IMA4
2016
2017

POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Polytech Lille
Avenue Paul Langevin
Villeneuve-d'Ascq, 59650

UNG Nicky & MACHEREZ Alexis
IMA4SC

[BROUILLEUR D'ONDES LORA – RAPPORT DE PROJET]

Ce document est un rapport du projet P32 du Semestre 8 IMA4SC de MACHEREZ Alexis et UNG Nicky, et porte sur la réalisation du brouillage d'un réseau de communication pour l'IoT : le réseau LoRa.

Remerciements

Nous remercions tout d'abord nos tuteurs de projets, M. Thomas VANTROYS, M. Alexandre BOÉ et M. Xavier REDON, pour l'aide qu'ils nous ont apporté en répondant à nos interrogations et en nous fournissant le matériel nécessaire, ainsi que pour le temps qu'ils nous ont consacré au cours des séances de projet.

Ensuite, nous remercions l'équipe pédagogique IMA, qui nous a enseigné les connaissances de base pour aborder au mieux ce projet, qui mêle diverses connaissances.

Nous souhaitons aussi remercier M. Thierry FLAMEN, dont les conseils et connaissances en cartes électroniques ont été d'une aide précieuse lors des phases de réalisation du dispositif brouilleur, ainsi que M. Laurent ENGELS pour le temps consacré au tournage et montage de notre vidéo, qui nous a permis d'illustrer les principales étapes de notre projet.

UNG Nicky et MACHEREZ Alexis

Table des matières

Contenu

1. Introduction

PRESENTATION GENERALE DU PROJET

Notre 4ème année de formation ingénieur IMA de l'école nous amène à réaliser un projet au cours du semestre 8. Ce projet mobilise les connaissances acquises jusque maintenant au cours de notre formation et permet d'appliquer ces compétences dans le but d'améliorer notre capacité de gestion d'un projet technique, que nous serons amenés à réutiliser dans nos futures fonctions d'ingénieur. Ce module nous laisse libres de décider du sujet technologique, et gérer le projet de bout en bout, du cahier des charges à la validation des fonctions, en passant par l'étude théorique puis l'élaboration d'un prototype suite à la phase de développement.

Ce rapport offre une présentation plus détaillée des différentes phases de notre projet, qui sont l'analyse théorique, la réalisation du prototype et sa programmation. Ces phases nous ont menés à des tests grâce auxquels nous avons pu tirer des conclusions et valider notre modèle, tout en nuançant avec les difficultés rencontrées et les améliorations possibles.

CONTEXTE

L'expansion de l'utilisation des objets connectés permet de nos jours d'accéder à des données ou de contrôler aisément d'autres systèmes connectés à distance. Avec l'augmentation de l'utilisation des réseaux de communication et leur développement, les objets connectés sont de plus en plus déployés mais leur sécurité n'est pas toujours testée. Par conséquent, de nombreuses informations sensibles sont susceptibles de transiter en permanence par voie hertzienne par le biais de multiples plages de fréquences.

Ceci amène donc un questionnement sur la sécurité des transferts de données entre la multitude d'objets connectés actuellement.

OBJECTIF DU PROJET

L'objectif principal de ce projet est de réaliser un brouilleur d'ondes RF (radiofréquence) dans la bande des 868 MHz capable de bloquer les communications LoRa.

CAHIER DES CHARGES INITIAL

A l'aboutissement du projet, nous devons être capable de détecter une communication LoRa dans la bande des 868 MHz, et de la brouiller elle seule, sans incidence sur les autres communications sur d'autres fréquences du réseau LoRa.

Pour ce faire, nous devions mettre en place un dispositif réalisant un brouillage sur la plage 863-870Mhz afin d'empêcher toute communication LoRa, en envoyant du bruit électronique sur toute la plage de fréquence.

Pour la suite du projet, nous voulions perfectionner ce montage pour qu'il puisse détecter une communication LoRa à une fréquence donnée (grâce au "join-request message" de 18 octets envoyés par le end-point/node, ou bien grâce aux informations données dans le préambule des paquets), afin d'envoyer des données erronées et de bloquer uniquement cette communication.

Selon le temps et le budget, nous pouvions aussi être amenés à brouiller la plage de fréquences autour des 433 MHz (aussi utilisé par le réseau LoRa, notamment pour la domotique), et pourquoi pas d'autres types de protocoles de communication. Le système doit être par conséquent le plus reconfigurable possible dans l'idéal.

De plus, le système doit être le plus portable et ergonomique possible, c'est-à-dire principalement tenir dans la main.

PROBLEMATIQUE

Comment donc réaliser le brouillage de communications du réseau LoRaWAN ? Dans quelle mesure est-il possible d'interférer avec des communications LoRa ?

2. Étude préliminaire et théorique

INTRODUCTION

Nous nous consacrons dans cette partie à l'étude préliminaire et théorique de notre sujet. Il convient dans un premier temps d'étudier les spécifications décrivant le réseau de communication LoRaWAN, les protocoles utilisés et les configurations recommandées. Dès lors, nous pouvons déterminer comment brouiller la communication que nous aurons nous-mêmes définie, et ainsi concevoir le prototype en choisissant le dispositif adapté.

Nous nous intéressons principalement aux communications LoRa sur la bande de fréquence centrée sur 868 MHz.

LE RESEAU LORAWAN ET LA MODULATION LORA

LoRaWAN est un protocole réseau optimisé pour les end-devices fixes ou mobiles alimentés par batterie. Il fait donc partie de la famille des LPWAN (Low Power Wide Area Network). Les end-devices communiquent bi-directionnellement avec les serveurs, parfois via des passerelles appelées gateways. En particulier, les end-devices communiquent avec les gateways ou les serveurs via une modulation LoRa ou une modulation FSK. Les communications s'effectuent sur différentes plages de fréquences et à des débits différents. Les end-devices changent de canal de transmission pseudo-aléatoirement pour limiter les risques d'interférences.

La modulation LoRa est une modulation sans-fil destinée aux applications longue distance à basse consommation d'énergie et bas débit (0,3 à 50 kbps). La modulation LoRa ne repose pas sur un principe de modulation d'amplitude ou de fréquence de base comme les OOK/ASK ou les GFSK/FSK. Elle repose sur le CSS (Chirp Spread Spectrum) dont les principales composantes sont la BW (bandwidth = bande passante), SP (spreading factor = facteur d'étalement) et CR (chirp rate = taux de variation du chirp, signal modulé en fréquence). En plus de mobiliser des principes complexes de schémas de modulation (théorème de Shannon-Hartley par exemple), elle possède aussi une correction d'erreur (le Forward Error Correction) théoriquement efficace contre les interférences. C'est cette modulation que nous cherchons à brouiller dans notre projet.

CHOIX DE LA COMMUNICATION A BROUILLER

Nous avons choisi de nous concentrer principalement sur la communication en modulation LoRa (mais accessoirement FSK) autour de 868 MHz. La modulation LoRa étant la plus difficile à brouiller, il faut considérer le brouillage de la modulation FSK aussi.

DIFFERENTS TYPES DE BROUILLAGES ET CHOIX TECHNIQUE

Le brouillage est défini comme « une émission intentionnelle de signaux radioélectriques destinée à interférer dans le fonctionnement d'un radar en saturant son récepteur soit avec un bruit puissant, soit avec de fausses données ».

Les deux techniques principales de brouillage électronique sont les techniques dites « de bruit » ou de « ré-émission ». Elles n'opèrent pas au même niveau, le premier relevant plutôt du hardware, et le deuxième du software.

