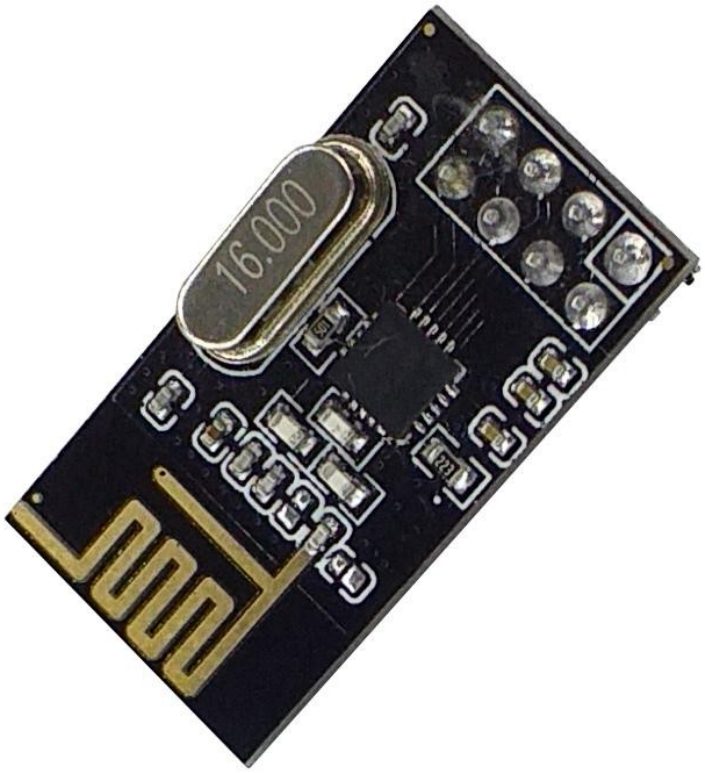
Bienvenue !

Merci beaucoup d'avoir acheté notre module nRF24L01 AZ-Delivery. Dans les pages suivantes, nous vous présenterons comment utiliser et configurer cet appareil pratique.

Amusez-vous bien !



Sommaire

De nombreux projets nécessitent une communication entre deux appareils. Dans la plupart des cas, les communications se font par câbles. Certains modules de communication sont capables de communiquer entre eux sans fil. Ainsi, avec ces modules de communication sans fil, nous sommes en mesure de créer un réseau sans fil de dispositifs envoyant des données d'un bout à l'autre. Ces modules sont principalement utilisés dans les appareils qui surveillent les données des capteurs, contrôlent les robots, la domotique, etc.

Le *nRF24L01* est l'un de ces modules sans fil. Il utilise des fréquences radio pour communiquer avec un autre module *nRF24L01*, une communication bidirectionnelle. Le *nRF24L01* est super bon marché, et très petit en taille, ce qui vous permet de l'interfacer dans presque tous vos projets.

Le module *nRF24L01* fonctionne dans la bande de fréquence *ISM* de *2,4GHz* et utilise la modulation *GFSK* pour la transmission de données, les taux de transfert de données étant l'un des suivants : *250kbps*, *1Mbps* et *2Mbps.*

La bande *2,4 GHz* est l'une des bandes industrielles, scientifiques et médicales *(ISM)*, réservée au niveau national à l'utilisation d'appareils de faible puissance sans licence. Les téléphones mobiles, les dispositifs *Bluetooth*, les dispositifs de communication en champ proche *(NFC)* ou les réseaux informatiques sans fil *(WiFis)* sont des exemples de ces dispositifs qui utilisent tous les fréquences *ISM.*

Caractéristiques

Gamme de fréquences :	2.4GHz, bande ISM
Taux maximum de données aériennes :	2Mb/s
Format de modulation :	GFSK
Puissance de sortie maximale :	0dBm
Tension d'alimentation en fonctionnement :	1,9V à 3,6V
Courant de fonctionnement maximal :	13.5mA
Min. Courant (mode veille) :	26µA
Entrées logiques :	Tolérance de 5V
Gamme de communication :	100+ mètres (ligne de vue)

La gamme de tension de fonctionnement du module est de 1,9V à 3,6V. Les broches logiques sont tolérantes à 5V, nous pouvons donc facilement le connecter à un Atmega328p ou à tout microcontrôleur à logique 5V sans utiliser de convertisseur de niveau logique.

