

MOUNEU
Olivier
FIA-4

TP Réseau Sans-Fil : Radio

Matériel / Logiciels :

Arduino
Teensyduino

[RadioHead: RadioHead Packet Radio library for embedded microprocessors
\(airspayce.com\)](http://airspayce.com)

Module radio RF22
Antenne 433 MHz

Partie 1 :

La bande de fréquence utilisée est 433 Mhz.

Une antenne Wi-Fi possède une fréquence de 2.4 GHz, on ne pourra pas l'utiliser pour recevoir des données sur une fréquence de 433Mhz

Partie 2 :

Le nœud écoute sur le canal 0.

Taille du message :

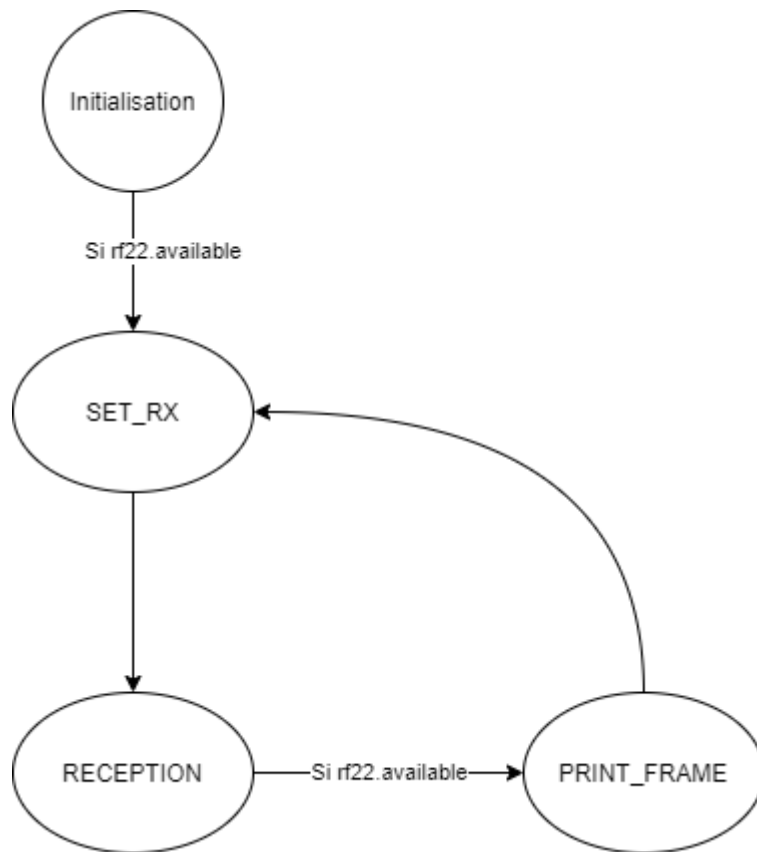
Il faut **5 octets** pour transmettre le message

Le 0x ne compte pas (il signifie qu'on envoie de l'hexadécimal)

Un caractère hexadécimal prend 4 bits ($\frac{1}{2}$ octet)

On a un message de 10 caractères hexadécimal.

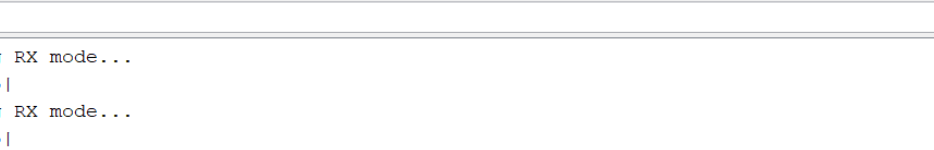
Schéma réception de données (réception) :



Partie 3 :

Le message envoyé dans le code est 0x65.

La capture ci-dessous présente le résultat à la réception du message.



```
COM5 (Teensy) Serial
|
Activating RX mode...
[177113]65|
Activating RX mode...
[177114]65|
Activating RX mode...
[177115]65|
Activating RX mode...
[177116]65|
Activating RX mode...
[177117]65|
Activating RX mode...
[177118]65|
Activating RX mode...
[177119]65|
Activating RX mode...
[177120]65|
Activating RX mode...

☒ Défilement automatique  Nouvelle ligne  Effacer la sortie
```

On voit que le message produit par notre arduino a bien été reçu.

