Pourtant, il reste de nombreux circuits qui ne peuvent pas se permettre la complexité d'un port USB. Par exemple, de nombreux montages DIY sont aujourd'hui conçus autour d'un microcontrôleur, de la famille PIC, AVR ou autres. La simplicité du port série réduit le prix et le temps de développement : pas de pilote complexe à mettre au point ou à configurer. Vous aurez probablement besoin de cette liaison pour lire ou envoyer des informations, charger un nouveau programme ou le déverminer.

De plus, les ports série restent associés à des rôles traditionnels comme la communication avec un modem (que ce soit avec un modem téléphonique ou un module GSM/GPRS, avec le protocole AT), avec un instrument de musique électronique (les synthétiseurs et boîtes à rythmes utilisent le protocole MIDI) ou un terminal utilisateur (on peut dialoguer avec une machine UNIX avec le protocole VT100, l'évolution électronique du télétype).

Dans le cas de nano-ordinateurs comme le Raspberry Pi, la liaison série permet justement d'accéder à un autre terminal, qui ne passe ni par l'écran ni par le réseau, ce qui permet de reconfigurer le système si aucune de ces interfaces ne fonctionne. Pour les cartes Arduino, le port série permet de reprogrammer la carte puis de dialoguer avec l'ordinateur hôte.

En fait, ces rôles traditionnels subsistent même si la communication n'est pas réellement sous forme série. C'est juste le plus petit dénominateur commun : un flux d'octets asynchrone, souvent bidirectionnel, donc très pratique pour le stockage. Par exemple, les fichiers .MID contiennent les informations transmises par un port MIDI. Et sous UNIX et ses dérivés (Linux, xBSD et MacOSX) on accède aux ports série au moyen de fichiers virtuels, ce qui est beaucoup plus simple que l'interface USB.

Le défaut principal d'un port série est que sa simplicité complexifie le contrôle de flux, comme par exemple l'annonce du début d'un paquet ou la distinction entre des octets de contrôle et des octets de données. Différentes approches (matérielles et logicielles) existent mais restent cantonnées à des applications spécifiques. Par exemple, le protocole MIDI indique un octet de contrôle avec le bit de poids fort à 1, ce qui réduit toutes les données transmises à 7 bits seulement...

1.1 Comment communiquer avec un port série ?

Tout d'abord, il faut s'entendre sur les caractéristiques du port. Les caractéristiques physiques sont le type de connecteur (DB9, DB25, DIN5...), le genre (mâle ou femelle ?), les tensions et le brochage. Dans le cas d'une communication avec un Pi ou un 'duino, vous devrez vérifier le voltage et le brochage des connecteurs. Une erreur peut empêcher la communication ou même griller vos circuits.

Ensuite il faut s'entendre sur les vitesses de transmission, exprimée en Bauds **[2]**. C'est le nombre de symboles binaires (0 ou 1) transmissibles par seconde, mais ce n'est pas le nombre de bits par seconde car une partie de ces symboles sert à la resynchronisation des données. Chaque modèle de port supporte une ou des vitesses particulières, dont les plus courantes sont :

```
   150 bauds
   300
   600
  1200
  2400
  4800
  9600
 19200
 28800
 31250
 38400
 57600
115200
230400
460800
921600
```

Selon vos circuits, d'autres débits sont possibles, lisez bien les documentations et faites des tests. Sélectionnez la vitesse maximale que l'émetteur et le récepteur ont en commun, mais n'essayez pas d'aller trop vite non plus. D'une part, les circuits et les logiciels risquent de ne pas tenir le rythme, d'autre part les données peuvent être altérées en chemin par les fils. Dans les deux cas, vous perdrez des données, surtout si aucun protocole de gestion de flux n'est utilisé (par exemple XON/XOFF ou CTS/RTS).

Il faut noter qu'un récepteur a besoin d'une horloge au moins quatre fois plus rapide que le débit de bits. Le suréchantillonnage minimise l'effet du déphasage entre les horloges d'émission et de réception, qui créerait des erreurs de transmission.

1.2 Bits de données, start, stop, parité...

Naïvement, chaque bit transmis serait un bit de donnée donc un lien à 115200 bits par seconde devrait fournir 14400 octets par seconde. Il n'en est rien car en plus des temps de réponse du logiciel, la liaison série est aussi ralentie par les informations de synchronisation.

Et d'abord, rien n'oblige de transmettre des octets de 8 bits. On trouve toutes sortes de liaisons avec 5, 6, 7, 8 ou 9 bits par mot. Nous sommes habitués aux mots de 8 bits mais d'anciens ordinateurs (comme les CDC/Cyber) fonctionnaient avec des mots de 6 bits. Certains microcontrôleurs utilisent un 9è bit à des fins d'adressage. Le code ASCII (déjà cinquantenaire) était limité à 128 codes (dont beaucoup sont des codes de contrôle destinés justement aux liaisons séries comme XON, XOFF, EOT...) donc on n'utilisait que 7 bits... Mais les mots de 8 bits sont aujourd'hui de rigueur.

Pour assurer la synchronisation, chaque trame de bits commence par un bit à zéro (le bit START) et finit par un bit à un (bit de STOP). Par convention, l'état inactif d'une liaison série est à un, car cela permet de détecter que l'appareil est bien connecté. En cas de déconnexion, le récepteur recevrait une suite infinie