Après avoir étudié les différentes caractéristiques de la modulation LoRa décrites dans les datasheets de SEMTECH et autres documents (voir Annexe, Liens utiles), nous sommes parvenus à identifier plusieurs méthodes, plus ou moins efficaces, pour le brouillage de la modulation LoRa :

- **Brouillage « informatique »** : On cherche à cibler les couches 2 à 4 du modèle OS, celles qui permettent généralement de corriger les erreurs. Cela s'apparente à une méthode électronique dite Digital Radio Frequency Memory (DRFM) qui consiste à enregistrer numériquement un signal radioélectrique et à le re-émettre après l'avoir modifié pour fausser les informations reçues par le récepteur.
    - Une méthode envisageable consisterait à détecter une communication LoRa à une fréquence donnée (grâce au "join-request message" de 18 octets envoyés par le end-point/node, ou bien grâce aux informations données dans le préambule des paquets), afin d'envoyer des données erronées vers le point de réception et de bloquer uniquement cette communication et non les autres.

- **Brouillage électronique** (ou radio) : Ce type de brouillage repose sur la capacité à émettre un signal qui interfère avec ceux d'un récepteur en le saturant par une émission plus puissante que celle émise par l'émetteur en concurrence.

- Une méthode envisageable est de réaliser un brouillage à balayage, consiste à faire varier la fréquence du brouillage. Bien que cette technique permette de brouiller plusieurs fréquences alternativement par des changements rapides, toutes les fréquences n'étant pas brouillées en même temps et l'efficacité dépendant du dispositif de traitement des erreurs du récepteur brouillé, l'efficacité du système reste limitée.

Nous avons choisi d'implémenter la technique du brouillage à balayage, qui représentait une méthode abordable d'un point de vue technique et complexité de réalisation. Nous allons donc essayer de brouiller un communication LoRa en balayant toute une plage de fréquence avec un signal assez puissant pour interférer avec les données reçues par le récepteur.

Pour la deuxième fonction, nous nous orientons vers un brouillage du même type que le premier, avec un scan d'une plage de fréquence avant de lancer un brouillage à balayage. La méthode envisageable de brouillage « informatique » décrite précédemment n'est pas réalisable dans nos conditions car cela requiert de connaître certaines caractéristiques de la modulation utilisée pour l'émission, comme le Spreading Factor et la taille du paquet.

LES FACTEURS INFLUANT SUR L'EFFICACITE DU DISPOSITIF

Au vu des caractéristiques de la modulation LoRa, l'efficacité du dispositif dépend donc de nombreux facteurs dus à l'architecture de celle-ci. Pour réussir à créer assez d'interférences pour brouiller, nous devrons tenir compte des facteurs suivants :

- On peut tout d'abord mentionner le principe du SNR, le signal-to-noise ratio, qui traduit la corruption du signal transmis, autrement dit le ratio minimum, entre la puissance du signal à transmettre et la puissance du bruit interférant, que le récepteur est capable de démoduler. C'est un facteur déterminant dans la mesure de signaux sur des canaux de communication. Un SNR important garantit de faibles distortions et erreurs dues au bruit, et donc la qualité et l'intégrité du signal transmis. Un ratio supérieur à 1 indique que le signal transmis est plus puissant que le bruit. Nous cherchons dans notre cas à faire baisser le SNR, dont la formule est

donnée ci-dessous, en-dessous de 1, et le plus possible pour assurer un brouillage total. Plus ce SNR tend vers 0 , plus l'on est sûrs que le brouillage est garanti.

$$SNR = \frac{P(signal)}{P(bruit)}$$

- Un autre facteur influant est la sensibilité, qui désigne en électronique et en télécommunication le niveau minimum de signal détectable par un récepteur. D'après les datasheets, la modulation LoRa permet au récepteur une plus grande sensibilité que la modulation FSK. Le récepteur nécessite donc moins de puissance reçue pour communiquer correctement.

- Le code de correction Forward Error Correction permet au récepteur de corriger des erreurs de bits corrompus par des interférences.

- En plus du schéma de modulation CSS, l'influence du Spreading Factor (facteur d'étalement) est non négligeable. L'étalement de spectre est notamment utilisé dans le domaine militaire pour garantir l'intégrité des communications et parer le brouillage.

- Le changement pseudo-aléatoire de fréquence de communication de la modulation LoRa, lorsque le canal est est trop occupé, rend difficile le brouillage continu d'une communication LoRa.

- Enfin la sélectivité de la modulation LoRa, en comparaison avec la FSK, décrite dans la datasheet AN 1200.22 indique que la modulation LoRa bénéficie d'une immunité de 10 à 20 dB supplémentaire sur les intéférences.

DEMARCHE

Le protocole du projet consiste donc dans un premier temps à développer le dispositif brouilleur et le programmer avec les deux fonctions annoncées. Après avoir pris en main le fonctionnement des modules LoRa, nous pouvons passer à la phase de test/vérification et de validation du fonctionnement. Enfin, cela permettra de conclure sur l'efficacité du dispositif, les difficultés rencontrées et améliorations possibles.

CHOIX TECHNIQUES: MATERIEL REQUIS

Afin de réaliser les fonctions que nous avons décrites précédemment, nous disposons donc du matériel suivant:

- un microcontrôleur CC430 muni d'un transceiver CC1101
- un récepteur LoRa (868MHz) communicant via protocole SPI avec le microcontrôleur
- 2 antennes 868 MHz
- Une batterie et quelques LEDs
- Deux modules LoRa 868 MHz (voir image ci-dessous) de puissance maximale d'émission 14 dBm (alimentées en USB)

Nous n'utilisons pas les LEDs, la carte CC430 fournie par M. BOÉ possédant déjà des LEDs, ni la batterie, car les modules et la carte peuvent être alimentés par un port USB. Nous n'utilisons pas non plus le feather LoRa initialement prévu car cela nécessite de connaître à l'avance le Spreading Factor et la taille du paquet transmise (nécessaire pour décoder le CRC et le préambule). De plus, le brouillage que nous réalisons pour la 2ème fonction ne requiert que de pouvoir recevoir en RF, donc seule l'antenne suffit.

LISTE DES TACHES

En fin de projet, les tâches effectuées comprennent finalement:

- Documentation sur le réseau LoRaWAN
  - Modes de transmission
  - Schémas de modulation
  - Trame des paquets
  - Puissance, Sensitivité, Séléctivité, Rapport Signal-Bruit
- Réalisation du premier montage brouillant toute une plage
  - Contrôler l'émetteur RF du CC430
  - Programmer un bruitage sur la plage 863-870MHz
  - Test et optimisation
- Ajout d'un récepteur LoRa pour détecter et brouiller une seule communication
  - Contrôler la réception RF du CC430
  - Détecter une communication LoRa à une fréquence donnée après avoir scanné une plage de fréquences
  - Contrôler l'émetteur RF afin de brouiller cette communication de manière « brutale »
  - Brouiller cette communication en tentant de changer les données du paquet *(non réalisée)*
  - Test et optimisation

HYPOTHESE DE FONCTIONNEMENT

D'après les éléments cités précédemment, de nombreux éléments peuvent influer sur le fonctionnement du dispositif brouilleur et donc son efficacité. En négligeant les effets liés à ces éléments, et en ne tenant compte que du ratio signal-bruit, on peut conjecturer que le brouillage sera réalisé totalement si la condition suivante est réalisée :

$$SNR = \frac{P(signal)}{P(bruit)} < 1$$

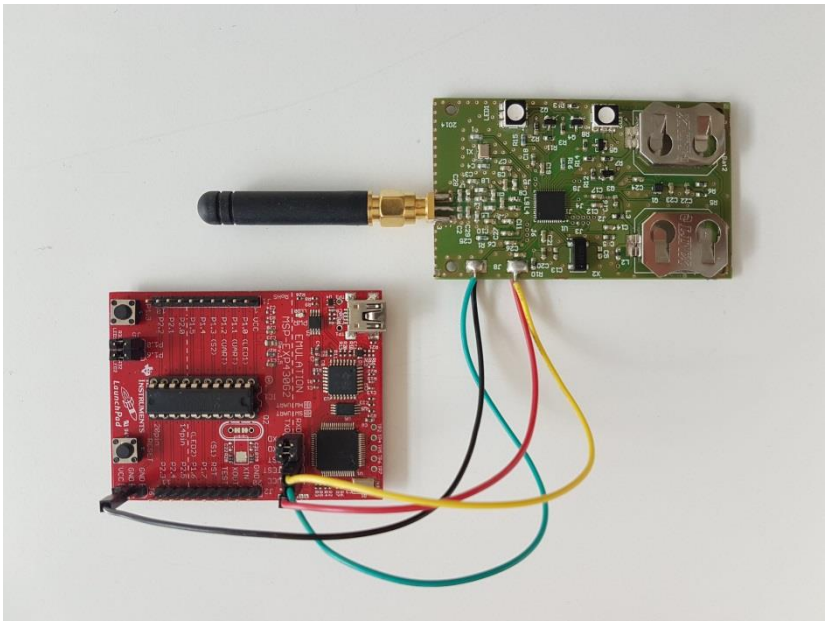
Il faut donc réussir à émettre un signal brouilleur au moins aussi puissant que le signal à transmettre. On appellera cette condition la « condition de brouillage » par la suite.