Le module prend en charge la puissance de sortie programmable dans l'une des gammes suivantes : *0dBm, -6dBm, -12dBm ou -18dBm.*

Le module consomme environ *12mA* pendant la transmission à *0dBm*, ce qui est encore plus faible qu'une simple *LED*. En mode veille, il consomme 26µA et en mode réduction de puissance *900nA*. C'est pourquoi il s'agit presque du meilleur dispositif sans fil pour les applications à faible consommation.

Le module utilise une antenne embarquée. Cela permet d'avoir une carte plus compacte du module. Cependant, l'antenne plus petite signifie également une portée de transmission plus faible, qui est d'environ *100m*, pour notre version du module. Ces *100m* sont la portée de transmission du module à l'extérieur dans un espace ouvert. La portée à l'intérieur, notamment à travers les murs, sera légèrement affaiblie.

Interface SPI

Le module utilise l'interface *SPI* pour communiquer avec les microcontrôleurs. Cela signifie que le module communique sur *4* fils avec un débit de données maximum de *10Mbps.* Tous les paramètres tels que le canal de fréquence *(125 canaux sélectionnables)*, la puissance de sortie *(0dBm, -6dBm, -12dBm ou -18dBm),* et le débit de données *(250kbps, 1Mbps, ou 2Mbps)* peuvent être configurés par le microcontrôleur via l'interface *SPI*.

Le bus *SPI* utilise le concept d'un *Master* et de *Slaves*, dans les applications les plus courantes où notre Atmega328p est le *Master* et le module *nRF24L01* est le *Slave*. Contrairement au bus *I2C*, le nombre de *Slaves* sur le bus *SPI* est limité, sur l'Atmegaa328p vous pouvez utiliser un maximum de deux *Slaves SPI,* c'est-à-dire deux modules *nRF24L01.*

Principe de fonctionnement

Le module transmet et reçoit des données sur une certaine fréquence appelée *channel*. Pour que deux modules émetteurs-récepteurs ou plus puissent communiquer entre eux, ils doivent être sur le même canal. Ce canal peut être n'importe quelle fréquence de la bande *ISM* de *2,4 GHz* ou, pour être plus précis, une fréquence comprise entre *2,400 GHz* et *2,525 GHz (soit 2 400 à 2 525 MHz).*

Chaque canal occupe une bande passante de moins de *1MHz.* Cela nous donne *125* canaux possibles avec un espacement de *1MHz*. Le module peut donc utiliser *125* canaux différents, ce qui permet d'avoir un réseau de *125* modems fonctionnant indépendamment en un seul endroit.

Un canal occupe une largeur de bande de moins de *1MHz* à *250kbps* et *1Mbps* de débit de données aériennes. Cependant, à un débit de *2Mbps*, une bande passante de *2MHz* est occupée (plus large que la résolution du réglage de la fréquence du canal *RF*). Ainsi, pour garantir l'absence de recouvrement des canaux et réduire la diaphonie en mode *2Mbps,* vous devez maintenir un espacement de *2MHz* entre deux canaux.

La fréquence du canal *RF* de votre canal sélectionné est réglée selon la formule suivante : *Freq (Sélectionné) = 2400 + CH (Sélectionné)*
Par exemple, si vous sélectionnez 108 comme canal pour la transmission de données, la fréquence du canal *RF* de votre canal sera de *2508MHz (2400 + 108).*

Réseau multi-récepteurs

Le module offre une fonction appelée *Multiceiver*. C'est une abréviation de *Multiple Transmitters Single Receiver*. Cela signifie que chaque canal *RF* est logiquement divisé en *6 canaux* de données parallèles appelés *Data Pipes*. En d'autres termes, un *Data Pipes* est un canal logique dans le canal *RF physique*. Chaque tuyau de données a sa propre adresse physique *(Data Pipe Address)* et peut être configuré.

Dans ce cas, le récepteur primaire fait office de récepteur central et collecte simultanément les informations provenant de six nœuds d'émission différents. Le récepteur central peut arrêter d'écouter à tout moment et agir comme un émetteur. Mais cela ne peut se faire que pour un tuyau/noeud à la fois.

Protocole ShockBurst amélioré

Ce module utilise une structure de paquets connue sous le nom de *Enhanced ShockBurst.* Cette structure de paquet simple est décomposée en 5 champs différents :

» Preamble

» Address

» Packet Control Field (PCF)

- payload length
- packet ID
- no acknowledge

» Payload

» Cyclic Redundancy Check (CRC)

La structure originale de *ShockBurst* ne comprenait que les champs préambule, adresse, charge utile et le contrôle de redondance cyclique *(CRC)*. Le *ShockBurst* amélioré a apporté une plus grande fonctionnalité pour des communications améliorées en utilisant un champ de contrôle de paquet *(PCF)* nouvellement introduit.

