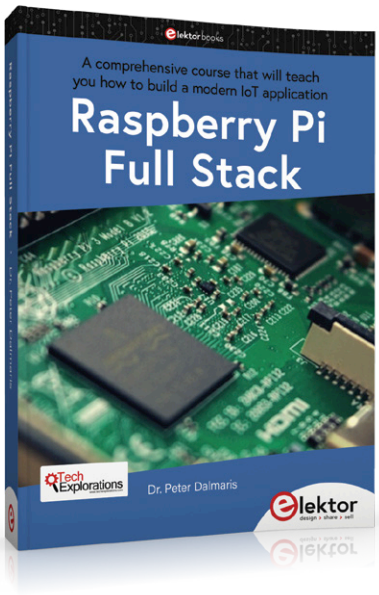
Raspberry Pi Full Stack

RPi et RF24 au cœur d'un réseau de capteurs



Peter Dalmaris (Australie)

Cet extrait d'un livre d'Elektor propose en condensé deux chapitres du livre de Peter Dalmaris *Raspberry Pi Full Stack*, publié récemment par Elektor. Destiné aux utilisateurs de RPi curieux et désireux d'associer matériel et logiciel dans leur voyage initiatique, le potentiel éducatif de ce livre est riche. Voici comment y est décrite en détail la combinaison d'un RPi avec un module RF24 pour lire des capteurs distants intégrés dans un petit réseau.

Nœud de base du réseau RF24, le Raspberry Pi traite les valeurs des capteurs qu'il reçoit du nœud Arduino (ou des nœuds, s'il y en a plusieurs). Le nœud Arduino est décrit dans les chapitres précédents du livre, non repris ici. Ce chapitre explique la connexion du module RF24 au Raspberry Pi. Vous pouvez la réaliser sur une carte d'essai ou utiliser mon module HAT spécialement conçu à cet effet.

J'ai réalisé mon module RF24 Raspberry Pi en utilisant mon module HAT (**fig. 1**). Le HAT est équipé d'un module radio nRF24, d'un DHT22, d'un poussoir et de deux LED (témoins d'alimentation et d'activité). Les fichiers Gerber du HAT sont téléchargeables sur la page du projet [1]. Vous pouvez aussi commander ce circuit imprimé directement auprès de PCBWay [2].

La liste des connexions (**fig. 2**), et le schéma du circuit imprimé (**fig. 3**) vous permettront de réaliser un prototype de ce circuit sur plaque d'essai. Vous pouvez obtenir une version à haute résolution du schéma du HAT sur la page du projet [3].

La liste suivante contient tous les composants nécessaires pour réaliser le circuit de la fig. 2. La connexion de certains composants, dont le poussoir et le capteur DHT22, a déjà été décrite plus tôt dans le projet. Les seuls nouveaux composants sont l'émetteur-récepteur nRF24 et son condensateur de découplage.

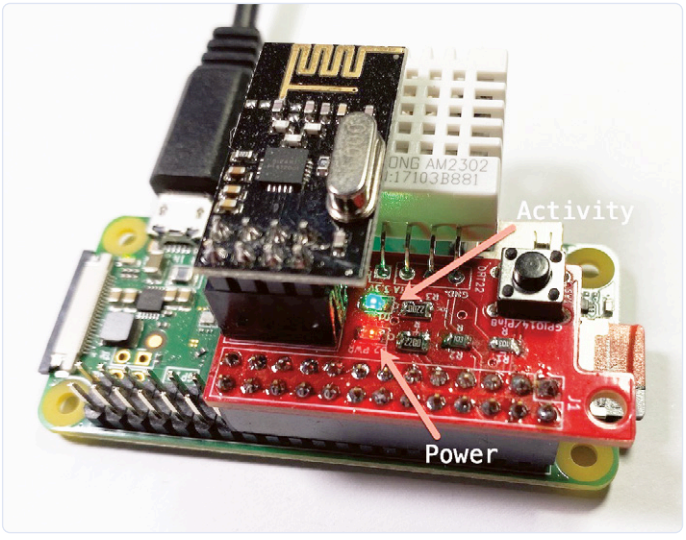


Figure 1. Le HAT installé sur un Raspberry Pi Zero W.

