Projet 33 : “Sécurité : Ingénierie inverse d’un protocole LoRa”

Informatique Microélectronique Automatique

4ème année : Systèmes communicants

Réalisé par : NAANAA Oumaima

Encadré par :

Mr. VANTROYS Thomas
Mr. BOE Alexandre
Mr. REDON Xavier

Année universitaire 2016-2017

Remerciements

Je remercie Mr.Alexandre BOE ,Mr.Xavier REDON et Mr.Thomas VANTROYS pour leur encadrements et leur disponibilité .

Je tiens à remercier également Mr Thierry FLAMEN pour son aide et ses conseils précieux .

Sommaire

SOMMAIRE :

Introduction

Dans le cadre de ma formation en Informatique Microélectronique et Automatique en 4ème année j'ai réalisé un projet en sécurité intitulé ingénierie inverse de protocole réseau.

Dans ce rapport , je décrirai le protocole LoRa . Je m'intéresserai plus au format du signal émis par un module LoRa dans le but de vérifier le format de la trame LoRa.

Pour cela, j'ai d'abord effectué la communication entre un émetteur et un récepteur LoRa que je détaillerai dans la première partie .

Ensuite , j'ai réalisé des mesures qui consistent à décoder la trame LoRa . Dans cette partie , j'ai intercepté la communication en essayant de visualiser , à l'aide d'un oscilloscope adéquat , le signal émis dans l'air par l'antenne du module LoRa.

1-Présentation du protocole LoRa

1-1 Description du LoRa

LoRa *“ Long Range “* est un protocole réseau qui permet de réaliser une communication à longue portée avec une faible consommation d’énergie à un débit compris entre 0,3 et 50 Kbps.
Cette technologie est particulièrement utilisée dans des applications du type : Objets connectés , M2M , bâtiments intelligents .

En comparant LoRa aux autres réseaux utilisés dans des applications pareilles , on peut distinguer les avantages suivant :
-Sa principale fonction : réseau de collecte sur plusieurs kilomètres de distance avec une faible consommation.
-Coût moins cher
-S’adapte au canal

L'inconvénient de LoRa par rapport à la 3G ou WiFi est son bas débit , mais suffisant pour des objets connectés .

Présentation du module LoRa

Pour réaliser la communication avec le protocole LoRa , j’ai utilisé deux cartes Adafruit Feather contenant un module RFM95 LoRa .

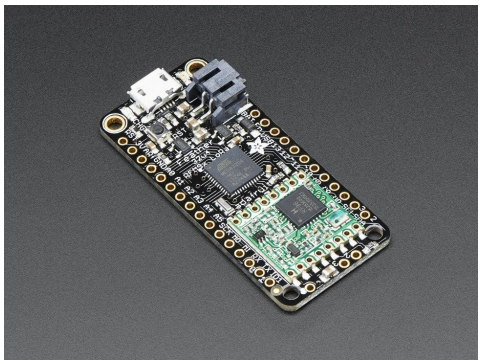


Image 1 : Feather Adafruit M0

Cette carte est composée principalement d’un microcontrôleur , ce qui nous permettra de passer par le logiciel Arduino IDE afin de générer les codes d'émission ainsi que de reception .

Ce qui nous permettra d’avoir LoRa est le module RFM95 déja présent sur la carte .

Afin de tester la communication entre deux cartes feather , il faut d’abord installer le Adafruit SAMD sur Aruino IDE ce qui permettra à l’IDE de reconnaître les Feather , ensuite importer la bibliothèque RadioHeadLibrary grâce à laquelle on va pouvoir configurer les modules RFM95

Pour faire un premier test, j'ai utilisé les codes exemples proposés par la datasheet Adafruit Feather M0 .

Le module RFM95 propose deux modes de modulation , le mode FSK/OOK , et le mode LoRa . En jouant sur les registres définit dans les fichier ".h" ainsi que les fonctions qui les gèrent dans le fichier ".cpp" dans la bibliothèque RadioHead on peut basculer d'un mode à l'autre . On s'intéresse à la modulation LoRa définit par défaut dans les codes exemples .

1-2 Modulation LoRa :

La modulation LoRa utilise la modulation par étalement spectral (Spread spectrum modulation) et des techniques de détection d'erreurs pour une robustesse et portée importante de la communication radio contrairement à la modulation FSK/OOK.

On peut distinguer trois éléments de performances de la modulation LoRa :

La bande passante

La bande passante est la différence entre la fréquence minimale et maximale . Le module RFM95 propose une plage de bandes passantes allant de 7.8 KHz à 500 KHz pour deux fréquences 434 MHz et 868 MHz.

