XBee

Table des matières

"XBee" est une famille de composants sans-fil prêts à l'emploi développés qui implémentent différents protocoles, dont 802.15.4 et sa version de plus haut-niveau ZigBee, ainsi que des protocoles de réseau ad-hoc ("mesh") spécifique.

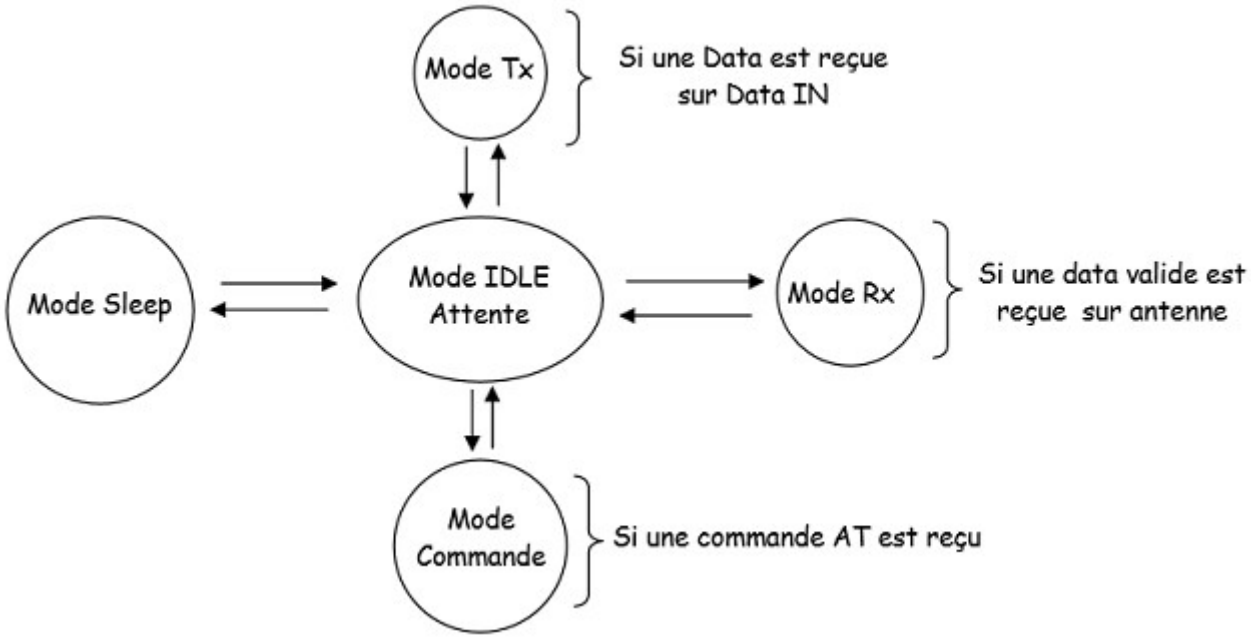
1. Présentation du XBee

Les produits XBee sont des modules de communication sans fil certifiés par la communauté industrielle ZigBee Alliance. La certification Zigbee se base sur le standard IEEE 802.15.4 qui définit les fonctionnalités et spécifications des réseaux sans fil à dimension personnelle (Wireless Personal Area Networks : WPANs).

Les principales caractéristiques du XBee :

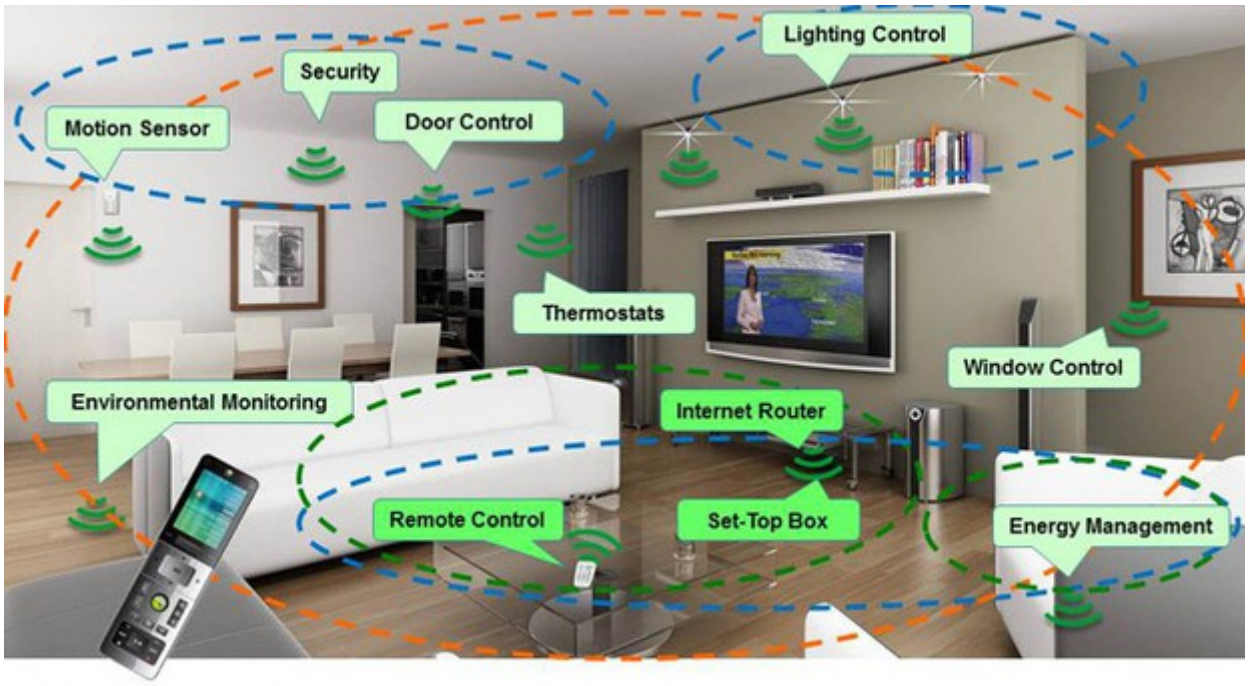
- fréquence porteuse : 2.4Ghz
- portées variées : assez faible pour les XBee 1 et 2 (10 - 100m), grande pour le XBee Pro (1000m)
- faible débit : 250kbps
- faible consommation : 3.3V @ 50mA (inférieure à 10 µA en mode "sleep").
- entrées/sorties : 6 10-bit ADC input pins, 8 digital IO pins
- sécurité : communication fiable avec une clé de chiffrement de 128-bits
- faible coût : ~ 25€
- simplicité d'utilisation : communication via le port série
- ensemble de commandes AT et API
- flexibilité du réseau : sa capacité à faire face à un nœud hors service ou à intégrer de nouveaux nœuds rapidement
- grand nombre de nœuds dans le réseau : 65000
- topologies de réseaux variées : maillé, point à point, point à multipoint

1.1. Modes de fonctionnement du module XBee



1.2. Applications

Le ZigBee a été conçu pour réaliser l'Internet des objets, un ensemble d'objets communicants voir "autonomes", une extension d'Internet aux objets physiques. La domotique est l'exemple le plus parlant.



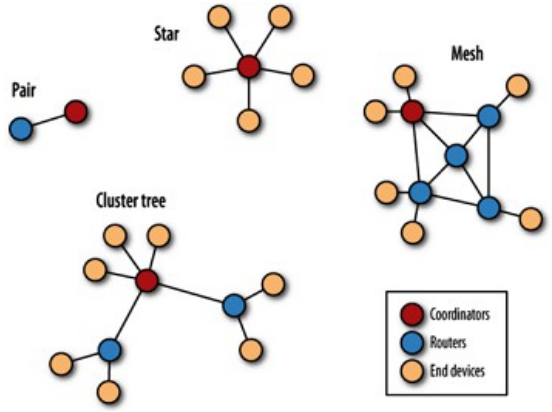
1.3. Réseaux Zigbee

Un réseau Zigbee est composé de trois types de nœuds :

- Nœud de type « end-device » (équipement RFD : Reduced Function Device). II s'agit de nœuds simples réalisant une seule fonction : allumage d'une lampe, d'un appareil de détection ...
- Nœud de type « routeur » (équipement FFD : Full Function Device). Ces noeuds permettent la transmission de messages. Ils sont indispensables pour étendre le réseau par acheminements des trames d'un noeuds à un autre. Ils permettent aussi aux autres modules de s'enregistrer sur le même réseau, et non exclusivement chez le coordinateur.
- Nœud de type « coordinateur ». Ce type nœud unique dans un réseau assure les fonctions telles que l'authentification, l'initiation de la communication, la sécurité et l'ajout des noeuds au réseau ... II doit être actif en permanence pour répondre à tout moment aux requêtes des autres éléments du réseau. II est donc alimenté à plein temps.

Les réseaux Zigbee peuvent présenter plusieurs types de topologie :

- Point à point (Pair).
- Etoile (Star).
- Arbre (Clustertree).
- Maille (Mesh).



La topologie, la plus utilisée en Zigbee est le réseau maillé (mesh). Dans ce type de réseau, les nœuds sont interconnectés avec d'autres nœuds de sorte que de multiples voies permettent de connecter chaque nœud. Les connexions entre les nœuds sont misent à jour dynamiquement et optimisées par une table de routage intégrée dans la maille.

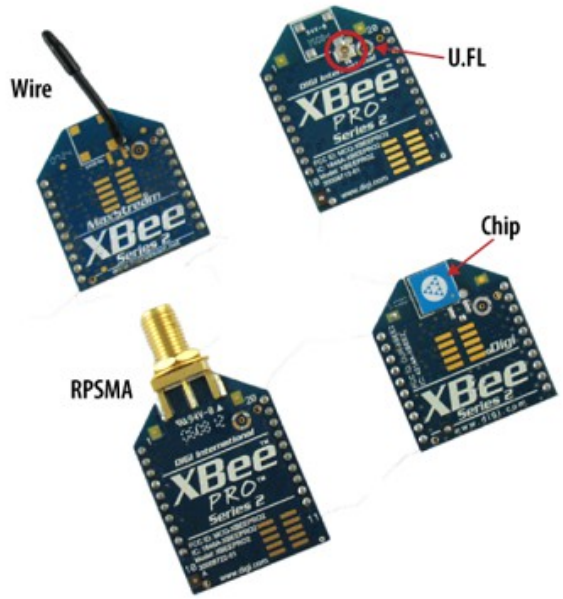
1.4. Séries 1 et 2

Il y a deux catégories de XBee : la série 1 et la série 2. Les modules de la série 1 ont souvent un "802.15.4" qui s'ajoutent à leurs noms. Les modules de la série 2 sont disponibles en plusieurs versions : XBee ZNet 2.5 (obsolète), le ZB (l'actuel) et le 2B (le plus récent). Des XBee Pro font la même chose, mais avec de plus grandes capacités, notamment la portée qui peut aller jusqu'à 1000 mètres ! Voir le tableau de comparaison des modules Xbee.

- les modules des séries 1 et 2 ne sont pas compatibles entre eux ;
- la portée et la consommation sont sensiblement les mêmes ;
- le nombre d'entrées et sorties est différent et surtout la série 2 ne possède pas de sorties analogiques PWM ;
- les topologies de réseaux possibles ne sont pas les mêmes. Avec la série 1, l'architecture est simple : point à point (pair) ou multipoint (star). La série 2 permet en plus de créer des réseaux plus complexes : maillés (mesh) ou en "arbre" (cluster tree).

	Series 1	Series 2
Typical (indoor/urban) range	30 meters	40 meters
Best (line of sight) range	100 meters	120 meters
Transmit/Receive current	45/50 mA	40/40 mA
Firmware (typical)	802.15.4 point-to-point	ZB ZigBee mesh
Digital input/output pins	8 (plus 1 input-only)	11
Analog input pins	7	4
Analog (PWM) output pins	2	None
Low power, low bandwidth, low cost, addressable, standardized, small, popular	Yes	Yes
Interoperable mesh routing, ad hoc network creation, self-healing networks	No	Yes
Point-to-point, star topologies	Yes	Yes
Mesh, cluster tree topologies	No	Yes
Single firmware for all modes	Yes	No
Requires coordinator node	No	Yes
Point-to-point configuration	Simple	More involved
Standards-based networking	Yes	Yes
Standards-based applications	No	Yes
Underlying chipset	Freescale	Ember
Firmware available	802.15.4 (IEEE standard), DigiMesh (proprietary)	ZB(ZigBee 2007), ZNet 2.5 (obsolete)
Up-to-date and actively supported	Yes	Yes

1.5. Antennes



- wire : simple, radiations omnidirectionnelles ;
- chip : puce plate en céramique, petite, transportable (pas de risques de casser l'antenne), radiations cardioïdes (le signal est atténué dans certaines directions) ;
- U.FL : une antenne externe n'est pas toujours nécessaire;
- RPSMA : plus gros que le connecteur U.FL, permet de placer son antenne à l'extérieur d'un boîtier.

1.6. Communication avec l'ordinateur

Pour établir une communication avec l'ordinateur, il y a deux options : l'assemblage de différents éléments ou le XBee USB Explorer.

La communication en direct sans passer par une Arduino permet de configurer rapidement le XBee.

Les paramètres importants sont :

- PAN ID (Personal Area Network) : Identifiant du réseau personnel. Cet identifiant doit être le même pour les modules XBee qui doivent appartenir au même réseau.
- SH (Serial Number High) : Bits de poids fort (32 bits) du numéro de série du module XBee.
- SL (Serial Number Low) : Bits de poids faible (32 bits) du numéro de série du module XBee
- DH (Destination Address High) : Bits de poids fort du numéro de série du module XBee avec lequel vous désirez "converser". Mettre 0 pour répondre au coordinateur du réseau.
- DL (Destination Address Low) : Bits de poids faible du numéro de série du module XBee avec lequel vous désirez "converser". Mettre 0 pour répondre au coordinateur du réseau.