3. Développement et tests

Lors de la phase de développement du brouilleur, nous avons dû réaliser un programme pour contrôler la carte CC430 fournie par M. BOÉ, et commandée par un microcontrôleur branché en série. Cette phase de développement nous a permis de réaliser des tests pour vérifier la validité de nos fonctions annoncées dans le cahier des charges et de les comparer par rapport aux résultats attendus théoriquement, vus dans la partie précédente.

DEVELOPPEMENT DU DISPOSITIF BROUILLEUR

Nous avons programmé la carte réalisant le brouillage en langage C, et expliquons ci-dessous le fonctionnement du dispositif. Pour flasher la carte, il suffit d'effectuer un make deploy vers celle-ci après avoir créé l'exécutable avec make. Nous vérifions le fonctionnement de la carte en essayant de brouiller deux modules LoRa communiquant à 868 MHz en ping pong sur des terminaux.



La carte CC430 possédait en premier lieu une antenne patch mais après des tests, nous nous sommes aperçus qu'elle n'était pas adaptée car elle avait du mal à brouiller la modulation LoRa efficacement. Nous avons donc décidé de la remplacer en la coupant puis en soudant par-dessus un connecteur SMA 1,6 mm destiné à accueillir une antenne plus puissante.

L'antenne utilisée est une antenne 868 MHz, qui s'apparente à un quart d'onde. En effet, l'antenne mesure environ 5 cm, et la formule $\lambda = \frac{c}{f}$ donne un quart d'onde pour 868 MHz à 8,6 cm.

Pour améliorer l'efficacité du dispositif, l'ajout d'un amplificateur d'adaptation entre la carte et l'antenne est envisageable si celui-ci ne rajoute pas d'encombrement, la carte devant être portable. Nous n'avons cependant pas traité ce point.

1ERE FONCTION : BROUILLAGE D'UNE PLAGE DE FREQUENCES DEFINIE

Cette première fonction a pour objectif de pouvoir brouiller efficacement une plage de fréquences définie au préalable dans le code. Nous avons au départ programmé la carte pour qu'elle brouille la plage de fréquence 863-870 MHz annoncée dans le cahier des charges, mais le brouillage s'avérait n'être pas assez efficace. Nous avons donc opté pour une plage 866-870 MHz.

Le CC430 peut être utilisé en définissant une fréquence de base, il est ensuite possible d'utiliser les canaux pour changer de fréquence de manière plus efficace.

$$f_{carrier} = \frac{f_{XOSC}}{2^{16}} \cdot \left(FREQ + CHAN \cdot \left((256 + CHANSPC\_M) \cdot 2^{CHANSPC\_E-2}\right)\right)$$

Nous avons commencé par initialiser tous les registres du cc1101 en se concentrant principalement sur les paramètres suivants :

- la fréquence : 866MHz

$$f_{carrier} = \frac{f_{XOSC}}{2^{16}} \cdot FREQ[23:0]$$

- l'espacement des canaux : 200kHz

$$\Delta f_{CHANNEL} = \frac{f_{XOSC}}{2^{18}} \cdot (256 + CHANSPC\_M) \cdot 2^{CHANSPC\_E}$$

- le mode de modulation : GFSK

Le détail et le header d'initialisation des registres se trouvent sur l'annexe Rfconfig1.h. L'initialisation de tous les registres se fait ensuite par le biais de la fonction rf_init :

```
void rf_init(void){
  SetVCore(3);
  ResetRadioCore();
»       PMMCTL0_H = 0xA5;
»       PMMCTL0_L |= PMMHPMRE_L;
»       PMMCTL0_H = 0x00;
  WriteBurstReg(IOCFG2, (unsigned char*)RF1A_REGISTER_CONFIG, CONF_REG_SIZE);
  WritePATable();
}
```

Cette fonction fixe le voltage du microcontrôleur avec SetVCore, celui-ci doit se trouver entre 2V et 3,6V, sinon le microcontrôleur risque de buguer. Ensuite la fonction active le mode écriture puis écrit dans les registres avec WriteBurstReg, RF1A_REGISTER_CONFIG étant le tableau contenant les valeurs définies dans le header précédent à appliquer aux registres. Enfin la fonction WritePATable définit la puissance d'émission.

```
void WritePATable(void)
{
  unsigned char valueRead = 0;
  while(valueRead != 0xC0)
  {
    unsigned char i = 0;
    while( !(RF1AIFCTL1 & RFINSTRIFG));
    RF1AINSTRW = 0x7EC0;
    while( !(RF1AIFCTL1 & RFINSTRIFG));
      RF1AINSTR1B = RF_PATABRD;
    for (i = 0; i < 7; i++)
    {
      while( !(RF1AIFCTL1 & RFDOUTIFG));
      valueRead  = RF1ADOUT1B;
    }
    while( !(RF1AIFCTL1 & RFDOUTIFG));
    valueRead  = RF1ADOUTB;
  }
}
```

Ici, la fonction définit la puissance à 12dBm (0xC0). Cette fonction est alors appelée dans le main.

```
int main(void)
{
  port_mapping();
  unsigned char chann=0x00;
  unsigned char led2=0x10;
  WDTCTL = WDTPW + WDTHOLD;
  rf_init();
  ReceiveOff();
  _BIS_SR(GIE);
  set_color(0x10, 0x00, 0x00, 0x00, 0x00, 0x00);

  while(1)
  {
    ChangeChannel(chann);
    send_cmd();
    if(chann>=20)
    {
      chann=0x00;
      led2=led2^0x10;
      set_color(0x10, 0x00, 0x00, led2, 0x00, 0x00);
    }
    else
    {
      chann++;
    }
    __delay_cycles(DELAY);
  }
  return 0;
}
```

Ce programme effectue un port-mapping afin de bien configurer les pins du CC430, puis stoppe le watchdog qui ne sera pas utilisé. Il appelle ensuite rf_init pour initialiser les registres et coupe la réception (puisqu'elle ne sera pas utilisée). La fonction _BIS_SR autorise les interruptions. La boucle while incrémente les canaux de 0 à 19, sachant que les canaux font 200kHz chacun, la boucle parcourt bien la plage 866-870MHz. A chaque itération, l'émetteur envoie des paquets dans le but de brouiller la bande. Les paquets sont envoyés par le biais de la fonction rf_transmit.

```
void rf_transmit(unsigned char *buffer, unsigned char length)
{
  RF1AIES |= BIT9;
  RF1AIFG &= ~BIT9;
  RF1AIE |= BIT9;
  WriteBurstReg(RF_TXFIFOWR, buffer, length);
  Strobe( RF_STX );
}
```

Cette fonction commence par effacer les interruptions en cours puis écrit dans le buffer FIFO de l'émetteur. Elle finit par vider le buffer pour préparer la prochaine émission.

