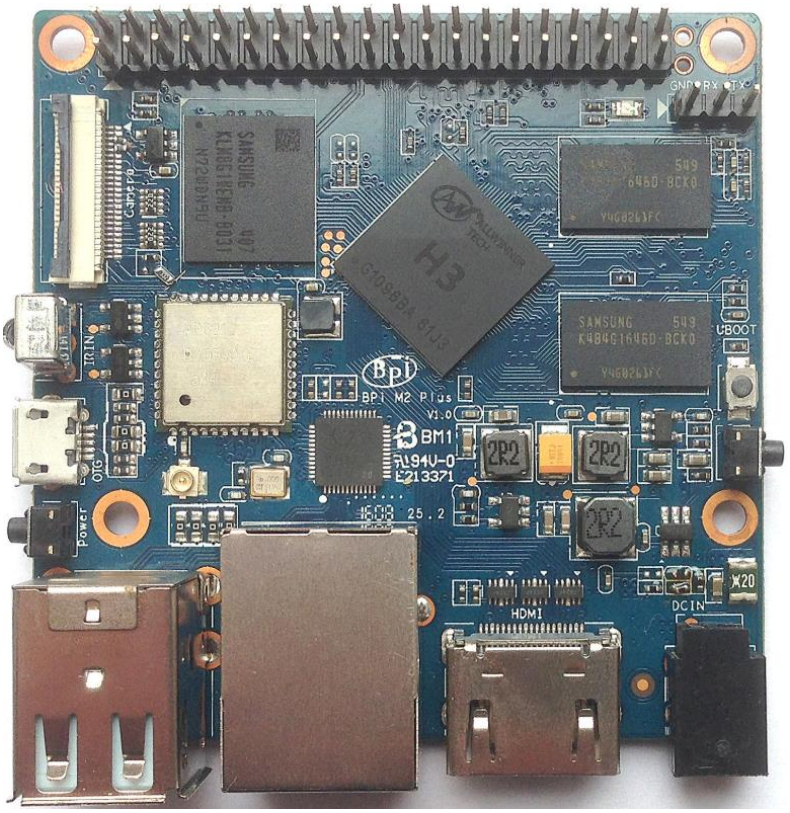
TP 2 : Nrf et transmission de données

IUT de Nice département GEII

IOT

Bananapi M2+



Jean-Louis Salvat
17/03/2019

Table des matières

1 Introduction

Dans ce TP, vous allez utiliser un module radio nrf24l01 afin de communiquer des données entre le banana pi et un microcontrôleur arduino.

Il vous sera fourni, une gateway, ou la connexion entre le banana pi et le nrf24l01 est déjà faite,
si cette Gateway, ne vous ait pas fourni, il faudra brancher le module manuellement file par file en prenant bien soin d'éteindre le banana pi avant de faire le branchement, pensez aussi à valider car des erreurs de câblages peuvent détruire les pins de l'arduino si vous n'utilisez pas la Gateway.

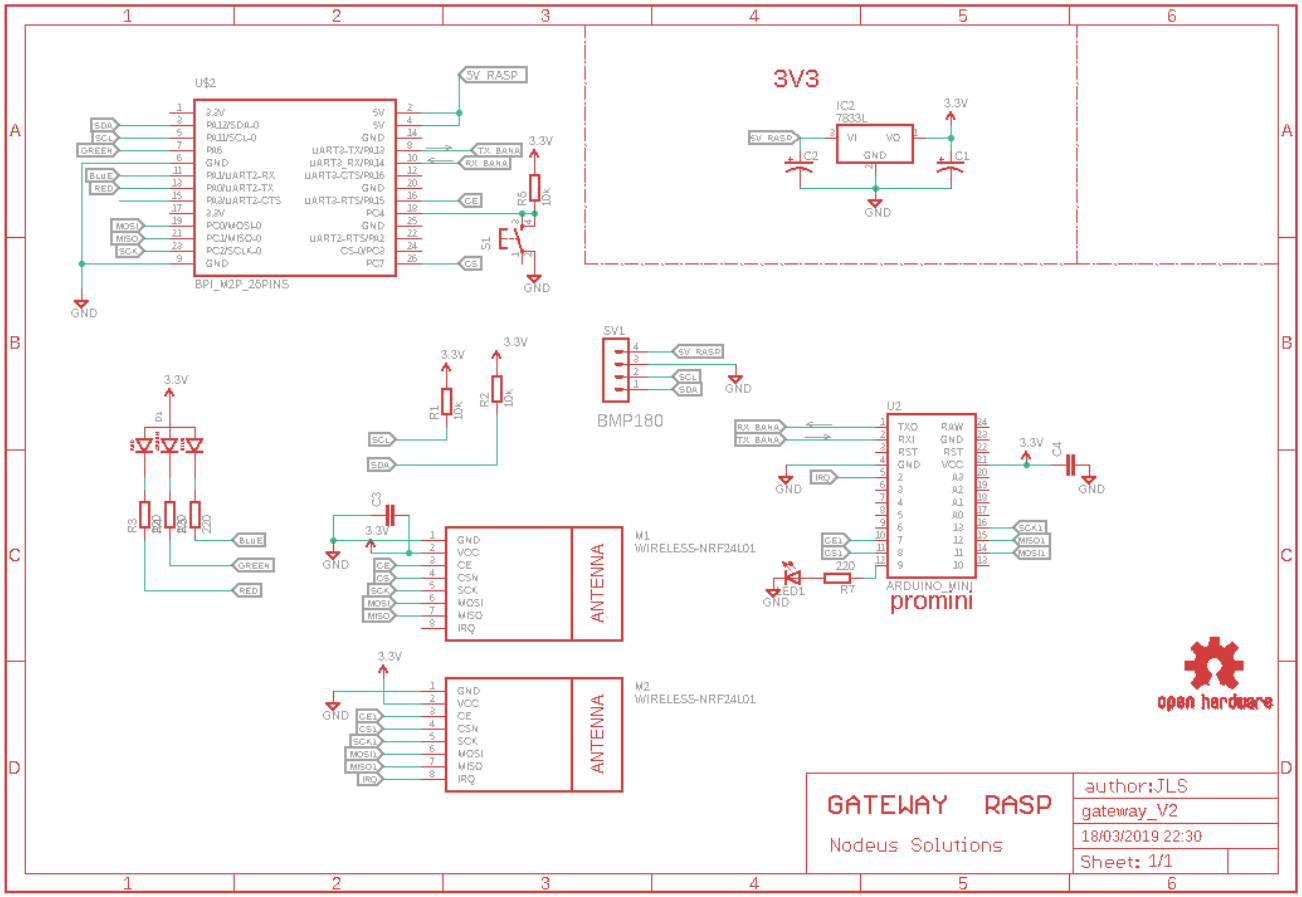
Dans cette gateway, une connexion nrf24l01 → arduino est aussi présente.
Une led connectée à l'arduino intégré à la Gateway est utilisable (p9 de l'arduino).

Et une autre connectée au banana Pi est aussi présente.

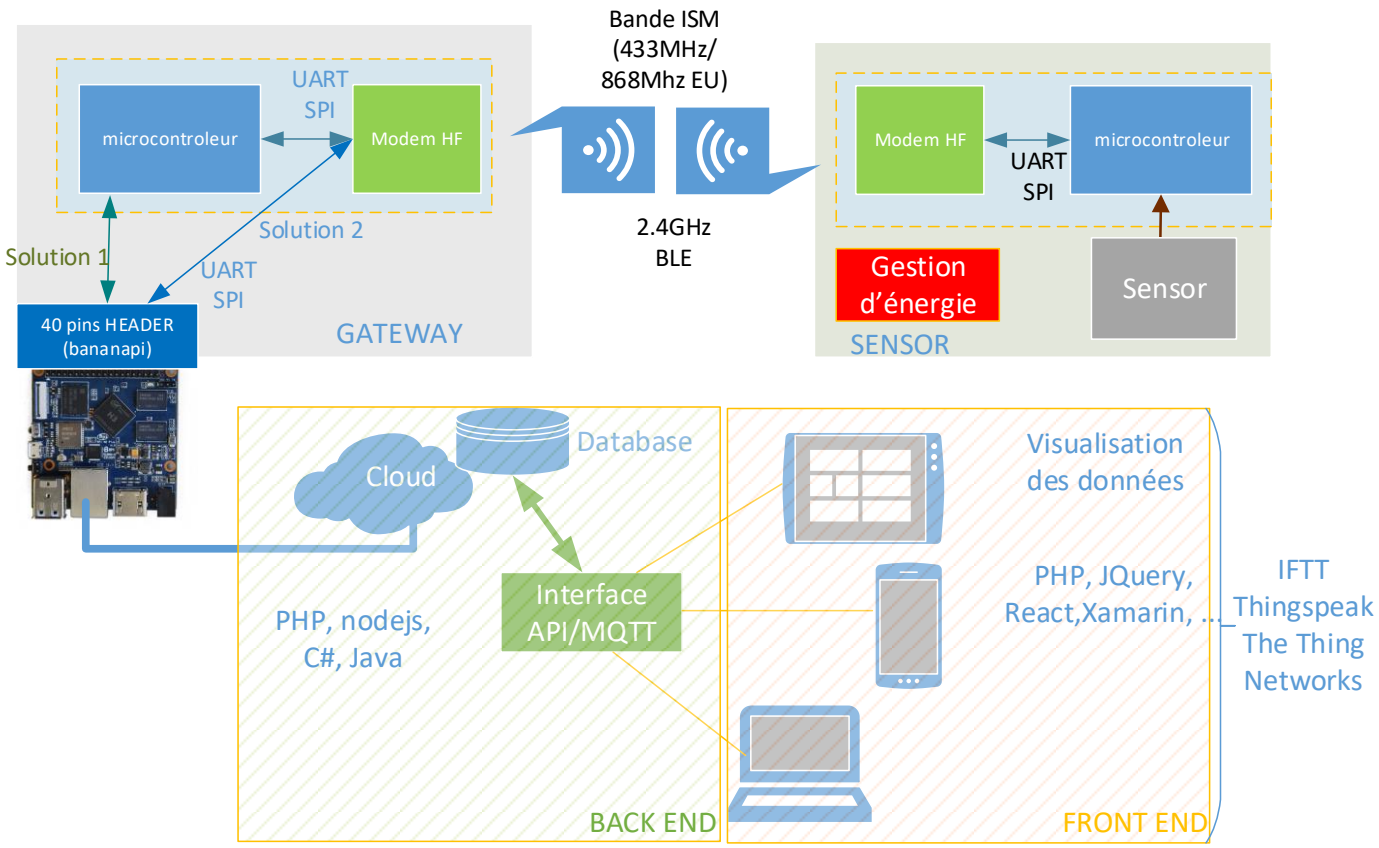
La Gateway permet donc d'avoir dans une même interface plusieurs protocoles de communication utilisables, ce qui vous permet de ne pas avoir à faire les branchements avec des câbles à chaque fois.

Pour savoir tout ce qu'il y a dans la gateway ainsi que les pins utilisés referez vous pouvez vous referez au schématic ci-dessous :

NB : La pin CSN du module radio doit en realité être connectée à la PIN 24 du banana PI.