Raspberry Pi - RF24 - DHT22

Raspberry Pi	nRF24	DHT22	Comments
3.3V	Vcc	Pin 1 - Power	220uF elec. cap pin "+".
GND	GND	Pin 4- GND	220uF elec. cap pin "-".
GPIO 11	SCK		
GPIO 9	MISO		
GPIO 10	MOSI		
GPIO 8	CSN		
GPIO 7	CE		
GPIO4 (Pin 7)		Pin 2 - Data	Add a 10KOhm pull up resistor between Data and GND.

Raspberry Pi Full Stack | Tech Explorations

Figure 2. Les détails de connexion pour les modules nRF24 et DHT22.

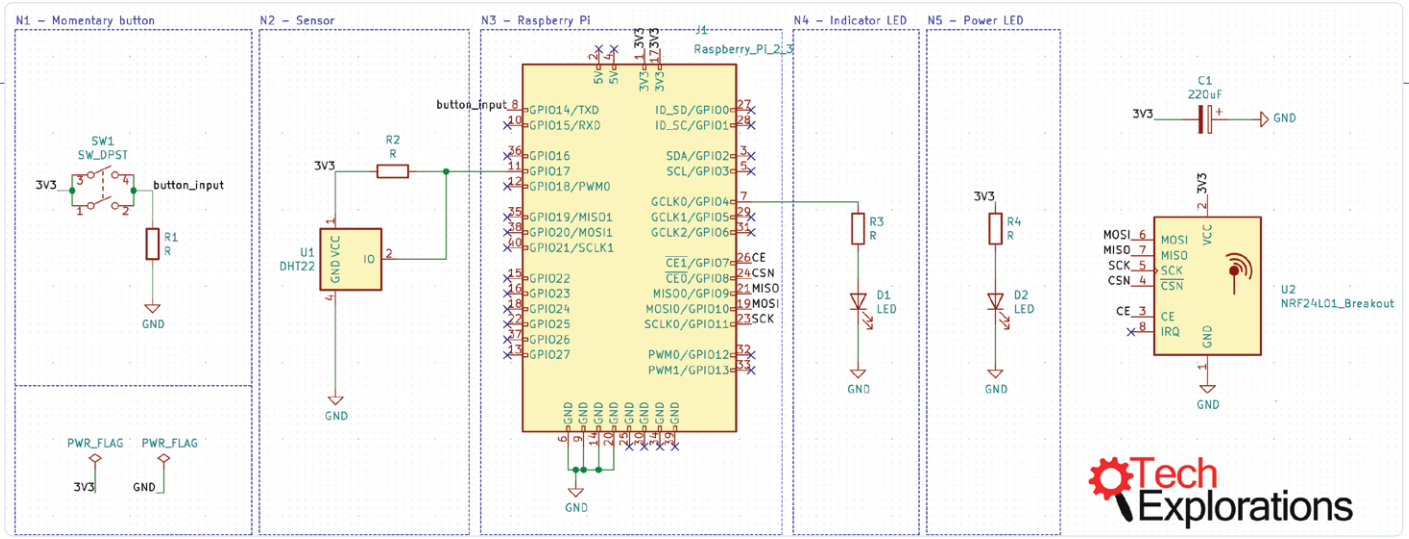


Figure 3. Le schéma du HAT RPi pour la carte d'accès au nRF24 et au DHT22.

Composants nécessaires :

- un capteur DHT22
- un module d'accès nRF24
- deux résistances de 10 kΩ
- deux résistances de 330 Ω
- une LED rouge (témoin d'alimentation)
- une LED bleue (témoin d'activité)
- un condensateur électrolytique de 220 µF ou équivalent

Le script récepteur du RPi nRF24

Ce chapitre présente la fonction du script Python récepteur qui tourne sur le RPi et prend en charge les communications nRF24. Ce script n'est pas prêt à l'emploi. Il dépend des pilotes en langage C de RF24 et RF24Network et des encapsulations Python dont la configuration est traitée dans le prochain chapitre. Pour l'instant, occupons-nous du script Python du récepteur. Le code source complet de ce script est disponible sur la page du projet [4]. J'ai écrit ce script pour qu'il puisse tourner en arrière-plan contrôlé par `systemd` (comme a été configuré le script d'application web plus tôt dans ce projet). Mais ce sera pour plus tard, car il faut d'abord s'assurer que le script s'exécute correctement en avant-plan.

