2018
2019

Projet Drone Sous-marin

COMMUNICATION - ALIMENTATION

REBILLON ERIC

YNOV CAMPUS |89 Quai des Chartrons

Table des matières

Table des illustrations

Introduction

Pour introduire le sujet voici un résumé du cahier des charges pour mieux comprendre la tâche à réaliser :

Dans ce projet nous allons réaliser un drone sous-marin qui devras effectuer un déplacement fluide et omnidirectionnels dans l'eau et doit pouvoir gérer une profondeur de 10m maximum avec une vitesse de 4 nœud et donc doit pouvoir être commandé d'une portée de 15m maximum.

Le projet sera partagé en 5 tâches à réaliser :

1. Carte-Mère Drone sous-marin
2. Déplacement Drone sous-marin
3. Alimentation drone sous-marin et Communication entre Drone Sous-Marin et Télécommande
4. Carte-mère télécommande
5. Affichage de la caméra drone sous-marin sur l'écran télécommande

Pour ce projet j'aurai à gérer les besoins en tension et courant de chaque partie se trouvant sur le drone sous-marin et gérer la communication des flux de données entre le drone sous-marin et la télécommande mais le flux vidéo à envoyer à la télécommande ne sera pas géré dans cette partie mais dans la partie affichage de la caméra drone sous-marin sur l'écran télécommande.

Contrainte majeur de ce projet c'est l'environnement dans lequel le drone devra se mouvoir. La contrainte d'être dans l'eau va changer la communication car le sans-fil est impossible à gérer dans ce milieu car aucune fréquence aura une portée suffisante et vitesse suffisante pour faire communiqué en sans-fil car les hautes fréquences ont une portée de quelque centimètre donc seul les très basse fréquence passe dans ce milieu mais l'atténuation et d'autre facteur vont faire que la communication sans-fil sera impossible. Et la force qui sera exercer sur les moteurs sera plus importante dans ce milieu et pour le bon déplacement du drone il faut une stabilité parfaite, et plus important il faut que le drone sous-marin soit entièrement étanche pour ne pas abîmer les cartes électronique et autre systèmes étant à l'intérieure.
Ce milieu demande de prendre en beaucoup plus de contrainte et de précaution que les projet n'étant pas dans ce milieu.

Planning Prévisionnel

Ci-dessous les dates de début et fin de chaque tâches et le nom de la tâche à effectuer.

Nom	Date de début	Date de fin
Recherche et choix de solution	10/12/18	28/12/18
Prise en main et répartion des tâ...	20/11/18	10/12/18
Mise en place de la solution	28/12/18	13/02/19
Test et validation	13/02/19	13/03/19
Intégration	05/03/19	02/04/19

Figure 1 : Date planning prévisionnels

Diagramme représentatif :

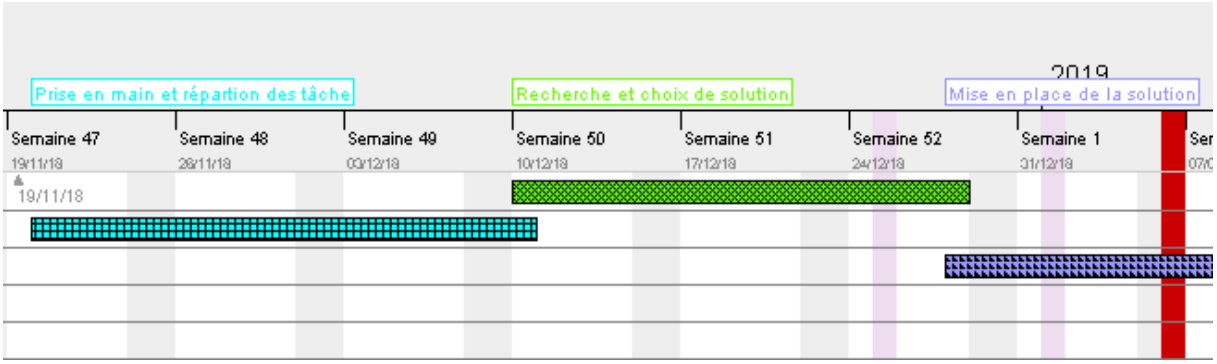


Figure 2: Diagramme 1 planning prévisionnels

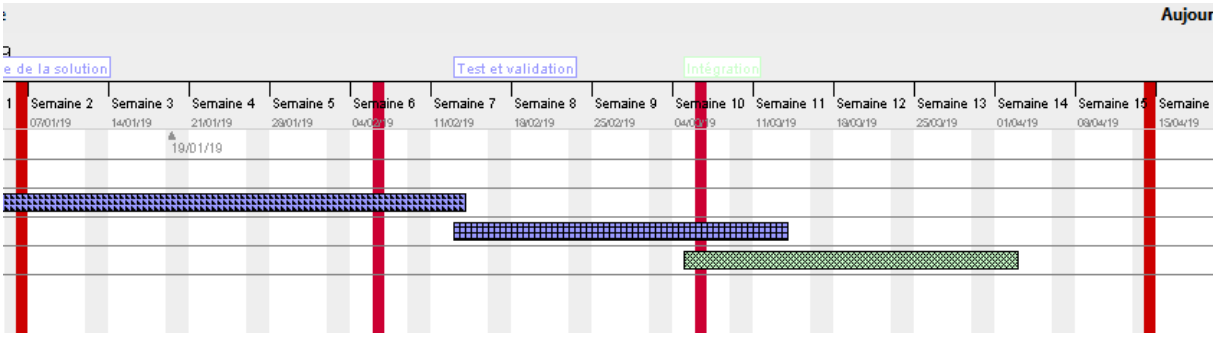


Figure 3 : Diagramme 1 planning prévisionnels

Sur ce planning la tâche la plus dure et longue à réaliser sera la réalisation de la solution choisit qui logiquement sera suivit de l'intégration de chaque partie mais la partie intégration doit pouvoir être testé rapidement et souvent car c'est la partie la plus sensible du projet car il aura la mise à l'eau du prototype ce qui peut tourner mal si l'étanchéité n'est pas complétée.

Communication

La partie Communication consiste à gérer les flux de données entrant et sortant entre la télécommande et le drone Sous-marin. Pour représenter cela :

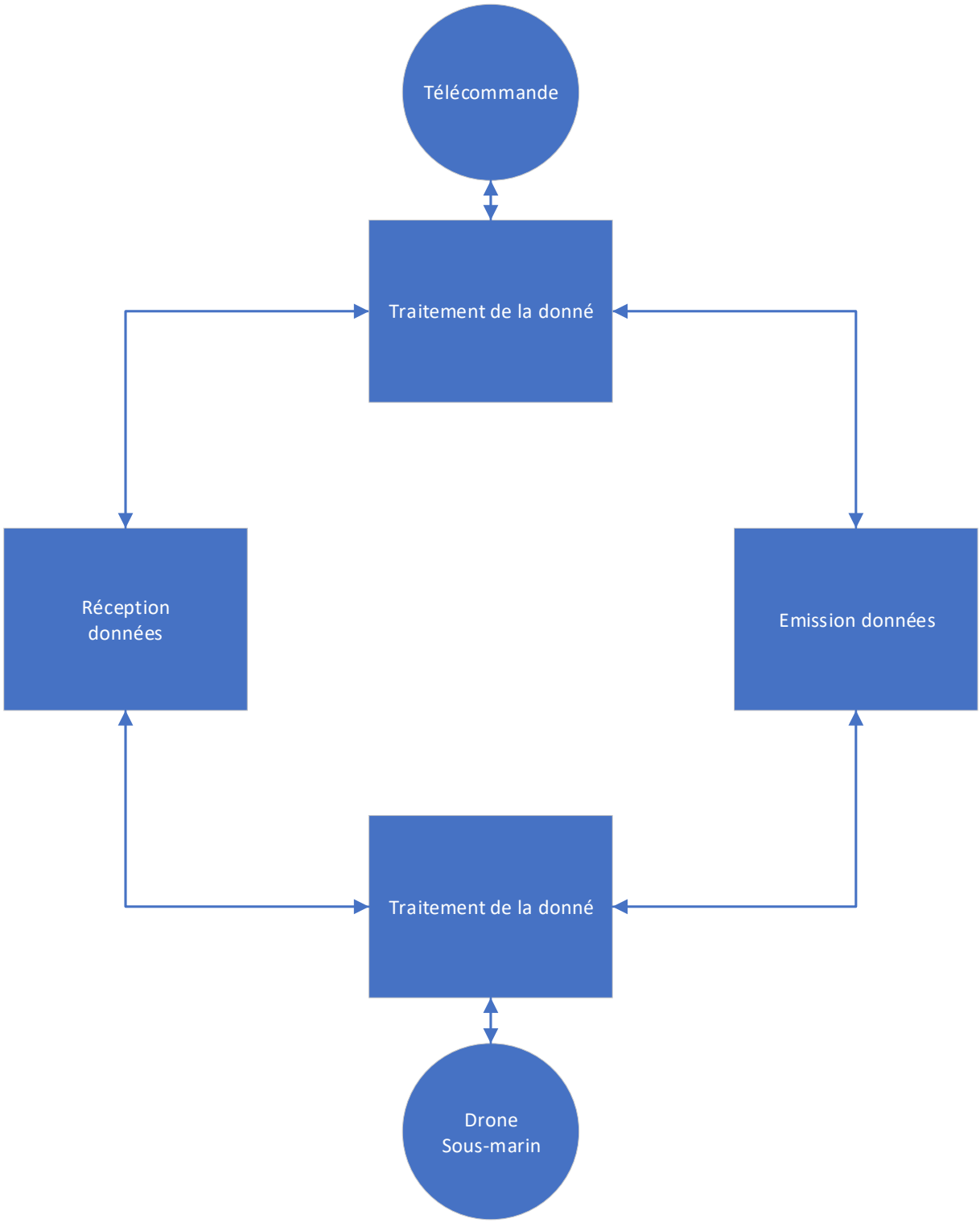


Figure 4: Diagramme Communication

Ce diagramme ci-dessus va permettre la visualisation de ce que j'ai à réaliser pour cette partie. Sachant que l'émission sera faite par le même module de transmission.
Le drone va transmettre les données à envoyer aux module de transmission, ces données qui seront stockés temporairement dans la partie traitement le temps de l'envoie, une fois que le module est prêt à envoyer alors l'envoie de la données stocké sera envoyer.
Si le module détecte une données reçues alors le module sera en mode réception est toutes les données seront stockés et traité puis envoyer au drone.
Le traitement données sera ce qui dira si on est en émission ou en réception et qui permettra l'utilisation de l'utilisation des données reçues. Sachant que la plupart des modules de communication peuvent à la fois transmettre et recevoir.
Voilà comment la communication fonctionnera grossièrement sachant que l'on a fait aucun choix de protocole et de comment réaliser faire cette communication.

