ETML

Modules de transmission 868MHz

2017

Simonet Yoann
16.06.2017

Table des matières

1 Pré-étude

1.1 Description détaillée des objectifs à atteindre.

1.1.1 Description du projet.

Il faut concevoir un module radio 868MHz permettant d'échanger des données entre deux ou plusieurs modules.

Ces modules doivent être simple à utiliser, ils doivent se comporter comme un port série, donc des caractères ascii envoyés en UART.

On doit aussi avoir la possibilité de faire du broadcast.

1.1.2 Cahier des charges.

Alimentation 3.3V
Signaux UART 3.3V : RX, TX, RTS, CTS.
Signal de statut de la connexion : LINK.
Signal de requête de connexion : CONNECT.
Signal de reset actif bas : RSTB.
IOS génériques : GPIO0, GPIO1, GPIO2, GPIO3.
Il faut utiliser le module radio nRF905.
Le microcontrôleur doit obligatoirement être un PIC32MX.
Une petite mémoire flash pour pouvoir y stocker un identifiant unique (UUID).
L'antenne est directement implémenté sur le PCB.
Le module sera connecté avec un connecteur mezzanine.
Idéalement il devra aussi avoir un connecteur compatible avec un xbee.

1.1.3 Option possible sur les 4 GPIO

Quelques exemples d'utilisation possible des GPIO.

Enclencher un mode sleep.
Utiliser le 4 GPIO comme des ports sans file, exemple: on mette des 1 sur les GPIO d'un module et les GPIO d'un autre module passent à 0.
On peut les utiliser pour choisir des modes de transmission par exemple plus d'un caractère ASCII à la fois.

1.2 Choix des composants.

1.2.1 Composants imposés

PIC32MX**:**
On a besoin de:

- 4 GPIO
- Une sortie LINK et une entrée CONNECT (2IO)
- Un UART (4IO)
- Un i2C (2IO) pour la mémoire externe.
- Un SPI (4IO) pour communiquer avec le module radio.
- 6 IO pour commander le module radio.
- Un clear.
- Les 2 pines de programmation (PGC, PDC).

Nous avons besoins au minimum de 25 I/O je choisis donc le PIC32MX120 44 pin (PIC32MX120F060D).

Remarque:
Un pic18 aurait très bien pu faire l'affaire.

Module radio: nRF905.

1.2.2 Composants choisis

Mémoire externe: 24AA02UID cette IC de mémoire a pour avantage de déjà avoir un ID unique est d'être très bon marché.

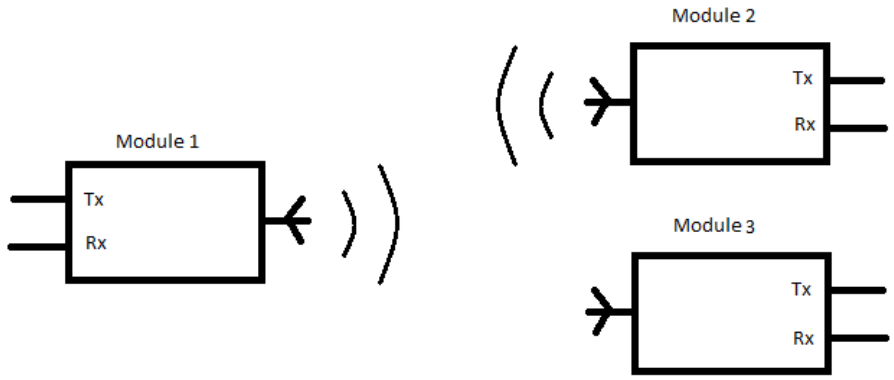
Quartz externe pour le Module radio: ABM3C-16.000MHZ-D4Y-T.

2x Connecteur barrette 10 pôles traversant (pour la compatibilité Xbee) :
PREC010SAAN-RC.

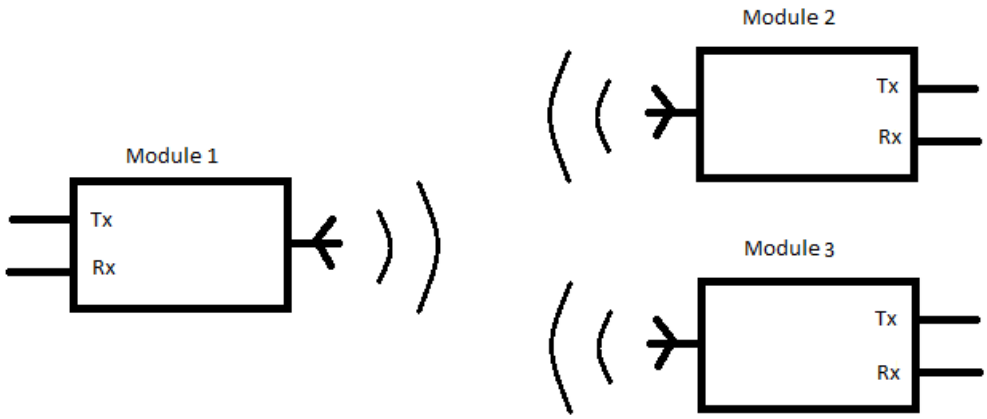
Connecteur mezzanine 31 pôles avec détrompeur : DF9-31S-1V(69)
Je choisi volontairement un connecteur avec trop de pole pour qu'il soit plus robuste.

1.3 Schéma général du système et méthode de communication

Un module doit pouvoir parler seulement avec un autre module (exemple le1 avec le 2).



Mais il doit aussi pouvoir envoyer un message à tous les modules en même temps.



On aurait donc besoin d'une adresse de broadcast exemple 00.00.00.00.mais malheureusement le nRF905 ne prévois pas d'option pour faire du broadcast (ou alors je ne l'ai pas trouvé dans le datasheet).

Il va donc falloir un réseau avec deux couches : la Couche utilisateur et la couche radio.

2. Couche utilisateur:
C'est ce que l'utilisateur vas devoir envoyer au module en UART.

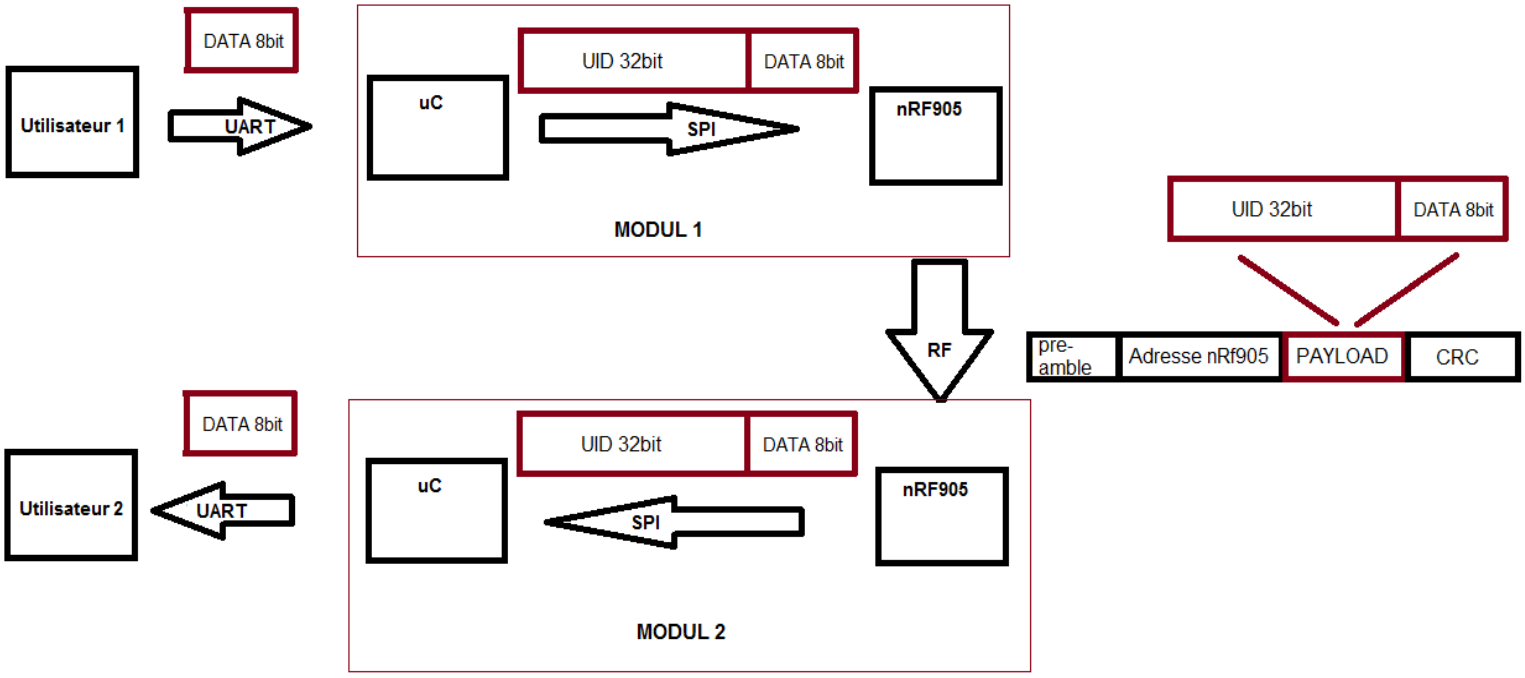
Il faut définir l'UID de destination (l'UID sera utilisé comme une adresse) avant de commencer à communiquer, l'UID 00.00.00.00 sera réservé pour le broadcast.

On pourrait aussi transmettre l'UID source en plus de l'UID de destination pour des applications plus compliquées.

1. Couche Radio générer par le nRF905 :

Les adresses du nRF905 seront identique dans tous les modules radios et c'est le microcontrôleur en allant lire l'UID de la couche utilisateur qui va choisir de jeter ou de garder le paquet 8 bit de DATA.

Exemple de l'envoi d'un caractère ASCII de l'utilisateur 1 à l'utilisateur 2:

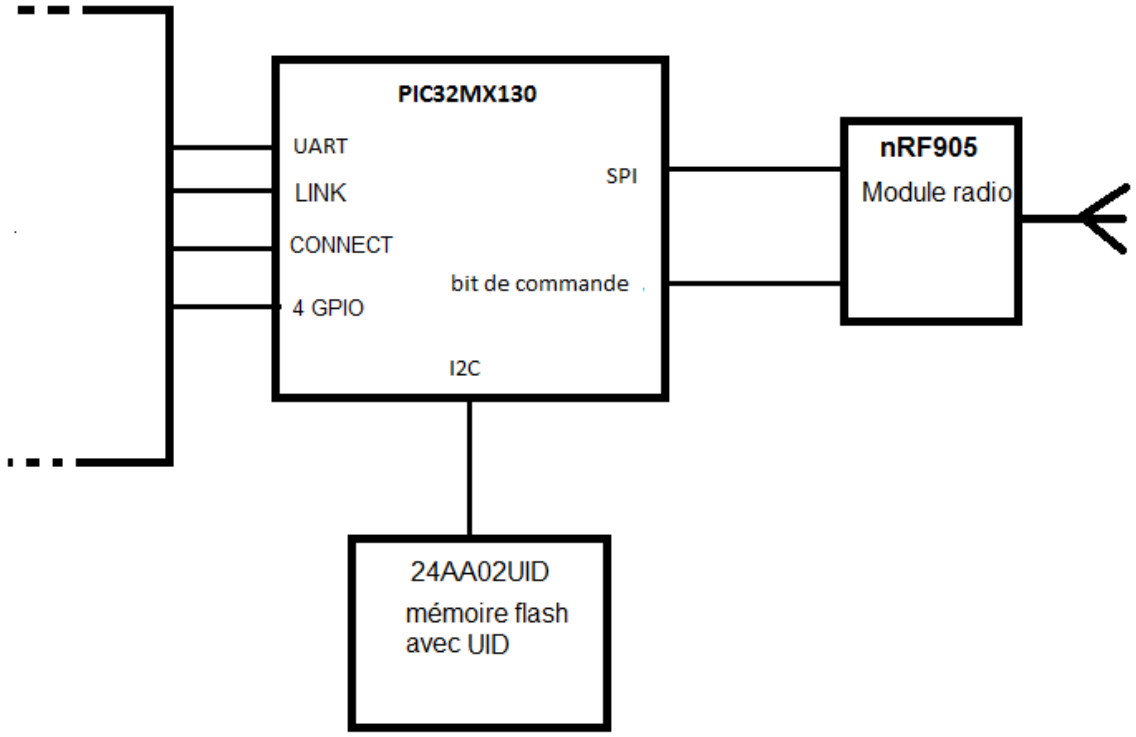


L'utilisateur 1 envoie son caractère ASCII (8bit) au module 1 via UART.
Le microcontrôleur ajoute l' UDI de destination et envoie le tout au nRF905 via SPI.
Le nRF905 Tx transmet les informations ,via onde radio en utilisant sa trame de donné spécifique ,L'UID et le caractère ASCII (8bit) sont dans PAYLOAD.Le nRF905 Rx va retransmettre L'UID et le caractère ASCII (8bit) en SPI au microcontrôleur.
Si l'UID correspond à son propre UID il va transmettre le caractère ASCII (8bit) à l' utilisateur 2 via UART.

Remarque:
Je vais prévoir des commandes UART pour aller modifier UID de destination et lire son propre UID.

Transmettre le préambule, les 32 bits d' UID, les 32 bits d' adresse nRF et un CRC pour juste 8bit de DATA n' est pas un solution très efficace, je vais donc aussi prévoir une commande pour envoyer plusieurs caractère ASCII à la fois.

1.4 Schéma bloc du hardware.



Remarque:
Le but est d'avoir le hardware le plus simple possible.

1.5 Evaluation des coûts.

Module radio nRF905 : ~ 5 Fr.

Microcontrôleur PIC32MX120F060D: ~ 3.5Fr

Mémoire externe 24AA02UID : ~0.3 Fr

Composant discret environ 20x : ~ 3 Fr

Connecteur barrette plus mezzanine : ~2 Fr

PCB commende chez Euro circuit : ~ 67 Fr si on en commande plusieurs.

La réalisation de trois PCB + les composant = (67 + 13) *3=~ 240 frs

Remarque:
Ces modules radio 868MHz pourraient être meilleur marché si l'école décidait d'en produire beaucoup car la seul pièce chère c'est le PCB et le prix du PCB dépend du nombre de commande.

1.6 Planning du projet.

Voir annexe.

1.7 Conclusion.

On peut constater que l'utilisation de deux couches de données complique la communication du module, une solution à ce problème serait de trouver un moyen de faire du broadcast directement avec le nRF905.