La fonction `log_values()` est une copie presque identique de la même fonction dans le script `env_log.py`, déjà configuré pour suivre un programme horaire. Recevant les valeurs des capteurs comme arguments, elle les écrit dans la base de données locale et dans le tableau Google Sheet dans le nuage.

Ci-dessous, je passe en revue certains éléments du script, en particulier ceux qui se rapportent aux communications RPI-nRF24.

- Juste après la définition de la fonction `log_values()`, le script initialise le module nRF24 en commençant par la variable `radio` où il utilise le constructeur RF24. Ce dernier fait partie de la bibliothèque Python d'encapsulation de RF24 qui permet d'utiliser le pilote C de RF24 à partir du script Python. Vous pouvez vous familiariser avec cette fonction en étudiant le code source du pilote [5].
- Une fois l'objet radio créé et initialisé, le script crée l'objet `network` à l'aide du constructeur RF24Network. C'est cet objet qui permet au Raspberry Pi de recevoir un envoi du nœud Arduino. Le seul paramètre nécessaire pour créer l'objet réseau est l'objet `radio` créé sur la ligne précédente. Pour en savoir plus sur le constructeur du RF24Network, examinez le code source de RF24Network.h [6] à la ligne 373.
- À la ligne suivante, avec `octlit = lambda n : int(n, 8),` nous créons une fonction lambda de conversion décimal en octal, nécessaire car le pilote du réseau RF24 code les adresses de nœuds en octal. En Python, une fonction `lambda` [7] ressemble à une fonction normale (définie avec le mot-clé `def`) mais n'a pas de nom. On l'utilise pour l'évaluation d'une seule expression, comme c'est le cas ici : `octlit` reçoit un nombre décimal et renvoie l'équivalent octal en utilisant la fonction Python `int()` [8].
- Dans `this_node = octlit(«00»)`, nous utilisons `octlit` pour obtenir l'équivalent octal de `00`, et nous stockons cette valeur dans la variable `this_node`. Il s'agit de l'adresse réseau RF24 du Raspberry Pi.
- Dans les six prochaines lignes, jusqu'au bloc `while`, le script va :
  - démarrer la radio RF24 ;
  - attendre 0,1 seconde que la radio soit prête ;
  - démarrer le réseau sur le canal 90 ;
  - afficher la configuration de la radio et du réseau sur la console ;
  - remettre à zéro le compteur `packets_sent` ;
  - et remettre à zéro le compteur `last_sent`.
- Maintenant que la radio et le réseau sont prêts, le script entre dans une boucle infinie d'attente d'une transition d'un nœud Arduino.
- Au début de chaque boucle, il appelle la méthode `update()` [9] de l'objet réseau pour guetter vos messages.
- S'il n'y a pas de nouveau message, le script s'endort pendant 1 seconde, puis relance la boucle.
- S'il y a un nouveau message, le script utilise la fonction `read()` [10] pour lire les 12 octets (la charge utile) du message, et les passer à la variable `payload`. La fonction read récupère aussi l'en-tête du message et le transmet à la variable `header`. Rappelez-vous que dans le sketch Arduino, nous avons créé une charge utile de 12 octets de la forme : `51.23,23.09`.
- Sur les deux lignes suivantes, j'utilise `print` pour envoyer la charge utile vers la console. Cela m'a servi dans mon travail de décodage de la charge utile en chiffres dans le bloc `try` qui suit.
- La variable `payload` contient une chaîne de caractères de la forme : `51.23,23.09`. Le script utilise `decode()` [11] pour la coder en UTF-8, et la fonction `split()` [12] pour séparer les