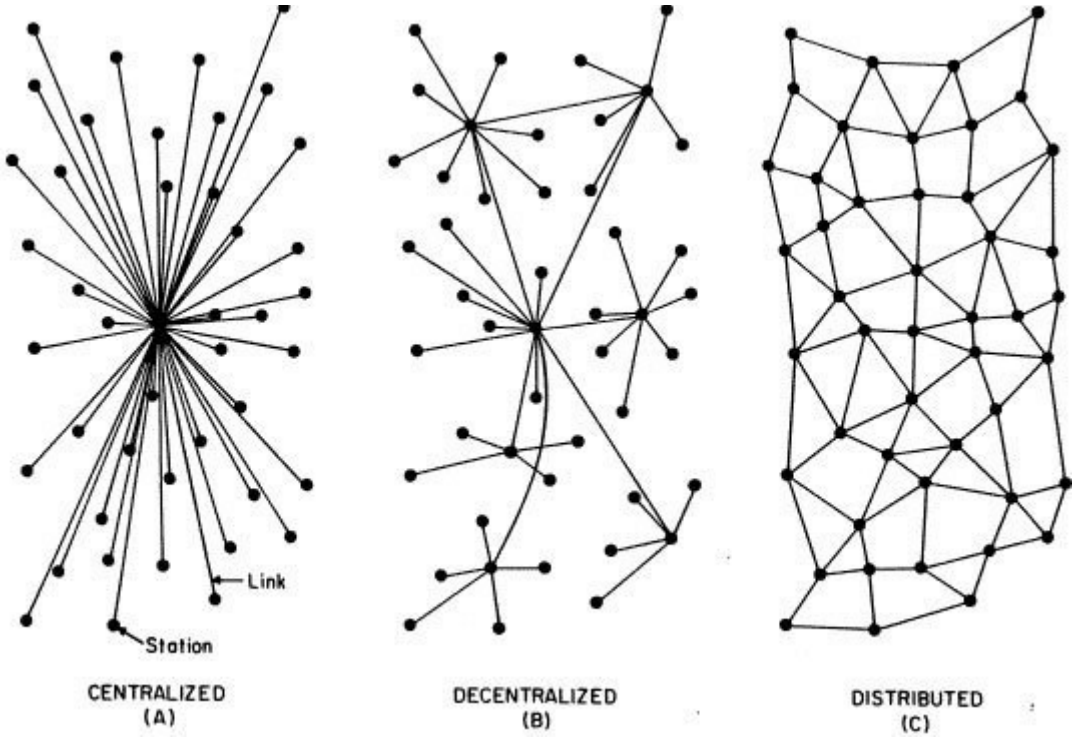
- BD (Baud Rate) : Vitesse de transmission en bit/s.
- RO (Packetisation Timeout) : Nombre de caractères tamponnés dans le XBee avant de lancer une transmission.

2. Notions de réseaux

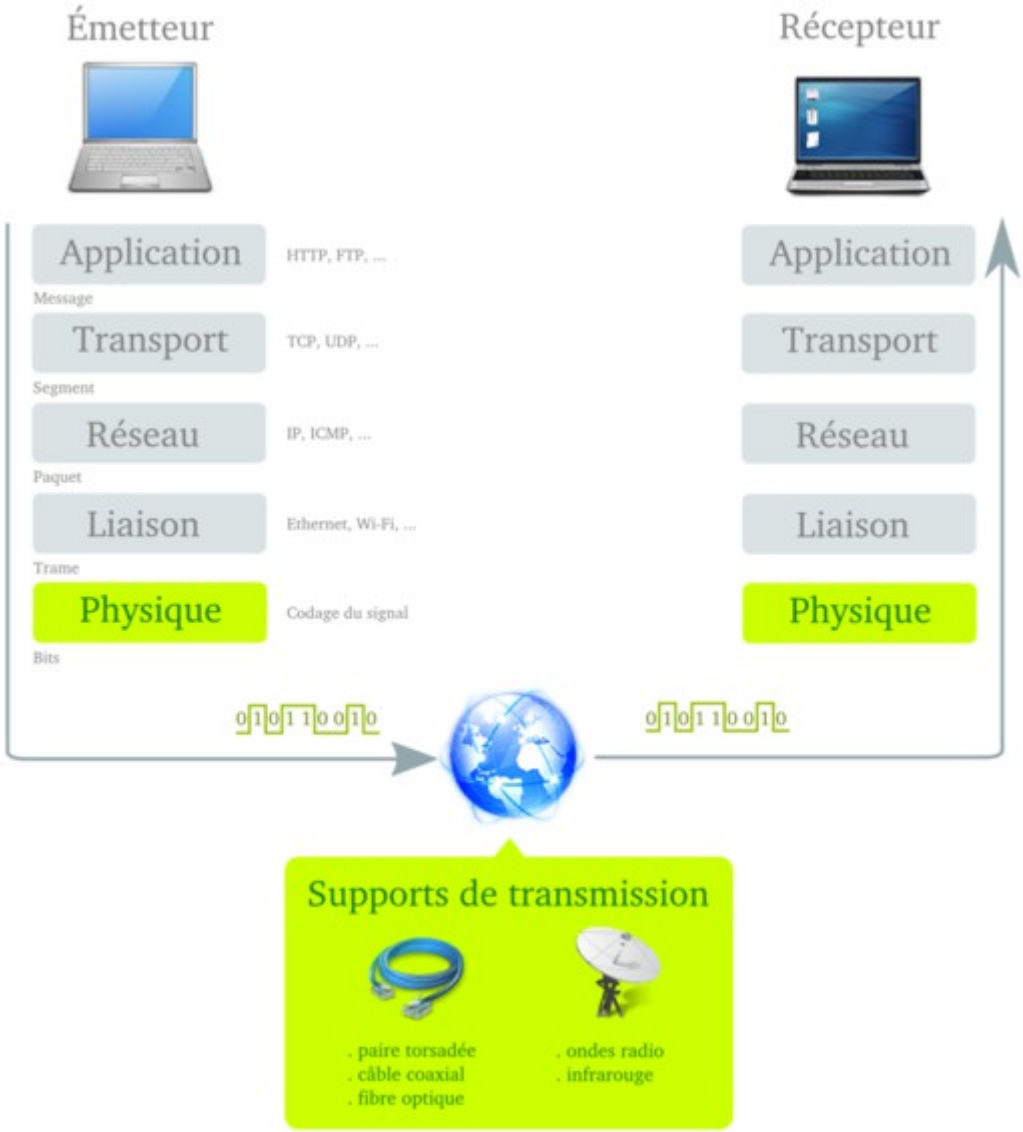
2.1. Protocoles de communication

Entre 1960 et 1964, l'idée de commutation de paquets est formulée conjointement par Leonard Kleinrock, Donald Davies et Paul Baran.

Paul Baran part du constat qu'un réseau centralisé est vulnérable. Si un nœud tombe en panne, l'information n'arrivera pas à son destinataire. C'est l'époque de la guerre froide, ces questions sont stratégiques. Une architecture distribuée avec des nœuds interconnectés parait la solution la plus flexible et la plus fiable.



Le message à envoyer est découpé en paquets (datagram). On leur ajoute généralement une étiquette composée de son adresse, celle du récepteur et son ordre dans le message original. Il est envoyé par des chemins divers évitant les congestions du réseau. Le destinataire remet ensuite le message dans l'ordre grâce aux étiquettes. On oppose cette technique à la commutation de circuits utilisée par le réseau téléphonique qui bloque un circuit pour une seule communication. Avec la commutation de paquets les ressources sont mutualisées, une ligne physique achemine différents bouts de messages provenant de différents expéditeurs.

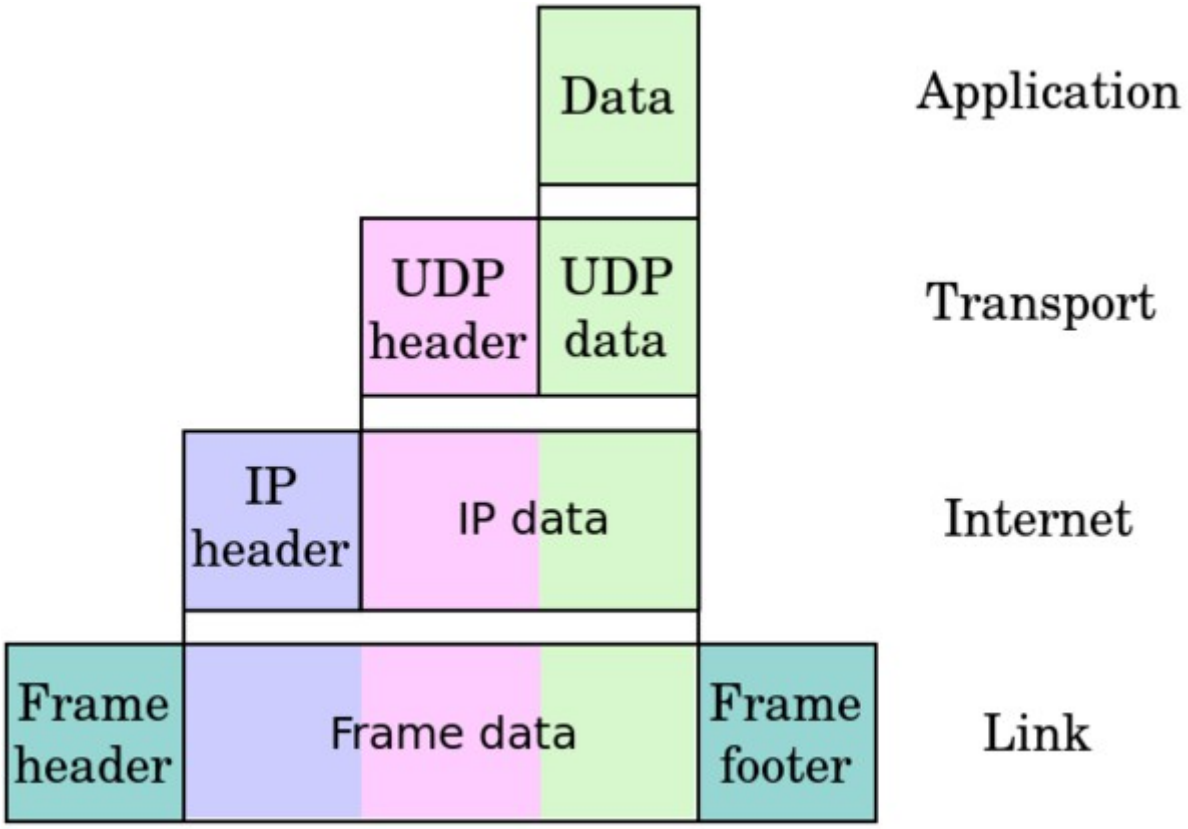


Le modèle Internet qui est une simplification du modèle OSI[1] donne une cohérence à la construction et l'envoie de messages. Il propose une architecture en couches, on parle aussi de pile (stack), définies et délimitées avec les notions de service, de protocole et d'interface. Chaque couche effectue des tâches spécifiques (services) et doit respecter certaines règles pour communiquer avec les couches inférieures et supérieures (protocole).

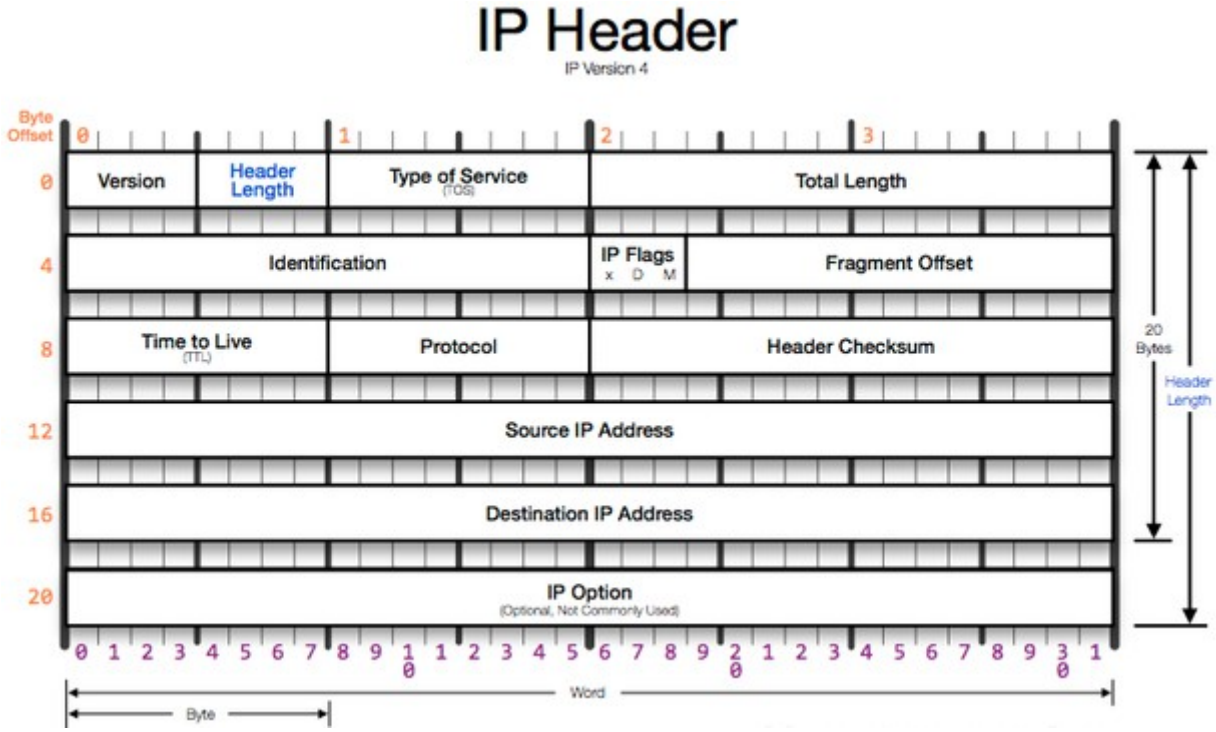
Le modèle OSI nous permet de ne plus confondre le Web et Internet : le Web, via le protocole HTTP[2], est seulement une des nombreuses applications fonctionnant sur Internet, qui lui est un ensemble de réseaux interconnectés.

