# Communications sans fil

## Réseaux sans fils

L'objectif des liaisons sans fils est de remplacer les fils par des ondes.

- Avantages coût, mobilité, flexibilité, ...
- Inconvenients sécurité, interférences, performance, ...

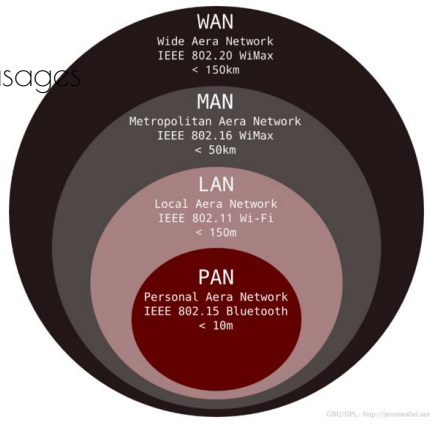
On distingue les réseaux en fonction des usages

**WAN** Wide Area Network (World Wide)
→ GSM, GPRS, UMTS(3G), ...

**MAN** Metropolitan Area Network (50km)
→ WiMax, LoRaWan

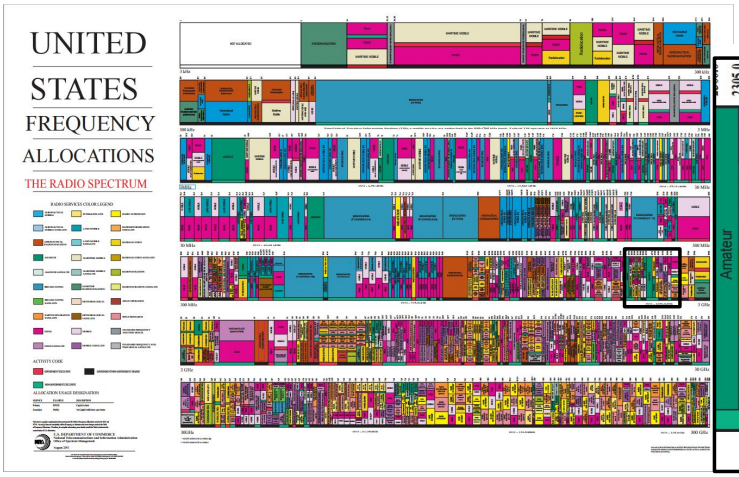
**LAN** Local Area Network (100m)
→ Wifi, ZigBee,

**PAN** Personnal Area Network (10m)
→ Bluetooth, ZigBee

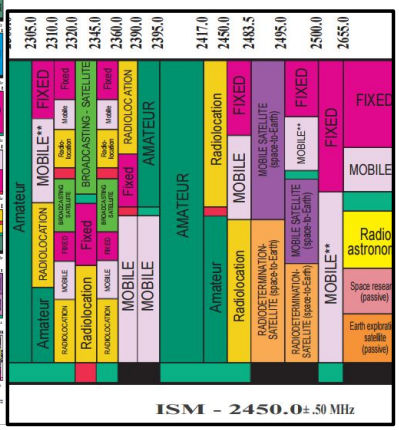


# Usage des fréquences

Les fréquences utilisées pour la communication sont très règlementées



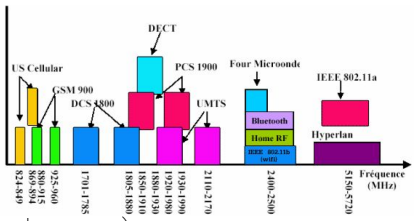

source : wikipedia




# Bandes ISM

Fréquences **libres** pour les usages
**I**ndustriels **S**cientifiques et **M**édicaux

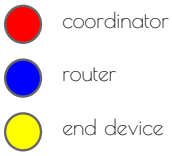
- pas de demandes aux autorités pour les usages
- niveau de puissance limité
- fréquences différentes suivant les continents (voire les pays)



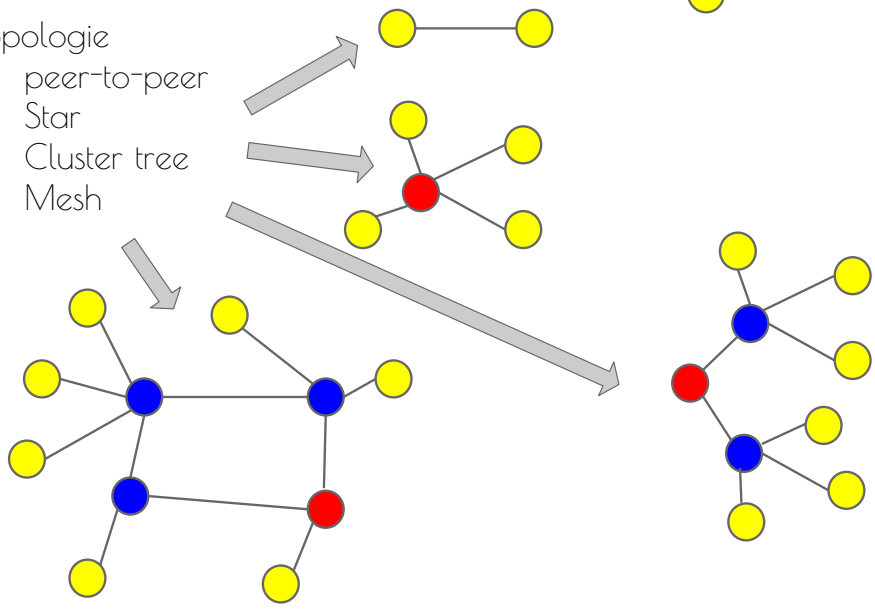

| Frequency range | | Bandwidth | Center frequency | Availability |
|---|---|---|---|---|
| 6.765 MHz | 6.795 MHz | 30 kHz | 6.780 MHz | Subject to local acceptance |
| 13.553 MHz | 13.567 MHz | 14 kHz | 13.560 MHz | Worldwide |
| 26.957 MHz | 27.283 MHz | 326 kHz | 27.120 MHz | Worldwide |
| 40.660 MHz | 40.700 MHz | 40 kHz | 40.680 MHz | Worldwide |
| 433.050 MHz | 434.790 MHz | 1.74 MHz | 433.920 MHz | Region 1 only and subject to local acceptance (within the amateur radio 70 cm band) |
| 902.000 MHz | 928.000 MHz | 26 MHz | 915.000 MHz | Region 2 only (with some exceptions) |
| 2.400 GHz | 2.500 GHz | 100 MHz | 2.450 GHz | Worldwide |
| 5.725 GHz | 5.875 GHz | 150 MHz | 5.800 GHz | Worldwide |
| 24.000 GHz | 24.250 GHz | 250 MHz | 24.125 GHz | Worldwide |
| 61.000 GHz | 61.500 GHz | 500 MHz | 61.250 GHz | Subject to local acceptance |
| 122.000 GHz | 123.000 GHz | 1 GHz | 122.500 GHz | Subject to local acceptance |
| 244.000 GHz | 246.000 GHz | 2 GHz | 245.000 GHz | Subject to local acceptance |

"Radio Regulations, Edition of 2012".