La fonction set_color gère l'allumage des leds. La led1 est allumée en rouge lorsque le CC430 est sous tension, et la led2 clignote en rouge à chaque fois que la plage de fréquence a été balayée. Les headers utilisés dans le programme main sont définis dans main1.c (voir annexe).

Les fonctions utilisées dans ce programme sont disponibles en pseudo-code ou directement en c sur le site texasinstrument.com. Afin de mieux les comprendre nous avons eu accès à un projet IMA5 2015-2016 les utilisant.

1ERE FONCTION : TEST, MESURES, OBSERVATIONS

Nous avons à chacun de nos tests réalisé une vérification à l'analyseur spectral des puissances émises en coaxial par la carte et les modules LoRa. Cela nous a permis d'émettre des prédictions sur l'efficacité du brouillage défini pour la configuration de test, et de le vérifier grâce à la communication en Ping-Pong de deux modules LoRa, observée en série sur les terminaux de deux machines.

Les tests ont été réalisés :

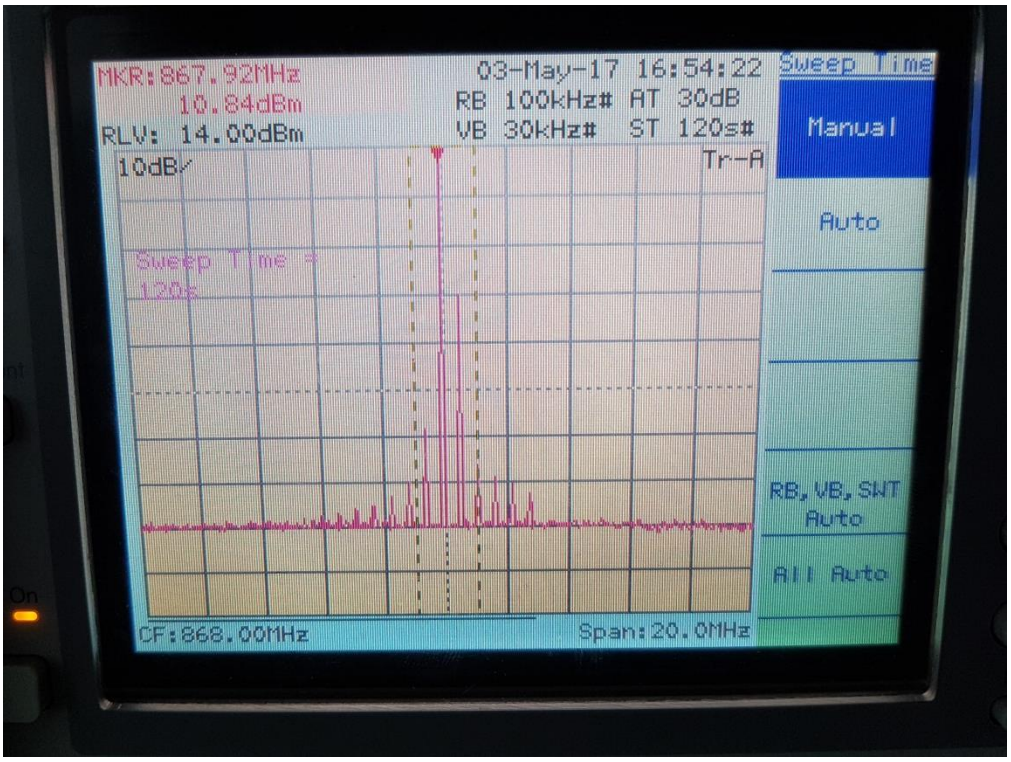
- Avec des communications LoRa pour les modules LoRa
    - 6, 8, 10, 12, 14 dBm de puissance d'émission
- Avec des communications FSK pour les modules LoRa
    - 10, 12, 14 dBm de puissance d'émission
    - Modes 2-FSK, 4-FSK, GFSK
    - Plage de fréquence 863-870

- Modes ASK, OOK, 2-FSK, 4-FSK, GFSK pour le brouilleur
    - Plages de fréquences 863-870, 865-870, 866-870

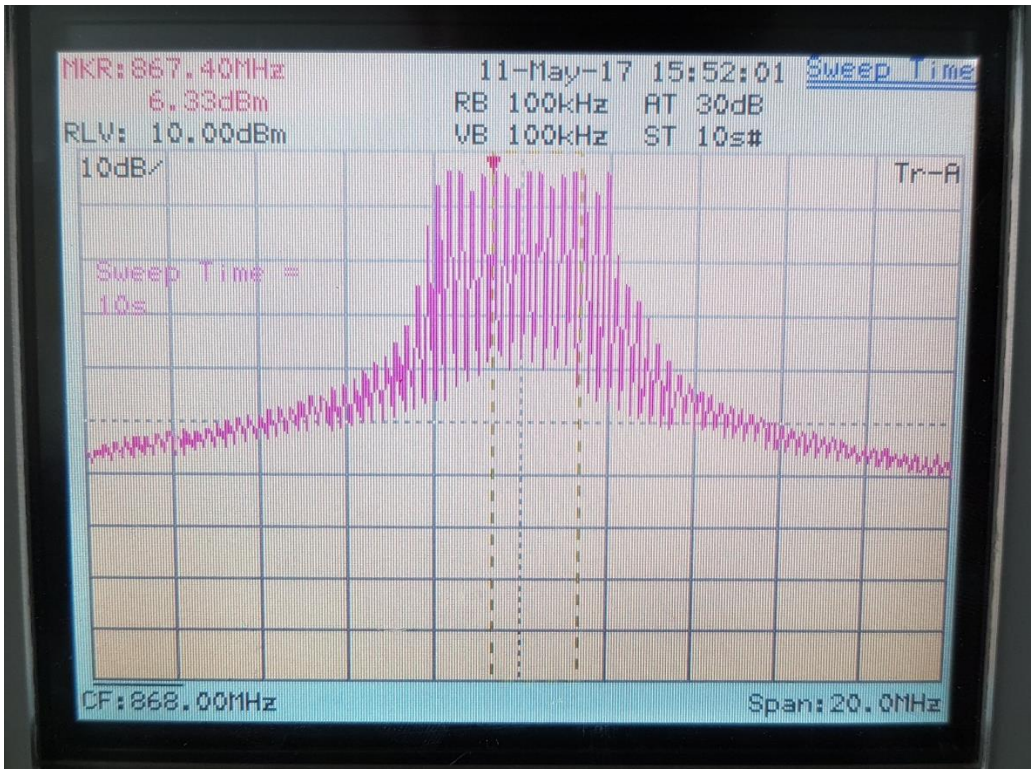
Le code effectuant le Ping-Pong est le même pour les deux modules. Le premier module alimenté envoie « ping... ». Lorsqu'un deuxième module est alimenté celui-ci reçoit alors le ping du premier et répond « pong... ». Ce code en C++ est disponible sur le forum des développeurs sur le site mbed.com, nous n'avons eu que quelques modifications à lui apporter, et nous l'avons directement compilé sur le site. Le compilateur du site intègre toutes les bibliothèques nécessaires et permet de télécharger le binaire produit.

Ci-dessous, voici un exemple de vérification à l'analyseur spectral effectuée pour une configuration avec carte brouilleuse réglée à 12dBm et modules LoRa à 10dBm. Nous observons le balayage en fréquence de l'analyseur spectral grâce à la connexion au connecteur SMA des sorties à 868 MHz de la carte et des modules. En réglant un sweep time long, nous avons les graphes suivants :

- Le spectre d'émission en fréquence des modules LoRa : on relève un pic de puissance 10,84 dBm à 867,92 MHz (10 dBm attendus)



- Le spectre d'émission en fréquence du brouilleur : on relève un pic de puissance 6,33 dBm à 867,40 MHz (12 dBm attendus)



On peut alors calculer le SNR qui vaut pour cette configuration :

$$SNR = \frac{P(signal)}{P(bruit)} = \frac{10{,}84}{6{,}33} = 1{,}71$$

$$SNR\ théorique = \frac{P(signal)}{P(bruit)} = \frac{10}{12} = 0.83$$

On note un écart considérable étant donné que le résultat est opposé, dans un cas, le brouillage est censé se réaliser et pas dans l'autre. On note que cette configuration est la configuration seuil pour la modulation LoRa, c'est-à-dire qu'avec ces paramètres, le déclenchement du brouillage oscille fortement entre brouillage réalisé et brouillage non réalisé. On le remarque avec le Ping Pong instable entre les modules LoRa, tantôt bloqué, tantôt passant.