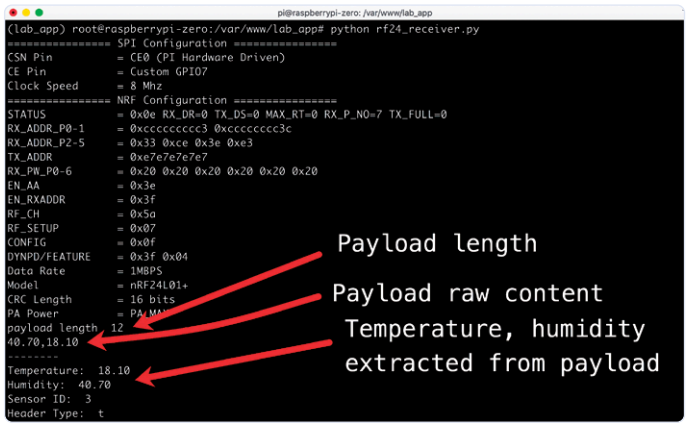


Figure 4. Exemple de sortie du script du récepteur RF24.

valeurs de température et d'humidité au niveau de la virgule et les stocker dans la variable `values` (un tableau de chaînes de caractères).

- Dans le bloc `try`, le script utilise la fonction `float()` [13] pour extraire les nombres stockés dans la charge utile et les convertir en nombre à virgule flottante. La fonction `float()` reçoit une chaîne de caractères, et si elle peut la convertir en un nombre, elle le renvoie.
- Avec `float(values[1][0:5])`, le script prend les 5 premiers caractères de la chaîne stockée dans l'index 1 du tableau `values`, et utilise la fonction `float()` pour les convertir en un nombre et le ranger dans la variable `temperature`.
- L'humidité subit le même traitement en utilisant `float(values[0][0:5])`
- Si l'une de ces deux opérations échoue, le bloc `try` se termine et un message d'erreur est envoyé à la console. Une conversion peut échouer si la charge utile n'est pas formatée correctement par l'Arduino, ou si elle est altérée pendant l'émission ou la réception. Des interférences peuvent provoquer une telle altération.

Prenez tout le temps nécessaire pour vous familiariser avec ce code. Lorsque vous êtes prêt, copiez-le dans votre répertoire applicatif. Utilisez Vim pour créer un nouveau fichier nommé «rf24_receiver.py». Copiez le code dans le tampon de `Vim` depuis le dépôt du projet [14].

Avant de pouvoir tester les communications RF24, vous devez compiler les pilotes en langage C et les encapsulations Python de RF24 et RF24Network. Vous le ferez dans le prochain chapitre. En attendant, voici à quoi ressemble l'exécution du script étudié (**fig. 4**). J'ai annoté cette capture d'écran pour que vous puissiez voir les impressions intégrées dans le script. Passons au chapitre suivant avec la configuration des pilotes RF24 et RF24Network.
Une fois que vous avez assemblé le circuit, passez au chapitre suivant pour mettre en place le script Python de gestion des communications nRF24.

Comment installer les modules Python nRF24 sur le RPi

Dans ce chapitre, nous allons voir comment compiler et installer les pilotes en langage C et les encapsulations Python nécessaires au module nRF24. Deux modules sont concernés : le module principal RF24 et le sous-module RF24Network. Il a plusieurs variantes du projet RF24 ; celle que vous allez utiliser est gérée par TMRh20 [15]. Elle est optimisée pour le module RPi.
Vous pouvez obtenir le code source des deux modules à partir de leurs dépôts Github respectifs :

- Dépôt RF24 : https://github.com/nRF24/RF24 ;
- Dépôt RF24Network : https://github.com/nRF24/RF24Network ;

Pour chaque module, le processus d'installation comprend ces étapes :

- télécharger le code source du module à partir de Github ;
- compiler et installer le pilote C ;
- compiler et installer l'encapsulation Python.