# Types de réseau




- Topologie
  - peer-to-peer
  - Star
  - Cluster tree
  - Mesh



# Solutions grand public

### ZigBee → réseau d'objets

- protocole simple (code 4ko à 32ko)
- 64 kilo noeuds : star, cluster, mesh
- autonomie plusieurs années
- 250kb/s
- portée 100m

### Bluetooth → remplacement de câbles

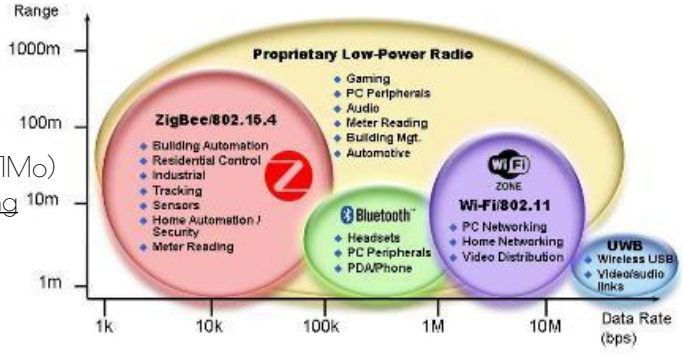
- protocole complexe (code 250ko)
- 7 noeuds actifs : star, scatternet
- autonomie en semaines
- 0.7 à 2 Mb/s
- portée 10m

### Wi-Fi → internet

- protocole très complexe (1Mo)
- 32 noeuds : star + roaming
- autonomie en heures
- 11Mb/s à 50Mb/s
- portée 100m

### Bluetooth Low Energy → IOT

- protocole complexe (code 250ko)
- autonomie en années
- P-à-P ou Star nbre de noeuds illimité
- découverte de services
- 300 kb/s
- portée 10m à 100m



# Cahier des charges

Communications locales Raspberry Pi ⟷ capteur/actionneur

- Faible coût
  → Quelques euros par noeud
- Basse consommation
  → Fonctionnement sur batterie
- Faible latence - Faible débit moyen
  → (re)connexion rapide d'un noeud
  → Le débit n'est pas une contrainte
- Simplicité de programmation
  → Le capteur ou l'actionneur doit être facilement programmable
- Mise en réseau
  → L'accès au capteur à l'actionneur peut passer par des routeurs

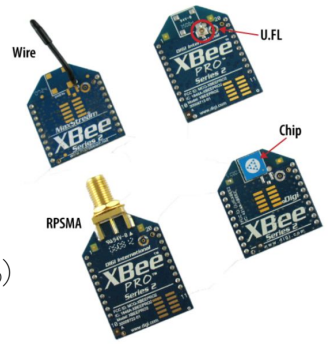
# Présentation ZigBee - XBee

ZigBee® Alliance

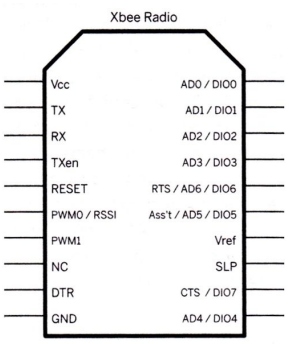
- Protocole de communication à très basse consommation pour les réseaux de dimension personnelle en radio fréquence
- (Wireless Personal Area Networks : WPAN).
- Norme IEEE 802.15.4 (2003) : 868MHz - 915MHz - 2.4GHz
- ZigBee Alliance (2004)

XBee™

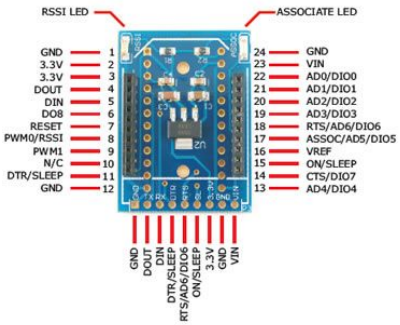
- Modules fabriqués par l'entreprise Digi International
- 2 types de modèles
  - 1 / 2 mW **XBee** (20$)
  - 100 / 500 mW **XBee-PRO** (30$)



# Connectique Xbee

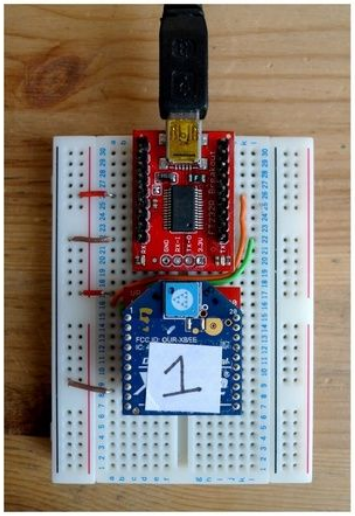


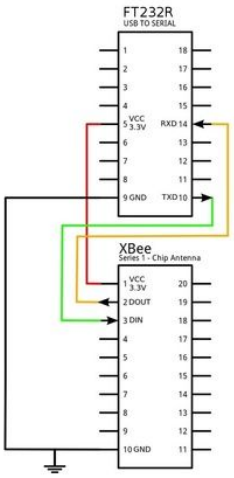

- Protocole rs232 de **9600** à 115200 bauds
- 4 fils (minimum)
  - TX (data out)
  - RX (data in)
  - Vcc (2.8 à 3.6V)
  - GND
- ports IO Digi/Ana/PWM
- Adaptateur (exemple)

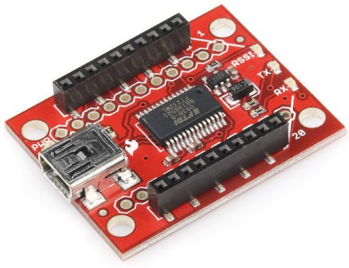



| Pin # | Name | Direction | Description |
|---|---|---|---|
| 1 | VCC | - | Power supply |
| 2 | DOUT | Output | UART Data Out |
| 3 | DIN / **CONFIG** | Input | UART Data In |
| 4 | DO8* | Output | Digital Output 8 |
| 5 | **RESET** | Input | Module Reset (reset pulse must be at least 200 ns) |
| 6 | PWM0 / RSSI | Output | PWM Output 0 / RX Signal Strength Indicator |
| 7 | PWM1 | Output | PWM Output 1 |
| 8 | [reserved] | - | Do not connect |
| 9 | $\overline{\text{DTR}}$ / SLEEP_RQ / DI8 | Input | Pin Sleep Control Line or Digital Input 8 |
| 10 | GND | - | Ground |
| 11 | AD4 / DIO4 | Either | Analog Input 4 or Digital I/O 4 |
| 12 | **CTS** / DIO7 | Either | Clear-to-Send Flow Control or Digital I/O 7 |
| 13 | ON / **SLEEP** | Output | Module Status Indicator |
| 14 | VREF | Input | Voltage Reference for A/D Inputs |
| 15 | Associate / AD5 / DIO5 | Either | Associated Indicator, Analog Input 5 or Digital I/O 5 |
| 16 | **RTS** / AD6 / DIO6 | Either | Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6 |
| 17 | AD3 / DIO3 | Either | Analog Input 3 or Digital I/O 3 |
| 18 | AD2 / DIO2 | Either | Analog Input 2 or Digital I/O 2 |
| 19 | AD1 / DIO1 | Either | Analog Input 1 or Digital I/O 1 |
| 20 | AD0 / DIO0 | Either | Analog Input 0 or Digital I/O 0 |