Spreading Factor

Le facteur d'étalement est le nombre de bits codés pour un symbole. Ce qui va nous aider pour déterminer la taille du paquet . On nous propose sept configurations pour chaque bande passante (entre 6 bits par symbole et 12 bits par symbole par un pas de 1).

Coding rate CR:

Le taux de codage est le point fort du réseau LoRa , cela lui permet d'améliorer sa robustesse. On calcule de manière cyclique l'erreur et sa correction. Cela nous permet d'avoir une bonne performance certes mais entraîne également une surcharge sur la transmission de données.

En choisissant les configurations utilisant un minimum de données , cela va nous permettre de réduire la taille du paquet dans le but de simplifier son décodage .

1-3 Structure du paquet LoRa :

LoRa utilise 2 types de paquets : implicite ou explicite .

La différence est dans l’ajout d’un en-tête et un CRC entre le préambule et les données pour le mode explicit . Cet en-tête nous informe de la taille du préambule , le CR utilisé et la présence d’un CRC de 16 bits pour les données.

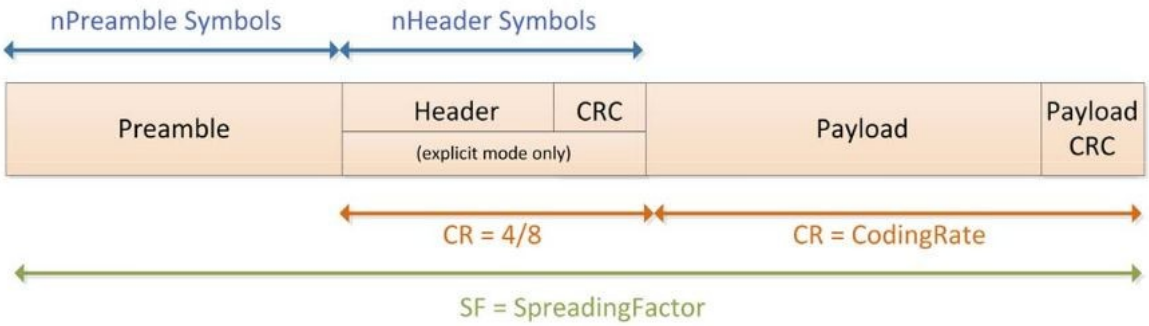


Image 2 : Format de la trame LoRa

Preamble :

Le préambule sert à synchroniser le récepteur avec le flux de données entrant . Sa taille est de 12 symboles définit par défaut . Elle modifiable via le registre “PreambleLength” avec la fonction “setPreambleLength” . Le minimum qu’on peut avoir est de 10 symboles . On peut noter que la taille du preambule n’affecte en aucun cas la communication mais il faut obligatoirement définir la même taille du coté du recepteur et de l’emetteur , sinon le maximum qui est de 65539 sera pris automatiquement.

Header:

Comme est cité précédemment , il existe deux modes de trames : avec ou sans entête . Il est préférable d’utiliser le mode explicit mais dans notre cas , il est plutôt plus facile d’utiliser le mode implicit afin de réduire la taille de la trame.

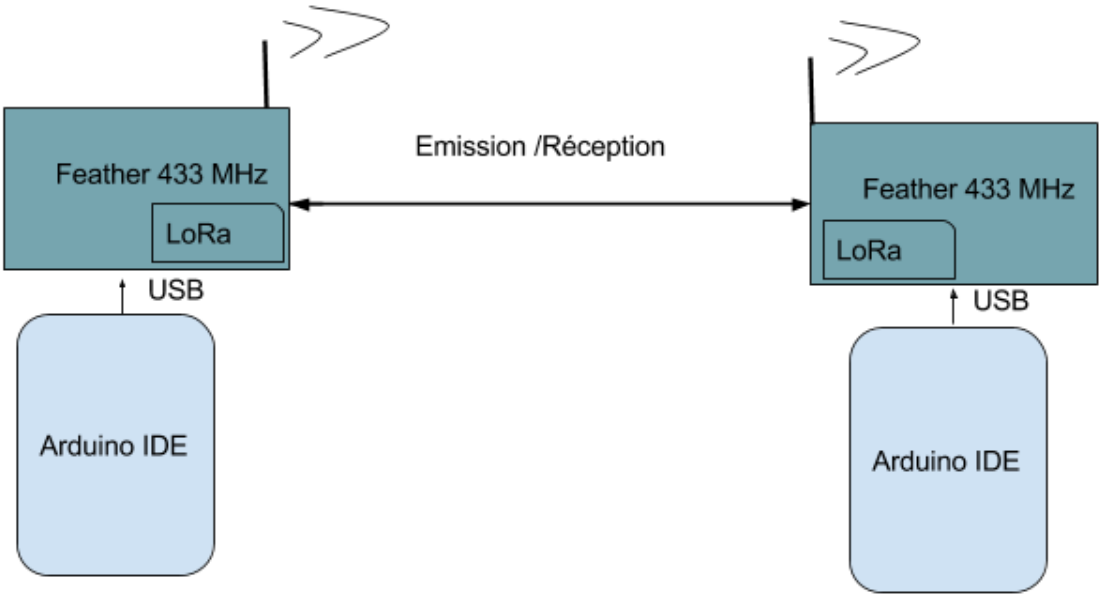
Payload :
Il contient les données codées soit par le taux de codage choisis dans le mode explicit ou par le taux de code prédéfinis dans le registre convenant en mode implicit. Un CRC optionnel des données pourra être présent.

