Gros probléme d'emails avec Yahoo (/annonces/gros-probleme-demails-avec-yahoo/)

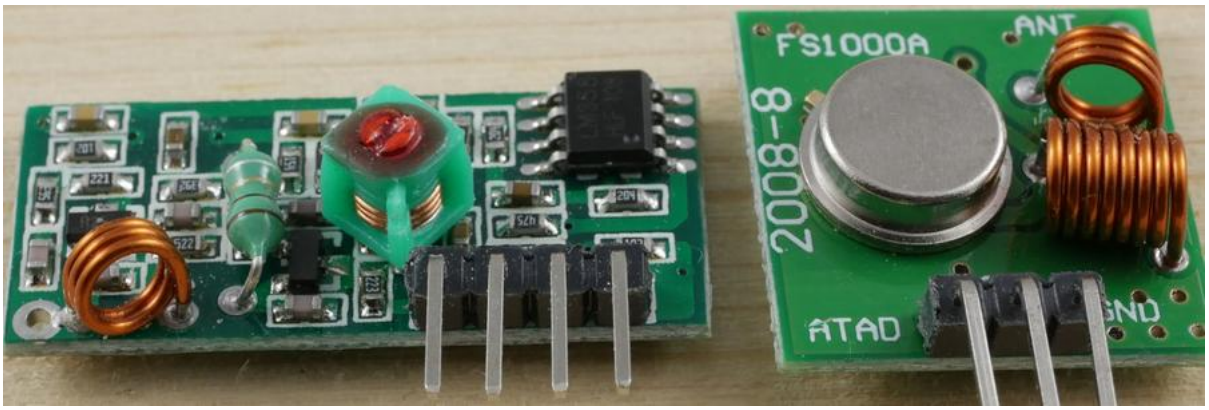
Yahoo refuse tous les emails du site. Si vous avez une adresse chez un autre prestataire, c'est le moment de l'utiliser 😉

En cas de soucis, n'hésitez pas à aller faire un tour sur la page de contact en bas de page.

par skywodd (/membres/skywodd/), le fév. 21, 2017 à 7:27 après-midi

Communiquer sans fil en 433MHz avec la bibliothèque VirtualWire et une carte Arduino / Genuino

Bip bip bip biiiiiip bip



par skywodd (/membres/skywodd/) | juin 11, 2016 | © Licence (voir pied de page)

Catégories : Tutoriels (/articles/categories/tutoriels/) Arduino (/articles/categories/tutoriels/arduino/) | Mots clefs : Arduino (/articles/tags/arduino/) Genuino (/articles/tags/genuino/) Sans fil (/articles/tags/sans-fil/) RF (/articles/tags/rf/) 433MHz (/articles/tags/433mhz/) VirtualWire (/articles/tags/virtualwire/)

Cet article n'a pas été mis à jour depuis un certain temps, son contenu n'est peut être plus d'actualité.

Dans un précédent article, je vous avais montré comment utiliser un module radio complet, riche en fonctionnalités. Dans ce tutoriel, nous allons voir plus petit et plus low-cost en utilisant de simples modules de communication radio 433MHz à quelques euros pièce. Nous verrons quand et comment utiliser ces modules et comment les mettre en oeuvre avec la bibliothèque Arduino VirtualWire. En bonus, nous verrons comment fabriquer une télécommande sans fil simpliste.

Sommaire

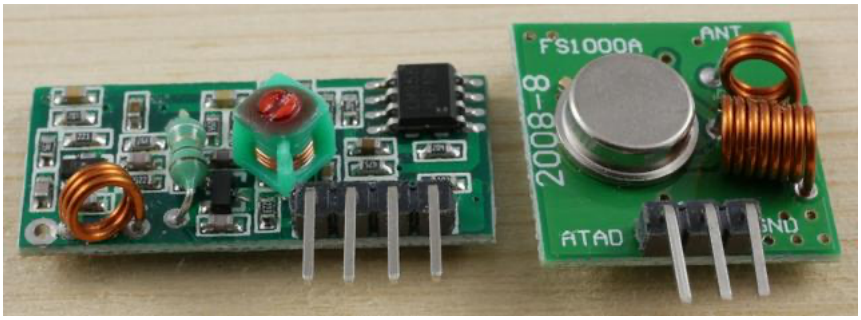
Bonjour à toutes et à tous !

Dans un précédent article (https://www.carnetdumaker.net/articles/communiquer-sans-fil-avec-un-module-nrf24l01-la-bibliotheque-mirf-et-une-c arduino-genuino/), je vous avais présenté un module radio complet comprenant énormément de fonctionnalités et disposant dans un même boitier émetteur, d'un récepteur et d'un circuit de traitement des communications.

Cependant, prenons un peu de recul et posons-nous une question toute simple : a-t-on vraiment besoin de communications bidirectionnelles avec accu réception ? En fait, pas vraiment. Cela dépend des projets et de l'environnement. Parfois, on peut tout simplement se passer de toutes ces fonctionnalités, très pratiques, mais coûteuses.

Dans cet article, nous allons-nous intéresser à un type de module radio "low cost" bien connu des amateurs d'Arduino : les modules radio 433MH accompagnement, nous verrons comment utiliser la bibliothèque VirtualWire avec ces modules.

Les modules radio 433MHz



(https://www.carnetdumaker.net/images/modules-radio-433mhz/)

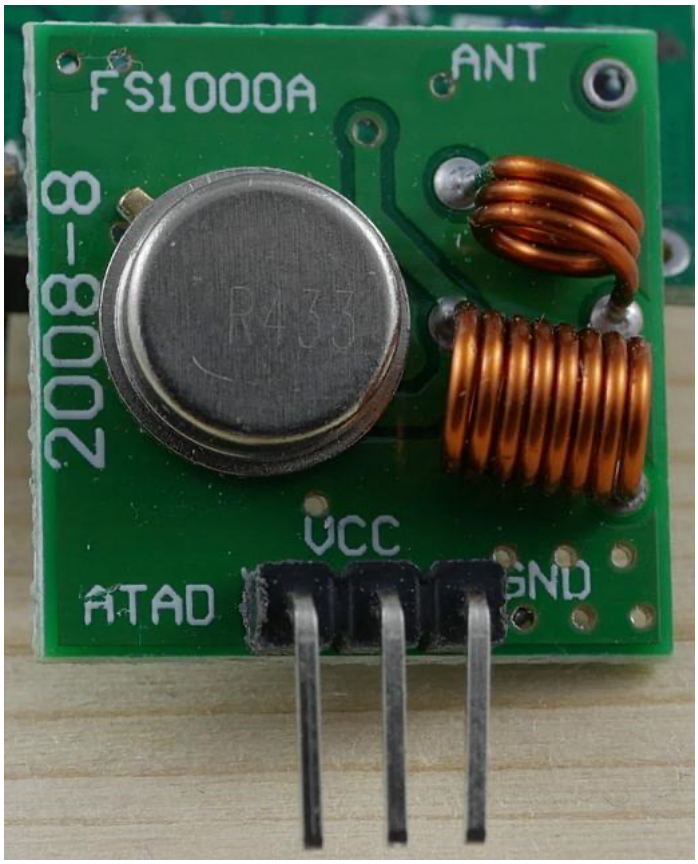
Modules radio 433MHz

Ces modules radio sont disponibles un peu partout sur internet pour des prix allant de quelques euros à une dizaine d'euros le lot de deux (un récepteur émetteur).

Ce sont des modules de transmission radio extrêmement simpliste. Il n'y a vraiment que le strict nécessaire sur ces modules, ce qui explique les prix ridicule faibles. Cependant, bien qu'extrêmement simplistes, ces modules peuvent être la solution idéale en fonction du projet.

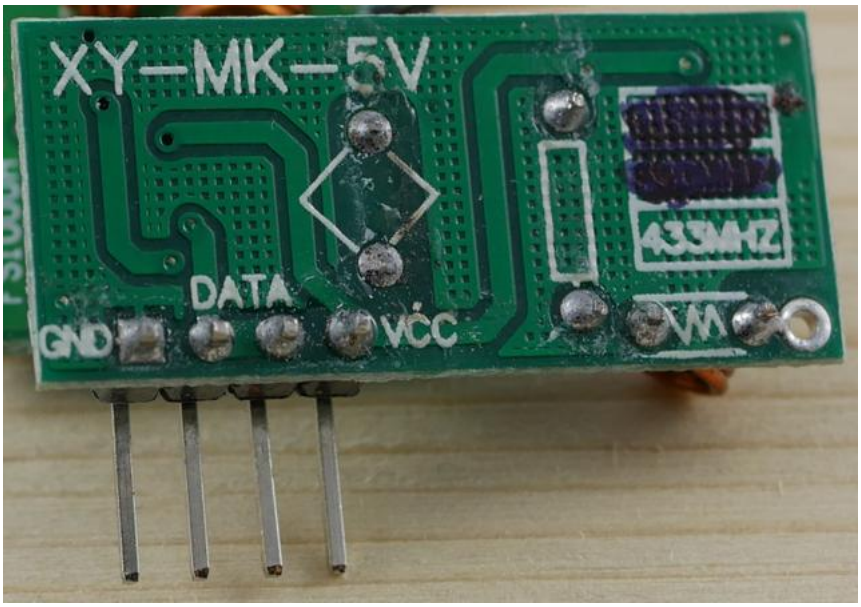
Ces modules radio sont disponibles en trois variantes : émetteurs (pour envoyer des messages), récepteurs (pour recevoir des messages) et émetteur-réce ("transceiver" en anglais, pour envoyer et recevoir des messages).