# Commandes par USB



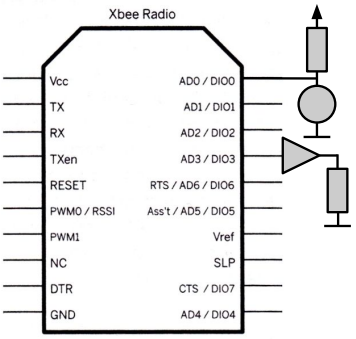



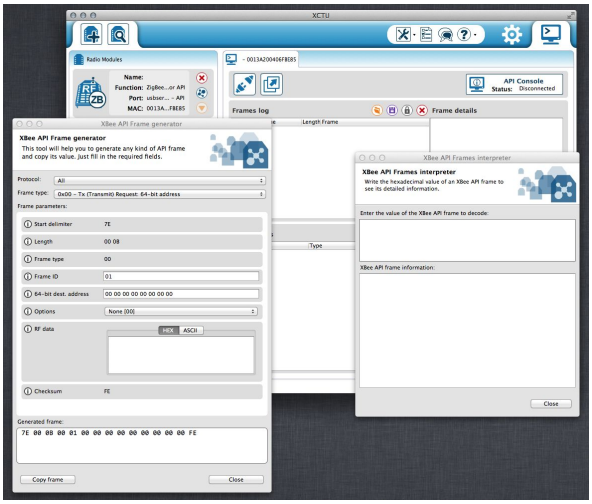



# XCTU : Configuration des nœuds

Logiciel XCTU de Digi International (windows / mac)
permet de le commander et de changer le firmware :
Coordinateur, router, end-device et programmation des ports

Programmation par XCTU
des end-devices pour
demander la lecture
périodique de capteurs
et la commandes de leds




# Commandes AT

- Etablir une communication rs232 9600bauds avec le module
  - Taper +++ → attendre OK *// mode commande*
  - ATMY1234 → attendre OK *// programmation de l'adresse*
  - ATMY → 1234 *// vérification de la prog.*

- Commandes AT (un document de référence est cité à la fin)

| Commande AT | Nom | Description | Exemple |
|---|---|---|---|
| ATCN | Command end | Fin du mode commande | ATCN |
| ATND | Node Discover | Découvrir les XBee présents sur le réseau | ATND |
| ATID | Pan ID | Définir le numéro d'identification du réseau | ATID 35 |
| | | Lire le numéro d'identification du réseau | ATID |
| ATNI | Node Identifier | Définir le nom de nœud | ATNI E35_Ordi |
| | | Lire le nom de nœud | ATNI |
| ATWR | Write | Sauvegarder la configuration en mémoire non-volatile. Permet de conserver la configuration même si le XBee n'est plus alimenté. | ATWR |

# Frame maker de Digi Internationnal

Frame Type: 0x17 Remote AT Command | API Mode: API 2 | Device Type: ZigBee

| FieldName | FieldValue | DataType | Description |
|---|---|---|---|
| Delimiter | 7E | Byte | Start Delimiter |
| Length | 0014 | Word | Number of bytes between length and checksum fields. |
| API | 17 | Byte | Remote AT Command |
| FrameID | 01 | Byte | Identifies the UART data frame for the host to match with a subsequent response. If zero, no response is requested. |
| 64DestAddr | 0013A200606FBD6F | EUI64 | Destination 64-bit (MAC/EUI64) address.<br>The following addresses are also supported:<br>0x0000000000000000 - Reserved for the coordinator.<br>0x000000000000FFFF - Broadcast address |
| 16DestAddr | FFFE | NWK16 | Destination 16-bit network address, if known.<br>Use 0xFFFE if the address is unknown, or if sending a broadcast.<br>Other reserved addresses:<br>0xFFFC - broadcast to all routers;<br>0xFFFD - broadcast to all non-sleepy devices;<br>0xFFFF - broadcast to all devices including sleepy ED. |
| CmdOptions | 02 | Byte | 0x02 - Apply changes on remote device.<br>*NOTE: If this bit is not set, an AC (or WR+FR) command must be sent before changes will take effect.*<br>All other bits must be set to zero. |
| AT Cmd | D3 | ATCmd | Command name of two ASCII characters. |
| AT CmdData | 5 | Variable | If present, set the register to this value.<br>If absent, get the value of the register.<br>String values should be terminated with a zero byte. |
| Checksum | bc | Byte | 0xFF minus 8-bit sum of bytes between the length and checksum fields. |
| Packet | 7E 00 10 17 01 00 7D 33 A2 00 60 6F BD 6F FF FE 02 44 33 05 BC | Build | |

message utile 2 octets → 21 octets

# Sources Xbee - ZigBee

## Sites officiels ou cours

- www.zigbee.org
- Zigbee overview
- wikipedia Zigbee
- Zigbee Standard
- cours routage Zigbee
- Digi Zigbee Mesh networking (ppt)
- commandes AT
- frame maker de digi

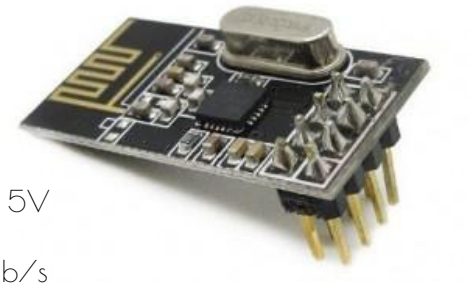
## Sites persos

- blog découverte Zigbee avec détail d'une frame
- présentation détaillée Zigbee avec usage par arduino
- commandes AT et arduino
- exemple de frame XBee
- club électronique utilisant XBee
- exemple usage XBee serie 1
- présentation XBee serie 1

# Présentation nRF24L01+

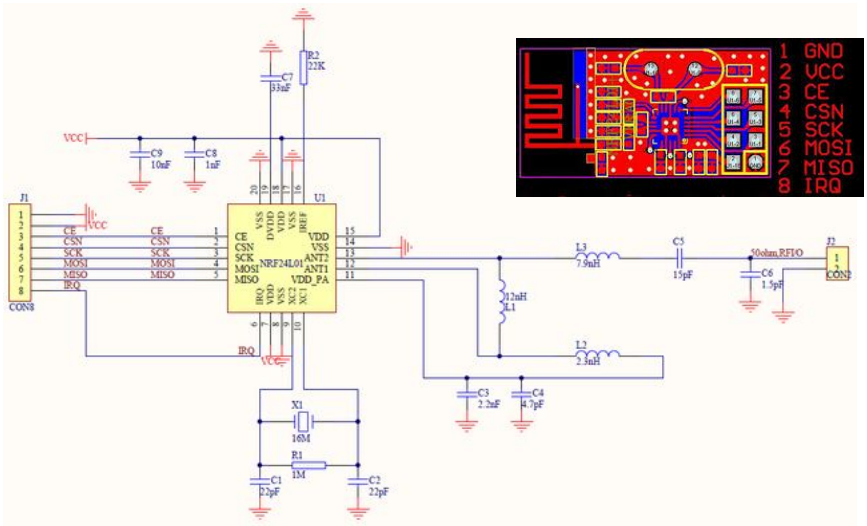
nRF24L01P_Product_Specification_1_0.pdf

- protocole très simple (qq ko)
- intègre un 8051 µC 8bits 2kB + 256B RAM + 16kB flash
- bande 2.4 à 2.525 GHz
- 125 canaux
- 6 noeuds : star
- 250kb/s à 2Mb/s
- portée 50m (10m in 70m out)
- 4 puissance réglable 0, -6, -12, -18db
- alimentation de 1.9 à 3.6V
  - mais 2.7 à 3.3V pour être tolérant 5V
- Ultra low power
  11.3mA Tx pour 1 mW  13.4mA RX à 2Mb/s
  26 µA standby-I and 900 nA  power down
  autonomie plusieurs années
- Interface SPI
- prix < 1$