1.1 L'IOT avec Gateway



Le microcontroleur :

- ✓ Arduino
- ✓ ARM (STM32, Kinetis, Philips,…)
- ✓ Microchip, Texas Instrument, Silabs, …

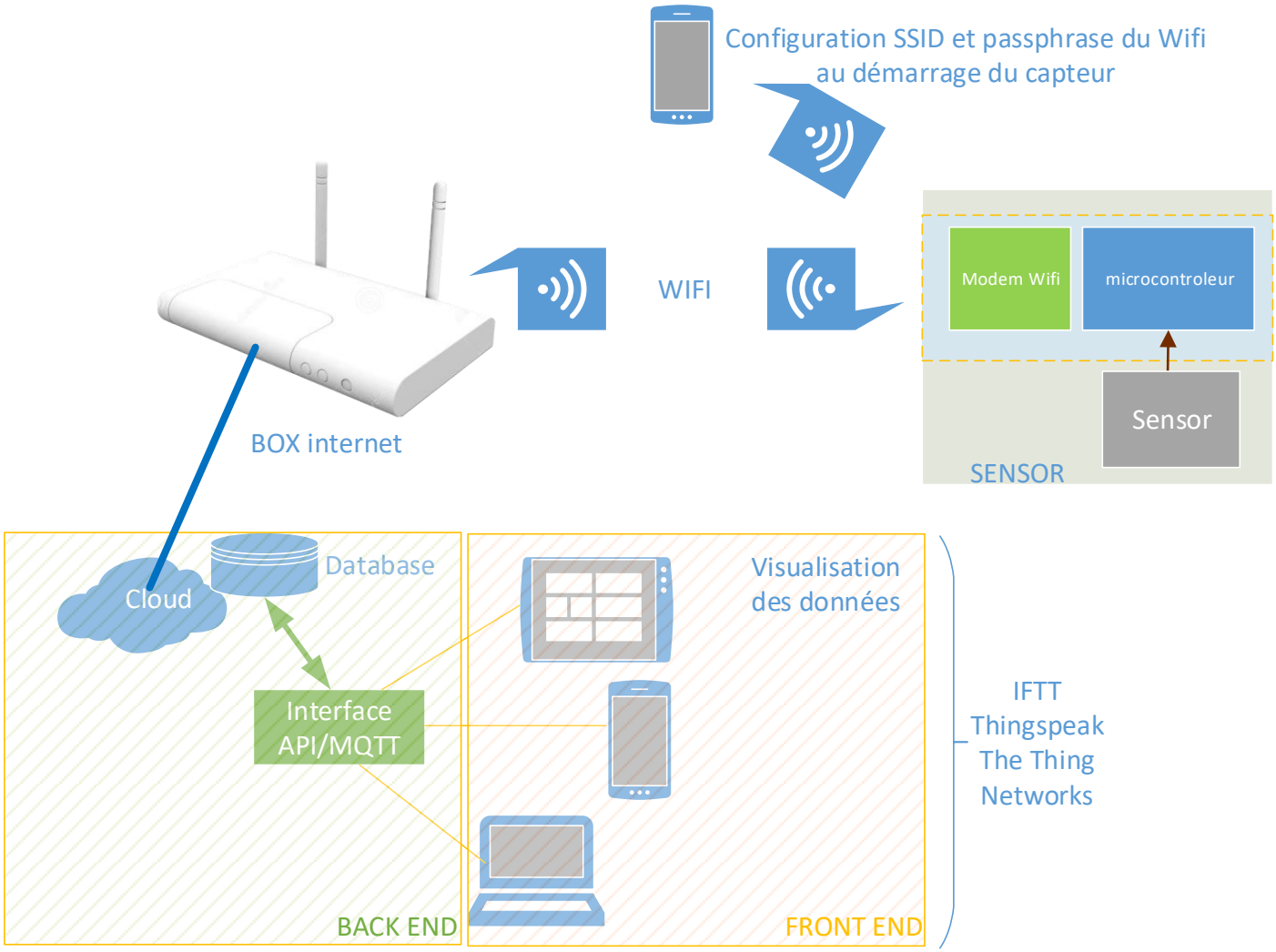
Le modem HF :

- ✓ Récepteur super hétérodyne RXB6 433Mhz et Emetteur 433Mhz
- ✓ Nfr24L01+ (transceiver propriétaire 2.4Ghz)
- ✓ RFM69CW (transceiver propriétaire 868MHz EU ou 433MHz)
- ✓ Semtech Lora (transceiver 868MHz EU ou 433MHz long range)
- ✓ CCxxxx de Texas Instrument (Transceiver propriétaire Sub-GHz ou 2.4Ghz)
- ✓ Zigbee (multi-constructeur)
- ✓ Z-Wave

Le module MCU+HF ou SOC (System On Chip) HF ou Wireless MCU :

- ✓ EFR32 (MCU+HF multi-protocole) de Silicon Lab
- ✓ CCXXXX de Texas Intrument (série Wireless MCU)
- ✓ SAMR21 de Microchip
- ✓ Nrf51 (Bluetooth MCU)

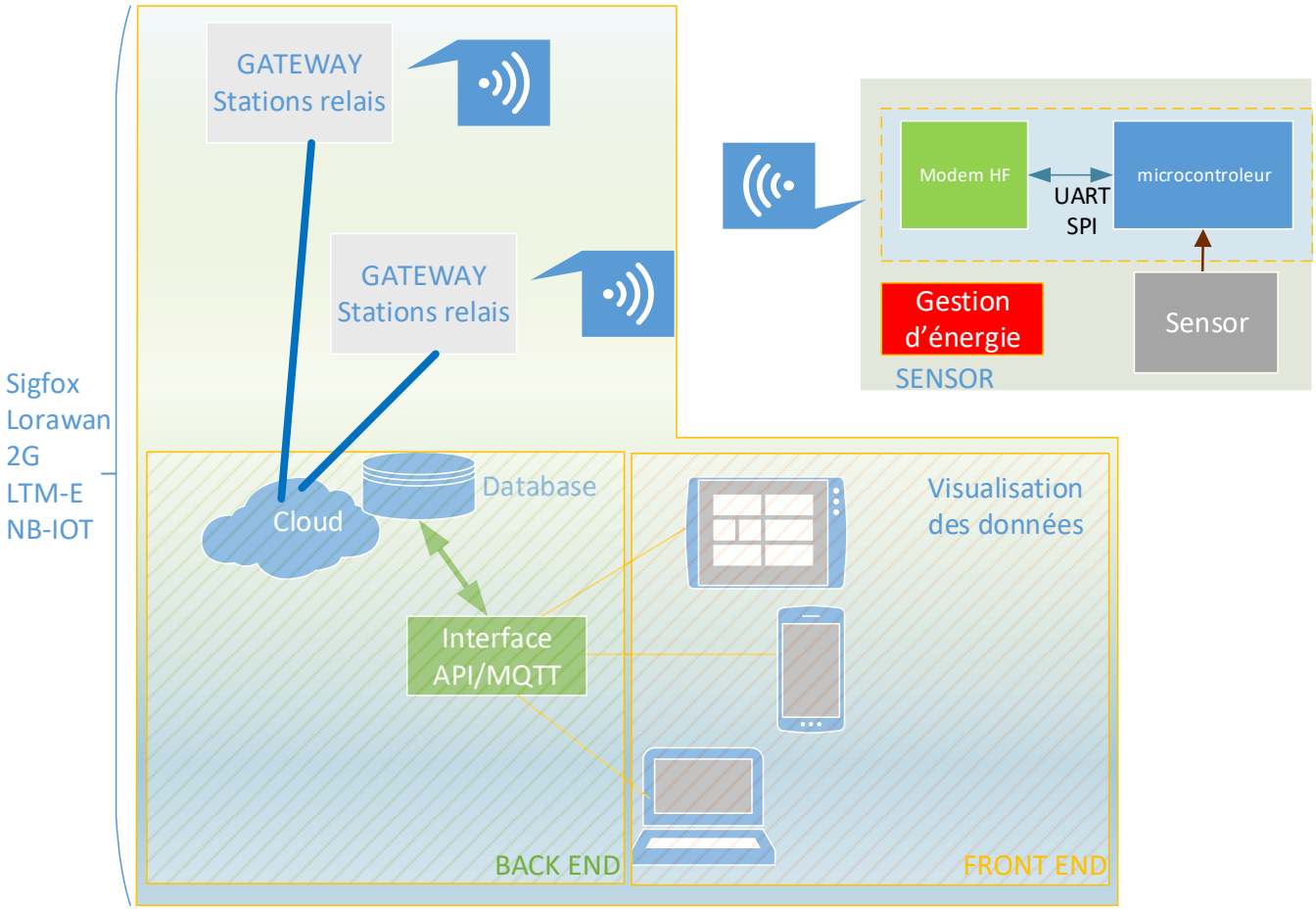
1.2 IOT sans Gateway (avec Wifi)

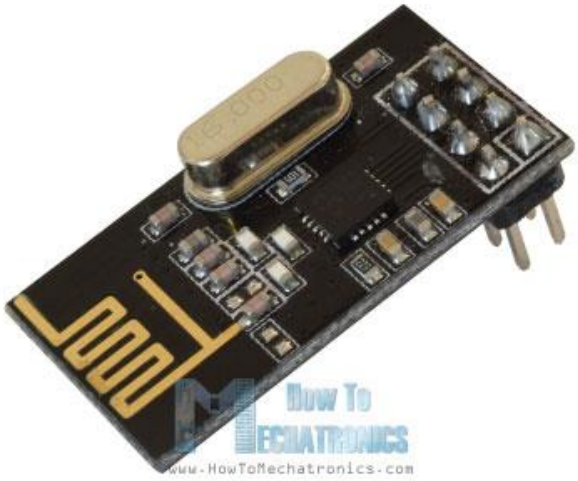


Les modules Wifi + CPU :

- ✓ ESP32 ou ESP8266 (Espressif Wifi CPU)
- ✓ CC31xx ou 32xx Texas Instrument Wifi CPU

