INFO2 : INFORMATIQUE EMBARQUEE
CM : Description de la liaison série asynchrone

1 Liaison et bus

Comme en témoigne la **Figure 1**, la communication entre deux systèmes d'informatique industrielle peut se faire via un **bus** (RS485, I2C, SPI, CAN, USB, Ethernet, etc.) ou bien via une **liaison point à point** (c'est à dire exclusive) entre ces deux systèmes (RS232, Centronics, etc.). Un bus est plus complexe à mettre en œuvre car il suppose un **adressage** (exemple : bus I2C) ou une **sélection** (exemple : bus SPI). Une liaison présente l'avantage de la simplicité mais son côté exclusif ne permet pas d'ajouter des systèmes. Ces deux solutions ne s'opposent pas, elles coexistent au sein des installations.

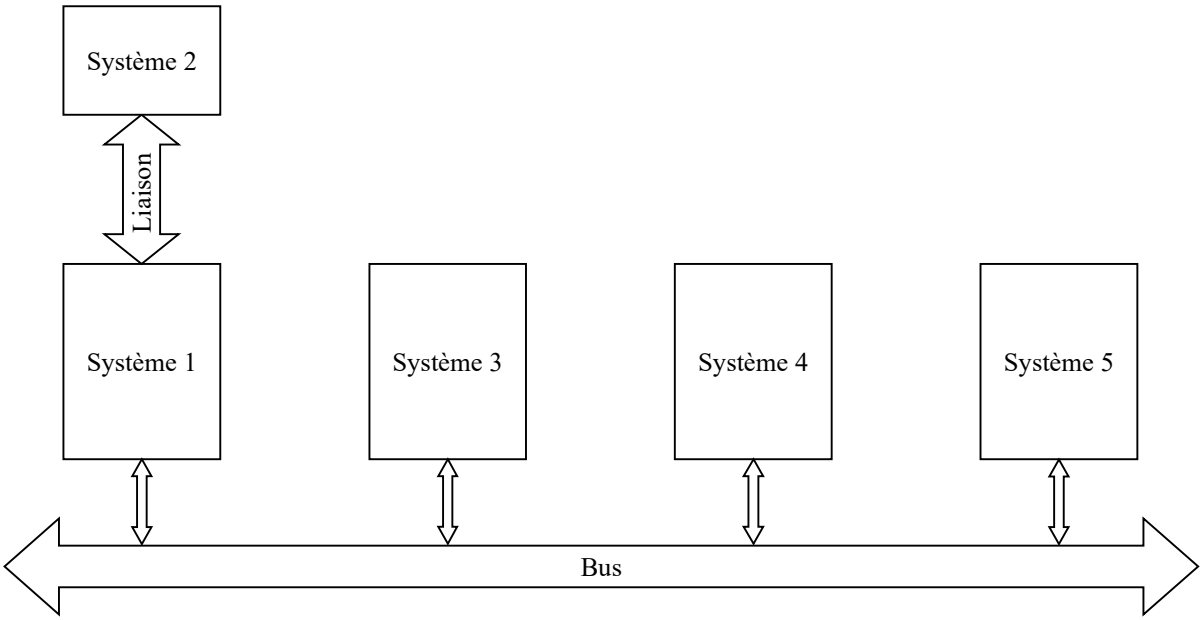

Figure 1. Liaison (en haut) et bus (en bas)

2 Liaison parallèle et liaison série

On se préoccupe de faire communiquer deux systèmes, un PC et un microcontrôleur, via une **liaison** entre eux. Dans ce cadre, si l'information à transmettre est codée en **octets** (8 bits repérés par D7 à D0), deux possibilités s'offrent à nous : **émettre / recevoir les octets en parallèle** ou **émettre / recevoir les octets en série**.

2.1 Liaison parallèle

Dans ce cas, il faut 8 conducteurs de données et un conducteur pour la masse comme indiqué sur la **Figure 2**. Notons qu'avec cette configuration, un seul sens d'échange est possible à un moment donné (sinon il faut doubler le nombre de conducteurs de données). Cette solution était par exemple celle utilisée pour relier un PC et une imprimante entre eux jusqu'en l'an 2000 environ, elle était connue sous le nom de « **liaison Centronics** ». On s'en doute, sur de grandes longueurs, une transmission parallèle est gourmande en cuivre ; en outre, elle pose des problèmes de diaphonie (perturbation des signaux entre eux) et de désynchronisation des bits. Hormis des cas spécifiques (la partie d'un circuit imprimé autour d'un processeur par exemple), on ne trouve plus de liaison parallèle pour faire communiquer des systèmes.