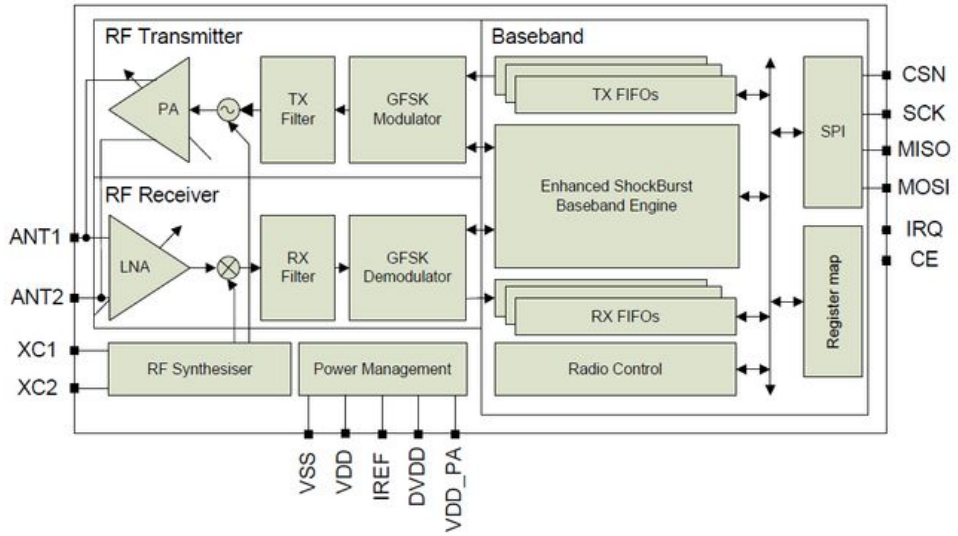


# Interface nRF24L01

La communication avec le composant utilise le protocole SPI



# Block Diagram




# Enhanced ShockBurst

Protocole de communication de paquets

- 1 à 32 octets
- Gestion automatique
  - de la construction de paquet
  - de l'acquittement
  - de la retransmission
- jusqu'à 6 receveurs pour des réseaux STAR 1:6

PTX ⮌ PRX (Primary TX to/from Primary RX)

1. PTX envoie un paquet à PRX □ PTX se met en écoute de l'acquittement.
2. PRX reçoit le paquet, envoit l'acquittement, et revient en écoute.
3. si PTX n'a pas reçu l'acquittement, il attend un delai et recommence 1.

C'est une procédure automatique sans intervention du µC,
le nombre d'essais est programmable

# Enhanced Packet format

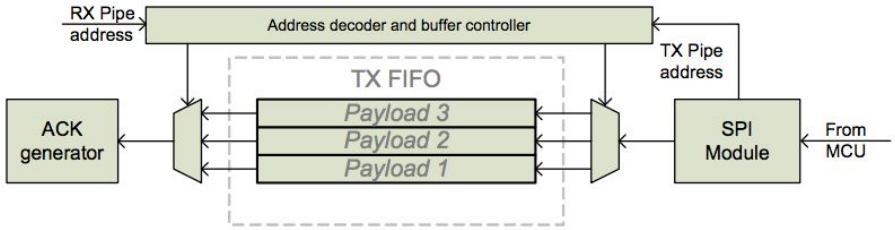
| Preamble 1 byte | Address 3-5 byte | Packet Control Field 9 bit | Payload 0 - 32 byte | CRC 1-2 byte |
|---|---|---|---|---|

| | |
|---|---|
| Packet Control Field | 6 bits longueur 0 à 32, 33 pour une longueur variable<br>2 bits PID Packet IDentity (+1 à chaque nouveau packet)<br>1 bits NO_ACK |
| Payload | packet |
| CRC | 1 ou 2 octets CRC |

| | | | charge utile |
|---|---|---|---|
| pour 4 octets (32bits) | → | 1+3+1+4+1 = 10 octets (81bits) | 40% |
| pour 8 octets (64bits) | → | 1+3+1+8+1 = 14 octets (113bits) | 56% |
| pour 32 octets (256bits) | → | 1+3+1+32+2 = 39 octets (313bits) | 81% |

# Gestion de transaction automatique

- Acquittement automatique

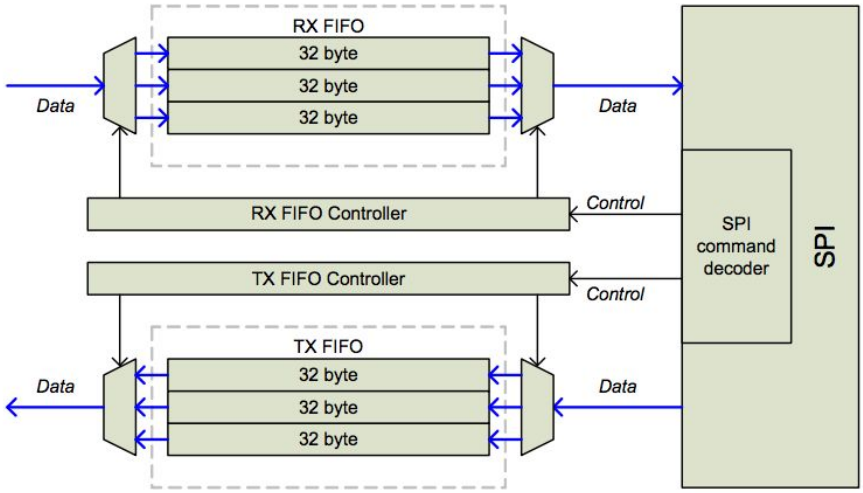



  - PRX charge un message d'acquittement (jusqu'à 3 en avances) en attente d'un paquet de PTX

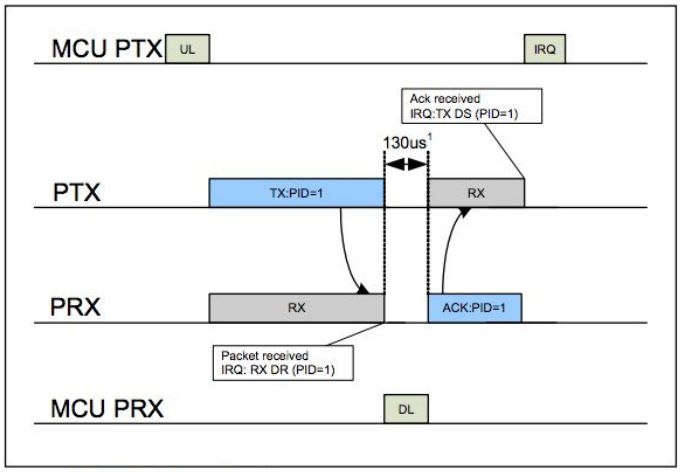
- Auto-retransmission en cas acquittement demandé pas bien reçu
  Le nombre de retransmission est borné
  - PTX à trop attendu, le délai dépend de la taille du paquet et du débit
  - PTX n'a pas reçu de paquet avec la bonne adresse
  - le CRC était faux