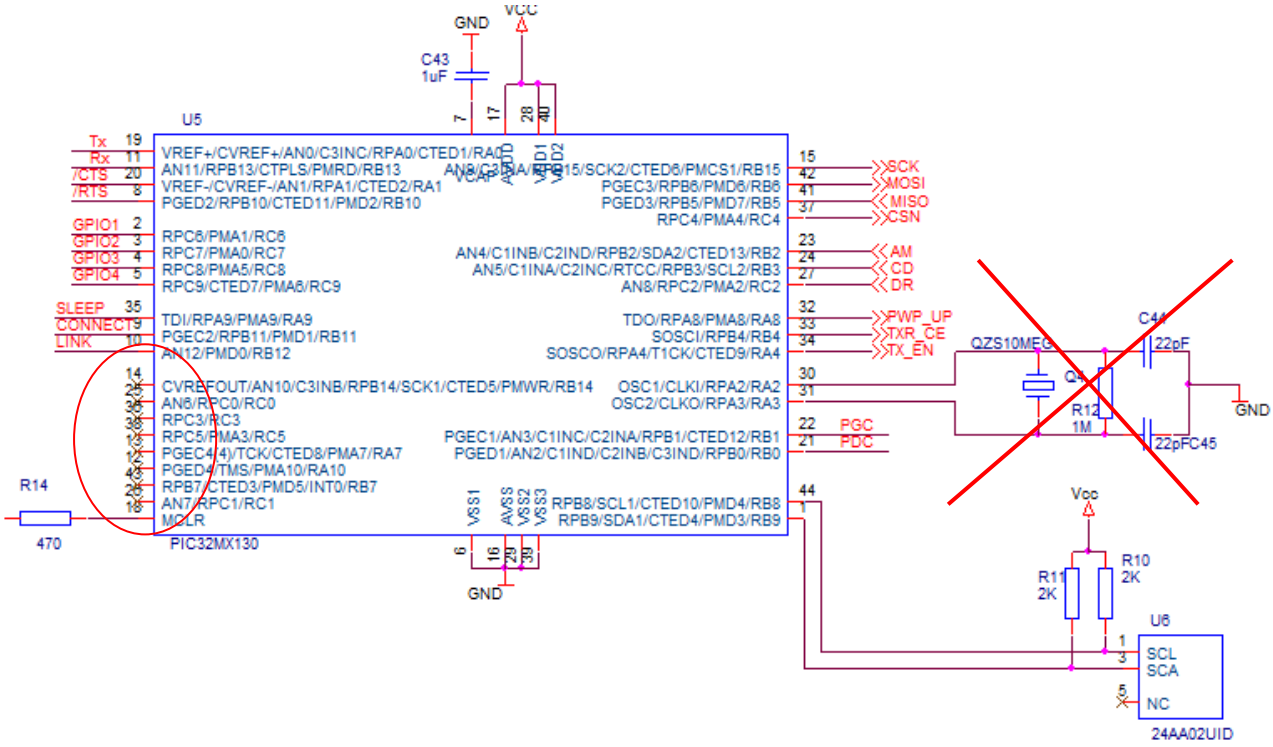
Pour simplifier l'utilisation aux utilisateurs qui veulent juste communiquer entre deux modules, on peut mettre UID de destination par défaut en broadcast (00.00.00.00).
Comme cela ils pourront directement envoyer les données sans rien avoir à configurer.

2 Design

2.1 Redimensionnement du Microcontrôleur.

Duran la pré-étude j' avais choisis d' utiliser un PIC32MX130 44 pin (PIC32MX130F060D).

J'avais choisis d' utiliser un pic 44 pin au lieu d' un pic 28 pin car on avait besoins de 25 IO(voir p.4) donc on ne pouvait juste pas utiliser le pic 28 pin car il manquait 5 pins.



Le problème avec cette solution c'est qu'en plus d'avoir 8 pines non utiliser avec le pic 44 pin, on s'est rendu compte qu'il serait trop gros pour tenir sur une board d'à peu-prés la même taille qu'un module Xbee.

Donc pour regagner 5 IO on a décidé de :

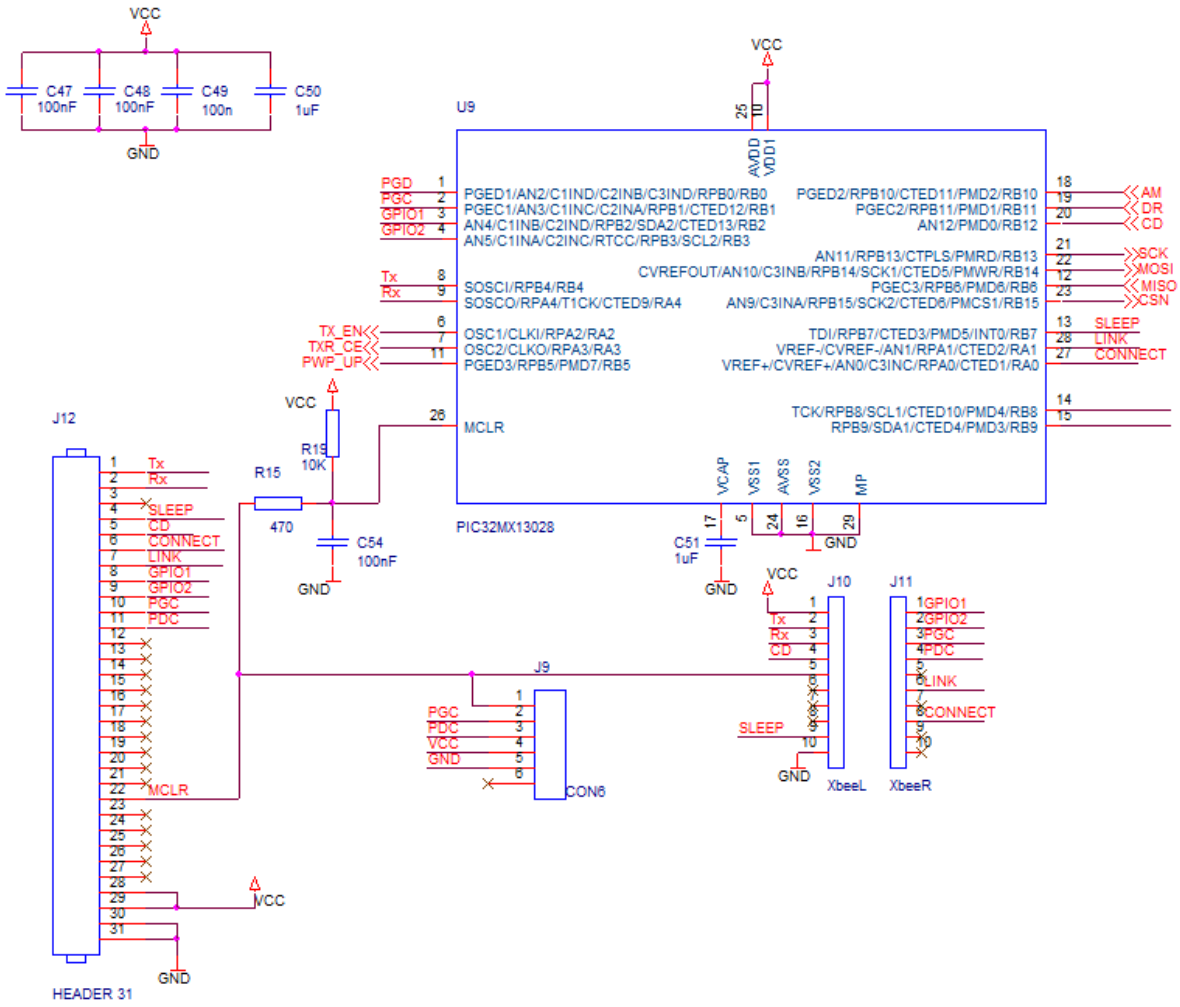
Supprimer l'oscillateur externe (- 2 IO).
De ne pas utiliser le CTS RTC de l'UART(- 2IO).
De mettre les pines PDC PGD sur les GPIO (- 2IO).

Ce qui nous laisse même une IO de libre sur le pic 28 pins pour mettre une entrée sleep.

2.2 Description détaillées du schéma par bloc.

Voir Schéma bloc p.8.

2.2.1 Microcontrôleur.



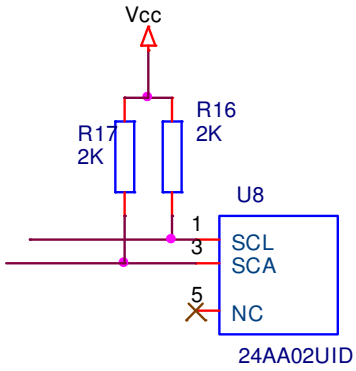
On peut voir que j'ai réussi à mettre toutes les IO que je voulais utiliser sur le PIC 28 pines.

Dimensionnement:

Pour les condensateurs de découplage j'ai décidé de suivre application note du pic et de rajouter un condensateur de 10uF.

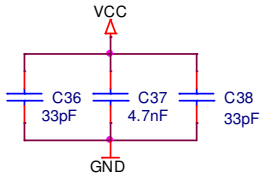
Pour la valeur les composants qui vont sur le MCLR j'ai suivi les indications de monsieur Yersin.

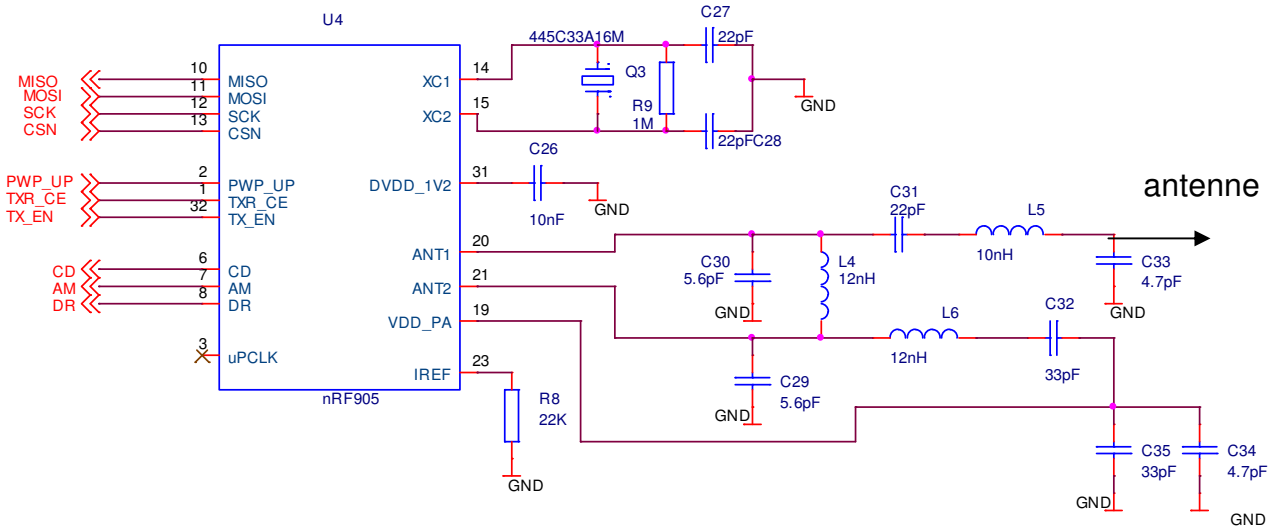
2.2.2Mémoire externe avec UID



Dimensionnement:
Pour la valeur des résistances de PULL-UP j'ai suivi l'application note mais si on veut baisser la consommation on pourra toujours en mettre des plus grande (mais il faudra aussi diminuer la vitesse de communication).

2.2.3Module radio nRF905





Dimensionnement:

Pour le placement et le dimensionnement des composants j'ai suivi l'application note

2.3 Concepts du logiciels

2.3.1 Utilisation standard des modules

Avant de réfléchir à comment va être le code il faut préciser comment vont être utilisé les modules.

Exemple d'une utilisation simple du module :

Mettre un 0 sur l'entrée connecte pour indiquer au module que l'on veut le paramétrer et luis envoyer des commande.

Envoyer en UART l'adresse du module avec le quelle on veut communiquer ou 00.00.00.00 si on veut faire du broadcast.

Définir la taille des trames que l'on veut envoyer (8 bits par défaut).

Mettre l'entrée connecte à 1 et attendre que la patte LINK passe à 1.

Si LINK passe à 1 celas veux dire que les modules sont bien connecter et on a plus qu'à envoyer nos données comme dans une communication UART classique.

2.3.2 Liste des commandes possibles pour paramétrer le module

#Rmyadd : lire son adresse.

#Wmyadd : changer son adresse.

#Wtxadd : changer l'adresse de destination.

#Wwide : changer la taille des trames.

#Wmode1 : choisir le mode de communication simplifié.

#Wmode2 : choisir le mode de communication standard (par défaut).

#Wmode3 : choisir le mode de communication avancé (à coder si j'ai le temps).

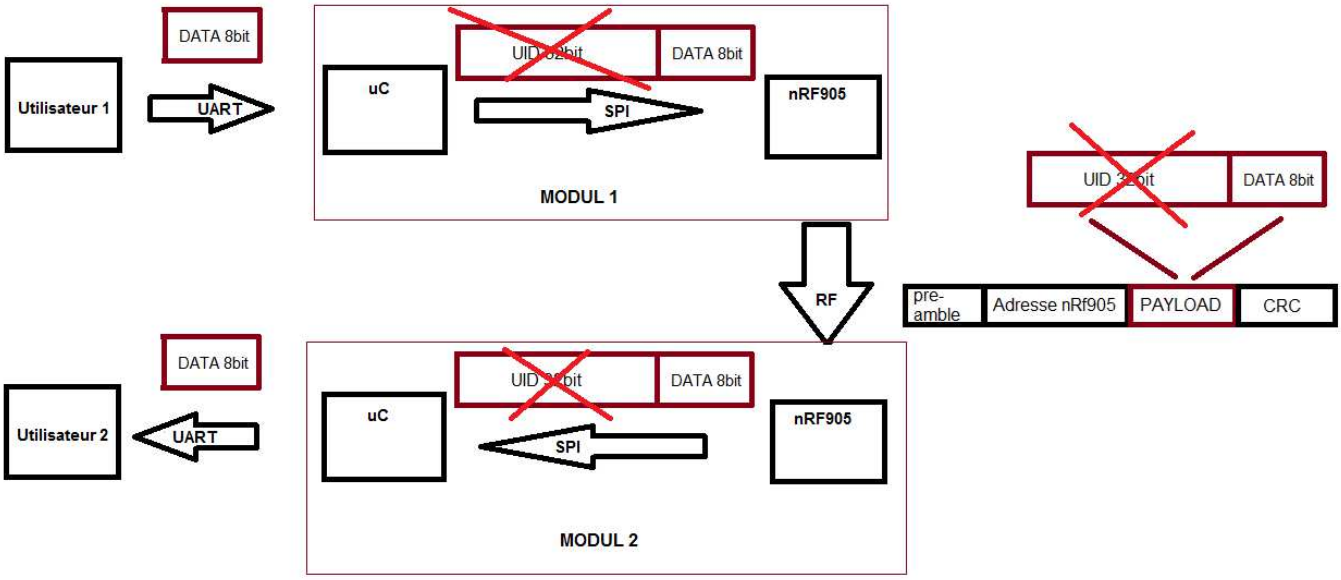
2.3.3 Mode de communication simple

Ce mode de communication est très basique il sera utilisé pour faire communiquer simplement deux modules qui sont seuls dans la pièce.