N.B. Les variantes émetteur-récepteur sont plus cher et disposent de plusieurs broches supplémentaires pour contrôler le mode de fonctionnement. Cet a détaillera uniquement l'utilisation des deux variantes de base : émetteurs seuls et récepteurs seuls. Cependant, il est bon de garder en tête que la version " en un" est tout à fait utilisable avec la bibliothèque VirtualWire.



(https://www.carnetdumaker.net/images/brochage-emetteur-433mhz/)

Brochage émetteur 433MHz



(https://www.carnetdumaker.net/images/brochage-recepteur-433mhz/)

Brochage récepteur 433MHz

Les modules radio "low cost" ont trois (parfois quatre) broches : `VCC` , `DATA` et `GND` , ainsi qu'une broche pour l'antenne. Ces modules sont généralement co pour fonctionner avec une alimentation de 5 volts sur la broche `VCC` .

Les signaux en entrée de la broche `DATA` de l'émetteur et en sortie de la broche `DATA` du récepteur sont de type logique TTL (tout ou rien, 0 volt ou 5 volts). a donc pas besoin d'ajouter de conversion de niveaux logiques entre ces modules radio et une carte Arduino "classique" fonctionnant en 5 volts.

PS Dans le cas d'une carte Arduino fonctionnant en 3.3 volts, un convertisseur de niveaux logiques est requis au niveau du récepteur pour ne pas endomm l'entrée logique de la carte Arduino.

Je tiens à remercier Alexandre (@Anderson69s (https://twitter.com/Anderson69s)) qui m'a dépanné en m'envoyant deux lots d'émetteur / récepte 433MHz pour cet article. Allez faire un tour sur son blog d'électronique / informatique (http://anderson69s.com/), ça lui fera plaisir 😉

Petite parenthèse technico-légale

Ceci est une parenthèse technico-légale, si vous achetez vos modules radio en France chez un revendeur de confiance, vous pouvez sauter ce chapitre. Gard juste en tête que la fréquence et la méthode de modulation du signal doivent être la même entre l'émetteur et le récepteur pour que ceux-ci puiss communiquer entre eux.

Il existe actuellement sur le marché une multitude de modules radio comme ceux en photo ci-dessus. Il est important de faire attention à deux chose fréquence de la porteuse et la modulation.

La fréquence de la porteuse correspond à la fréquence sur laquelle le module radio émet ou écoute. L'émetteur et le récepteur doivent fonctionner sur la m fréquence pour pouvoir communiquer correctement.

En France, seules trois bandes de fréquences "grand publiques" sont utilisables sans licence de radioamateur : les bandes (https://fr.wikipedia.org/wiki/Bande_industrielle,_scientifique_et_m%C3%A9dicale), qui correspondent aux fréquences autour de 2,4GHz (WiFi, Bluet modules radio grand public), 868MHz (clefs de voiture, alarmes de détection d'intrusion, modules de communication longues distances) et 433MHz (port garage, sonnettes sans fil, modules radio "low cost").

L'utilisation de ces bandes de fréquences est très réglementée pour que tout le monde puisse en profiter. Vous ne pouvez pas monter un émetteur 433MHz lo portée de 100W dans votre jardin. Si vous le faisiez, toutes les sonnettes, portes de garage, stations météo et autres modules sans fil dans un périmèt plusieurs kilomètres seraient mis hors services (dit comme ça, cela fait un peu scénario catastrophe).

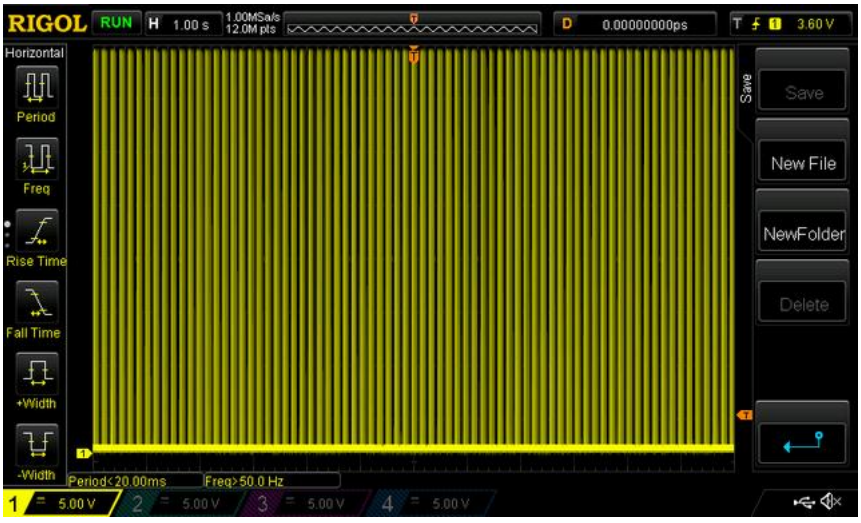
En 433MHz, le maximum théorique accepté est de 10mW. En 868MHz, le maximum théorique accepté varie entre 10mW, 25mW et 500mW en fonction fréquence exacte. Attention, ces puissances vont de pair avec des pourcentages maximums de temps d'utilisation de la fréquence. Je vous recommande de li article des radioamateurs de France (http://www.radioamateurs-france.org/?page_id=813) pour avoir plus de détails.

Pour ne pas avoir de problème, le plus important est de ne pas utiliser de modules radio conçus pour des gammes de fréquences réglementées en Europe internet, il est très (trop) simple d'acheter par erreur de module radio 315MHz ou 915MHz, conçus pour les États unis, ou pour d'autres pays où la législatio l'utilisation des gammes de fréquences est différente. À moins de vouloir voir la police sonner à votre porte, évitez d'utiliser un module radio sur une fréqu interdite.

De plus, pour respecter la réglementation en matière de temps d'utilisation de la fréquence, le plus simple est de ne pas émettre de message plus d'une fois les quarts-heures. Cependant, dans les faits, inutile de s'inquiéter avec ce genre de chose. Les modules radio vendus dans le commerce ont des porte seulement quelques mètres ou dizaines de mètres. Même si vous ne respectez pas les temps légaux d'utilisation de la fréquence, avec une portée si co personne ne sera inquiété (à par vous si vous avez plusieurs modules radio dans la même pièce).

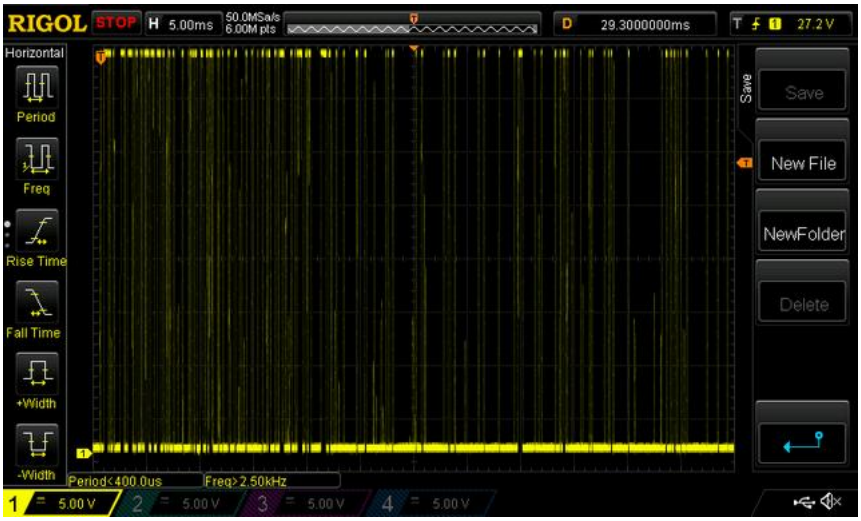
On s'en tamponne le coquillard de la réglementation !

Vous devez sûrement vous dire "mais pourquoi faire tant de blabla pour si peu ?". Et bien voici ce que reçoit un récepteur 433MHz modulation ASK sa aucun émetteur à proximité au sein de l'atelier du TamiaLab :



(https://www.carnetdumaker.net/images/capture-decran-doscilloscope-du-bruit-ambiant-sur-la-frequence-433mhz/)

Capture d'écran d'oscilloscope du bruit ambiant sur la fréquence 433MHz



(https://www.carnetdumaker.net/images/capture-decran-doscilloscope-du-bruit-ambiant-sur-la-frequence-433mhz-details/)

Capture d'écran d'oscilloscope du bruit ambiant sur la fréquence 433MHz (détails)

On a ici un bon exemple de module radio qui ne respecte ni la puissance d'émission réglementaire ni le temps d'utilisation de la fréquence. Heureusement signal radio est suffisant faible pour être couvert par mes propres modules radio. Seulement, leur portée est réduite par le bruit de fond sur la fréquence.

Gardez en tête que la bande de fréquence 433MHz est plus que saturée. Tout le monde l'utilise pour tout et n'importe quoi. Rien ne garantit que la giroue de la station météo de votre voisin ne va pas brouiller (littéralement) le signal de votre télécommande de porte de garage.