Cette nouvelle structure est intéressante à plus d'un titre. Premièrement, elle permet des charges utiles de longueur variable avec un spécificateur de longueur de charge utile, ce qui signifie que les charges utiles peuvent varier de *1 à 32 octets*. Deuxièmement, elle fournit à chaque paquet envoyé un *packet ID*, ce qui permet au dispositif récepteur de déterminer si un message est nouveau ou s'il a été retransmis (et peut donc être ignoré). Enfin, et surtout, chaque message peut demander l'envoi d'un accusé de réception lorsqu'il est reçu par un autre dispositif.

Traitement automatique des paquets

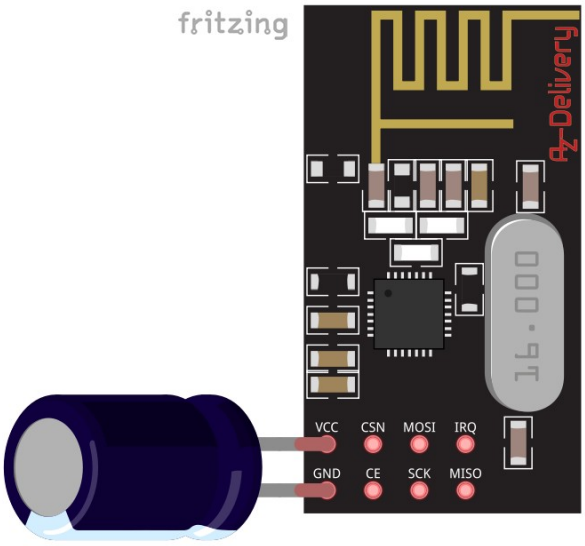
Expliquons trois scénarios afin de mieux comprendre comment deux modules transigent entre eux.
D'abord une transaction avec reconnaissance et interruption : l'émetteur commence une communication en envoyant un paquet de données au récepteur. Une fois que le paquet entier est transmis, l'émetteur attend *(environ 130µs)* que le paquet d'accusé de réception *(paquet ACK)* soit renvoyé pour le récepteur. Lorsque le récepteur reçoit le paquet, il envoie un paquet *ACK* à l'émetteur. Lorsque l'émetteur reçoit le paquet *ACK*, il l'affirme en générant le signal d'interruption *(sur la broche IRQ)* pour indiquer que les nouvelles données sont disponibles.
Le deuxième est une transaction avec un paquet de données perdu : il s'agit d'un scénario négatif où une retransmission est nécessaire en raison de la perte du paquet transmis. Après la transmission du paquet, l'émetteur attend la réception du paquet *ACK*. Si l'émetteur ne le reçoit pas dans le délai *ARD (Auto-Retransmit-Delay)*, le paquet est retransmis. Lorsque le paquet retransmis est reçu par le récepteur, le paquet *ACK* est retransmis à l'émetteur qui, à son tour, génère une interruption au niveau de l'émetteur.
Troisièmement, une transaction avec accusé de réception perdu : Il s'agit encore d'un scénario négatif où une retransmission est nécessaire en raison de la perte du paquet *ACK.* Ici, même si le récepteur reçoit le paquet à la première tentative, en raison de la perte du paquet *ACK,* l'émetteur pense que le récepteur n'a pas reçu le paquet du tout. Ainsi, une fois le délai de retransmission automatique écoulé, il retransmet le paquet. Maintenant, lorsque le récepteur reçoit le paquet contenant le même *packet ID* que le précédent, il le rejette et envoie à nouveau un paquet *ACK*.
L'ensemble du traitement des paquets est effectué automatiquement par la puce nRF24L01 sans intervention du microcontrôleur.

Améliorer la portée du module

Un paramètre clé pour tout système de communication sans fil est la portée de communication. Dans de nombreux cas, c'est le principal facteur de choix d'une solution RF. Voici quelques exemples de la manière dont nous pouvons améliorer la portée de nos modules.