Nous n'avons cependant pas eu de problème à brouiller une modulation FSK efficacement en respectant la « condition de brouillage », et ce sur toute la plage initialement annoncée de 863 à 870 MHz.

2EME FONCTION : SCAN D'UNE PLAGE DE FREQUENCE PUIS BROUILLAGE D'UNE COMMUNICATION CIBLEE

Cette deuxième fonction a pour objectif de pouvoir scanner une plage de fréquence, définir une zone contenant un pic d'amplitude plus élevée que la moyenne (correspondant à une communication en cours) et efficacement brouiller cette communication sur une petite bande de fréquences l'entourant.

Nous avons finalement choisi de nous en tenir à un brouillage électronique en se basant sur les principes de transmission de l'information et d'électronique analogique/numérique, contrairement au brouillage « informatique » que nous avions annoncé dans le cahier des charges. En effet, pour des raisons de temps et de complexité, utiliser les trames échangées pour le brouillage semblait moins réaliser que de brouiller le pic le plus haut en amplitude après un avoir scanné une plage de fréquence.

Pour réaliser le scan sur la plage de fréquence 863-870MHz, nous avons réalisé une fonction inspirée d'un pseudo-code disponible sur le site de Texas Instruments.

```
void freq_scan(void)
{
  unsigned char rssi;
  long rssi_dbm;
  unsigned char chan_scan=START_FREQ_CHAN;
  long rssi_max=0;
  set_color(led1, 0x00, 0x00, 0x00, led2, 0x00);
  ChangeChannel(chan_scan);
  ReceiveOn();
  while((Strobe(RF_SNOP) & 0x70) != 0x10);
  while(chan_scan<STOP_FREQ_CHAN)
  {
    ChangeChannel(chan_scan);
    __delay_cycles(DELAY_RSSI);
    rssi = ReadSingleReg(RSSI);
    if (rssi >= 128)
    {
      rssi_dbm = (long)((long)( rssi - 256) / 2) - RSSI_OFFSET;
    }
    else
    {
      rssi_dbm = (rssi / 2)  - RSSI_OFFSET;
    }
    if(rssi_dbm>rssi_max)
    {
      rssi_max=rssi_dbm;
      chan_jam=chan_scan-10;
    }
    chan_scan++;
    Strobe( RF_SFRX);
  }
  set_color(led1, 0x00, 0x00, 0x00, 0x00, 0x00);
}
```

Cette fonction change de canal de façon à avoir une fréquence de 863Mhz, puis autorise la réception, la première boucle while s'assure que le cc1101 est passé en mode RX. La deuxième boucle while incrémente les canaux par pas de 200kHz, la bande passante étant de 200kHz, cela correspond bien.

Le delay après le changement de canal permet de laisser au registre RSSI de se mettre à jour. Ce delay dépend de la taille du paquet reçu et est de l'ordre de 10 microsecondes. Cependant, ne pouvant pas savoir à l'avance la taille du paquet reçu, nous avons décidé de fixer le delay à 30 microsecondes.

Après s'être assuré que le registre est à jour, la variable rssi récupère la valeur du RSSI se trouvant dans le registre du même nom. La valeur récupérée dans le registre étant codée en binaire complément 2, nous devons ensuite la convertir en dBm et la placer dans un entier. La formule de conversion (comportant un offset RSSI_OFFSET=74 donnée dans la datasheet) est définie dans le pseudo-code de Texas Instruments. On vérifie ensuite si cette valeur est supérieure à la rssi_max (valeur maximale mesurée jusqu'à présent ou 0dBm). Si c'est le cas, le numéro du canal correspondant à cette valeur maximale est stocké dans une variable globale pour être utilisé par la suite dans le main (voir annexe main_2).

Afin de commencer le brouillage 2MHz en dessous de la fréquence à laquelle la puissance maximale a été mesurée, le canal de brouillage (chan_jam) reçoit la valeur du canal du scan moins 10. La boucle se termine en vidant le buffer FIFO du récepteur.

La fonction set color allume la led2 en bleue pendant le scan et a donc un but uniquement visuel.

Dans le but de ne jamais avoir un canal de brouillage négatif, la fréquence de base a été changée pour 860MHz, et la constante START_FREQ_SCAN est égale à 15 (ce qui commence bien le scan à 863Mhz). Il y a donc un nouveau fichier de headers Rfconfig2.h (voir annexe).

2EME FONCTION : TEST, MESURES, OBSERVATIONS

Après des tests, nous avons pu observer que notre fonction n'a pas l'effet escompté. En effet, la led restant bleue, on peut en déduire que la fonction reste bloquée dans la boucle de scan et ne lance pas de balayage pour brouiller.

4. Analyse des résultats

Tout au long du projet, nous avons été confrontés à de nombreuses zones d'ombre concernant les résultats obtenus et l'efficacité de notre dispositif. L'analyse des résultats que nous ont donnés les différents tests que nous avons réalisés nous permettront d'émettre des hypothèses, et éventuellement des conclusions, sur l'amélioration du brouillage.

FONCTIONNEMENT DU BROUILLEUR

D'après les observations que nous avons émises suite aux tests, et à l'étude préliminaire, on peut affirmer que notre dispositif fonctionne avec le résultat attendu. Nous arrivons à réaliser le brouillage par balayage avec la carte configurée en modulation GFSK / 12 dBm:

- Sur une modulation FSK avec une puissance maximale de 10 dBm et une plage balayée de 863-870 MHz : brouillage total
- Sur une modulation LoRa avec une puissance maximale de 10 dBm et une plage balayée réduite à 866-870 MHz : brouillage instable (c'est la configuration seuil)

LIMITES DU BROUILLEUR

D'après nos tests, on observe que même lorsque le brouilleur est à puissance maximale et placé à côté du récepteur, le brouillage de la modulation LoRa à 10 dBm est très instable. On ne peut pas éloigner le brouilleur à plus d'un vingtaine de centimètres, sans quoi il n'y a plus d'effet sur la communication des modules. L'émission du brouilleur étant relativement directionnelle, il faut aussi veiller à bien diriger l'antenne du brouilleur vers celle du récepteur. La première fonction est donc validée mais avec dans certaines conditions.

La deuxième fonction ne fonctionne pas et n'est pas validée après tests. Comme vu précédemment, la fonction ne sort pas de la fonction de scan, ce qui suppose qu'elle n'arrive pas soit à déterminer la fréquence de la raie d'amplitude maximale, soit que la carte n'arrive pas à recevoir et lire les communications qui sont échangées.

FACTEURS D'INFLUENCE SUR LE BROUILLAGE

En addition à tous les facteurs d'influence cités en partie 2 (étude théorique), nous pouvons catégoriser l'influence sur le brouillage d'autres éléments :

- Distance entre le brouilleur et le récepteur : influence forte
- Respect de la « condition de brouillage » : influence forte
- Absence d'un étage d'amplification en sortie du brouilleur : influence moyenne à forte
- Choix d'un schéma de modulation adapté sur le brouilleur : influence moyenne
- Inconnues sur les caractéristiques de la modulation LoRa configurée : influence moyenne
- Distance entre les deux modules : influence faible à notre échelle

Tous ces facteurs influent sur l'efficacité du brouillage et ne font que renforcer les incertitudes que nous pouvons avoir sur le fonctionnement du brouilleur.