Possibilités

Pour la communication deux possibilités :

1. De faire une communication filaire relié directement à un module extérieur à la télécommande et à partir du module de communiqué en sans-fil du module à la télécommande.

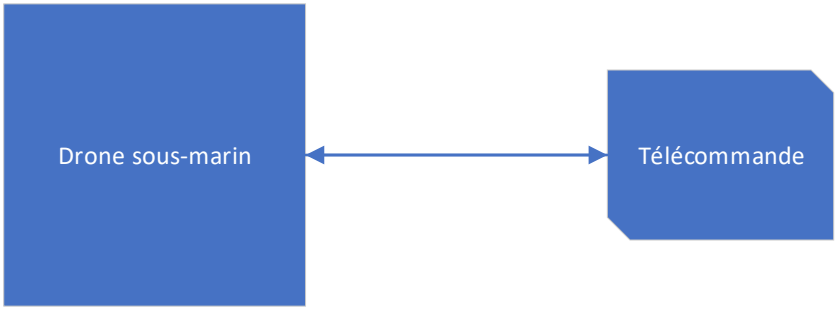


Figure 5: Solution n°1

2. De réaliser une sorte de bouée qui sera sur l'eau relié en filaire au drone et la bouée emmétra les flux de données en sans-fil jusqu'à la télécommande.

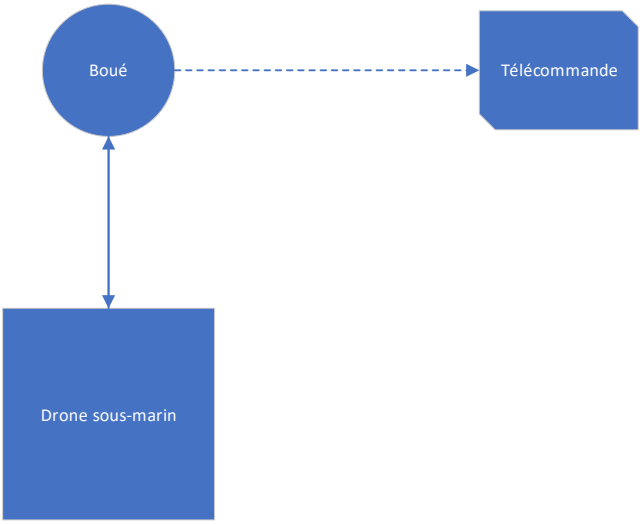


Figure 6: Solution n°2

L'avantage de la solution 1 c'est la simplicité de réalisation et un prix faible pour sa production mais inconvénient majeur de cette solution c'est qu'il faut avoir un câble extrêmement long et beaucoup de difficultés à le déplacé.

Pour la solution 2 apporte un avantage pas besoin de déplacer un long câble et plus longue portée, juste à mettre à l'eau la boue et on peut directement s'en servir. Inconvénients cela coutera plus chère que la solution 1 car cette solution a besoin de 4 module de communication.

La solution 2 me semble plus adapté plus facile à transporté,

Choix module radio

Caractéristique et Explication

Technologie possible existante en communication sans fil : module RF, module wifi, module Ethernet, module Bluetooth
Module Bluetooth et wifi ne pourras pas être utilisé pour les problèmes d'interférences
Module RF avantage de la portée
Module wifi et Bluetooth facilité de mise en œuvre mais interfère avec le module d'envoie vidéo.

Choix protocoles pour le module : UART ou SPI
Choix module radio en reliant le drone jusqu'à une bouée puis émettre jusqu'à la télécommande.
Module caractéristique : transmitter, receiver, transceiver
Ce définit par rapport aux données à envoyer. Sachant que des données devront être envoyer sur le drone pour le contrôler, de plus des données devront être envoyé sur la télécommande pour informer sur l'états général du drone. Donc choix de prendre un module transceiver qui pourra gérer à la fois la transmission mais aussi la réception de donner.
On aura deux flux de donnée (vidéo et celle de la communication) est donc pour éviter les perturbation on doit utiliser deux bande de fréquence différente, sachant que le module de transmission vidéo est en bande wifi soit 2.4GHz, pour la transmission de l'autre flux de données sera forcement sur une bande de fréquence plus faible.
Ont utilisé un câble de communication direct entre la bouée et le drone pour éviter les problèmes de transmission sous l'eau.
Pour le choix du câble il est impératif qu'il respect la norme IP68 car elle nous permet de savoir que ce produit peut résister a la poussière et l'eau jusqu'à une profondeur de 10m.
Pour rendre un câble Ethernet croisé étanche en plus de le prendre ip68 il faut rendre étanche la connectique du coté drone en s'assurant qu'elle est parfaitement étanche. Pour cela je prendre un coupleur étanche.
En ce qui concerne la lecture écriture prendre un module Ethernet et un module de communication RF SPI ou UART (avec convertisseur UART vers SPI) pour l'utilisation du câble derrière.

Caractéristique du module :
Mode transceiver (réception et émission)
Ressource disponible sur le module : 4 PWM, 5 CAN et 4 GPIO

Protocole UART => simplicité de mise en œuvre
Choix de la bande de fréquence inférieure à 2.4GHz pour éviter les problèmes d'interférences car l'émission de la vidéo se fait sur 2.4GHz
Choix de la fréquence
Porté entre les deux modules
Technologies arm : compatibilité avec la cartes STM32 du drone sous-marin et la Raspberry de la télécommande, la technologie arm sera obligatoire.

Module de communication possible

Module Rf	*Prix TTC en €*	*Constructeur*
RF EnOcean TCM 300	*28,29*	*farnell*
RFLORA	*18,32*	*Digikey*
ALPHA-TRX433S Emetteur-récepteur	*10,38*	*RS-online*
RF EnOcean TCM 320	*27,59*	*farnell*
Module RF LPRS eRIC-LoRa,868MHz	*25,14*	*RS-online*
Module ERIC4 transceiver radio 433MHz	*17,12*	*farnell*
RF Lora click transceiver module 433MHz et 868MHz	*52.07*	*RS*
SPIRIT click transceiver module 868MHz	*49.91*	*RS*

Figure 7: Tableau Communication

Les modules Lora et ERIC4 on l'avantage d'avoir un microcontrôleur intégré ce qui allégerai le calcul mais faudrait acheter une antenne de communication séparés mais sont en technologies microcontrôleur RISC est donc incompatible avec la STM32.
Il a la possibilité des modules SPIRIT et LORA click, utilisable par une STM32 et la SPIRIT en plus possède déjà l'antenne ce qui réduit le coût
Module choisi pour la communication RF :
Permet de réaliser une communication en protocoles UART(RS232) sur une fréquence de 433 et 868MHz. Avantage d'une portée de 15km et d'une consommation faible mais dur à mettre en œuvre.

Fonctionnement module

Module Présentation

Le module de communication choisit est le Lora Click de Mikroelektronika. Pour le présenter on va d'abord présenter le fonctionnement du protocole et puis la présentation des broches Module UART, le protocole du module est le suivant :

- Vitesse de transmission de données : 57600 BPS
- Largeur de la trame de données est de 8 bits
- 1 bits de STOP
- Module transceiver
- Pas de contrôle hardware

Sur cette photo on voit de quoi est composé le module :

- 2 connecteurs SMA pour deux antennes différentes (chaque émettent sur une bande de fréquence différente)
- Composant RN2483 réalisant la communication UART
- Composant TXB0106 réalisant une conversion de tension
- Et d'une LED d'indication de puissance.

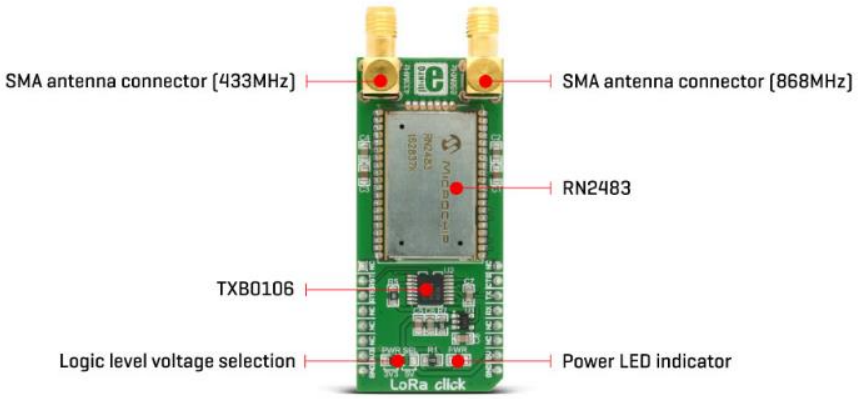


Figure 8 Lora Click en détaille

Pour comprendre le fonctionnement de ce module il faut aller voir comment fonctionne le module RN2483 pour cela nous allons voir de quoi il est composé.

Broche	RST	RTS	CTS	TX	RX	+5V	+3.3V	GND
Utilité	1	2	3	4	5	6	7	8

Figure 9 Tableau broche Lora click

- 1 : réinitialiser l'ensemble
- 2 : indique si le module est prêt à envoyer
- 3 : broche devant être réinitialiser pour permettre l'envoie de données
- 4 : broche de transmission de données
- 5 : broche de réception de données
- 6 et 7 : alimentation du module
- 8 : masse du module

Ces broches seront branchées sur des broches GPIO quelconque sur le microcontrôleur sauf pour TX et RX qui doivent être connectés à des broches GPIO qui sont indiqué comme appartenant à des broches UART du microcontrôleur.
Ces paramètres seront utilisés lors des configurations de l'UART sur la STM32, pour mettre en place une communication viable entre la STM32 et le module, si ces paramètres ne sont pas respecté le module ne répondra pas.

Commande processeur

Ci-dessous représente le RN2483 en détails.

