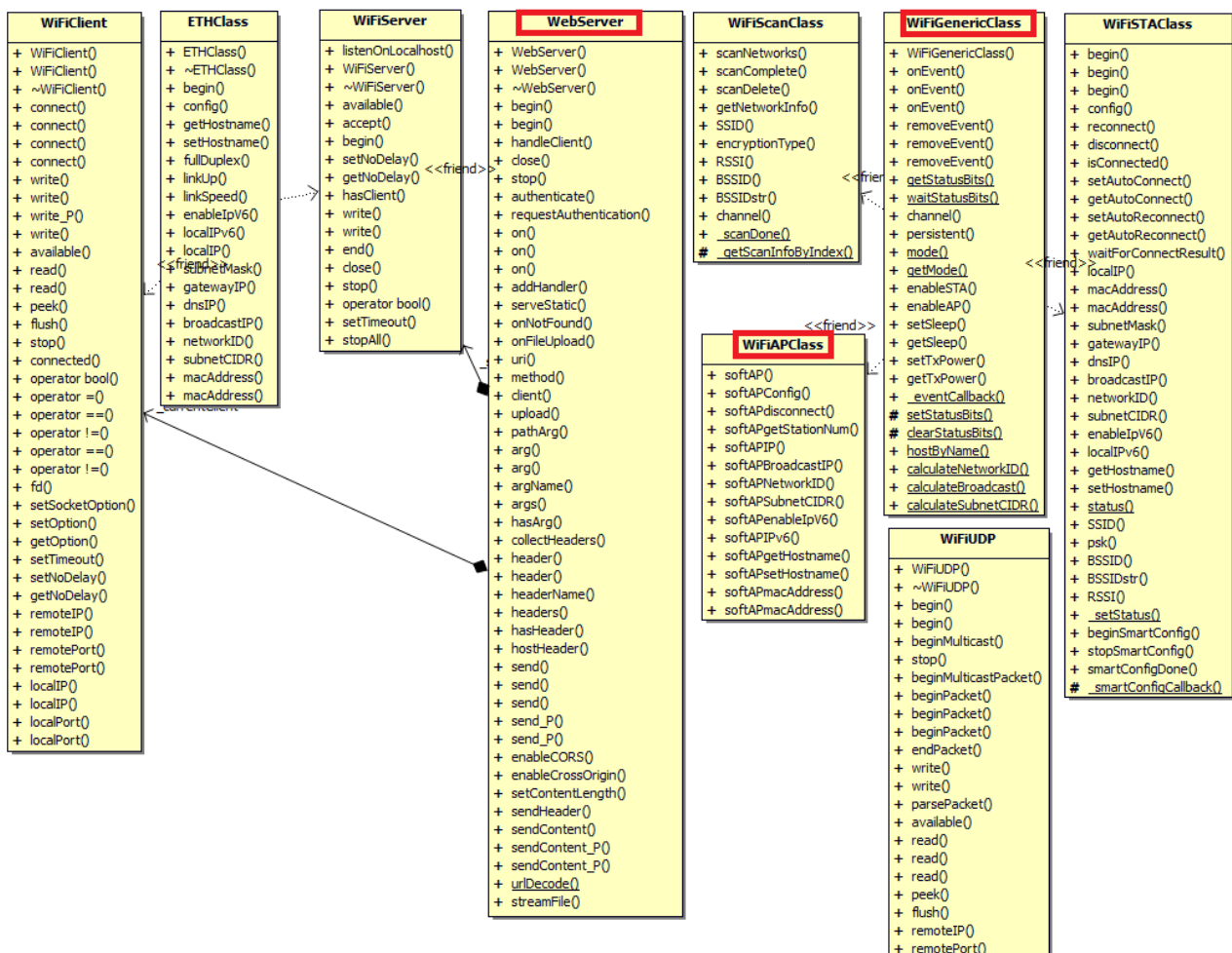
2.5.1- Conception du serveur web

Quand on regarde les exemples fournis sur la programmation d'un serveur web avec un ESP32, la liste des méthodes disponibles est parfois incomplète ou peut être confondue avec un ESP8266. Pour cela j'ai utilisé l'utilitaire Bouml afin d'avoir une synthèse des classes et des méthodes sur le wifi et le serveur web.

On peut trouver les fichiers h du serveur web, ainsi que pour le Wifi dans le répertoire d'installation suivant : esp32\hardware\esp32\1.0.3\libraries\

Dans le cadre du projet, j'utilise principalement les trois classes suivantes :

- WiFiAPClass pour la connexion wifi en point d'accès ;
- WiFiGenericClass (méthode mode) pour arrêter le WiFi quand on dépasse 2000 mètres ;
- WebServer pour la gestion de la page web.



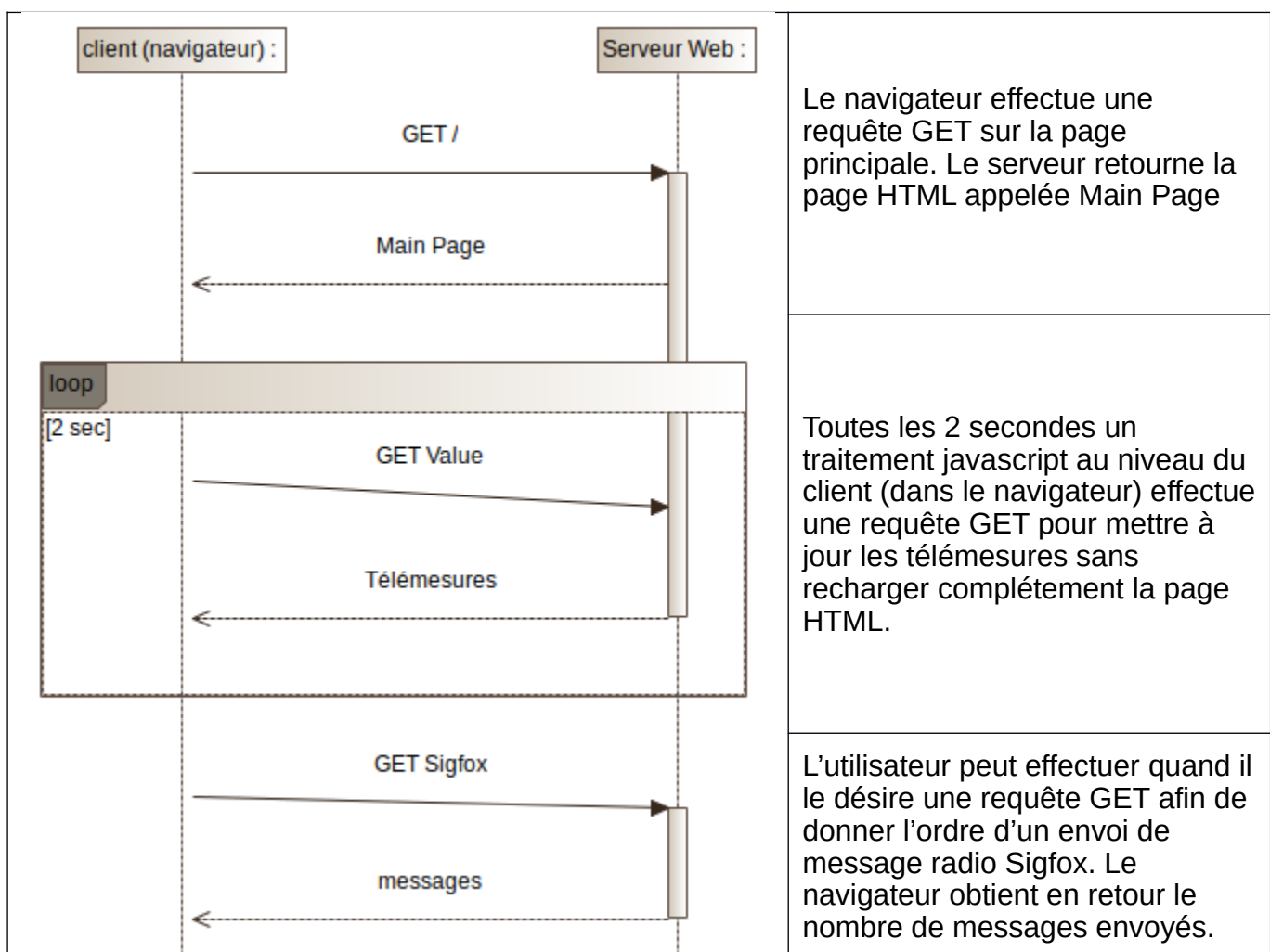
Toutes les méthodes ne sont pas utilisées, je ne vais détailler que celles que j'utilise.