Les fonctions utilisées pour l'émission :

rf22.send(données, taille)

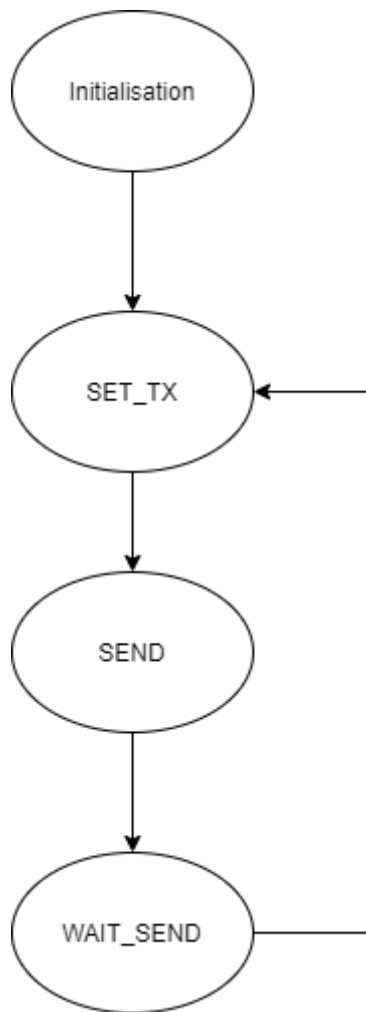
La variable *données* est un tableau contenant des variables de type `uint8_t` (la notation de ce tableau peut s'écrire sous la forme d'un pointeur : `uint8_t *`)

La variable *taille* est représente la taille en octet du message, elle est de type `uint8_t`.

rf22.waitPacketSent()

Cette fonction ne prend aucun paramètre. Elle permet d'attendre la fin d'émission d'un message par le module rf22.

Schéma envoi de données (émission) :



Code émetteur :

Ce code envoie le message 0x65 via la bande 433MHz sur le canal 0.

Les états de l'émetteur (SET_TX, SEND et WAIT_SEND) définis dans le graphe ci-dessus sont regroupés dans un même état 'SEND' dans le code. En effet ces états doivent nécessairement s'enchaîner pour que le processus d'envoi de données se déroule normalement.

La fonction *setup* définit les paramètres du module RF22. La fonction *loop* envoie en continu les mêmes données via le module RF22. Enfin d'envoyer des données, l'antenne est dans un premier temps passé en mode Tx (envoi) puis on ordonne l'envoi du message via la fonction *send* enfin on attend que l'envoi soit terminé (fonction `waitPacketSent` du module RF22).

```
#include <SPI.h>
#include <RH_RF22.h>
#define SET_TX 1
#define SEND 2
#define WAIT_SEND 3
#define REPLY 4
#define canal 0

RH_RF22 rf22(SS,9);
uint8_t state;
uint8_t rxbuf[RH_RF22_MAX_MESSAGE_LEN];
uint8_t rxbuflen = RH_RF22_MAX_MESSAGE_LEN;
uint8_t rxlen = RH_RF22_MAX_MESSAGE_LEN;
int rxf = 0;

// the setup routine runs once when you press reset:
void setup() {

  Serial.begin(11200);
  delay(5000);
  SPI.setSCK(14);
  if(!rf22.init()){
    Serial.println("init failed");
    while(1);
  }else
    Serial.println("init OK");
  rf22.setTxPower(RH_RF22_TXPOW_8DBM);
  rf22.setModemConfig(RH_RF22::GFSK_Rb125Fd125);
  rf22.setFrequency(433.1+canal*0.1, 0.05);
  state = SEND;
  delay(3000);
  Serial.println("On to the main loop...");
}
```

```
// the loop routine runs over and over again forever:
void loop() {
  switch(state){
    case SEND:
      Serial.println("Activating TX mode...");
      rf22.setModeTx();

      uint8_t data[5];
      data[0] = 0x65;

      rf22.send(data, sizeof(data));

      Serial.println("Sending...");

      rf22.waitPacketSent();

      break;

    break;
    default:
    break;
  }
}
```