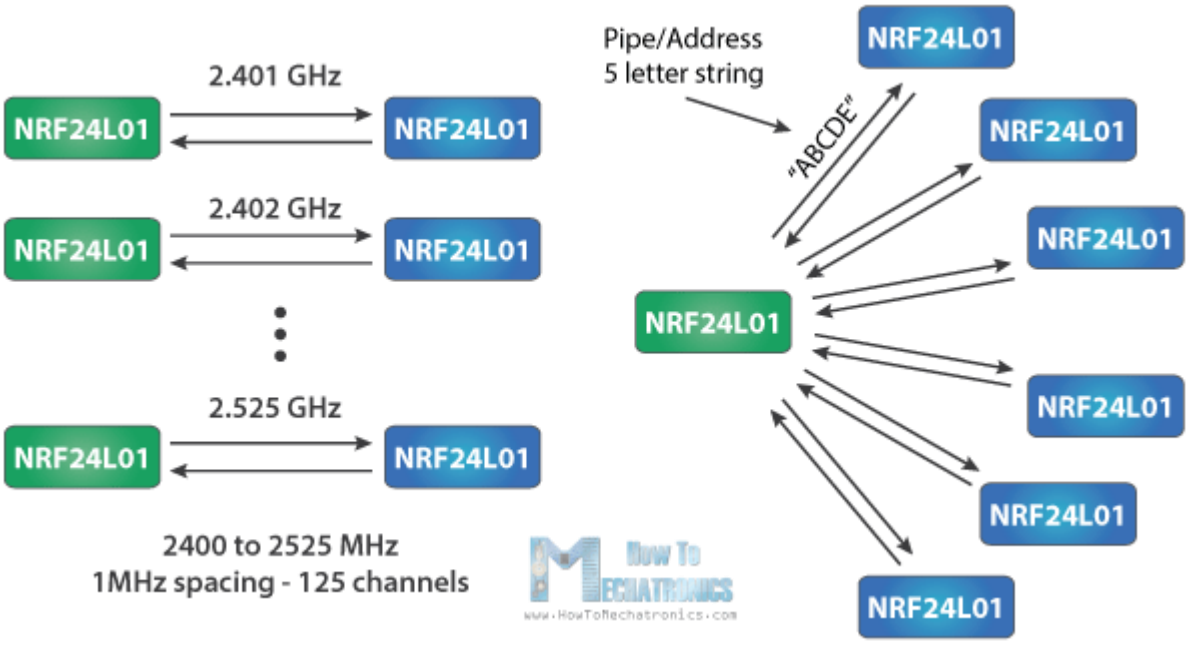
1.3 L'IOT sans Gateway

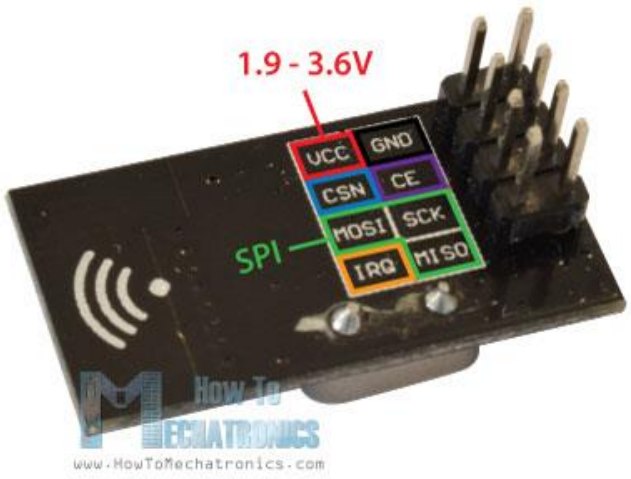




Le nrf utilise la bande de 2.4 GHz et il peut transmettre à des vitesses allant de 250Kbps à 2Mbps.En l'utilisant dans un environnement sans obstacles, il peut effectuer des transmissions jusqu'à 100m.

Le module peut utiliser 125 canaux et chaque canal peut avoir jusqu'à 6 adresses. Le module consomme environ 12mA pendant la transmission et sa plage d'utitlisation de tension est situé enter 1.9 et 3.6V, certains pins peuvent cependant supporter du 5V, ce qui nous permet de facilement l'utiliser avec un arduino.





Les 3 pins en verts utilisent le protocole de communication SPI, elles doivent donc être connectées aux PINS SPI de l'arduino/banana Pi.

Le pin CE et CSN peuvent être connectées à n'importe quel pin de l'arduino, La pin IRQ est une PIN d'interruption, qui n'est pas necessaire d'être connectée.

L'utilisation du nrf24l01 pour communiquer avec le banana pi comporte plusieurs solutions.

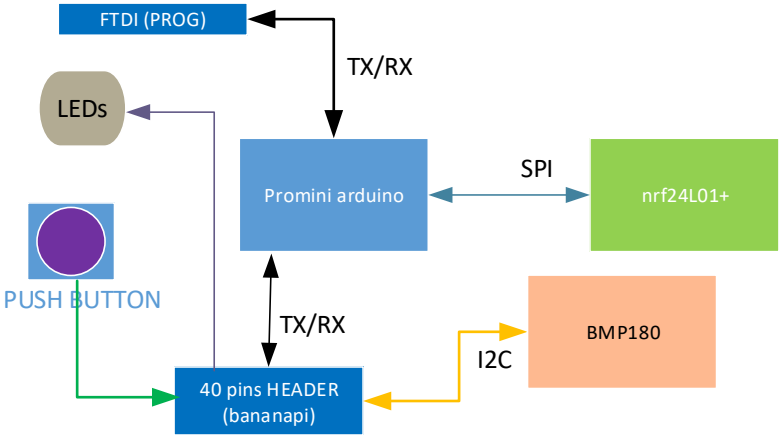
On peut utiliser le nrf24l01 pour communiquer en SPI avec l'arduino, et utiliser l'arduino pour communiquer en TX/RX avec le banana pi : (M2 sur le schématic)

Pour communiquer avec un autre NRF, il faut configurer un canal de lecture dans l'un en etant en écoute, pendant que l'autre utilise ce même canal en écriture afin de lui transmettre des données.

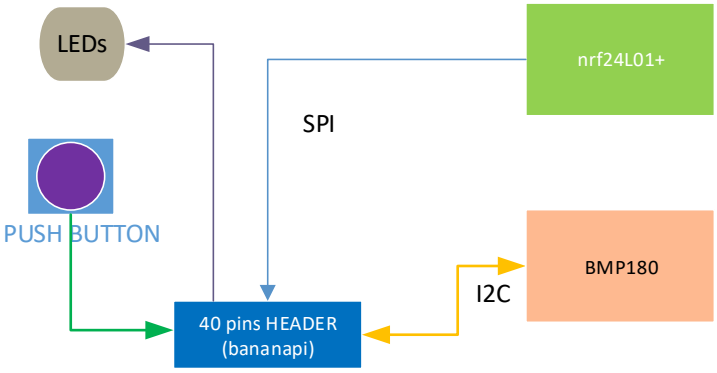
Il faut donc que le canal d'écriture de l'un ait la même adresse que le canal de lecture de l'autre, nous verrons cela à la fin du TP, lorsqu'on va communiquer entre 2 nRF.

L'utilisation du nrf24l01 pour communiquer avec le banana pi comporte plusieurs solutions.

On peut utiliser le nrf24l01 pour communiquer en SPI avec l'arduino, et utiliser l'arduino pour communiquer en TX/RX avec le banana pi : (M2 sur le schématic)



On peut aussi communiquer entre le nrf 24l01+ et le banana pi en SPI (M1 sur le schematic) diretement.



Bien entendu, le but principal, est de communiquer avec le banana pi avec le nrf, d'utiliser le nrf pour envoyer des données à l'arduino avec le banana pi et ensuite les lire sur l'arduino.

Nous allons commencer par voir la deuxième solution :

1 – Coté banana pi :

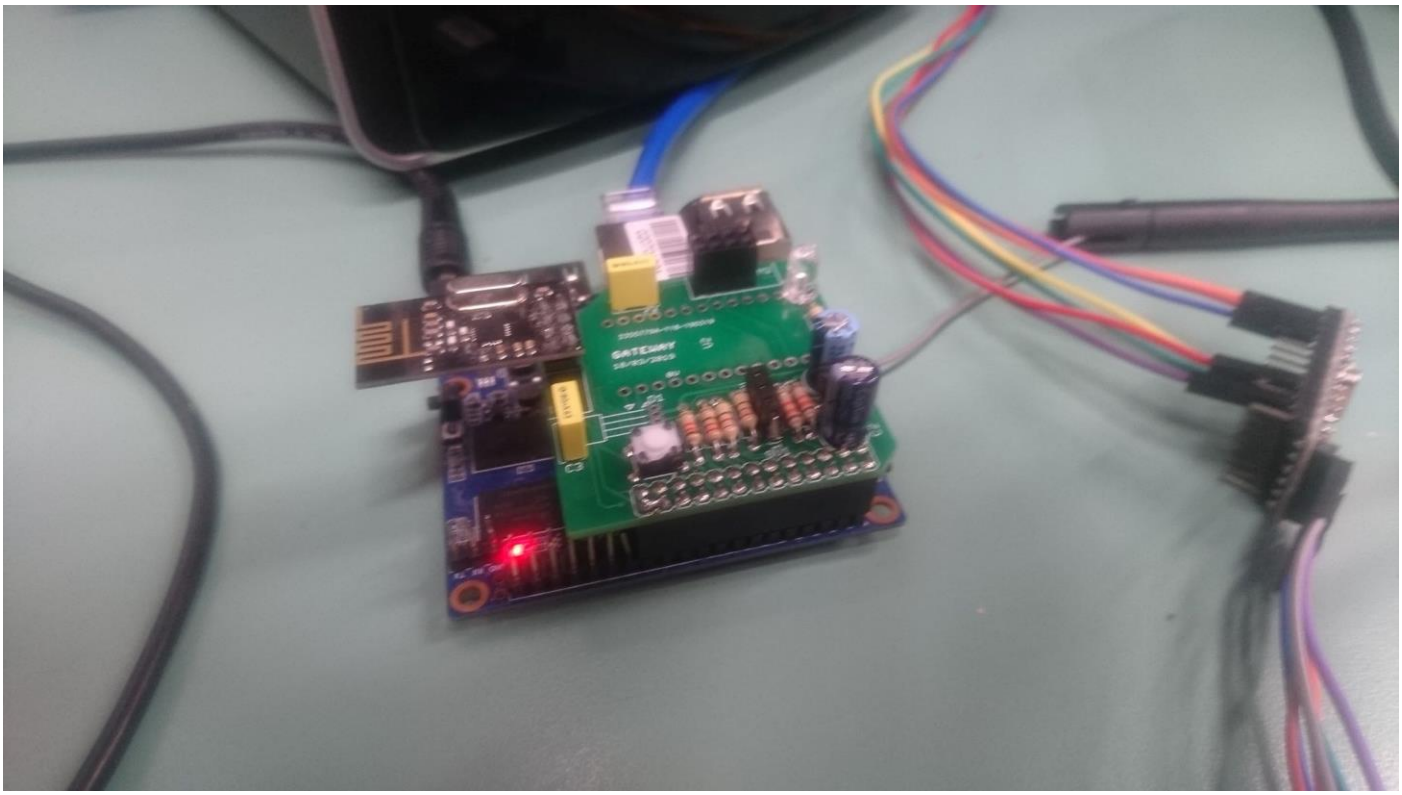
Liens Utiles :

http://spencernusbaum.com/wordpress/index.php/2018/07/22/working-with-the-nrf24l01-transcievers-on-the-raspberry-pi-and-arduino/

http://www.rvq.fr/linux/bananapi.php

https://www.framboise314.fr/faire-dialoguer-un-raspberry-et-un-arduino-via-nrf24l01/

http://linux-sunxi.org/SPIdev



Tout d'abord, nous devons activer le spi sur notre banana pi, pour cela, Il faut en fait modifier le fichier /boot/armbianEnv.txt comme expliqué ici :

https://docs.armbian.com/User-Guide_Allwinner_overlays/#example-bootarmbianenvtxt-contents

On tape donc : **nano /boot/armbianEnv.txt** afin d'acceder à ce fichier et on le modifie comme suit :

```
verbosity=1
logo=disabled
console=both
disp_mode=1920x1080p60
overlay_prefix=sun8i-h3
rootdev=UUID=928d1dc5-2a97-4fcb-bd20-149f2a3bb701
rootfstype=ext4
overlays=i2c0 pwm spi-spidev uart3
param_spidev_spi_bus=0
usbstoragequirks=0x2537:0x1066:u,0x2537:0x1068:u
```

Il faudra ensuite rajouter deux elements à ce fichier :

Dans overlays =

Rajouter spi-spidev

Et rajouter la ligne :

Param_spidev_spi_bus=0

Rebootez ensuite pour appliquer les changements **! sudo reboot** et verifier la présence de spi en tapant : ls –la /dev/spi*

```
geii@yhstg:~$ ls -la /dev/spi*
crw------- 1 root root 153, 0 Apr 30 13:24 /dev/spidev0.0
```

Si vous avez spidev0.0 qui s'affiche ! Vous pouvez passer à la suite, sinon, verifier que vous avez bien modifier le fichier precedent !

