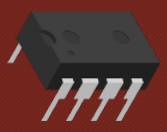
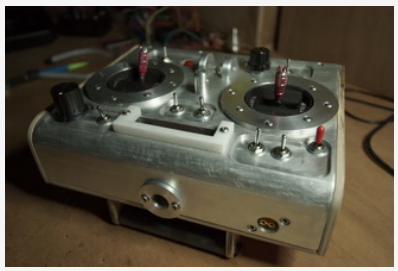
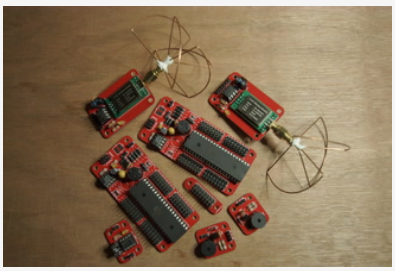


Une radiocommande R/C avec Nrf24l01p.h

Nrf24l01p.h permet de transmettre de manière **bi-directionnelle** des informations (sur 32 bits quelles qu'elles soient) sur la bande fréquence des **2.4Ghz**:





Cette classe utilise le composant **nRF24L01+** qui communique en **SPI** avec le microcontrôleur.

Ports des automates programmables concernés par le SPI:

> ***Automate programmable MODULABLE M20:***
> *- Port 11 (PB2) = SS (slave select)*
> *- Port 12 (PB3) = MOSI (master output slave input)*
> *- Port 13 (PB4) = MISO (master input slave output)*
> *- Port 14 (PB5) = SCK (serial clock)*
>
> ***Automate programmable MODULABLE M32:***
> *- Port 5 (PB4) = SS (slave select)*
> *- Port 6 (PB5) = MOSI (master output slave input)*
> *- Port 7 (PB6) = MISO (master input slave output)*
> *- Port 8 (PB7) = SCK (serial clock)*

Exemple d'émetteur:

```
#include "../module/1284p/Nrf24l01p.h"

int main()
{
	Nrf24l01p myChannel = Nrf24l01p (1);
	unsigned char increment = 0;

	Nrf24l01p::start (5, 32, 1524003746, true);

	while (true)
	{
		myChannel.transmit (increment);

		if (increment < 255)
		{
			increment++;
		}
		else
		{
			increment = 0;
		}
	}

	return 0;
}
```

Dans cet exemple, un objet **myChannel** de type **Nrf24l01p** est déclaré, en paramètre est indiqué le canal **1** sur lequel communiquer les informations (il y a 64 canaux en tout). À la ligne suivante une variable de type **unsigned char** (32 bits non signés) est déclarée avec une valeur de **0**, elle va servir à incrémenter un nombre entier.

Puis, le composant nRF24L01+ est démarré en appelant la fonction statique **start** prenant plusieurs paramètres:
- Le 1er paramètre **5** est le numéro du port de l'automate programmable sur lequel est connectée la broche SS (slave select) du composant nRF24L01+.

> *Ce paramètre est utile lorsque vous souhaitez connecter en SPI différents composants en série ou en parallèle avec le microcontrôleur.*

- Le 2ème paramètre **32** est le numéro du port de l'automate programmable laissé libre (en l'air) relié en interne au convertisseur analogique/numérique du microcontrôleur. Ce port sert au système anti-collisions qui utilise du bruit analogique pour générer de l'aléatoire. Si vous ne souhaitez pas utiliser le système anti-collisions, indiquez **0** en paramètre.

> *Le système anti-collisions de la classe Nrf24l01p.h permet d'éviter que deux (ou plus) nRF24L01+ ne se transmettent des données exactement au même moment. Il est vivement conseillé dans le cadre d'une **communication bi-directionnelle d'activer le système anti-collisions sur au moins l'un des deux montages** en sélectionnant un port approprié.*

Ports des automates programmables concernés par l'analogique:

> ***Automate programmable MODULABLE M20:***
> *- Port 15 (PC0)*
> *- Port 16 (PC1)*
> *- Port 17 (PC2)*
> *- Port 18 (PC3)*
> *- Port 19 (PC4)*
> *- Port 20 (PC5)*
>
> ***Automate programmable MODULABLE M32:***
> *- Port 25 (PA7)*
> *- Port 26 (PA6)*
> *- Port 27 (PA5)*
> *- Port 28 (PA4)*
> *- Port 29 (PA3)*
> *- Port 30 (PA2)*
> *- Port 31 (PA1)*
> *- Port 32 (PA0)*

- Le 3ème paramètre **1524003746** correspond à la clé unique qui permet de sécuriser la communication entre deux (ou plus) nRF24L01+.