Dans ce mode:

1. On n'utilise pas les adresses (on reste tout le temps en broadcast).
2. Il n'y aura pas de contrôle pour vérifier si les informations sont bien arrivées.
3. Il n'y aura pas de système de connexion (toujours considérer comme connecté).

On ne seras donc pas obligé d' utiliser le modèle de communication a deux couches évoquer dans la prés-étude(P.6) .



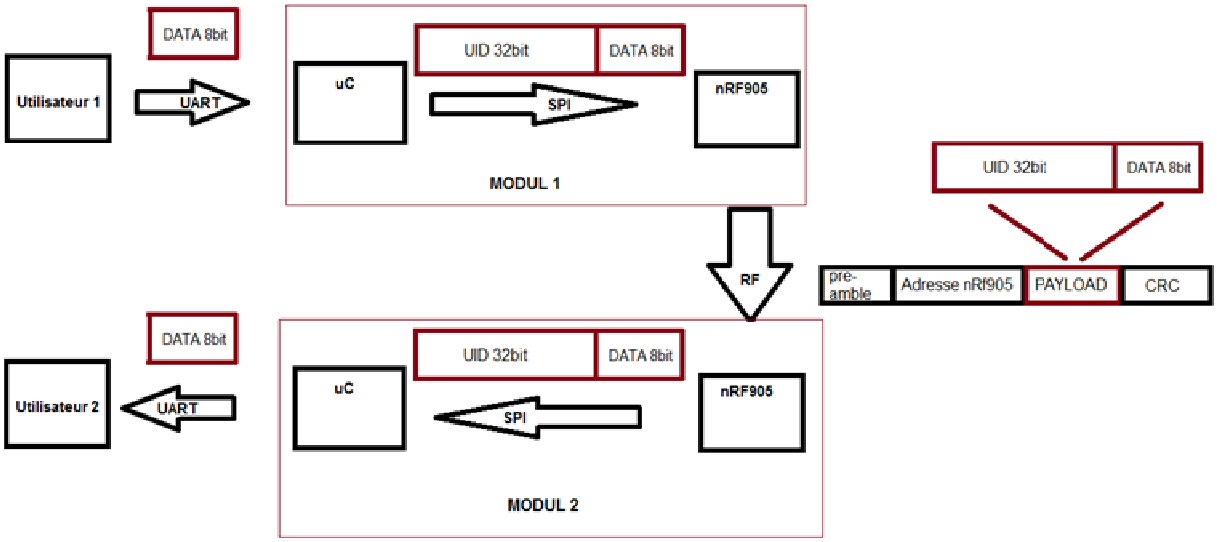
2.3.4Mode de communication standard

Ce mode de communication sera le mode par défaut. Il permettra à plusieurs modules de communiquer en parallèle (deux par deux) tout en pouvant faire du browdcast.

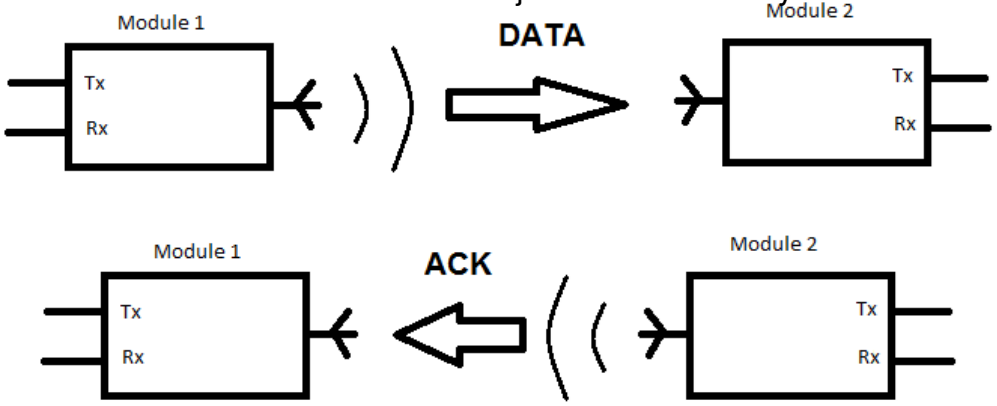
Dans ce mode:

1. On utilise les adresses 32 bit des IC UID.
2. Il y aura un contrôle pour vérifier si les informations sont bien arrivées.
3. Il y aura un système de connexion (pairage) simple.

On ne sera obligé d'utiliser le modèle de communication à deux couches évoqué dans la prés-étude(P.6).



Contrôle:
Pour contrôler si les data sont bien arrivé je vais utiliser un système d'acknowledge.

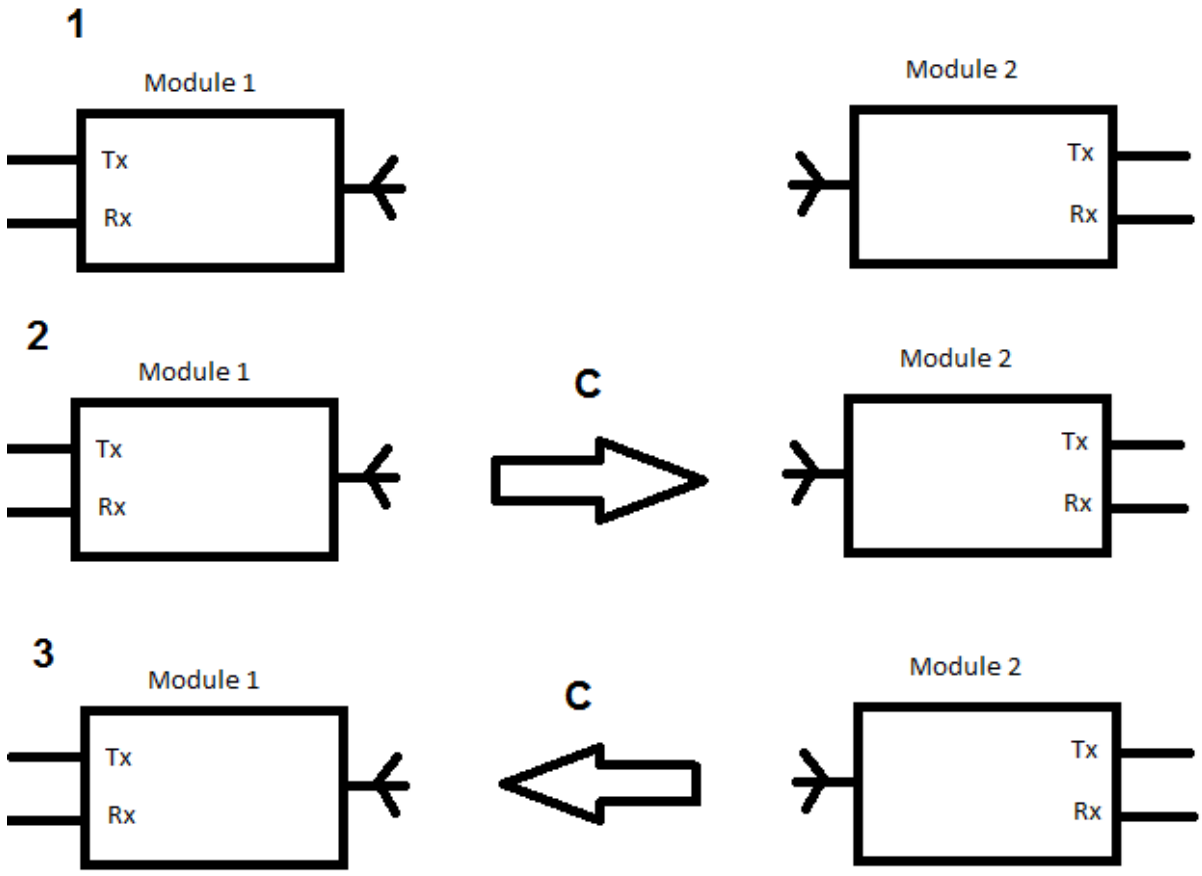


Connexion:
Pour que deux modules soient connectés:

le module 1 va commencer par écouter pendent 3s (ou un temps aléatoire pour éviter les collisions entre les deux modules si ils veulent se connecter en même temps).

si il ne reçoit rien pendent ses 3s il va essayer de communique avec le module 2.

Quand le module 2 reçois la demande de connexion ('c') il se considère comme connecté et renvoie un 'c' au module 1 pour que lui aussi se considère comme connecté.



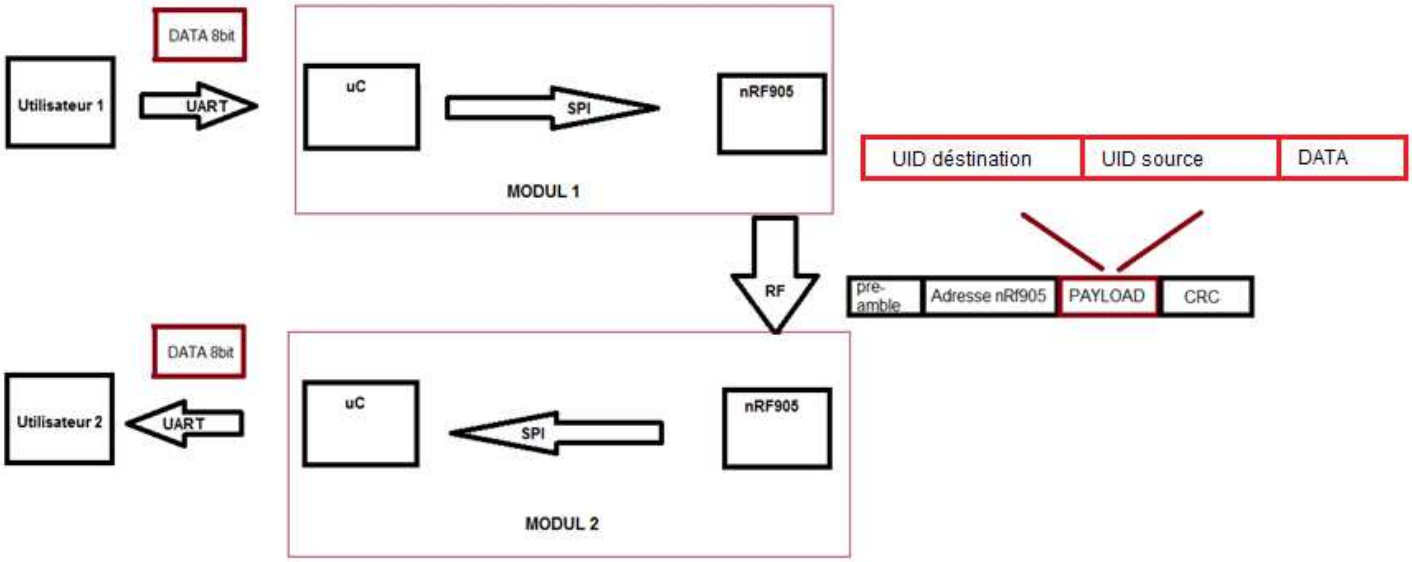
2.3.5Mode de communication avancé.

Ce mode de communication sera le mode le plus compliqué. Il permettra à un module de communiquer avec n'importe quel module sans avoir besoin de se connecter tout en pouvant faire du broadcast.

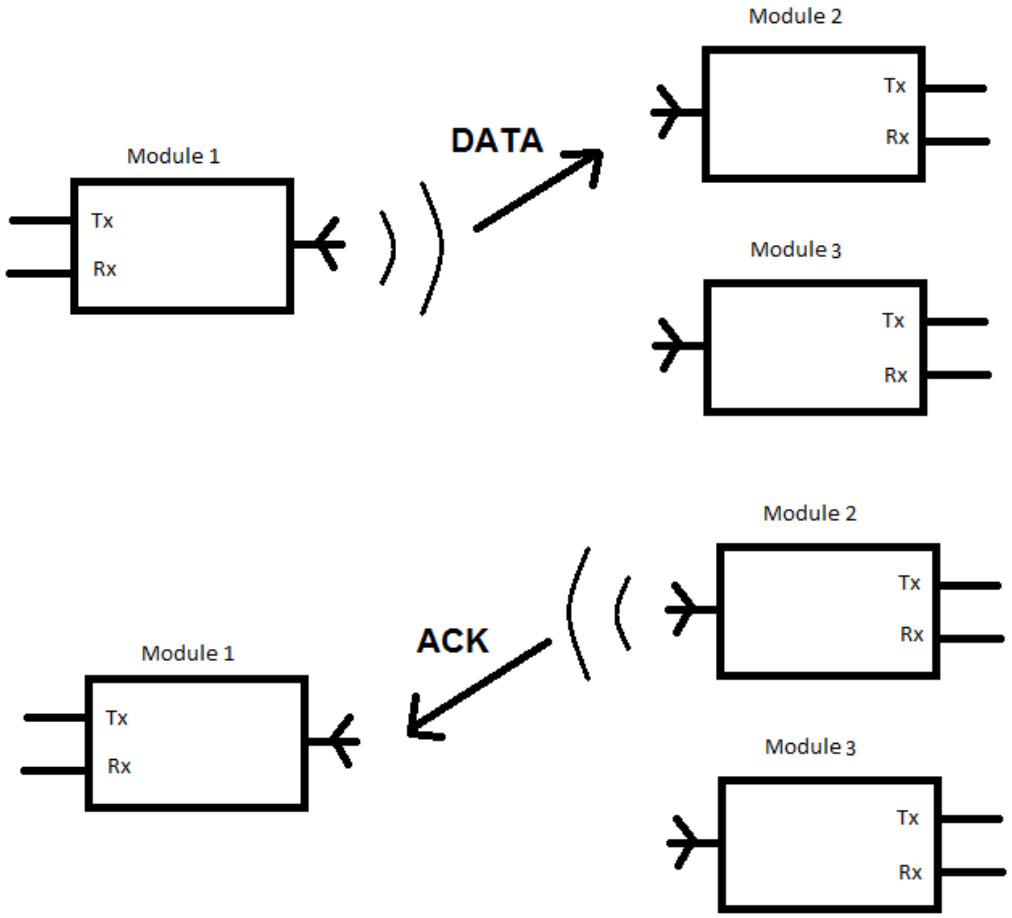
Les modules pourront savoir qui leur parle en allant lire un ID de source.

Dans ce mode:

1. On utilise les adresses 32 bit des IC UID.
2. Il aura un contrôle pour vérifier si les informations sont bien arrivées.
3. Il n'y aura pas de système de connexion (fonctionnera plus comme un réseau).



Contrôle:
Pour le contrôle il faudra renvoyer un acknowledge à l' ID source.



Détaille Tx

Pour envoyer une trame il faut commencer par écouter si le canal (880MHz) est occupé grâce à la pine CD du nRF980.

Si le canal est libre on peut configurer les bits de contrôle pour envoyer la trame.

PWR_UP	TRX_CE	TX_EN	Operating Mode
0	X	X	Power down and SPI programming
1	0	X	Standby and SPI programming
1	X	0	Read data from RX register
1	1	0	Radio Enabled - ShockBurst™ RX
1	1	1	Radio Enabled - ShockBurst™ TX

Il faut ensuite immédiatement se remettre en mode Rx.