1. Réduire le bruit de l'alimentation électrique

Tout *circuit RF* qui génère un signal de radiofréquence *(RF)* est très sensible au bruit de l'alimentation électrique. Si le bruit de l'alimentation n'est pas contrôlé, il peut réduire considérablement la portée que vous pouvez obtenir. À moins que la source d'alimentation ne soit une batterie autonome, il y a de fortes chances qu'il y ait du bruit associé à la génération de l'alimentation. Pour empêcher ce bruit de pénétrer dans le système, il est conseillé de placer un condensateur de filtrage de *10µF* entre les broches *VCC* et *GND*, et aussi près physiquement que possible du module *nRF24L01*, comme sur l'image ci-dessous (attention à la polarité du condensateur !).

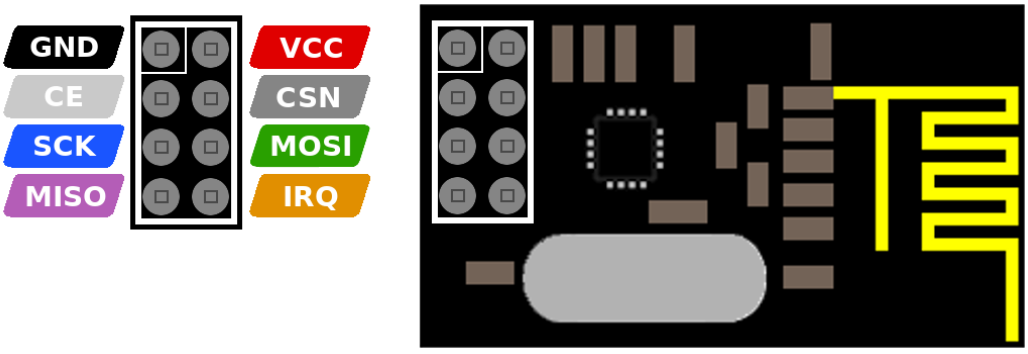


2. Changer la fréquence de votre canal
Une autre source potentielle de bruit pour un circuit *RF* est l'environnement extérieur, notamment si des réseaux voisins sont réglés sur le même canal ou si des interférences sont produites par d'autres appareils électroniques. Pour éviter que ces signaux ne causent des problèmes, nous vous suggérons d'utiliser les *25 canaux* les plus élevés de votre module. La raison en est que le *WiFi* utilise la plupart des canaux inférieurs.

3. Taux de données inférieur
Le module *nRF24L01* offre la sensibilité de réception la plus élevée à la vitesse de *250Kbps* qui est de *-94dBm*. Cependant, à un taux de données de *2MBps*, la sensibilité du récepteur tombe à *-82dBm*. Nous pouvons en conclure que le récepteur à *250Kbps* est presque *10* fois plus sensible qu'à *2Mbps.* Cela signifie que le récepteur peut décoder un signal qui est *10* fois plus faible. Que signifie la sensibilité du récepteur *(Rx)* ? La sensibilité du récepteur est le niveau de puissance le plus faible auquel le récepteur peut détecter un signal *RF*. Plus la valeur absolue du nombre négatif est grande, meilleure est la sensibilité du récepteur. Par exemple, une sensibilité de récepteur de *-94dBm* est meilleure de *12dB* qu'une sensibilité de récepteur de *-82dBm*. Ainsi, la réduction du débit de données peut améliorer considérablement la portée que vous pouvez atteindre. De plus, pour la plupart de vos projets, une vitesse de *250Kbps* est plus que suffisante.

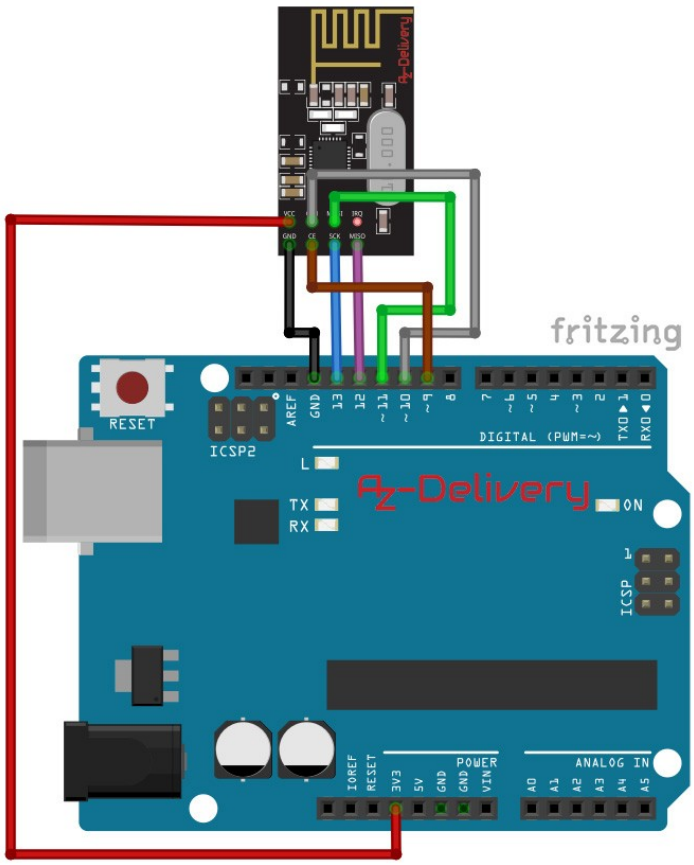
4. Puissance de sortie plus élevée
Le réglage de la puissance de sortie maximale peut également améliorer la portée de la communication. Le module *nRF24L01* vous permet de choisir une puissance de sortie parmi les valeurs suivantes : *0dBm (max), -6dBm, -12dBm ou -18dBm (min).* La sélection de la puissance de sortie de *0dBm* envoie un signal plus fort sur l'air.