2-Travail réalisé :

2-1- Communication entre deux modules :

Après avoir installé les bibliothèques d'Adafruit nécessaires et soudé un fil de longueur 16.5 cm qui nous sert d'antenne de fréquence 434MHz , j'ai réalisé le montage suivant :

2-1-1 Montage :



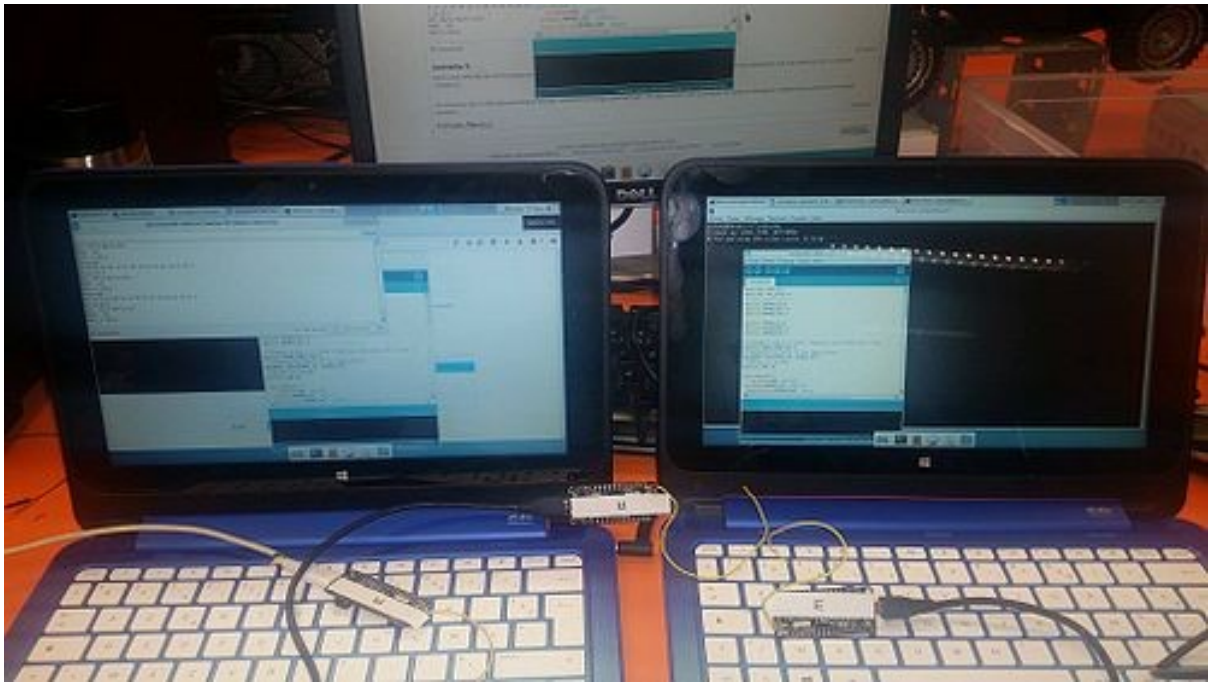


Image3 : Montage un émetteur et deux récepteurs

Les codes emission/réceptions contiennent deux parties :

En-tête commun :

```
#include <SPI.h>
#include <RH_RF95.h>

//Définition des variables globales relatives à l'instance du driver radio
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3

// choix de la fréquence  434MHz
#define RF95_FREQ 434.0

// Instance du driver radio
RH_RF95 rf95(RFM95_CS, RFM95_INT);
```

Cette configuration est donnée par le constructeurs , les valeurs des variables globales dépendent du type de carte utilisée . Ici on utilise M0.