2.4 Détaille Rx

Il faut commencer par ce mettre en mode Rx avec les bits de commande

PWR_UP	TRX_CE	TX_EN	Operating Mode
0	X	X	Power down and SPI programming
1	0	X	Standby and SPI programming
1	X	0	Read data from RX register
1	1	0	Radio Enabled - ShockBurst™ RX
1	1	1	Radio Enabled - ShockBurst™ TX

Il faut ensuite attendre qui il y a un adresse match ensuite si il la patte DR passe à 1 c'est que les DATA sont valide sinon il faudra indiquer au module qu'il a une erreur de transmission.

3 Routage

3.1 Contrainte de routage

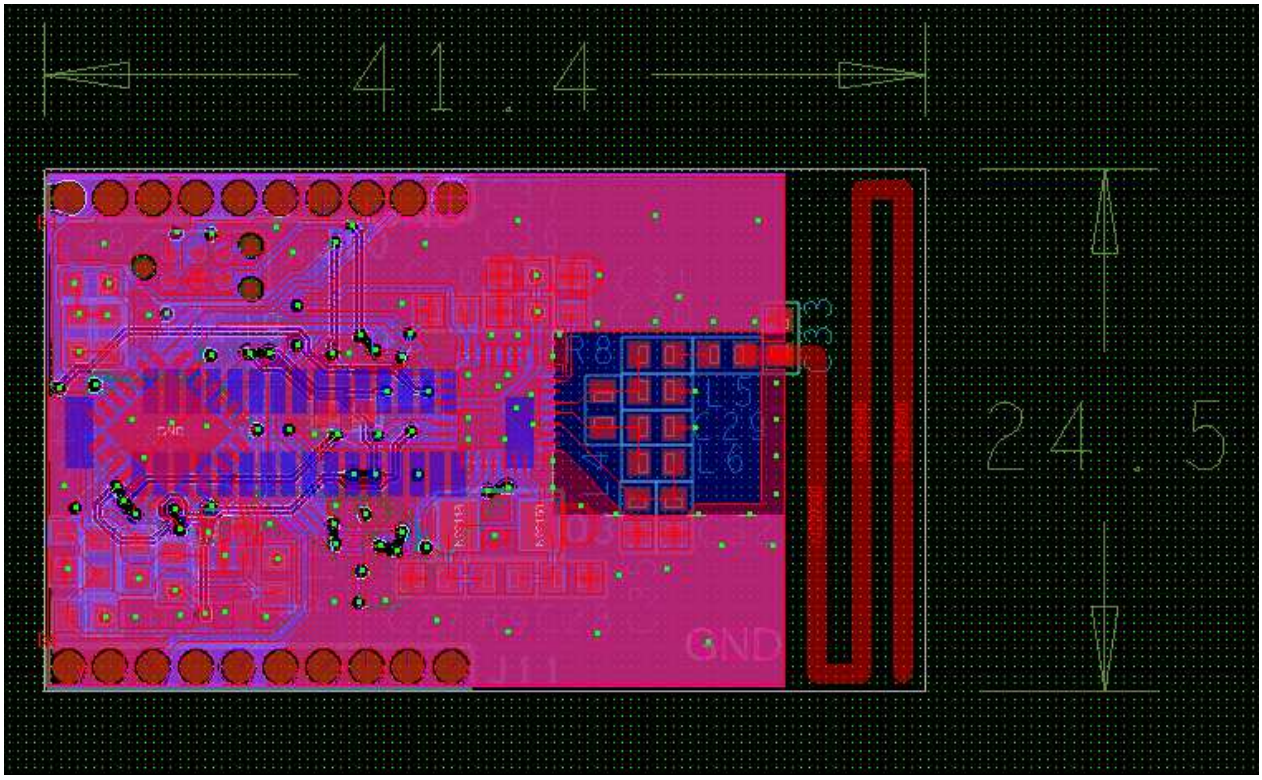
3.1.1 Nombre de couche:

Le routage est réalisé sur 4 couches pour être le plus compact possible car le module RF sera utilisé dans d'autres projets avec des boîtiers compacts.

3.1.2 Taille de la board

La board devait idéalement être de la même taille qu'un XBee soit 24.4 sur 27.6 mm mais le module radio et le connecteur mezzanine empêche de pouvoir mettre des composants sur la face BOTTOM.

Donc après discussion avec mes collègues nous avons défini que la taille de 24.5 sur 43 mm était acceptable pour eux.



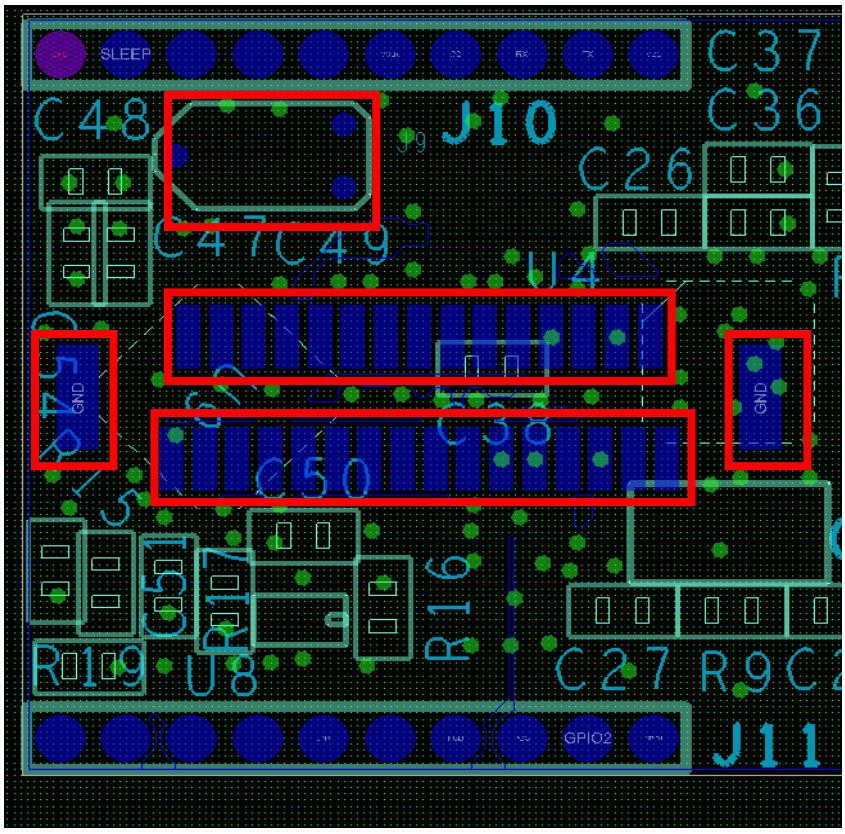
3.2 Particularité du routage

3.2.1 Partie microcontrôleur et UID

Pour cette partie du routage il n' y a pas de grosses particularités les seules contraintes sont l'écartement des connecteurs et le routage qui doit être le plus compact possible.

La principale difficulté est que l'on ne peut pas faire de via au-dessus du connecteur mezzanine et du Tag Connect (zone entourée en rouge sur image).

De plus il y a un nombre important de pistes car les signaux TX, RX, IO1, IO2, SLEEP, CONNECT et LINK sont doublés car ils doivent aller dans le connecteur barrette et mezzanine, les signaux PGC, PGD et MCLR sont triplés car ils doivent en plus aller au Tag Connect.



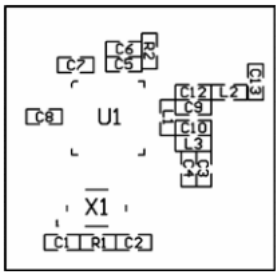
Remarque:
Pour pouvoir gagner de la place j'ai dès que possible fait passer les vias des signaux correspondant directement dans les pads du connecteur mezzanine.

J aussi décider de mettre le PIC à 45 degré pour faciliter le routage.

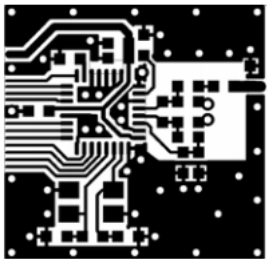
3.2.2Partie RF

Le routage de la partie RF et de l'antenne est une opération très compliquée pour une personne qui n'a jamais touché à de la RF car il y a de la haute fréquence (868MHz).

Il faut donc impérativement respecter les footprints, le placement, le routage et les plans de masse conseillés par le fabricant dans le datatheet et savoir improviser pour faire l'antenne.

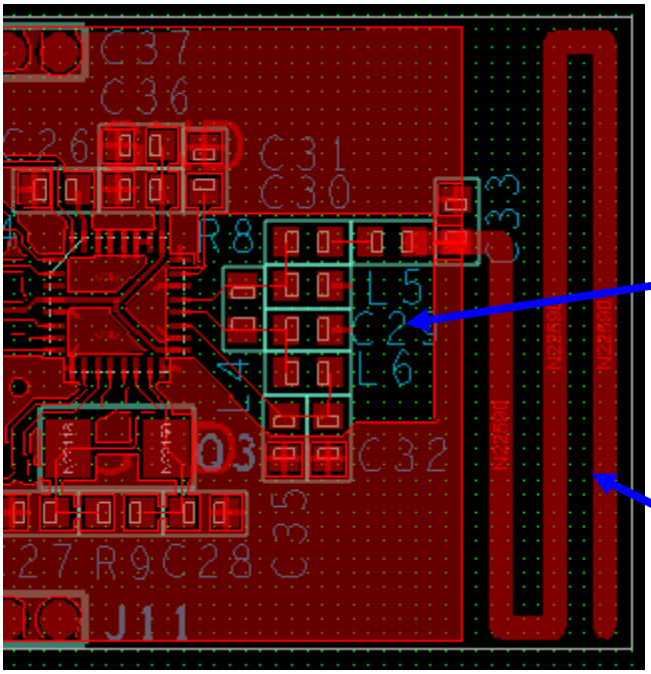


Top silk screen



Top view

Routage de la board:



Zone RF très sensible elle ne doit pas se trouver proche d'un plan de masse.

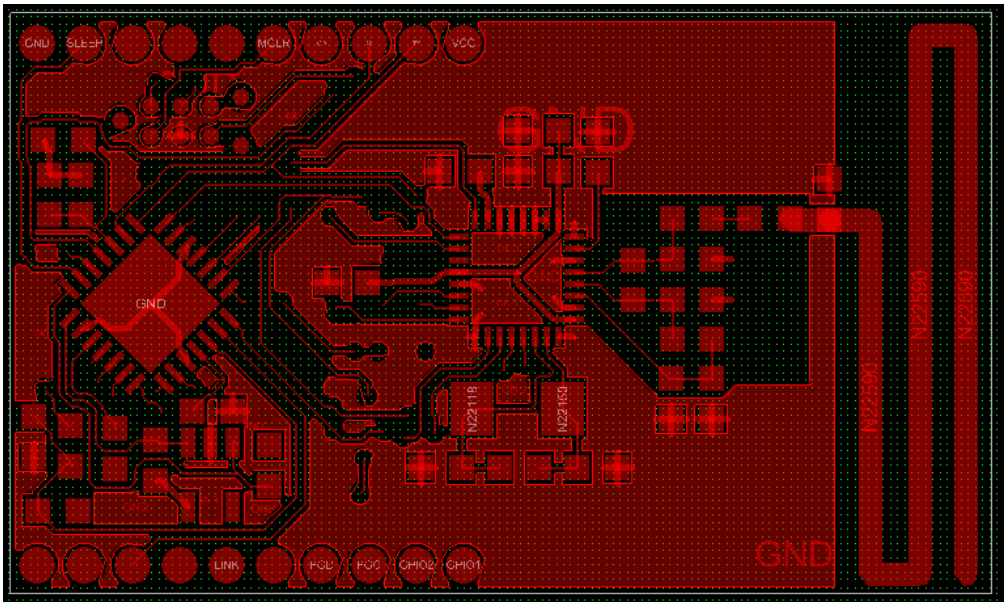
Antenne designer par M. Yersin.

Remarque:
Pour gagner du temps et éviter les problèmes cette partie du routage a été réalisée par M. Yersin.

3.3 Description des couches

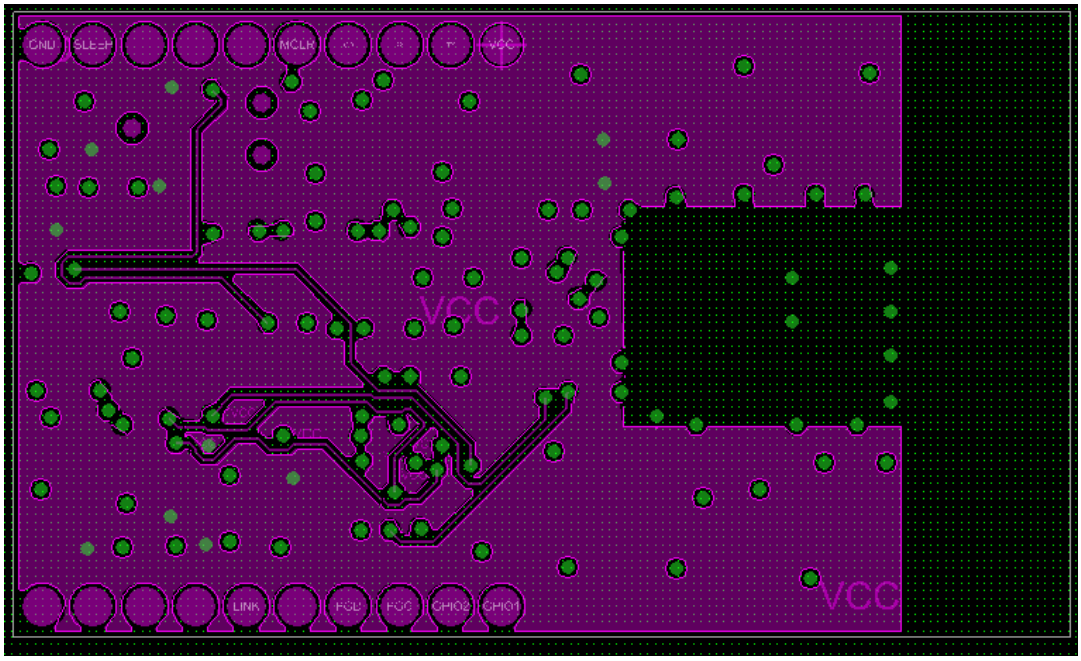
3.3.1TOP

C'est la couche où se trouve les composants. C'est sur cette couche que j'ai essayé de faire passer la plupart des pistes.