Les protocoles peuvent être ouverts et devenir des standards. Ils sont alors décris dans des textes publics dont nous avons accès et qui sont approuvés par des organismes de normalisation nationaux, internationaux ou privés. L'intérêt est de poser un référentiel commun pour rendre le système ouvert, stable et modulaire.

1 Open Systems Interconnection
2 HyperText Transfer Protocol



Dans la construction d'un paquets, les données utiles, celles de l'utilisateur ou du processus sont encapsulées par des informations gérées par les protocoles inférieures, comme les en-têtes (header). Celles-ci répondent aux besoins de chaque protocole et identifie le paquet à travers le réseau.



Chaque en-tête a des champs spécifiques qui ont une taille précise mesurée en octets et qui sont

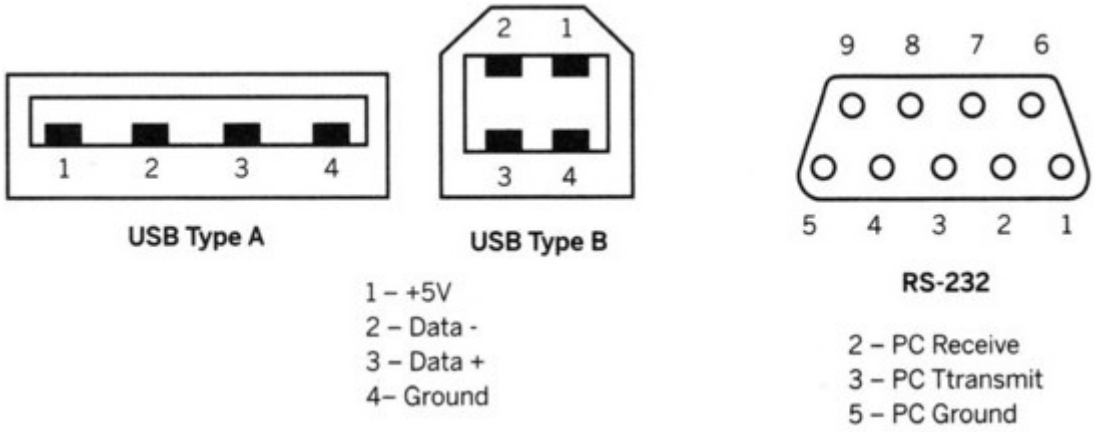
codés. Par exemple, dans l'en-tête IPv4[3], un champ codé sur 8 bits définit le protocole utilisé par la couche supérieure, la couche transport. S'il s'agit d'un paquet UDP[4], ce champ est 00010001, soit 17 en décimal ou 0x11 en hexadécimal. Une liste des nombres utilisés permet de déterminer le protocole. Ces champs peuvent être visualisés par des programmes qui analysent les paquets ou "sniffers)" comme Wireshark.

2.2. Communication série

Pour transmettre des données, il faut :

- coder les données (émetteur) pour qu'il y ait le moins de pertes possibles ;
- les acheminer via un support physique ;
- les décoder (récepteur) suivant les mêmes règles.

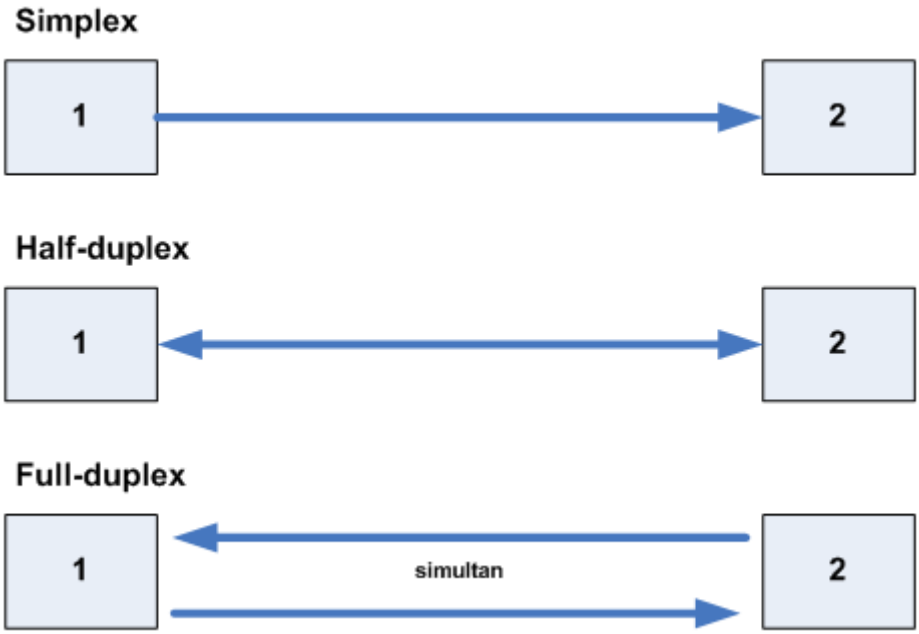
Au niveau physique, il s'agit surtout de l'envoi en série d'états électriques binaires (0 ou +5V par exemple). Le signal numérique est converti en signal analogique par des modems et transporté sur des supports filaires à base de cuivre ou de fibre optique, ou bien à travers le milieu aérien pour les transmissions non filaires. La transmission numérique des données est un ensemble de techniques fascinantes, qui consiste à trouver la meilleure solution pour transmettre les niveaux électriques représentant les bits.



2.2.1. Duplex / transceiver

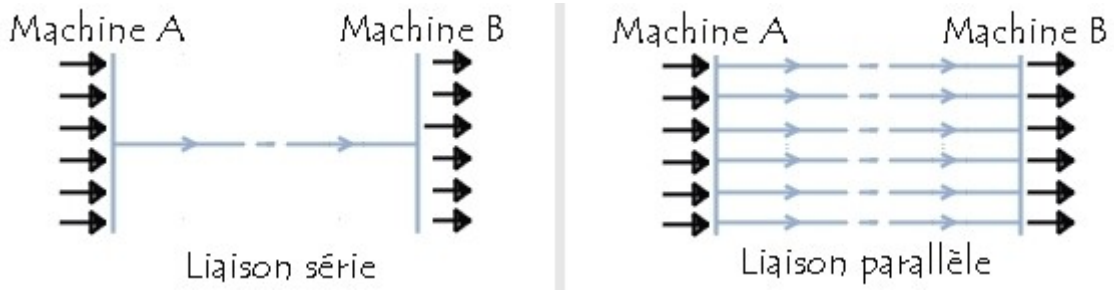
La communication avec le module XBee s'établit par une communication série asynchrone. Il suffit de quatre fils : deux pour l'alimentation (la masse et le +), un pour la réception et un pour l'émission. Le XBee permet de recevoir et d'émettre des données en même temps, on dit qu'il est full duplex, contrairement à la radio FM qui envoient les informations dans un seul sens (simplex) et au talkie-walkie qui ne permet pas à deux émetteurs de parler en même temps (half-simplex). On dit aussi que le XBee est un transceiver qui est la contraction de TRANSmitter (émetteur) et de reCEIVER (récepteur).

3 Internet Protocol version 4
4 User Datagram Protocol



2.2.2. Liaison série / parallèle

On distingue ensuite deux types de communications : série ou parallèle. La première nécessite moins de fils, toutes les données sont envoyées à la suite les unes des autres sur le même fil. La seconde est aujourd'hui moins utilisée du fait des perturbations dûes à la promiscuité des fils sur une nappe parallèle et aussi grâce à la rapidité de traitement des ordinateurs actuels.

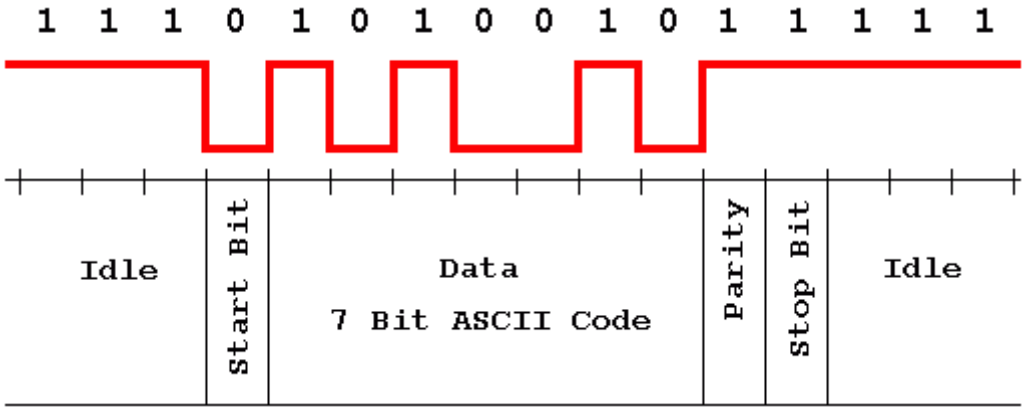


Dans une liaison en série, les données sont envoyées bit par bit sur la voie de transmission. Toutefois, étant donné que la plupart des processeurs traitent les informations de façon parallèle, il s'agit de transformer des données arrivant de façon parallèle en données en série au niveau de l'émetteur, et inversement au niveau du récepteur. Ces opérations sont réalisées grâce à un contrôleur de communication (la plupart du temps une puce UART[5]. Il fonctionne grâce à un registre à décalage. Le registre de décalage permet, grâce à une horloge, de décaler le registre (l'ensemble des données présentes en parallèle) d'une position à gauche, puis d'émettre le bit de poids fort (celui le plus à gauche) et ainsi de suite.

Toute l'astuce d'une liaison série asynchrone repose sur la forme des signaux envoyés qui permettent une synchronisation du récepteur sur chaque caractère reçu. Au repos (idle) la ligne de transmission est à l'état logique haut. La transmission débute par le passage à 0 de cette ligne pendant une période de l'horloge de transmission ce qui constitue le bit de start. Les bits du mot à transmettre sont ensuite envoyés derrière ce bit de start comme dans une transmission série

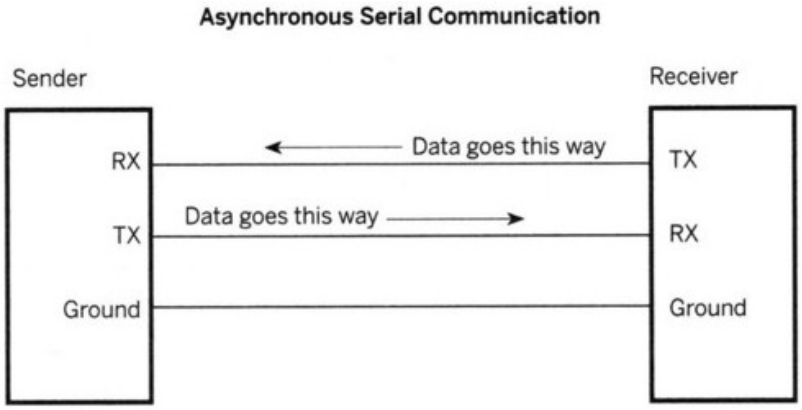
5 Universal Asynchronous Receiver Transmitter

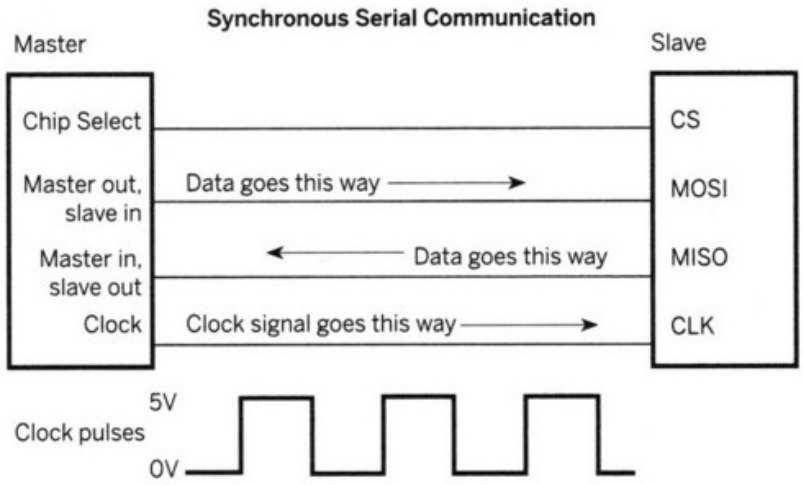
synchrone et, après le dernier bit utile, la ligne passe à nouveau à l'état haut pendant une ou deux périodes d'horloge pour constituer ce que l'on appelle le ou les bits de stop.