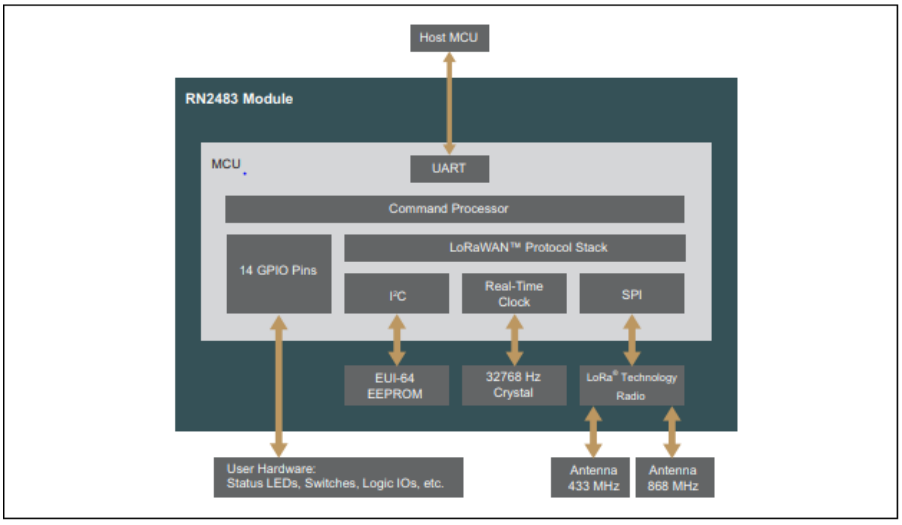


Figure 10 Module RN2483

Sur ce module réagissant au protocole UART nous pouvons remarquer un intermédiaire entre le matérielle et l'émission/réception en protocole LoraWan (protocole IOT de communication), et le microcontrôleur ce sont les commandes processeur, ce sont elles qui vont par l'intermédiaire de commande vont réaliser le protocole LoraWan est donc gérer les différentes phase d'émission/réception.
Ces différentes commandes vont nous permettre de contrôler le module. Pour mieux comprendre ce module, cette image ci-dessous.

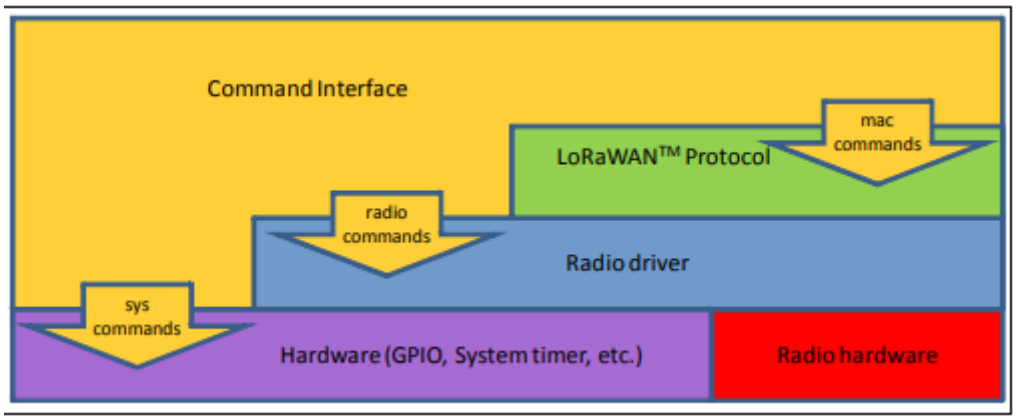


Figure 11 Commande UP du RN2483

Sur l'image ci-dessus, ont peut observer les commandes interfaces composé de 3 parties :

- Mac commandes permettant l'accès et la configuration du protocoles LoraWan
- Radio commandes permettant la configuration radio (fréquence, protocole, vitesse de transmission, …)
- Sys commandes permettant l'accès au différents matérielle présent (capteur, Timer, …)

Ces commandes permettent une sécurité sur l'accès et permettent une configuration plus rapide pour avoir un contrôle du module plus précis.

Pour réaliser ces commandes il faut les drivers fournit par Mikroelektronika qui vont nous permettre nous seulement de réaliser les commandes processeur du module mais aussi de proposer des fonctions d'envoie et de réception préfète pour avoir une utilisation plus simple du module. Ces drivers sont pour les cartes STM32L0, STM32L1, STM32L2, STM32L3, STM32L4 et nous utilisons une STM32F4 (pour les tests), et STM32VL. Donc les drivers ont besoin d'adaptation pour quelles soit utilisable par ces deux cartes.

Fonction préprogrammes et adaptation drivers

Dans les drivers utilisation de fonction qui ne sont pas programmés ou de redéfinition de fonction qui devait être corrigé.
Pour corrigés les problèmes de redéfinition de fonction il faut tout d'abord regarder la déclaration de la fonction en question puis regarder la différence avec la définition de la fonction est si ces deux fonctions déclarent des variables de type différentes alors il y a redéfinition de fonction.

Pour les fonctions préprogrammées il y a les fonctions HAL du module Lora click n'ont pas été codées et les délais.
Pour les délais ils existent trois possibilités dans mon cas :

- Délais manuels autrement dit créer les délais en utilisant un nombre de cycle processeur calculer par rapport à la vitesse de traitement du processeur et réaliser ce nombre de cycle en réalisant une boucle.
- Délais réaliser avec un timer qui comptera jusqu'à un évènement (évènement qui sera le temps du delais) pour pouvoir traitais la valeur donnée au timer comme temps definir des fonctions calculant ce nombre de cycle et qui prendra ce temps et la traduira en valeur brute pour le timer, valeur brute qui deviendra l'évènement de fin du timer
- Délais préprogrammés par les fonctions arm HAL qui pour faire fonctionner juste donner le temps à réaliser en ms

Pour notre programme nous allons d'abord utiliser les fonctions HAL qui sera plus simple à utilisé mais peuvent créer des problèmes lors de l'utilisation car l'importation de librairies HAL dans le driver peuvent touchés d'autre fonctions et créer des erreurs et puis le code dois pouvoir être utilisé sur d'autre carte donc les drivers doivent être portatif sur la STM32F4 et STM32VL Discovery donc nous allons préférer les délais manuel ou délais timer.
Pour les fonctions drivers HAL n'ont réaliser dans les drivers, c'est des fonctions HAL liées à l'utilisation de l'UART. Pour les définitions de ces fonctions il faut suivre les commentaires laisser par le constructeur qui dit les fonctions à réaliser.

Une fois ces différentes fonctions réaliser nous pouvons utiliser les fonctions préprogrammées par le constructeur et utilisé pour réaliser l'envoie et réception de données.

Par exemple ils existent en fonctions préprogrammés : lora_init, lora_tx, lora_rx.

Une fois ces fonctions validées nous pouvons utiliser les fonctions dans la main pour valider chaque fonction. Pour valider ces fonctions il faut lancer le programme pas à pas et valider ligne par ligne les fonctions.

Comme n'ayant pas terminé la partie validation ligne par ligne je vais expliquer la partie qui suis celle-ci. Avec les différentes étapes et les améliorations possibles.

Validation et test de la transmission et réception

La partie validation et celle qui envoie et reçois en boucle les données et l'amélioration en envoyant les données en interruption.

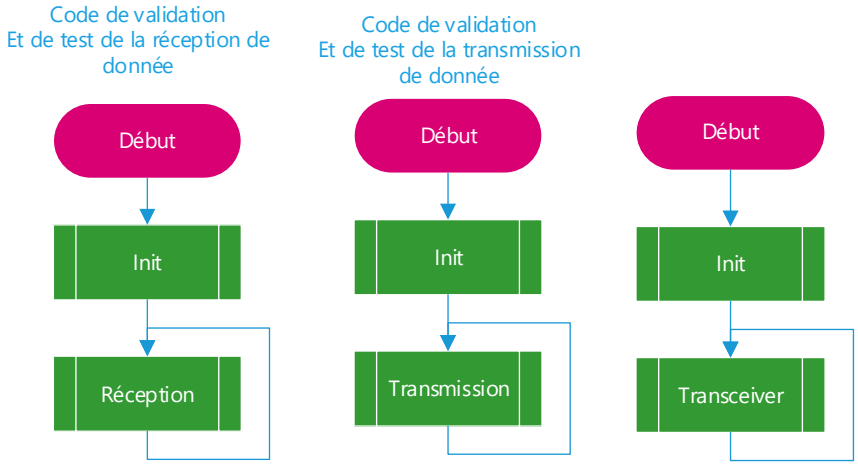


Figure 12 Test transmission et réception

La validation de la transmission me permet de confirmer un envoi de donnée.

On passera par l'intermédiaire des drivers en prenant les fonctions préprogrammes
Cette donnée devra être envoyée par le module RF Lora. Pour le processus transmission, on utilisera une librairie arm incluse dans toutes les cartes arm car elles sont dans les librairies standard, cette librairie est « HAL » permettant de faire appel à des fonctions optimisées et de réaliser un code portatif sur plusieurs cartes. L'appel de ces fonctions par l'intermédiaire des librairies permet par le simple appel à la fonction de gérer tout le buffer et de gérer l'envoi octets par octets. J'utiliserez Les librairie HAL dans les drivers Lora et adapter dans les fonctions de base préprogrammés pour l'utilisation du Lora.

Je répéterai le processus avec la réception et le transceiver (transmission et réception) avantage de la fonction HAL transceiver c'est qu'elle utilise le même buffer de données pour la réception et transmission.

Problèmes rencontrés

Lors des tests de fonctionnement, l'envoi ou réception de données à rencontrés des problèmes liés au driver qui ne sont pas adaptés à notre carte et les drivers de base du module ne sont pas complète elles ont besoin de quelque adaptation pour le fonctionnement du module. De plus le Lora détecte aucune commande est reste en mode « sleep » et donc envoie ou reçois aucune donnée.

Code transceiver en interruption

Une fois les problèmes résolus il faudra régler le code pour éviter l'envoie continue des données et envoyer en interruption pour envoyer les données contenues dans le buffer uniquement quand cela est nécessaire et de ce fait économisé les calculs du processeur. L'interruption permet de détecter si un envoi de données va être effectué ou si une réception est détectée, si c'est le cas alors l'interruption appellera la fonction mise dans l'interruption réalisant soit la transmission, soit la réception, et fera appel aux fonctions uniquement lorsqu'elle est utilisée.
Dans les librairies Lora pour le mode interruption, il existe des fonctions que l'on peut utiliser mais avec des adaptations de code et de l'initiation.