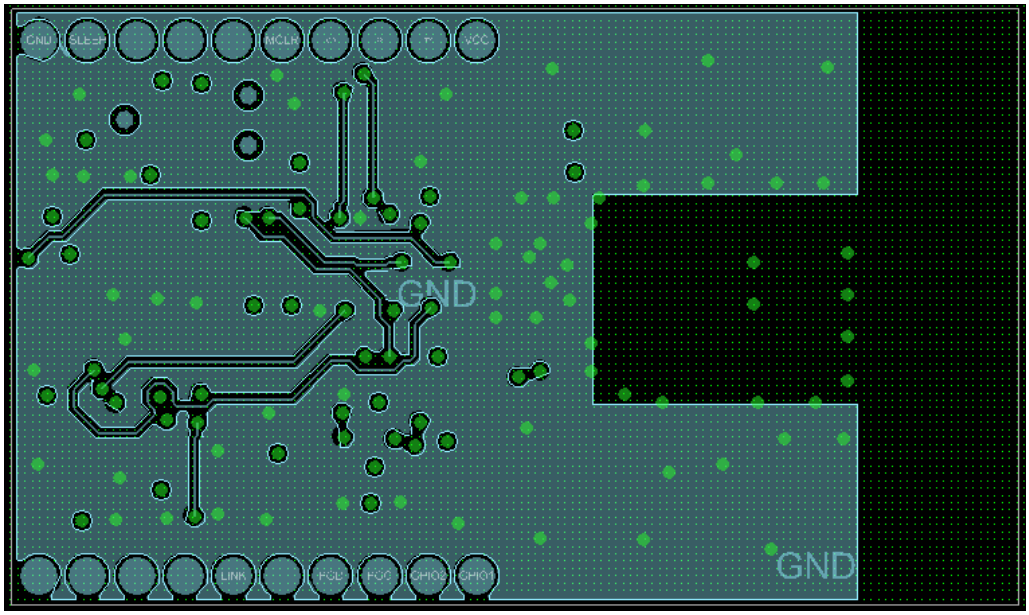
3.3.2La couche VCC

C'est le plan de Vcc pour éviter les perturbations j'ai essayé de limiter les pistes sur cette couche.



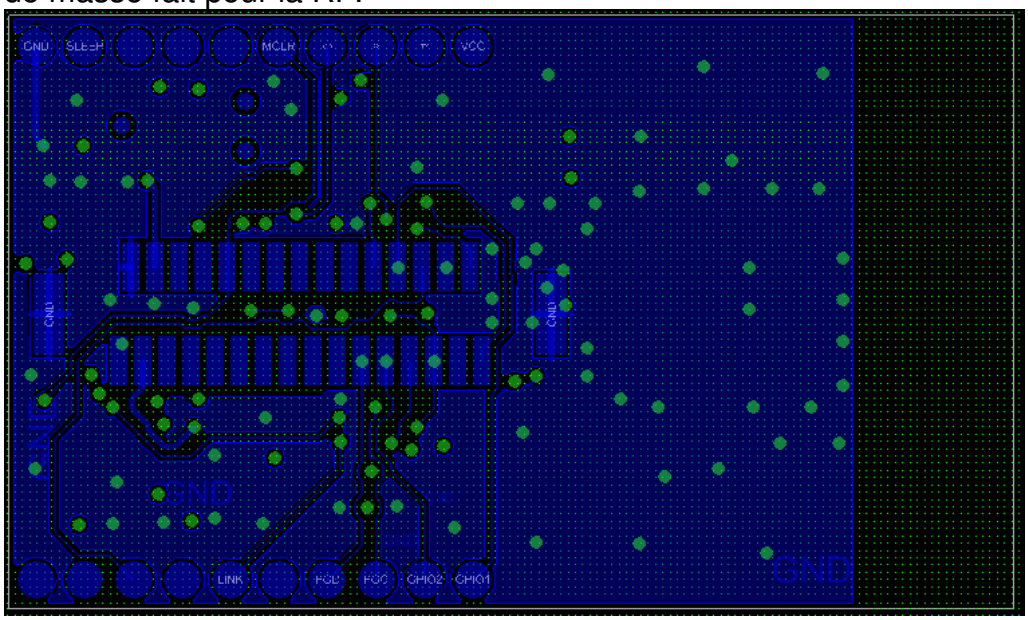
3.3.3La couche GND

C'est le plan de GND pour éviter les perturbations j'ai essayé de faire passer le moins possible de pistes sur cette couche et aussi de réduire au maximum les zones sans plan de masse (à part sous la zone RF).



3.3.4BOT

C'est la couche de routage secondaire c'est ici que j'essaye de faire passer les pistes après le TOP, on peut aussi remarquer que l'on choisit de reboucher le trou dans le plan de masse fait pour la RF.

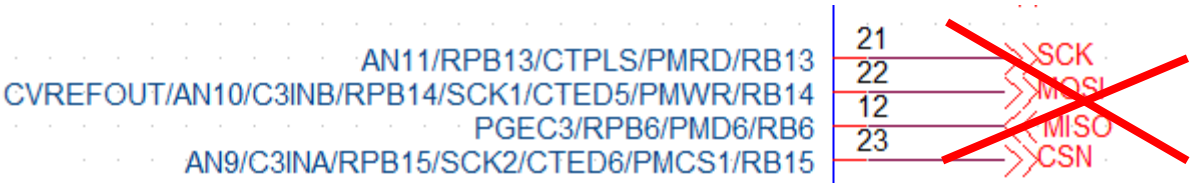


3.4 Correction des erreurs de design et de routage

Suite à un changement de boîtier (SSOP ->QFN) je n'ai pas tout de suite trouvé comment changer de boîtier sur Harmony.

J'ai donc dû placer mes signaux en utilisant le datasheet et malheureusement j'ai commis des erreurs en reportant les signaux du SPI sur le PIC.

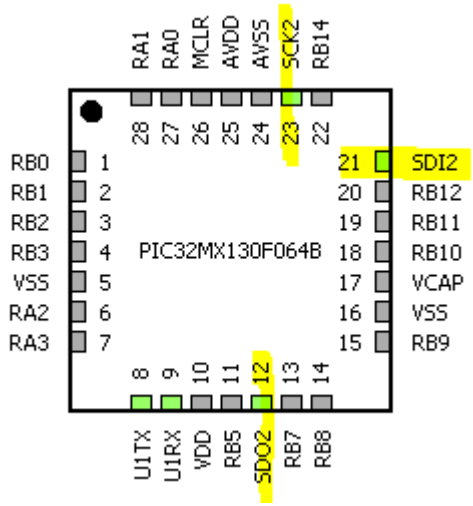
Je m'en suis rapidement rendu compte car mon pic fonctionnait correctement mais il se mettait à consommer 10 fois plus quand je voulais utiliser le SPI (30mA au lieu de 4mA).



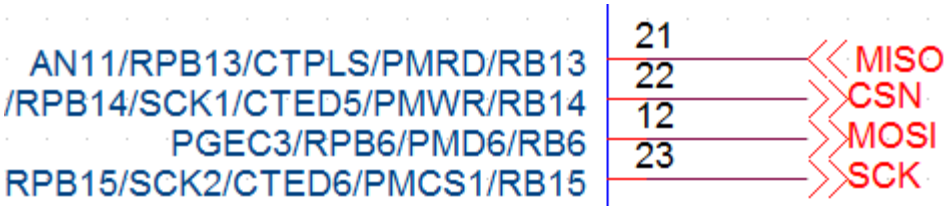
3.4.1 Correction du design :

Après avoir trouvé comment changer de boîtier sur Harmony et refait un BSP j'ai pû corriger les signaux SPI sur le schéma.

Nouveau BSP pour le boîtier (QFN) :

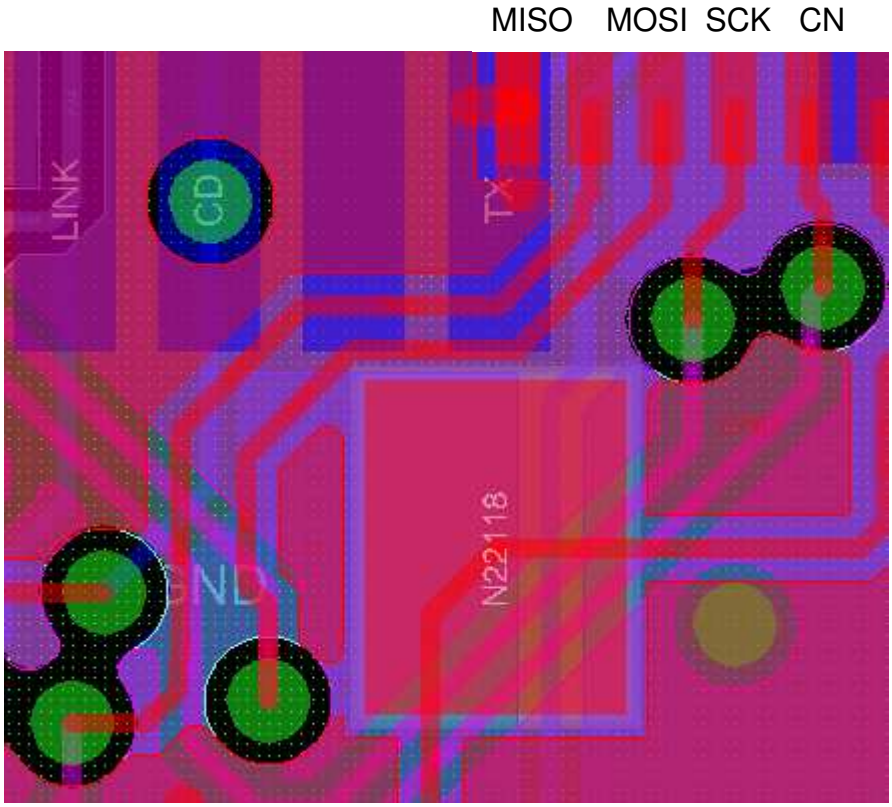


Implémentation du SPI corrigée sur le PIC :



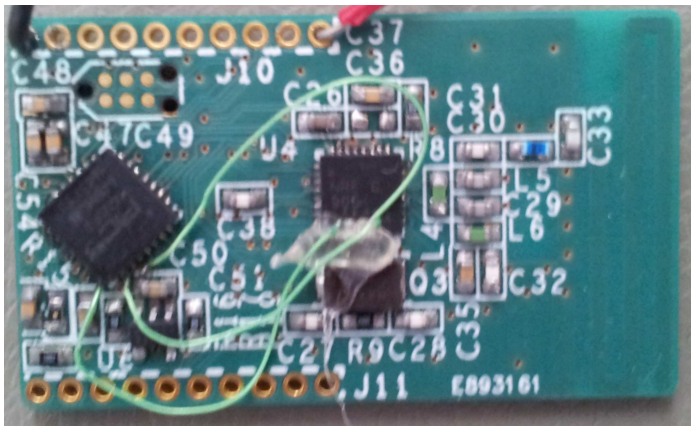
3.4.2Correction du routage

Comme j'avais économisé la couche VCC et GND, la correction du routage n'a pas été très compliquée il fallait juste remettre les 4 signaux du SPI au bon endroit.



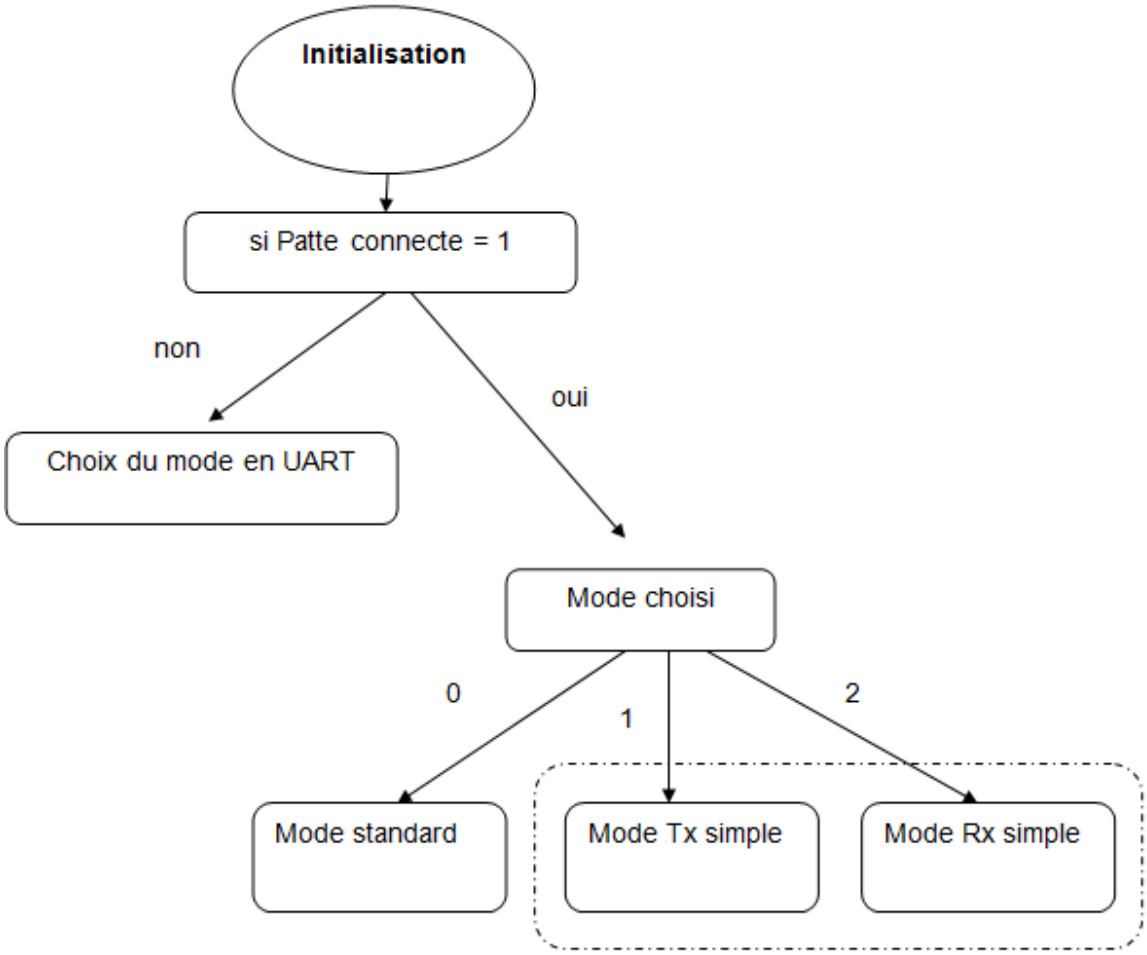
3.4.3Correction du PCB

J'ai essayé de corriger les erreurs sur le PCB avec des fils mais comme on utilise des boîtiers QFN très petits chaque correction prend un temps considérable et risque d'engendrer plus d'erreur qu'elle n'en corrige donc M.Yersin a pris la décision de recommander des PCB corrigés.