La bande de fréquence de 868MHz est beaucoup plus "propre", mais aussi beaucoup plus réglementée, car utilisée principalement pour les alarmes et systèmes de sécurité (clefs de voiture, etc.).

Et pour rendre les choses encore plus compliquées, en plus de la fréquence et de la puissance d'émission, il faut aussi faire attention à la méthode de modul utilisée par le module lors de l'achat.

Il existe une dizaine de méthodes de modulation (= manière de transmettre le signal radio). Les plus communes sont : `ASK`, `FSK`, `PSK` et `OOK`. Ce choix n' d'incidence légale, il faut simplement faire un choix et s'y tenir pour l'émetteur et le récepteur. La méthode de modulation la plus communément utilisée p modules radio "low cost" est la modulation ASK (https://en.wikipedia.org/wiki/Amplitude-shift_keying).

La bibliothèque VirtualWire

La bibliothèque VirtualWire permet d'envoyer et de recevoir des messages de moins de 77 octets (27 octets pour les versions avant 1.17) de manière simp efficace avec n'importe quels modules radio capables de transmettre un signal logique TTL (tout ou rien, haut ou bas). Cette bibliothèque est très communé utilisée en développement Arduino avec des modules radio "low cost" comme ceux présentés dans le chapitre précédent. Elle est aussi couramment utilisée des talkies-walkies, des modules radio haut de gamme et même des systèmes filaires pour des transmissions longues distances.

PS Pour ceux qui voudraient faire des tests sans module radio, la bibliothèque VirtualWire fonctionne même avec un simple fil entre l'émetteur et le réce (un fil de masse entre l'émetteur et le récepteur est dans ce cas obligatoire).

N.B. La bibliothèque VirtualWire implémente un protocole de communication propriétaire qui n'est pas compatible avec télécommandes sans fil du commerce, comme celles fournies avec les jouets radiocommandes, les portes de garage ou les stations m Chaque fabricant implémente son propre protocole de communication. La bibliothèque VirtualWire est uniquement conçue communiquer avec d'autres cartes électroniques utilisant la bibliothèque VirtualWire.

La bibliothèque VirtualWire est disponible sur http://www.airspayce.com/mikem/arduino/VirtualW (http://www.airspayce.com/mikem/arduino/VirtualWire/). Pour les utilisateurs de cartes Teensy (https://www.pjrc.com/teensy/index.html), une var spécialement conçue pour ces cartes est disponible sur le site de PJRC (https://www.pjrc.com/teensy/td_libs_VirtualWire.html).

PS La bibliothèque VirtualWire est compatible avec une grande quantité de cartes non Arduino, comme les cartes MSP430/Energia, les cartes Maple, les c Teensy et d'autres cartes à base de microcontrôleurs d'Atmel comme les ATtiny45 et ATtiny85.

ℹ La bibliothèque VirtualWire est en fin de vie

La bibliothèque VirtualWire vient tout juste d'être annoncée comme "en fin de vie". Cela signifie qu'elle ne recevra plus de mises à jour, de corrections bugs ou d'améliorations. Cependant, pas la peine de paniquer, la bibliothèque VirtualWire est très mature et ne demande plus vraiment de mises à jour.

L'auteur de la bibliothèque VirtualWire a fait le choix de migrer ses effort sur une nouvelle bibliothèque de code plus générique, capable de contrôler nombre bien plus conséquent de modules radio. Cette nouvelle bibliothèque s'appelle RadioHe (http://www.airspayce.com/mikem/arduino/RadioHead/). Celle-ci est cependant beaucoup plus complexe et beaucoup plus "lourde".

Personnellement, je ne recommande pas l'utilisation de cette nouvelle bibliothèque. VirtualWire est une solution simple, fiable et efficace. Le choix de ren la bibliothèque VirtualWire "obsolète" est pour moi une erreur.

Quand utiliser la bibliothèque VirtualWire

💓
Ce chapitre est très important. Prenez le temps de bien le comprendre pour ne pas vous retrouver en difficulté par la suite !

La bibliothèque VirtualWire est conçue sans notion d'adressage ou d'acquittement. Cela signifie qu'un message envoyé n'a absolument aucune garantie reçu par le destinataire. De plus, n'importe quel récepteur à portée du signal recevra le message.

N.B. Choisir d'utiliser la bibliothèque VirtualWire dans un domaine d'utilisation où celle-ci n'est pas adaptée entrainera au mieux pertes d'informations, au pire, des conséquences bien plus graves.

Le choix d'utiliser ou non la bibliothèque VirtualWire se résume à répondre à ces deux questions :

- Est-ce que les données doivent être envoyées vers un destinataire en particulier ou à n'importe quel récepteur à portée du signal ? Si la réponse est "ve destinataire en particulier", la bibliothèque VirtualWire n'est pas la bonne solution à employer.
- Est-ce que perdre un ou plusieurs messages (de suite ou aléatoirement) peut poser des problèmes ? Si oui, la bibliothèque VirtualWire n'est pas la b solution à employer.

PS J'ajouterai entre parenthèses qu'aucune forme de chiffrement ou de signature des données n'est effectuée par la bibliothèque VirtualWire.

Ce type de communication sans adressage et sans garantie de réception s'utilise classiquement pour transmettre des données non critiques, comme température, une hygrométrie, etc. Si votre station météo ou votre système domotique ne reçoit pas la température extérieure pendant quelques minutes / h à cause d'interférences, ce n'est pas grave.

Ce type de communication est aussi parfois utilisé dans des domaines où les données transmises sont plus sensibles, comme en aéromodélisme ou en robot pour la transmission des données de télémétries et de contrôle (vitesse, direction, etc.). Dans ce cas précis, un système de sécurité est obligatoir aéromodélisme par exemple, une sécurité "classique" est de couper l'alimentation des moteurs si le signal de l'opérateur au sol n'est pas reçu pendant plus secondes.

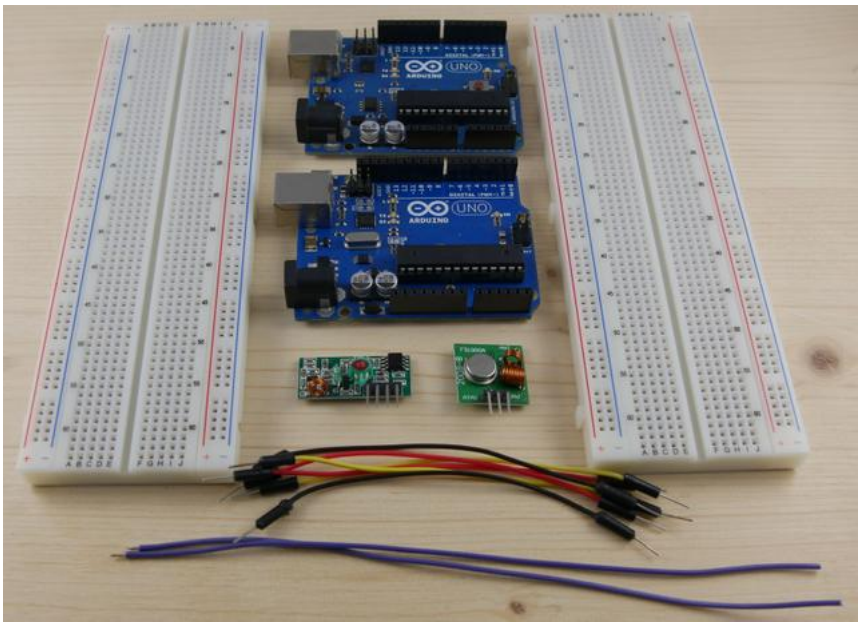
N.B. Si votre projet nécessite une communication bidirectionnelle, bien qu'il soit possible de mettre un émetteur et un récepteur sur chaque carte, il est so bien plus simple et moins coûteux d'utiliser des modules radio "tout intégré". Les modules radio "low cost" sont conçus pour des usages bien p généralement pour des communications unidirectionnelles. N'essayez pas de réinventer la roue carrée 😉

Remarque : la bibliothèque VirtualWire ne gère aucun adressage et n'assure aucune forme d'acquittement, mais elle inclut une détection des erreu transmission. Ainsi, si un message est reçu, il est possible de savoir si le message a été reçu correctement ou non.

De plus, pour contourner les interférences dues à l'utilisation massive de la fréquence 433MHz, la bibliothèque VirtualWire envoie plusieurs fois de suite ch message, afin de maximiser les chances que le récepteur reçoive un message complet, sans erreur.