Avant de commencer, assurez-vous que l'interface SPI de votre RPi est activée. Le module nRF24 communique avec le RPI par SPI. Pour le vérifier (et activer le SPI si ce n'est pas déjà fait), connectez-vous à votre RPi et saisissez la commande suivante :

```
sudo raspi-config
```

Utilisez les flèches du clavier et la touche Entrée pour accéder aux options d'interface, puis à SPI. Activez SPI si nécessaire, et quittez raspi-config. Continuez en mettant à jour Raspbian :

```
sudo apt-get update
sudo apt-get upgrade
```

Vous devez télécharger le code source des modules dans votre répertoire de travail. Vous devez également compiler les encapsulations Python avec l'environnement virtuel Python de votre application activé afin qu'elles soient disponibles pour les scripts Python de votre application.
Préparez votre travail avec ces commandes :

```
$ sudo su
# cd /var/www/lab_app/
# . bin/activate
# mkdir rf24libs
# cd rf24libs
```

La compilation et l'installation des deux modules peuvent commencer. Il va y avoir beaucoup de texte affiché sur la console. Mes commandes sont en gras.

Compilation et installation de RF24

Créez un clone du code source de RF24, à partir du git à https://github.com/tmrh20/RF24.

```
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs# git clone
https://github.com/tmrh20/RF24.git RF24
Cloning into 'RF24'...
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (46/46), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 3545 (delta 16), reused 15 (delta 4),
   pack-reused 3499
Receiving objects: 100% (3545/3545), 1.54 MiB |
   679.00 KiB/s, done.
Resolving deltas: 100% (2119/2119), done.
```

Dans votre répertoire de code source, il y a maintenant un nouveau répertoire : «RF24».

```
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs# ls -al
total 12
drwxr-xr-x 3 root root 4096 May 13 01:38 .
drwxr-xr-x 14 root root 4096 May 13 01:39 ..
drwxr-xr-x 9 root root 4096 May 13 01:38 RF24
```

Passez sous le répertoire RF24 et installez le pilote :

```
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs# cd RF24/
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24# make install
[Running configure]
[SECTION] Detecting arm compilation environment.
[OK] arm-linux-gnueabihf-gcc detected.
[OK] arm-linux-gnueabihf-g++ detected.
… (omitted output)…
[Installing Libs to /usr/local/lib]
[Installing Headers to /usr/local/include/RF24]
```

Le pilote C est installé. Passons à l'encapsulation Python. Cette opération peut prendre plusieurs minutes sur le RPi Zero W, plus lent.

```
(lab_app) root@rpi4:/var/www/lab_app/rf24libs/RF24/
   pyRF24# cd pyRF24/
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24/pyRF24#
python setup.py install
running install
running bdist_egg
running egg_info
creating RF24.egg-info
… (omitted output)…
Installed /var/www/lab_app/lib/python3.8/
   site-packages/RF24-1.3.4-py3.8-
linux-armv6l.egg
Processing dependencies for RF24==1.3.4
Finished processing dependencies for RF24==1.3.4
```

Le module RF24 est installé, et l'encapsulation Python est prête à être utilisée. Poursuivons avec RF24Network.

```
RF24Network
```

Retournez à la racine du répertoire rf23libs et créez un clone du code source de RF24Network, depuis https://github.com/nRF24/RF24Network.git.

```
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24/pyRF24# cd
../..
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs# git clone
https://github.com/nRF24/RF24Network.git
Cloning into 'RF24Network'...
remote: Enumerating objects: 2249, done.
remote: Total 2249 (delta 0), reused 0 (delta 0),
   pack-reused 2249
Receiving objects: 100% (2249/2249), 1.60 MiB |
   733.00 KiB/s, done.
Resolving deltas: 100% (1342/1342), done.
```

L'opération de clonage a créé un nouveau répertoire, «RF24Network». Passez sous le nouveau répertoire et compilez le code source :

```
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs# cd RF24Network/
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24Network# make
install
g++ -Wall -fPIC -c RF24Network.cpp
g++ -shared -Wl,-soname,librf24network.so.1 -o
   librf24network.so.1.0
RF24Network.o -lrf24-bcm
[Install]
[Installing Headers]
```

Le pilote est installé. Continuez avec l'encapsulation Python pour le réseau RF24.