2.2.3. Synchrone / asynchrone

On distingue encore deux modes de communication qui spécifie le type de synchronisation entre l'émetteur et le récepteur : synchrone et asynchrone. Il faut en effet échantillonner le signal à la même cadence pour récupérer les bits dans l'ordre qui constitueront le message d'origine. Dans le mode asynchrone, il s'agit de se mettre d'accord à la connexion sur une vitesse de transmission, la synchronisation se faisant ensuite par les bits de start et de stop. Le mode synchrone dédie un fil pour la synchronisation, c'est le signal d'horloge (clock).

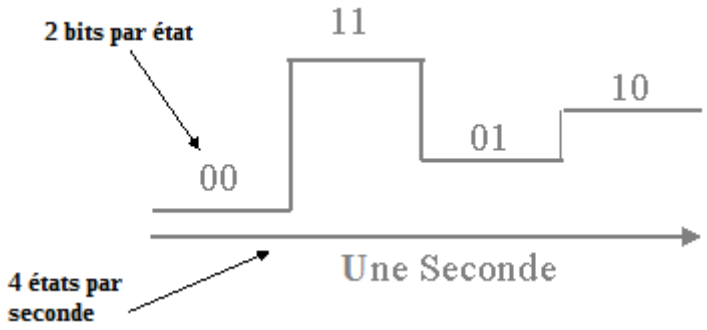




2.2.4. Baud / bits par seconde

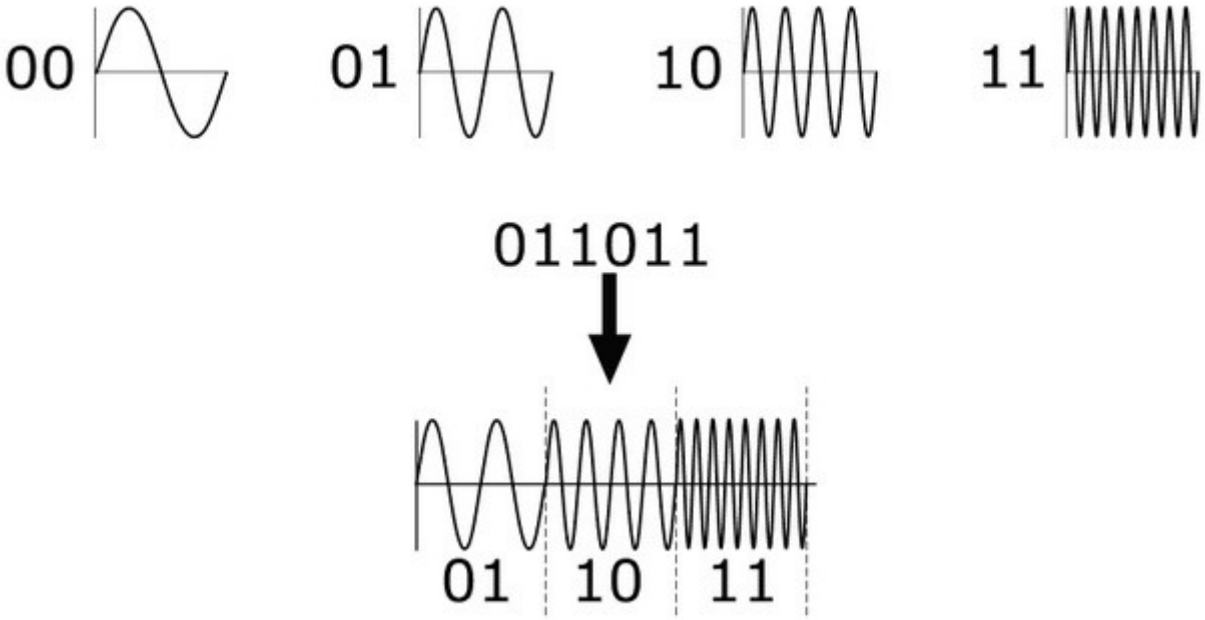
La vitesse de transmission est mesurée en bauds qu'il ne faut pas confondre avec le débit binaire qui mesure la quantité d'informations en bits par seconde bps. Le baud est le nombre de symboles transmis physiquement par seconde, alors que le bit désigne une unité d'information. Il est souvent possible de transmettre plusieurs bits par symbole. La mesure en bps de la vitesse de transmission est alors supérieure à la mesure en baud.

La formule est la suivante : bps = baud * nombre de bits par baud.



modulation d'amplitude

Dans l'exemple ci-dessus, le nombre de symboles possible à un instant donné est de quatre : 00, 01, 10, 11. Ces symboles peuvent correspondre à des états électriques différents comme -5V, -3V, +3V, +5V ou à des modulations d'amplitude, de fréquences ou de phases différentes. Le débit en bauds est de 4 symboles par seconde, et 2 bits sont transmis par symbole. Le débit binaire est donc le double du débit en bauds. 8 bps = 4 baud x 2 bits per baud.



modulation de fréquences

La valeur 9600 bps est souvent utilisée. Elle reflète la limite (ancienne) de certains modems de ne transmettre que 1200 baud et la technique communément utilisée de modulation. Cette modulation d'amplitude en quadrature permet de coder 8 bits par baud en associant une modulation d'amplitude à une modulation de phase. 9600bps = 1200 baud * 8 bits per baud. Pour Arduino, les débits

possibles sont : 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200 bps.

2.2.5. Norme RS 232

Cette norme est encore très souvent utilisée notamment au travers des puces FTDI[6] qui équipent les anciennes Arduinos et dans les cartes d'interface qui permettent de communiquer en série avec le module XBee. Les nouvelles cartes Arduino Uno utilisent un ATmega328 à la place.

Les signaux logiques aux niveaux TTL[7] ou CMOS[8] acceptent assez mal de voyager sur plus de quelques centimètres car leurs formes se dégradent alors à un point tel que leur exploitation devient impossible. Pour établir une liaison série sur une distance raisonnable, allant de quelques dizaines de centimètres à plusieurs centaines de mètres, diverses normes ont donc vu le jour.

Même si elle commence à être quelque peu attaquée par la norme RS 422 ; la norme RS 232 a encore un bel avenir devant elle vu sa présence quasi constante sur tous les équipements informatiques. Cette norme RS 232, appelée aussi CCITT[9] V24 ou encore V24 « tout court », est d'origine américaine et définit deux choses : les niveaux électriques des signaux utilisés pour la transmission mais aussi un certain nombre de lignes, autres que les lignes d'émission et de réception de données, ayant des fonctions de contrôle.

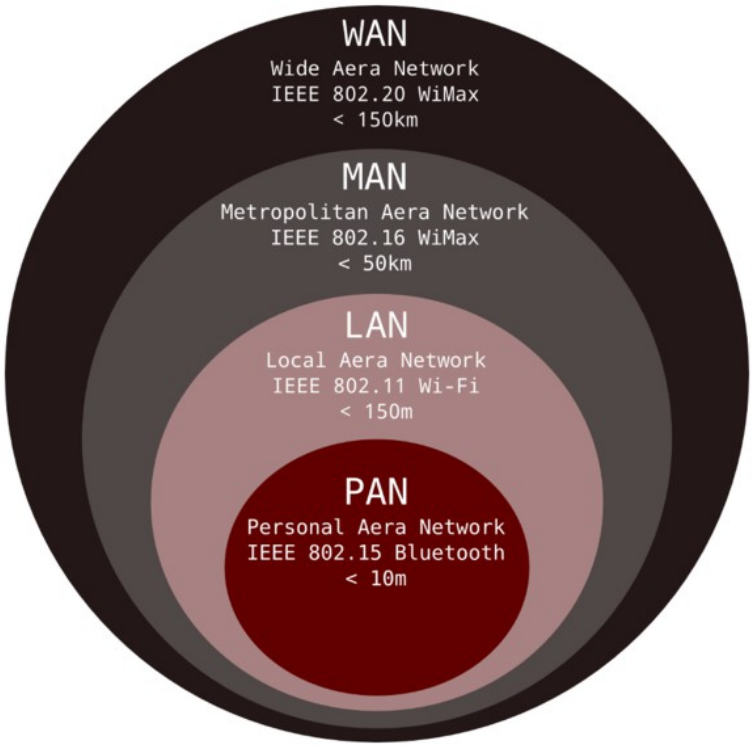
Au point de vue niveau, cette norme est très simple : tout signal de niveau compris entre +3 et +25 volts est considéré comme étant au niveau logique A alors que tout signal compris entre -3 et -25 volts est considéré comme étant au niveau logique B. A et B sont quelconques et peuvent être 0 ou 1 selon que l'on travaille en logique positive ou négative.

2.3. Réseaux sans fils

Le standard IEEE[10] 802.15.4 décrit les règles et fonctionnalités sur lequel se base le ZigBee mais pas uniquement. Il fait partie d'un ensemble plus vaste, dont la racine est le groupe de travail 802.15 et encore plus en amont le groupe 802 qui spécifie les standards des réseaux personnels sans fil (WPAN).

Quand on aborde les réseaux sans fil, on est confronté au paramètre de portée, c'est-à-dire jusqu'à quelle distance l'information peut-elle être transportée en bonne état. Des catégories de réseau ont ainsi été créées pour différencier les zones géographiques : PAN, LAN, MAN, WAN. Cela ne concerne pas uniquement les réseaux sans fils puisqu'un réseau Ethernet, très commun dans les entreprises ou les associations, peut être un PAN ou un LAN.

6 Future Technology Devices International
7 Transistor-Transistor logic
8 Complementary Metal Oxide Semiconductor
9 Comité consultatif international téléphonique et télégraphique
10 'Institute of Electrical and Electronics Engineers



Pour la transmission de données dans le milieu aérien, plusieurs options sont possibles. On utilise souvent des ondes électromagnétiques dans le domaine radio utilisant des fréquences porteuses réservées selon les pays.

FM 433 / 868 MHz
Toute la gamme
Radiometrix

AM 433 / 868 MHz
Modules
Telecontrolli

ZigBee®
Jennic
Firmtech

Bluetooth®
Free2move
Firmtech

Modules WLAN
Version boîtiers
ou modules OEM

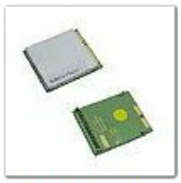
Modems Radio
Longue portée
868 MHz

Modules 2,4 Ghz
Transceivers
et Modems

FM 169 MHz
Toute la gamme
Radiometrix

Modules
GSM/GPRS
Modules Telit
Modules Teltonika

Modules divers
Circuits codeurs
et décodeurs



Transmission
Vidéo
Bande 2,4 GHz
Bande 5,8 GHz

Antennes
169 / 433 / 868 MHz
2,4 Ghz / GSM ...

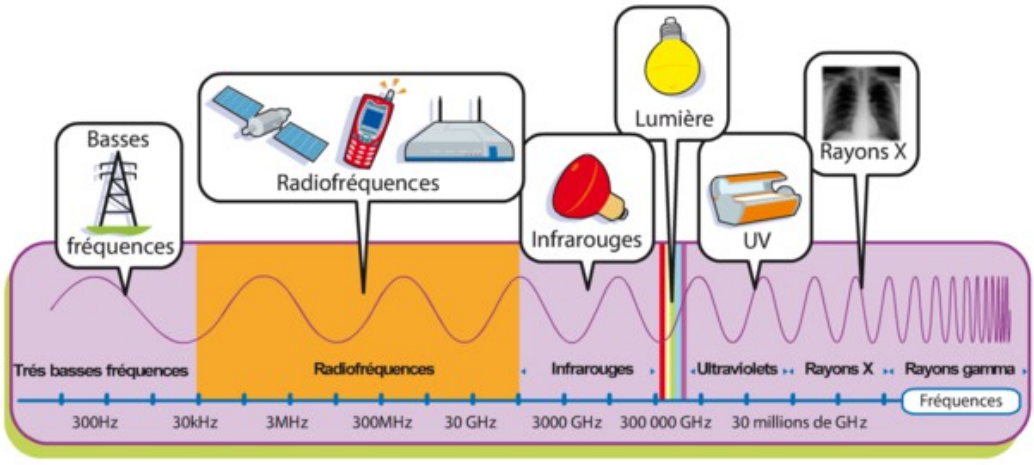
Hyperfréquence
Têtes HF
9,9 / 10,587 MHz