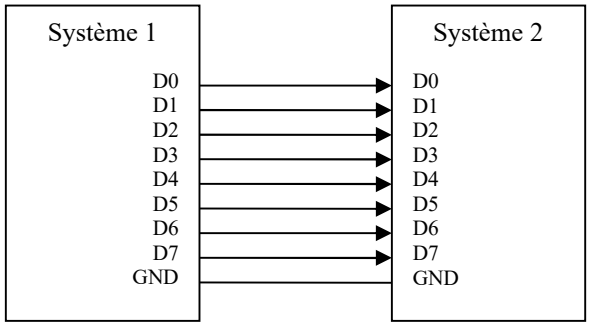

Figure 2. Liaison parallèle unidirectionnelle (les éventuels signaux de synchronisation ne sont pas représentés)

2.2 Liaison série asynchrone

2.2.1 Principe

On peut plutôt choisir de transmettre **séquentiellement** les bits constituant les octets d'information. A l'émission, les octets sont **sérialisés** pour être envoyés bit par bit. A la réception, les bits sont **désérialisés** pour reconstituer les octets d'origine.

Comme en témoigne le schéma de principe de la **Figure 3**, au niveau électronique, ce sont des **registres à décalage** (*shift registers*) qui s'occupent des opérations de sérialisation (à l'émission) et de désérialisation (à la réception).

Dans le **registre d'émission**, on place l'octet à émettre. Celui-ci est recopié dans le registre à décalage associé. A chaque front de l'horloge, il y a un décalage et le bit qui déborde (à droite) est déposé sur la broche de sortie. Celle-ci présente donc successivement la valeur de D0 puis celle de D1 puis celle de D2 etc.

Avec un principe symétrique, dans le **registre de réception** figure l'octet émis par l'autre système. Cet octet a été progressivement reconstitué grâce au registre à décalage associé qui a décalé successivement les bits reçus.

Sur un tel dispositif, la broche d'émission s'appelle en général **Tx** (*Transmit*) et celle de réception **Rx** (*Receive*).