Le montage de démonstration

Afin de tester la bibliothèque VirtualWire, nous allons réaliser ensemble un petit montage de démonstration. Celui-ci servira aussi de base pour le chapitre b
😉



(https://www.carnetdumaker.net/images/photographie-du-materiel-necessaire-la-realisation-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire/)

Matériel nécessaire

Pour réaliser ce montage, il va nous falloir :

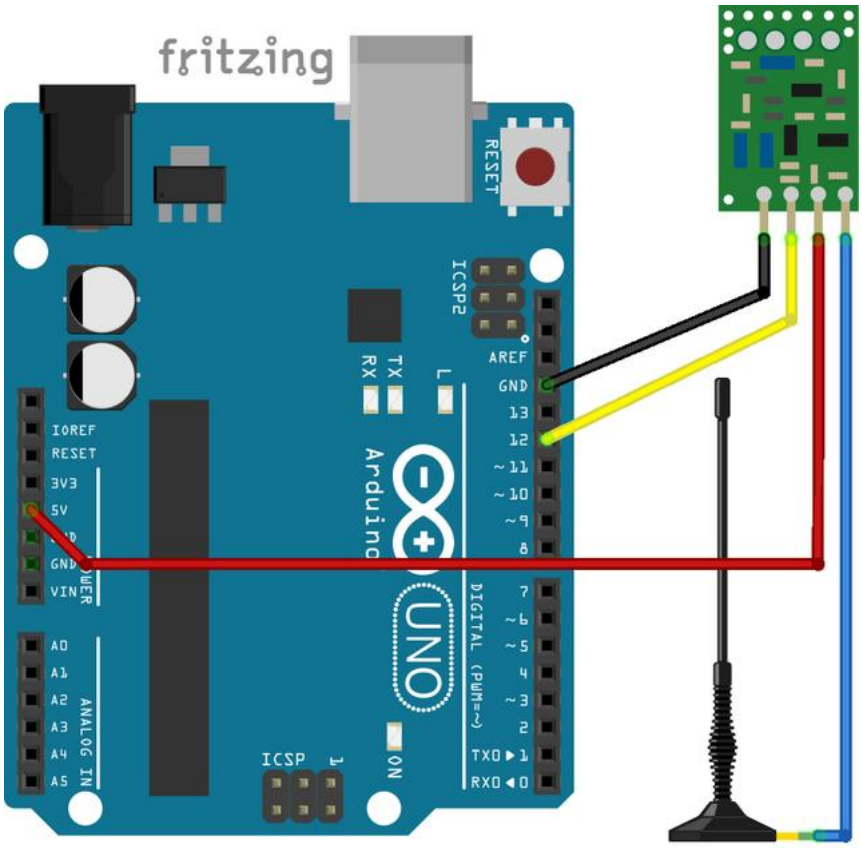
- Deux cartes Arduino UNO (et deux câbles USB),
- Deux modules radio 433MHz (un émetteur et un récepteur),
- Deux plaques d'essai et des fils pour câbler notre montage,
- Deux fils de 173mm de long pour les antennes des modules radio.

Pourquoi une antenne de 173mm exactement ?

Excellente question, cette longueur du fil d'antenne est liée à la longueur de l'onde radio à 433MHz.

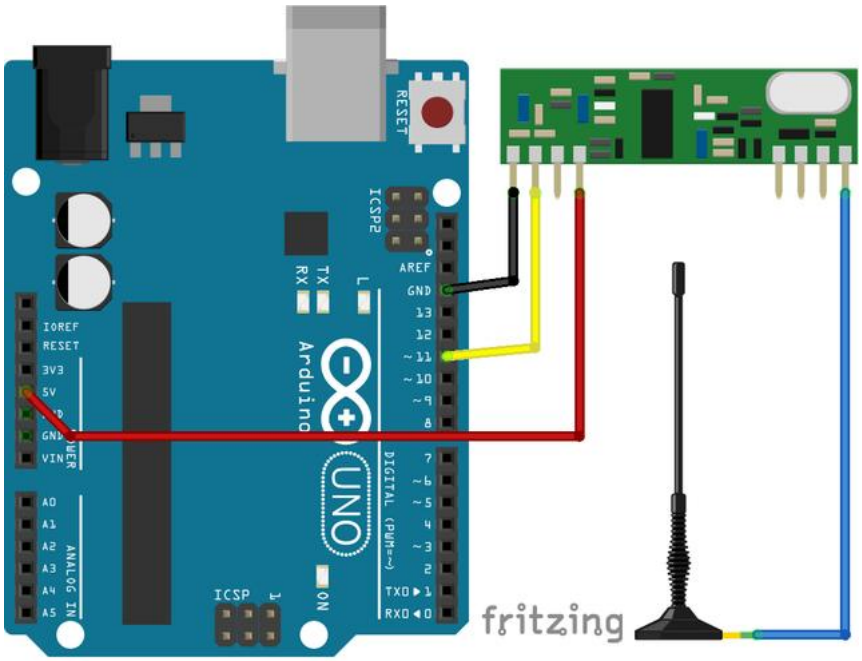
Il existe des calculettes en ligne (https://www.easycalculation.com/physics/electromagnetism/antenna-wavelength.php) pour calculer ce genre de longue de fil. Pour ce tutoriel, j'utilise une antenne "quart d'onde", ce qui est très classique pour une utilisation avec des modules radio "low cost".

PS Inutiles d'être très précis, les modules radio "low cost" sont généralement plus ou moins bien accordés sur la fréquence de la porteuse. Une longueur fil entre 18cm et 16,5cm marche sans trop de problèmes dans la plupart des cas.



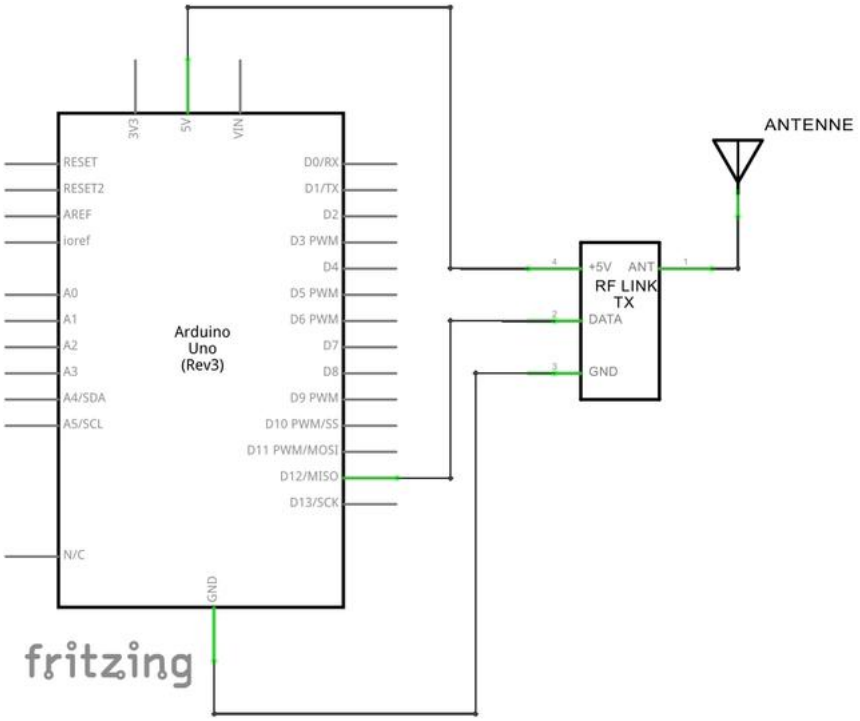
(https://www.carnetdumaker.net/images/vue-prototypage-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire-partie-emetteur/)

Vue prototypage du montage (partie émetteur)



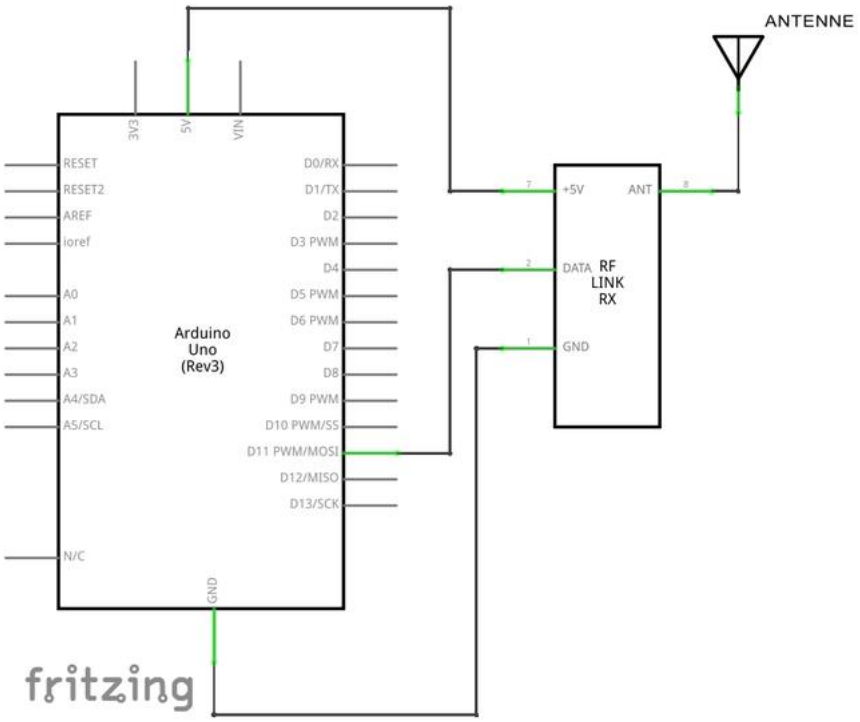
(https://www.carnetdumaker.net/images/vue-prototypage-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire-partie-recepteur/)

Vue prototypage du montage (partie récepteur)



(https://www.carnetdumaker.net/images/vue-schematique-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire-partie-emetteur/)

Vue schématique du montage (partie émetteur)