4 Description du logiciel

4.1 Vue générale du programme

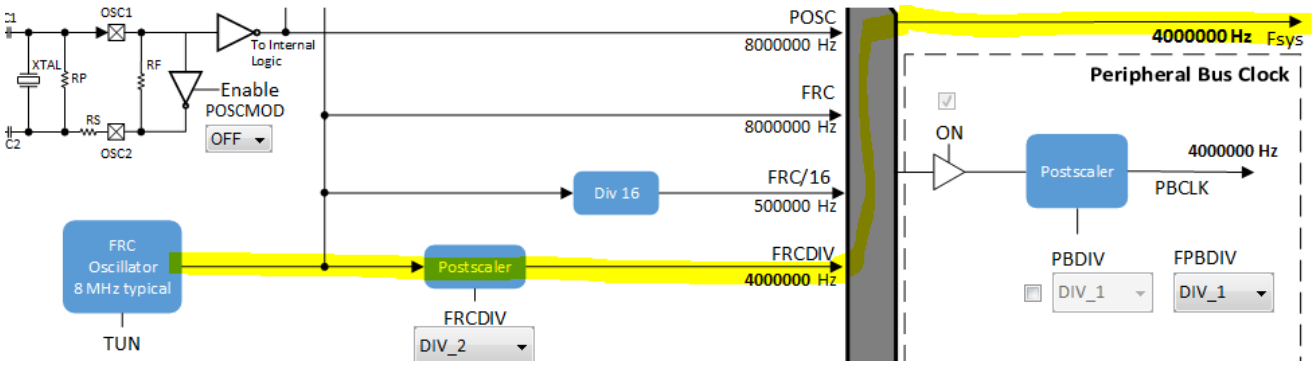


Remarque:
Explication mode Tx /Rx simple p.16.
Explication mode standard P.17.

4.2 Configuration des drivers Harmony

4.2.1 Horloge

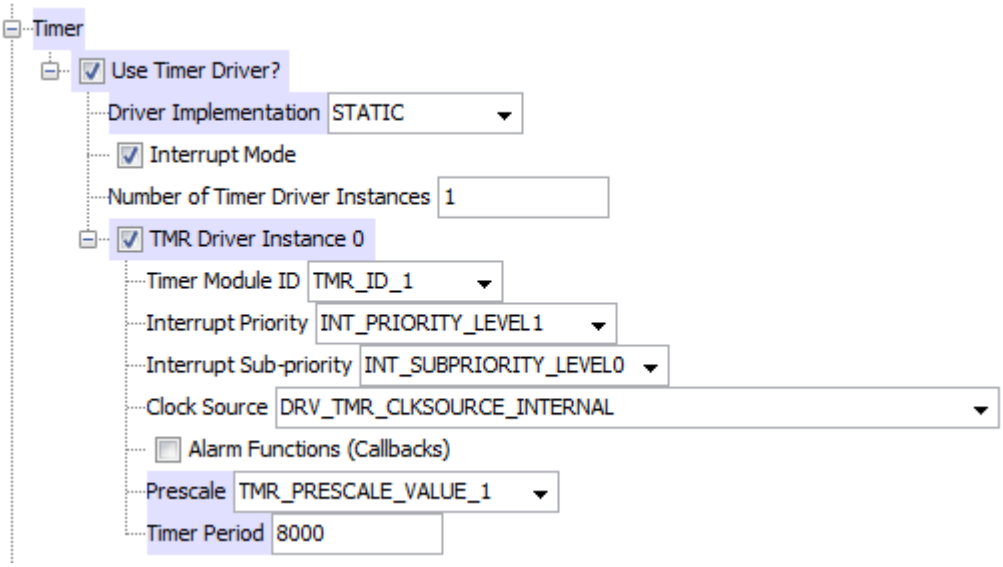
L'horloge est réglée en mode oscillateur interne 4MHz.



Remarque

on pourrais utiliser la PLL pour avoir un clock plus rapide mais comme le module doit consommer le moins possible pour pouvoir être installé sur des systèmes embarqué.

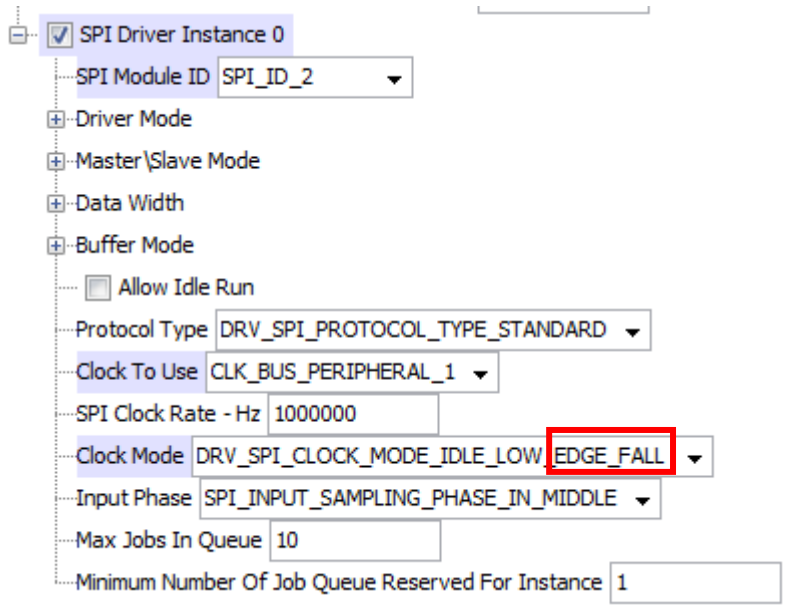
Timer 1



Remarque :

Timer réglé pour faire une interruption toutes les 2 ms

4.2.2SPI 1



Remarque :
SPI réglé à 1MHz avec validation sur flanc montant (explication EDGE_FALL plus loin)

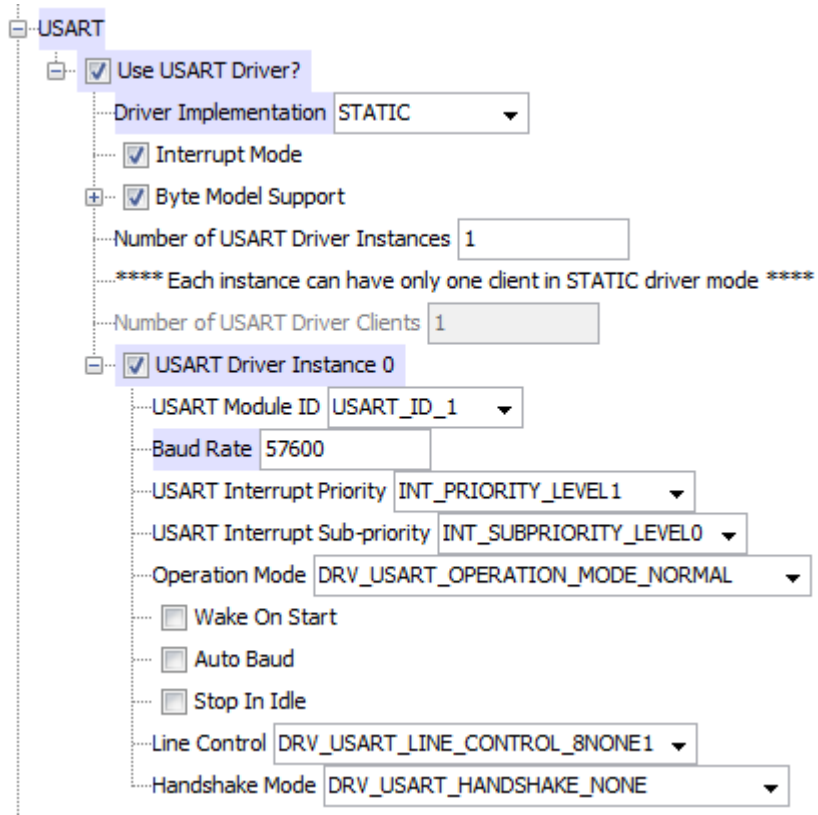
4.2.3I2C 2

J'ai décidé de ne pas configurer I2c avec Harmony mais avec les fonctions de la PLIB en m'inspirant d'une configuration déjà faite pour un TP de MINF

```
44  void i2c_init( bool Fast )
45  {
46      PLIB_I2C_Disable(I2C_24AA02);      // Ajout CHR
47
48      PLIB_I2C_HighFrequencyEnable(I2C_24AA02);   // OK comme cela
49      if (Fast)  {
50         PLIB_I2C_BaudRateSet(I2C_24AA02,
51                 SYS_CLK_PeripheralFrequencyGet(CLK_BUS_PERIPHERAL_1), I2C_CLOCK_FAST);
52      } else {
53          PLIB_I2C_BaudRateSet(I2C_24AA02,
54                 SYS_CLK_PeripheralFrequencyGet(CLK_BUS_PERIPHERAL_1), I2C_CLOCK_SLOW);
55      }
56      // PLIB_I2C_HighFrequencyDisable(I2C_24AA02);   // celon driver
57
58      PLIB_I2C_SlaveClockStretchingEnable(I2C_24AA02);  // ajout CHR
59
60      PLIB_I2C_Enable(I2C_24AA02);
61
62      I2cConReg = I2C2CON;
63      I2cBrg = I2C2BRG;
64  }
```

Remarque :
I2C réglé en mode fast.
J'utilise I2C 2 car le 1 est déjà utilisé par le SPI (en fait ce sont les mêmes modules)

4.2.4UART 1



Remarque :

- Baud rate 57600 b/s
- Sans contrôle de flux
- Avec interruption

4.3 Gestion de l'UART

4.3.1 Principe UART

Pour utiliser l' UART J'utilise le même principe vu dans les TP mais sans CRC :

Pour l'émission on va remplir un FIFO logiciel qui va être progressivement envoyé via une interruption.
Pour la réception une interruption va remplir un FIFO logiciel que l'on pourra ensuite lire quand on le souhaite.

On va ensuite gérer cet FIFO avec les fonctions GetMessage et SendMessage.

4.3.2 Structure de la trame

```
typedef struct {
   uint8_t Start;
   uint8_t com;
   U_manip32 add;
   int8_t  data;

} StruMess;
```

Dans la trame UART on trouve:

1byte de start 0xAA.
1byte de commande si on veut configurer le module.
4bytes d'adresse pour indiquer l'adresse du module avec lequel on veut parler.
1byte de data.

Tous les utilisateurs du module RF devront utiliser cette structure pour pouvoir communiquer.

4.4 Gestion de L' I2C et lecture de l'UID

4.4.1 Principe I2C

Pour aller lire l'UID qui se trouve dans l' 24AA02UID on va simplement utiliser les fonctions qui se trouvent dans Mc32_I2cUtilCCS et reprendre la fonction utilisée pour aller lire EEPROM du KIT.

Il faut juste trouver les bonnes adresses dans le datatheet.

Adresse du composant :

Operation	Control Code	Chip Select	R/$\overline{W}$
Read	1010	Chip Address	1
Write	1010	Chip Address	0

Comme le chip select vaut dans notre cas toujours 0 cela donne :

Read = 0xA1
Write = 0xA0

Adresse du registre UID :

FIGURE 9-2: SERIAL NUMBER PHYSICAL MEMORY MAP EXAMPLE

Description	Manufacturer Code	Device Code	32-bit Serial Number			
Data	29h	41h	12h	34h	56h	78h
Type	Fixed		Serialized			
Array Address	FAh	FBh	FCh	FDh	FEh	FFh

MSB = FC
Byte2 = FD
Byte 1= FE
LSB = FF
On peut commencer à lire à FC et ensuite continuer et les adresses s'incrémentent automatiquement.

4.5 Utilisation du SPI pour communiquer avec le NRF905

4.5.1 Principe de communication avec le NRF905

Dès que le CS passe à 0 le NRF905 s'attend à recevoir une commande (1byte).

Il y a deux types de commandes une qui permet d'écrire dans un registre et une qui permet de le lire.

On peut ensuite aller lire ou écrire d'une seule traite dans le registre voulu.

4.5.2 Exemple d'une fonction de configuration pour le NRF905

```
void SPI_InitNRF905(void)
{


   SPI_Configure();

   // action de configuration
   CS = 0;
   //aller dans le registre de configuration
   spi_write2(0x00);
   //écrire config

   //byte 0
   spi_write2(0x75); //ch_NO[7:0] = '0111 0101' soit 868.2MHz

   //byte 1 = '0000 1110'
   spi_write2(0x2E);//AUTO_RETRAN ='0',RX_RED_PWR = '0',Output power ='11',PLL ='1',ch_NO[8] = '0'

   //byte 2 = 0001 0001'
   spi_write2(0x11);// RX_AFW ='1', TX_AFW ='1' soit un byte d'addresse

   //byte 3 et 4 payload width Rx et Tx = '0000 0101'
   spi_write2(0x05); //32 bits d' addresse + 8bits de data soit 5bytes
   spi_write2(0x05); //32 bits d' addresse + 8bits de data soit 5bytes

   //byte 5 à 8 Rx add
   spi_write2(0xE7);
   spi_write2(0xE7);
   spi_write2(0xE7);
   spi_write2(0xE7);

   //byte 9 = '1101 1111'
   //CRC_MODE = '1',CRC check enable ='1',Crystal 16Mhz ='011' Output clock enable = '1',Output clock frequency
   spi_write2(0xDE);
    CS = 1;
} // SPI_Init
```

Remarque :
C'est la configuration que j'utilise dans le projet

4.6 Machine d'état qui gère la communication RF

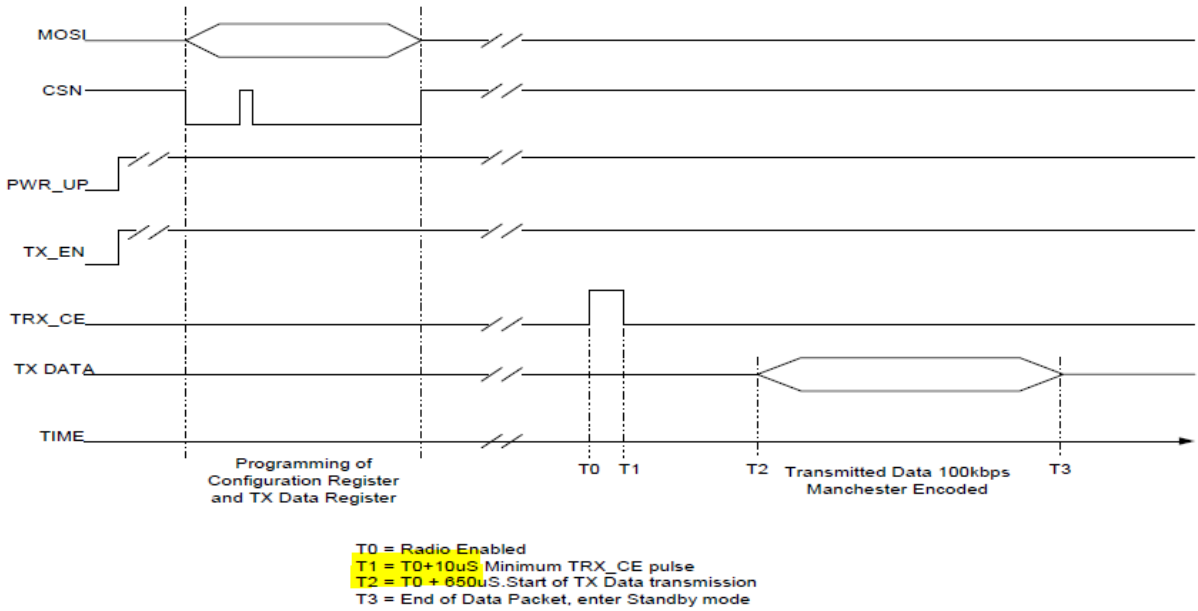
C'est une Machine d'état qui va nous permettre de savoir si les modules sont connectés et qui va gérer la communication bidirectionnelle avec acknowledge.