> *C'est à vous de choisir cette clé unique, plus le nombre est complexe et plus la communication sera sécurisée et protégée contre les parasites, mais il faut reconnaître que n'importe quel nombre sur 32 bits différent de 0 avec seulement quelques chiffres, est largement suffisant.*

- Le 4ème paramètre **true** indique d'émettre à la puissance maximale. Une valeur sur **false** serait la puissance minimale, ce qui peut être suffisant en communication courte distance (par exemple en intérieur).

Puis dans la boucle **while**, la fonction **transmit** de l'objet **myChannel** est utilisée pour transmettre une information 32 bits à un autre nRF24L01+. Dans ce cas c'est la valeur de la variable **increment** qui est transmise, variable dont la valeur est incrémentée aux lignes suivantes.

> *Ce petit bout de programme est simple, mais il permet déjà d'assurer une communication 2.4Ghz efficace et sécurisée entre deux nRF24L01+.*

Envoyer la valeur d'une variable qui s'incrémente au cours du temps peut servir à programmer simplement un système à tolérance de pannes (fail-safe) sur la partie récepteur du montage. En effet, si la valeur reçue ne s'incrémente plus pendant un certain temps (1 seconde par exemple), il peut être alors intéressant de déclencher une mise au neutre des servo-moteurs, une coupure de la motorisation d'un aéronef, une procédure de vol automatisée par gyroscope, etc...

www.sylvainmahe.xyz

LE BLOG
de Sylvain Mahé
contact@sylvainmahe.xyz

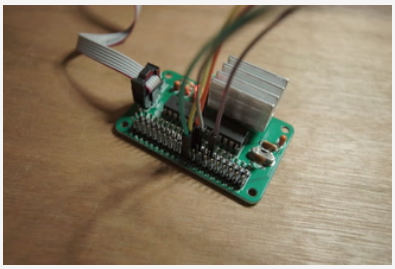
Exemple de récepteur:

```
#include "../module/1284p/Nrf24l01p.h"

int main()
{
        Nrf24l01p myChannel = Nrf24l01p (1);
        unsigned char increment = 0;

        Nrf24l01p::start (5, 32, 1524003746, true);

        while (true)
        {
                myChannel.receive();

                //myChannel.data sont les données reçues sur ce canal:
                increment = myChannel.data;
        }

        return 0;
}
```

Dans cet exemple le seul changement notable est l'appel de la fonction **receive** de l'objet **myChannel** afin de recevoir les informations émises, en l'occurrence ici la valeur de la variable incrémentée dans le montage coté émetteur.

Connexions (nRF24L01+ sur automates programmables):

> *- Broches VDD (+3.3V) sur broche +3.3V disponible via un régulateur de tension adapté.*
> *- Broches VSS (GND) sur broche GND disponible.*
> *- Broche CSN (slave select) sur port SS ou tout autre port d'entrée/sortie disponible.*
> *- Broche MOSI (master output slave input) sur port MOSI.*
> *- Broche MISO (master input slave output) sur port MISO.*
> *- Broche SCK (serial clock) sur port SCK.*



> *Attention, le composant **nRF24L01+** fonctionne avec une tension de **+3.3V**, il convient donc d'utiliser un régulateur de tension adapté !*

> *La classe Nrf24l01p.h dispose d'une fonction **reset** ce qui permet de remettre à l'état **false** les variables **received** et **transmitted**. Ces variables sont utiles pour savoir si une ou plusieurs données ont été reçues ou transmises avec succès. La variable **data** (qui correspond aux données reçues ou émises) est également réinitialisée à **0**.*
>
> *Une fonction statique **stop** existe également et permet d'éteindre le circuit d'émission/réception du composant nRF24L01+, un nouvel appel à la fonction **start** permet de redémarrer le circuit.*

Récapitulatif des fonctions et variables de cette classe:

```
signed long data = 0;
bool received = false;
bool transmitted = false;
Nrf24l01p (const unsigned char ADDRESS);
static void start (const unsigned char PIN_SS, const unsigned char PIN_ANTI_COLLISION,
void receive();
void transmit (const signed long DATA);
void reset();
static void stop();
```

Exemple d'un système R/C 5 voies:

Dans l'exemple suivant, 4 potentiomètres (2 par manche) sont connectés aux ports 25, 26, 27, et 28 de l'automate programmable. Ce sont des GPIO connectées au convertisseur analogique/numérique du microcontrôleur. Un interrupteur est également connecté au port 1 de l'automate programmable, ce qui va servir d'interrupteur de coupure moteur dans ce cas précis.

> *Les 4 potentiomètres et l'interrupteur servent **d'interface utilisateur entre l'homme et la machine**.*

5 objets de type **Nrf24l01p** (correspondants aux 5 voies) servent à transmettre les valeurs brutes des 4 potentiomètres et de l'interrupteur (respectivement des valeurs allants de 0 à 1023 pour les potentiomètres et 0 ou 1 pour l'interrupteur).