2.5.2- Description du serveur web

L'ESP32 supporte la norme WIFI IEEE-802.11B, IEEE-802.11G, IEEE802.11N et dispose des bibliothèques permettant de réaliser un point d'accès sans fil et un serveur WEB.

Le serveur WEB situé dans l'ESP32 a pour objectif d'être certain que le système embarqué dans la nacelle fonctionne correctement. L'utilisateur se connecte au point d'accès WIFI géré par l'ESP32 avec un smartphone, et visualise l'ensemble des télémesures sur une page WEB. Ensuite on vérifie que l'envoi des trames sur le réseau Sigfox est opérationnel. L'ordre est alors donné à l'Aero-technicien de lâcher le ballon.

Le diagramme de séquence suivant montre les échanges de données entre l'utilisateur et le serveur web. Dès que l'utilisateur se connecte au point d'accès, celui-ci se voit attribuer une adresse ip 192.168.4.x/24 par le serveur DHCP de l'esp32. L'utilisateur n'a plus qu'à saisir l'adresse ip du serveur Web dans un navigateur (192.168.4.1).



Cette méthode permet de configurer un réseau protégé par mot de passe. Le premier paramètre de cette fonction est requis, les quatre autres sont facultatifs.

boolean WiFi.softAP(ssid, password, channel, hidden, max_connection)

- ssid : chaîne de caractères contenant le SSID du réseau (max. 31 caractères)
- mot de passe : chaîne de caractères facultative avec un mot de passe. Pour le réseau WPA2-PSK, il doit comporter au moins 8 caractères. S'il n'est pas spécifié, le point d'accès sera ouvert à tout le monde pour se connecter (max. 63 caractères).

La 2eme méthode que j'ai utilisée est pour obtenir l'adresse IP du point d'accès afin de l'afficher sur le moniteur série. Par défaut, l'adresse IP est toujours 192.168.4.1.

WiFiAPClass
+ softAP(in ssid : char, in passphrase : char = NULL, in channel : int = 1, in ssid_hidden : int = 0, in max_connection : int = 4) : bool
+ softAPConfig(in local_ip : IPAddress, in gateway : IPAddress, in subnet : IPAddress) : bool
+ softAPdisconnect(in wifiOff : bool = false) : bool
+ softAPgetStationNum() : uint8_t
+ softAPIP() : IPAddress
+ softAPBroadcastIP() : IPAddress
+ softAPNetworkID() : IPAddress
+ softAPSubnetCIDR() : uint8_t
+ softAPenableIPv6() : bool
+ softAPIPv6() : IPv6Address
+ softAPgetHostname() : char
+ softAPsetHostname(in hostname : char) : bool
+ softAPmacAddress(inout mac : uint8_t) : uint8_t
+ softAPmacAddress(in : void) : String

Pour la classe WebServer, j'ai utilisé le constructeur et les méthodes suivantes (marquées en rouge) :

WebServer
+ WebServer(in addr : IPAddress, in port : int = 80)
+ WebServer(in port : int = 80) ■
+ ~WebServer()
+ begin() : void ■
+ begin(in port : uint16_t) : void
+ handleClient() : void ■
+ close() : void
+ stop() : void ■
+ authenticate(in username : char, in password : char) : bool
+ requestAuthentication(in mode : HTTPAuthMethod = BASIC_AUTH, in realm : char = NULL, in authFailMsg : String = String("")) : void
+ on(in uri : String, in handler : THandlerFunction) : void ■
+ on(in uri : String, in method : HTTPMethod, in fn : THandlerFunction) : void
+ on(in uri : String, in method : HTTPMethod, in fn : THandlerFunction, in ufn : THandlerFunction) : void
+ addHandler(inout handler : RequestHandler) : void
+ serveStatic(in uri : char, inout fs : fs::FS, in path : char, in cache_header : char = NULL) : void
+ onNotFound(in fn : THandlerFunction) : void ■
+ onFileUpload(in fn : THandlerFunction) : void
+ send(in code : int, in content_type : char = NULL, in content : String = String("")) : void
+ send(in code : int, inout content_type : char, in content : String) : void
+ send(in code : int, in content_type : String, in content : String) : void ■
+ send_P(in code : int, in content_type : PGM_R in content : PGM_P) : void
+ send_P(in code : int, in content_type : PGM_R in content : PGM_R in contentLength : size_t) : void

WebServer(in port : int=80) : permet d'instancier un objet serveur web avec comme paramètre optionnel le port.

begin() : pour démarrer le serveur Web

on(in uri : String , in handler fonction) :void : effectue un lien automatique entre le nom de la page web et la fonction à appeler (on crée un événement automatique)

onNotFound(in handler fonction) :void : En cas d'erreur dans la programmation coté WEB, il est intéressant d'avoir un message d'information si le nom de la page demandée par le client (navigateur) n'existe pas.

Cette méthode envoie une réponse au client (navigateur) quand celui-ci va effectuer une requête.

send(in code : int,in content_type : String, in content : String) : void

Les paramètres sont les suivants :

- Le code [résultat http](#) de la requête normalement 200 ou 404 quand la page est non trouvée.
- Le type Multipurpose Internet Mail Extensions (type MIME) permettant d'indiquer la nature et le format d'un document dans le projet :
« text/plain »
- Le contenu du document à envoyer.

stop() : arrête le serveur

handleClient() : cette méthode permet de scruter les connexions entrantes et est appelée fréquemment . Si cette méthode est appelée de manière irrégulière, le serveur WEB aura des latences. Elle est placée dans la boucle loop() du programme principal. Cela est intéressant puisque l'application est majoritairement gérée par des tâches freeRTOS.