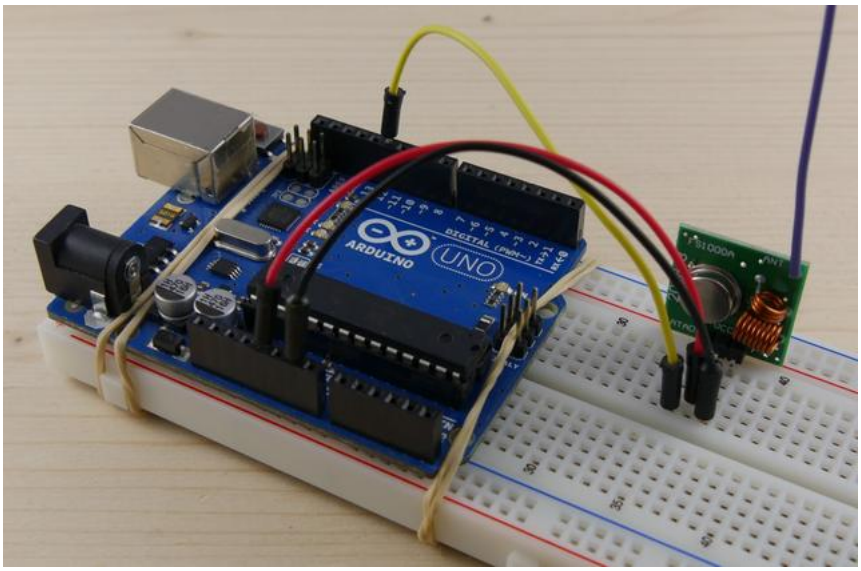


(https://www.carnetdumaker.net/images/vue-schematique-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire-partie-recepteur/)

Vue schématique du montage (partie récepteur)

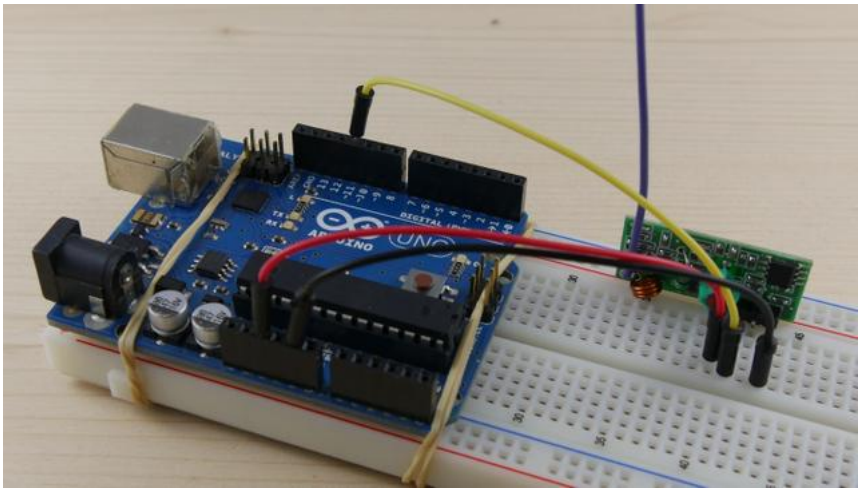
Le montage de cet article est double : un émetteur et un récepteur.

Dans les deux cas, il faudra commencer par câbler la broche `VCC` du module radio à l'alimentation `5V` de la carte Arduino au moyen d'un fil. On reliera ensu
broche `GND` du module radio à la broche `GND` de la carte Arduino.



(https://www.carnetdumaker.net/images/photographie-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire-partie-emetteur/)

Le montage fini (partie émetteur)


(https://www.carnetdumaker.net/images/photographie-du-montage-de-demonstration-de-la-bibliotheque-arduino-virtualwire-partie-recepteur/)

Le montage fini (partie récepteur)

Une fois le câblage des alimentations achevé, il ne restera plus qu'à câbler les broches de communication. Pour le récepteur, la sortie de communication doi reliée à la broche `D11` de la carte Arduino. Pour l'émetteur, la sortie de communication doit être reliée à la broche `D12` de la carte Arduino.

Astuce de bricoleur : les cartes Arduino UNO font exactement la même largeur qu'une plaque de prototypage classique. Un bête élastique permet d'obtenir une carte de test tout-en-un qui tient dans la main et que l'on peut balader avec soi pour faire des tests de portée par exemple.

Utilisation de la bibliothèque VirtualWire

Une fois la bibliothèque VirtualWire installée, il suffit d'ajouter ces quelques lignes en début de programme pour l'utiliser :

```
1  #include <VirtualWire.h>
```

Ensuite dans la fonction `setup()`, il suffit d'appeler la fonction `vw_setup()` pour initialiser la bibliothèque VirtualWire avec les options de configuratio défaut (on verra plus en détails les options possibles dans le chapitre suivant) :

```
1  void setup() // Fonction setup()
2  {
3    vw_setup(2000); // initialisation de la bibliothèque VirtualWire à 2000 bauds
4  }
```

Dans le cas du récepteur, il sera nécessaire d'ajouter un appel à `vw_rx_start()` après `vw_setup()` pour lancer l'écoute du signal radio :

```
void setup() // Fonction setup()
{
  vw_setup(2000); // initialisation de la bibliothèque VirtualWire à 2000 bauds
  vw_rx_start();
}
```

Les variables et fonctions de la bibliothèque VirtualWire

```
vw_set_tx_pin(broche);
```

Cette fonction permet de définir la broche utilisée pour transmettre les données, par défaut il s'agit de la broche `D12`.

```
vw_set_rx_pin(broche);
```

Cette fonction permet de définir la broche utilisée pour recevoir les données, par défaut il s'agit de la broche `D11`.

```
vw_set_ptt_pin(broche);
```

Cette fonction permet de définir la broche utilisée comme broche PTT ("Press to talk"), par défaut il s'agit de la broche `D10`.

La broche PTT permet d'utiliser des talkies-walkies ou des modules radio de radioamateurs qui n'émettent pas continuellement, mais seulement quan bouton est appuyé. La broche PTT permet à la bibliothèque VirtualWire de simuler l'appui sur ce bouton avant de transmettre des données.

PS Dans le cas des modules radio présentés dans le premier chapitre, ceux-ci n'utilisent pas de broche PTT. Ils émettent continuellement quand un signa transmis sur la broche `DATA`.

```
vw_set_ptt_inverted(est_inversé);
```

Cette fonction permet de définir l'état "actif" de la broche PTT. Par défaut, la broche PTT est active à l'état bas (`LOW`). En passant `true` en paramètre fonction, la broche PTT sera active à l'état haut (`HIGH`).

Il est nécessaire de vous reporter au manuel de votre émetteur radio pour savoir quel état de la broche PTT active l'émission.

```
vw_set_rx_inverted(est_inversé);
```

Cette fonction permet d'inverser l'état logique de la broche de réception. Il arrive que certains modules radio (en particulier les modules de radioamat inversent l'état du signal lors de la réception. Un `1` côté émetteur devient un `0` côté récepteur et inversement. En passant `true` en paramètre de la fonc l'état logique de la broche de réception sera inversé avec de traiter le signal.

Par défaut, la bibliothèque VirtualWire s'attend à avoir un signal à l'état bas (`LOW`) au repos et des impulsions à l'état haut (`HIGH`) lors de la transmissio données.

PS Dans le cas d'une utilisation avec un module "low cost" comme ceux présentés dans le premier chapitre, cette option n'est pas nécessaire.

N.B. Après l'appel d'une fonction `vw_set_*`, vous devrez toujours rappeler la fonction `vw_rx_start()` ou `vw_setup()` sinon vous ne recevrez jama messages.

```
vw_setup(vitesse_bps);
```

Cette fonction initialise la bibliothèque VirtualWire et configure la vitesse de transmission en bits par seconde.

N.B. Plus la vitesse de communication est lente, plus la portée est élevée. Au contraire, une vitesse de communication élevée diminuera la portée du si Dans le cas des modules "low cost" du premier chapitre, ne dépassé jamais 2600 bits par secondes. Dans le cas contraire, vous aurez beaucoup d'erreu communication.

```
vw_rx_start();
```

Déclenche le processus de réception du signal. Une fois cette fonction appelée, vous commencerez à recevoir les messages.

```
vw_rx_stop();
```

Arrête le processus de réception du signal. Une fois cette fonction appelée, vous ne recevrez plus aucun message. Pour redémarrer le processus de réceptio signal, il faudra rappeler `vw_rx_start()`.

```
vx_tx_active();
```

Retourne vrai (`true`) si la partie transmission de la bibliothèque VirtualWire est active (données en cours d'envoi).

```
vw_wait_tx();
```

Fonction bloquante permettant d'attendre la fin de la transmission en cours avant de rendre la main à la suite du programme.

```
vw_wait_rx();
```

Fonction bloquante permettant d'attendre l'arrivée d'un nouveau message avant de rendre la main à la suite du programme.

```
vw_wait_rx_max(unsigned long milliseconds);
```

Fonction équivalente à `vw_wait_rx()`, mais avec un timeout (en millisecondes). Cette fonction attend l'arrivée d'un nouveau message ou le dépasseme timeout avant de rendre la main à la suite du programme.

Retourne `true` si un message a été reçu avant le timeout. Retourne `false` si le timeout a été atteint.

```
vw_send(byte* buf, byte len);
```

Cette fonction permet de transmettre un message de taille `len` à partir des données contenues dans le tableau d'octets `buf`. La fonction rend la main à la du programme immédiatement. La transmission effective est réalisée en tâche de fond.