RFID
Lecteurs / Tags
Antennes

GPS
Récepteurs et
modules OEM

Téléecommandes
1 à 8 canaux
10 à 300 mètres

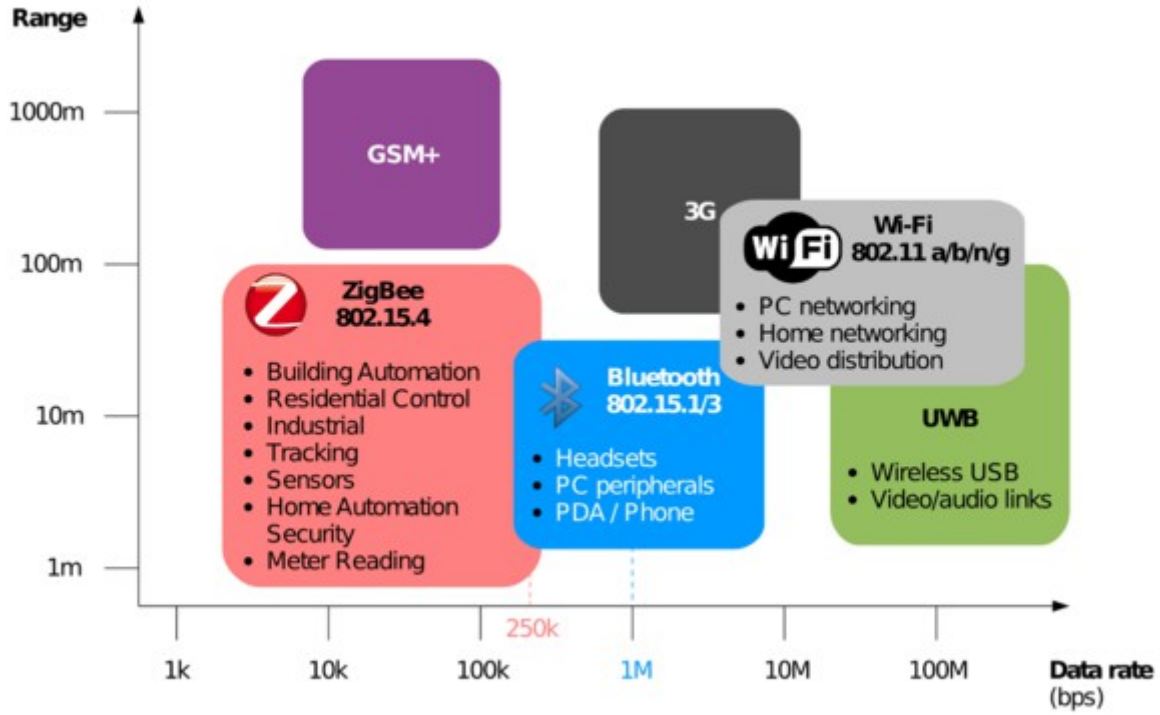
Ne sont répertoriés ici que les systèmes utilisant les radio-fréquences. Il existe aussi des transmissions par infra-rouges, qui sont directionnelles alors que les ondes radio sont omnidirectionnelles. Cela limite beaucoup leur utilisation. Les infra-rouges sont aussi des ondes électromagnétiques mais leur fréquence est beaucoup plus élevée. Elles n'utilisent donc pas d'antennes mais des émetteurs et récepteurs infra-rouges sous la forme de LEDs le plus souvent.



NB : les montages avec des dispositifs 433Mhz sont très économiques (~8€), et peuvent suffire dans certains cas. La différence de taille avec une transmission Bluetooth, Wi-Fi ou ZigBee c'est que les données sont envoyées sans contrôles, c'est-à-dire que l'on n'est pas sûr qu'elles soient arrivées

correctement du fait des interférences, des obstacles et de tout ce qui peut gêner les ondes. Si l'application n'est pas vitale, si quelques erreurs de temps en temps sont acceptables par le système alors cette solution peut être pertinente. En revanche, si les données sont critiques, reliées par exemple à un robot géant dans un lieu public ou à des appareils médicaux, cela n'est pas envisageable. On préférera dans ce cas des transmissions contrôlées et sécurisées par un protocole comme le Bluetooth, le Wi-Fi ou bien le ZigBee. Pour le 433Mhz, deux autres solutions sont tout de même possibles pour contourner le problème : vérifier les erreurs par programme ou le APC220 qui semble pouvoir fournir les vérifications demandées.

Le schéma ci-dessous permet de comparer le ZigBee, le Bluetooth et le Wi-Fi en termes de portée et de débits.

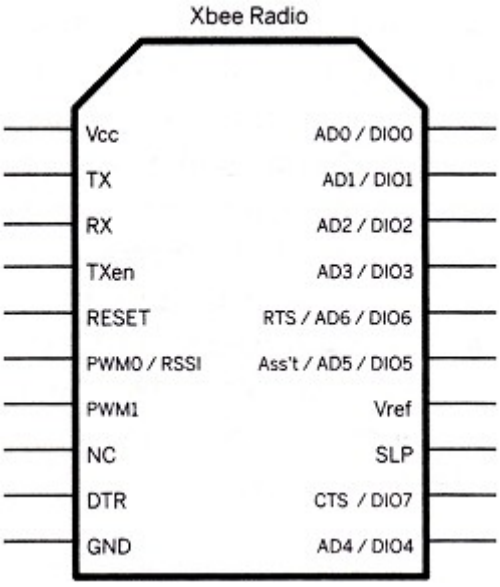


Cette vidéo présente les caractéristiques principales dans le choix des technologies sans fil.

3. Configuration

3.1. Brochage

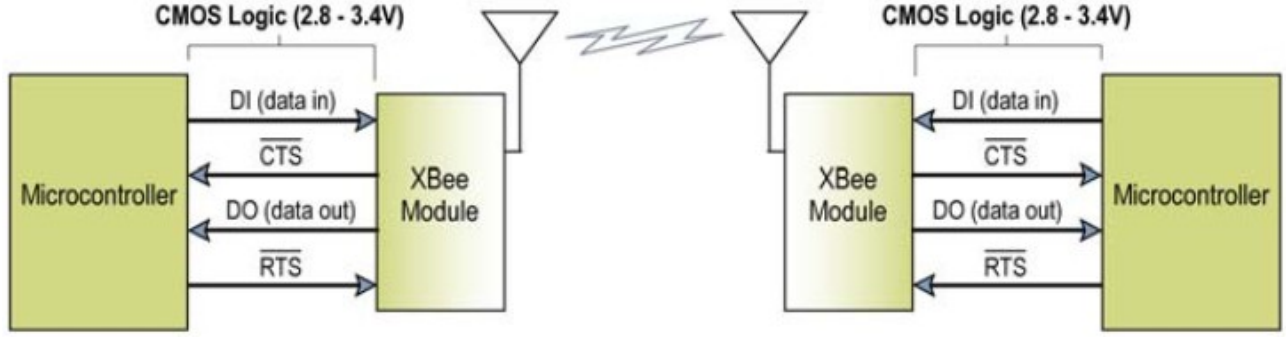
Le XBee series 1 possède un certain nombre d'entrées et sorties. Les sorties analogiques sont PWM0 et PWM1. Les entrées et sorties numériques sont DIO1, DIO2, DIO3, DIO4, DIO5, DIO6, DIO7 ("DIO" pour Digital Input Output). Les entrées analogiques sont : AD1, AD2, AD3, AD4, AD5 ("AD" pour Analog Digital, l'échantillonnage des tensions analogiques converties en numérique). Pour trouver la bonne commande AT associée à la configuration souhaitée, se référer aux pages du manuel (.pdf) ou à l'image ci-dessous.



Broche	Nom	Sens	Description
1	VCC		Tension d'alimentation
2	DOUT	Sortie	Sortie liaison série (Tx)
3	DIN	Entrée	Entrée liaison série (Rx)
4	DO8	Sortie	Sortie numérique
5	RESET	Entrée	Remise à 0 du module
6	PWM0	Sortie	Sortie PWM
	RSSI		Indicateur de puissance du signal reçu
7	PWM1	Sortie	Sortie PWM
8			Non connectée
9	DTR/	Entrée	Contrôle de Flux matériel liaison série
	SLEEP_RQ		Entrée de mise en veille
	DI8		Entrée numérique
10	GND		Masse
11	AD4	Entrée/Sortie	Entrée analogique
	DIO4		Entrée Sortie numérique
12	CTS/	Entrée/Sortie	contrôle de Flux matériel liaison série
	DIO7		Entrée Sortie numérique
13	ON_SLEEP/	Sortie	Indicateur de l'état du module
14	VREF	Entrée	Tension de référence du CAN
15	Associate	Entrée/Sortie	Indicateur d'association
	AD5		Entrée analogique
	DIO5		Entrée Sortie numérique
16	RTS	Entrée/Sortie	contrôle de Flux matériel liaison série
	AD6		Entrée analogique
	DIO6		Entrée Sortie numérique
17	AD3	Entrée/Sortie	Entrée analogique
	DIO3		Entrée Sortie numérique
18	AD2	Entrée/Sortie	Entrée analogique
	DIO2		Entrée Sortie numérique
19	AD1	Entrée/Sortie	Entrée analogique
	DIO1		Entrée Sortie numérique
20	AD0	Entrée/Sortie	Entrée analogique
	DIO0		Entrée Sortie numérique

3.2. Contrôle de flux

- Flux entrant sur Data IN par CTS : Quand le buffer émission est plein, le XBee le signale en mettant CTS à "1" pour que l'on stoppe l'envoi des données sur Data IN. Dés que le buffer est libre, CTS repasse à "0", et on peut renvoyer des données sur Data IN.
- Flux sortant sur Data OUT par RTS. Pour que le contrôle de flux par RTS soit actif il faut envoyer d'abord une commande AT pour l'autoriser : commande ATD6 suivie du paramètre "1". Quand la commande est active, si un "1" est appliquée à RTS, le XBee ne sort plus de données sur Data OUT. Quand on applique un "0" sur RTS, les données ressortent du XBee par Data OUT.

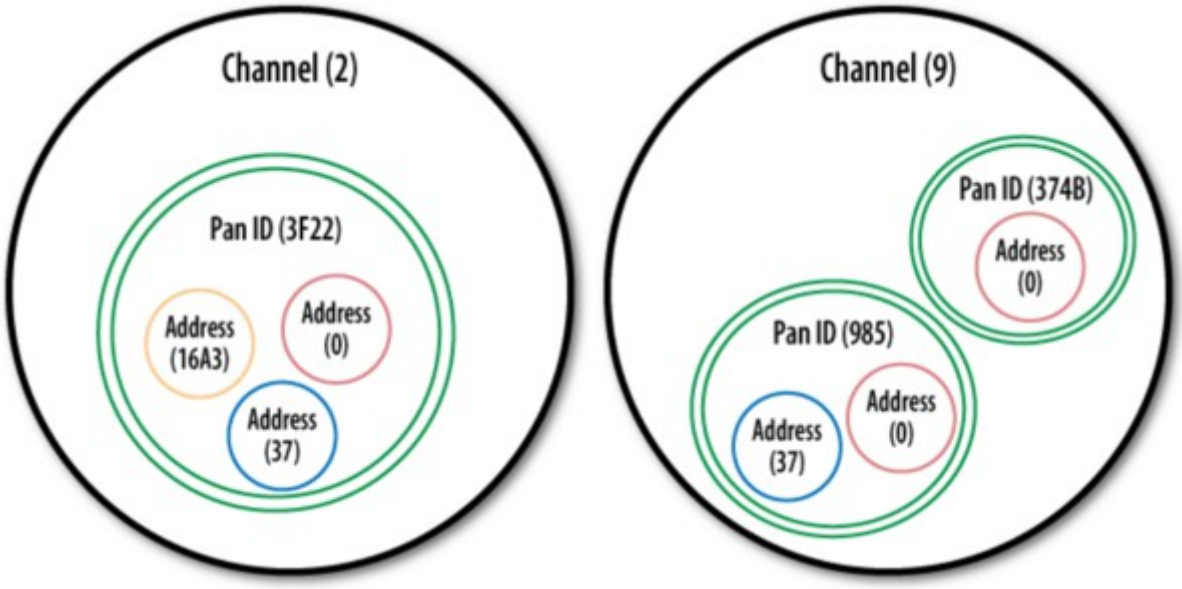


La configuration des paramètres de la liaison RS232 (Vitesse, parité, start et stop) se fait avec une commande AT.

Le protocole 802.15.4 utilisé par le module Xbee ajoute au paquet des data transmises, une adresse de la source et une adresse du destinataire.

3.3. Adressage

Pour tout XBee, il faut impérativement définir l'adresse du réseau ATID, son adresse personnelle ATMY et si besoin, l'adresse de destination des paquets ATDL.



Le module dispose des registres suivant :

- "MY" pour donner l'adresse source sur 16 bits.
- "SH" et "SL" pour donner respectivement les 32 bits MSB et 32 bits LSB de l'adresse source sur 64 bits. Cette adresse est un n° de série unique donné en usine par le constructeur et se trouve dans les registres "SH" et "SL".
- "DH" et "DL" qui donnent respectivement les 32 bits MSB et 32 bits LSB de l'adresse du destinataire.

Il y a 2 types d'adressage possible. Par adresse courte sur 16 bits et par adresse longue sur 64 bits.

ADRESSE COURTE :

Il faut mettre la valeur de l'adresse sur 16 bits, inférieure à 0xFFFE dans le registre "MY" et l'adresse sur 16 bits dans "DL" avec les 32 bits de "DH" à "0".

Par défaut les modules sont programmés avec MY=00, donc en adresse courte et DH=00 et DL=00.

Exemple avec 2 modules :

Un module sera à l'adresse courte : 0001 et l'autre aura l'adresse : 0002.

	MODULE 1	**MODULE 2**
MY (16 bits)	00 01	00 02
DH (32 bits)	00 00 00 00	00 00 00 00
DL (32 bits)	00 00 00 02	00 00 00 01

ADRESSE LONGUE :

il faut mettre 0xFFFF ou 0xFFFE dans MY pour désactiver l'adressage court. L'adresse longue utilisée est la valeur des 64 bits du n° de série usine contenus dans les registres SH et SL. L'adresse de destination est alors les 64 bits contenus dans DH et DL.

MODE UNICAST :

Dans ce mode de fonctionnement, le module récepteur, envoi un "ACK" à celui qui a émis le paquet de data. Si l'émetteur ne reçoit pas ce "ACK ", il renvoie jusqu'à 3 fois le paquet de data.

MODE BROADCAST :

Dans ce cas il n'y a pas de "ACK", envoyé par le récepteur, ni de répétition d'envoi par l'émetteur.

Tous les modules reçoivent et acceptent le paquet de data.