2.5.3- Réalisation du serveur WEB

Schéma algorithmique :	Fichier index.h
Instanciation du serveur WEB (server)	<p>Le fichier index.h contient les codes HTML, CSS, et JAVASCRIPT.</p> <p>Celui-ci est placé en mémoire flash afin d'économiser de la RAM. (PROGMEM)</p>
Initialisation (setup) Configuration de l'ESP32 en point d'accès (AP) Lien entre le nom des requêtes et les fonctions (server.on) Mise en marche du serveur.	
Fonction pageIndex() Envoie au client la page HTML complète située dans le fichier index.h	
Fonction pageMesures() Envoie au client les télémesures	
Fonction commandeSigfox() Enfile la demande d'émission Sigfox(queueEnvoiBouton) Envoie au client le nombre de messages émis	
Fonction handle_NotFound() Envoie au client le code d'erreur 404	

La particularité du programme est la mise à jour des télémesures. Lorsque le serveur effectue une requête GET (/Value) de mise à jour, les télémesures doivent être facilement identifiées. Le principe est de concaténer les télémesures en insérant le caractère « ; » entre chaque information.

```
server.send ( 200, "text/plain", String(carteSd.temperature) + ";" + String(carteSd.pression) + ";"
+ String(carteSd.humidite) + ";" + String(carteSd.impulsions / RAPPORT_USVH) + ";" +
String(position) + ";" + String(carteSd.altitude));
```

Quand le client reçoit la réponse, (this.responseText) le javascript effectue l'opération « split ». Les différentes télémesures se retrouvent dans le tableau « values ». Il n'y a plus qu'à utiliser la méthode getElementById pour effectuer la mise à jour des id de la page HTML.

Lorsque l'utilisateur appuie sur le bouton Sigfox dans le navigateur, le client envoie une requête GET (/Sigfox). A ce moment le bouton est désactivé pendant 20 secondes le temps que l'esp32 traite la donnée, envoie la trame par Radio, et que l'information soit disponible dans le backend Sigfox. Cette sécurité est nécessaire afin que l'utilisateur ne puisse pas envoyer trop de messages à suivre au vu de la limitation de 140 messages Sigfox par jour.

2.5.4- IHM de la page web



2.6- Fiche de test unitaire global

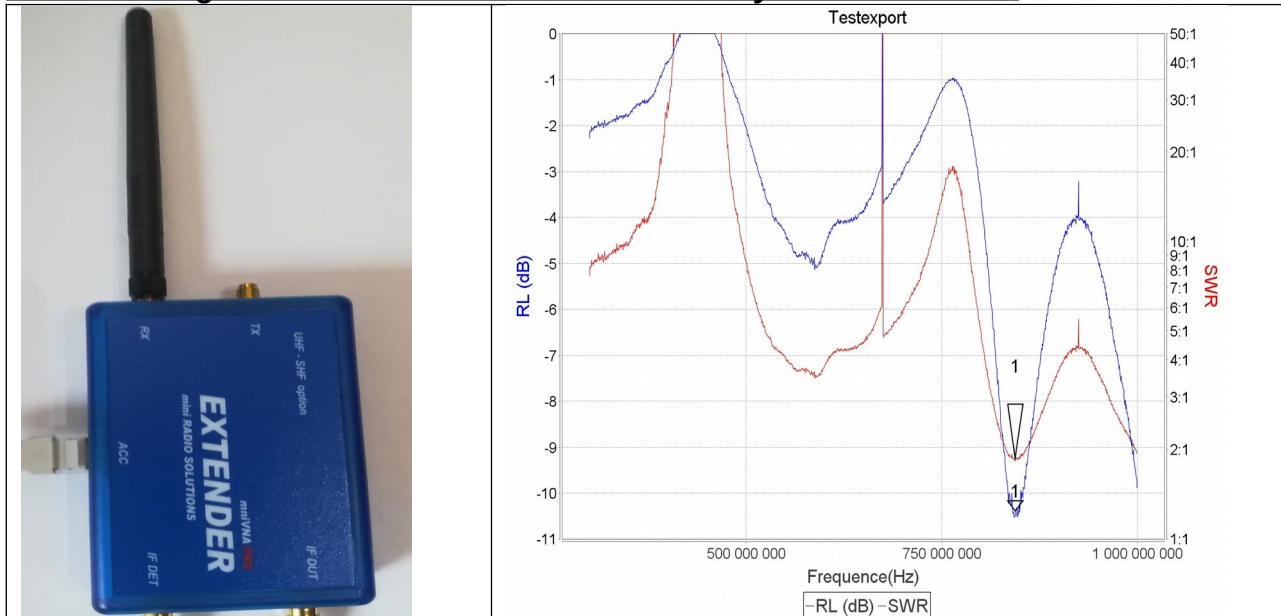
Fiche de tests			
Nature :	Fonctionnel	Référence :	
Module :	Classe		
Objectif :	Vérifier la fonctionnalité de l'ensemble du programme		
Condition du test			
État initial du module		Environnement du test	
Programme	Projet Ballon Sonde	PC avec Arduino / Netbeans	
Conditions initiales			
L'ESP32 est sous tension. La carte SD est insérée dans la l'ESP32.			
Procédure de test			
Repère	Opérations	Résultats attendus	
1	Vide le contenu du fichier de la carte SD	aucun	
2	Récupère les données dans les files sigfox et MSD	Affichage dans le moniteur série d'un message de vérification si les données sont dans la file sigfox et MSD	
3	Sauvegarde les données récupérées dans la carte SD	Affichage dans le moniteur série d'un message de vérification si les données sont bien enregistrées dans la carte sd	
4	Convertit les données récupérées de la file en hexadécimal	aucun	
5	Forge la trame avec les données converties en hexadécimal	aucun	
6	Émet la trame en hexadécimal	Affichage dans le moniteur série des données en hexadécimal	
7	Attend la réception du OK en provenance du module Sigfox	Affichage 'Message envoyé avec succès'	
8	Vérifie la réponse du backend sigfox	2020-03-19 16:49:34 211 b75241424f640a3f0221a900 Les données s'affichent bien dans le backend Sigfox	
9	Émet pendant 4 heures 30 minutes	Les données s'affichent bien dans le backend Sigfox	
10	Récupération de la carte sd	aucun	
11	Insertion de la carte sd dans l'ordinateur pour traiter les données	aucun	
12	Entre les repères 1 et 7, l'utilisateur peut forcer l'émission d'une trame sigfox	Le bouton envoi est désactivé pendant 20 sec	
13	Vérifie la réponse du backend sigfox	Les données s'affichent bien dans le backend Sigfox	

2.7- PHYSIQUE

2.7.1- Analyse de l'antenne quart d'onde fournie par Sigfox

L'antenne fournie par Sigfox est rigide. Or, le cahier des charges de Planète Sciences oblige l'utilisateur à embarquer une antenne filaire. Pour cela, il faut concevoir une antenne adaptée à la nacelle. Ma démarche consiste à analyser plusieurs modèles d'antennes et de choisir la plus pertinente.