# Fifos TX et RX

Ces fifos permettent de réduire les latences entre paquets

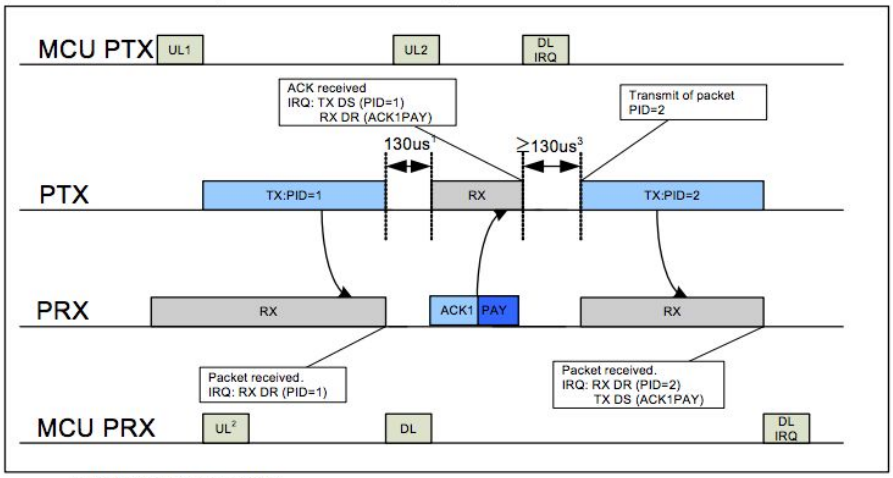



# Transaction simple



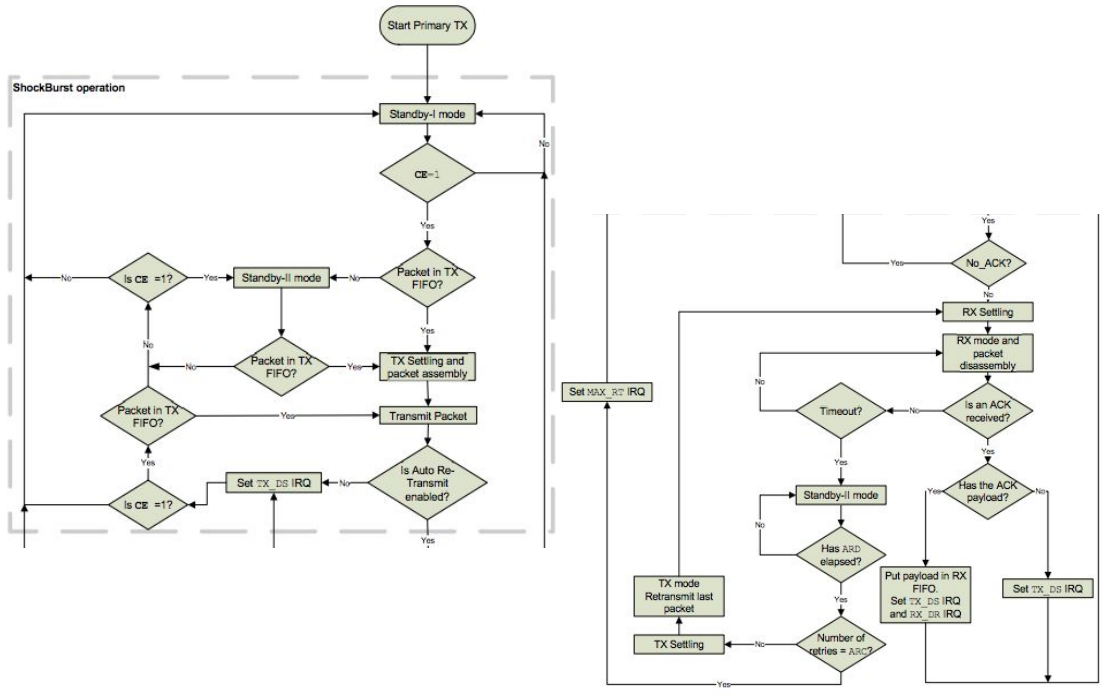

1 Radio Turn Around Delay

# Transaction avec donnée en retour

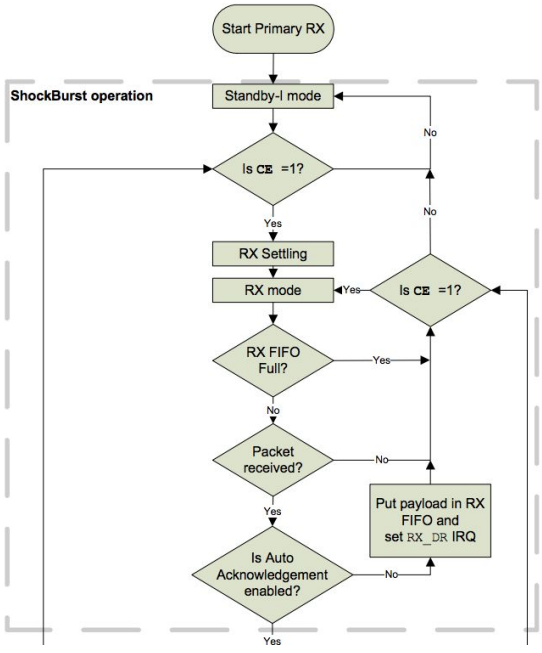


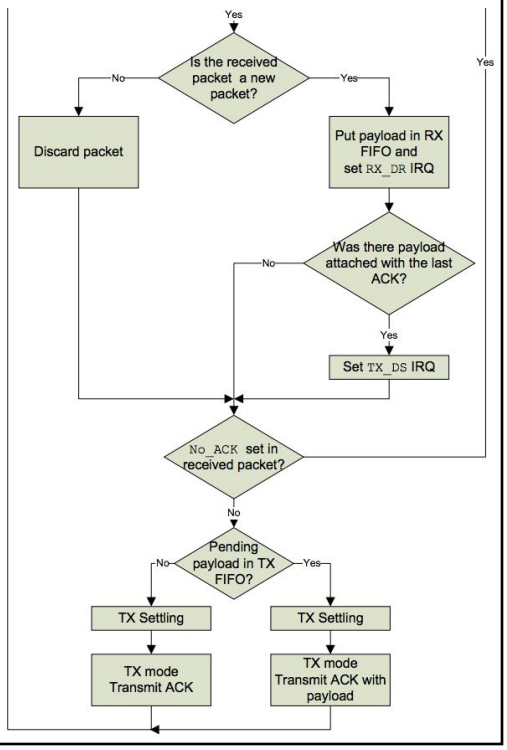

1 Radio Turn Around Delay
2 Uploading Payload for Ack Packet
3 Delay defined by MCU on PTX side, ≥ 130us

# Fonctionnement PTX



# Fonctionnement PRX



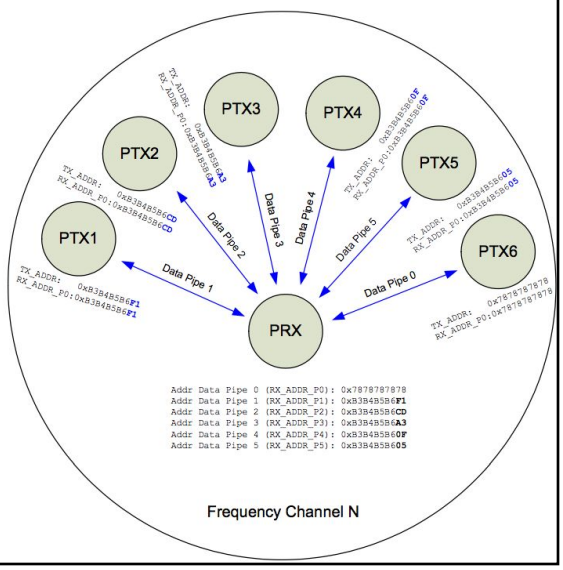




# MultiCeiver

Un nRF24L01+ est configuré comme un PRX et

- Select sur 6 PTX différents mais un seul PTX à la fois
- Chaque PTX utilise un Data Pipe mais ils partagent :
  - CRC mode
  - taille de l'adresse 3 à 5 bytes
  - fréquence
  - puissance
  - débit 250kb, 1Mb, 2Mb
- L'adresse du pipe0 est unique les autres partagent les MSB

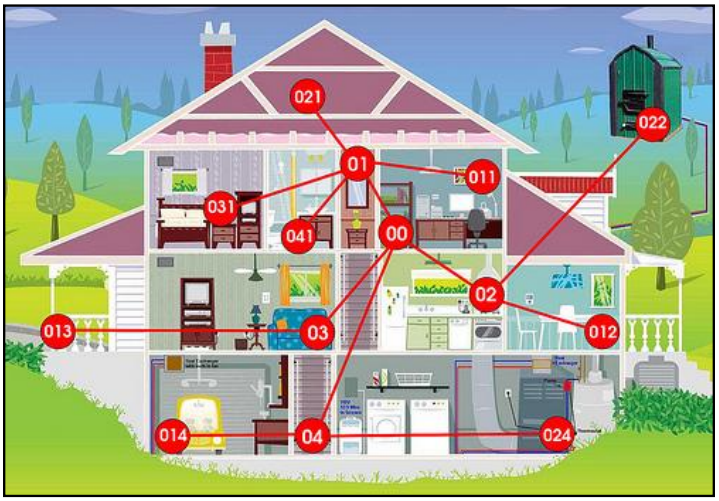
| | Byte 4 | Byte 3 | Byte 2 | Byte 1 | Byte 0 |
|---|---|---|---|---|---|
| Data pipe 0 (RX_ADDR_P0) | 0xE7 | 0xD3 | 0xF0 | 0x35 | 0x77 |
| Data pipe 1 (RX_ADDR_P1) | 0xC2 | 0xC2 | 0xC2 | 0xC2 | **0xC2** |
| Data pipe 2 (RX_ADDR_P2) | 0xC2 | 0xC2 | 0xC2 | 0xC2 | **0xC3** |
| Data pipe 3 (RX_ADDR_P3) | 0xC2 | 0xC2 | 0xC2 | 0xC2 | **0xC4** |