Initialisation émetteur :

```
void setup()
{
pinMode(RFM95_RST, OUTPUT);
digitalWrite(RFM95_RST, HIGH);
while (!Serial);
Serial.begin(9600);
delay(100);
Serial.println("Feather LoRa TX Test!");
// manual reset
digitalWrite(RFM95_RST, LOW);
delay(10);
digitalWrite(RFM95_RST, HIGH);
delay(10);
while (!rf95.init()) {
Serial.println("LoRa radio init failed");
while (1);
}
Serial.println("LoRa radio init OK!");
```

```
if (!rf95.setFrequency(RF95_FREQ)) {
Serial.println("setFrequency failed");
while (1);
}
Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Paramètre par défaut : 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf = 7
(128chips/symbol), CRC on

rf95.setTxPower(23, false);
}
int16_t packetnum = 0; // compteur qui s'incrémente une fois une emission est effectuée
```

Boucle d'émission :

Ce code nous permets d'envoyer un "Hello Word" et attendre un réponse du recepteur . On affiche également le RSSI :

```
void loop()
{
Serial.println("Sending to rf95_server");
// On envoie hello world
char radiopacket[20] = "Hello World #";
itoa(packetnum++, radiopacket+13, 10);
Serial.print("Sending "); Serial.println(radiopacket);
radiopacket[19] = 0;
Serial.println("Sending..."); delay(10);
rf95.send((uint8_t *)radiopacket, 20);
Serial.println("Waiting for packet to complete..."); delay(10);
rf95.waitPacketSent();
// Now wait for a reply
uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
uint8_t len = sizeof(buf);
Serial.println("Waiting for reply..."); delay(10);
```

```
if (rf95.waitAvailableTimeout(1000))
{
// Should be a reply message for us now
if (rf95.recv(buf, &len))
{
Serial.print("Got reply: ");
Serial.println((char*)buf);
Serial.print("RSSI: ");
Serial.println(rf95.lastRssi(), DEC);
}
else
{
Serial.println("Receive failed");
}
}
else
{
Serial.println("No reply, is there a listener around?");
}
delay(1000);
}
```

Initialisation réception :

```
void setup()
{
pinMode(LED, OUTPUT);
pinMode(RFM95_RST, OUTPUT);
digitalWrite(RFM95_RST, HIGH);
while (!Serial);
Serial.begin(9600);
delay(100);
Serial.println("Feather LoRa RX Test!");
// manual reset
digitalWrite(RFM95_RST, LOW);
delay(10);
digitalWrite(RFM95_RST, HIGH);
delay(10);
while (!rf95.init()) {
Serial.println("LoRa radio init failed");
while (1);
}
Serial.println("LoRa radio init OK!");
if (!rf95.setFrequency(RF95_FREQ)) {
Serial.println("setFrequency failed");
while (1);
```

```
}
Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

// Configuration après initialisation 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf =
128chips/symbol, CRC on

rf95.setTxPower(23, false);
}
```

Boucle de réception :

Ce programme nous permet d'afficher le message reçu "Hello world" et répondre par un "Hello back to you" :

```
void loop()
{
if (rf95.available())
{
// Should be a message for us now
uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
uint8_t len = sizeof(buf);
if (rf95.recv(buf, &len))
{
digitalWrite(LED, HIGH);
RH_RF95::printBuffer("Received: ", buf, len);
Serial.print("Got: ");
Serial.println((char*)buf);
Serial.print("RSSI: ");
Serial.println(rf95.lastRssi(), DEC);
delay(10);
// Send a reply
uint8_t data[] = "And hello back to you";
rf95.send(data, sizeof(data));
rf95.waitPacketSent();
Serial.println("Sent a reply");
digitalWrite(LED, LOW);
}
else
{
Serial.println("Receive failed");
}
}
}
```

2-1-2 Tests et resultats :

J’ai pu voir la communication par ping-pong sur le moniteur série de l’IDE :