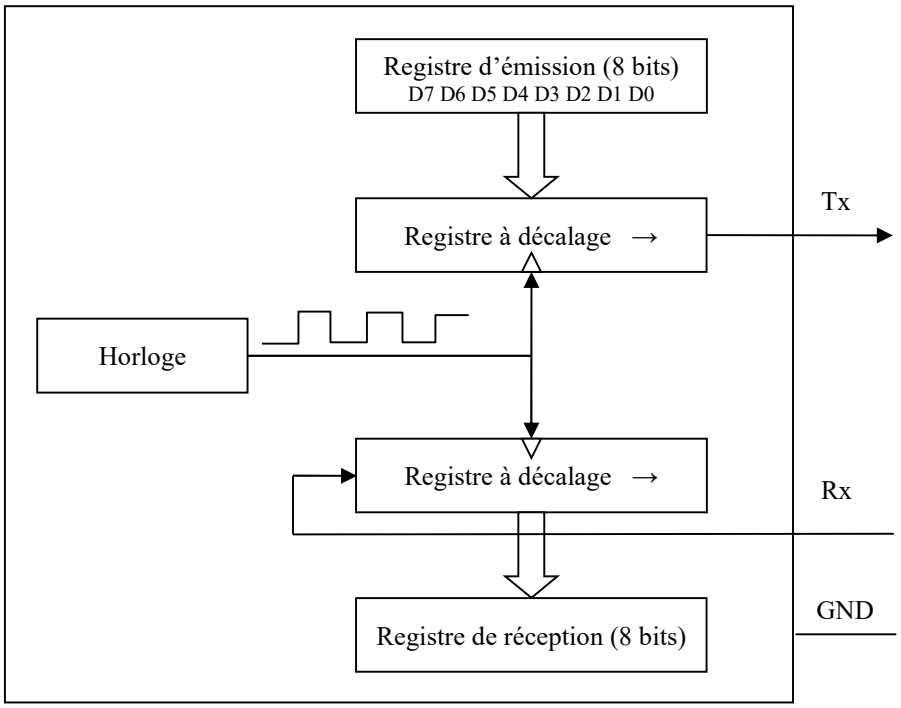

Figure 3. Principe de l'électronique permettant l'émission / réception série

Ainsi, pour constituer une liaison série bidirectionnelle[1], chacun des deux systèmes est équipé de l'électronique de la **Figure 3**, la broche Tx du premier est connectée à la broche Rx du second, de même la broche Tx du second est connecté à la broche Rx du premier. Trois conducteurs suffisent ; c'est ce que traduisent la **Figure 4** et la **Figure 5**.

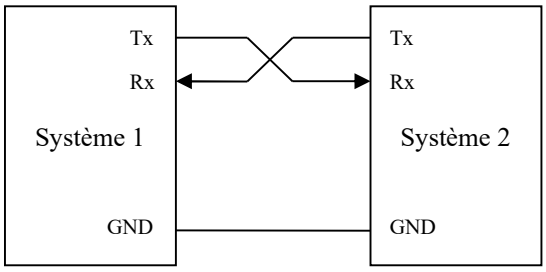

Figure 4. Liaison série bidirectionnelle entre deux systèmes

[1] Une liaison où la communication est possible dans les deux sens est dite bidirectionnelle. Si les deux sens peuvent avoir lieu en même temps, elle est dite ***full duplex*** (analogie : discussion téléphonique), sinon elle est dite ***half duplex*** (analogie : discussion par talkie-walkie).

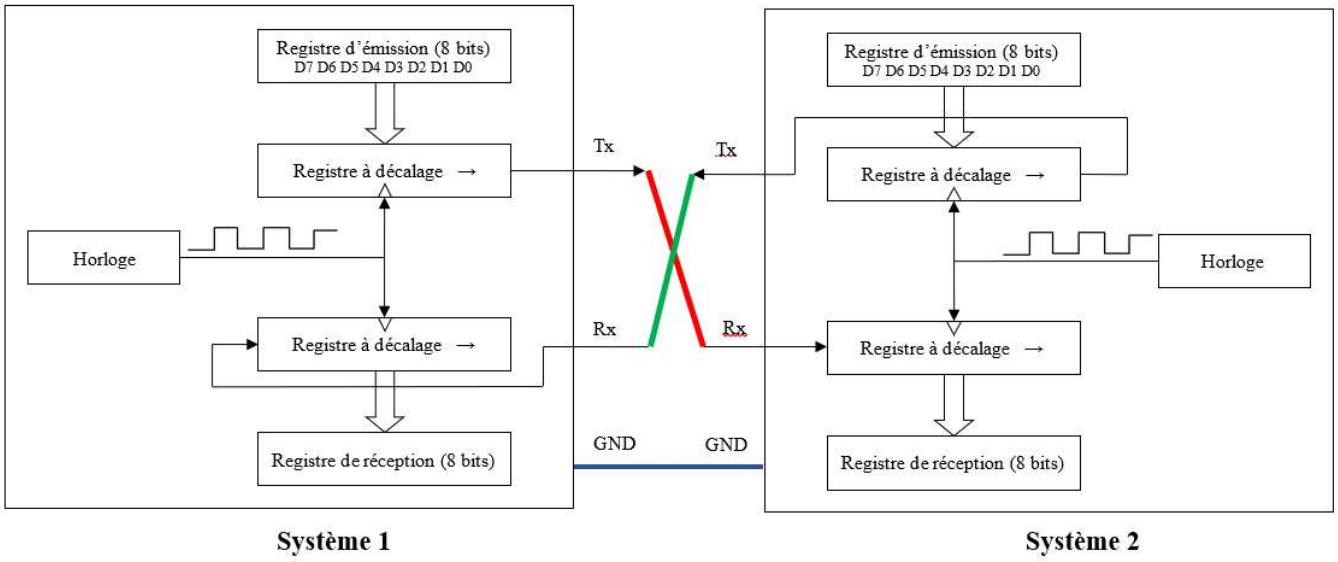

Figure 5. Liaison série bidirectionnelle (en détail)

2.2.2 Comment synchroniser l'échange entre les deux systèmes ?

Prenons le cas de l'émission d'un octet du Système 1 vers le Système 2. Le Système 2 doit :

- détecter que sur sa broche Rx le **premier** bit de l'octet transmis arrive ;
- ensuite être capable de séparer chaque nouveau bit qui arrive.