| Data pipe 4 (RX_ADDR_P4) | 0xC2 | 0xC2 | 0xC2 | 0xC2 | **0xC5** |
| Data pipe 5 (RX_ADDR_P5) | 0xC2 | 0xC2 | 0xC2 | 0xC2 | **0xC6** |



# Réseau en étoile

Par défaut, tous les noeuds écoutent (sauf les feuilles) et quand un noeud reçoit un message il n'analyse et le transmet, une seule route possible

https://maniacbug.wordpress.com/2012/03/30/rf24network/

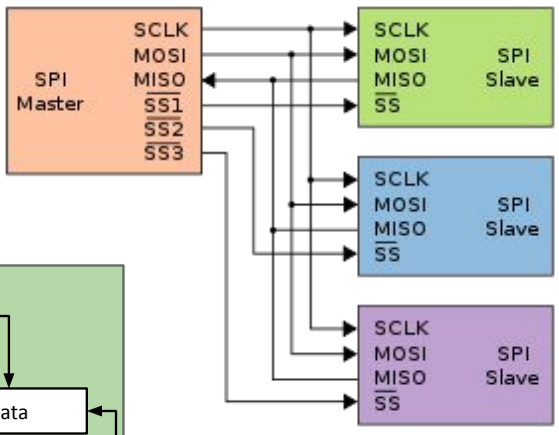


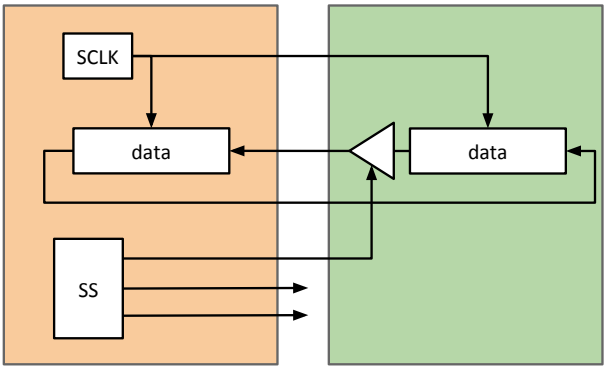

# Protocole SPI

wikipedia

SPI : Serial Peripheral Interface

- full duplex
- 1 maitre - N esclave
  - le maitre choisit l'esclave
  - le maitre impose l'horloge



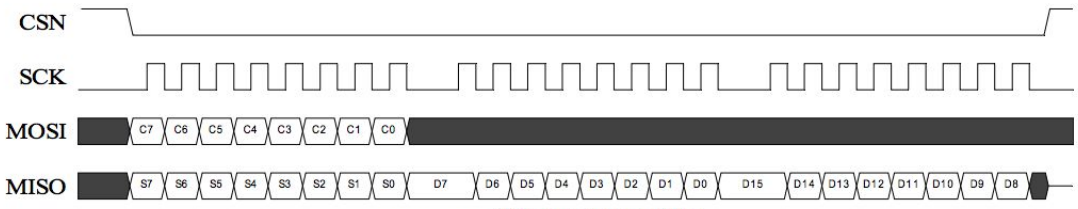




- débit variable → 100Mb/s
- data variable 8 à n bits

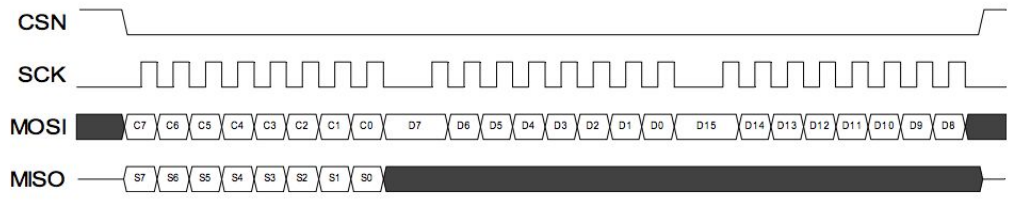
# Interface SPI

- **GND** Ground.
- **VCC** 3.3V.
- **CE** Chip (RX/TX) Enable, actif à l'état haut (RADIO)
  Si actif, le module envoie ou recoit
- **CSN** Chip Select Not, active à l'état bas (SPI Select)
  Si actif, le circuit communique en SPI
- **SCK** SPI Shift Clock jusqu'à 10 MHz.
- **MOSI** Master-Out-Slave-In microcontrôleur → nRF24
- **MISO** Master-In-Slave-Out nRF24 → microcontrôleur
- **IRQ** Interrupt Request
  Signale qu'un paquet à été reçu, émis ou à échoué.