Image 4: Moniteur série du côté d'émission

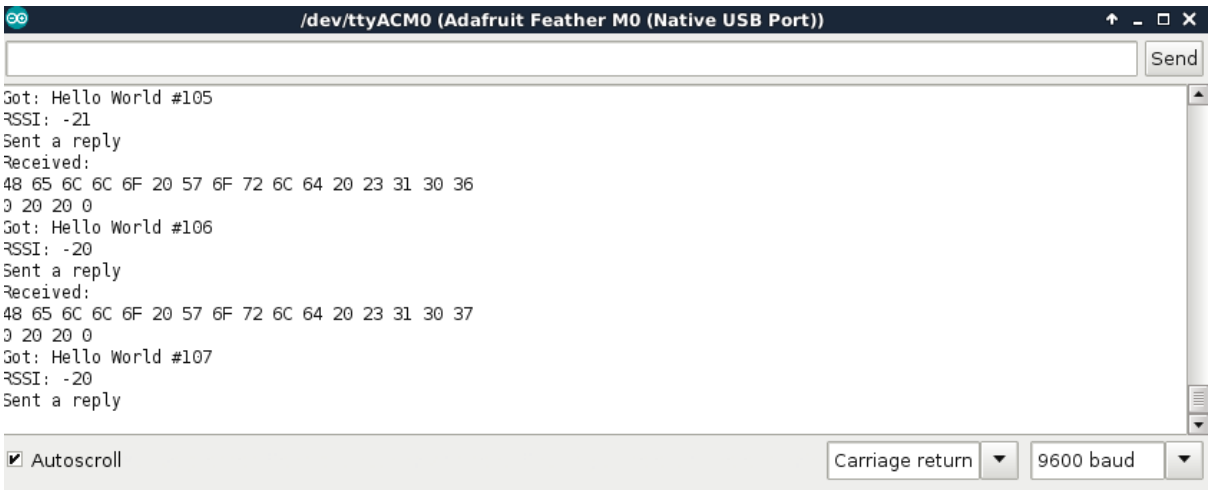


Image 5 : Moniteur série du coté du récepteur

1er problème rencontré :

Je n’ai pas réussi à afficher la réponse “and hello back to you” sur le moniteur d'émission , même si j’étais sur de l’envoyer par le récepteur avec le message “ Sent a reply” . L'émetteur n’avait également aucun problème de réception puisqu’il n’affichait pas le message d’erreur “Receive failed “.

1er test effectué :

Pour tester l'influence de plusieurs feather connectés à la fois sur le RSSI (Received Signal Strength Indication) , j'ai ajouté un 3 eme module configuré en réception avec le même code .

Résultat :

j'ai remarqué que le RSSI ne change quasiment pas (entre |21| et |19|) . On peut conclure que la présence de plusieurs récepteurs n'affecte pas la puissance du signal transmis .

2-2 Décodage de la trame LoRa :

2-2-1 Montage :

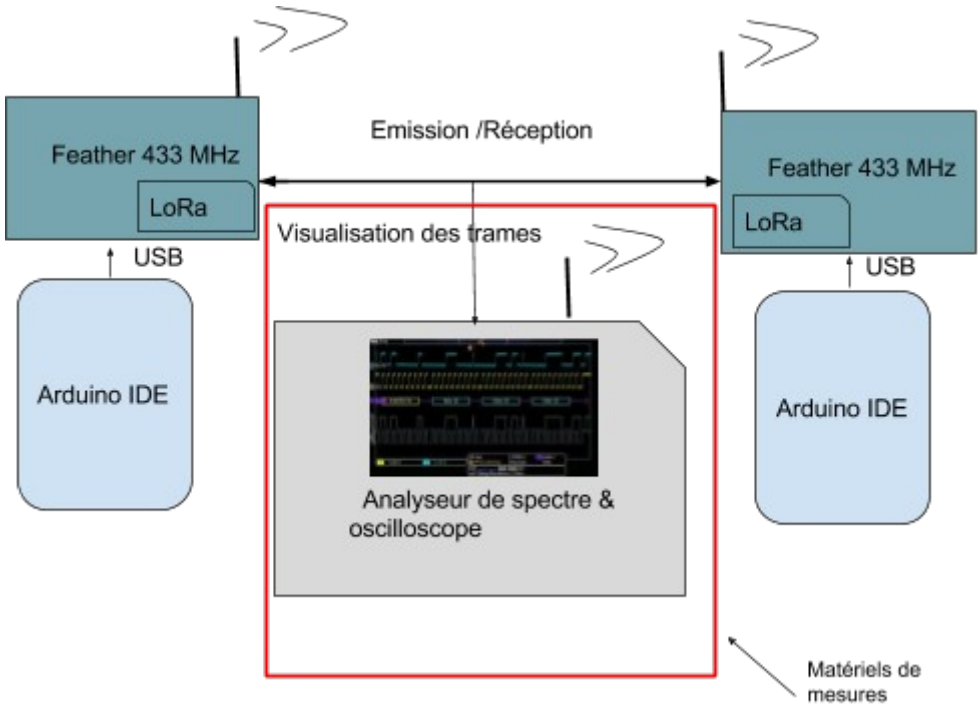


Image 6: Montage pour décoder la trame LoRa

Pour pouvoir décoder la trame LoRa , j'ai commencé par changer les codes en remplaçant les messages "Hello word" et "And hello back to you" par des "0" ou des "1" .