Pour tester si on peut utiliser spidev0.0, on peut utiliser spi-tool :

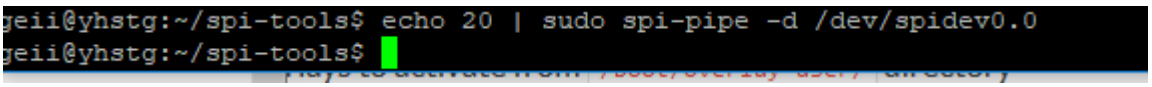
git clone https://github.com/cpb-/spi-tools.git

cd spi-tools

sudo apt install automake

```
autoreconf -fim
./configure
make
```

```
sudo make install
echo 20 | sudo spi-pipe -d /dev/spidev0.0
```

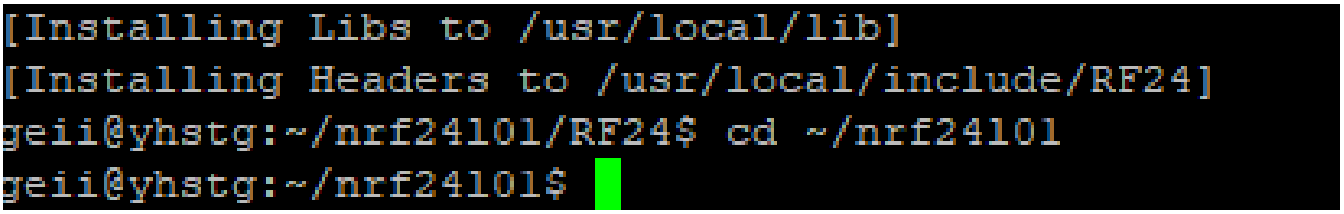


Si vous n'avez pas d'erreurs c'est quon a reussi à utiliser le SPI, et vous pouvez donc passer à la suite :

Il faut maintenant importer les librairies du nrf24l01 dans votre cible, et ensuite la compiler :

Vous n'avez qu'à suivre les commandes suivantes :

cd
mkdir nrf24l01
cd nrf24l01
git clone https://github.com/TMRh20/RF24.git
cd RF24
make
sudo make install



Une fois vos libraries instalées, et votre SPI activé, il est temps de voir si votre nrf24l01 fonctionne.

Pour ce premier test, aller dans utility/SPIDEV avec la commande cd :

cd ~/nrf24l01/RF24/utility/SPIDEV

ensuite ls pour voir si vous avez bien le fichier spi.cpp

```
geii@bpstage:~/nrf24l01/RF24$ cd utility
geii@bpstage:~/nrf24l01/RF24/utility$ cd SPIDEV/
geii@bpstage:~/nrf24l01/RF24/utility/SPIDEV$ ls
compatibility.c  gpio.cpp  includes.h   interrupt.h         spi.cpp
compatibility.h  gpio.h    interrupt.c  RF24_arch_config.h  spi.h
geii@bpstage:~/nrf24l01/RF24/utility/SPIDEV$ sudo nano spi.cpp
```

Il faut modifier ce fichier pour lui indiquer qu'on utilise spidev0.0 :

Dans la fonction begin :

```
void SPI::begin(int busNo,uint32_t spi_speed){

   /* set spidev accordingly to busNo like:
    * busNo = 23 -> /dev/spidev2.3
    *
    * a bit messy but simple
    * */
     char device[] = "/dev/spidev0.0";
    char output[40];
//     device[11] += (busNo / 10) % 10;
//     device[13] += busNo % 10;

     if(this->fd >=0) // check whether spi is already open
     {
          close(this->fd);
          this->fd=-1;
     }
```

Ensuite revenir dans le dossier RF24 en tapant à deux reprise la commande : **cd ..**

Nous allons maintenant utiliser le fichier éxecutable configure afin de créer le makefile :

```
./configure --extra-cflags=-DRF24_SPIDEV_SPEED=100000
```

Une fois cela fait, vous pouvez utiliser le make afin de créer les librairies dont on aura besoin pour plus tard :

make

sudo make install

Bien vous avez maintenant tout ce qu'il faut pour utiliser le nrf, testons le :

Dans le dossier RF24, vous trouvez un dossier qui s'appelle examples_linux :

cd ~/nrf24l01/RF24/examples_linux

```
geii@bpstage:~/nrf24l01/RF24$ cd examples_linux
geii@bpstage:~/nrf24l01/RF24/examples_linux$ ls
extra                          Makefile.examples  receive.cpp
gettingstarted                 Makefile.save      sender
gettingstarted_call_response.cpp  pingpair_dyn.cpp   sender.cpp
gettingstarted.cpp             pingpair_dyn.py    transfer.cpp
interrupts                     readme.md
Makefile                       receive
```

Dans ce dossier avec ls, vous trouverez un fichier gettingstarted.cpp, à l'aide la commande nano, ouvrir ce fichier et l'adapter aux pins utilisées avec notre gateway PIN 16 et 24 : (PA15 et PC3)

```
geii@bpstage:~/nrf24l01/RF24/examples_linux$ sudo nano gettingstarted.cpp
RF24 radio(15,67);// PA 15 et pc 3 (64+3)
```

Vous pouvez ensutie compiler gettingstarted avec :

make gettingstarted

```
geii@bpstage:~/nrf24l01/RF24/examples_linux$ make gettingstarted
arm-linux-gnueabihf-g++ -Ofast -Wall -pthread  -I/usr/local/include/RF24/.. -
I.. -L/usr/local/lib gettingstarted.cpp -lrf24 -o gettingstarted
```

Et enfin lancer avec :

sudo ./gettingstarted

```
geii@bpstage:~/nrf24l01/RF24/examples_linux$ sudo ./gettingstarted
RF24/examples/GettingStarted/
STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1     = 0x3130303030 0x3030303030
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0x3130303030
RX_PW_P0-6       = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA            = 0x3f
EN_RXADDR        = 0x03
RF_CH            = 0x4c
RF_SETUP         = 0x03
CONFIG           = 0x0e
DYNPD/FEATURE    = 0x00 0x00
Data Rate        = 1MBPS
Model            = nRF24L01+
CRC Length       = 16 bits
PA Power         = PA_LOW

 ************ Role Setup ***********
Choose a role: Enter 0 for pong_back, 1 for ping_out (CTRL+C to exit)
>
```

(Ctrl + c pour arrêter)

Erreur cablage ou gateway HS :

```
RF24/examples/GettingStarted/
STATUS           = 0x00 RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=0 TX_FULL=0
RX_ADDR_P0-1     = 0x0000000000 0x0000000000
RX_ADDR_P2-5     = 0x00 0x00 0x00 0x00
TX_ADDR          = 0x0000000000
RX_PW_P0-6       = 0x00 0x00 0x00 0x00 0x00 0x00
EN_AA            = 0x00
EN_RXADDR        = 0x00
RF_CH            = 0x00
RF_SETUP         = 0x00
CONFIG           = 0x00
DYNPD/FEATURE    = 0x00 0x00
Data Rate        = 1MBPS
Model            = nRF24L01
CRC Length       = Disabled
PA Power         = PA_MIN
```

Si ça affiche **cant open device**, retourner à la partie ou on modifie spi.cpp, et recommencer :

Si votre cablâge est correct :

Vous pouvez aller sur ce lien :

https://gist.github.com/Mizaystom/2d12d42dad42b34d42a374b457a6712c

Et utiliser votre méthode preferée pour mettre le fichier .cpp dans le dossier linux_examples :

Méthodes possibles :

. Créer un fichier avec nano sender.cpp, et copier coller le code.

. Utiliser WINscp et faire un tirer glisser du code télechargé

. Télecharger avec commande git ! se placer dans le bon répertoire avant

Toujours dans linux_examples :

Sudo nano Makefile

Dans PROGRAMMS =

Rajouter sender !

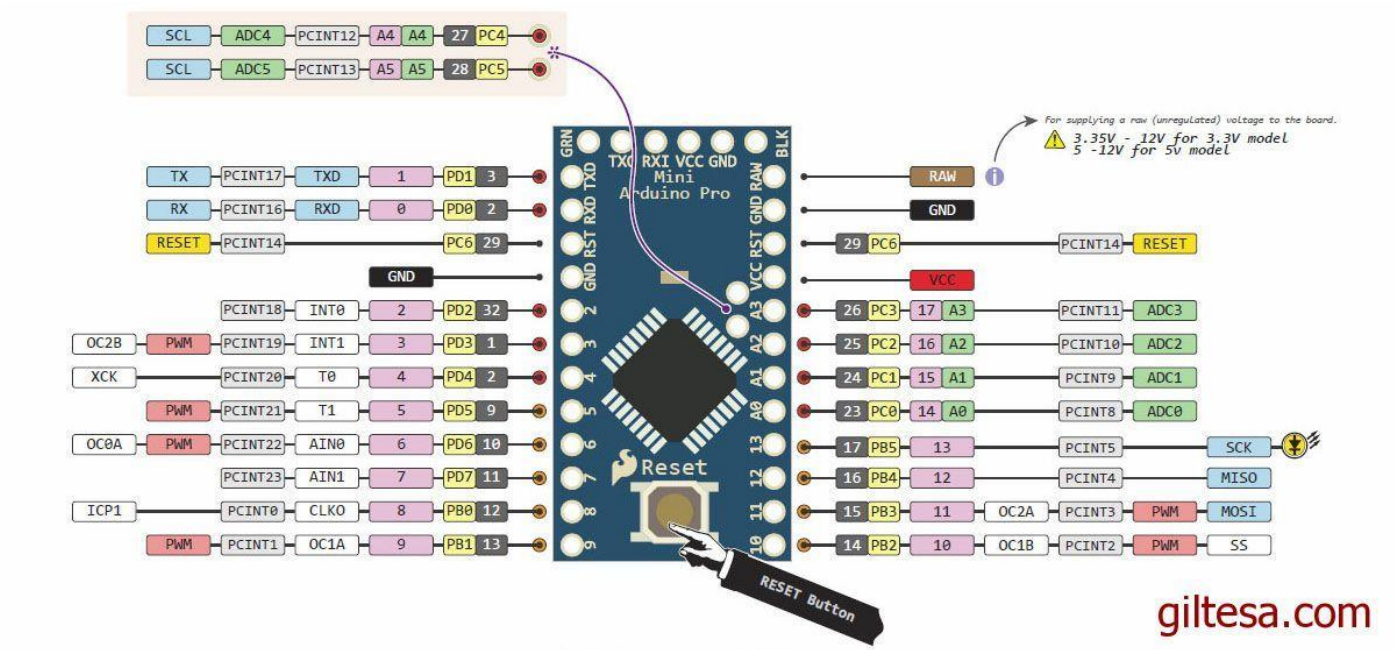
```
# define all programs
PROGRAMS = gettingstarted receive sender
```

Ctrl X et Y pous sauvegarder.