Pour envoyer des data sans tenir compte de l'adresse destinataire sur 16 ou 64 bits, il faut positionner l'adresse destinataire : DH =0x 00 00 00 00 et DL = 0x 00 00 FF FF.

Quand on programme le module, les paramètres sont entrés en hexadécimal. Les zéros non significatifs peuvent être omis.

3.4. Commandes de configuration

Pour modifier ou lire les paramètres du module, on va dialoguer par des commandes "AT" à 9600 Baud.

Il faut tout d'abord passer dans le mode "commande" en envoyant 3 fois le caractère "+" soit 0x2B en héxa en moins de 1 seconde. On doit respecter un temps de garde (de 1 seconde) avant et après l'envoi de ces 3 caractères. Le module répond par "OK" + "CR" (Carriage Return, soit 0x0D).

Le caractère "+" et le temps de garde sont modifiables par une commandes AT.

3.4.1. Commandes AT

Dans les télécommunications, l'ensemble de commandes Hayes est un langage de commandes spécifiques développé pour le modem Hayes en 1981. Les commandes sont une série de mots courts qui permettent de contrôler le modem avec un langage simple : composer un numéro de téléphone, connaître l'état de la ligne, régler le volume sonore, etc. Ce jeu de commandes s'est ensuite retrouvé dans tous les modems produits.

Elle est constituée des 2 caractères ASCII : "A" et " T" suivis de 2 caractères spécifiques à la commande, puis suit ou pas le caractère "Espace" et enfin suit un paramètre optionnel. On termine la commande par un "CR". Le module répond par "OK" suivi d'un "CR". Pour lire un paramètre, il suffira de laisser le champ paramètre en blanc. C'est le module qui renvoi alors la valeur de son paramètre.

"AT" + "ASCII commande" + "Espace" (option) + Paramètre (option) + "CR"

Si aucune commande AT n'est parvenue au module après son passage en mode commande pendant un temps de TIME OUT de 10 secondes (paramétrable par commande AT), le module retourne en mode IDLE.

Pour quitter le mode commande avant les 10 secondes du Time OUT, il faut envoyer la commande AT suivante : ATCN et le module répond alors par "OK"

Exemple :

ATDL 1F	Cette commande fixe la valeur du registre DL à 0x1F. Le module répond par "OK".
ATDL	Le module renvoi 0x1F valeur dans DL.

On peut envoyer plusieurs commandes à la suite.

Exemple :

ATDL 1F,WR,CN	Cette commande fixe la valeur du registre DL à 0x1F. puis sauve les paramètres dans la mémoire EE PROM et fait sortir le module du mode AT. Le module répond par "OK","OK","OK".

Remarques :

- A la mise sous tension du Xbee, il faut que RTS=1 , sinon il n'est pas disponible pendant environ 10 secondes.
- Pour flasher le module avec un nouveau Firmware, il faut que DTR = 0 ou bien le câbler sur la RS232, afin que le logiciel X-CTU de MaxStream le gère lui même pour le flash.

  La broche DTR peut rester en l'air dans les autres cas d'utilisation (terminal, commande AT).
- Attention de ne pas avoir d'autres modules Xbee sous tension pendant le Flash, car ils

risqueraient de répondre et de perturber la programmation du module.

3.4.2. Principales commandes AT

ATCN :	Pour quitter le mode commande.
ATCT + paramètre (0xFFFF):	Modifie ou lit le Time Out qui fait repasser le module en mode IDLE si aucune commande AT ne parvient. Le paramètre est le nbre de 100 ms. Par défaut il y a 0x64 soit 100ms x 100 = 10 sec.
ATGT + paramètre (0xFFFF):	Modifie ou lit le temps de garde. Le paramètre est le nbre de 1 ms. Par défaut il y a 0x3E8 soit 1ms x 1000 = 1 sec.
ATCC + paramètre (0xFF):	Modifie ou lit le caractère ASCII utilisé pour passer en mode commande. Par défaut on a 0x2B soit"+".
ATWR :	Sauve les paramètres dans la mémoire non volatile. Il faut impérativement attendre la réponse "OK" du module avant de lui envoyer une nouvelle commande.
ATCH + paramètre (0x0C à 0x17) :	Modifie ou lit le canal utilisé dans la bande 2,4 GHz. Par défaut il y a 0x0C.
ATDH + paramètre (0xFFFFFFFF) :	Modifie ou lit les 32 bits MSB de l'adressage destinataire. Par défaut il y a 0x00000000.
ATDL + paramètre (0xFFFFFFFF) :	Modifie ou lit les 32 bits LSB de l'adressage destinataire. Par défaut il y a 0x00000000.
ATMY + paramètre (0xFFFF) :	Modifie ou lit les 16 bits de l'adressage source. Par défaut il y a 0x0000.
ATSH :	Lit les 32 bits MSB du n° de série du module.
ATSL :	Lit les 32 bits LSB du n° de série du module.
ATNI + paramètre (20 octets ASCII) :	Sauve une chaîne de 20 caractères max pour l'identification du réseau : NI. Le caractère "espace" met fin à la commande.
ATND :	Cherche et donne les modules trouvés. Pour chacun o n obtient: MY + SH + SL + DB + NI. La commande se termine au bout de 2,5 seconde s et le module renvoie un "CR". On peut faire suivre la commande d'un paramètre constitué des 20 caractères du NI d'un module. Dans ce cas on obtient en répons e uniquement les paramètres de ce module.
ATPL + paramètre (0 à 4) :	Modifie ou lit la puissance de sortie du module. Par défaut il y a 4 soit la puissance max de 60 mW.

0	10 dBm soit 10 mW
1	12 dBm soit 16 mW
2	14 dBm soit 25 mW
3	16 dBm soit 40 mW
4	18 dBm soit 60 mW

ATBD + paramètre (0 à 7) : Modifie ou lit la vitesse en Baud de la liaison RS232. Par défaut on a 3 soit 9600 bauds.

0	1200 Bauds
1	2400 Bauds
2	4800 Bauds
3	9600 Bauds
4	19200 Bauds
5	38400 Bauds
6	57600 Bauds
7	115200 Bauds

ATID + paramètre (0xFFFF) : Modifie ou lit l'adresse du Pan ID. Il faut que cette valeur soit la même pour que les modules puissent communiquer entre eux.

ATRE : Restaure les paramètres par défaut du module. Cette commande ne réinitialise pas le champ ID

ATNT + paramètre (0xFC) : Défini ou lit le temps qu'acceptera le module pour découvrir d'autre nœud du réseau lorsque ND est activé.

Temps = paramètre x 100 ms

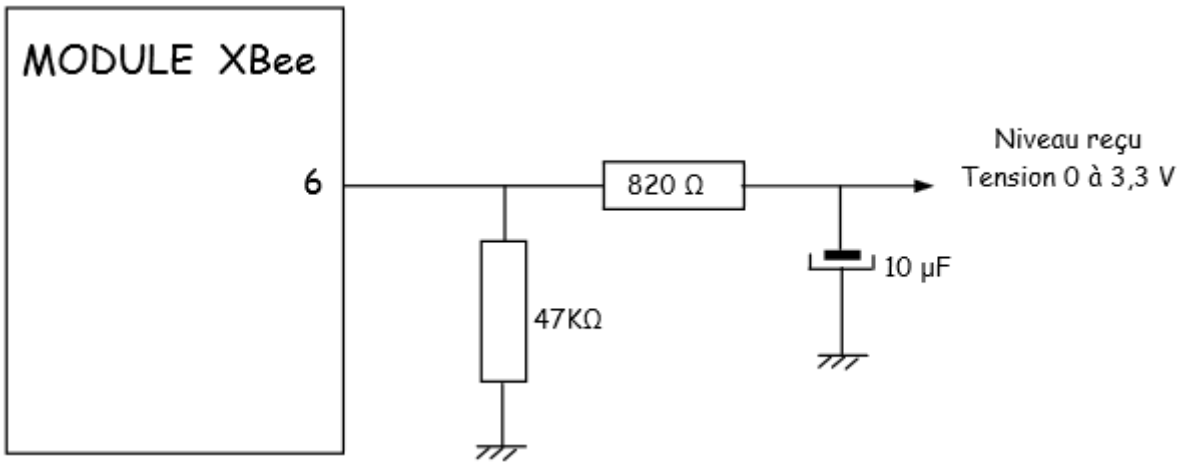
4. Mise en sommeil

Elle se fait par la pin 9 (DTR). Si on met un niveau "1" sur la pin 9, le module passe en mode "Sleep"et ne consomme plus que 10 µA sous 3 V. Pour le repasser en mode normal il faut mettre un "0" sur la pin 9. Le temps de réveil est d' environ 13 ms.

Il faudra auparavant paramétrer le module par la commande AT suivante : SM=1.

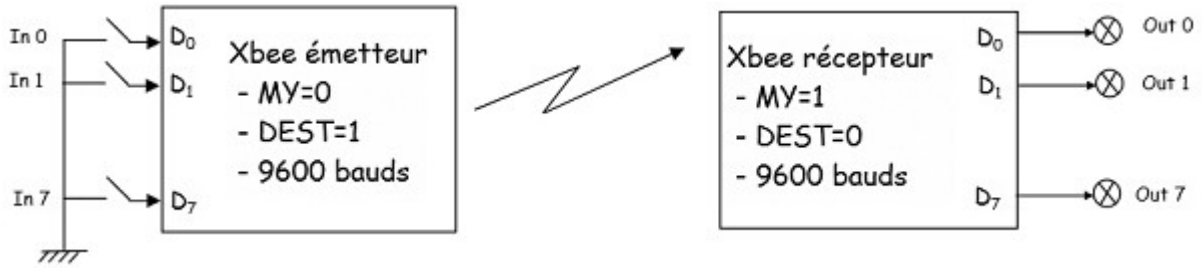
Indication du niveau reçu :

Sur la pin 6 on peut récupérer un signal PWM à 120 Hz dont le temps au travail est fonction du niveau reçu. Le montage suivant permet d'intégrer ce signal et d'obtenir une tension proportionnelle au signal reçu comprise entre 0 et 3,3V.



5. Fonctions spéciales I/O

Ces fonctions permettent de mettre soit à "1" soit à "0" directement une des 8 I/O du module via un autre module. Il faut impérativement que le firmware[11] soit en version 10A0 au minimum.



Il faut configurer la vitesse par ATBD et les adresses MY et DEST sur les 2 modules.

Les modules doivent être sur le même canal et avoir le même Pan ID.

Les I/O en entrée peuvent être tirées par des pull up sur le module émetteur en faisant ATPR = 0xFF. Pour les désactiver mettre le bit correspondant à "0".

CONFIGURATION EMETTEUR

- ATIU = 1 pour autoriser émission des I/O sans passer par l'UART
- ATD0 = 3 Lire le signal sur pin 20 : D0 (faire de même si on veut d'autres I/O parmi les 8)
- ATIR = h'14' vitesse d'échantillonnage des I/O = 100ms x 20 = 20 ms

CONFIGURATION RECEPTEUR

- ATIU = 1 pour autoriser émission des I/O sans passer par l'UART
- ATD0 = 5 Sortie numérique sur D0 avec repos = "1" et si ATD0 = 4 alors repos = "0".
- ATIA = 0 sorties modifiées par module d'adresse "0" (si ATIA=FFFF par tous les modules)

Éventuellement on peut configurer T0 pour que la sortie ne reprenne sa valeur de repos qu'après un certain temps quand l'émission aura cessé.

ATT0 = 3 Time out de la sortie D0 de 3 x 100 ms. Revient à sa valeur repos 300 ms après que l'émission ai cessé.

6. Association en réseau

Les modules Xbee peuvent fonctionner suivant 2 modes réseau:

- "PEER TO PEER" sans maître. Chaque module du réseau peut tenir le rôle de maître ou d'esclave. C'est le mode par défaut des Xbee. Chaque module est configuré comme un 'END DEVICE" en positionnant CE à "0" et en interdisant l'association par A1=1. Il faudra également mettre le même PANID (identification du réseau personnel) et le même canal RF. Le PANID est une valeur sur 2 octets comprise entre 0 et 0xFFFF.
- "Avec COORDINATEUR". Dans ce cas un module sera le coordinateur du réseau. Il faudra l'initialiser avec CE à "1". Les autres modules seront des "END DEVICE" configurés par CE à "0". On a ainsi constitué un PAN (Personnal Area Network).

  Chaque module du PAN aura un ID qui sera le même pour tout le PAN. Ce PANID devra

11 micrologiciel ou microcode

être unique pour éviter des communications entre les PAN.

Un "END DEVICE" pourra s'associer à un coordinateur dans un PAN, sans en connaître ni le PANID ni le canal RF.

La flexibilité de l'association sera configurée par la valeur du paramètre A1 pour le "END DEVICE" et par le paramètre A2 pour le "COORDINATOR".

6.1. Paramètre d'association d'un END DEVICE

Bit 0 : Scan de la bande RF

- b0=1 Recherche d'un canal et permet l'association sur n'importe quel canal.
- b0=0 Utilise uniquement le canal programmé dans l'EEPROM du module.

Bit 1 : Scan du PAN ID

- b1=1 Recherche un PANID et permet l'association avec n'importe lequel.
- b1=0 Utilise uniquement le PANID programmé dans l'EEPROM du module.

Bit 2 : Auto association.

- b2=1 Permet l'association avec un Coordinateur.
- b2=0 Pas d'association possible

6.2. Paramètre d'association d'un COORDINATEUR

Bit 0 : Scan du PAN ID

- b0=1 Le coordinateur fait un "active scan". Il choisit un canal et transmet un 'beacom request" en mode Broadcast. Il écoute ensuite le canal et note les éventuels Coordinateurs et leur PANID. Il explore ainsi tous les canaux et peut se choisir un PANID libre.
- b0=0 Le coordinateur garde son PANID et ne fait pas "d'active scan".