Le brochage du nRF24L01



GND est la broche de terre. Elle est généralement marquée par le placement de la broche dans un carré afin qu'elle puisse être utilisée comme référence pour identifier les autres broches.
VCC fournit l'alimentation du module, de *1.9V* à *3.9V*. Vous pouvez le connecter à la sortie *3.3V* de votre Atmega328p.
CE *(Chip Enable)* est une broche active *HIGH*. Lorsqu'elle est sélectionnée, le *nRF24L01* transmet ou reçoit, selon le mode dans lequel il se trouve.
CSN *(Chip Select Not)* est une broche active *LOW* et est normalement maintenue *HIGH*. Lorsque cette broche passe au niveau *LOW*, le *nRF24L01* commence à écouter les données sur son port *SPI* et les traite en conséquence.
SCK *(Serial Clock)* accepte les impulsions d'horloge fournies par le maître du bus *SPI.*
MOSI *(Master Out Slave In)* est l'entrée *SPI* du *nRF24L01*.
MISO *(Master In Slave Out)* est la sortie *SPI* du *nRF24L01*.
IRQ est une broche d'interruption qui peut alerter le maître lorsque de nouvelles données sont disponibles pour être traitées.

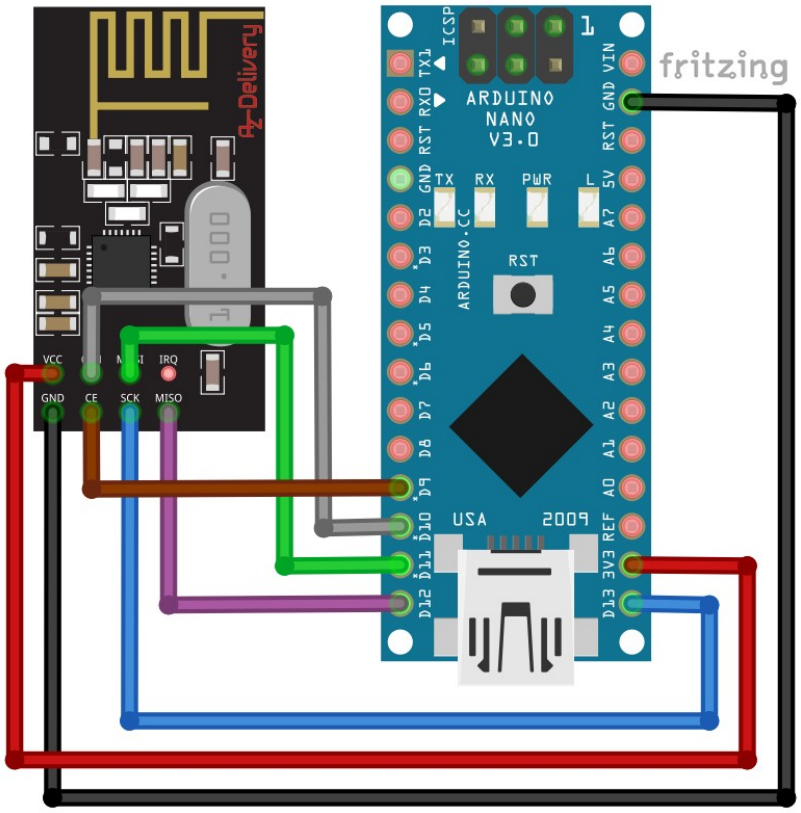
Rappelez-vous que connecter *VCC* à la broche *5V* détruira probablement votre module *nRF24L01* !