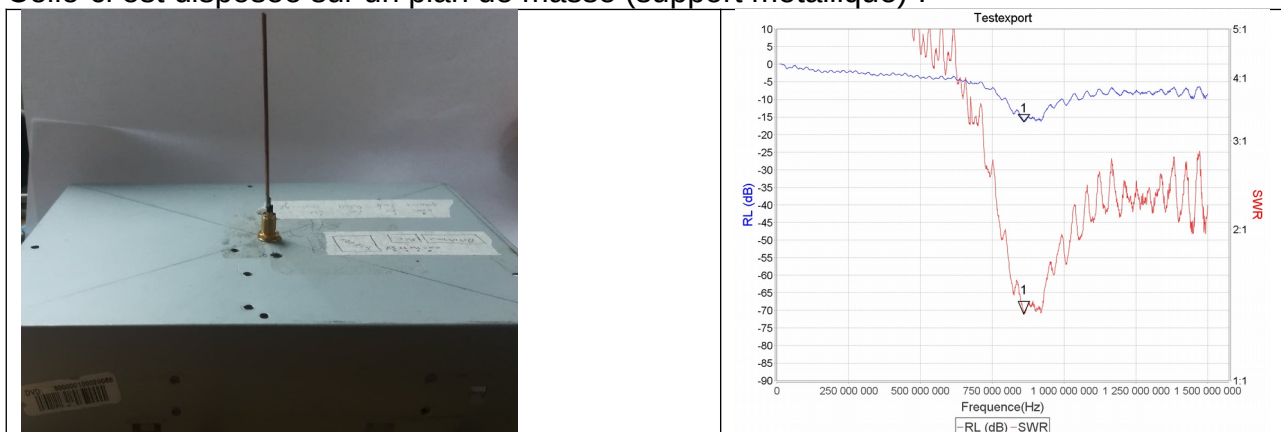
L'antenne Sigfox branchée directement sur l'analyseur d'antenne :



Le SWR est de 2 pour une fréquence de 868MHz sachant que l'antenne est accordée sur 840MHz

2.7.2- Antenne quart d'onde (brin rayonnant)

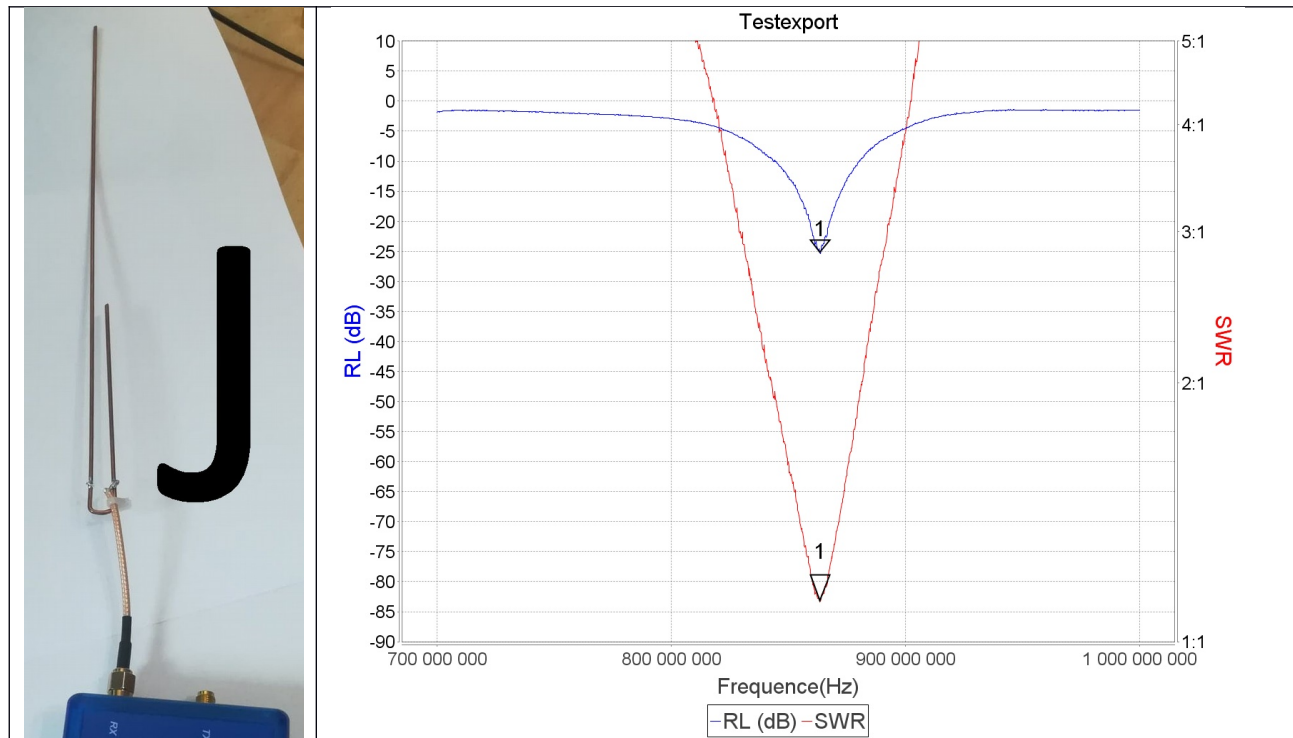
Celle-ci est disposée sur un plan de masse (support métallique) :



Le SWR est de 1,38 pour une fréquence 868MHz. La bande passante de l'antenne est comprise entre 802MHz et 986MHz. Cette antenne peut couvrir aussi la région des USA (915MHz). Le fait d'ajouter un plan de masse et de couper l'antenne à la bonne longueur, améliore le SWR par rapport à l'antenne fournie par Sigfox.

2.7.3- Antenne J-Pole

Cette antenne en forme de J est très sélective et elle n'a pas besoin de plan de masse. Son SWR est de 1,13. C'est l'antenne idéale pour tout objet connecté.

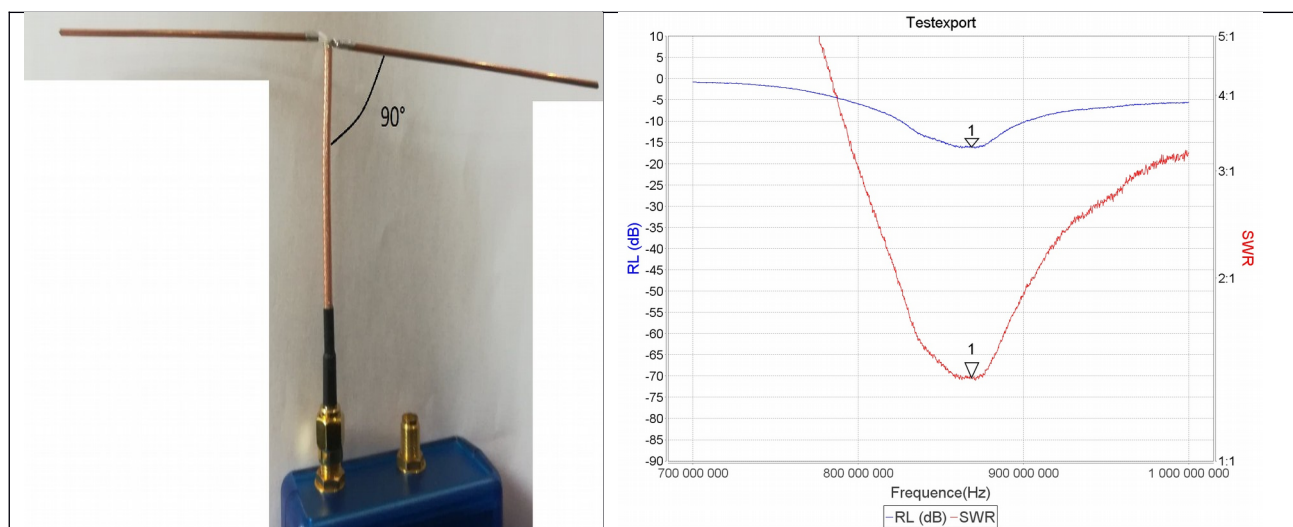


2.7.4- Antenne Dipôle

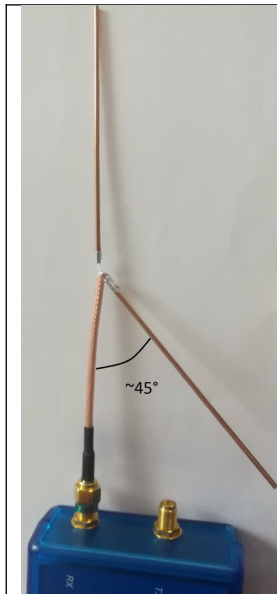
Cette antenne demi onde a un SWR de 1,37 pour une fréquence de 868MHz. La bande passante est comprise entre 826MHz et 900MHz.