Code d'émission :

```
void loop() {
 Serial.println("Sending to rf95_server");
 int i;
 char radiopacketS[RH_RF95_MAX_MESSAGE_LEN];
```

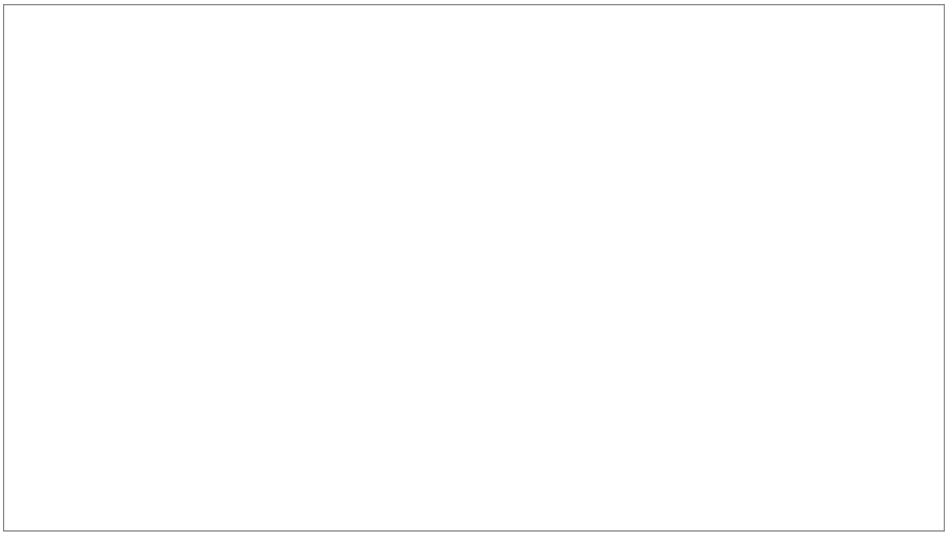
```
 for(i=0;i<RH_RF95_MAX_MESSAGE_LEN;i++){
          while(Serial.available()<=0) delay(10);
          radiopacketS[i] =Serial.read();
 }
#if 0
 //itoa(packetnum++, radiopacket+13, 10);
  char radiopacketS[RH_RF95_MAX_MESSAGE_LEN];
#endif
 Serial.print("Sending ");
 for(i=0;i<RH_RF95_MAX_MESSAGE_LEN;i++){

 Serial.print(radiopacketS[i]);
 }
 for (i=0;i<8;i++) {
          radiopacketS[i] = 0;
 }
 while(1){
 rf95.send((uint8_t *)radiopacketS, RH_RF95_MAX_MESSAGE_LEN);
 rf95.waitPacketSent();
 uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
 uint8_t len = sizeof(buf);
  if (rf95.waitAvailableTimeout(1000))
 {
          // Should be a reply message for us now
          if (rf95.recv(buf, &len))
  {
                   Serial.println((char*)buf); // on affiche ce qu'on reçoit du recepteur

          }
          else
          {
          Serial.println("Receive failed");
          }
 }
 else
 {
          Serial.println("No reply, is there a listener around?");
 }
 delay(1000);
}
```