L'émetteur:

```
#include "../module/1284p/AnalogRead.h"
#include "../module/1284p/GpioRead.h"
#include "../module/1284p/Nrf24l01p.h"

int main()
{
        AnalogRead stickThrottle = AnalogRead (25);
        AnalogRead stickPitch = AnalogRead (26);
        AnalogRead stickRoll = AnalogRead (27);
        AnalogRead stickYaw = AnalogRead (28);
        GpioRead buttonCut = GpioRead (1, true, 20);
        Nrf24l01p channelThrottle = Nrf24l01p (1);
        Nrf24l01p channelPitch = Nrf24l01p (2);
        Nrf24l01p channelRoll = Nrf24l01p (3);
        Nrf24l01p channelYaw = Nrf24l01p (4);
        Nrf24l01p channelCut = Nrf24l01p (5);

        Nrf24l01p::start (5, 32, 1524003746, true);

        while (true)
        {
                stickThrottle.read();
                stickPitch.read();
                stickRoll.read();
                stickYaw.read();
                buttonCut.read();

                channelThrottle.transmit (stickThrottle.value);
                channelPitch.transmit (stickPitch.value);
                channelRoll.transmit (stickRoll.value);
                channelYaw.transmit (stickYaw.value);
                channelCut.transmit (buttonCut.continuous);
        }

        return 0;
}
```

L'exemple suivant est la partie récepteur du montage. 5 objets de type **PwmWrite** sont utilisés pour générer des signaux **PWM** ce qui permet de faire fonctionner 4 servo-moteurs:
- Le 1er pour les gaz (throttle).
- Le 2ème pour l'axe de tangage (pitch).
- Le 3ème pour l'axe de rouli (roll).
- Le 4ème pour l'axe de lacet (yaw).

Une fois reçues avec les fonctions **receive**, les valeurs brutes sont transformées à l'aide de la classe **Math.h** (qui permet de créer entre autre des courbes) en valeurs adaptées PWM (modulation de la largeur d'impulsion en microsecondes) afin de les envoyer aux servo-moteurs par les ports 1, 2, 3, et 4 de l'automate programmable.

Le récepteur:

```
#include "../module/1284p/Nrf24l01p.h"
#include "../module/1284p/PwmWrite.h"
#include "../module/1284p/Math.h"

int main()
{
	Nrf24l01p channelThrottle = Nrf24l01p (1);
	Nrf24l01p channelPitch = Nrf24l01p (2);
	Nrf24l01p channelRoll = Nrf24l01p (3);
	Nrf24l01p channelYaw = Nrf24l01p (4);
	Nrf24l01p channelCut = Nrf24l01p (5);
	PwmWrite servoThrottle = PwmWrite (1);
	PwmWrite servoPitch = PwmWrite (2);
	PwmWrite servoRoll = PwmWrite (3);
	PwmWrite servoYaw = PwmWrite (4);

	Nrf24l01p::start (5, 32, 1524003746, true);
	PwmWrite::start (50);

	while (true)
	{
		channelThrottle.receive();
		channelPitch.receive();
		channelRoll.receive();
		channelYaw.receive();
		channelCut.receive();

		if (channelCut.data == true)
		{
			servoThrottle.us (1000);
		}
		else
		{
			servoThrottle.us (Math::curve (0, channelThrottle.data, 1023, 1
			servoPitch.us (Math::curve (0, channelPitch.data, 1023, 1000, 2
			servoRoll.us (Math::curve (0, channelRoll.data, 1023, 1000, 200
			servoYaw.us (Math::curve (0, channelYaw.data, 1023, 1000, 2000,
		}
	}

	return 0;
}
```

La 5ème voie (cut) permet la **coupure des gaz** quel que soit la position du manche de gaz, c'est une des sécurités fondamentales notamment en aéromodélisme.

> *À ce propos, je ne pourrais être tenu pour responsable si vous faites une mauvaise utilisation de mes exemples !*

Ce qui signifie que vous utilisez ma programmation en toute connaissance de cause et en règle avec la loi en vigueur dans votre pays (notamment en ce qui concerne les lieux de vols autorisés, les fréquences et puissances d'émissions des radio-émetteurs, etc...).

> *Libre à vous d'explorer les possibilités de **MODULE** afin d'utiliser et de modifier ces exemples pour vos applications !*

En l'occurrence je pense au retour air/sol de la pression atmosphérique et de la température à l'aide du baromètre **BMP180** et de la classe **Bmp180.h**, de l'utilisation des gyroscopes **MPU6050** et **BNO055** à l'aide des classes **Mpu6050.h** et **Bno055.h**, de l'ajout de trims à l'émetteur, d'un écran de contrôle, d'une alarme batterie faible, d'une temporisation, de réglages divers (exponentiel, double débattements), etc...