Drivers

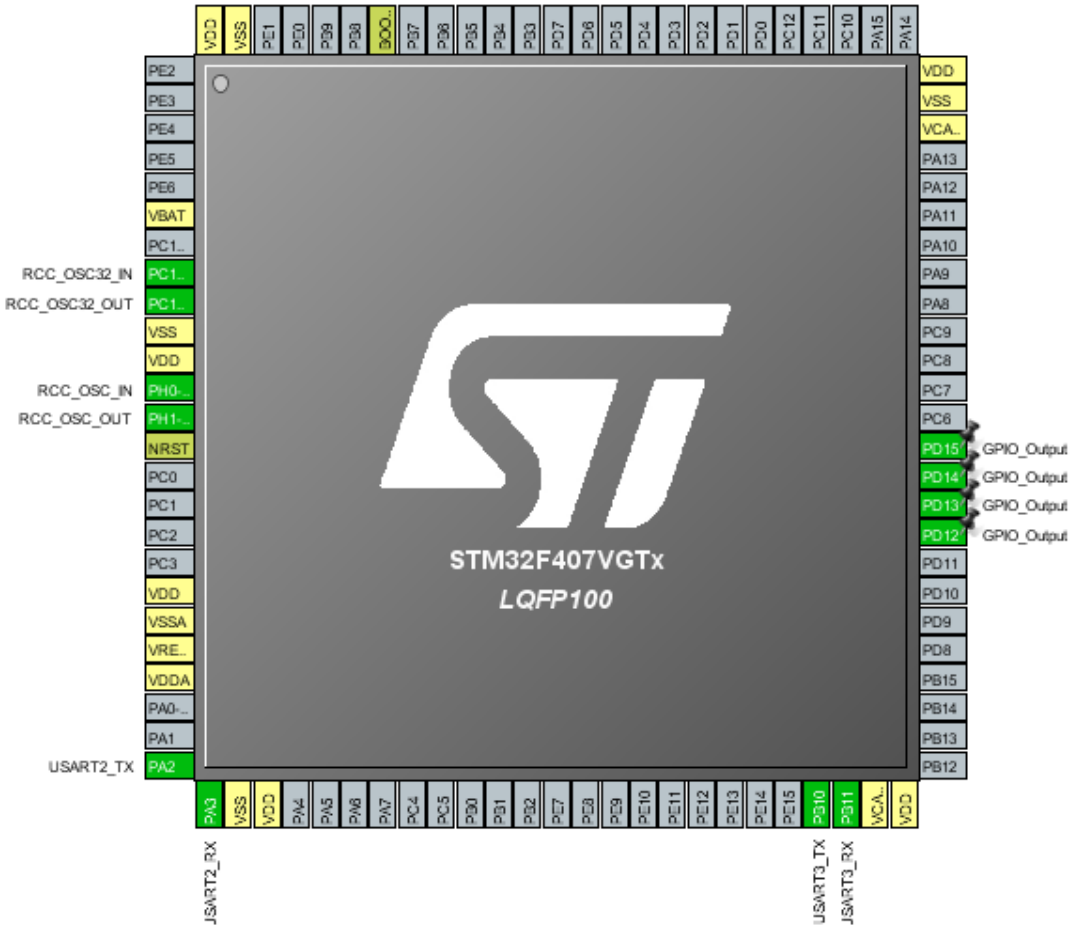
Les drivers mise à disposition par le constructeur pour mettre en place le module Lora Click rapidement ne fonctionne pas, et ne sont pas adaptés à notre carte. Ils existent de nombreux projets réalisant cela mais ils sont trop compliqués et pas adaptés à notre utilisation, comme un systèmes temps réel réalisé avec Lora avec son propre Kernel (MBed), mais cela est pas du tout adaptés à notre besoin.
Du coup nous allons partir de zéro sans driver pour le module et mettre en place seulement les commandes qui nous intéresse.

Pour commencer nous allons juste mettre en place une communication UART entre le module et la STM32. Envoyé les commandes mettant en place l'envoient et réception de donnée radio depuis la STM32 et attendre les réponses du module, s'il a bien envoyé les données.

Configuration de CubeMX pour initialisation du programme

CubeMX, logiciel permettant de réaliser toutes les configurations de chaque composant présent sur la carte, et permet de réaliser un code fonctionnel plus rapidement. CubeMX permet de gérer n'importe qu'elle carte en 32 bits, rapidement et de gérer facilement les composants et permet d'éviter de faire manuellement toute l'initialisation Et donc de gagner du temps.

Choix de chaque fonction et composant utile pour le projet

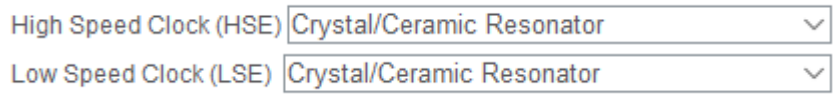


Pour le programme du module RN2481, il faut juste :

- USART2 : pour la visualisation sur un terminale de ce que l'on envoie au module et ce que le module nous envoie
- USART3 pour pouvoir communiquer avec le module
- Horloge
- Diode électroluminescente (optionnel) utile pour la visualisation de l'ensemble

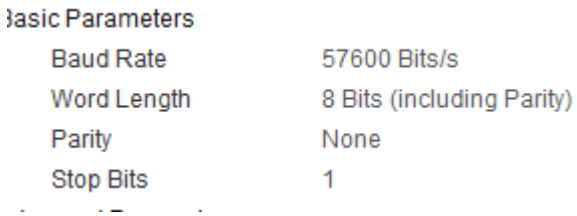
Pour les paramètres de l'USART, il faut d'abord se référer aux données constructrices, pour savoir la vitesse de communication et tous les autres paramètres qui seront utile à la mise en place du module de communication.

Pour les paramètres de l'horloge, il suffit juste de le configurer sur le Crystal présent sur la carte.



Ensuite il faut mettre en place la configuration des paramètres de l'UART 2 et 3 qui doivent les mêmes pour éviter les problèmes lors de la communication entre les deux.
D'après le constructeur le module aurait besoin de :

- 8 bits de donnée
- 1 bit de STOP
- Pas de parité
- Mode Transmission et réception
- Pas de Flux de contrôle matérielle



Une fois ces paramètres mise en place nous pouvons sélectionner la toolchain pour la compilation et le debug, elle permet de sélectionner avec qu'elle logiciel utilisé ces paramètres.

Dans notre cas nous prendrons SW4STM32 pour System Workbench STM32. Ensuite il faut générer le code et le logiciel génère les fonctions d'initialisation de chaque composant.

Pour voir les fonctions générer par CubeMX cf. Voir Annexe.

Une fois l'initialisation de la carte et de ces composants terminés nous pouvons passer à la partie programmation du module.

Algorithme du module de communication

Algorithme d'envoie globale

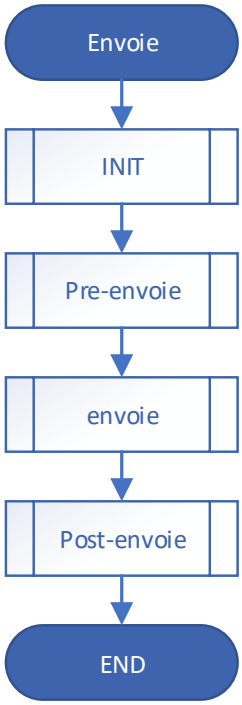


Figure 13 Algorithme d'envoie

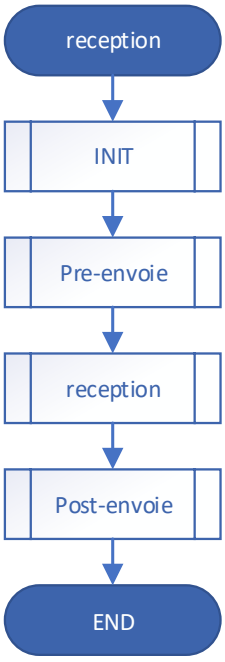
Le sous-processus INIT et indépendants des autres parties et ne va pas aider aux fonctionnements, cela permet par mesure de précaution de reset le module et d'avoir un module fonctionnel, exemple la commande mac après un long non fonctionnement retourne 0 de disponible sur la pile et le « sys reset » permet d'enlever se problèmes et donc pour valider un fonctionnement à chaque redémarrage de la carte et de l'ensemble du code on réinitialise le module.

La commande qui va réaliser cela est « sys reset » mais comme le module est en CRLF autrement dit retour à la ligne puis saut de ligne, il faut après chaque commande envoyé « \r\n » et ainsi le module comprendra qu'une commande à été envoyé et si le module ne reçoit pas cela, la commande ne sera pas vue.

Pour l'algorithme d'envoie ces simples, chaque commande radio doit être précède de « mac pause » et finir par « mac resume ».

Sous-processus Pre-envoie : « mac pause » permet de savoir combien de place dans la pile, le module possède et donc permet de savoir si le module peut réaliser la commande ou pas.

Sous-processus Post-Envoie : Ensuite l'envoie d'octets puis la commande « mac resume » permet de reprendre la fonctionnalité de la pile, ce qui permet de reprendre une fonctionnalité normale. La commande réalisant l'envoie radio est



Sous-processus envoie : « radio tx <data> suivit de « \r\n », pour la donnée il faut que le caractère qui sera envoyer soit dans le tableau ASCII.

Une fois cela terminé, l'envoie est terminé.

Pour une réception de donnée cela revient à faire la même chose en changeant la commande, pour une réception la commande sera « radio rx 0 » toujours suivit de « \r\n ».

Algorithme de réception globale

Figure 14 Algorithme réception

Pour la réception, il suffit de refaire la même chose, appeler les deux processus : pré-envoie et Post-envoie.

Comme la commande de réception utilise les commandes radio aussi il faut utiliser le même procéder autrement dit faire une commande avant et une autre après l'utilisation de la commande radio.