Avantage : cette antenne a un gain de 2,14 dBi.

Inconvénient : son impédance est de 75 ohms, donc il sera difficile d'avoir un SWR de 1.

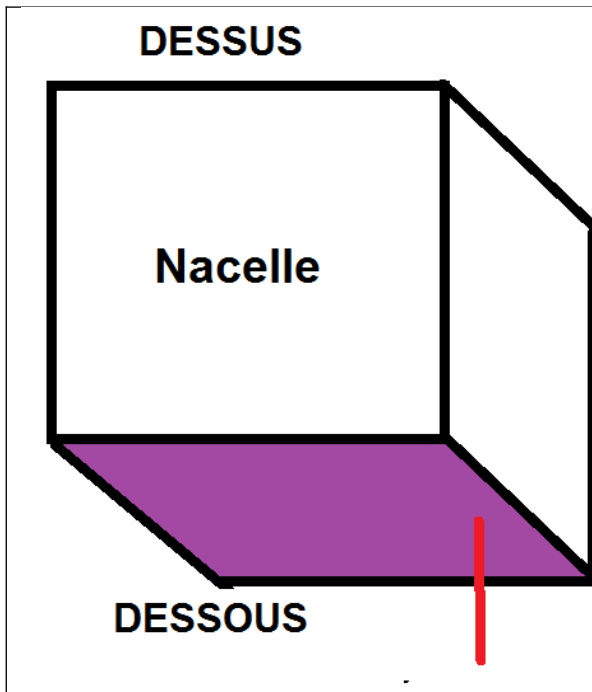


2.7.5- Antenne Dipôle en ground plane



En transformant le dipôle en ground plane, il est possible d'améliorer sensiblement le SWR. Ici, on a un SWR de 1,09. Pour avoir une impédance de 50 ohms, le radian doit être positionné à un angle de 45° par rapport à la verticale.

2.7.6- Antenne dans le ballon pour l'émission Sigfox

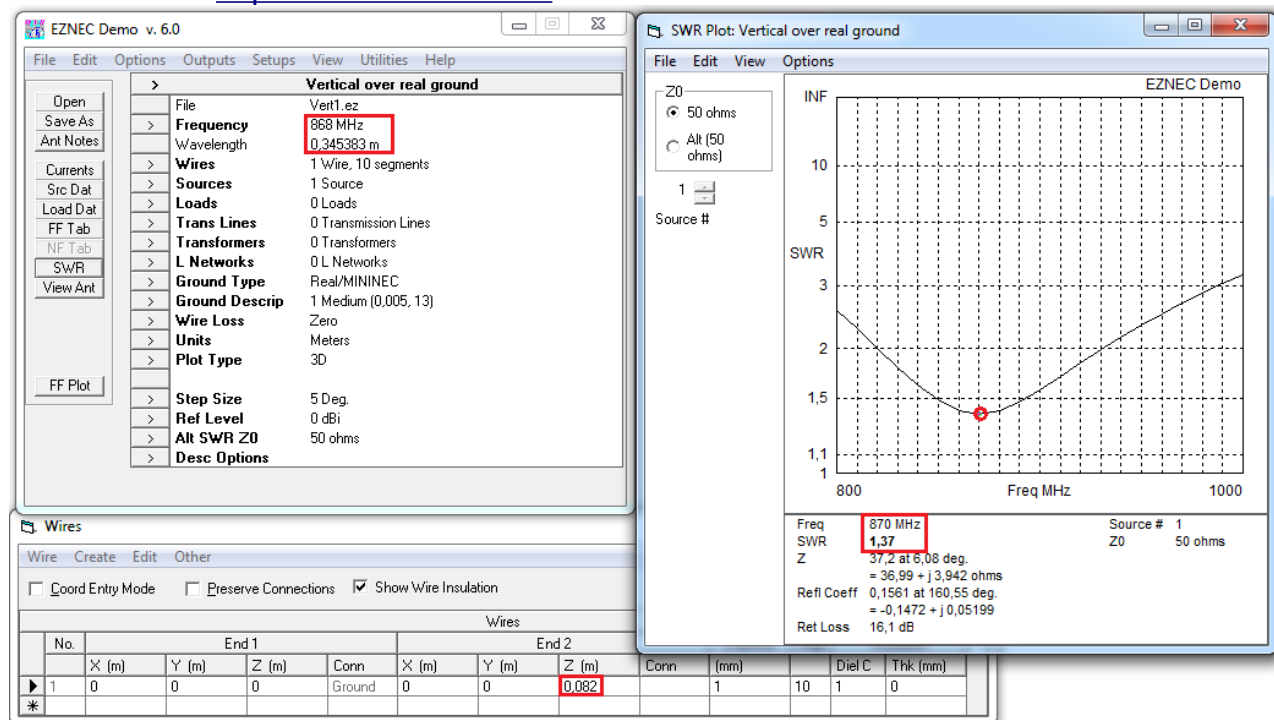


En fonction des contraintes de dimensionnement de la nacelle, j'ai choisi une antenne adaptée au ballon sonde. La nacelle est un cube de 30 cm de côté. En altitude le rayonnement doit se faire vers le bas. L'antenne est disposée sous la nacelle. Elle est composée de câbles souples afin de ne pas blesser quelqu'un à l'atterrissage et donc ne pas avoir d'éléments métalliques à l'extérieur. Afin d'assurer une meilleure isolation thermique, la nacelle est entourée d'une couverture de survie. Une couverture de survie est un élément conducteur et pourra être utilisée comme plan de masse. (En violet : le plan de masse, En rouge le brin rayonnant quart d'onde)

Étant donné que la nacelle est en rotation constante, il est impossible d'utiliser une antenne directive. L'utilisation du plan de masse permet de réaliser une antenne Ground plane.