# Chronogrammes SPI




*SPI read operation*




*SPI write operation*

| Abbreviation | Description |
|---|---|
| Cn | SPI command bit |
| Sn | `STATUS` register bit |
| Dn | Data Bit (**Note:** LSByte to MSByte, MSBit in each byte first) |

Le registre status est rendu

| Command name | Command word (binary) | # Data bytes | Operation |
|---|---|---|---|
| R_REGISTER | 000A AAAA | 1 to 5 LSByte first | Read command and status registers. AAAAA = 5 bit Register Map Address |
| W_REGISTER | 001A AAAA | 1 to 5 LSByte first | Write command and status registers. AAAAA = 5 bit Register Map Address<br>Executable in power down or standby modes only. |
| R_RX_PAYLOAD | 0110 0001 | 1 to 32 LSByte first | Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode. |
| W_TX_PAYLOAD | 1010 0000 | 1 to 32 LSByte first | Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload. |
| FLUSH_TX | 1110 0001 | 0 | Flush TX FIFO, used in TX mode |
| FLUSH_RX | 1110 0010 | 0 | Flush RX FIFO, used in RX mode<br>Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed. |
| REUSE_TX_PL | 1110 0011 | 0 | Used for a PTX device<br>Reuse last transmitted payload.<br>TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deacti-vated during package transmission. |
| R_RX_PL_WID[a] | 0110 0000 | 1 | Read RX payload width for the top R_RX_PAYLOAD in the RX FIFO.<br><br>**Note:** Flush RX FIFO if the read value is larger than 32 bytes. |
| W_ACK_PAYLOAD[a] | 1010 1PPP | 1 to 32 LSByte first | Used in RX mode.<br>Write Payload to be transmitted together with ACK packet on PIPE PPP. (PPP valid in the range from 000 to 101). Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled using first in - first out principle. Write payload: 1– 32 bytes. A write operation always starts at byte 0. |
| W_TX_PAYLOAD_NO ACK[a] | 1011 0000 | 1 to 32 LSByte first | Used in TX mode. Disables AUTOACK on this specific packet. |
| NOP | 1111 1111 | 0 | No Operation. Might be used to read the STATUS register |

Il y a 28 registres

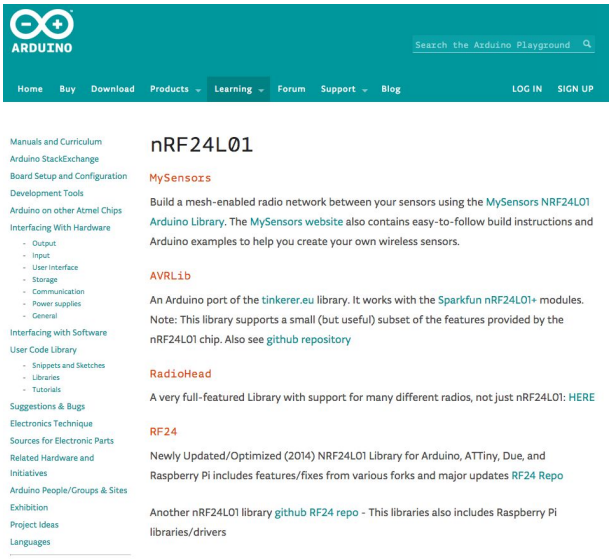
| Address (Hex) | Mnemonic | Bit | Reset Value | Type | Description |
|---|---|---|---|---|---|
| | | | | | |
| 00 | CONFIG | | | | Configuration Register |
| | Reserved | 7 | 0 | R/W | Only '0' allowed |
| | MASK_RX_DR | 6 | 0 | R/W | Mask interrupt caused by RX_DR<br>1: Interrupt not reflected on the IRQ pin<br>0: Reflect RX_DR as active low interrupt on the IRQ pin |
| | MASK_TX_DS | 5 | 0 | R/W | Mask interrupt caused by TX_DS<br>1: Interrupt not reflected on the IRQ pin<br>0: Reflect TX_DS as active low interrupt on the IRQ pin |
| | MASK_MAX_RT | 4 | 0 | R/W | Mask interrupt caused by MAX_RT<br>1: Interrupt not reflected on the IRQ pin<br>0: Reflect MAX_RT as active low interrupt on the IRQ pin |
| | EN_CRC | 3 | 1 | R/W | Enable CRC. Forced high if one of the bits in the EN_AA is high |
| | CRCO | 2 | 0 | R/W | CRC encoding scheme<br>'0' - 1 byte<br>'1' – 2 bytes |
| | PWR_UP | 1 | 0 | R/W | 1: POWER UP, 0:POWER DOWN |
| | PRIM_RX | 0 | 0 | R/W | RX/TX control<br>1: PRX, 0: PTX |
| | | | | | |
| 01 | EN_AA Enhanced ShockBurst™ | | | | Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 75 |
| | Reserved | 7:6 | 00 | R/W | Only '00' allowed |
| | ENAA_P5 | 5 | 1 | R/W | Enable auto acknowledgement data pipe 5 |
| | ENAA_P4 | 4 | 1 | R/W | Enable auto acknowledgement data pipe 4 |
| | ENAA_P3 | 3 | 1 | R/W | Enable auto acknowledgement data pipe 3 |
| | ENAA_P2 | 2 | 1 | R/W | Enable auto acknowledgement data pipe 2 |
| | ENAA_P1 | 1 | 1 | R/W | Enable auto acknowledgement data pipe 1 |
| | ENAA_P0 | 0 | 1 | R/W | Enable auto acknowledgement data pipe 0 |
| | | | | | |
| 02 | EN_RXADDR | | | | Enabled RX Addresses |
| | Reserved | 7:6 | 00 | R/W | Only '00' allowed |
| | ERX_P5 | 5 | 0 | R/W | Enable data pipe 5. |
| | ERX_P4 | 4 | 0 | R/W | Enable data pipe 4. |
| | ERX_P3 | 3 | 0 | R/W | Enable data pipe 3. |
| | ERX_P2 | 2 | 0 | R/W | Enable data pipe 2. |
| | ERX_P1 | 1 | 1 | R/W | Enable data pipe 1. |
| | ERX_P0 | 0 | 1 | R/W | Enable data pipe 0. |
| | | | | | |