Algorithme de pré-envoie

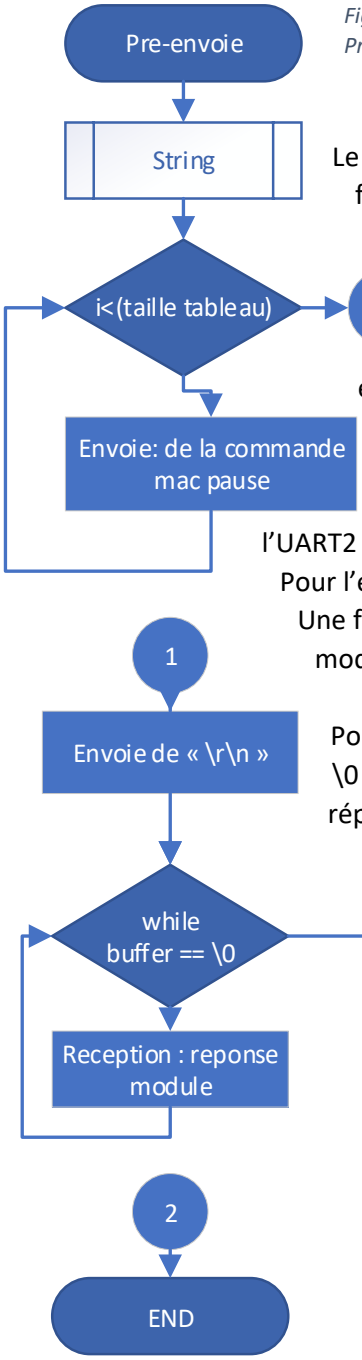


Figure 15 Algorithme Sous processus Pre-envoie

Le sous-processus string permet de vider tous les tableaux utilisés au cours de la fonction précédente. Et permet d'éviter un tableau contenant des caractères restant de la précédente commande et évite les erreurs de réception et l'affichage de caractères en plus.

Pour l'envoie de chaque commande vaut mieux l'envoyer caractères par caractères par intermédiaire d'un tableau de caractères pour éviter les décalages et les caractères qui changent voir même suppression de caractères. Du coup pour réaliser cela il faut faire une boucle est tant que le dernier caractère contenu dans le tableau ne sera pas envoyé la boucle continuera. Dans la boucle on réalise un envoie du caractère à la fois sur l'UART2 pour la visualisation et sur l'UART3 pour l'envoyer au module.

Pour l'envoie, on passe par l'intermédiaire des librairie HAL

Une fois la commande envoyée, on attend la réponse du module est tant que le module ne répond pas, le programme attendra.

Pour réaliser cela on fait un while est on teste si le buffer de réception est égal à \0 ce qui revient à tester si le tableau est vide alors le programme attendra la réponse du module. Pour recevoir la réponse du module il faut savoir combien de caractères va contenir la réponse du module.

Une fois la réponse du module reçue on affiche cette réponse sur l'UART2 et la commande de pré-envoie est valider.

Une fois la commande envoyer nous passons à la commande d'envoie radio.

Algorithme de post-envoie

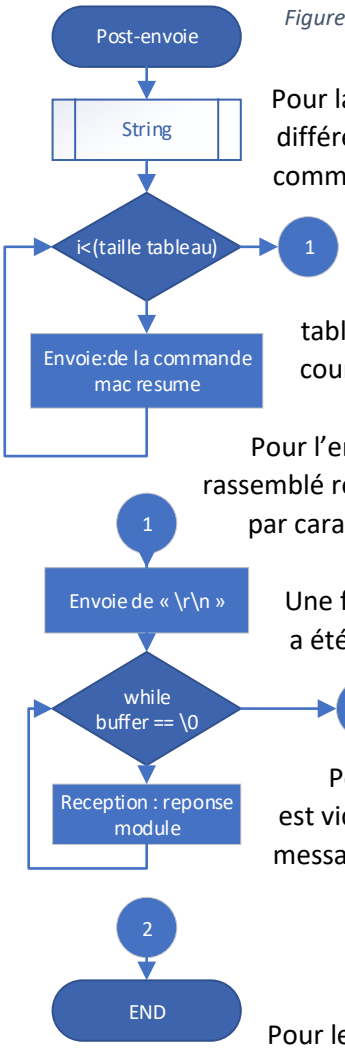


Figure 16 Algorithme Sous-Processus Post-envoie

Pour la commande post-envoie cela revient à refaire la même chose à la différence du nombre de caractères à envoyer car ce n'est pas la même commande donc pas le même nombre de caractères à recevoir car le module ne répondra pas la même chose.

La fonction string supprime tous les caractères contenus dans chaque tableau utilisé dans les fonctions précédente et réutilisé dans la fonction en cours de fonctionnement.

Pour l'envoie de la commande on stocke les caractères dans un tableau qui rassemblé représentera la commande. Ce tableau de caractères est envoyé caractères par caractères au module.

Une fois cela réaliser on envoie « \r\n » pour dire au module qu'une commande a été envoyer puis on attend sa réponse.

Pour l'attente de la réponse du module il faut tester si le buffer de réception est vide si cela est le cas alors nous restons dans la boucle sinon c'est que le message est reçu et que nous pouvons passer à la suite.

Pour les fonctions post et pré-envoie doivent être réaliser dans des fonctions sépares de celles d'envoie car chaque commande doit avoir sa fonction car sinon cela risque de perturber la réception des réponses modules.

Algorithme de la fonction envoie

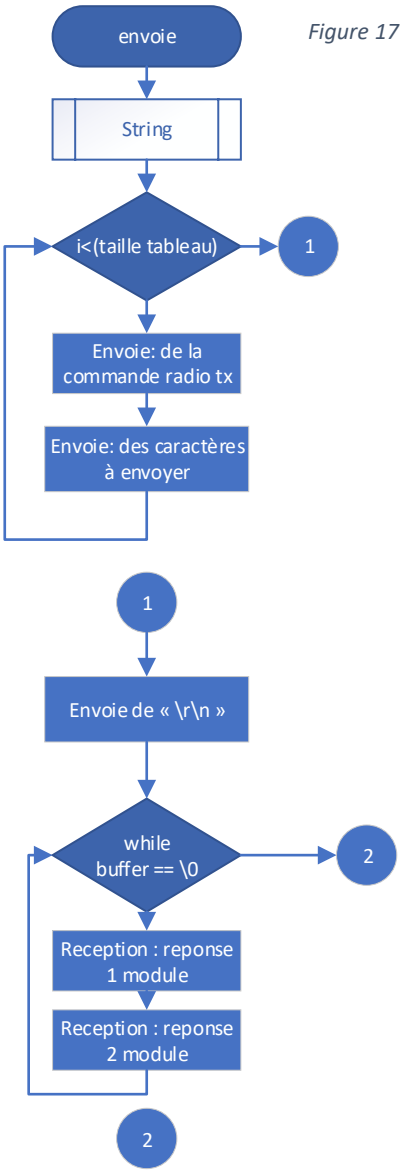


Figure 17 Algorithme d'envoie

Toujours réinitialisation de tous les tableaux réutilisés.

On stocke la commande d'envoie dans une chaîne de caractères puis envoie caractères par caractères de la commande au module puis une fois cela terminé on envoie les caractères mis en argument dans la fonction et pour valider la commande on envoie « \r\n ».

Si la commande est envoyée et valider, la première réponse sera « ok » sinon si la commande n'est pas comprise le module envoie « invalid_param ».

Pour la deuxième réponse la commande sera terminée et le module nous préviens qu'il a terminé d'envoyé, la donnée est envoyée si le module répond « radio_tx ok ».

Au lieu d'attendre les 2 réponse séparément on attend la première et on attend la deuxième et la condition pour sortir de la boucle d'attente réponse on attend que la deuxième réponse du module soit stockée dans le deuxième buffer.

L'envoie et la réception sont très similaire à la différence que la deuxième réponse sera reçue que si le module aura reçu une donnée.

Relevés de la communication avec le module.

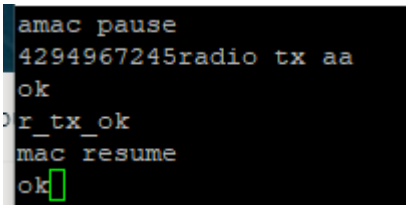


Figure 18 relevés émission

Tous les algorithmes d'envoie rassemblé donne ce relevé. Relevé qui nous font observer les bonnes réponses avec un envoie des commandes parfaites à l'exception de la remise à la ligne qui n'a pas été reçue par le buffer de réception des réponse modules. Réponse du module qui nous confirme que la transmission de données a été faite.

Algorithme de réception

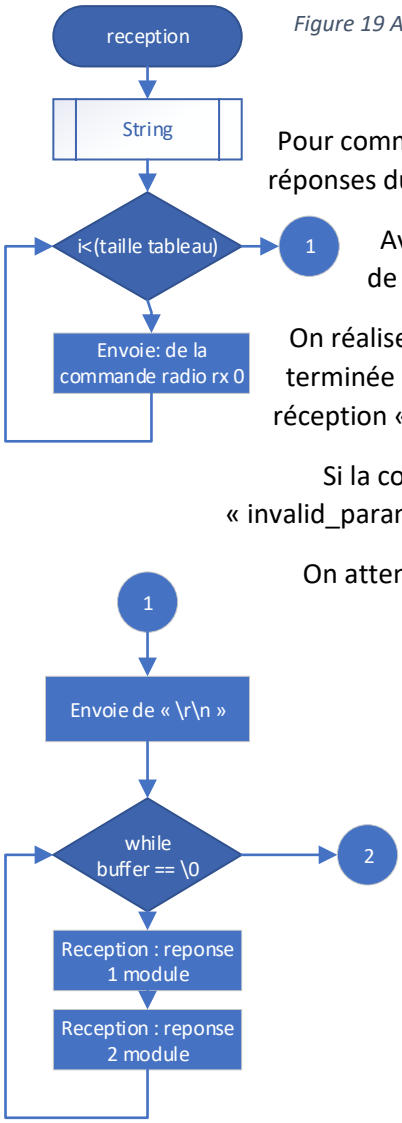


Figure 19 Algorithme de réception

Pour commencer, on réinitialise les buffers de réception qui stocke les réponses du module.

Avant cela on met dans un tableau chaque caractère de la commande de réception.

On réalise une boucle sur la taille du tableau est quand la boucle est terminée on envoie « \r\n » pour valider l'envoie de la commande de réception « radio rx 0 ».

Si la commande est validée la première réponse reçue sera « ok » sinon « invalid_param ».

On attend la deuxième réponse, si elle est reçue on sort de la boucle.

Le différent algorithme permette de réaliser une communication entre deux modules.

Relevés communication module pour la réception

```
mac pause
4294967245radio rx 0
ok
radio_err
mac resume
ok
```

Ces relevés montrent les réponses à la réception, pas de module connecté. La réponse du module dit qu'il ne reçoit aucune donnée ou qu'il y a une erreur lors de la réception. Problèmes de connexion entre les 2 modules car la réception pour qu'elles fonctionnent doivent être fait simultanément sinon cela ne nous dit pas de données reçues.

