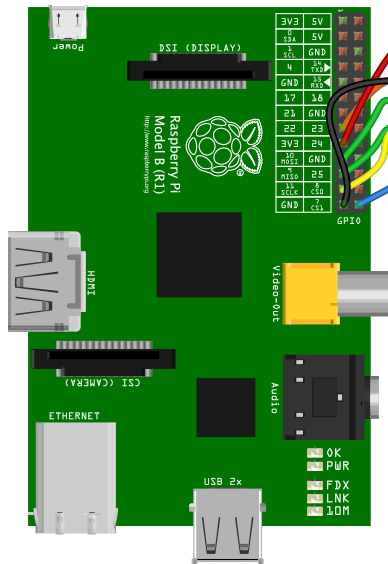
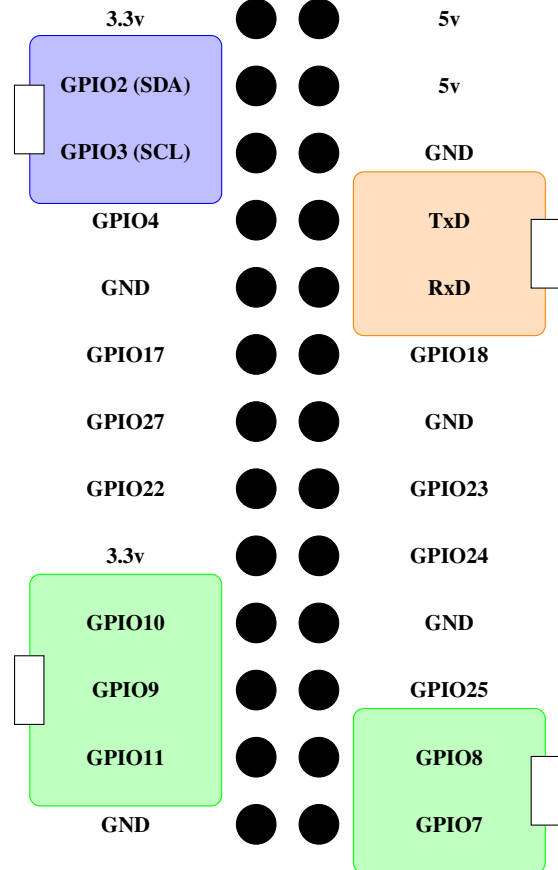


Communication dans un MANet basé sur des Raspberry Pi & radio nRF24L01

■ ■ ■ Connexion du composant nRF24L01 au Raspberry Pi

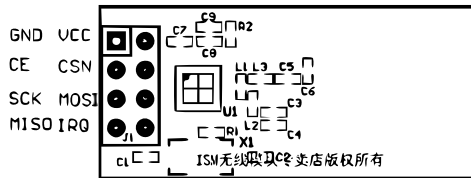


Les broches « GPIOs » du Raspberry Pi



fritzing

Version non sérigraphiée du nRF24L01 :



Vous devez localiser la broche entourée d'un carré.

Les correspondances entre broches du Raspberry Pi et du nRF24L01 :

nRF24L01	Raspberry Pi	Usage
IRQ	GPIO17	non utilisée
CE	GPIO22	8 ^{ème} en partant du haut
VCC	3.3v	juste en dessous
MO	GPIO10	juste en dessous
MI	GPIO9	juste en dessous
SCK	GPIO11	juste en dessous
GND	GND	juste en dessous
CSN	GPIO7	juste à côté