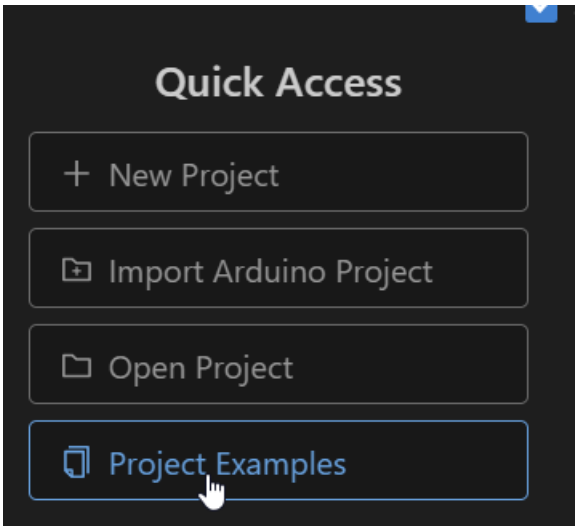
Vous pouvez compiler le programme avec : **make sender**

Ce programme nous servira à envoyer une info à un autre NRF, dans notre cas à celui qui sera connecté directement à l'arduino, ce que nous allons voir tout de suite :

2 – Coté Arduino

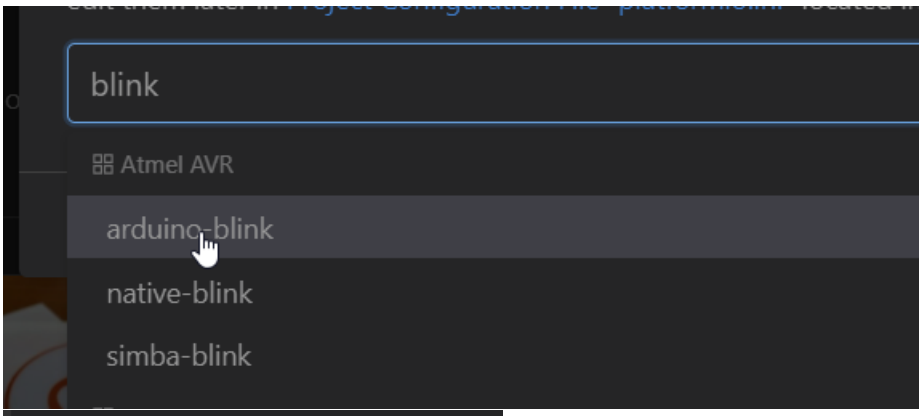


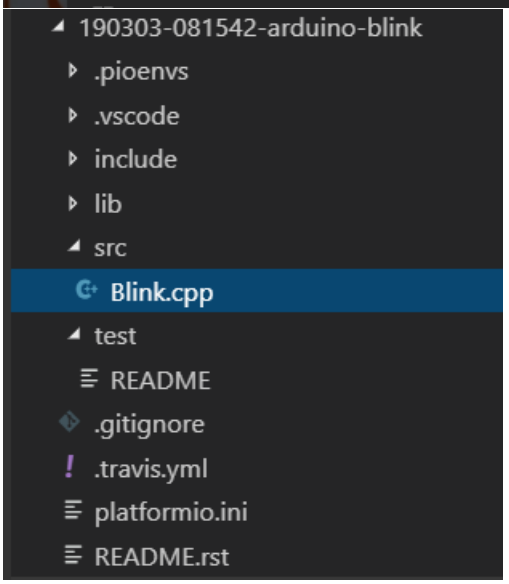
Nous utilisons un arduino PRO MINI, ça sera celui la que l'on aura dans la gateway, mais nous allons tout d'abord faire les tests en le connectant directement au PC à l'aide d'un FTD et en utilisant Plateform IO, ou l'on va créer un nouveau projet :



On va pour cela choisir des petits projets déjà tout fait en allant dans Project Examples

Chercher ensuite le projet arduino-blink

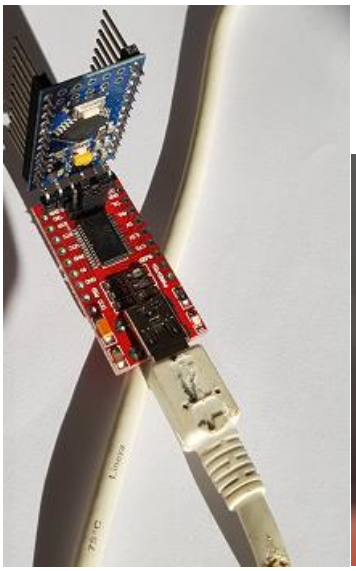


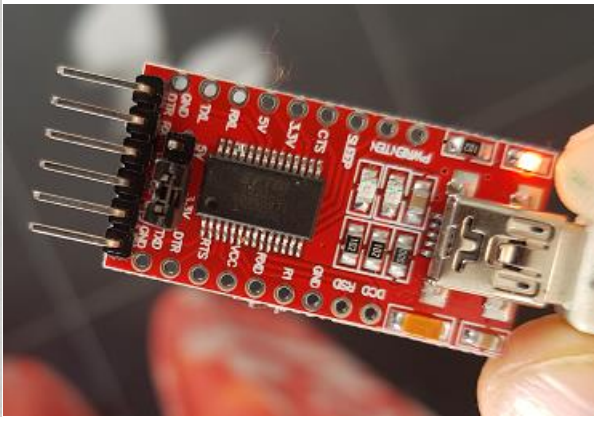


Une fois votre projet crée,

Modifier plateform.ini (qui est dans le projet) comme ceci :

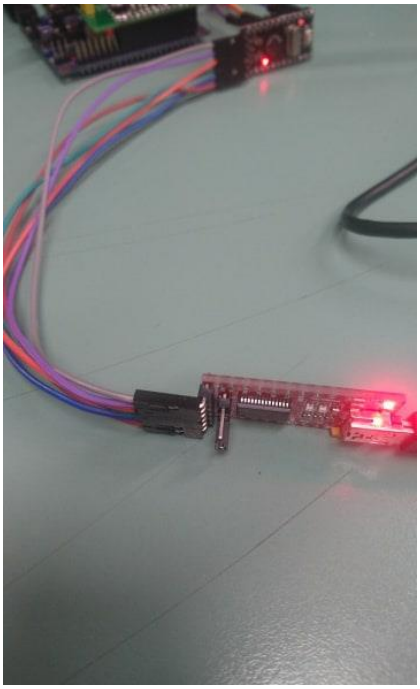
```
[env:pro16MHzatmega328]
platform = atmelavr
framework = arduino
board = pro16MHzatmega328
```





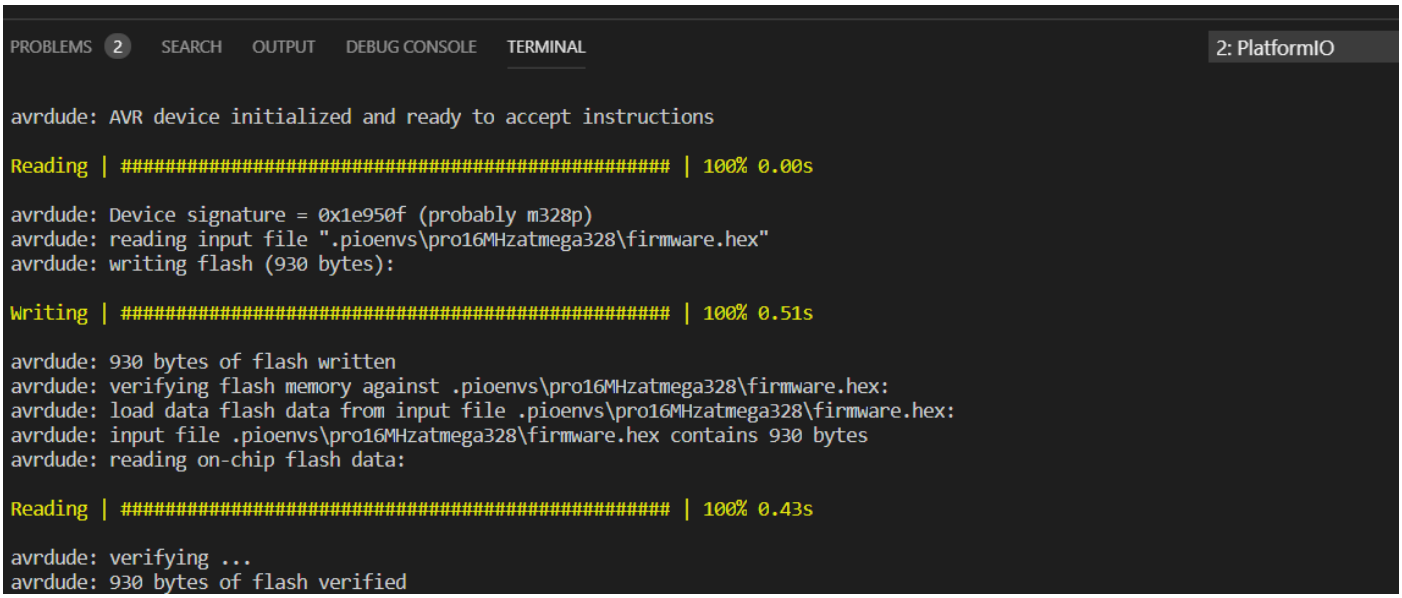
Sur le FTD, mettre le petit capuchon sur le côté 3V3.

Ensuite relier les pins de la partie supérieure de l'arduino à celles du FTD (faites bien attention à relier les bonnes PINS entres elles.



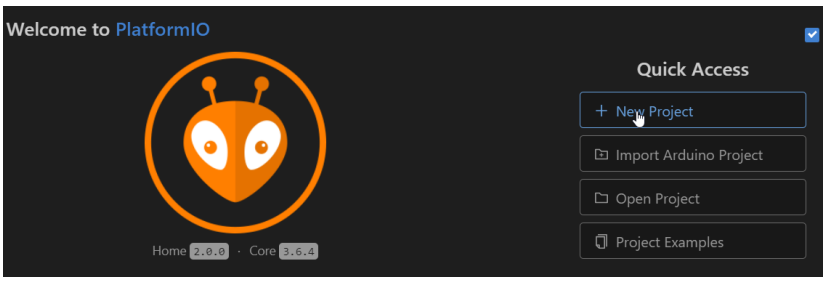
Imbriqué l'un dans l'autre avec les trous, s'il ya des PINS soudées, utiliser des cables femelles comme ci-dessus, quand vous imbriquez faites attention à le mettre dans le bon sens.