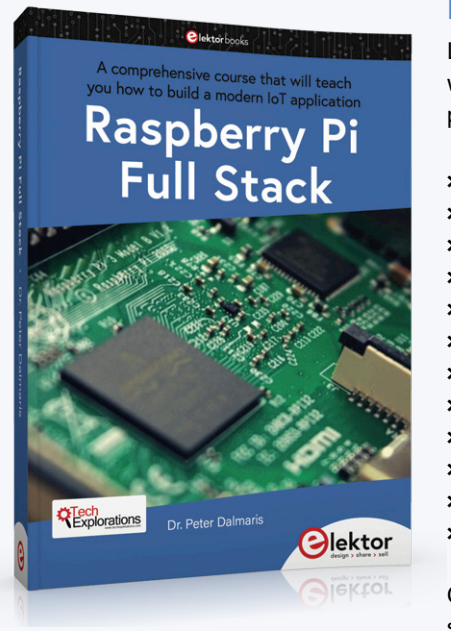
```
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24Network# cd
../RF24/pyRF24/pyRF24Network/
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24/pyRF24/
pyRF24Network# python setup.py install
running install
running build
running build_ext
building 'RF24Network' extension
… (omitted output)…
running install_egg_info
Removing /var/www/lab_app/lib/python3.8/
   site-packages/
RF24Network-1.0-py3.8.egg-info
Writing /var/www/lab_app/lib/python3.8/site-packages/
RF24Network-1.0-py3.8.egg-info
```

Le réseau RF24 en Python est installé. Testons-le.

Test

La façon la plus simple de vérifier que les modules RF24 et RF24Network fonctionnent correctement est d'exécuter les programmes d'exemple Python livrés avec le code source. Vous trouverez ces exemples dans le répertoire `examples` dans `pyRF24Network`. Exécutez l'exemple 'rx' comme ceci :

```
(lab_app)root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24/pyRF24/
pyRF24Network# cd examples/
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24/pyRF24/
pyRF24Network/examples# ls
helloworld_rx.py helloworld_tx.py
(lab_app) root@raspberrypi-zero:/var/www/lab_app/
   rf24libs/RF24/pyRF24/
pyRF24Network/examples# python helloworld_rx.py
================ SPI Configuration ================
CSN Pin = CE0 (PI Hardware Driven)
CE Pin = Custom GPIO22
Clock Speed = 8 Mhz
================ NRF Configuration ================
STATUS = 0x0e RX_DR=0 TX_DS=0 MAX_RT=0 RX_P_NO=7
```



LIVRE RASPBERRY PI FULL STACK

Le livre *Raspberry Pi Full Stack* est un tour d'horizon du développement d'applications web sur le Raspberry Pi. Il montre comment réaliser une application de A à Z, et vous permet d'acquérir expérience et savoir-faire techniques sur des sujets tels que :

- le système d'exploitation Linux et son langage de commande
- le langage de programmation Python
- les broches d'entrée-sortie à usage général (GPIO) du RPi
- le serveur web Nginx
- le micro framework d'applications web Flask Python
- JQuery et CSS pour la création d'interfaces utilisateurs
- la gestion des fuseaux horaires
- créer des tableaux avec Plotly et Google Charts
- enregistrement des données avec Google Sheet
- développer des applets avec IFTTT
- sécuriser votre application avec SSL
- recevoir des notifications par SMS sur votre ordiphone en utilisant Twilio

Ce livre vous apprendra également comment installer un nœud de capteurs Arduino sans fil et en recueillir les données. Votre application web RPi sera capable de traiter les données de nœuds Arduino de la même manière qu'elle traite les données de son capteur embarqué.

Avec le livre *Raspberry Pi Full Stack* vous acquerrez de nombreuses compétences essentielles à la réalisation d'applications Web et Internet des objets. L'application que vous construirez dans le cadre de ce projet est une plateforme que vous pouvez étendre. Ce n'est qu'un début de ce que vous pouvez faire avec un RPi et les composants logiciels et matériels utilisés. L'auteur offre aux lecteurs de son livre un espace de discussion en ligne.