Bit 1 : Scan de la bande RF

- b1=1 Cherche un canal libre par "Energy Scan" et se l'attribut.
- b1=0 Garde le canal programmé dans l'EEPROM du module

Bit 2 : Auto association.

- b2=1 Permet aux END DEVICE de s'associer à ce module.
- b2=0 Interdit aux modules END DEVICE de s'associer à ce module.

Exemple :

En général on donnera un PANID et un canal RF au coordinateur.

Les "END DEVICE" seront configurés avec A1=07 ce qui leur imposera avant de s'associer à rechercher par SCAN de la bande 2,4 GHz un coordinateur et de choisir, s'il y en a plusieurs, celui dont la qualité de transmission est la meilleure. Il restera ensuite sur ce canal pour trafiquer et s'attribuer le PANID du coordinateur choisit.

Son adresse MY est alors changée en 0xFFFE ce qui signifie que pour l'adresser il faudra utiliser son adresse unique de série sur 64 bits (qui se trouve dans SH et SL).

Pour connaître les différentes adresses des modules associés, afin de pouvoir leur envoyer par la suite des données, le coordinateur devra faire un ATND (découverte des modules présents dans le réseau). Chaque module va ensuite répondre en donnant son MY (qui sera à 0xFFFE s'il est associé et à une autre valeur quelconque s'il ne l'est pas), SH et SL le n° de série particulier du module, suivi des caractères ASCII de son nom (que l'on aura initialisé précédemment dans ce module) ainsi que la puissance du signal reçu de ce module.

Le microcontrôleur qui gère le coordinateur devra se constituer un tableau pour garder ces informations en mémoire.

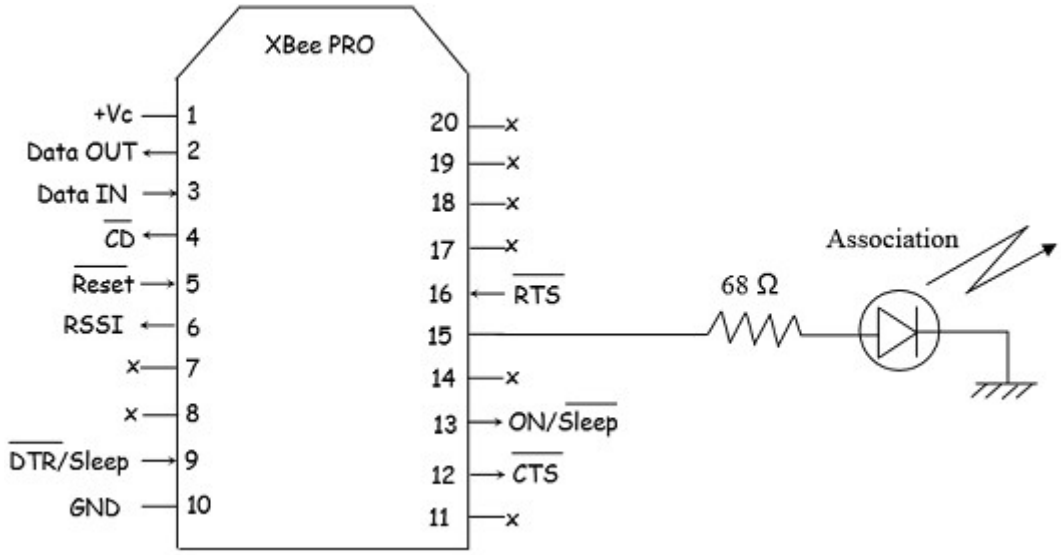
Quand il voudra envoyer une data à un module particulier, il devra configurer les valeurs DL et DH de son module XBee par une commande AT.

6.3. LED"ASSOCIATION"

Une LED est prévue sur la broche 15 du Xbee (D5). Un "END DEVICE" mettra cette LED dans l'état allumé fixe tant qu'il n'est pas associé à un coordinateur et ensuite elle clignotera dés que l'association sera faite.

Un coordinateur démarre dés qu'il a trouvé un canal et un PANID libre (si on lui a programmé un "active scan" et un "energy scan" par A2) et le signale par le clignotement de la LED. Cette Led était allumé fixe tant qu'il n'avait pas démarré.

On doit donc démarrer en premier le Coordinateur et attendre que cette Led clignote pour allumer les "End Device" qui vont chercher à s'associer.



7. Modes

Le XBee possède trois modes : TRANSPARENT, COMMAND et API.

- Le mode transparent est le mode par défaut à la mise en marche du module. Il consiste à permettre la communication entre deux systèmes numériques par le biais de la liaison série asynchrone RS232 (TX-RX) en mode full-duplex, comme si la liaison était filaire :
  Tout caractère émis depuis le SystèmeNumérique1 est reçu sur le SN2.
  Tout caractère émis depuis le SN2 est reçu sur le SN1.
- Le mode COMMAND permet de configurer le module, ses entrées, ses sorties, son adresse, l'adresse de destination de ses messages, etc... à l'aide des commandes AT.
- Le mode API permet de commander le module distant. Évidement, le mode API est incompatible avec le mode transparent. Les commandes AT existent encore, mais elles sont intégrées dans des commandes API.

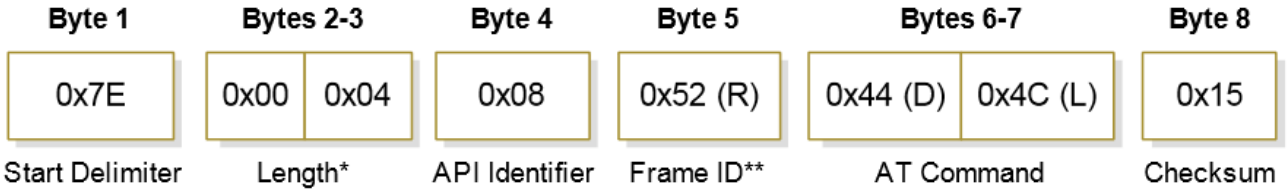
7.1. Les commandes API

Il est intéressant de supprimer l'arduino distant et de laisser le module XBEE distant gérer les Entrées/Sorties. Les commandes API permettent de commander le module distant. Il existe une vingtaine de commandes API. Deux d'entre elles sont particulièrement intéressantes :

- Envoi d'une commande AT au module local (0x08 – AT command)
- Envoi d'une commande AT du module distant (0x17 – Remote AT command).

MSB = Most Significant Byte, LSB = Least Significant Byte

Exemple de paquet API :



* : longueur (octet) = identifiant de la commande API + identifiant trame + structure de la commande API

** : valeur choisie arbitrairement

7.2. Calcul du checksum[12] d'un paquet API

La somme de contrôle, parfois appelée « empreinte », est un nombre qu'on ajoute à un message à transmettre pour permettre au récepteur de vérifier que le message reçu est bien celui qui a été envoyé. L'ajout d'une somme de contrôle à un message est une forme de contrôle par redondance. Cette redondance accompagne les données lors d'une transmission. Plus tard, il est possible de

12 Somme de contrôle

réaliser la même opération sur les données et de comparer le résultat à la somme de contrôle originale, et ainsi conclure sur la corruption potentielle du message. Si le nombre d'altérations durant la transmission est suffisamment petit, alors elles sont détectées. L'utilisation d'une unique somme de contrôle permet la détection mais non la correction des erreurs.

Soit le paquet de données suivant : 7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8

7E	Start Delimeter
00 0A	Length Bytes
01	API Identifier (commande API)
01	API Frame ID
50 01	Destination Address low
00	Option Byte
48 65 6C 6C 6F	Data Packet
B8	Checksum

Pour calculer la somme de contrôle, il faut ajouter tous les octets du paquet à l'exception du délimiteur de trame et de sa longueur (octets 2 et 3).

7E 00 0A 01 01 50 01 00 48 65 6C 6C 6F B8

Le total vaut : 01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F = 0x247

On garde les 8 derniers bits du résultat (0x47 dans l'exemple) que l'on soustrait à 0xFF, soit 0xFF - 0x47 = 0xB8.

Remarque : si un paquet API possède une somme de contrôle incorrecte, le destinataire va tout simplement ignorer le paquet.

Pour vérifier une somme de contrôle, ajouter tous les octets du paquet à l'exception du délimiteur de trame et de sa longueur (octets 2 et 3). Les 8 derniers bits doivent être égales à 0xFF.

Ex : 01 + 01 + 50 + 01 + 00 + 48 + 65 + 6C + 6C + 6F + B8 = 0x2FF

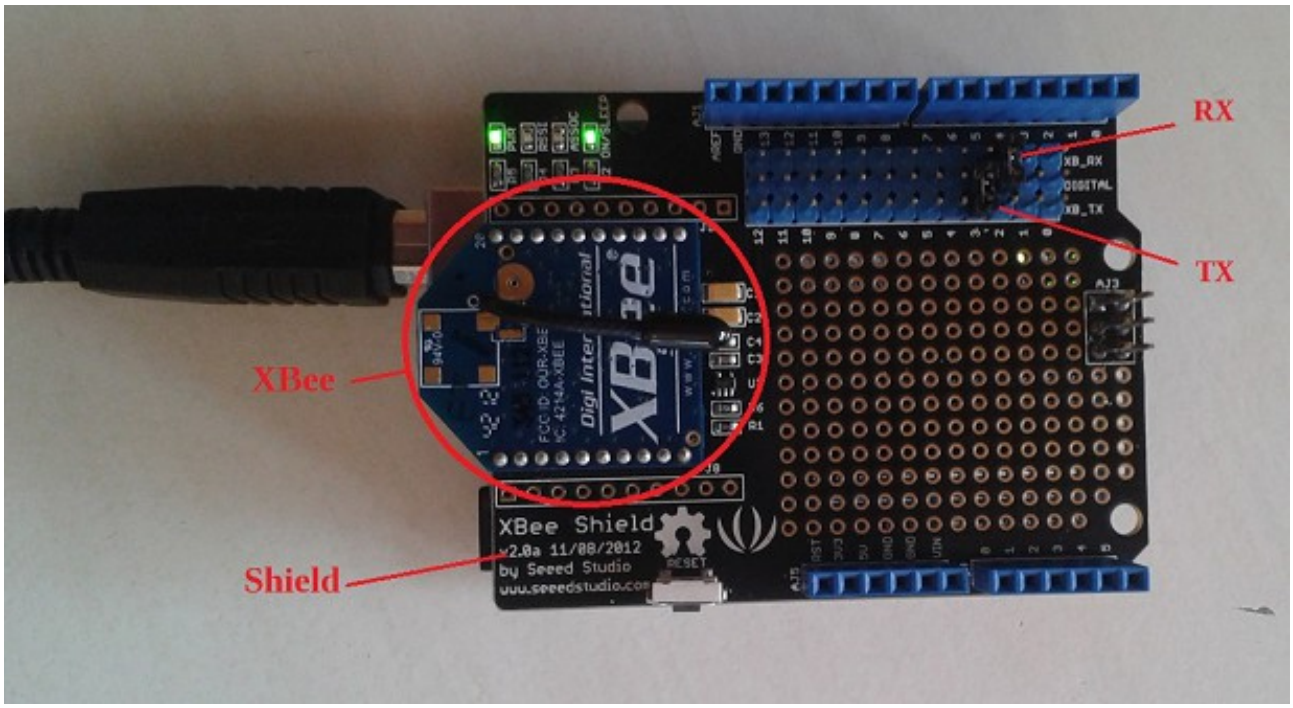
Pour vous aider, ce composeur de trame API est utilisable en ligne.

8. Exemples de programmation

8.1. Mode transparent

Cet exemple illustre un simple chat en mode simplex. On utilise 2 Xbee reliés à un PC à travers un shield pour les configurer.

Attention à positionner les cavaliers du shield pour la réception et l'émission des données.



Un Xbee émetteur envoie des caractères reçus au travers du terminal Arduino côté PC vers un Xbee récepteur. Les caractères reçus par le récepteur sont affichés sur un deuxième terminal Arduino.