Module filaire Drone – Bouée

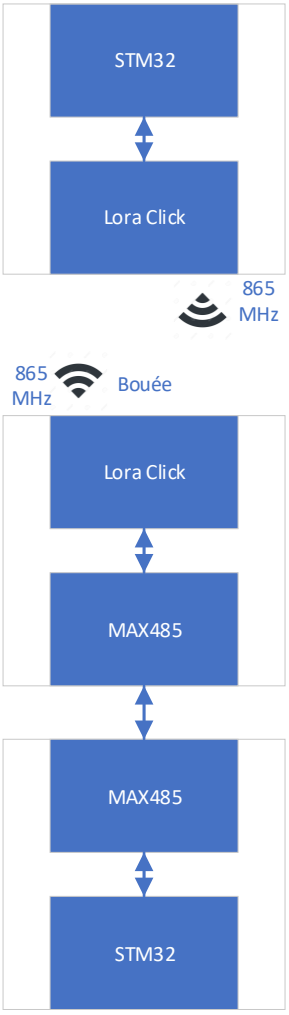
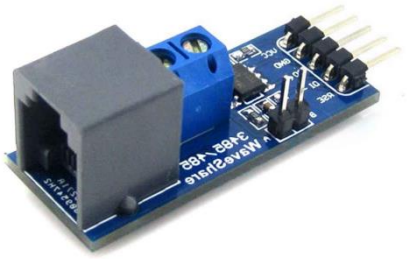


Figure 20 Communication Filaire avec module MAX485

Pour réaliser la liaison filaire entre le drone et la bouée, nous prenons un module qui permet de passer le protocole UART en filaire. Avec un porté supérieur à celle que nous allons utiliser.
Le module utilise une connectique RJ45.
Voici Le MAX485 :



Il peut communiquer jusqu'à une vitesse de communication max de 2,5Mbps sachant que le module Lora Click envoie en 57600bps.
Module basse consommation énergétique. Peut-être alimenter en 5V et 3V avec une consommation variant de quelque µA en émission à quelque mA en réception apres cela varie suivant l'utilisation que l'on va lui donner
La norme RJ45 prévoit une porté max de 100m, sachant que l'on utilise quelque dizaine de mètres, donc largement suffisant pour nous.

L'avantage de ce module c'est qu'il permet de passer en UART une communication filaire sur une distance suffisante pour nous et nous simplifie la vie en réalisant l'adaptation entre une communication UART avec la norme RJ45. Ce module nous permet de ne pas faire d'adaptions de protocole en réalisant une carte et nous permet de réaliser un gain de temps car autrement on aurait dû faire une carte électronique réalisant un changement de protocole avec un module passant de UART à SPI et une autre de SPI à UART.

Ce module comme indiqué sur le schéma permet de communiquer avec le module depuis la stm32 et donc d'envoyer des commandes au module de communication depuis la STM32 se trouvant sur le drone et le module sur la bouée.

Pour la mise en place du module il suffit de connectés des deux côtés le module avec un câble RJ45. Et réaliser comme indiquer sur le schéma une connexion entre module et STM32 et de l'autre cotés sur la bouée une connexion entre module et Lora Click. Et une fois le schéma réaliser tester si les commandes sont bien reçues.

Alimentation

Pour l'alimentation j'aurai à gérer le besoin en tension et courants de toutes les cartes et les moteurs présents sur le drone sous-marin pour cela la partie alimentation sera séparée en deux parties :

- Adapté la tension
- Sélection des caractéristiques de la/les batterie/s pour que le drone tienne le temps indiqué sur le cahier des charges

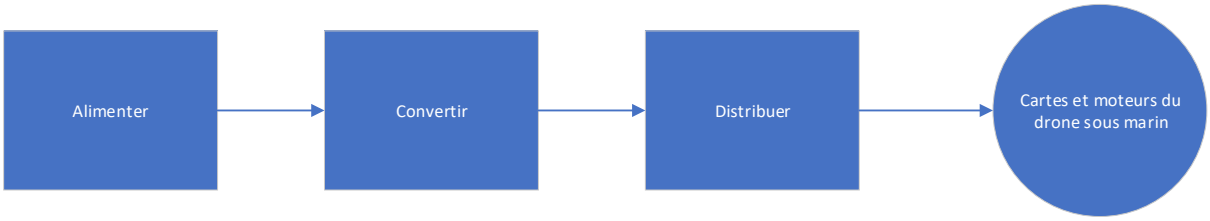


Figure 21 : Diagramme Alimentation

Alimenter : généré tension et courant pour le drone sous-marin.
Convertir : adapté les tensions et mettre les bonnes protections en tension et courant du drone sous-marin.

- La partie alimenter va être réaliser par une batterie Lithium-ion Polymère avec le choix des différentes caractéristiques.
- La conversion (ou adaptation) et distribution sera réaliser par la/les cartes alimentation qui s'occuperont des différentes adaptations.

Batterie et caractéristique

Pour la technologie de batterie à utilisé sera de la lithium polymère car cette technologie est la plus adapté pour les drones car son rapport Energie par Masse et le plus avantageux par rapport au autres technologies qui pour la même capacité vont prendre beaucoup plus de place. Cette technologie est celle qui possède une forte intensité de décharge et une faible autodécharge.
Les caractéristique à définir majoritairement et la capacité et le nombre de cellules, ces deux choses imposeront plein de chose comme :

- Gamme de Tension
- Décharge instantané maximum
- Poids et dimension

Partie à alimenter	*Consommation en courant (A)*	*Tension (V)*
Déplacement	*40 A + 0.26 A*	*12V*
Carte-mère	*0.001A*	*5V/3.3V*
Communication	*Non Déterminé*	*Non Déterminé*
Caméra	*Non Déterminé*	*Non Déterminé*

Figure 22: Tableau Consommation

Pour la consommation de la communication et de la caméra sera de l'ordre de 30 mA mais globalement négligeable devant celle du moteur mais à prendre en compte dans les calcul de la limite de courant de l'abaissement de tension car la tension moteur sera différents de celle des autres et donc demandera une régulation séparés.
C'est différente consommation nous indique plusieurs chose la capacité, les tensions à abaisser et celle à adapté et la tension la plus élevé nous indique que l'on aura besoin d'une batterie.

$$Capacité = Courant\ total * Autonomie = 42A * 2H = 84\ A.h$$

Batterie de 84 AH est trop grande pour être acheté en 1 seul : deux possibilités revoir à la baisse les consommation ou acheter plusieurs batterie qui mis en parallèle fera 82Ah ou troisième possibilité revoir à la baisse la contrainte de l'autonomie étant à la base de 2h, la baisser à 30 min ou 45 min car au-dessus de cette temps de fonctionnement la capacité de la batterie devient démesurée. Si nous mettons la contrainte de 2h à 30min on aurait besoin d'une capacité 21000 mAh et 45 min 31500 mAh.
Sachant qu'une tension nominal de batterie est un peu près de 3.7V sachant qu'il nous faut aux alentours de 12V :

$$Nombre\ de\ cellule = \frac{tension\ voulue}{tension\ nominal\ 1\ cellule} = \frac{12}{3.7} \pm 3\ cellule$$

D'après ce calcul on voit que l'on aura besoin de 84000mAh et de 3 cellule. Sachant que l'on a la possibilité de partager la capacité total en plusieurs batterie pour répartir le poids total du drone sous-marin et de simplifier le raccordements interne du drone sous-marin.
Ces différentes caractéristiques nous amènent à choisir une batterie avec 8400mAh et 12V nominale. Le nombre de cellule dépendra à la fois de la tension demandé mais aussi la disponibilité de la capacité demandé avec ce nombre de cellule.
Donc je choisis de diminuer la contrainte d'autonomie à 30 min ce qui fait 21000mAh.

Adapté les tensions

Explication

D'après les tensions indiqués j'ai besoin d'élevé une tension à 12V car la batterie bougera de tension et va varier entre 13 et 11 V et les moteurs ont besoins d'une tension fixe et continue et extrêmement stable et d'un courants de blocage pouvant aller jusqu'à 20A par moteur.
Donc il faut un tension régulé à 12V et résistants à 42 A donc le composants le plus adaptés un régulateur élévateur de tension avec un circuit sécurisé car courants élevés.

Pour les autres tension il suffit juste de les abaisser à 5 et 3.3V avec des courants très faible par rapport à celui du moteur.
Donc pour résumé :

- Régulateur élévateur de tension : 12V, et résistants à un courants de blocage de 42 A

Figure 23: Diagramme Solution 1

- Régulateur abaisseur de tension : 5 et 3.3V avec moins de 100mA

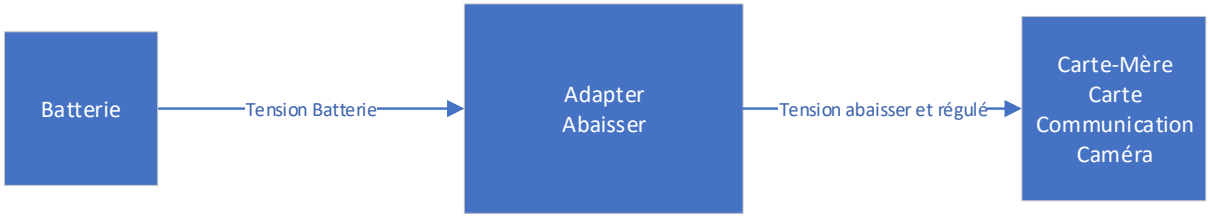


Figure 24: Diagramme Solution 2

Ces deux schéma nous font visualiser à quoi va ressembler le schéma final de régulation et de nous faire visualiser qu'elles sont les entrés et sortie de chaque carte.

La demande en courant élevé du moteur dues à son courant de blocage, et le régulateur gérant cette régulation devra gérer ce courant et ne pas ce bloqué à chaque blocage moteur. Ce qui nous amène à un régulateur extrêmement couteux et délicat à fabriquer donc la moindre erreur faite au cours de la fabrication pourra casser le régulateur donc nous amène au choix suivant d'acheter une carte toute faite qui coûtera moins chère que le régulateur et qui certifie le fonctionnement sécurisé du régulateur.

Possibilités

L'avantage de prendre ces modules c'est qu'ils ne sont pas chers et qui sont fonctionnel et donc n'abîme pas les autres cartes brancher aux différentes cartes d'alimentation, et si nous la réalisons nous-même rien qu'en prix de composant on dépassera le prix.
Ces différentes cartes sont les possibilités de carte d'alimentation fonctionnel et adaptatif pour l'élévation de tension.