Les deux fonctions qui seront principalement utilisées dans la machine d'état sont sendRF et receiverRF.

4.6.1 sendRF

Cette fonction sert à envoyer la trame RF au module et le mettre en mode Tx(voir page 21) pour qu'il commence la transmission. Une fois la transmission terminée on désactive le mode Tx.

Delay important pour le mode Tx :



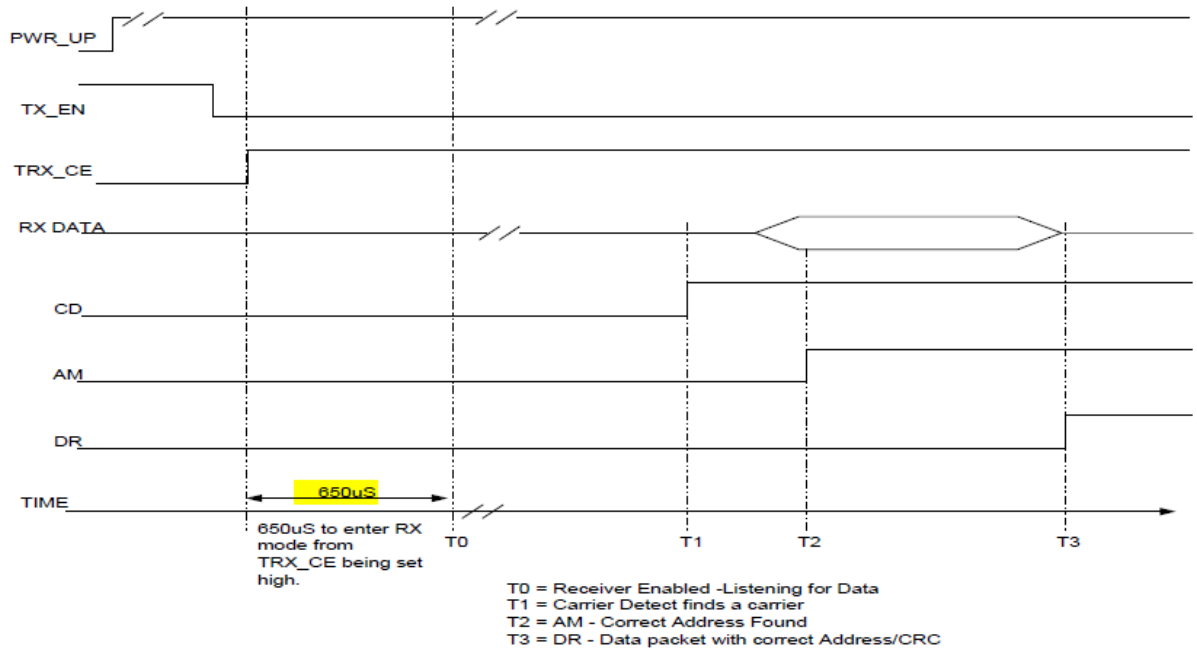
Remarque :
Pour savoir quand la transmission est terminée il faut attendre que la patte DR passe à 1.

4.6.2receiverRF

Cette fonction sert à mettre le NRF905 en mode Rx (voir page 21) puis à écouter la patte DR pour savoir s'il y a eu une communication.

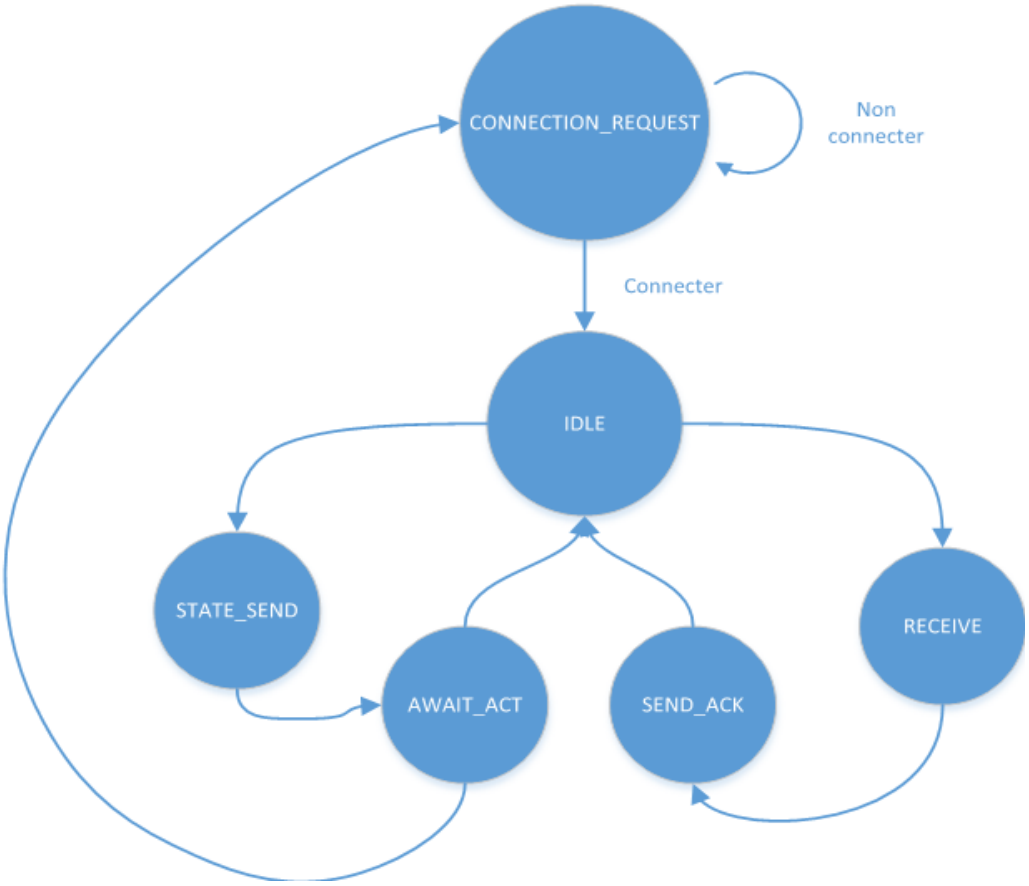
Une fois la patte DR à 1 on va lire le registre RxPalyload et si l'adresse correspond on valide la trame et on la passe plus loin.

Delay important pour le mode Rx :



Remarque :
Il faux 650 us pour passer en RX si on est en Tx.

4.6.3Machine d'état



4.6.3.1 CONNECTION_REQUEST

C'est l'état qui va permettre au module de se connecter entre eux en utilisant le principe expliqué à la page 18.

4.6.3.2 IDLE

Etat où on attend d'envoyer ou recevoir une trame. Si trop de temps s'écoule dans cet état on refait une demande de connexion.

4.6.3.3 SEND et RECEIVE

État où l'on reçoit ou envoie une trame en utilisant les fonctions vues plus haut.

4.6.3.4 SEND_ACK

Envoi d'un acknowledge.

4.6.3.5 AWAIT_ACK

On attend un acknowledge si on attend trop longtemps on se déconnecte et on retourne dans l'état CONNECTION_REQUEST pour essayer de se reconnecter.

4.6.4Description de App.c

C'est dans App.c que sont gérés les commandes qui vont déterminer le mode de communication.

Si la patte connecte est à 0 on est en mode configuration c'est dans ce mode que l' on choisit le type de communication que l' on veut ou que l' on demande l' UID .

Si la patte connecte est à 1 on Start la communication dans le mode choisit :

```
if(CONNECT)
{
    switch (mode)
    {
        case 0:
        {
            RF_DoTasks();//bidirectionelle
            break;
        }
        case 1:
        {
            if(GetMessage(&RX_UART)) // mode TX simple
            {
              sendRF(RX_UART.add,RX_UART.data);
            }
            LINK =1;
            break;
        }
        case 2:
        {
            if(receiverRF(&TX_UART.data))//mode RX simple
            {
                SendMessage(&TX_UART);
            }
            LINK =1;
            break;
        }
```

Remarque :
Les modes 1 et 2 (Tx Rx simple) ne gère pas la connection et n' utilise pas d'acknowledge c' est le mode décrit en page 16.

Le mode 0 est le mode de communication standard (page 17) géré par la machine d'état (page 37).

4.7 Problèmes rencontrés

4.7.1 Configuration du SPI

Quand on règle le SPI en mode :

On s'attend à avoir les datas valides sur le flanc montant du clock mais en réalité ils sont sur le flanc descendant.

Pour les avoir sur le flanc descendant il faut mettre le configurateur en mode EDGE_FALL voir page 30.

4.7.2 Buffer SPI

Le SPI fonctionne avec un Buffer de lecture et d'écriture comme le SPI est bidirectionnel à chaque fois que l'on écrit on va aussi remplir le buffer de réception.

Donc si on écrit 5 fois et qu'ensuite on veut lire les données, elles se trouvent dans la 6ème case du buffer.

Pour éviter ce problème il suffit de vider le buffer de réception à chaque écriture en allant le lire.

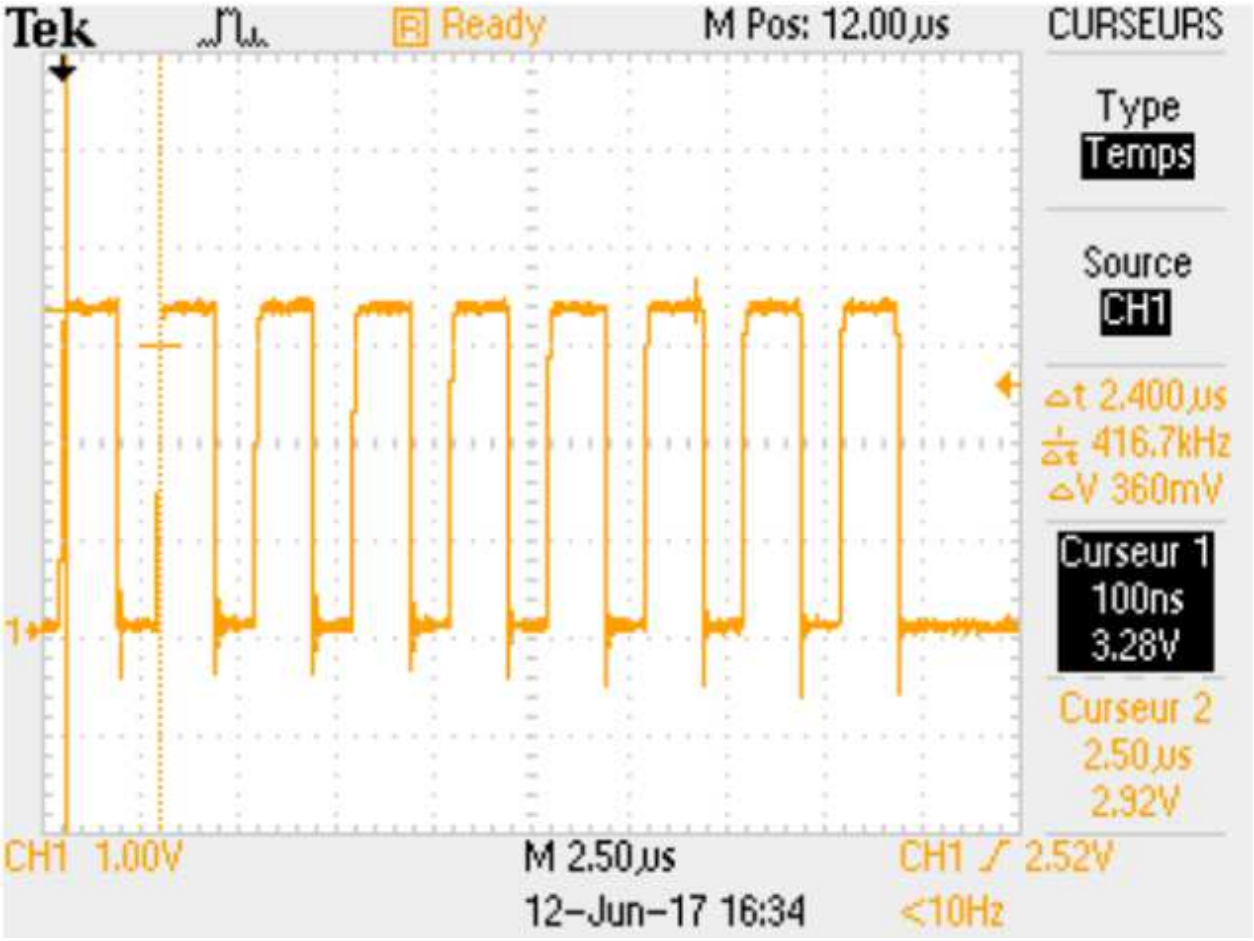
```
void spi_write2( uint8_t Val){
   int SpiBusy;

   PLIB_SPI_BufferWrite(SPI_ID_2, Val);
   do {
     SpiBusy =  PLIB_SPI_IsBusy(SPI_ID_2) ;
   } while (SpiBusy == 1);
  PLIB_SPI_BufferRead(SPI_ID_2);
}
```

5 Validation du design et du code

5.1.1 Test UART et lecture de UID en I2C

On va lire UID en I2C et envoyer l' information au KIT via UART

5.1.1.1 Mesure clock I2C

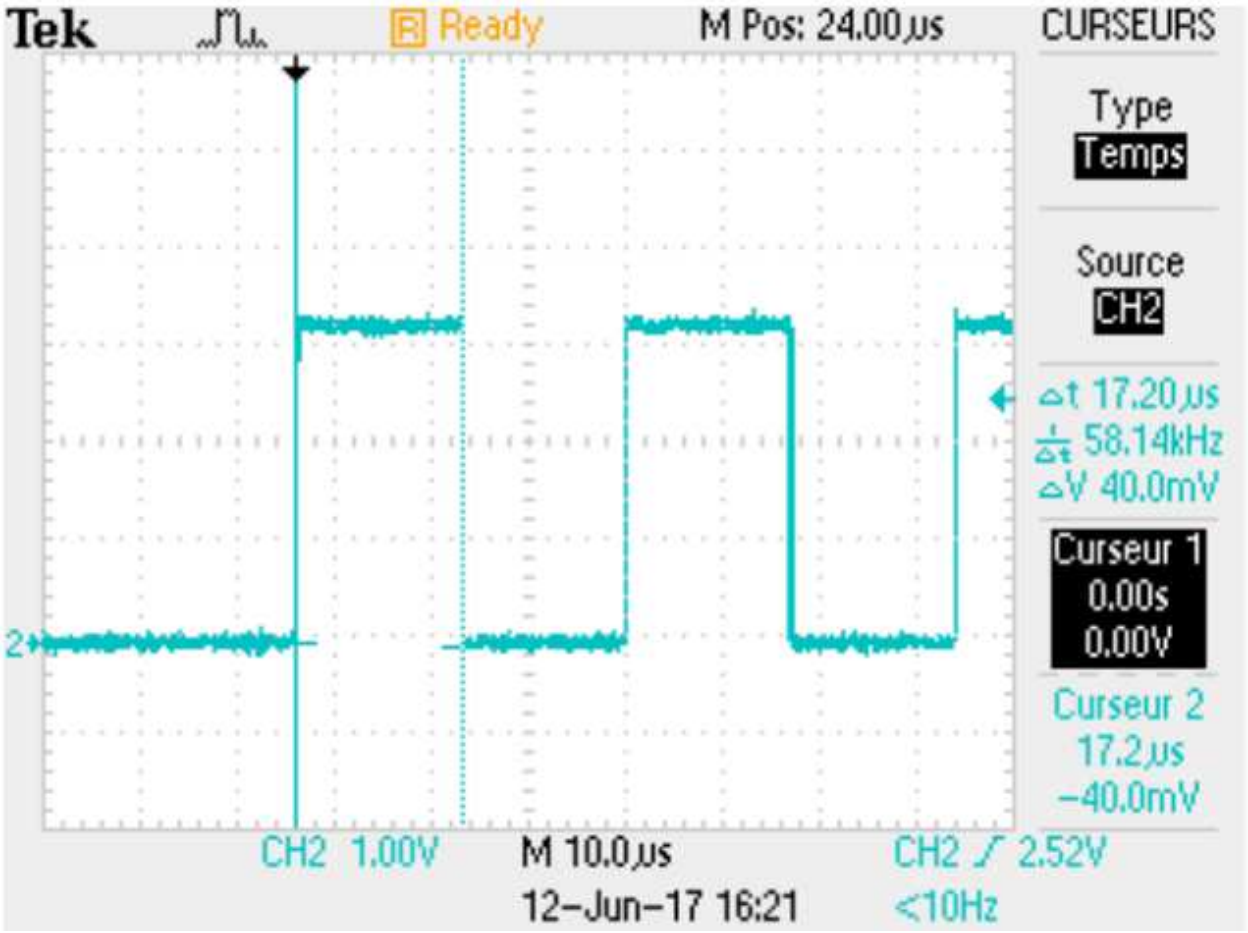


Remarque:
Le mode fast génère un clock à environ 400KHz on est donc bien à la fréquence maximum de l' IC.