Activer le bus SPI sur le Raspberry PI

```
xterm
$ sudo raspi-config
```

Sélectionner l'option n°9 « *Advanced Options* » :

```
xterm
Raspberry Pi Software Configuration Tool (raspi-config)

 1 Expand Filesystem           Ensures that all of the SD card s
 2 Change User Password        Change password for the default u
 3 Boot Options                 Choose whether to boot into a des
 4 Wait for Network at Boot    Choose whether to wait for networ
 5 Internationalisation Options Set up language and regional sett
 6 Enable Camera               Enable this Pi to work with the R
 7 Add to Rastrack             Add this Pi to the online Raspber
 8 Overclock                   Configure overclocking for your P
 9 Advanced Options            Configure advanced settings
 0 About raspi-config          Information about this configurat

                <Select>                <Finish>
```

puis les options « *A6 SPI* », « *enable SPI* », « *load kernel module by default* », « *Finish* » et « *Reboot* » :

```
xterm
Raspberry Pi Software Configuration Tool (raspi-config)

 A1 Overscan                   You may need to configure oversca  ↑
 A2 Hostname                   Set the visible name for this Pi
 A3 Memory Split               Change the amount of memory made
 A4 SSH                        Enable/Disable remote command lin
 A5 Device Tree                Enable/Disable the use of Device
 A6 SPI                        Enable/Disable automatic loading
 A7 I2C                        Enable/Disable automatic loading
 A8 Serial                     Enable/Disable shell and kernel m
 A9 Audio                       Force audio out through HDMI or 3
 AA GL Driver                   Enable/Disable experimental desk  ↓

                <Select>                <Back>
```

Pour mettre à jour le firmware du Raspberry, ③, et son système d'exploitation, ① et ② :

```
pi@raspberrypi
① $ sudo apt-get update
② $ sudo apt-get upgrade
③ $ sudo rpi-update
$ sudo reboot
```

■■■■ Configuration logicielle pour le projet

Installation de la bibliothèque de gestion de composant nRF24L01 :

```
xterm
$ git clone https://git.p-fb.net/pef/RF24.git
$ cd RF24/RF24
$ sudo make install
$ cd examples
$ make
$ sudo ./gettingstarted
```

Pour l'installation de la bibliothèque « PyCrypto » :

```
xterm
$ sudo apt-get install python-dev
$ sudo apt-get install python-setuptools
$ sudo easy_install pip
$ sudo pip install pycrypto
```

■ ■ ■ Création d'un programme de « communication » entre Raspberry PI & nRF24L01

Nous allons écrire un programme, `send_receive`, de dialogue entre deux terminaux mobiles où chaque terminal pourra communiquer avec l'autre de **manière alternée** suivant un protocole préétabli.

Nous pouvons concevoir le programme de la manière suivante :

1. le composant radio nRF24L01 est configuré :

- ◊ choix du canal de communication ;

Canal	Frequence (Mhz)	Description
0 à 82	2400 à 2482	autorisé mais bruité (conflits avec les réseaux WiFi Bluetooth etc.)
83 à 99	2483 à 2499	non autorisé pour les mobiles
100	2500	restreint
101 à 119	2501 à 2519	autorisé
120 à 125	2520 à 2525	restreint

En cas d'utilisation de 2Mbps, la transmission utilise 2 canaux simultanément.

- ◊ de la puissance d'émission ;
- ◊ du nombre de tentative de réémission en cas d'échec ;
- ◊ des adresses source et destination (pour la bibliothèque RF24, le lien est un « pipe ») ;

```
// Setup for GPIO 22 CE and CE1 CSN with SPI Speed @ 8Mhz
RF24 radio(RPI_V2_GPIO_P1_15, RPI_V2_GPIO_P1_26, BCM2835_SPI_SPEED_8MHZ);

// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

int main(int argc, char** argv)
{
    // Setup and configure rf radio
    radio.begin();
    radio.enableDynamicPayloads();
    radio.setRetries( 5,15);
    radio.setChannel(0x4c);
    radio.setPALevel(RF24_PA_LOW);
    // To transmit
    radio.openWritingPipe(pipes[0]);
    radio.openReadingPipe(1,pipes[1]);
```

2. le programme effectue une boucle sur l'entrée clavier pour lire une commande de l'utilisateur :

- ◊ commande « 1 », pour l'envoi :
 - * bascule du composant radio nRF24L01 en mode envoi ;
 - * envoi du message et gestion des acquittements/réémissions automatiques grâce au mode « *Shockburst™* » du composant ;
- ◊ commande « 2 », pour la réception :
 - * passage en mode d'attente de réception ;
 - * réception d'un message et écriture du contenu sur le canal non bufferisé `stderr`.

```
radio.startListening();
radio.read(message, sizeof(char)*32);
fprintf(stderr, "%s", message);
```