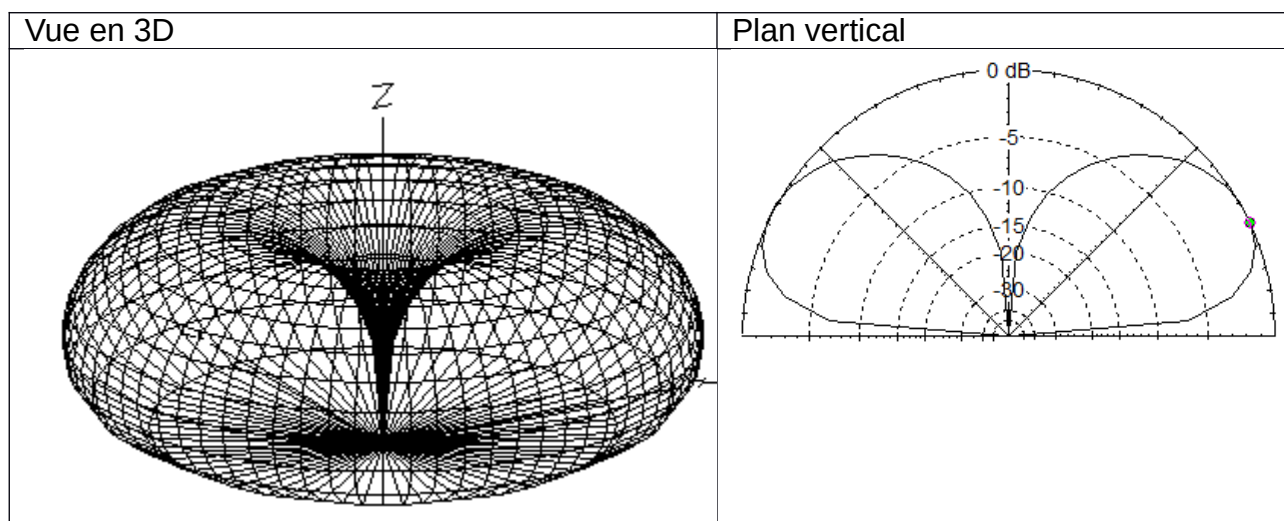
2.7.7- Diagramme de rayonnement

Le tracé du diagramme de l'antenne 868Mhz a été fait avec le logiciel de simulation EZNEC démo <https://www.eznec.com/>



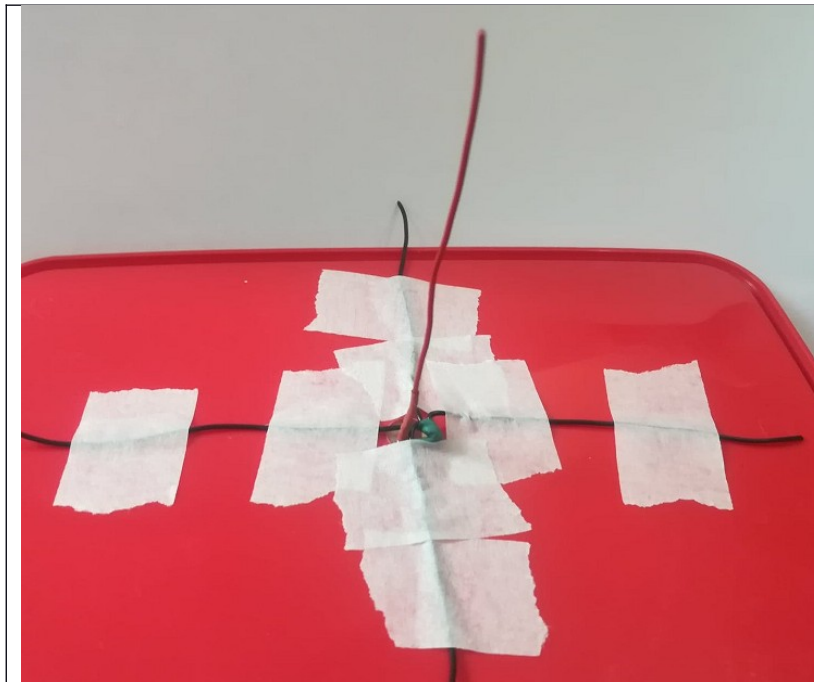
Après réglage du brin rayonnant à 8.2Cm, on remarque que le SWR est de 1.37 pour 870 Mhz

2.7.7.1- Tracé du diagramme de rayonnement

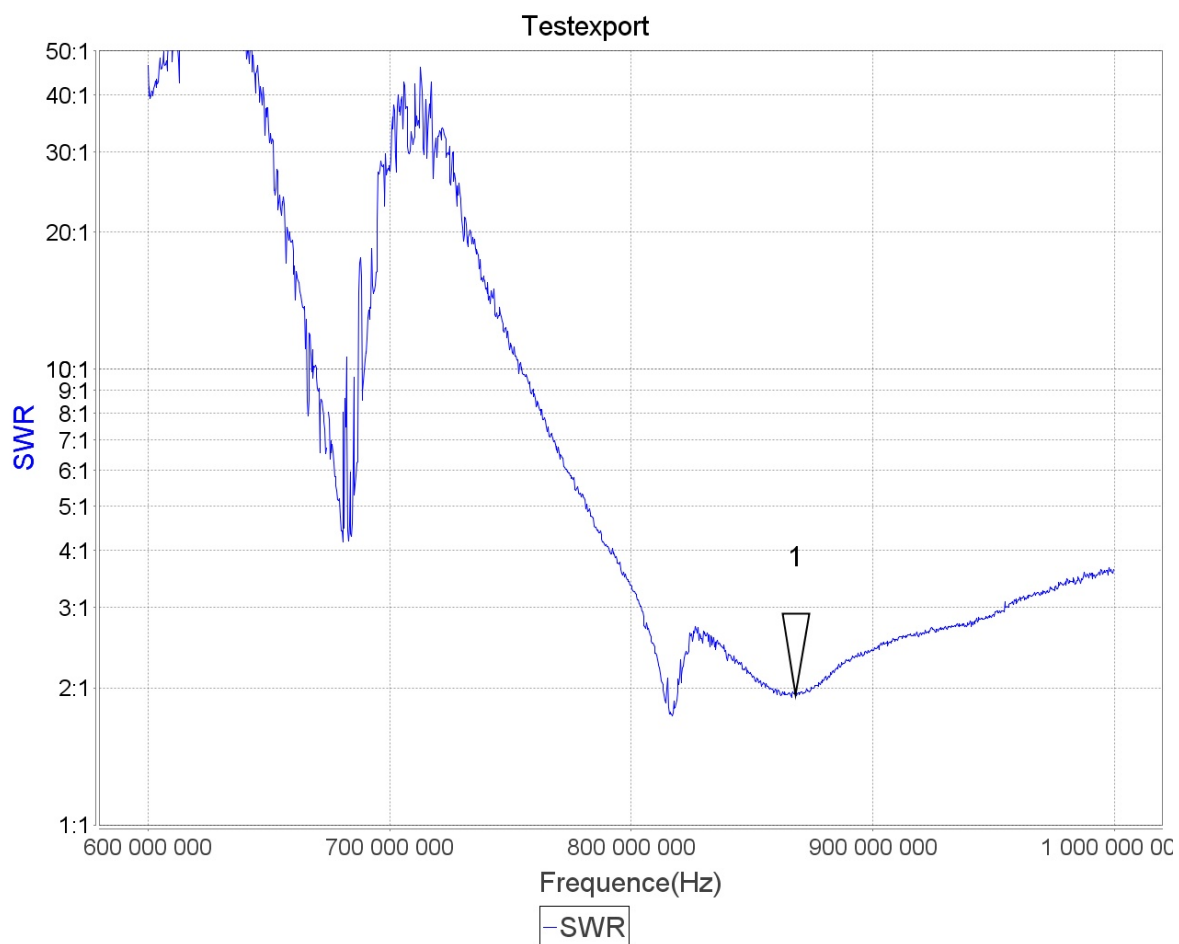


Le rayonnement est omnidirectionnel dans le plan horizontal. L'antenne étant placée sous la nacelle, les différentes stations de réception Sigfox pourront recevoir le signal. Le gain maximum correspond à une élévation de 25 à 30 degrés.

2.7.7.2- Essais préliminaires



Dans ce prototype le SWR est de 2 pour une fréquence de 868MHz. Un rappel des caractéristiques d'une antenne se trouve **page 40**.



3- CONCLUSION

Ce projet a été particulièrement intéressant. Il m'a permis de découvrir la programmation des tâches et des files avec freeRTOS dans un système embarqué, tout en gérant une page WEB avec JAVASCRIPT.