Retourne `true` s'il n'y a pas d'incohérence entre la taille des données et la taille maximum d'un message (77 octets pour les versions 1.17 et supérieurs, 27 pour les versions avant 1.17). Retourne `false` si le message ne peut pas être envoyé.

```
vw_have_message();
```

Retourne `true` si un message a été reçu et est prés à être lu.

N.B. Cette fonction retourne `true` même si le message reçu est corrompu !

```
vw_get_message(byte* buf, byte* len);
```

Copie le message reçu dans `buf` avec comme limite à ne pas dépasser `len`, si le message dépasse len celui-ci sera tronqué. Une fois le message copié dans la taille du message est copiée dans `len`.

La fonction retourne `true` si le message reçu est correct. Elle retournera `false` si le message est corrompu.

N.B. Le buffer doit être un tableau d'octets et len doit être un pointeur vers une variable de type `byte`. La valeur de la variable len impérativement correspondre à la taille de buf avant l'appel à `vw_get_message()`.

N.B. Le message est copié dans `buf` même si celui-ci est corrompu. Cependant, la fonction ne retournera `true` que si le messa correctement été reçu, sinon elle retournera `false`.

```
vw_get_rx_good();
```

Retourne le nombre de messages correctement reçus.

Attention, cette fonction retourne un compteur sur 8 bits, soit des valeurs comprises entre `0` et `255`. Ce compteur retourne à zéro quand la valeur dépasse À vous de gérer le retour à zéro si cela est nécessaire.

```
vw_get_rx_bad();
```

Retourne le nombre de messages reçus, mais corrompus.

Attention, cette fonction retourne un compteur sur 8 bits, soit des valeurs comprises entre `0` et `255`. Ce compteur retourne à zéro quand la valeur dépasse À vous de gérer le retour à zéro si cela est nécessaire.

Émission et réception

L'envoi d'un paquet de données se fait en trois étapes :

```
byte paquet[27];
strcpy(paquet, "Hello World!"); // Préparation du paquet

vw_send(paquet, 27); // Envoi du paquet

vw_wait_tx(); // Attente de la fin de l'envoi
```

1. Mettre en forme les données pour que le tout donne un joli tableau d'octets ne dépassant pas la taille maximum d'un message.
2. Envoyer le paquet avec `vw_send()`.
3. Attendre la fin de l'envoi avec `vw_wait_tx()`.

La réception se fait en deux étapes :

```
byte paquet[27];
byte taille_paquet = 27;

vw_wait_rx(); // On attend de recevoir quelque chose

if (vw_get_message(paquet, &taille_paquet)) { // Réception du paquet
  // Le paquet reçu est correct, on peut le traiter
  // A ce stade "taille_paquet" contient la taille du paquet de données reçu
}
```

1. On prépare un tableau d'octets suffisamment grand pour recevoir un paquet de données, de même qu'une variable de type `byte` que l'on initialise à la du tableau d'octets.
2. On attend de recevoir un paquet avec `vw_wait_rx()`.
3. On copie les données du paquet reçu dans le tableau d'octets avec `vw_get_message()`. On teste la valeur de retour de `vw_get_message()`. Si la valeu vraie (`true`), on peut considérer le message comme valide et le traiter.

Codes d'exemples

Comme je sais que beaucoup d'entre vous doivent se poser des questions du style "mais comment j'envoie un `long` / `int` / `char` / `float` ?", "comment je faire pour allumer une LED à distance", etc. Je vous propose quelques codes d'exemples qui devraient couvrir une grande majorité des cas d'usage classique bibliothèque VirtualWire.

N.B. J'utilise des messages d'au maximum 27 octets dans mes exemples pour être compatible avec toutes les versions de VirtualWire. La cons `VW_MAX_MESSAGE_LEN` fournie par la bibliothèque VirtualWire contient la taille maximum en octet d'un message.

Envoi de texte

Premier code d'exemple, le but ici est d'envoyer et de recevoir une chaine de caractères de taille variable.

Le texte saisi dans le moniteur série du client se retrouve affiché dans le moniteur série du serveur.

Le client (envoi) :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Client d'envoi de texte
 */

#include <VirtualWire.h>

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);

  Serial.println("Go !");
}

void loop() {
  byte message[VW_MAX_MESSAGE_LEN];
  // N.B. La constante VW_MAX_MESSAGE_LEN est fournie par la lib VirtualWire

  // Lit un message de maximum 26 caractères depuis le port série
  int len = Serial.readBytesUntil('\n', (char*) message, VW_MAX_MESSAGE_LEN - 1);
  if (!len) {
    return; // Pas de message
  }
  message[len] = '\0'; // Ferme la chaine de caractères

  vw_send(message, len + 1); // On envoie le message
  vw_wait_tx(); // On attend la fin de l'envoi
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/55/) (le lien de téléchargement en contient le projet Arduino prêt à l'emploi).

Le code de la fonction `setup()` se contente d'initialiser le port série, d'initialiser la bibliothèque VirtualWire avec les options par défauts et d'afficher un me de bienvenue.

Le code de la fonction `loop()` fait trois choses :

- le code attend que l'utilisateur finisse de saisir une chaine de caractères dans le moniteur série.

  N.B. La chaine de caractères est lue en mode octets, il est donc nécessaire de fermer cette chaine de caractère par un caractère vide `'\0'` en fin de ch

- le code envoie la chaine de caractères,

- Pour finir, le code attend que la transmission se termine. [/ul]

Le serveur (réception) :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Serveur d'envoi de texte
 */

#include <VirtualWire.h>

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);
  vw_rx_start(); // On peut maintenant recevoir des messages

  Serial.println("Go !");
}

void loop() {
  byte message[VW_MAX_MESSAGE_LEN];
  byte taille_message = VW_MAX_MESSAGE_LEN;
  // N.B. La constante VW_MAX_MESSAGE_LEN est fournie par la lib VirtualWire

  /*
   La variable "taille_message" doit impérativement être remise à
   la taille du buffer avant de pouvoir recevoir un message.
   Le plus simple est d'utiliser une variable locale pour ne pas
   avoir à réassigner la valeur à chaque début de loop().
   */

  // On attend de recevoir un message
  vw_wait_rx();

  if (vw_get_message(message, &taille_message)) {
    // On copie le message, qu'il soit corrompu ou non

    Serial.println((char*) message); // Affiche le message
  }
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/56/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Même code pour la fonction `setup()`, à l'exception de l'ajout de `vw_rx_start()` pour activer la réception de messages.

Pour le code de la fonction `loop()`, rien de bien extraordinaire. Le code attend un paquet de données avec `vw_wait_rx()`, le copie en mémoire `vw_get_message()` et l'affiche sous forme de texte dans le moniteur série si celui-ci est correct.

Envoi de variable

Pour ce code d'exemple, imaginons que l'on veut transmettre un float qui provient d'une mesure d'un capteur. Le but est uniquement de transmett `float`, rien d'autre.

N.B. J'utilise un `float` dans cet exemple, mais cela peut être n'importe quel type de variable. Cela inclut tous les types de base (`int`, `long`, `char`, les tableaux de valeurs, mais aussi les structures.

Le code du client :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Client d'envoi de variable
 */

#include <VirtualWire.h>

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);

  Serial.println("Go !");
}

void loop() {

  // Lit un nombre depuis le port série
  float valeur = Serial.parseFloat();

  vw_send((byte *) &valeur, sizeof(valeur)); // On envoie le message
  vw_wait_tx(); // On attend la fin de l'envoi
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/57/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Dans la fonction `loop()`, on génère une valeur quelconque à transmettre, ici en lisant le port série, puis on envoi la valeur et on attend la fin de l'envoi.

Pour envoyer une variable comme s'il s'agissait d'un tableau d'octets, il est nécessaire de faire un cast de pointeur (une façon de dire au compilateur qu sait ce que l'on fait). Le `&` permet d'avoir un pointeur sur la variable cible (pas besoin de faire cela si la variable est déjà un pointeur ou un tablea valeurs). On utilise ensuite un cast `(byte*)` pour ordonner au compilateur de traiter le pointeur fraichement obtenu comme un pointeur sur un tal d'octets. En mémoire, il n'y a que des octets, cette astuce permet donc d'obtenir une représentation sous forme de tableau d'octets de n'importe q variable.

La fonction sizeof() (https://fr.wikipedia.org/wiki/Sizeof) permet de connaitre la taille en octets du type de variable que l'on souhaite transmettre.

N.B. Cette façon de faire est dépendante de l'architecture processeur. Si vous tentez d'utiliser cette méthode pour transmettre un `float` ou un simple entre deux cartes utilisant des processeurs d'architectures différentes, vous aurez des erreurs d'interprétations à la réception.

Le code du serveur :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Serveur d'envoi de variable
 */

#include <VirtualWire.h>

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);
  vw_rx_start(); // On peut maintenant recevoir des messages

  Serial.println("Go !");
}

void loop() {
  float valeur;
  byte taille_message = sizeof(float);

  /*
   La variable "taille_message" doit impérativement être remise à
   la taille de la variable avant de pouvoir recevoir un message.
   Le plus simple est d'utiliser une variable locale pour ne pas
   avoir à réassigner la valeur à chaque début de loop().
   */

  // On attend de recevoir un message
  vw_wait_rx();

  if (vw_get_message((byte *) &valeur, &taille_message)) {
    // On copie le message, qu'il soit corrompu ou non

    Serial.println(valeur); // Affiche le message
  }
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/58/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Du côté serveur, la fonction `loop()` utilise cette même astuce du cast pour faire passer la variable comme un tableau d'octets aux yeux de la fon `vw_get_message()`. Une fois la valeur copiée par dans la variable par `vw_get_message()`, il suffit d'utiliser la variable normalement.