Pin nRF24L01		> Pin Mc	
VCC	>	3.3V	**Câble Rouge**
GND	>	GND	**Câble Noir**
CSN	>	D10	**Câble Gris**
CE	>	D9	**Câble Marron**
MOSI	>	D11	**Câble Vert**
SCK	>	D13	**Câble Bleu**
IRQ	>	pas connecté	
MISO	>	D12	**Câble Violet**

Les broches *CSN* et *CE* peuvent être connectées à n'importe quelle broche numérique de l'Atmega328p. Dans notre cas, elles sont connectées aux broches numériques *10* et *9* respectivement.

Connexion du module avec l'Nano V3.0



Pin nRF24L01 > Pin Nano V3.0

VCC	>	3V3	**Câble Rouge**
GND	>	GND	**Câble Noir**
CSN	>	D10	**Câble Gris**
CE	>	D9	**Câble Marron**
MOSI	>	D11	**Câble Vert**
SCK	>	D13	**Câble Bleu**
IRQ	>	pas connecté	
MISO	>	D12	**Câble Violet**

Les broches *CSN* et *CE* peuvent être connectées à n'importe quelle broche numérique de l'Atmega328p. Dans notre cas, elles sont connectées aux broches numériques 10 et 9 respectivement.

Bibliothèque pour l'IDE Arduino

Si vous n'avez pas d'*Arduino IDE*, allez sur le link, téléchargez-le et installez-le. L'installation est très simple, il suffit de suivre les instructions de l'assistant d'installation.
L'une des bibliothèques les plus populaires est *RF24*. Cette bibliothèque existe depuis plusieurs années. Elle est simple à utiliser pour les débutants, mais offre néanmoins beaucoup pour les utilisateurs avancés. Vous pouvez télécharger la dernière version de la bibliothèque RF24 GitHub. Lorsque vous la téléchargez, vous obtenez un fichier *zip*. Pour l'installer, ouvrez l'*IDE Arduino*, allez dans *Sketch* > *Include Library* > *Add .ZIP Library*, puis sélectionnez le fichier *zip* que vous venez de télécharger.

Nous devons réaliser deux de ces circuits, un pour l'émetteur et l'autre pour le récepteur. Le câblage des deux est identique.

Exemple de code simple pour un émetteur

Dans cet exemple, nous allons simplement envoyer un message "Hello World" de l'émetteur au récepteur. Voici le croquis de l'émetteur :

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(9, 10);  // create an RF24 object, CE, CSN
const byte address[6] = "00001";  // address
void setup() {
      radio.begin();
      radio.openWritingPipe(address);  // set the address
      radio.stopListening();  // set module as transmitter
}
void loop() {
      const char text[] = "Hello World";  // message
      radio.write(&text, sizeof(text));  // send message
delay(1000);
}
```

Le sketch commence par inclure les bibliothèques. La bibliothèque *SPI.h* gère la communication *SPI* tandis que *nRF24L01.h et RF24.h* contrôlent le module.

Ensuite, nous devons créer un objet *RF24*. L'objet prend comme paramètres deux numéros de broches auxquels sont connectés les signaux *CE* et *CSN.*
RF24 radio(CE, CSN) ;

Il faut ensuite créer un tableau d'octets qui représentera l'adresse du tuyau par lequel deux modules communiquent.
const byte address[6] = "00001";

On peut changer la valeur de cette adresse en n'importe quelle chaîne de *5* lettres comme *"node1"*. L'adresse est nécessaire si vous avez quelques modules dans un réseau. Grâce à l'adresse, vous pouvez choisir un module particulier avec lequel vous souhaitez communiquer, donc dans notre cas, nous aurons la même adresse pour l'émetteur et le récepteur.

Dans la fonction *setup*, nous devons initialiser l'objet radio en utilisant *radio.begin()* et en utilisant la fonction *radio.openWritingPipe(address)*. Nous définissons l'adresse de l'émetteur. On utilise la fonction *radio.stopListening()* qui définit le module comme émetteur.

Dans la section *loop*, nous créons un tableau de caractères auquel nous attribuons le message *"Hello World"*. À l'aide de la fonction *radio.write(),* nous allons envoyer ce message au récepteur. Le premier argument est le message que nous voulons envoyer. Le second argument est le nombre d'octets présents dans ce message : *radio.write(&text, sizeof(text)) ;*

Grâce à cette méthode, vous pouvez envoyer jusqu'à *32* octets à la fois. Car c'est la taille maximale d'un seul paquet que le *nRF24L01* peut gérer. Si vous avez besoin d'une confirmation que le récepteur a reçu les données, la méthode *radio.write()* renvoie une valeur bool. Si elle renvoie *TRUE*, les données ont atteint le récepteur. Si elle renvoie *FALSE*, les données ont été perdues.

Une chose dont vous devez vous souvenir, la fonction *radio.write()* bloque le programme jusqu'à ce qu'il reçoive la confirmation ou qu'il ait épuisé toutes les tentatives de retransmission.

Exemple de code simple pour le récepteur

Voici le croquis de notre récepteur :

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(9, 8);  // CE, CSN
const byte address[6] = "00001";  // address
void setup() {
      while(!Serial);
            Serial.begin(9600);
      radio.begin();
      radio.openReadingPipe(0, address); // set the address
      radio.startListening();  // set module as receiver
}
void loop() {
// Read the data if available in buffer
      if(radio.available()) {
            char text[32] = {0};
            radio.read(&text, sizeof(text));
            Serial.println(text);
      }
}
```