La solution retenue pour répondre à ces deux contraintes est la suivante :

1. **Détection par le récepteur qu'un octet va arriver :** quand aucun octet n'est transmis, la ligne, ici dans notre exemple Tx1-Rx2, est « au repos », c'est-à-dire qu'elle présente un niveau logique défini, disons « 1 ». Pour annoncer qu'un octet va être transmis, le Système 1 envoie un bit spécial, dit **bit de *start*** qui consiste à mettre sur la ligne le niveau logique opposé à celui du repos donc ici un « 0 ». Voyant que la ligne a quitté son état de repos, le Système 2 sait que le prochain bit qui arrivera après le bit de *start* sera le premier de l'octet de données, c'est-à-dire D0.
2. **Séparation des bits constituant l'octet :** les deux systèmes ont été préalablement configurés avec **les mêmes vitesses de transmission et réception**, par exemple 9600 bits/s [2]. Ils « savent » donc tous les deux que la durée de transmission d'un bit est de 1/9600 s soit 0.104 ms. Le bit de *start* dure donc 0.104 ms, puis c'est le bit D0 qui est transmis pendant 0.104 ms, puis D1 pendant la même durée, etc. C'est le bloc horloge de la **Figure 3** qui assure ce rythme. Même si tout n'a pas encore été expliqué à ce stade, on peut jeter un œil au chronogramme de la **Figure 9** pour comprendre ces premières considérations.

On voit donc que dans la liaison présentée, **il n'y a pas d'horloge transmise**. Dans ce cas, on parle de **liaison série asynchrone**. Sur un système, l'électronique implémentant les fonctions d'émission / réception de la **Figure 3** s'appelle alors une **UART** (*Universal Asynchronous Receiver Transmitter*). À l'origine c'était une puce dédiée (par exemple UART MC68681 pour la famille Motorola 68000, UART 16550 pour le monde PC), maintenant c'est un peu de silicium au sein d'une puce complexe (ASIC, FPGA, microcontrôleur, etc.).

Remarque : Signalons qu'il existe aussi des **communications séries synchrones**. Dans ce cas **un signal d'horloge est transmis**. À chaque front montant (ou descendant, c'est une convention à choisir) de celle-ci, un nouveau bit est transmis. Les bus I2C et SPI sont des exemples de bus série synchrones. Les signaux d'horloge générés par le maître s'appellent respectivement SCL et SCK.

3 La liaison série asynchrone

La liaison série asynchrone qui s'est imposée dans le monde PC et dans celui de l'instrumentation dès les années 1980 est régie par la norme « **RS232** » (RS pour *Recommended Standard*), parfois également rencontrée sous le sigle « V24 ». Cette norme définit les aspects **protocolaire (entre autres la définition de la trame série asynchrone)**, **électrique** et **mécanique** de la liaison.

Certains systèmes implémentent encore les 3 aspects du standard RS232 mais souvent, il ne reste que l'aspect « définition de la trame série asynchrone ».

[2] Comme un bit est bivalent (0 ou 1), une vitesse de 1 bit/s correspond à 1 baud.
[3] Dans certains microcontrôleurs (c'est le cas du MCF5272), on peut avoir une horloge pour la partie émission et une autre pour la partie réception.

3.1 État des lieux

3.1.1 RS232 sur les PC

À l'origine, les PC étaient équipés la plupart du temps de deux interfaces RS232. Par exemple, une servait à connecter la souris et l'autre le modem. Elles étaient implémentées sur une carte à enficher dans le bus ISA (puis PCI) des PC. Ensuite, elles firent partie du *chipset* de la carte mère. Une photo du connecteur utilisé, dit « DB9 », est donnée **Figure 15**.

Sous Windows, les interfaces RS232 sont repérées par « COM1 », « COM2 », etc. (d'où le nom très répandu de « ports COM ») et sous Linux par « ttyS0 », « ttyS1 », etc.