Choix de module d'alimentation

Carte alimentation	*Prix TTC*	*Constructeur*
Module Buck boost réglable,20A,1200W, 8-60V, 12-83V	*19,84*	*Amazon*
Module Buck boost réglable,8-32V à 9-46V 12V à 24V 150W	*13,66*	*Amazon*
Module Buck boost réglable,1200W 20A 12V/24V/36V/48V/60V/72VDC	*19,39*	*Amazon*

Figure 25: Tableau des modules d'alimentation élévateur

La carte la plus adapté est la carte qui peut avoir 1200W de dissipation et 20A max, adapté pour un moteur.

Les cartes d'alimentation possible pour l'abaissement de tension.

Module	*Prix TTC*	*Constructeur*
Module d'alimentation 5V, 3A maximum 6piece	*9,59*	*Amazon*
Convertisseur abaisseur réglable,800mA	*5,69*	*Amazon*

Figure 26 Tableau d'alimentation abaisseur

Validation du fonctionnement des modules d'alimentation

Module choisi pour réaliser l'alimentation de l'ensemble du drone sous-marin

Modules abaisseurs : Modules élévateurs :

Figure 27 : Module Abaisseur

Figure 28 : Module élévateur

Test en tension

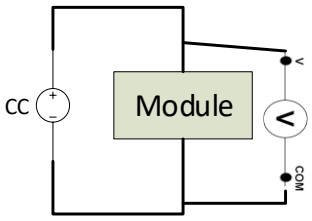


Figure 29 : Schéma de test en tension

Pour valider le fonctionnement de ces deux modules différents, j'ai utilisé une alimentation stabilisée. On envoie une tension d'entrée de 12V et si le module fonctionne (avec les bons réglages) il renvoie une tension de 5V pour le premier module et 12V pour le deuxième.
Je ferai varier la tension d'entrée pour vérifier que les tensions de sortie restent stables. Module régulant pour la STM, caméra autrement dit le module 3A abaisseur.
Les mesures de l'abaisseur sans réglages préalable du potentiomètre donnent :

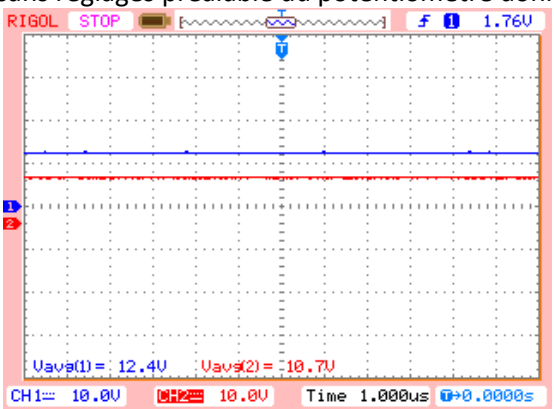


Figure 30 Oscilloscope tension Module abaisseur sans réglages.

ynov AÉRONAUTIQUE & SYSTÈMES EMBARQUÉS

Après réglages, ayant besoin d'alimenter en 5V et 3.3V il faudra réaliser deux réglages sur deux modules différents. Pour le réglage il suffira de modifier la valeur du potentiomètre en surveillant la tension de sortie jusqu'à tomber sur la tension recherchée.

Réglages pour un tension de 5V, 3A :

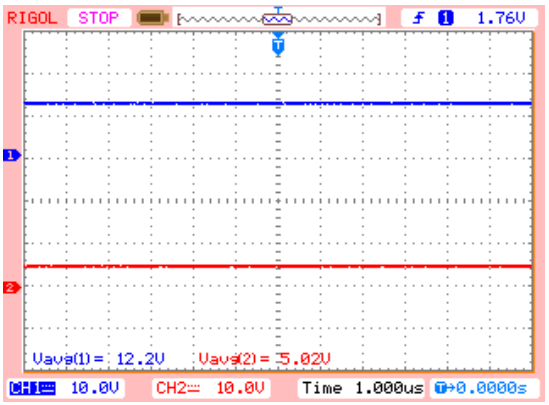


Figure 31 : Oscilloscope tension Module abaisseur réglages 5V.

Réglages pour une tension de 3.3V, 3A :

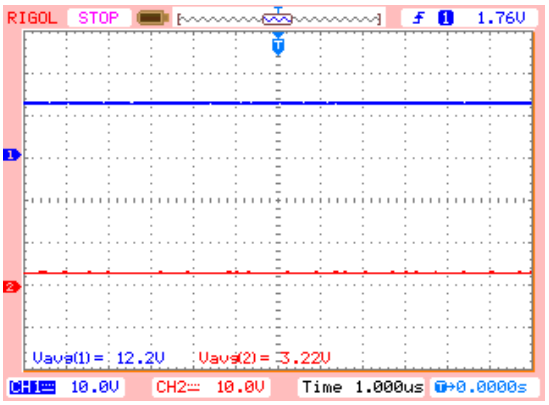


Figure 32: Oscilloscope tension Module abaisseur réglages 3.3V.

L'avantage du module c'est que le module coute 1.56€, 9.59€car il est vendue par 6 ceux qui permettrait de réaliser une carte pour chaque partie à alimenter.
Pour le module 20 A réserver à l'utilisation moteur valide le fonctionnement et les réglages sont effectué.

Test en courant

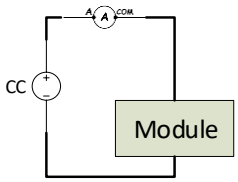


Figure 33 : Schéma test en courant

Pour vérifier sa résistance en tension, je testerai avec l'alimentation stabilisée si je peux tirer du courant et valider le courant limite indiqué par le constructeur.
J'ai répété cette opération avec l'alimentation en élévation prenant un courant élevé avec une étape de plus pour vérifier qu'il peut résister à la consommation moteur et pour cela nous pouvons essayer directement sans étapes intermédiaire car c'est une alimentation déjà validée et certifiée et donc un fonctionnement déjà validé donc la seule chose à réaliser et le réglage de tension de sortie par l'intermédiaire du potentiomètre de sortie.
Après avoir brancher l'alimentation sur batterie et redistribuant la tension au moteur. L'alimentation fournit bien le courant et tension nécessaire au moteur et grâce à une batterie LIPO 5000mAh. Lors des tests effectués, j'observe que l'ESC n'a pas besoin d'une alimentation externe et qui régule automatique la tension du moteur pour s'alimenter.

Pour conclure, les modules d'alimentation ont tous validé leurs fonctionnements et fournit la bonne tension à chaque partie.
Après test nous prenons deux batterie LIPO de 5000 mAh récupérer, contrairement annoncé par le constructeur le moteur consomme beaucoup moins que prévus (sauf en buté) et permet d'augmenter l'autonomie et donc la capacité batterie calculée auparavant qui était de 21000mAh peut être ramené à une capacité de 10000mAh ce qui est suffisant pour notre utilisation.

Conclusion

Dans ce projet, l'alimentation est fonctionnelle, tester et valider. En ce qui concerne la communication pratiquement fonctionnelle car envoie de données validé quelques détails sur la réception à régler.
Ce qu'il reste à réaliser :

- Finir la partie carte mère
- Faire l'intégration de toutes les parties

En ce qui concerne la partie carte mère non-réaliser à cause d'un imprévue (personne en charge partie en stage). Pour l'intégration il faudrait que toutes les parties soit fonctionnels.

Ce projet m'aura permis de découvrir l'univers des module LORA avec son protocole et ces commandes.

Annexe

Main.c

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
 * @file           : main.c
 * @brief          : Main program body

******************************************************************************
 ** This notice applies to any and all portions of this file
 * that are not between comment pairs USER CODE BEGIN and
 * USER CODE END. Other portions of this file, whether
 * inserted by the user or by software development tools
 * are owned by their respective copyright owners.
 *
 * COPYRIGHT(c) 2019 STMicroelectronics
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *   1. Redistributions of source code must retain the above copyright notice,
 *      this list of conditions and the following disclaimer.
 *   2. Redistributions in binary form must reproduce the above copyright
notice,
 *      this list of conditions and the following disclaimer in the
documentation
 *      and/or other materials provided with the distribution.
 *   3. Neither the name of STMicroelectronics nor the names of its
contributors
 *      may be used to endorse or promote products derived from this software
 *      without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
```

```
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *

******************************************************************************
 */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------
*/
#include "main.h"
#include <stm32f4xx_hal.h>
#include "uart.h"
#include "init_uart.h"
#include <string.h>
/* Private includes ----------------------------------------------------------
*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
*/
/* USER CODE BEGIN PD */


/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
*/


/* USER CODE BEGIN PV */
/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------------
*/


/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration--------------------------------------------------------
--*/

    /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */


    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    MX_USART3_UART_Init();
```

```
    /* USER CODE BEGIN 2 */

    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */


    HAL_UART_Transmit(&huart2,(uint8_t*)"\033c",5,10);
    //for(int i =0 ; i<sizeof(symbole);i++)
    //  HAL_UART_Transmit(&huart2,(uint8_t*)symbole[i],sizeof(symbole),1000);
    //HAL_UART_Transmit(&huart2,(uint8_t*)symbole,sizeof(symbole),10);
    //HAL_UART_Transmit(&huart3,(uint8_t*)"mac reset\r\n",11,10);
    init_lora(1);
    //HAL_UART_Receive_IT(&huart3,bufferdata1,sizeof(bufferdata1));
    //envoie_bis(10,2,"aa");
    global_envoie();

    //global_reception();
    while (1)
    {
        /* USER CODE BEGIN 3 */

    }
    /* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */



/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{

    if(huart==&huart3)
    {
        HAL_UART_Transmit(&huart2,bufferdata1,sizeof(bufferdata1),tx_timout);
    }
}

/* USER CODE END 4 */
```

```
/**
 * @brief  This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return
state */

    /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
 * @param  file: pointer to the source file name
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line
number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