La plupart de ce sketch est identique à un sketch pour module émetteur-récepteur. Voici les différences :

Au début de la fonction de configuration, nous lançons la communication série. Nous utilisons ceci pour pouvoir montrer le message reçu dans le *Serial Monitor*.

Ensuite, en utilisant la fonction *radio.setReadingPipe()*, nous définissons la même adresse que l'émetteur et de cette façon, nous permettons la communication entre l'émetteur et le récepteur.
radio.openReadingPipe(0, address)

Le premier argument est le numéro du flux. Vous pouvez créer jusqu'à *6* streams qui répondent à des adresses différentes. Nous avons créé une seule adresse pour le stream numéro *0*. Le deuxième argument est l'adresse à laquelle le stream va réagir pour recueillir les données.

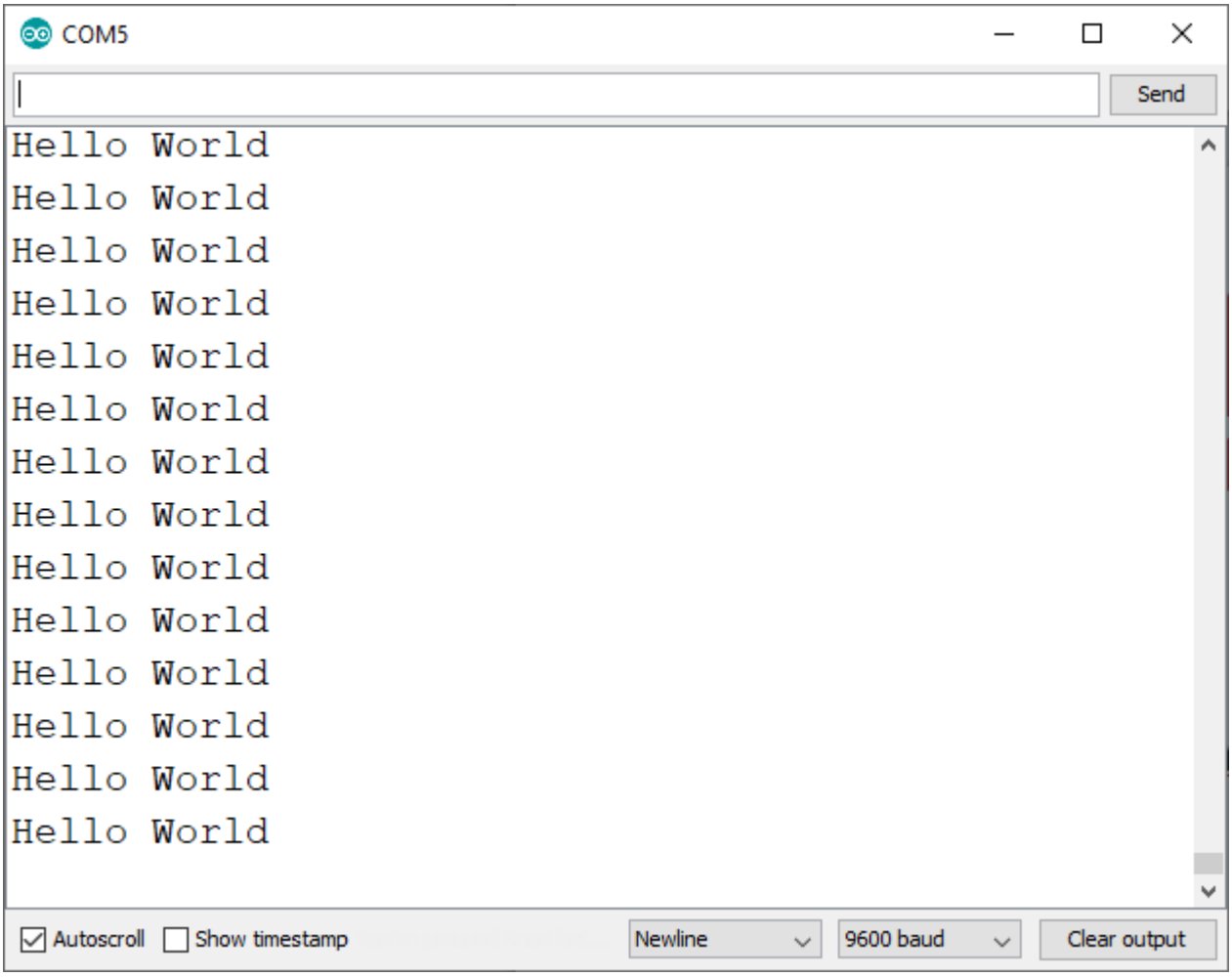
L'étape suivante consiste à définir le module comme un récepteur et à commencer à recevoir des données. Pour ce faire, nous utilisons la fonction *radio.startListening()*. À partir de ce moment, le modem attend les données envoyées à l'adresse spécifiée.