5. Difficultés rencontrées et améliorations

Nous développons dans cette partie les difficultés rencontrées et améliorations ou pistes de solutions apportées à la difficulté en question. Nous avons fait face à des difficultés lors du développement et de l'implémentation du code de la carte du brouilleur.

Pour le brouillage par balayage, voici les difficultés rencontrées et les améliorations apportées ou pistes de solutions:

- Puissance d'émission de l'antenne patch trop faible pour brouiller : nous avons changé l'antenne pour une antenne plus adaptée
- Difficulté à brouiller toute la plage 863-870 MHz pour la modulation LoRa : nous avons réduit cette plage à 866-870 MHz et le brouillage est ainsi plus efficace
- La correction d'erreurs (FEC), la sensibilité, le schéma de modulation CSS, le chanegement de fréquence, la sélectivité du LoRa permettent la réception de paquets malgré le brouillage

Pour la 2ème fonction, voici les difficultés rencontrées et les améliorations apportées ou pistes de solutions:

- La fonction de scan peut renvoyer un canal channr négatif au main auquel cas la fonction ne marche pas : nous avons changé la fréquence de base pour que le canal de départ soit supérieur à 10, le canal renvoyé au main est donc supérieur à 0 (sachant que le canal renvoyé par la fonction est égal au canal détecté – 10)
- La fonction de scan ne détecte aucune communication :
    - Ceci peut être dû à une mauvaise conversion en dBm de notre part
    - Le palier de détection est peut-être trop élevé
    - Le CC430 n'a peut-être pas le temps de passer en mode RX
    - Le registre RSSI n'a peut-être pas le temps de se mettre à jour après chaque changement de canal

Pour finir, de nombreuses incertitudes liées aux mesures et au fonctionnement attendu du matériel viennent entraver nos conclusions. En effet, les mesures n'ont parfois pas donné les résultats et valeurs attendues.

6. Conclusion

En conclusion de ce projet, nous avons réalisé un brouilleur d'ondes à partir d'une puce cc430f5137 qui remplit une partie des fonctions demandées dans le cahier des charges.

Nous sommes parvenus à valider la première partie du cahier des charges. Notre système est capable de brouiller par balayage des communications LoRa et FSK. Cependant il comporte certaines limites, particulièrement lors du brouillage du LoRa. En effet dans ce dernier cas, le système doit rester à une distance raisonnable du module d'émission ou de réception pour réaliser le brouillage. De plus, il doit aussi être placé entre les deux modules LoRa. Enfin, le protocole LoRa intègre un code de correction d'erreurs qui arrive de temps à autre à contrer le brouillage.

En ce qui concerne la deuxième partie du cahier des charges, nous avons réalisé une fonction permettant de détecter une communication dans la plage des 863-870MHz pour ensuite la brouiller. Malheureusement, l'ajout de cette fonction dans notre programme n'a pas fonctionné, et la deuxième partie du cahier des charges n'est pas validée.

Durant ce projet nous avons pu appréhender le métier d'ingénieur, notamment la définition d'un cahier des charges, les solutions apportées pour réaliser un système répondant au cahier des charges et les problèmes rencontrés durant la réalisation du système. En outre, nous avons distingué deux phases lors de notre projet, la première étant la phase de conception, suivie d'une deuxième phase de test et d'optimisation.

7. Annexes

```
#include <msp430.h>
#include <stdint.h>

#include "RF1A.h"
#include "RF_Connection.h"
#include "PMM.h"
#include "ports.h"
#include "RFconfig.h"
#include "led_commands.h"

#define BUFFER_SIZE                10        // maximum size of data sent by rf



#define PWM_PERIOD                 255
#define DELAY 1000 //3ms


volatile uint8_t buffer[BUFFER_SIZE];
volatile uint8_t temp[NB_BANGLES][2];
volatile uint8_t fin[NB_BANGLES][NB_BANGLES-1];

//useless
void packet_received(void)
{

}


void send_cmd(void)
{
  buffer[0] = 0xFD;
  buffer[1] = 0xAA;
  buffer[2] = 0xFF;
  buffer[3] = 0x11;
  buffer[4] = 0xFF;
  buffer[5] = 0x66;
  buffer[6] = 0xFF;
  buffer[7] = 0x33;
  buffer[8] = 0x1F;
  buffer[9] = 0xF8;
  rf_transmit((uint8_t *)buffer, 2);
}

int main(void)
{
```

```
  port_mapping();
  unsigned char chann=0x00;
  unsigned char led2=0x10;
  WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
  rf_init(); //reset, setVcore & patable
  ReceiveOff();
  _BIS_SR(GIE);
  set_color(0x10, 0x00, 0x00, 0x00, 0x00, 0x00); //led1 red if CC430 powered

  while(1)
  {
    ChangeChannel(chann);
    send_cmd();
    if(chann>=20) //866-870
    {
      chann=0x00;
      led2=led2^0x10;
      set_color(0x10, 0x00, 0x00, led2, 0x00, 0x00); //led2 blink red at each ramp
    }
    else
    {
      chann++;
    }
    __delay_cycles(DELAY);
  }
  return 0;
}
```

Code main1.c de la première fonction

```
#include <CC430f5137.h>
#include <stdint.h>
#include <msp430.h>

//863-870Mhz => 26MHz crystal

//MDMCFG0.CHANSPC_M=0 & MDMCFG1.CHANSPC_E=2 ==> chann spacing~100kHz

//start at 863 Mhz
//FREQ2=0x21   FREQ1=0x31   FREQ0=0x3A
//high VCO FSCAL2=0x2A

//867 : 21589A
//866 : 214EC2
//865 : 2144EA
```

```
//for 433Mhz : freq=0x10A766 and low vco (FSCAL2=0x0A)
//432 : 0x109D8E

//gfsk, 866, chann space ~200k, high vco currently


#define SMARTRF_CC430F5137_H
#define SMARTRF_RADIO_CC430F5137

#define SMARTRF_SETTING_IOCFG2     0x29 //default
#define SMARTRF_SETTING_IOCFG1     0x2E //default
#define SMARTRF_SETTING_IOCFG0     0x02 //assert if TX fifo above trsh
#define SMARTRF_SETTING_FIFOTHR    0x07 //default (0x0F may be better)
#define SMARTRF_SETTING_SYNC1      0xD3 //default
#define SMARTRF_SETTING_SYNC0      0x91 //default
#define SMARTRF_SETTING_PKTLEN     0xFF //default (disabled by PKCTRL0 set to 0x05)
#define SMARTRF_SETTING_PKTCTRL1   0x04 //default
#define SMARTRF_SETTING_PKTCTRL0   0x05 //normal pkt format, variable pkt lenght
#define SMARTRF_SETTING_ADDR       0x00 //default (device address)
#define SMARTRF_SETTING_CHANNR     0x00 //0 (will be incremented in main.c)
#define SMARTRF_SETTING_FSCTRL1    0x06 //default 0f (IF frequency)
#define SMARTRF_SETTING_FSCTRL0    0x00 //default (freq offset)
#define SMARTRF_SETTING_FREQ2      0x21 /*freq*/
#define SMARTRF_SETTING_FREQ1      0x4E //866
#define SMARTRF_SETTING_MDMCFG4    0x2C //default (BW=203KHz)
#define SMARTRF_SETTING_MDMCFG3    0x22 //default (data rate=115kBaud)
#define SMARTRF_SETTING_MDMCFG2    0x10 // GFSK, no byte preamble
#define SMARTRF_SETTING_MDMCFG1    0x22 // 2 bytes preamble, chanspc_e=2
#define SMARTRF_SETTING_MDMCFG0    0xF8 //chan space=200khz
#define SMARTRF_SETTING_DEVIATN    0x47 //default 47khz deviation
#define SMARTRF_SETTING_MCSM2      0x10 //direct RX terminaison based on rssi
#define SMARTRF_SETTING_MCSM1      0x63 //stay in TX mode when transmitting & stay
in RX mode when receving
#define SMARTRF_SETTING_MCSM0      0x18 //autocal when going in TX or RX
#define SMARTRF_SETTING_FOCCFG     0x36 //default (frenquency offset for
demodulation)
#define SMARTRF_SETTING_BSCFG      0x6C //default
#define SMARTRF_SETTING_AGCCTRL2   0x07 // (maximum possible gain)
#define SMARTRF_SETTING_AGCCTRL1   0x40 //default
#define SMARTRF_SETTING_AGCCTRL0   0x91 //default
#define SMARTRF_SETTING_WOREVT1    0x80 //given by smartRF studio
#define SMARTRF_SETTING_WOREVT0    0x00 //given by smartRF studio
#define SMARTRF_SETTING_WORCTRL    0xFB //given by smartRF studio
#define SMARTRF_SETTING_FREND1     0x56 //default
#define SMARTRF_SETTING_FREND0     0x10 //patable=0 (useless for gfsk)
#define SMARTRF_SETTING_FSCAL3     0xE9 //smartRF studio
#define SMARTRF_SETTING_FSCAL2     0x2A //high VCO choosen
#define SMARTRF_SETTING_FSCAL1     0x00 //smartRF studio
#define SMARTRF_SETTING_FSCAL0     0x1F //smartRF studio
#define SMARTRF_SETTING_FSTEST     0x59 //default (test only)
```

```
#define SMARTRF_SETTING_PTEST      0x7F //default (test only)
#define SMARTRF_SETTING_AGCTEST    0x3F //default (test only)
#define SMARTRF_SETTING_TEST2      0x81 //smartRF studio
#define SMARTRF_SETTING_TEST1      0x35 //smartRF studio
#define SMARTRF_SETTING_TEST0      0x09 //smartRF studio 0x09 if freq>862MHz
```