Il m'a également apporté une bonne expérience dans la gestion des objets connectés avec le prestataire Sigfox et l'utilisation de leur backend, la difficulté étant de trouver des solutions pour l'utilisateur en tenant compte de la limite d'octets transmissibles imposée par le prestataire.

Il est néanmoins regrettable de ne pas avoir pu envoyer le ballon sonde, au vu des circonstances actuelles de pandémie.

J'espère qu'il sera toutefois possible, pour une autre équipe, de tester le système embarqué lors d'un prochain vol stratosphérique.

4- ANNEXES

4.1- Définition d'une antenne

Une antenne permet d'établir une liaison radio depuis un émetteur vers un récepteur.

En émission, elle permet de transformer le signal électrique en une onde électromagnétique.

En réception, l'onde électromagnétique est transformée en signal électrique.



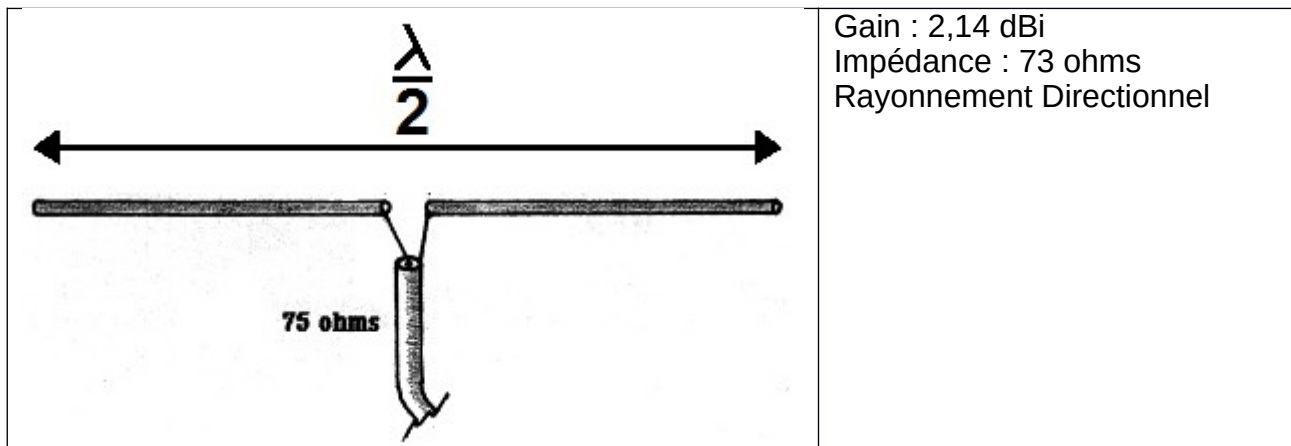
Les caractéristiques d'une antenne sont les suivantes :

- ...Sa forme mécanique (dimensions, poids, charge au vent...)
- Sa polarisation (verticale, horizontale...)
- Sa ou ses fréquences de fonctionnement (MHz, GHz...)
- Sa bande passante (MHz...)
- Son impédance (50, 75, 300 Ohms...)
- Sa directivité (ou son diagramme de rayonnement)
- Son gain (en dBi ou en dBd)
- Sa puissance maximale d'utilisation (en W)

Il existe des dizaines d'antennes différentes, parmi les suivantes :

Antenne ground plane. (radians perpendiculaire au brin rayonnant)	Antenne ground plane ou antenne 1/4 d'onde (radians inclinés de 45°)	Antenne Yagi (dite « râteau »)	Antenne Jpole
Gain : 2,14 dBi Impédance : 36 ohms Omnidirectionnelle	Gain : 2,14 dBi Impédance : 50 ohms Omnidirectionnelle	Gain en fonction du nombre de brins directeurs. Impédance : 28 ohms Directionnelle	Gain : 2,4 dBi Impédance : 50 ohms Omnidirectionnelle

Une autre antenne facile à réaliser est l'antenne dipôle :



La longueur d'onde de l'antenne se calcule avec la formule suivante :

$$\lambda = \frac{c}{f}$$

Sachant que la fréquence de l'émetteur Sigfox est de 868Mhz (Bande UHF), la longueur d'onde sera :

$$\lambda = \frac{3 \times 10^8}{868 \times 10^6} = 0,3456 \text{ mètre}$$

- formule simplifiée: $\lambda = 300/f(\text{MHz})$ soit $\lambda = 300/868 = 0,34562212$
- demi onde = $\lambda / 2 = 0,17281106$
- quart d'onde = $\lambda / 4 = 0,08640553$

En pratique, on applique un coefficient $K=0,95$ sur les dimensions des antennes soit :

formule simplifiée : $\lambda * 0,95 = 0,34562212 * 0,95 = 0,328 \text{ m}$

demi onde: $(\lambda/2) * 0,95 = 0,17281106 * 0,95 = 0,164 \text{ m}$

quart d'onde : $(\lambda/4) * 0,95 = 0,08640553 * 0,95 = 0,082 \text{ m}$

Le coefficient K dépend du rapport entre la longueur d'onde et du diamètre des brins qui sont utilisés pour réaliser l'antenne. http://f5ad.free.fr/ANT-QSP_F5AD_Valeur_de_K.htm
Pour accorder une antenne, il faut utiliser un analyseur vectoriel et ainsi recouper les brins à la bonne longueur.

Appareil de mesure utilisé : il s'agit du mini VNA Pro et de son module d'extension de chez mini RADIO SOLUTIONS.



miniVna avec l'extension UHF

Frequency range 200-1500 MHz

Calibration :

OPEN/LOAD/SHORT

Connector type : SMA

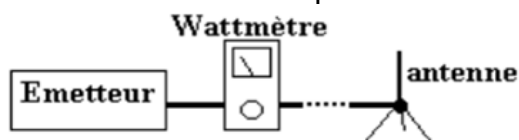
Output Power -10 dbm (aprox.)

Power consumption : 150 mA.

Cet appareil de mesure permettra d'accorder précisément l'antenne qui est disposée à l'extérieur de la nacelle.

Le ROS (ou SWR Standing Wave Ratio) est le Rapport d'Ondes Stationnaires exprimé par un chiffre sans unité de 1 à l'infini. Le ROS est un déséquilibre de l'impédance de la charge (antenne) par rapport à celle de la source (émetteur).