Init.c

```c
/*
 * init_uart.c
 *
 *  Created on: 1 avr. 2019
 *      Author: Ricod
 */

#include "init_uart.h"
#include <stm32f4xx_hal.h>

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /**Configure the main internal regulator output voltage
     */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
    /**Initializes the CPU, AHB and APB busses clocks
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
    /**Initializes the CPU, AHB and APB busses clocks
     */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
    {
        Error_Handler();
    }
}
```

```c
/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
void MX_USART2_UART_Init(void)
{

    /* USER CODE BEGIN USART2_Init 0 */

    /* USER CODE END USART2_Init 0 */

    /* USER CODE BEGIN USART2_Init 1 */

    /* USER CODE END USART2_Init 1 */
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 57600;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART2_Init 2 */

    /* USER CODE END USART2_Init 2 */

}

/**
 * @brief USART3 Initialization Function
 * @param None
 * @retval None
 */
void MX_USART3_UART_Init(void)
{

    /* USER CODE BEGIN USART3_Init 0 */

    /* USER CODE END USART3_Init 0 */
```

```
    /* USER CODE BEGIN USART3_Init 1 */

    /* USER CODE END USART3_Init 1 */
    huart3.Instance = USART3;
    huart3.Init.BaudRate = 57600;
    huart3.Init.WordLength = UART_WORDLENGTH_8B;
    huart3.Init.StopBits = UART_STOPBITS_1;
    huart3.Init.Parity = UART_PARITY_NONE;
    huart3.Init.Mode = UART_MODE_TX_RX;
    huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart3.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart3) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART3_Init 2 */

    /* USER CODE END USART3_Init 2 */

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
void MX_GPIO_Init(void)
{

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

}

}
```

Uart.h

```
/*
 * uart.h
 *
 *  Created on: 1 avr. 2019
 *      Author: Ricod
 */

#ifndef UART_H_
#define UART_H_

#include <stm32f4xx_hal.h>
#include <stdlib.h>


#define init "mac pause"
#define size_init 12


#define tx_timout 2
#define rx_timout 2

UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;


static uint8_t send[10] = "radio tx ";
static uint8_t send1[12] = "mac pause";
static uint8_t receive[13] = "radio rx 0";
static uint8_t after[13] = "mac resume\r\n";

uint8_t bufferdata[20];
uint8_t bufferdata1[20];
uint8_t bufferdata2[40];
uint8_t test[20];

void envoie_function(void);
void global_envoie(void);
void global_reception(void);
void init_lora(uint8_t);
void preenvoie(void);
void postenvoie(void);
void envoie_lora(uint8_t delay, int nbre_caractere ,uint8_t
carac_envoie[nbre_caractere+1]);
void reception_lora(uint8_t);
```

```
void string(void);


#endif /* UART_H_ */
```

Uart.c

```
/*
 * uart.c
 *
 *  Created on: 1 avr. 2019
 *      Author: Ricod
 */

#include "init_uart.h"
#include "uart.h"
#include <stm32f4xx_hal.h>
#include <stdlib.h>
#include <stdio.h>


//Toute commande doit finir par deux chose \r\n, appeler CRLF car le module
//pour comprendre que l'envoie de la commande est terminer

void fin_envoie_huart3(void)
{
    HAL_UART_Transmit(&huart3,(uint8_t*)"\r",1,10);
    HAL_UART_Transmit(&huart3,(uint8_t*)"\n",1,10);
}
void fin_envoie_huart2(void)
{
    HAL_UART_Transmit(&huart2,(uint8_t*)"\r",1,10);
    HAL_UART_Transmit(&huart2,(uint8_t*)"\n",1,10);
}

void end_send(void)
{
    fin_envoie_huart2();
    fin_envoie_huart3();
}

// fonction permettant de vider les buffer de reception pour viter les
// problemes de commentaire en trop ou restant de la derniere reponse

void string(void)
{
```

```
    for(int i=0 ; i<sizeof(bufferdata) ; i++)
    {
        bufferdata[i]=0;
        bufferdata1[i]=0;
        bufferdata2[i]=0;
    }
}
void init_lora(uint8_t delay)
{
    string();
    // permet la reinitialisation du module

    HAL_UART_Transmit(&huart3,"sys reset\r\n",11,tx_timout);
    HAL_UART_Transmit(&huart2,"sys reset\r\n",11,tx_timout);
    while(bufferdata2[0]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata2,30,rx_timout);
        HAL_UART_Transmit(&huart2,bufferdata2,30,tx_timout);
    }
    //Permet de savoir la fonctionnalit? de la pile, si tout va bien cens?
r?pondre 4294967295
    //commande permettant la reinitialisation du module
    /*for(int i=0 ; i<9; i++)
    {
        HAL_UART_Transmit(&huart3,&send1[i],1,tx_timout);
        HAL_UART_Transmit(&huart2,&send1[i],1,tx_timout);
    }
    end_send();
    while(bufferdata[0]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata,10,rx_timout);
        HAL_UART_Transmit(&huart2,(uint8_t*)bufferdata,10,tx_timout);
    }*/

}

void preenvoie(void)
{
    string();
    //end_send();
    //envoie de la commande mac pause
    for(int i=0 ; i<9; i++)
    {
        HAL_UART_Transmit(&huart3,&send1[i],1,tx_timout);
        HAL_UART_Transmit(&huart2,&send1[i],1,tx_timout);
    }
```

```
    end_send();
    //attente de la r?ponse du module
    //HAL_UART_Receive_IT(&huart3,bufferdata1,12);
    while(bufferdata[0]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata,12,rx_timout);
        HAL_UART_Transmit(&huart2,(uint8_t*)bufferdata,12,tx_timout);
    }
    // Apres avoir effectue la commande init "mac pause" on doit recevoir
:4294967245
}
void postenvoie(void)
{
    string();
    //envoie de la commande mac resume
    for (int i=0 ; i < 13 ; i++)
    {
        HAL_UART_Transmit(&huart3,&after[i],1,tx_timout);
        HAL_UART_Transmit(&huart2,&after[i],1,tx_timout);
    }
    //HAL_UART_Receive_IT(&huart3,bufferdata1,4);
    while(bufferdata2[0]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata2,4,rx_timout);
        HAL_UART_Transmit(&huart2,bufferdata2,4,tx_timout);
    }
}

void envoie_lora(uint8_t delay, int nbre_caractere ,uint8_t
carac_envoie[nbre_caractere+1])
{
    //uint8_t rep_lora[20]="radio_tx_lora";
    string();
    //preenvoie();
    for(int i=0 ; i< 10/*sizeof((send))*/;i++)
    {
        HAL_UART_Transmit(&huart3,&send[i],1,tx_timout);
        HAL_UART_Transmit(&huart2,&send[i],1,tx_timout);
    }
    for(int i = 0; i <  nbre_caractere;i++ )
    {
        HAL_UART_Transmit(&huart3,&carac_envoie[i],1,tx_timout);
        HAL_UART_Transmit(&huart2,&carac_envoie[i],1,tx_timout);
    }
    end_send();
    /*HAL_UART_Receive_IT(&huart3,bufferdata1,4);
```

```
    HAL_UART_Receive_IT(&huart3,bufferdata1,13);*/
    while(bufferdata2[4]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata2,19,rx_timout);
    }
    HAL_UART_Transmit(&huart2,bufferdata2,17,tx_timout);
    /*while(bufferdata2[0]=='\0')
    {

    }*/
    //HAL_UART_Receive(&huart3,bufferdata2,11,rx_timout);
    string();
}

void reception_lora(uint8_t delais)
{
    string();
    for(int i = 0; i < 10/* sizeof(carac_envoie)*/;i++ )
    {
        HAL_UART_Transmit(&huart3,&receive[i],1,tx_timout);
        HAL_UART_Transmit(&huart2,&receive[i],1,tx_timout);
    }
    end_send();

    while(bufferdata1[4]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata1,19,rx_timout);
        HAL_UART_Transmit(&huart2,bufferdata1,17,tx_timout);
    }
    /*
    while(bufferdata2[10]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata2,13,rx_timout);
        HAL_UART_Transmit(&huart2,bufferdata2,13,tx_timout);
    }*/

    /*while(bufferdata1[0]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata1,4,rx_timout);
        HAL_UART_Transmit(&huart2,bufferdata1,4,tx_timout);
    }
    while(bufferdata2[4]=='\0')
    {
        HAL_UART_Receive(&huart3,bufferdata2,13,rx_timout);
        HAL_UART_Transmit(&huart2,bufferdata2,13,tx_timout);
    }*/
```

```
    string();
}

void envoie_bis(uint8_t delay, int nbre_caractere ,uint8_t
carac_envoie[nbre_caractere+1])
{
    int var = 0;
    //attente de fin envoie et de confirmation d'envoie termin
    switch (var) {
    case 0:
        string();
        //end_send();
        //envoie de la commande mac pause
        for(int i=0 ; i<9; i++)
        {
            HAL_UART_Transmit(&huart3,&send1[i],1,tx_timout);
            HAL_UART_Transmit(&huart2,&send1[i],1,tx_timout);
        }
        end_send();
        //attente de la r
ponse du module
        HAL_UART_Receive_IT(&huart3,bufferdata1,12);
        var++;
        break;
    case 1:
        for(int i=0 ; i< 10/*sizeof((send))*/;i++)
        {
            HAL_UART_Transmit(&huart3,&send[i],1,tx_timout);
            HAL_UART_Transmit(&huart2,&send[i],1,tx_timout);
        }
        for(int i = 0; i <  2;i++ )
        {
            HAL_UART_Transmit(&huart3,&carac_envoie[i],1,tx_timout);
            HAL_UART_Transmit(&huart2,&carac_envoie[i],1,tx_timout);
        }
        end_send();

        HAL_UART_Receive_IT(&huart3,bufferdata1,4);
        HAL_UART_Receive_IT(&huart3,bufferdata2,13);
        var++;
        break;
    case 2:
        for (int i=0 ; i < 13 ; i++)
        {
            HAL_UART_Transmit(&huart3,&after[i],1,tx_timout);
            HAL_UART_Transmit(&huart2,&after[i],1,tx_timout);
        }
```

```
        HAL_UART_Receive_IT(&huart3,bufferdata1,4);

        var++;
        break;
    default:
        break;
    }
}

void global_envoie(void){
    preenvoie();
    HAL_Delay(10);
    envoie_lora(10,2,(uint8_t*)"aa");
    //reception_lora(10);
    postenvoie();
    //HAL_Delay(42);
}
void global_reception(void){
    preenvoie();
    HAL_Delay(10);
    reception_lora(100);
    postenvoie();
}
/*void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart==&huart3)
    {
        HAL_UART_Transmit(&huart2,bufferdata1,sizeof(bufferdata1),tx_timout);
    }
}*/

/*
 * lora.c
 *
 *  Created on: 2 avr. 2019
 *      Author: Ricod
 */
```