Depuis 2005, avec l'avènement de l'USB, seul les PC « Tour » disposent encore d'une interface RS232 mais si besoin il existe des adaptateurs USB/RS232 d'une vingtaine d'Euros (cf. **Figure 6**), souvent conçus à base de puce FTDI (encapsulation/décapsulation de la trame série dans/depuis USB) et de circuit MAX232 (adaptation des tensions).

Figure 6. ***USB to serial adapter***

Sur la **Figure 7**, on trouve une copie du gestionnaire de périphériques Windows lorsqu'un adaptateur USB/RS232 a été connecté. L'OS a affecté le numéro 5 au port COM émulé.

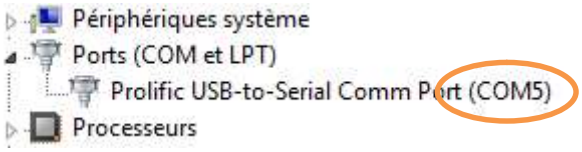

Figure 7. Utilisation du gestionnaire de périphériques Windows pour trouver le numéro affecté au port COM émulé

3.1.2 RS232 dans l'industrie

La liaison RS232 reste encore très utilisée dans l'industrie, dans les systèmes automatisés notamment (transfert de Grafcet, programmation de machines-outils). Dans le domaine des réseaux, la plupart des routeurs (*routers*) et commutateurs (*switches*) se configurent initialement par une liaison RS232.

Elle permet en outre des longueurs de câbles intéressantes, bien-sûr en lien avec la vitesse de transmission, comme en témoigne le **Tableau 1** :

Vitesse (bauds ou bit/s)	Longueur max. de la liaison (m)
2400	60
4800	30
9600	15
19200	7.6
38400	3.7
57600	2.6

Tableau 1. Longueurs maximales d'une liaison RS232 en fonction de la vitesse de transmission (préconisations Cisco, citées dans Wikipédia)

3.1.3 Liaison série asynchrone dans les microcontrôleurs

Nous n'employons plus ici l'expression « RS232 » car dans les microcontrôleurs seul l'aspect « définition de la trame série asynchrone » est respecté.

Tous les microcontrôleurs disposent d'une ou deux interfaces séries asynchrones pour communiquer avec l'extérieur (un PC typiquement). En langage C, le flux standard de sortie *stdout* est généralement la broche Tx d'une des interfaces série asynchrone et le flux standard d'entrée *stdin* correspond généralement à la broche Rx de la même interface série. Très simples à mettre en œuvre, les trames séries asynchrones peuvent être en outre encapsulées dans des trames Bluetooth et USB, ce qui explique le succès toujours présent de cette liaison malgré son âge avancé et ses faibles débits. Signalons aussi que les puces GPS envoient leurs informations de position à un microcontrôleur via une telle liaison (cf. **Figure 8**).

Figure 8. Une puce GPS communiquant par liaison série asynchrone 5 V

3.2 Aspect protocole : la trame série asynchrone

3.2.1 Constitution de la trame

Nous avons déjà commencé à évoquer comment un octet était transmis. Terminons cette présentation ici en s'appuyant sur la **Figure 9** :

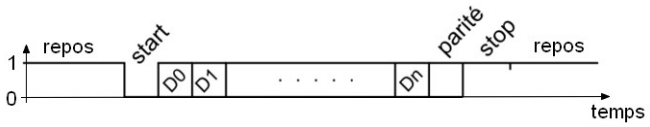

Figure 9. Constitution d'une trame série asynchrone

Dans une trame série asynchrone, le niveau logique de repos est 1. En outre, une telle trame est constituée des bits suivants :

- un **bit de *start*** toujours à 0 : il sert à la synchronisation du récepteur ;
- les **bits de données** : leur nombre peut varier entre 5 (cas d'un télex, ancêtre du fax) et 9 mais la plupart du temps c'est 8 car on transmet essentiellement des octets entre équipements. La transmission de caractères alphanumériques se fait en utilisant le code ASCII dont une table est donnée à la partie **4** ;
- éventuellement un **bit de parité paire ou impaire** (étudié un peu plus loin) ;
- un **bit de *stop*** toujours à 1 (la durée peut être réglée entre 1, 1.5 et 2 temps bit).