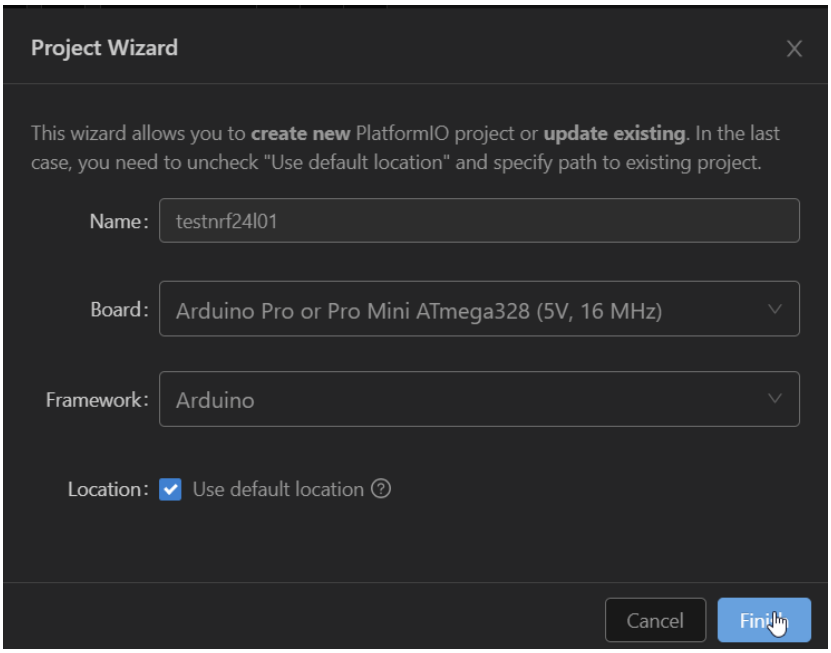
Vous pouvez ensuite compiler et télecharger le code sur plateform IO à l'aide de ce bouton ou dans PIO HOME en appuyant sur upload.


PROBLEMS 2
SEARCH
OUTPUT
DEBUG CONSOLE
TERMINAL
2: PlatformIO
avrdude: AVR device initialized and ready to accept instructions
Reading | ## | 100% 0.00s
avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file ".pioenvs\pro16MHzatmega328\firmware.hex"
avrdude: writing flash (930 bytes):
Writing | ## | 100% 0.51s
avrdude: 930 bytes of flash written
avrdude: verifying flash memory against .pioenvs\pro16MHzatmega328\firmware.hex:
avrdude: load data flash data from input file .pioenvs\pro16MHzatmega328\firmware.hex:
avrdude: input file .pioenvs\pro16MHzatmega328\firmware.hex contains 930 bytes
avrdude: reading on-chip flash data:
Reading | ## | 100% 0.43s
avrdude: verifying ...
avrdude: 930 bytes of flash verified

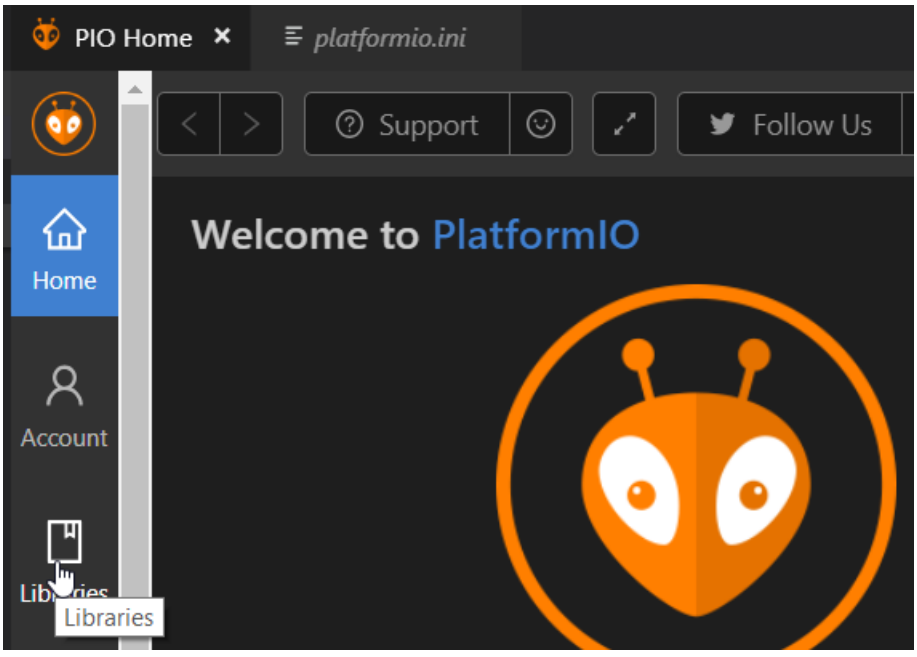
Bien maintenant que votre programme est testé, nous pouvons attaquer le nrf !

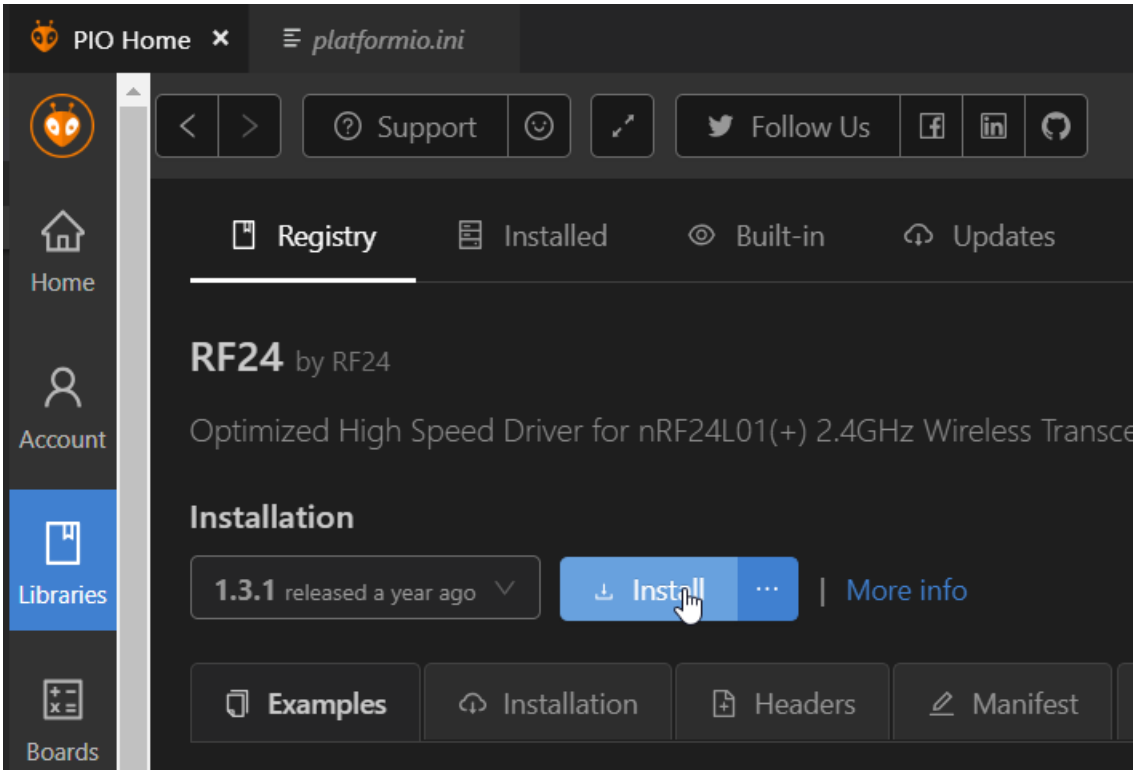
Nous allons pour cela créer un nouveau projet, en choisissant la carte Pro Mini ATmega 5V 16 Mhz.





Nous allons ensuite, en restant dans PIO Home, telechargé les librairies RF24 et LowPower :



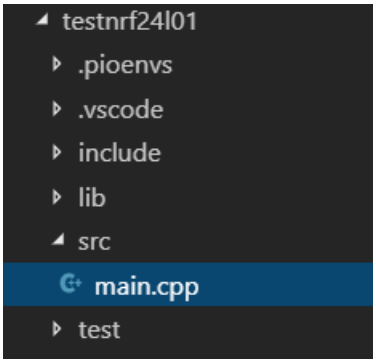


Une fois les deux librairies installées, allez dans plateformio.ini et le changer comme suit :

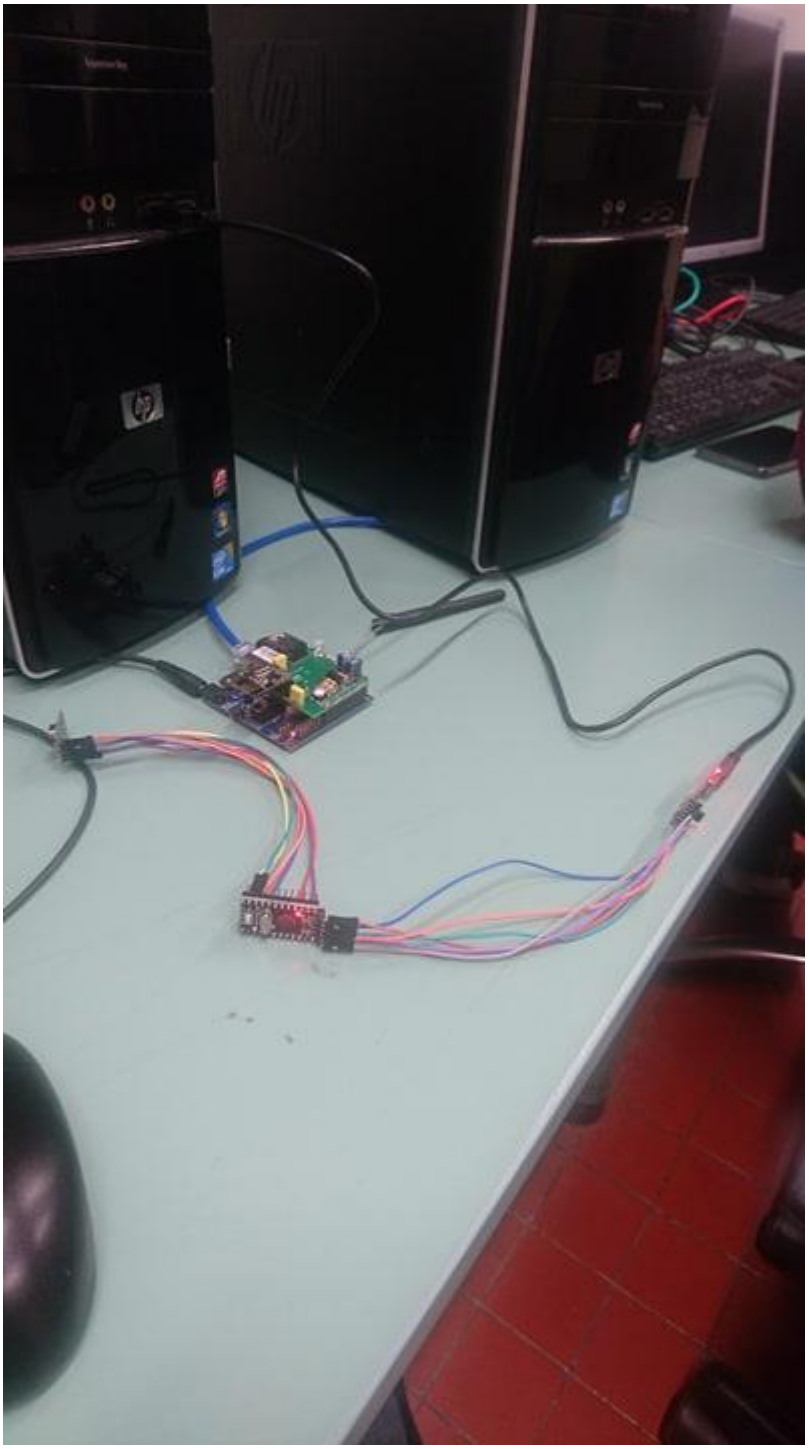
```
[env:pro16MHzatmega328]
platform = atmelavr
board = pro16MHzatmega328
framework = arduino
lib_deps =
    RF24
    Low-Power
monitor_speed = 115200
```