Afin d'être complet, sachez qu'il est aussi possible de copier une variable dans un tableau d'octets au moyen de la fonction mem (http://www.cplusplus.com/reference/cstring/memcpy/). Cela n'apporte aucun avantage par rapport à la solution du cast, mais c'est parfois plus sim plus lisible pour des débutants.

Voilà ce que donnerait le code des fonctions `loop()` du client et du serveur en utilisant `memcpy()` :

```
void loop() {
  byte message[VW_MAX_MESSAGE_LEN];

  // Lit un nombre depuis le port série
  float valeur = Serial.parseFloat();

  // Copie le float dans le message
  memcpy(message, &valeur, sizeof(valeur));

  vw_send(message, sizeof(message)); // On envoie le message
  vw_wait_tx(); // On attend la fin de l'envoi
}
```

```
void loop() {
  byte message[VW_MAX_MESSAGE_LEN];
  byte taille_message = VW_MAX_MESSAGE_LEN;
  float valeur;

  // On attend de recevoir un message
  vw_wait_rx();

  if (vw_get_message(message, &taille_message)) {
    // On copie le message, qu'il soit corrompu ou non

    memcpy(&valeur, message, sizeof(valeur));
    Serial.println(valeur); // Affiche le message
  }
}
```

Envoi de structure

Dans le chapitre précédent, j'ai précisé que l'astuce du cast fonctionne avec n'importe quel type de données. Cela est aussi vrai pour les types de dor complexes réalisés au moyen de structure de données.

Exemple de code client :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Client d'envoi de structure
 */

#include <VirtualWire.h>

typedef struct {
  char commande;
  int valeur;
} MaStructure;

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);

  Serial.println("Go !");
}

void loop() {
  MaStructure message;

  // Lit un nombre depuis le port série
  while(!Serial.available()); // Attend un caractère
  message.commande = Serial.read();
  message.valeur = Serial.parseInt();

  vw_send((byte*) &message, sizeof(message)); // On envoie le message
  vw_wait_tx(); // On attend la fin de l'envoi
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/59/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Et de code serveur :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Serveur d'envoi de structure
 */

#include <VirtualWire.h>

typedef struct {
  char commande;
  int valeur;
} MaStructure;

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);
  vw_rx_start(); // On peut maintenant recevoir des messages

  Serial.println("Go !");
}

void loop() {
  MaStructure message;
  byte taille_message = sizeof(MaStructure);

  /*
   La variable "taille_message" doit impérativement être remise à
   la taille de la structure avant de pouvoir recevoir un message.
   Le plus simple est d'utiliser une variable locale pour ne pas
   avoir à réassigner la valeur à chaque début de loop().
   */

  // On attend de recevoir un message
  vw_wait_rx();

  if (vw_get_message((byte *) &message, &taille_message)) {
    // On copie le message, qu'il soit corrompu ou non

    Serial.print("commande="); // Affiche le message
    Serial.print(message.commande);
    Serial.print(" valeur=");
    Serial.println(message.valeur);
  }
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/60/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Comme vous pouvez le voir, il est possible d'envoyer des données structurées assez facilement. En utilisant une structure bien pensée, il est possib transmettre à peu près n'importe quelle information ou commande.

⚠ Les pointeurs mordent

Attention aux pointeurs dans les structures, lors de l'envoi, seule l'adresse du pointeur est transmise et non la valeur pointée.

Je vous invite à lire ma remarque sur ce sujet dans l'article sur le stockage de structure en mémoire EEPRO (https://www.carnetdumaker.net/articles/stocker-des-donnees-en-memoire-eeprom-avec-une-carte-arduino-genuino/#bonus-lire-et-ecrire-des-donnees-structurees-en-memoire-eeprom) pour plus de détails.

Envoi de tableaux de valeurs

Toujours dans le même esprit, l'astuce du cast fonctionne aussi avec des tableaux de valeurs, comme par exemple un tableau de `float`, `int`, `char`, et

Exemple de code client :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Client d'envoi de tableau de valeurs
 */

#include <VirtualWire.h>

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);

  Serial.println("Go !");
}

void loop() {
  int valeurs[6];

  // Lit les broches analogiques
  valeurs[0] = analogRead(0);
  valeurs[1] = analogRead(1);
  valeurs[2] = analogRead(2);
  valeurs[3] = analogRead(3);
  valeurs[4] = analogRead(4);
  valeurs[5] = analogRead(5);

  vw_send((byte *) &valeurs, sizeof(valeurs)); // On envoie le message
  vw_wait_tx(); // On attend la fin de l'envoi

  delay(1000);
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/61/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Et de code serveur :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Serveur d'envoi de tableau de valeurs
 */

#include <VirtualWire.h>

void setup() {
  Serial.begin(9600);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);
  vw_rx_start(); // On peut maintenant recevoir des messages

  Serial.println("Go !");
}

void loop() {
  int valeurs[6];
  byte taille_message = sizeof(valeurs);

  /*
   La variable "taille_message" doit impérativement être remise à
   la taille du tableau avant de pouvoir recevoir un message.
   Le plus simple est d'utiliser une variable locale pour ne pas
   avoir à réassigner la valeur à chaque début de loop().
   */

  // On attend de recevoir un message
  vw_wait_rx();

  if (vw_get_message((byte *) &valeurs, &taille_message)) {
    // On copie le message, qu'il soit corrompu ou non

    Serial.print("valeurs[0]=");
    Serial.println(valeurs[0]); // Affiche le message
    Serial.print("valeurs[1]=");
    Serial.println(valeurs[1]);
    Serial.print("valeurs[2]=");
    Serial.println(valeurs[2]);
    Serial.print("valeurs[3]=");
    Serial.println(valeurs[3]);
    Serial.print("valeurs[4]=");
    Serial.println(valeurs[4]);
    Serial.print("valeurs[5]=");
    Serial.println(valeurs[5]);
  }
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/62/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Envoi de commande simple

En modifiant quelques lignes du code d'exemple du serveur d'envoi de texte, il est possible de faire un système de commande à distance simpliste :

```
void loop() {
  byte message[VW_MAX_MESSAGE_LEN];
  byte taille_message = VW_MAX_MESSAGE_LEN;

  vw_wait_rx();

  if (vw_get_message(message, &taille_message)) {

    if (strcmp((char*) message, "on") == 0) {
      // Fait quelque chose si le message est "on"

    } else if (strcmp((char*) message, "off") == 0) {
      // Fait quelque chose d'autre si le message est "off"
    }
  }
}
```

La fonction strcmp() (http://www.cplusplus.com/reference/cstring/strcmp/) retourne `0` quand deux chaines de caractères sont identiques. Il suffit d'appeler `strcmp()` dans une série de `if()` pour tester la valeur de la commande et réaliser les actions nécessaires en conséquence.

Envoi de commandes complexes

Dans le même principe que le chapitre précédent, en modifiant quelques lignes du code d'exemple du serveur d'envoi de structure, il est possible de fai système de commande à distance relativement complet, avec paramètres :

```
void loop() {
  MaStructure message;
  byte taille_message = sizeof(MaStructure);

  vw_wait_rx();

  if (vw_get_message((byte *) &message, &taille_message)) {

    switch (message.commande) {
      case 'A':
         // Fait quelque chose
        break;

      case 'B':
         // Fait autre chose
        break;

      // ...

      default:
          // Fait quelque chose quand la commande n'est pas comprise
    }
  }
}
```

Dans le code d'exemple, j'avais utilisé une structure avec deux champs : `commande` (un caractère) et `valeur` (un nombre entier). En utilisant un swi (http://www.commentcamarche.net/contents/111-langage-c-les-structures-conditionnelles#l-instruction-bold-switch-bold) (équivalent à une séri `if()` sur des valeurs numériques), il est possible de faire plusieurs actions avec un paramètre. Exemple : `A` = allumer, `a` = éteindre, avec `valeur` numéro de la broche à allumer / éteindre à distance.