■ ■ ■ Encapsulation du « send_receive » dans Python

Il est possible de contrôler le programme de communication conçu précédemment dans un programme Python pour lui permettre de contrôler les messages en entrée comme en sortie du module radio, grâce au code suivant :

```
#!/usr/bin/python

import subprocess, os, pty

p = subprocess.Popen(["./send_receive"], stderr=subprocess.PIPE, stdin=subprocess.PIPE, bufsize=0)
message="hello"
# Envoi de la variable message
p.stdin.write("1\n")
p.stdin.write(message+'\n')
# reception d'un message
message = ''
p.stdin.write("2\n")
message = p.stderr.readline()
print message.rstrip('\n')
```

On utilise le canal de sortie d'erreur qui est non bufferisé.

étudiez et répondez aux questions suivantes :

- a. le canal par défaut est indiqué par la valeur $0 \times 4c$, à quelle fréquence correspond-t-il ?
- b. l'utilisation du canal de communication : partage ou non entre tous les terminaux présents dans la salle de TP ?
- c. l'usage des adresses d'émission et de réception des messages ;
 - ◊ la bande passante et la vitesse de transmission en rapport avec le taux d'erreur et la sensibilité ?
 - ◊ possibilité de broadcast ?
 - ◊ gestion des collisions ?
- d. La gestion des bascule « mode émetteur »/« mode récepteur » : Est-elle nécessaire ? Est-elle immédiate ?
- e. les possibilités de routage entre raspberry :
 - ◊ réseau « *single hop* » vs « *multi-hop* » ;
 - ◊ portée radio/découverte des voisins/relais des échanges.

Vous vous aiderez de la documentation du composant nRF24L01 accessible à http://p-fb.net/fileadmin/_migrated/content_uploads/nRF24L01_Product_Specification_v2_0.pdf

2 – Vous écrirez le code du programme C ou C++, « send_receive » réalisant :

- ▷ la lecture de la commande (1 pour envoi, 2 pour la réception)
- ▷ la lecture du message dans le cas d'un envoi et son envoi par le nRF24L01 ;
- ▷ la réception et l'affichage sur `stderr` du message dans le cas de la réception.

3 – Vous écrirez un programme Python permettant de faire des échanges sécurisés :

- a. vous définirez un format de message (les messages sont limitées à 32 octets) contenant :
 - ◊ le nom de l'expéditeur : 4 octets ;
 - ◊ le nom du destinataire : 4 octets ;
 - ◊ le contenu du message sur au plus 16 octets ;
- b. vous testerez l'envoi et la réception de vos messages avec le format indiqué en combinaison du programme Python et du « send_receive ».
- c. le contenu du message est **chiffré** maintenant.

Exemple de chiffrement :

- ◊ limitation des messages à 16 octets à cause du codage en base64 (taille augmentée de 30%)
 - ◊ avec une opération AES en mode CTR, « counter », utilisant un secret connu des deux interlocuteurs ;
- Le mode CTR permet de rendre le chiffrement dépendant du numéro de ce compteur : pour une même clé le chiffrement sera différent pour chaque valeur du compteur.*

Attention : pour permettre l'échange du message entre le programme Python et le « chat », votre message devra être encodé en base64 :

```

xterm
>>> import base64
>>> a=base64.encodestring('toto')
>>> a
'dG90bw==\n'
>>> base64.decodestring(a)
'toto'
    
```

Exemple d'utilisation du chiffrement par block AES en mode compteur :

```

#!/usr/bin/python
from Crypto.Cipher import AES

# le compteur doit etre donne sur 16 octets au travers d'une fonction (ou 24 ou 32)
# le secret doit faire 16 octets
cipher=AES.new('ma cle super secrete d'au moins 16 caracteres'[:16],
               AES.MODE_CTR, counter=lambda : "0"*16)
# le chiffre est une sequence de 16 octets
chiffre=cipher.encrypt("A"*16)
    
```