Header RFconfig1.c de la première fonction

```
#include <msp430.h>
#include <stdint.h>

#include "RF1A.h"
#include "RF_Connection.h"
#include "PMM.h"
#include "ports.h"
#include "RFconfig.h"
#include "led_commands.h"

#define BUFFER_SIZE                10      // maximum size of data sent by rf


#define PWM_PERIOD                 255
#define DELAY 1000 //3ms
#define DELAY_RSSI 10 //30us

#define START_FREQ_CHAN 15
#define STOP_FREQ_CHAN 50     //scanning 863-870Mhz with 200khz steps
#define RSSI_OFFSET 74 //cf datasheet

#define NUM_JAM_RAMP 50


volatile uint8_t buffer[BUFFER_SIZE];
volatile uint8_t temp[NB_BANGLES][2];
volatile uint8_t fin[NB_BANGLES][NB_BANGLES-1];


uint8_t RxBuffer[BUFFER_SIZE];
uint8_t RxBufferLength=sizeof(RxBuffer);



unsigned char chan_jam=30;
unsigned char led2=0x10;
unsigned char led1=0x10;
```

```
void packet_received(void)
{
  RxBufferLength = ReadSingleReg(RXBYTES);
  ReadBurstReg(RF_RXFIFORD, RxBuffer, RxBufferLength);
}


void freq_scan(void)
{
  unsigned char rssi;
  long rssi_dbm;
  unsigned char chan_scan=START_FREQ_CHAN;
  long rssi_max=0; //scan will take communication above 0dbm
  set_color(led1, 0x00, 0x00, 0x00, led2, 0x00); //led2 blue while scanning

  ChangeChannel(chan_scan);
  ReceiveOn();    //enable RX
  //while((Strobe(RF_SNOP) & 0x70) != 0x10); //make sure RX is ready
  while(chan_scan<STOP_FREQ_CHAN)
  {
    ChangeChannel(chan_scan);
    __delay_cycles(DELAY_RSSI);       //wait for rssi to be updated
    rssi = ReadSingleReg(RSSI);  //define in CC430f5137.h RSSI=0x34 corresponding to
addr of rssi value
    if (rssi >= 128)
    {
      rssi_dbm = (long)((long)( rssi - 256) / 2) - RSSI_OFFSET;
    }
    else
    {
      rssi_dbm = (rssi / 2)  - RSSI_OFFSET;
    }
    if(rssi_dbm>rssi_max)
    {
      rssi_max=rssi_dbm;
      chan_jam=chan_scan-10; //start jamming 2Mhz below max dbm received
    }
    chan_scan++;
    Strobe( RF_SFRX); //flush the RX FIFO
  }
  set_color(led1, 0x00, 0x00, 0x00, 0x00, 0x00);
}



void send_cmd(void)
{
  buffer[0] = 0xFD;
  buffer[1] = 0xAA;
```

```
  buffer[2] = 0xFF;
  buffer[3] = 0x11;
  buffer[4] = 0xFF;
  buffer[5] = 0x66;
  buffer[6] = 0xFF;
  buffer[7] = 0x33;
  buffer[8] = 0x1F;
  buffer[9] = 0xF8;
  rf_transmit((uint8_t *)buffer, 2);
}

int main(void)
{
  unsigned char chan_main=0x00;
  port_mapping();
  WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
  rf_init(); //reset, setVcore & patable
  _BIS_SR(GIE);
  set_color(led1, 0x00, 0x00, 0x00, 0x00, 0x00); //led1 red if CC430 powered
  int i;

  while(1)
  {
    freq_scan();
    ReceiveOff();
    chan_main=chan_jam;
    if (chan_main != 0)  //jam only if a communication xas detected with freq_scan
    {
      for(i=0 ; i<(20*NUM_JAM_RAMP) ; i++) //ramp start 2Mhz below chan_jam and stop
2Mhz above
      {
        ChangeChannel(chan_main);
        if (chan_main>= (chan_jam+20))
        {
          send_cmd();
          chan_main=0x00;
          led2=led2^0x10;
          set_color(led1, 0x00, 0x00, led2, 0x00, 0x00); //led2 blink red every two ramp
of jamming
        }
        else
        {
          send_cmd();
          chan_main++;
        }
      }
    }
    chan_main=0x00;
    __delay_cycles(DELAY);
  }
```

```
  return 0;
}
```