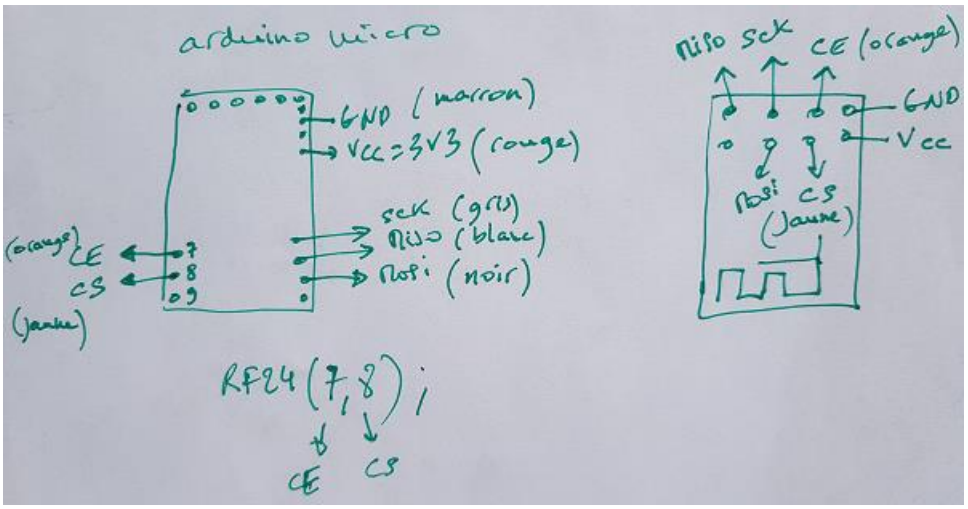
Une fois vos librairies installées, allez sur ce lien :
https://gist.github.com/Mizaystom/10a162db01b228476e740c8782bc07c0

Copier coller le code dans votre main.cpp qui se trouve dans src :



Vous pouvez maintenant connecté votre nrf24l01 à votre arduino, lui-même connecté au PC comme cela :





Une fois votre code importé et votre arduino connecté, vous pouvez compileret televerser le programme comme fait precedemment :

Vous pouvez aussi ouvrir un terminal et taper la commande pio device monitor

Vous devez avoir un résultat similaire à celui-ci :

La bonne nouvelle cest que le code que vous venez de compiler, lit dans le nœud ou le programme sender.cpp ecrit.

Si vous lancer le programme sur votre arduino (compile and monitor), et que sur votre banana pi vous lanciez sudo ./sender qu'on a compilé tout à l'heure, sender enverra à votre arduino des messages, et votre arduino reagira en les affichant et en changeant le statut des leds.

Sur le banana Pi :

sudo ./sender

```
Setup fait !
On arrête d'écouter pour envoyer
Texte envoyé
```

Sur l'arduino :

```
STATUS          = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1    = 0x3030303030 0x3130303030
RX_ADDR_P2-5    = 0xc3 0xc4 0xc5 0xc6
TX_ADDR         = 0x3030303030
RX_PW_P0-6      = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA           = 0x3f
EN_RXADDR       = 0x02
RF_CH           = 0x4c
RF_SETUP        = 0x23
CONFIG          = 0x0e
DYNPD/FEATURE   = 0x00 0x00
Data Rate       = 250KBPS
Model           = nRF24L01+
CRC Length      = 16 bits
PA Power        = PA_LOW
```

```
Message de l'arduino : Transmission recue
message de votre banana pi:Led arduino allumée
```

Bien entendu comme precedemment si vous avez des 0x00 ou des 0xFF partout, vous avez une erreur de cablage, donc faites attention à cela !

Communication avec envoie de données :

Maintenant que vous savez comment les nrf fonctionnent, à vous de jouer.

A l'aide du programme de lecture des données du capteur qu'on a fait au TP1, et en combinant avec sender.cpp, faites un code qui lit les données du capteur, qui les range dans un buffer et ensuiter envoyer ce buffer au nrf connecté à Arduino.

Il n'y aura rien à changer côté réception, mais parcontre côté reception, il y aura plusieurs choses à faire.

Une fois que vous avez terminé votre programme, vous pouvez suivre le cours sur le makefile (TP 0) pour le modifier et compilez avec, soit compiler directement votre code avec g++.

Afin que votre code marche, il faudra être dans le même dossier que sender, et il faudra aussi copier bmp180.h dans ce même dossier.

N'oubliez pas de lier votre bibliothéque rf24 lors de votre compilation si vous ne la faites pas avec votre makefile.

```
g++ captsend.cpp -o captsend -lrf24 -lwiringPi
```

Cahier des charges :

. Lecture des données du capteur

. Transmission seulement lors de l'appui du bouton de la gateway

. Allumage de la led en rouge lors de la transmission

. Passe en mode Low Power en fin de transmisson

Pour passer en mode économie d'energie après transmission, utiliser le tableau ci-dessous :

Function	Radio Mode	Current Consumption
radio.begin();	Standby-I	.026mA
radio.startListening();	Active	11-15mA
delay(1000);	Active	11-15mA
radio.stopListening();	Standby-I	.026mA
radio.write();	Active	11-15mA
delay(1000);	Standby-I	.026mA
radio.writeFast()	Active	11-15mA
delay(1000);	Standby-II	0.320mA
radio.powerDown();	PowerDown	.0009mA

Voici le résultat attendu :

Côté banana pi :

```
Temperature : 27.10 C
Pressure :    997.57 Pa
Altitude :    131.87 h
bp = 0
appuyez sur le bouton environ 1 seconde pour envoyer les donnees
On envoie les donnees au NRF receveur
STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1     = 0x3130303030 0x3030303030
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0x3130303030
RX_PW_P0-6       = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA            = 0x3f
EN_RXADDR        = 0x03
RF_CH            = 0x4c
RF_SETUP         = 0x23
CONFIG           = 0x0e
DYNPD/FEATURE    = 0x00 0x00
Data Rate        = 250KBPS
Model            = nRF24L01+
CRC Length       = 16 bits
PA Power         = PA_LOW
Setup du nrf fait !
On arrête d'écouter pour envoyer
Donnees envoyees
Donnees envoyees mode low power...
```

Côté Arduino :

```
Message de l'arduino : Transmission recue
message de votre banana pi:T=27 C P=997 Pa A=131 m
```

Correction :

https://gist.github.com/Mizaystom/3152790902cb258096da1ad1c8a41886

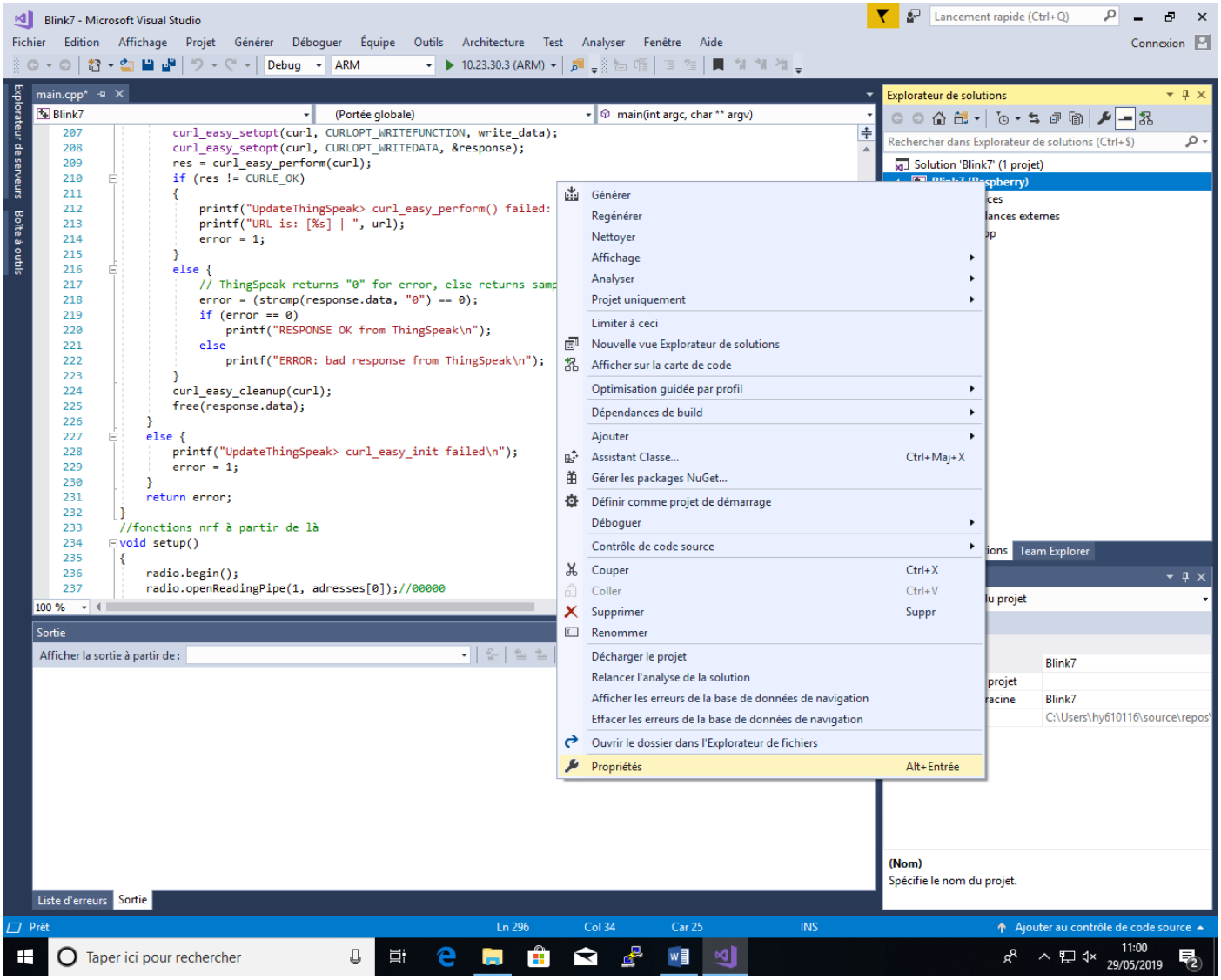
Afin de faciliter le travail quand vous voulez coder sur votre cible, nous allons voir comment vous pouvez faire ce code sur visual studio community.

Une fois votre code tapé sur le logiciel, il lui faudra en effet indiquer comment utiliser plusieurs librairies et fichier d'entêtes qui'il ne prends pas automatiquement.