}

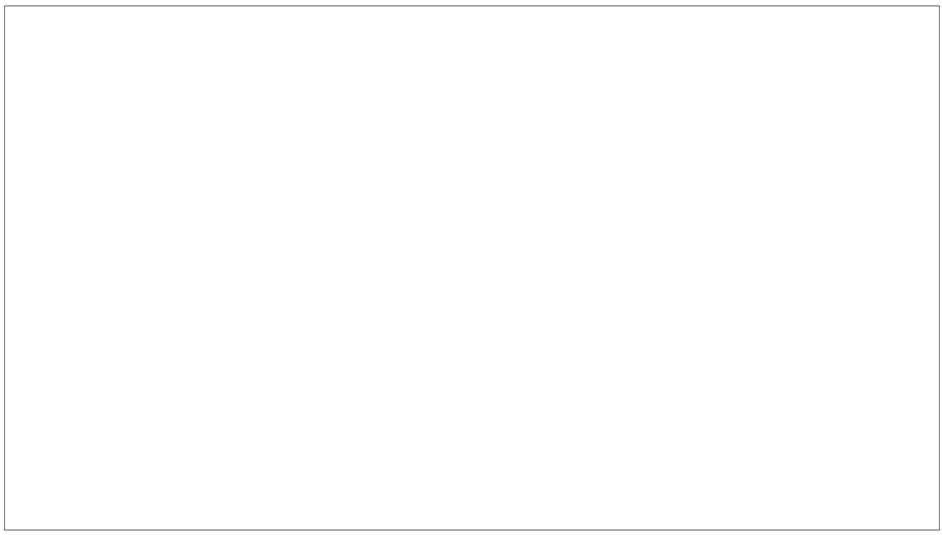
Ce programme nous permet d'envoyer 7 char de 8 bits ce qui nous fait un total de 56 bits. Sachant qu'on envoie 7 bits par symbole (SF= 7) on doit donc visualiser 8 sauts de phase chacun correspondant à un symbole .
Un SF=7 nous donne 128 chirps/ symbol , ce qui donne 1024 chirps à compter pour vérifier le payload .

Du coté de réception il suffit de renvoyer ce qu'on reçoit pour assurer la bonne transmission (Ping-pong du tableau de 7 char) .

Message envoyé :



Message reçu:



2-2-2 Tests et résultats:

J’ai visualisé d’abord sur l’analyseur de spectre le signal émis à l’aide d’une antenne patch tout en vérifiant la fréquence 434MHz:

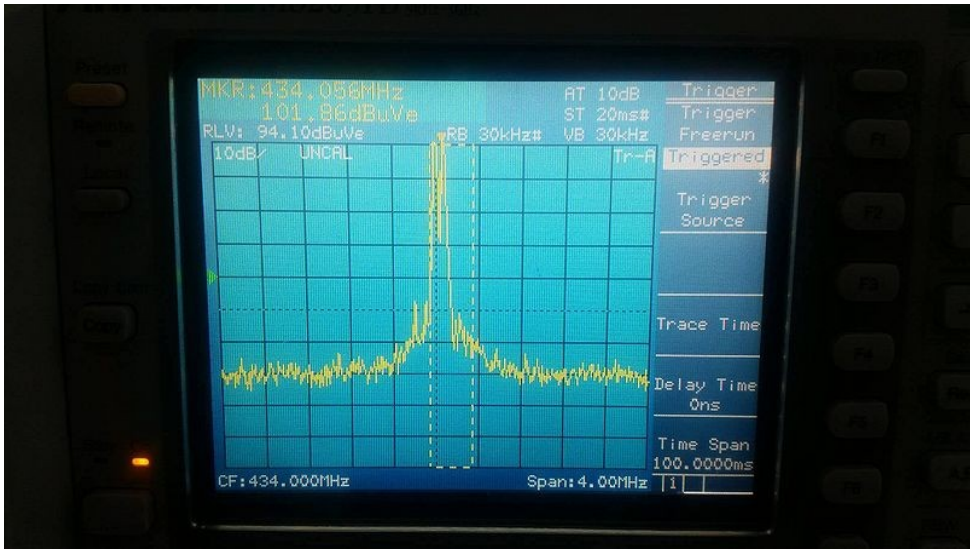


Image7 : Spectre du signal modulé en LoRa

Ensuite , en passant par un T on connecte l’antenne patch à l’oscilloscope 1GHz par un câble coaxial.

2ème problème rencontré :

on posant chaque module sur l’antenne patch ,j’ai pu voir les deux signaux suivants:

Image 8 : Emission Lora

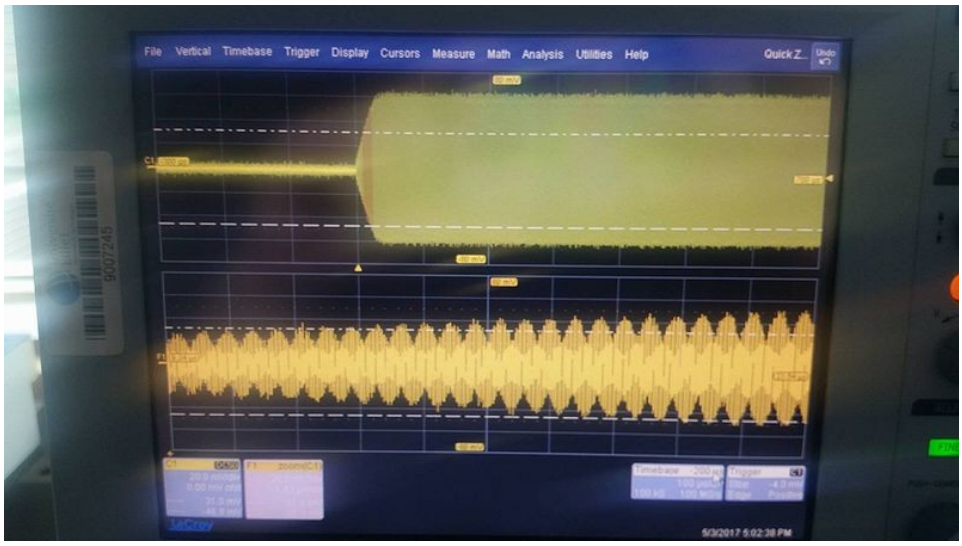


Image 9 : Réception LoRa

En comparant avec les propriété de la modulation LoRa qui en étalement spectral , le signal n'est identique à ce que je devrai avoir . Cela est du à l'antenne patch qui n'est pas adaptée (800MHz).

Solution :

En évitant de passer par l'antenne patch , et en connectant directement l'antenne du module d'émission (fil + GND) à l'oscilloscope via un câble coaxial , j'ai pu visualiser le signal prévu :

Image10 : Emission LoRa sans antenne patch

2-2-3 Analyses :

On peut remarquer la présence de deux trames , une correspond à la première émission et l'autre au retour amis par le module récepteur .

Le signal d'en bas qui est un zoom de la trame , nous montre les chirps qu'on devrait avoir .

En se basant sur la méthode suivant , on pourra décoder ce signal .

2-2-4 Méthode de décodage :

Le but est d'avoir la taille minimale possible pour rendre plus facile le parcours de la trame . Il faut d'abord penser à changer la bande passante et passer de 125 KHz qui a un débit de 293 bps à une bande passante de 8 KHz.

Preamble :
Comme la taille du préambule n'affecte pas la transmission , on peut donc la réduire au minimum configurable qui est 6 avec la fonction *setPreambleLength .* Ce qui nous fera 10 symboles pour le preambule . Il faudra donc trouver 10 sauts de phases.

Header :
En choisissant le mode SF=6 on force l'utilisation du mode implicit tout en réduisant également le nombre de chips par symbole:65 chips/symbol au lieu de 128 . Il faut penser à changer la taille du tableau de char en utilisant 6 char au lieu de 7 .

Payload :

En utilisant maintenant SF= 6 , on envoie donc 6 char (111111 par exemple) et on s'attend à avoir 8 sauts de phase et 8*65 chirps ce qui donne 520 chirps pour le payload qui reste quand même beaucoup à compter pour s'en assurer . On se basera donc uniquement sur le comptage de sauts pour compter 8 symboles. Il faut noter que la taille max du payload est de 251 bits , même si on envoie que 6 bits de 1 le reste est mis à 0 . Cette taille peut être réduite en utilisant une bande passante plus petite

CRC :
Comme on utilise le mode implicite , on n'est donc pas sur d'avoir un CRC , on vérifiera la présence de ce dernier sachant qu'il a une taille de 16 bits ce qui faut 3 symboles à peu près .

Conclusion

En plus des avantages techniques du protocole LoRa , ce qui le rend puissant est son taux de codage calculés de manière cyclique ce qui entraîne une surcharge de transmission de données .

Cette surcharge affecte non pas seulement le débit , mais aussi la taille de la trame qui reste importante si on veut la décoder , même après l’avoir réduite au minimum possible .

Concernant la sécurité d’envoie de données , si on ne passe pas par un réseau LoraWAN , il est plus difficile de sécuriser la transmission . Ce qu’on peut faire pour éviter que notre voisin qui utilise LoRa reçoit notre message, il est conseillé d’utiliser des modes de configurations différents car même avec une fréquence différente on arrive à communiquer .

Pour envoyer un paquet sans en-tête et être sûr de l'absence d’erreur , on peut faire appel à la redondance d’envoie sur des canaux différents .

Finalement , le seul inconvénient de LoRa est la taille de sa trame qui reste toujours plus importante que celle de son concurrent SigFox mais qui assure une qualité de service meilleure.

Annexes

Datasheet RFM95 :http://www.hoperf.com/upload/rf/RFM95_96_97_98W.pdf
Datasheet Feather : https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-m0-radio-with-lora-radio-module.pdf