# Registre Status

| 07 | STATUS | | | | Status Register (In parallel to the SPI command word applied on the **MOSI** pin, the STATUS register is shifted serially out on the **MISO** pin) |
|---|---|---|---|---|---|
| | Reserved | 7 | 0 | R/W | Only '0' allowed |
| | RX_DR | 6 | 0 | R/W | Data Ready RX FIFO interrupt. Asserted when new data arrives RX FIFO[c].<br>Write 1 to clear bit. |
| | TX_DS | 5 | 0 | R/W | Data Sent TX FIFO interrupt. Asserted when packet transmitted on TX. If AUTO_ACK is activated, this bit is set high only when ACK is received.<br>Write 1 to clear bit. |
| | MAX_RT | 4 | 0 | R/W | Maximum number of TX retransmits interrupt<br>Write 1 to clear bit. If MAX_RT is asserted it must be cleared to enable further communication. |
| | RX_P_NO | 3:1 | 111 | R | Data pipe number for the payload available for reading from RX_FIFO<br>000-101: Data Pipe Number<br>110: Not Used<br>111: RX FIFO Empty |
| | TX_FULL | 0 | 0 | R | TX FIFO full flag.<br>1: TX FIFO full.<br>0: Available locations in TX FIFO. |

# nRF24L01 et Arduino

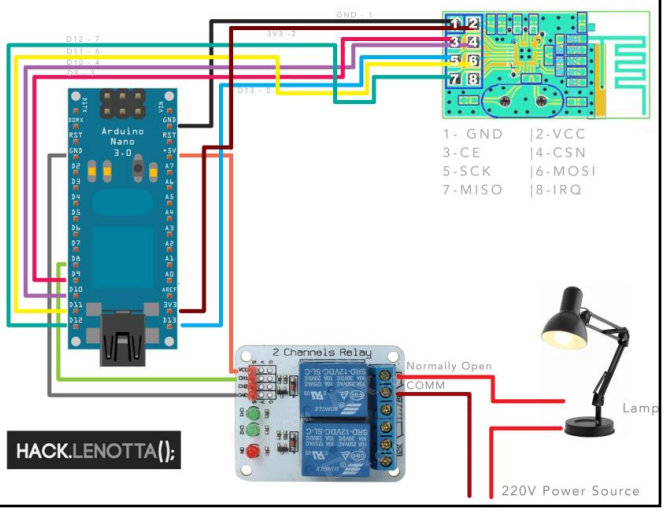
http://playground.arduino.cc/InterfacingWithHardware/Nrf24L01




et même http://tmrh20.github.io/RF24Ethernet/

# RF24 lib

- Communication point-à-point
- Abstrait les échanges entre le µC et le nRF24L01
- Paramétrage
  - canal
  - taille des données
  - acquittement oui/non
  - latence de réémission
  - nombre de tentative
  - taille du CRC




# RF24 fonctions principales

http://maniacbug.github.io/RF24/classRF24.html

**Public Member Functions**

**Primary public interface**
*These are the main methods you need to operate the chip*

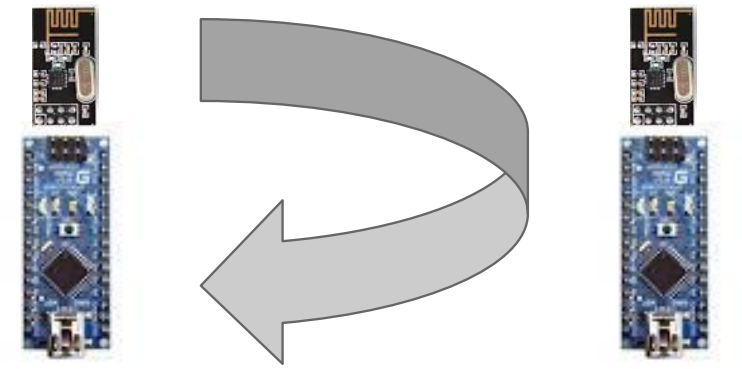
| | |
|---|---|
| `RF24 (uint8_t _cepin, uint8_t _cspin)` | Constructor. |
| `void begin (void)` | Begin operation of the chip. |
| `void startListening (void)` | Start listening on the pipes opened for reading. |
| `void stopListening (void)` | Stop listening for incoming messages. |
| `bool write(const void *buf, uint8_t len)` | Write to the open writing pipe. |
| `bool available (void)` | Test whether there are bytes available to be read. |
| `bool read (void *buf, uint8_t len)` | Read the payload. |
| `void openWritingPipe (uint64_t address)` | Open a pipe for writing. |
| `void openReadingPipe (uint8_t number, uint64_t address)` | Open a pipe for reading. |

# Exemple un ping

PTX envoie un timestamp à PRX

PRX envoie son timestamp en retour à PTX

PTX calcule la durée d'aller-retour



# PING-PONG

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"

RF24 radio(9,10);
const int role_pin = 7;
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
typedef enum { role_ping_out = 1, role_pong_back } role_e;
const char* role_friendly_name[] = { "invalid", "Ping out", "Pong back"};
role_e role;

void setup(void)
{
  pinMode(role_pin, INPUT);
  digitalWrite(role_pin,HIGH);
  delay(20);
  if ( ! digitalRead(role_pin) )
    role = role_ping_out;
  else
    role = role_pong_back;

  Serial.begin(57600);
  printf_begin();
  printf("\n\rRF24/examples/pingpair/\n\r");
  printf("ROLE: %s\n\r",role_friendly_name[role]);
  radio.begin();
  radio.setRetries(15,15);
  radio.setPayloadSize(8);

  if ( role == role_ping_out ) {
    radio.openWritingPipe(pipes[0]);
    radio.openReadingPipe(1,pipes[1]);
  } else {
    radio.openWritingPipe(pipes[1]);
    radio.openReadingPipe(1,pipes[0]);
  }
  radio.startListening();
  radio.printDetails();
}
```

```
void loop(void)
{
  if (role == role_ping_out) {
    radio.stopListening();
    unsigned long time = millis();
    printf("Now sending %lu...",time);
    bool ok = radio.write( &time, sizeof(unsigned long) );

    if (ok)
      printf("ok...");
    else
      printf("failed.\n\r");
    radio.startListening();
    unsigned long started_waiting_at = millis();
    bool timeout = false;
    while ( ! radio.available() && ! timeout )
      if (millis() - started_waiting_at > 200 )
        timeout = true;
    if ( timeout )   {
      printf("Failed, response timed out.\n\r");
    }  else {
      unsigned long got_time;
      radio.read( &got_time, sizeof(unsigned long) );
      printf("Got response %lu, round-trip delay:%lu\n\r",
             got_time,millis()-got_time);
    }
    delay(1000);
  }
  if ( role == role_pong_back ) {
    if ( radio.available() ) {
      unsigned long got_time;
      bool done = false;
      while (!done)  {
        done = radio.read( &got_time, sizeof(unsigned long) );
        printf("Got payload %lu...",got_time);
        delay(20);
      }
      radio.stopListening();
      radio.write( &got_time, sizeof(unsigned long) );
      printf("Sent response.\n\r");
      radio.startListening();
    }
  }
}
```