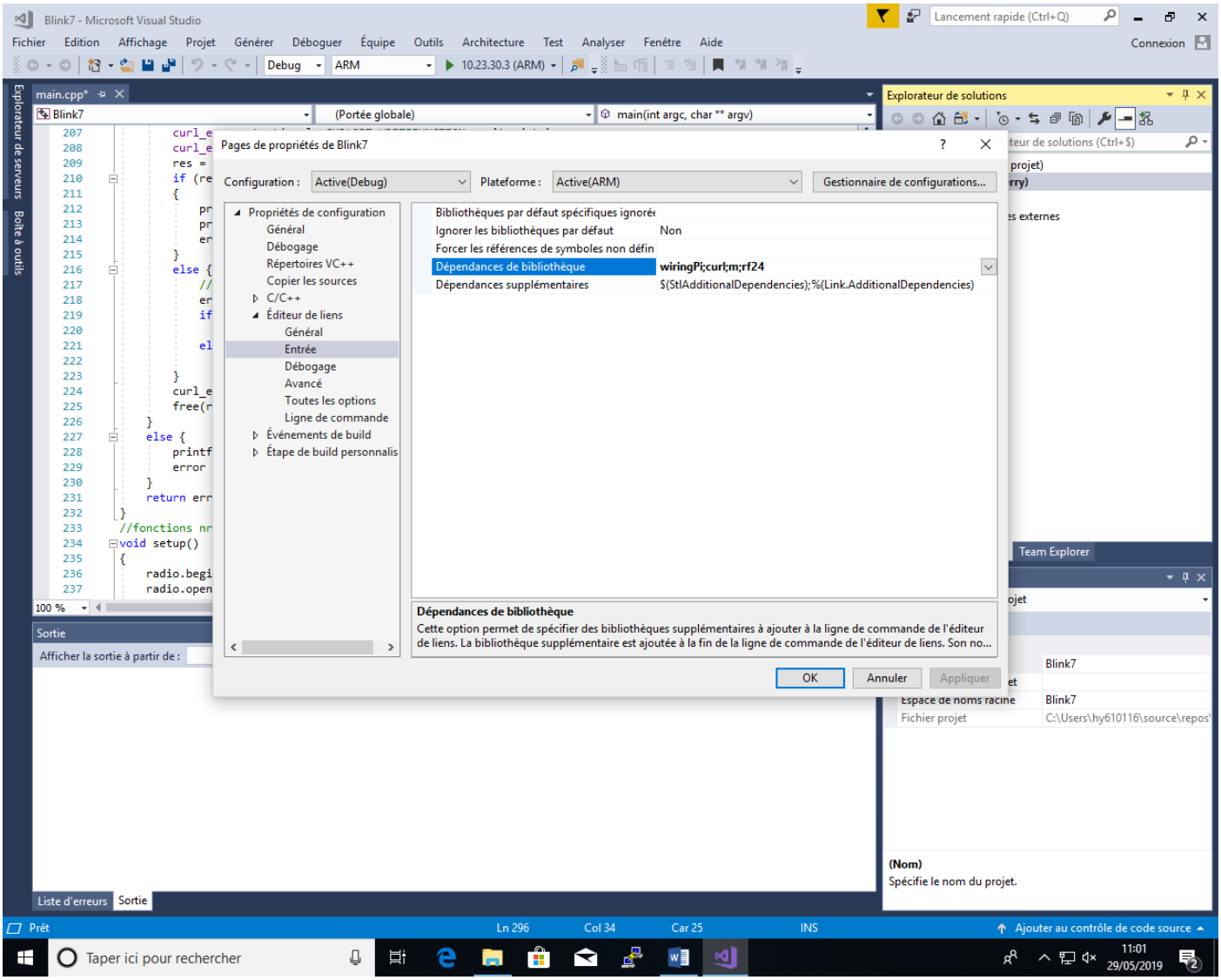
Nous allons maintenant à l'aide de ce que l'on va recevoir du nrf connecté au banana pi, transmettre les données à Thingspeak avec notre arduino.
Les bibiliothéques à indiquer aux logiciels sont :

-Bmp180.h
-rf24.h
-wiringPi
-m
-curl

Clic droit propritétes sur votre projet.



Dans éditeurs de Lien > Entrée > Dépendances de bibliothéque, rajouter toutes les bibliothéques sauf bmp180.h qui est une bibliothéque que nous avons créer nous même.



Pour qu'il détecte la bibliothéque bmp180.h, nous devons la rajouter au dossier ou le logiciel cherche pour appliquer les bibliothéques.

Pour cela nous pouvons chercher ou se trouve la biblothéque wiringPi pa exemple, et ensuite copier notre fichier bmp180.h dans ce dossier.

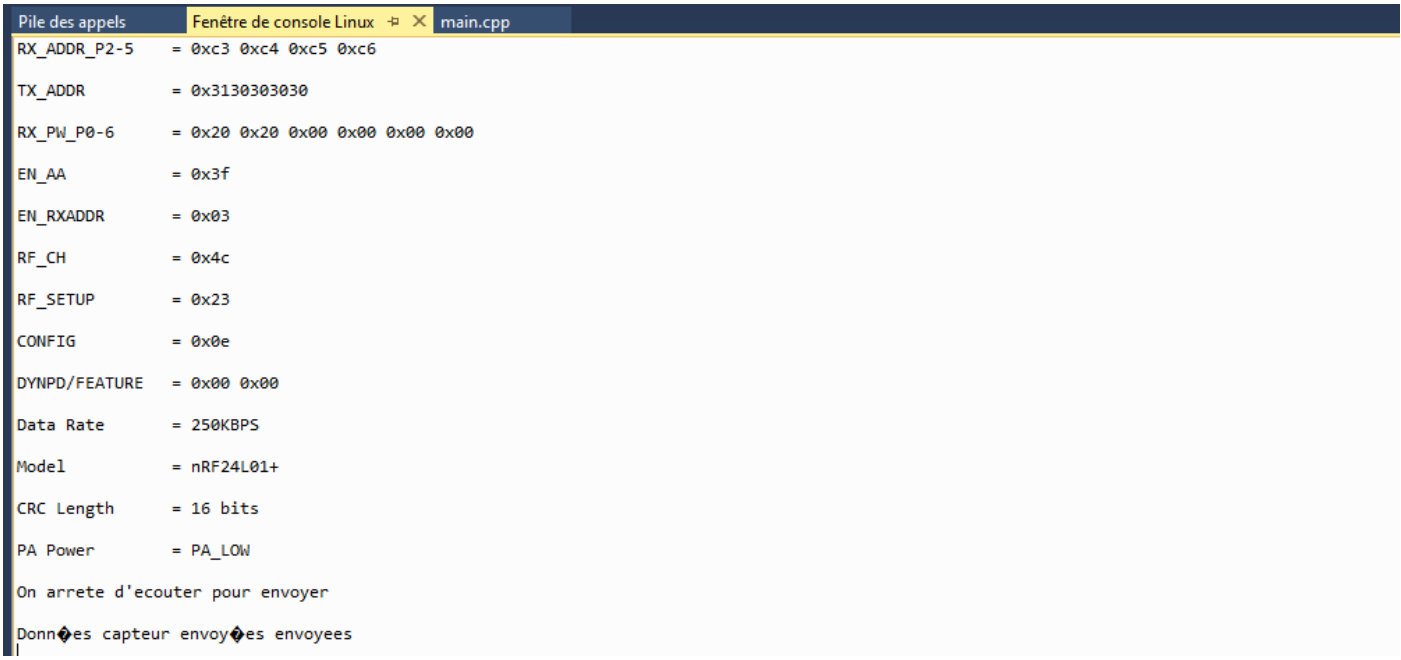
locate wiringPi

```
/usr/local/include/wiringPi.h
```

Vous pouvez ensuite copier le bmp180.h dans ce dossier :

```
sudo cp /repousetrouvevotrebmp /usr/local/include
```

Vous pouvez maintenant compilez tout vos programmes sur visual studio, ce qui vous simplifie la tâche.



Maintenant que nous avons envoyer des données de capteur du banana vers la carte, ce qui a en soi peu d'importance pour un projet IOT, sachant que la carte arduino ne peut communiquer sur internet.

Nous allons faire l'inverse, en envoyant des données de l'arduino vers le banana pi, et le banana pi se chargera de poster les données sur Internet.

Nous allons pour cela envoyer les données envoyées sur le premier TP avec l'esp.

Exercice final :

Côté arduino : (sur vs code)

- Comme avec l'esp generé un incrementeur et un nombre aléatoire à chaque fin de boucle
- Créer une valeur id qui prendre la valeur de votre table
- Envoyer l'id de votre table, votre nombre aleatoire, votre incrementeur toutes les 16 secondes dans un paquet (chercher sur internet comment faire)
- Lorsque vous n'envoyez rien, mettre votre MCU et votre nrf en sleep à l'aide des fonctions dans la bibliothéque Lowpower et nrf(faire une boucle for sur cette fonction pour attendre 16 secondes)

Côté Banana pi : (sur vs community /nano)

- Lire les données du capteur bmp180
- Recevoir les données du nrf
- Allumer la led toutes les fois que l'on recoit la donnée
- Si l'id que vous avez recu corrspond à l'id de votre table : envoyez les données de l'incrementeur et du nombre aléatoire, ainsi que les données du capteur.

Correction :

Côté arduino : https://gist.github.com/Mizaystom/8fd04e438b8a0b47fe885289ca3e10f5

Côté banana : https://gist.github.com/Mizaystom/84ad5e143fbea79ecbfc1121f44b03c9

Résultats attendu :

```
Message de l'arduino :Ouverture des Pipes

STATUS           = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1     = 0x3030303030 0x3130303030
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0x3030303030
RX_PW_P0-6       = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA            = 0x3f
EN_RXADDR        = 0x03
RF_CH            = 0x4c
RF_SETUP         = 0x23
CONFIG           = 0x0e
DYNPD/FEATURE    = 0x00 0x00
Data Rate        = 250KBPS
Model            = nRF24L01+
CRC Length       = 16 bits
PA Power         = PA_LOW
message send to banana pi:

260.00


23.00


57.00
```

```
STATUS            = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7 TX_FULL=0
RX_ADDR_P0-1      = 0x3130303030 0x3030303030
RX_ADDR_P2-5      = 0xc3 0xc4 0xc5 0xc6
TX_ADDR           = 0x3130303030
RX_PW_P0-6        = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA             = 0x3f
EN_RXADDR         = 0x02
RF_CH             = 0x4c
RF_SETUP          = 0x23
CONFIG            = 0x0e
DYNPD/FEATURE     = 0x00 0x00
Data Rate         = 250KBPS
Model             = nRF24L01+
CRC Length        = 16 bits
PA Power          = PA_LOW
en attente de reception
Donnees recues
id=260.000000 nmb1 =28.000000 nmb2=16.000000
 reception de la table = 260.000000
Temperature : 31.20 C
Pressure :    1007.59 Pa
Altitude :    46.56 h
UpdateThingSpeak> send: [https://api.thingspeak.com/update?api_key=3F88
5=1007.000000]RESPONSE OK from ThingSpeak
```

(A partir de 16 :10, les données precedentes étaient buguées)