- Livre imprimé : www.elektor.fr/raspberry-pi-full-stack
- E-book : www.elektor.fr/raspberry-pi-full-stack-e-book

LIENS

[1] **Fichiers Gerber pour le HAT RF24** : https://techexplorations.com/parts/rpifs-parts/

[2] **HAT PCB (sans les composants) de PCBWay** : www.pcbway.com/project/shareproject/Raspberry_Pi_Full_Stack_RF24_and_DHT22_HAT.html

[3] **Version haute résolution du schéma** : https://techexplorations.com/parts/rpifs-parts/

[4] **Script rf24_receiver.py** : https://github.com/futureshocked/RaspberryPiFullStack_Raspbian/blob/master/rf24_receiver.py

[5] **Voir la ligne 137 de RF24.h** : https://github.com/nRF24/RF24/blob/master/RF24.h#L137

[6] **Voir la ligne 373 de RF24Network.h** : https://github.com/nRF24/RF24Network/blob/master/RF24Network.h#L373

[7] **Pour en savoir plus sur les « expressions lambda »** : https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions

[8] **Pour en savoir plus sur 'int()'** : https://docs.python.org/3/library/functions.html#int

[9] **Informations complémentaires** : https://github.com/nRF24/RF24Network/blob/master/RF24Network.h#L413

[10] **Informations complémentaires** : https://github.com/nRF24/RF24Network/blob/master/RF24Network.h#L467

[11] **Pour en savoir plus sur 'decode()'** : https://docs.python.org/3/library/stdtypes.html#bytes.decode

[12] **Pour en savoir plus sur 'split()'** : https://docs.python.org/3/library/stdtypes.html#str.split

[13] **Pour en savoir plus sur 'float()'** : https://docs.python.org/3/library/functions.html#float

[14] **Mise à jour de rf24_receiver.py** : https://github.com/futureshocked/RaspberryPiFullStack_Raspbian/blob/master/rf24_receiver.py

[15] **La page du projet** : https://tmrh20.github.io/RF24/

```
   TX_FULL=0
RX_ADDR_P0-1 = 0xcccccccccc3 0xcccccccc3c
RX_ADDR_P2-5 = 0x33 0xce 0x3e 0xe3
TX_ADDR = 0xe7e7e7e7e7
RX_PW_P0-6 = 0x20 0x200x200x200x200x20
EN_AA = 0x3e
EN_RXADDR = 0x3f
RF_CH = 0x5a
RF_SETUP = 0x07
CONFIG = 0x0f
DYNPD/FEATURE = 0x3f 0x04
Data Rate = 1MBPS
Model = nRF24L01+
CRC Length = 16 bits
PA Power = PA_MAX
```

Notez que la sortie contient des statistiques du module RF24 au moment de son démarrage. Elle contient des valeurs valides pour le RX, une adresse (c'est-à-dire des valeurs non nulles) ainsi que le reste de la configuration nRF. Si les modules RF24 et RF24Network n'étaient pas installés correctement, ou si le RPi ne pouvait pas communiquer avec le module nRF24 via SPI, vous obtiendriez soit un message d'erreur du programme exemple (il ne démarrerait pas du tout), soit des valeurs de configuration vides. Maintenant que les modules nRF24 sont installés, le script Python que vous avez installé va pouvoir fonctionner.

200517-03

L'auteur

Peter Dalmaris est enseignant, ingénieur et passionné d'électronique. Il a donné plusieurs cours en ligne sur l'électronique et écrit plusieurs livres techniques. En tant que *Chief Tech Explorer* depuis 2013 au sein de *Tech Explorations*, la société qu'il a fondée à Sydney, en Australie, Peter s'est donné pour mission d'explorer la technologie et de contribuer à la diffusion du savoir dans le monde.

Ont contribué à cet article

Auteur : **Peter Dalmaris**
Illustrations : **Peter Dalmaris**
Rédaction : **Jan Buiting**
Maquette : **Giel Dols**
Traduction : **Helmut Müller**

Votre avis, s'il vous plaît ?

Vous pouvez envoyer un courriel à l'auteur peter@txplore.com avec vos questions ou vos commentaires sur le livre.