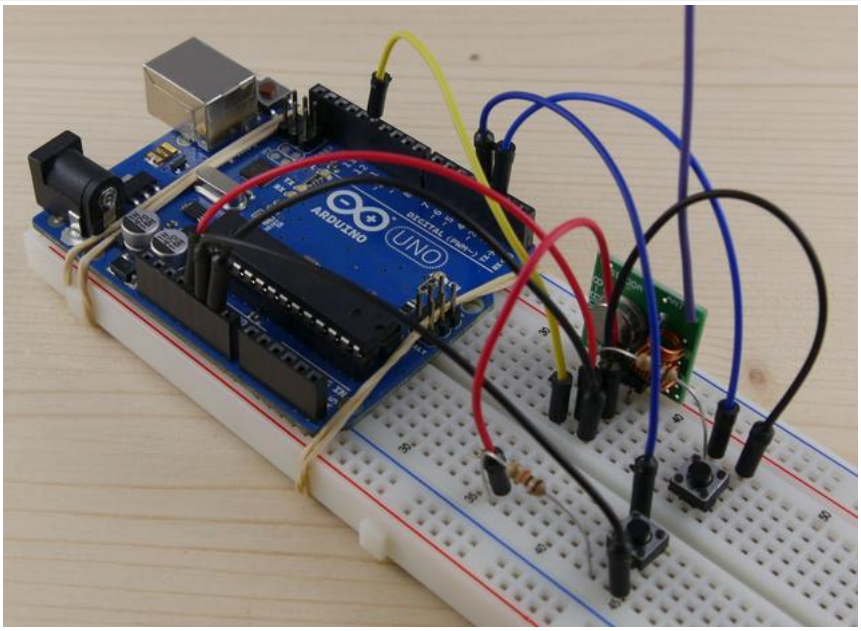
Libre à vous de faire la structure de données qui convient à votre projet 😉

Bonus : une télécommande DIY

Pour terminer cet article en beauté, je vous propose de fabriquer une télécommande sans fil.

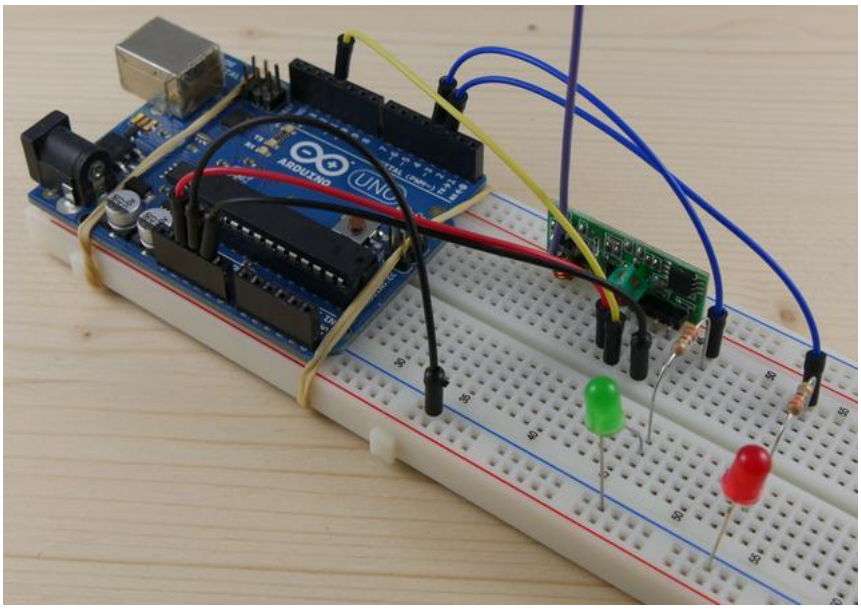
Cet exemple pratique pourra servir de base pour des projets domotiques ou de télécommandes à distance 😉

Le montage



(https://www.carnetdumaker.net/images/telecommande-diy-avec-un-module-433mhz-et-virtualwire-partie-emetteur/)

Télécommande DIY (partie émetteur)


(https://www.carnetdumaker.net/images/telecommande-diy-avec-un-module-433mhz-et-virtualwire-partie-recepteur/)

Télécommande DIY (partie récepteur)

Le montage est le même que celui en début d'article. Seulement, sur une des cartes, deux boutons poussoir sont câblés sur les broches `D2` et `D3`, alor sur l'autre carte il s'agit de LEDs.

Je vous renvoie vers mon article sur les entrées sorties numériques (https://www.carnetdumaker.net/articles/utiliser-les-entrees-sorties-numeri dune-carte-arduino-genuino/) pour le câblage des boutons poussoirs et des LEDs.

Le code

Tout d'abord l'émetteur avec son bouton :

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Télécommande DIY (émetteur)
 */

#include <VirtualWire.h>

/** Broches pour les boutons */
const byte PIN_BUTTON_A = 2;
const byte PIN_BUTTON_B = 3;

/** Différents messages de commande */
const char* CMD_BUTTON_A = "BPA";
const char* CMD_BUTTON_B = "BPB";

void setup() {
  Serial.begin(9600);

  /* Met les broches des boutons en entrées avec pull-up */
  pinMode(PIN_BUTTON_A, INPUT_PULLUP);
  pinMode(PIN_BUTTON_B, INPUT_PULLUP);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);

  Serial.println("Go !");
}

void loop() {
  byte message[VW_MAX_MESSAGE_LEN];
  // N.B. La constante VW_MAX_MESSAGE_LEN est fournie par la lib VirtualWire

  /* Envoi la commande adéquate */
  if (digitalRead(PIN_BUTTON_A) == LOW ) {

    vw_send((byte*) CMD_BUTTON_A, strlen(CMD_BUTTON_A) + 1); // On envoie le message
    vw_wait_tx(); // On attend la fin de l'envoi

    delay(50); // Attend que le bouton soit relâché
    while(digitalRead(PIN_BUTTON_A) == LOW);
    delay(50);

  } else if (digitalRead(PIN_BUTTON_B) == LOW) {

    vw_send((byte*) CMD_BUTTON_B, strlen(CMD_BUTTON_B) + 1); // On envoie le message
    vw_wait_tx(); // On attend la fin de l'envoi

    delay(50); // Attend que le bouton soit relâché
    while(digitalRead(PIN_BUTTON_B) == LOW);
    delay(50);
  }
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/63/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Le code est à peu près le même que celui de l'exemple pour envoyer des chaines de caractères. La seule différence est l'ajout du code pour les bou poussoirs en début de programme et dans `setup()`.

Le code est configuré pour envoyer une chaine de caractères bien précise (une par bouton) quand un bouton est appuyé (état `LOW`).

Afin de rendre l'utilisation de la télécommande plus simple, le code attend toujours que le bouton soit relâché avant de continuer. Un délai avant et l'attente évite les rebonds du bouton (cf article sur les entrées sorties).

```
/**
 * Exemple de code pour la bibliothèque VirtualWire – Télécommande DIY (récepteur)
 */

#include <VirtualWire.h>

/** Broches pour les LEDs */
const byte PIN_LED_A = 2;
const byte PIN_LED_B = 3;

/** Différents messages de commande */
const char* CMD_BUTTON_A = "BPA";
const char* CMD_BUTTON_B = "BPB";

void setup() {
  Serial.begin(9600);

  /* Met les broches des LEDs en sortie et à LOW */
  pinMode(PIN_LED_A, OUTPUT);
  digitalWrite(PIN_LED_A, LOW);
  pinMode(PIN_LED_B, OUTPUT);
  digitalWrite(PIN_LED_B, LOW);

  // Initialisation de la bibliothèque VirtualWire
  // Vous pouvez changez les broches RX/TX/PTT avant vw_setup() si nécessaire
  vw_setup(2000);
  vw_rx_start(); // On peut maintenant recevoir des messages

  Serial.println("Go !");
}

void loop() {
  byte message[VW_MAX_MESSAGE_LEN];
  byte taille_message = VW_MAX_MESSAGE_LEN;
  // N.B. La constante VW_MAX_MESSAGE_LEN est fournie par la lib VirtualWire

  /*
   La variable "taille_message" doit impérativement être remise à
   la taille du buffer avant de pouvoir recevoir un message.
   Le plus simple est d'utiliser une variable locale pour ne pas
   avoir à réassigner la valeur à chaque début de loop().
   */

  // On attend de recevoir un message
  vw_wait_rx();

  if (vw_get_message(message, &taille_message)) {
    // On copie le message, qu'il soit corrompu ou non

    if (strcmp((char*) message, CMD_BUTTON_A) == 0) {
      digitalWrite(PIN_LED_A, !digitalRead(PIN_LED_A));
      Serial.println("TOGGLE LED A");

    } else if (strcmp((char*) message, CMD_BUTTON_B) == 0) {
      digitalWrite(PIN_LED_B, !digitalRead(PIN_LED_B));
      Serial.println("TOGGLE LED B");
    }
  }
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (https://www.carnetdumaker.net/snippets/64/) (le lien de téléchargeme .zip contient le projet Arduino prêt à l'emploi).

Le code du récepteur est similaire au code de l'exemple de commande texte simple. Quand la chaine de caractères permettant de changer l'état d'une est reçue, le code lit l'état courant de la LED et l'inverse.

PS Un message est aussi envoyé sur le port série, pour ceux qui voudraient tester le code sans câbler de LEDs 😉

Conclusion

Ce tutoriel est désormais terminé.

Si ce tutoriel vous a plu, n'hésitez pas à le commenter sur le forum, à le diffuser sur les réseaux sociaux et à soutenir le site si cela vous fait plaisir.

♻ Articles en relation avec celui-ci

- Communiquer sans fil avec un module nRF24L01, la bibliothèque Mirf

et une carte Arduino / Genuino (/articles/communiquer-sans-fil-avec-un-module-nrf24l01-la-bibliotheque-mirf-et-une-carte-arduino-genuino/)
- Utiliser les entrées / sorties numériques d'une carte Arduino / Genuino (/articles/utiliser-les-entrees-sorties-numeriques-dune-carte-arduino-genuino/)

Cliquez ici pour accéder aux commentaires de l'article. (/forum/topics/57-communiquer-sans-fil-en-433mhz-avec-la-bibliotheque-virtualwire-et-une-ca arduino-genuino/)