La formule est donnée par :

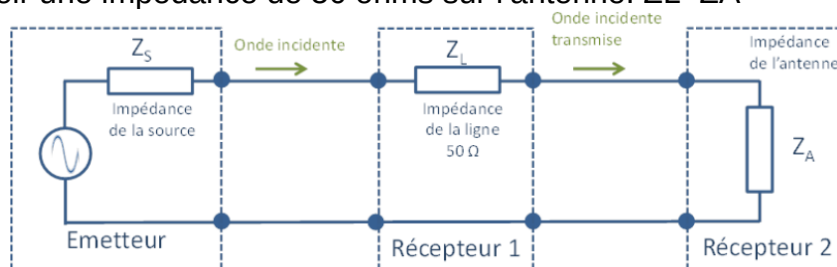


$$ROS = \frac{1 + \sqrt{\frac{Pr}{Pe}}}{1 - \sqrt{\frac{Pr}{Pe}}}$$

Pe: Puissance émise par l'émetteur en W

Pr: Puissance réfléchie retournant vers l'émetteur en W

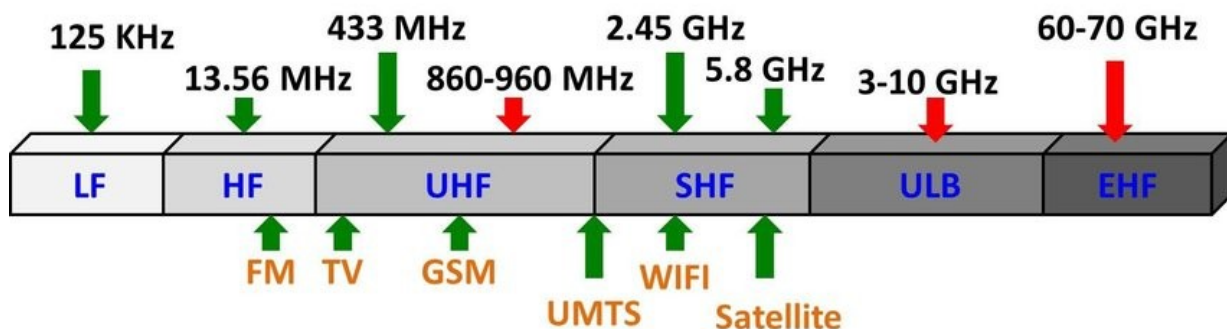
Dans le cas où on n'a aucune puissance réfléchie $Pr=0$, le $ROS=1$. C'est le cas idéal. On cherchera à avoir une valeur la plus proche de 1. En pratique on tolère un ROS max de 2. La puissance d'émission du modem Sigfox Breakout Sigfox BRKWS01-RC1 est de 14dBm soit 25mW. Afin de réaliser une bonne adaptation de l'émetteur sur l'antenne, on cherchera à avoir une impédance de 50 ohms sur l'antenne. $Z_L=Z_A$



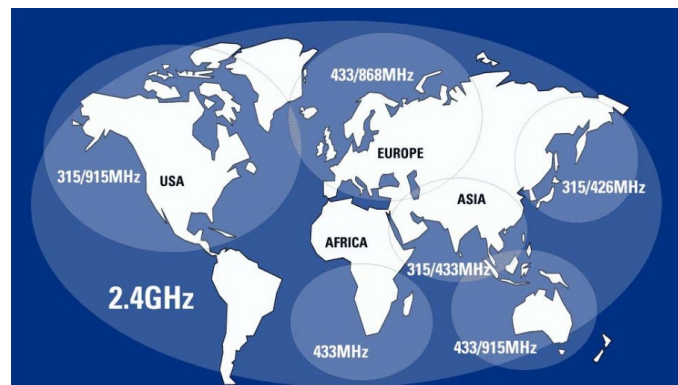
4.2- Pourquoi la fréquence 868Mhz du Réseau Sigfox ?

Sigfox a créé un réseau longue portée et à bas débit qui permet la communication de données de taille réduite entre les appareils connectés sans passer par un téléphone mobile.

Cette connexion à bas débit entre les objets connectés est possible grâce à sa technologie radio Ultra narrow band (UHF). Peu énergivore, elle utilise des bandes de fréquence libre de droit disponible pour le monde entier, comme les bandes ISM (Bande industrielle, scientifique et médicale). En Europe, il s'agit de l'ISM à 868 MHz. L'entreprise revendique une couverture de 92 % de la population française.



A noter que la fréquence de 868Mhz n'est valable qu'en Europe



4.3- Normes de la fréquence ISM

ISM Frequency Bands by World Region

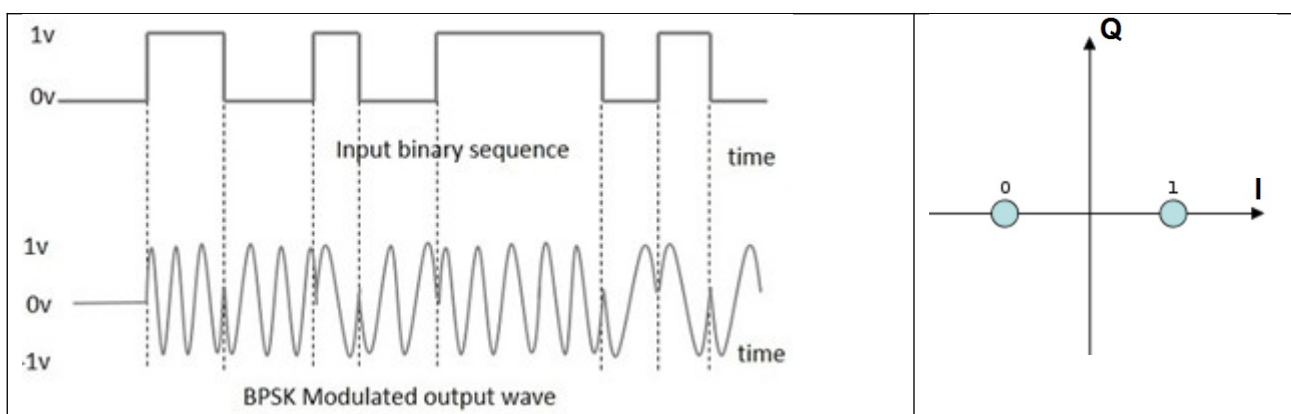
Region	Main Frequency	Regulatory Body and Standard
USA	902-928 MHz	FCC part 15.247
Europe	169 MHz and 868MHz	ETSI EN 300-220 / Wireless M-Bus
Japan	920 MHz	ARIB T-108
China	470-510 MHz	SRRC

https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/03.01.01_60/en_30022001v030101p.pdf

https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.02.01_30/en_30022002v030201v.pdf

Le phase-shift keying (ou PSK, soit « modulation par changement de phase¹ ») désigne une famille de formes de modulations numériques qui ont toutes pour principe de véhiculer de l'information binaire via la phase d'un signal de référence (porteuse), et exclusivement par ce biais.

Le BPSK (Bi ou 2-PSK : contient deux valeurs de phases possibles),



La portée et la qualité d'une liaison radio dépendent de très nombreux facteurs.

- Le RSSI : Received Signal Strength Indication (RSSI) Indication de l'intensité du signal reçu. Le RSSI est mesuré en dBm et est toujours négatif. Un RSSI de 0 correspond à une réception maximale
- Le SNR : Signal to Noise Ratio (SNR) ou Rapport signal/bruit. C'est le rapport entre la puissance du signal reçu et le niveau du bruit. Plus ce rapport est grand et plus il sera facile d'extraire le signal du bruit.
- La BW : largeur de bande. Le BW représente la largeur de la modulation de fréquence autour de la porteuse
- Le SF : Spreading factor (SF) : L'étalement de spectre représente l'excursion en fréquence utilisée pour transmettre un motif (généralement un octet). L'augmentation du Spreading Factor permet de couvrir une distance plus grande entre l'équipement et la passerelle au détriment de la bande passante disponible. Plus le motif est étalé, plus le débit baisse mais plus la portée augmente. (Cette technologie est utilisée dans le LoRaWAN Long Range Wide-Area Network principal concurrent de Sigfox)

4.4- Modulation D-BPSK

Sigfox utilise une modulation PSK (Phase Shift Keying). Le débit est de 100 bit/s et la bande passante est de 100 Hz. Le fait d'avoir un débit faible permet d'avoir un meilleur rapport signal sur bruit et ainsi d'augmenter la portée.