Dans la fonction *loop*, nous vérifions si des données sont arrivées à l'adresse en utilisant la méthode *radio.available()*. Cette méthode renvoie la valeur *TRUE* si des données sont disponibles dans le tampon.

```
if (radio.available()) {
    char text[32] = {0};
    radio.read(&text, sizeof(text));
    Serial.println(text);
}
```

Si les données sont reçues, alors il crée un tableau de *32* caractères rempli de zéros (plus tard le programme le remplira avec les données reçues). Pour lire les données, nous utilisons la méthode *radio.read(&text, sizeof(text))*. Ceci va stocker les données reçues dans notre tableau de caractères.

A la fin, nous imprimons simplement le message reçu dans *Serial Monitor*. Si vous avez tout fait correctement et il n'y a pas d'erreurs dans les connexions, vous devriez voir le message dans votre *Serial Monitor*, comme ceci :



Il existe trois autres paramètres que nous pouvons définir entre *radio.begin* et *radio.openReadingPipe (ou openWrittingPipe).* Dans nos exemples de croquis, ces paramètres ont une valeur par défaut.

La première est *radio.setChannel(value)*, où la *"valeur"* est le numéro du canal dans la gamme de *0* à *125*. Dans notre croquis, nous avons utilisé le canal *0*, dans la fonction *openReadingPipe(0, address).*

La deuxième est *radio.setPALevel(value)* où la *"valeur"* est une des valeurs suivantes :

» RF24_PA_MIN = -18dBm (default)

» RF24_PA_LOW = -12dBm

» RF24_PA_MED = -6dBm

» RF24_PA_HIGH = 0dBm

Avec cela, nous réglons le niveau de l'amplificateur de puissance utilisé par le module.

La troisième est *radio.setDataRate(value)*, où la *"valeur"* est l'une des valeurs suivantes :

» RF24_250KBPS for 250kbs (default)

» RF24_1MBPS for 1Mbps

» RF24_2MBPS for 2Mbps

Avec cela, nous établissons le taux de transmission des données.

C'est fait, vous pouvez maintenant utiliser votre module pour vos projets.

Il est maintenant temps d'apprendre et de réaliser les projets par vous-même. Vous pouvez le faire à l'aide de nombreux exemples de scripts et d'autres didacticiels, que vous trouverez sur Internet.

Si vous recherchez microélectronique et accessoires de haute qualité, AZ-Delivery Vertriebs GmbH est l'entreprise idéale pour vous les procurer. Vous recevrez de nombreux exemples d'application, des guides d'installation complets, des livres électroniques, des bibliothèques et l'assistance de nos experts techniques.

https://az-delivery.de

Amusez-vous !

Mentions légales

https://az-delivery.de/pages/about-us