Pour voir la vidéo, cliquer sur l'image ci-contre : 

code de l'émetteur :

```
/*
Configuration d'un Xbee en mode transparent
enoie de caractères
*/

const int RX = 2;    // pin Shield Arduino pour reception
const int TX = 3;    // pin Shield Arduino pour transmission

const char *PANID = "1111";      // adresse reseau
const char *MYID  = "1001";      // adresse du XBee
const char *DLID  = "1002";      // adresse de destination

#include <SoftwareSerial.h>          // Change pin for xbeeSerial

SoftwareSerial xbeeSerial(TX, RX) ;     // pin 2 = RX Arduino, pin3 = TX Arduino

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);            // liaison série de la console (débugage)
  xbeeSerial.begin(9600);        // Liaison série du module XBEE

  Serial.println("init");
  setXBEE("+++");

  // attente réponse OK de la commande AT
  char thisByte = 0;
  while ( thisByte != '\r' )
    if ( xbeeSerial.available() > 0 ) {
```

```
      thisByte = xbeeSerial.read();
      Serial.print(thisByte);
      }
  Serial.println("");

  char  buffer[11];
  setXBEE("ATRE\r");        // configuration usine par defaut
  setXBEE(strcat(strcat(strcpy(buffer,"ATID"), PANID), "\r"));    // adresse du reseau PAN
  setXBEE(strcat(strcat(strcpy(buffer,"ATMY"), MYID), "\r"));    // set my address using 16-bit
addressing
  setXBEE(strcat(strcat(strcpy(buffer,"ATDL"), DLID), "\r"));    // adresse de destination sur 16
bits
  setXBEE("ATDH0\r");       // pas de partie haute, adresse 16 bits
  setXBEE("ATWR\r");        // enregistrement
  setXBEE("ATCN\r");        // put the radio in data mode
}

void loop()
{
  if ( Serial.available() > 0 ) {
    char  outputByte;

    // saisie via terminal
    do {
      outputByte = Serial.read();
      if ( outputByte != '\n' ) {
        xbeeSerial.print(outputByte);
        Serial.print(outputByte);
        }
      } while ( outputByte != '\n' );

    xbeeSerial.print("~");  // caractère de fin
    Serial.println("");
    }
}

void setXBEE(char *buffer)
{
  Serial.println(buffer);
  Serial.print("sent: "); Serial.println(xbeeSerial.print(buffer));
  delay(100);
}
```

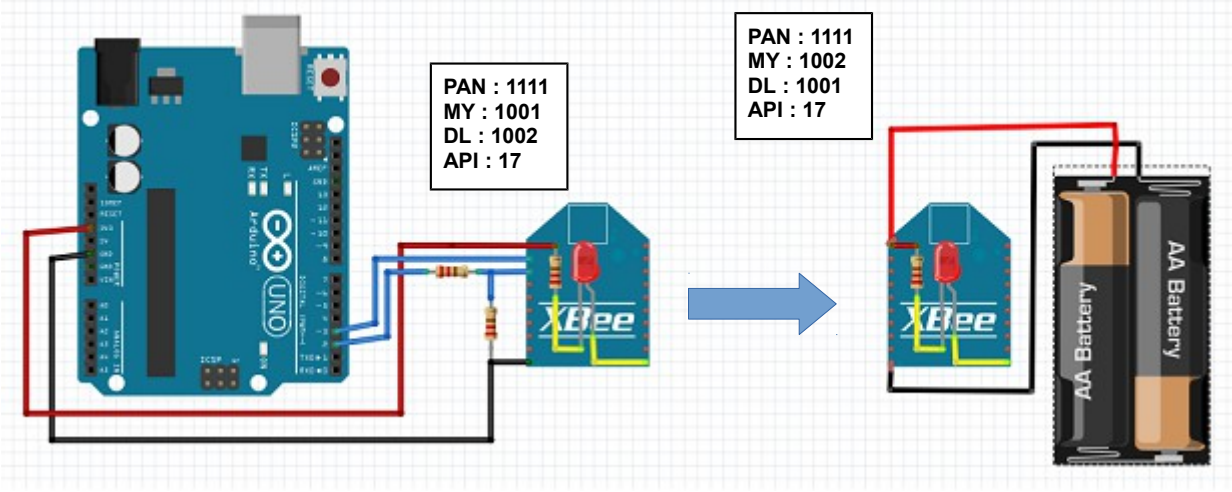
code du récepteur :

```
/*
Configuration d'un Xbee en mode transparent
lecture de caractères reçus
*/

const int RX = 2;    // pin Shield Arduino pour reception
const int TX = 3;    // pin Shield Arduino pour transmission

const char *PANID = "1111";      // adresse reseau
const char *MYID  = "1002";      // adresse du XBee
const char *DLID  = "1001";      // adresse de destination

#include <SoftwareSerial.h>          // Change pin for xbeeSerial

SoftwareSerial xbeeSerial(TX, RX) ;     // pin 2 = RX Arduino, pin3 = TX Arduino

void setup()
{
```

```
  Serial.begin(9600);            // liaison série de la console (débugage)
  xbeeSerial.begin(9600);        // Liaison série du module XBEE

  Serial.println("init");
  setXBEE("+++");

  // attente réponse OK de la commande AT
  char thisByte = 0;
  while ( thisByte != '\r' )
    if ( xbeeSerial.available() > 0 ) {
      thisByte = xbeeSerial.read();
      Serial.print(thisByte);
      }
  Serial.println("");

  char  buffer[11];
  setXBEE("ATRE\r");        // configuration usine par defaut
  setXBEE(strcat(strcpy(buffer,"ATID"), PANID), "\r"));    // adresse du reseau PAN
  setXBEE(strcat(strcat(strcpy(buffer,"ATMY"), MYID), "\r"));    // set my address using 16-bit
addressing
  setXBEE(strcat(strcat(strcpy(buffer,"ATDL"), DLID), "\r"));    // adresse de destination sur 16
bits
  setXBEE("ATDH0\r");       // pas de partie haute, adresse 16 bits
  setXBEE("ATWR\r");        // enregistrement
  setXBEE("ATCN\r");        // put the radio in data mode
}

void loop()
{
  char  inputByte = xbeeSerial.read();

  if ( inputByte > 0 )
    if ( inputByte == '~' )  // caractère de fin
      Serial.println("");
    else
      Serial.print(inputByte);
}

void setXBEE(char *buffer)
{
  Serial.println(buffer);
  Serial.print("sent: "); Serial.println(xbeeSerial.print(buffer));
  delay(100);
}
```

8.2. Mode API

Dans cet exemple, un Xbee émetteur donne l'ordre à un Xbee récepteur d'allumer une LED connectée sur sa broche D4 par une commande AT distante (commande API 0x17).



code de l'émetteur :

```
/*
Commande le clignotement d'une LED d'un XBee distant en mode API
*/

#include <SoftwareSerial.h>          // Change pin for xbeeSerial

const int RX = 2;    // pin Shield Arduino pour reception
const int TX = 3;    // pin Shield Arduino pour transmission

const char *PANID = "1111";      // adresse reseau
const char *MYID  = "1001";      // adresse du XBee
const char *DLID  = "1002";      // adresse de destination

// trame API : mise à 0 de D4 du module distant d'adresse 0x1002
char D4LOW[20]={
  0x7E,          // start delimiter
  0x00, 0x10,    // length (16 bytes)
  0x17,          // API ID (Allows for module parameter registers on a remote device to be queried
or set)
  0x01,          // UART data frame for the host to correlate with a subsequent ACK. If set to
‘0’, no AT Command Response will be given.
  0, 0, 0, 0, 0, 0, 0, 0,  // 64-bit address of the destination, MSB first, LSB last. Broadcast =
0x000000000000FFFF. This field is ignored if the 16-bit network address field equals anything
other than 0xFFFE.
  0x10, 0x02,    // 16-bit network address of the destination, MSB first, LSB last.
  0x02,          // Apply changes on remote
  0x44, 0x34,    // D4
  0x04,          // 4 (OFF)
  0x00};         // checksum
// trame API : mise à 1 de D4 du module distant d'adresse 0x1002
char D4HIGH[20]={
  0x7E,          // start delimiter
  0x00, 0x10,    // length (16 bytes)
  0x17,          // API ID
  0x01,          // frame ID
  0, 0, 0, 0, 0, 0, 0, 0,  // adresse sur 64 bits
  0x10, 0x02,  // adresse destinataire sur 16 bits
  0x02,
  0x44, 0x34,    // D4
  0x05,          // 5 (ON)
  0x00};         // checksum

SoftwareSerial xbeeSerial(TX, RX) ;      // pin 2 = RX Arduino, pin3 = TX Arduino
```

```
boolean  state = false;

void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);            // liaison série de la console (débugage)
  xbeeSerial.begin(9600);        // Liaison série du module XBEE

  // calcul le checksum des trames
  D4LOW[19]  = setCheckSum(D4LOW);
  D4HIGH[19] = setCheckSum(D4HIGH);

  Serial.println("init");
  setXBEE("+++");

  // attente réponse OK de la commande AT
  char thisByte = 0;
  while ( thisByte != '\r' )
    if ( xbeeSerial.available() > 0 ) {
      thisByte = xbeeSerial.read();
      Serial.print(thisByte);
      }
  Serial.println("");

  char  buffer[11];
  setXBEE("ATRE\r");        // configuration usine par defaut
  setXBEE(strcat(strcat(strcpy(buffer,"ATID"), PANID), "\r"));    // adresse du reseau PAN
  setXBEE(strcat(strcat(strcpy(buffer,"ATMY"), MYID), "\r"));    // set my address using 16-bit
addressing
//  setXBEE(strcat(strcat(strcpy(buffer,"ATDL"), DLID), "\r"));    // adresse de destination sur
16 bits
  setXBEE("ATDH0\r");       // pas de partie haute, adresse 16 bits
  setXBEE("ATAP1\r");       // Passage en mode API
  setXBEE("ATCN\r");        // put the radio in data mode
  setXBEE("ATWR\r");        // enregistrement

  // attente de la réponse de 6 "OK\r" suite aux commandes AT
  for (int i(0); i < 6;
    i += (xbeeSerial.read() == '\r') ? 1 : 0 );
}

void loop()
{
  state = !state;

  for (int i(0); i < 20; i++)
    xbeeSerial.print(state
      ? D4HIGH[i]  // envoi d'une trame API qui allume la DEL 4 du module XBEE distant
      : D4LOW[i]    // envoi d'une trame API qui eteind la DEL 4 du module XBEE distant
      );

  Serial.println(state ? "high" : "low");

  // affiche la reponse de la commande AT distante (API 0x97)
  // Frame ID of the Remote Command
  // 64 bit address of the remote module
  // 16- bit network address of the remote
  // Two ASCII characters that identify the AT command
  // Status ( byte 18) : 0 = OK, 1 = Error, 2 = Invalid Command, 3 = Invalid Parameter
  if ( xbeeSerial.available() > 0 ) {
    int c = xbeeSerial.read();
    if ( c == 0x7E )  // start delimiter
      while ( c != -1 ) {
```

```
        Serial.print(c, HEX);
        Serial.print(" ");
        c = xbeeSerial.read();
        }
    Serial.println("");
    }

  delay (3000);
}

void setXBEE(char *buffer)
{
  Serial.println(buffer);
  Serial.print("sent: "); Serial.println(xbeeSerial.print(buffer));
  delay(100);
}

int setCheckSum(char *frame)
{
  char *ptr = frame;
  int  i = 3, sum = 0;

  ptr += 3;  // skip start delimiter + length
  while ( i++ < 20 )
    sum += *ptr++;

  Serial.print("checksum: 0x"); Serial.println(0xFF - (sum & 0xFF), HEX);

  return 0xFF - (sum & 0xFF);
}
```

code du récepteur :

```
/*
Recoit la commande AT distante d'un XBee en mode API pour faire clignoter une LED
*/

const int RX = 2;    // pin Shield Arduino pour reception
const int TX = 3;    // pin Shield Arduino pour transmission

const char *PANID = "1111";      // adresse reseau
const char *MYID  = "1002";      // adresse du XBee
const char *DLID  = "0";      // adresse de destination (pas de destinataire, XBee juste
receveur)

#include <SoftwareSerial.h>          // Change pin for xbeeSerial

SoftwareSerial xbeeSerial(TX, RX) ;     // pin 2 = RX Arduino, pin3 = TX Arduino
char buffer[20] = "";          // buffer de reception
int  nb_byte    = 0;          // octets recus

void setup()
{
  Serial.begin(9600);            // liaison série de la console (débugage)
  xbeeSerial.begin(9600);        // Liaison série du module XBEE

  Serial.println("init");
  setXBEE("+++");

  // attente réponse OK de la commande AT
  char thisByte = 0;
  while ( thisByte != '\r' )
    if ( xbeeSerial.available() > 0 ) {
      thisByte = xbeeSerial.read();
```

```
      Serial.print(thisByte);
      }
  Serial.println("");

  char  buffer[11];
  setXBEE("ATRE\r");        // configuration usine par defaut
  setXBEE(strcat(strcat(strcpy(buffer,"ATID"), PANID), "\r"));    // adresse du reseau PAN
  setXBEE(strcat(strcat(strcpy(buffer,"ATMY"), MYID), "\r"));    // set my address using 16-bit
addressing
  setXBEE(strcat(strcat(strcpy(buffer,"ATDL"), DLID), "\r"));    // adresse de destination sur 16
bits
  setXBEE("ATDH0\r");       // pas de partie haute, adresse 16 bits
  setXBEE("ATAP1\r");       // Passage en mode API
  setXBEE("ATWR\r");        // enregistrement
  setXBEE("ATCN\r");        // put the radio in data mode

  // attente de la réponse de 8 "OK\r" suite aux commandes AT
  for (int i(0); i < 8;
    i += (xbeeSerial.read() == '\r') ? 1 : 0 );
}

void loop()
{
  // affiche la statut de la transmission (API 0x89)
  // Frame ID being reporte
  // Status (Byte 6) : 0 = Success, 1 = No ACK (Acknowledgement) received, 2 = CCA failure, 3 =
Purged
  if ( xbeeSerial.available() > 0 ) {
    int c = xbeeSerial.read();
    if ( c == 0x7E )  // start delimiter
      while ( c != -1 ) {
        Serial.print(c, HEX);
        Serial.print(" ");
        c = xbeeSerial.read();
        }
    Serial.println("");
    }
}

void setXBEE(char *buffer)
{
  Serial.println(buffer);
  Serial.print("sent: "); Serial.println(xbeeSerial.print(buffer));
  delay(100);
}

int getCheckSum(char *frame, const int frame_size)
{
  char *ptr = frame;
  int  sum = 0;

  ptr += 3;  // skip start delimiter + length
  for (int i(0); i <= frame_size; i++)
    sum += *ptr++;

  return 0xFF & sum;
}
```

Télécharger le code source :