Code main2.c de la deuxième fonction

```
#include <CC430f5137.h>
 #include <stdint.h>
 #include <msp430.h>

 //863-870Mhz => 26MHz crystal

 //MDMCFG0.CHANSPC_M=0 & MDMCFG1.CHANSPC_E=2 ==> chann spacing~100kHz

 //start at 863 Mhz
 //FREQ2=0x21   FREQ1=0x31   FREQ0=0x3A
 //high VCO FSCAL2=0x2A

 //867 : 21589A
 //866 : 214EC2
 //865 : 2144EA


 //for 433Mhz : freq=0x10A766 and low vco (FSCAL2=0x0A)
 //432 : 0x109D8E

 //gfsk, 866, chann space ~200k, high vco currently


 #define SMARTRF_CC430F5137_H
 #define SMARTRF_RADIO_CC430F5137

 #define SMARTRF_SETTING_IOCFG2     0x29 //default
 #define SMARTRF_SETTING_IOCFG1     0x2E //default
 #define SMARTRF_SETTING_IOCFG0     0x02 //assert if TX fifo above trsh
 #define SMARTRF_SETTING_FIFOTHR    0x07 //default (0x0F may be better)
 #define SMARTRF_SETTING_SYNC1      0xD3 //default
 #define SMARTRF_SETTING_SYNC0      0x91 //default
 #define SMARTRF_SETTING_PKTLEN     0xFF //default (disabled by PKCTRL0 set to 0x05)
 #define SMARTRF_SETTING_PKTCTRL1   0x04 //default (enable rssi)
 #define SMARTRF_SETTING_PKTCTRL0   0x02 //normal pkt format, infinite pkt lenght
 #define SMARTRF_SETTING_ADDR       0x00 //default (device address)
 #define SMARTRF_SETTING_CHANNR     0x00 //0 (will be incremented in main.c)
 #define SMARTRF_SETTING_FSCTRL1    0x0F //default (IF frequency)
 #define SMARTRF_SETTING_FSCTRL0    0x00 //default (freq offset)
 #define SMARTRF_SETTING_FREQ2      0x21 /*freq*/
 #define SMARTRF_SETTING_FREQ1      0x13 //860Mhz
 #define SMARTRF_SETTING_FREQ0      0xB1 /*freq*/
```

```
#define SMARTRF_SETTING_MDMCFG4    0x2C //default (BW=203KHz)
#define SMARTRF_SETTING_MDMCFG3    0x22 //default (data rate=115kBaud)
#define SMARTRF_SETTING_MDMCFG2    0x10 // GFSK, no byte preamble
#define SMARTRF_SETTING_MDMCFG1    0x22 // 2 bytes preamble, chanspc_e=2
#define SMARTRF_SETTING_MDMCFG0    0xF8 //chan space=200khz
#define SMARTRF_SETTING_DEVIATN    0x47 //default 47khz deviation
#define SMARTRF_SETTING_MCSM2      0x10 //direct RX terminaison based on rssi
#define SMARTRF_SETTING_MCSM1      0x63 //stay in TX mode when transmitting &
stay in RX mode when receving
#define SMARTRF_SETTING_MCSM0      0x18 //autocal when going in TX or RX
#define SMARTRF_SETTING_FOCCFG     0x36 //default (frenquency offset for
demodulation)
#define SMARTRF_SETTING_BSCFG      0x6C //default
#define SMARTRF_SETTING_AGCCTRL2   0x07 // (maximum possible gain)
#define SMARTRF_SETTING_AGCCTRL1   0x40 //default
#define SMARTRF_SETTING_AGCCTRL0   0x91 //default
#define SMARTRF_SETTING_WOREVT1    0x80 //given by smartRF studio
#define SMARTRF_SETTING_WOREVT0    0x00 //given by smartRF studio
#define SMARTRF_SETTING_WORCTRL    0xFB //given by smartRF studio
#define SMARTRF_SETTING_FREND1     0x56 //default
#define SMARTRF_SETTING_FREND0     0x10 //patable=0 (useless for gfsk)
#define SMARTRF_SETTING_FSCAL3     0xE9 //smartRF studio
#define SMARTRF_SETTING_FSCAL2     0x2A //high VCO choosen
#define SMARTRF_SETTING_FSCAL1     0x00 //smartRF studio
#define SMARTRF_SETTING_FSCAL0     0x1F //smartRF studio
#define SMARTRF_SETTING_FSTEST     0x59 //default (test only)
#define SMARTRF_SETTING_PTEST      0x7F //default (test only)
#define SMARTRF_SETTING_AGCTEST    0x3F //default (test only)
#define SMARTRF_SETTING_TEST2      0x81 //smartRF studio
#define SMARTRF_SETTING_TEST1      0x35 //smartRF studio
#define SMARTRF_SETTING_TEST0      0x09 //smartRF studio 0x09 if freq>862MHz
```

Header RFconfig2.c de la deuxième fonction

LoRaWAN :
https://docs.google.com/viewer?url=http://portal.lora-alliance.org/DesktopModules/Inventures_Document/FileDownload.aspx?ContentID=1397

Théorie – réseau LoRaWAN :
http://www.mdpi.com/1424-8220/16/9/1466/htm

Datasheet CC1101 :
http://www.ti.com/lit/an/swra146b/swra146b.pdf

Datasheet SEMTECH – principes du protocole LoRa (modulation, SNR, sensibilité) :
https://www.semtech.com/images/datasheet/LoraDesignGuide_STD.pdf

Protocoles IoT
http://www.linuxembedded.fr/2016/03/protocoles-de-communication-frameworks-et-systemes-dexploitation-pour-les-objets-connectes/

Présentation LoRa
http://lorawan.blogspot.fr/2015/11/cours-1-architecture-du-reseau-lora.html

Construire son propre réseau LoRa
https://docs.mbed.com/docs/lora-with-mbed/en/latest/intro-to-lora/

Décoder des paquets LoRa
https://www.npmjs.com/package/lora-packet

Décoder du LoRa
https://revspace.nl/DecodingLora

Modulation Modem LoRa
https://myriadrf.org/blog/lora-modem-limesdr/

Recevoir des messages radio LoRa
https://github.com/rpp0/gr-lora

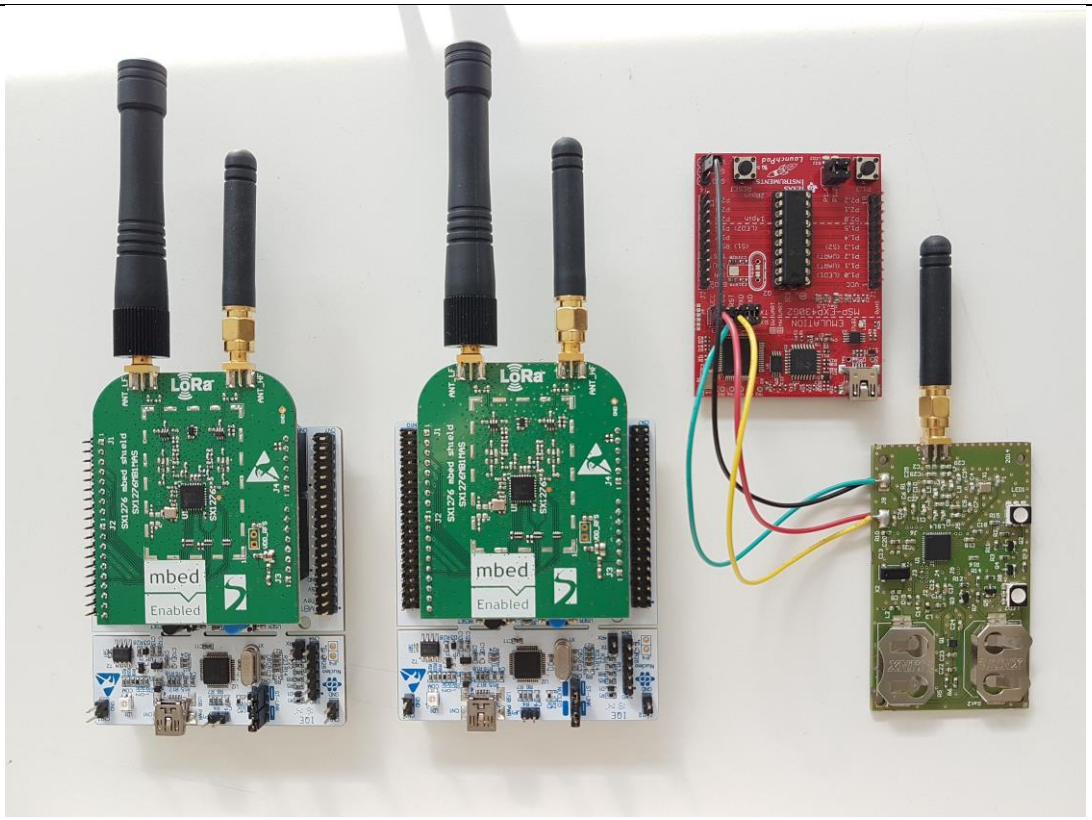
Exemple de communication LoRa
http://www.hoperf.com/upload/docs/rf/AN2003-LoRa%20communication%20example.pdf

Datasheet – Configuration LoRa
http://www.semtech.com/images/datasheet/an1200.23.pdf

Décryptage de la modulation LoRa
https://static1.squarespace.com/static/54cecce7e4b054df1848b5f9/t/57489e6e07eaa0105215dc6c/1464376943218/Reversing-Lora-Knight.pdf

Site web de MBED et Texas Instruments

Bibliographie et liens utiles



Matériel principalement utilisé : le brouilleur et les deux modules LoRa

Montage pour tester le brouillage, avec deux terminaux ouverts pour observer le Ping Pong entre les modules LoRa