Part Number	Vcc Range	Max. Clock Frequency
24AA02UID	1.7-5.5V	400 kHz(1)

Mais comme on est un peu en (416KHz) dessus il est plus sure de configurer le mode slow .

5.1.1.2 Mesure bauderate UART



Remarque :
On a bien un bauderaite à environ 57600 b/s (il y a sûrement une petite erreur de mesure)

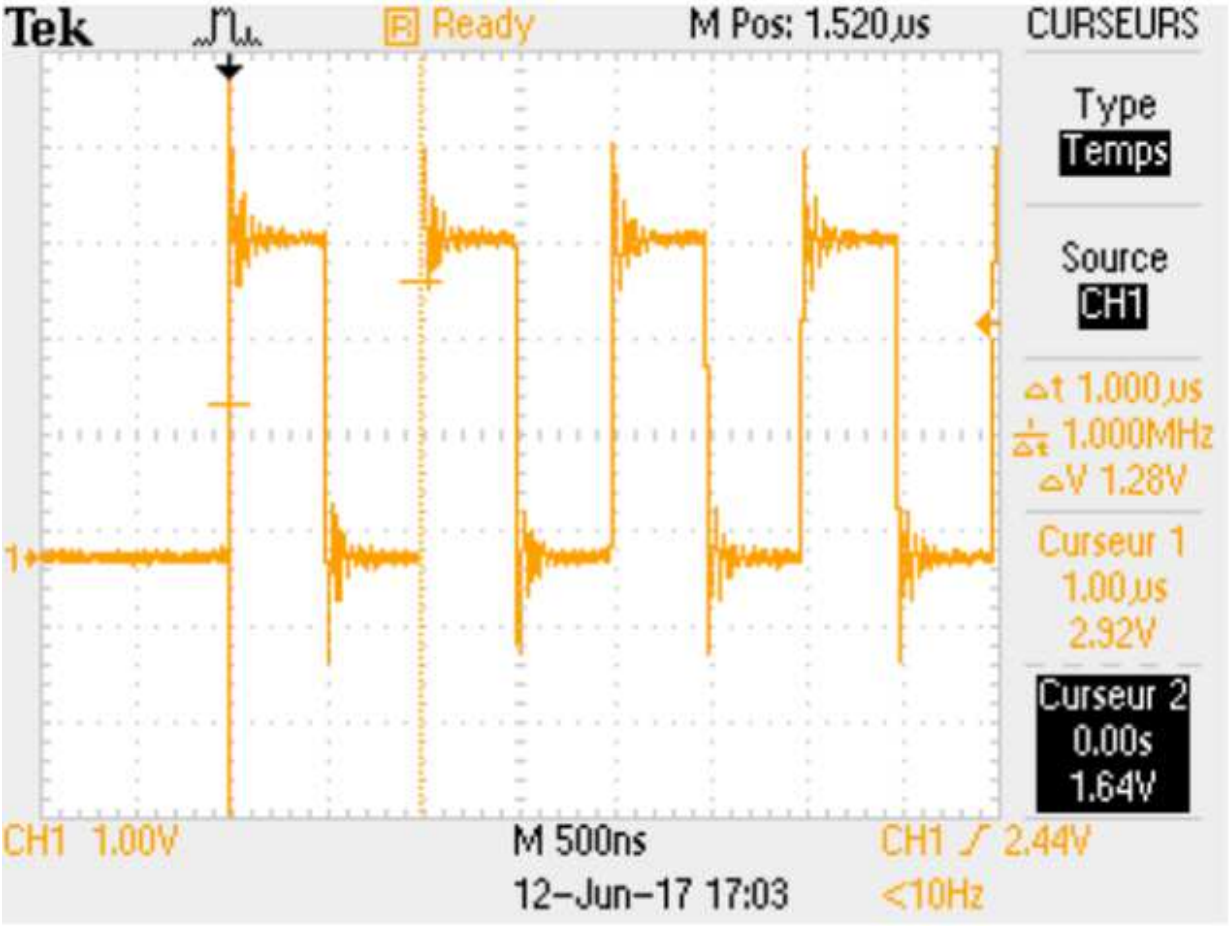
5.1.1.3 Photo du LCD du KIT avec UID affiché dessus

Remarque:
On peut croire qu' il manque le byte MSB mais en fait se byte est toujours à 0x00(vérifié en allant lire les 48bits de l' UID).

5.1.2Test du SPI et configuration NRF905

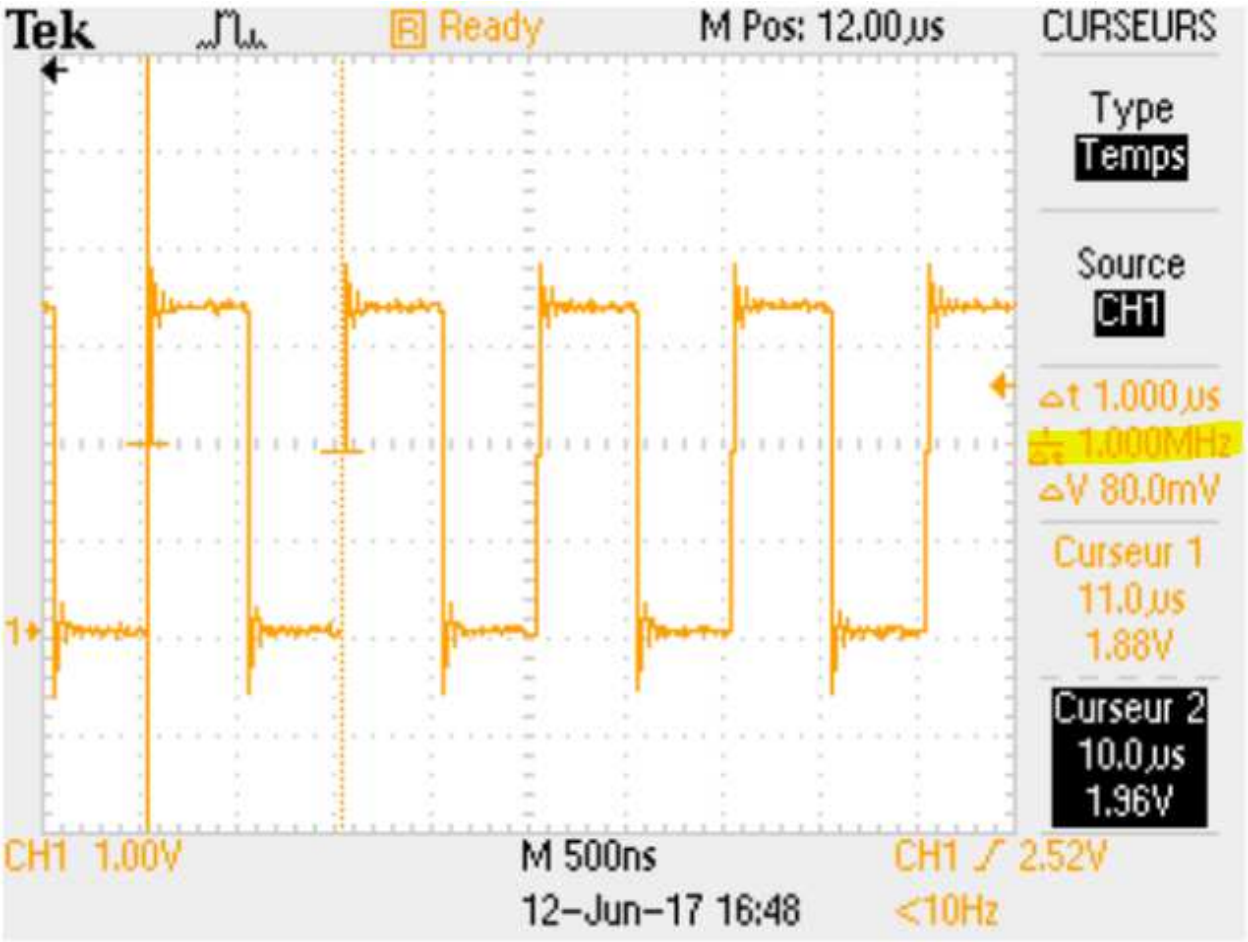
On configure le NRF905 en SPI et on mesure le clock externe du NRF905 qui doit être à 1 MHz si on a réussi à le configurer comme il faut.

Fréquence du SPI:



Remarque:
1MHz, C' est bien la fréquence réglée sur harmony

Mesure du clock externe



Remarque :
On a bien la fréquence voulue donc le NRF905 est bien configuré.

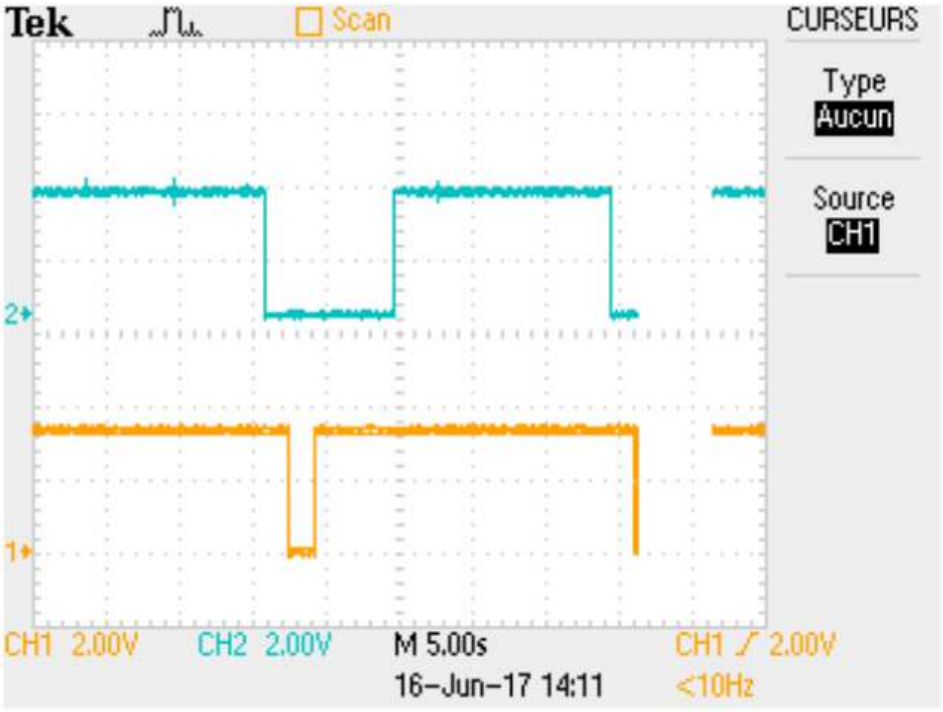
5.1.3Test de la communication RF en mode simple

On envoie une trame ('V') depuis un module et on regarde si on la reçoit sur l'autre module en affichant la data sur le LCD du KIT.

Remarque :
Le test a été effectué à une distance de environs 7 mètre.

5.1.4Test de la communication RF en mode standard

On essaie de se faire connecter deux module et on mesure la sortie LINK de chaque module pour connaitre l' état de connexion



Remarque :

Si on regarde sur 50s on voit que la connexion n' est pas stable dû tout,
C' est surement du à ma machine d' états RF ou à l' antenne.

6 Conclusion finale

6.1 État actuel du projet

6.1.1 Points fonctionnels et testés

Version corrigée du PCB commandé, monté et testé.
Communication I2C avec le chip 24AA02UID et la lecture de l' UID.
Communication UART entre le module et le KIT.
Commande qui permet d' envoyer l' UID en UART.
Configurer et commander le NRF905 en utilisant le SPI en lecture et en écriture.
Envoi et réception de data RF via le NRF905.
Mode de communication simple (voir page 17).

6.1.2 Points commencés et testés mais non fonctionnel à 100%

La connexion entre les modules dans le mode standard est instable.
Gestion des acknowledge n'est pas fonctionnelle à 100%.

6.1.3 Points non commencés

Mode de communication avancé (voir page 19).
Commande pour changer la taille des trames.
Commande pour changer sont UID.

6.2 Amélioration possible

On pourrait ajouter un connecteur d'antenne (comme sur le Xbee) pour laisser le choix à l'utilisateur entre l'antenne classique ou celle implémentée directement sur le PCB (avec un jupper)
Des points de test supplémentaire (SPI et I2C) ne seraient pas de trop.

6.3 Bilan personnel

Personnellement je pense que ma principale erreur et d'avoir sous-estimé la difficulté de ce projet, au départ j'ai cru qu' il allait être vite terminé. Mais des problèmes de design m'ont beaucoup retardé et ont limité le temps que j'avais à disposition pour faire le code.

Mais j' ai quand même réussi à terminer le mode simple et il ne manque pas grand-chose pour terminer le mode standard.

Pour conclure je dirai que la communication RF est un domaine compliqué mais très intéressant, ce projet m'a appris à prendre en compte tous les cas de figures et à les gérer.

7 Liste des annexes

- Schémas A et B
- PCB A et B
- Implantation
- Liste de pièces, coût réel
- Parties essentielles du code
- Planning initial et planning réel
- Journal de travail
- Mode d'emploi et datasheet du système
- Résumé du projet
- Affiche du projet

16.06.2017
Simonet Yoann