Les deux équipements souhaitant communiquer ensemble doivent bien-sûr observer : la même vitesse, le même nombre de bits de données, la même parité, le même nombre de bits de *stop* et le même type de contrôle de flux (étudié un peu plus loin).

3.2.2 Vitesses

Les vitesses les plus courantes sont, en bits/s ou bauds : 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 et 115200. Si on n'utilise pas de bit de parité, il faut 10 bits pour transmettre un octet (un de *start*, 8 de données, 1 de *stop*) et avec la vitesse maximale de 115200 bauds par exemple, le débit n'est au mieux que d'environ 10 ko/s octets par seconde. De manière générale, la liaison série asynchrone n'est pas une liaison rapide. À titre de comparaison, on obtient 1.5 Mo/s pour l'USB 1.1 et 60 Mo/s pour l'USB 2.0 mais l'USB est bien plus compliqué à mettre en œuvre.

3.2.3 Bit de parité pour le contrôle de parité

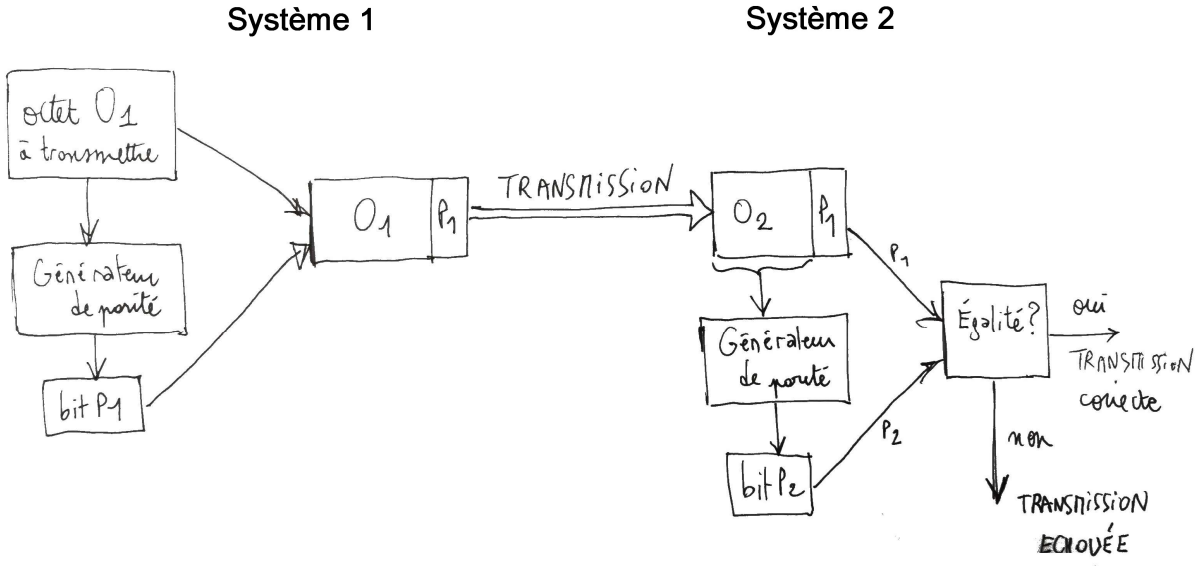

Figure 10. Contrôle de parité

Raisonnons sur des données de 8 bits pour cette partie. Le **bit de parité**, optionnel, est un bit supplémentaire qui est transmis par l'émetteur afin de contrôler côté récepteur la transmission sans erreur de l'octet de données (*parity check*). Le schéma général est donné **Figure 10**.

Pour les deux systèmes communiquant via une liaison RS232, il faut choisir la parité, paire ou impaire :

- Lorsque la **parité** est **paire**, le neuvième bit est généré de façon à ce que les 9 bits (8 bits de données + bit de parité) comportent un nombre pair de bits à 1.
- Lorsque la **parité** est **impaire**, le neuvième bit est généré de façon à ce que les 9 bits (8 bits de données + bit de parité) comportent un nombre impair de bits à 1.

Quand le récepteur reçoit une trame, il génère à son tour à partir des 8 bits de données reçus le bit de parité (avec une parité paire ou impaire, selon ce qui a été initialement choisi). Il compare ce neuvième bit qu'il vient de générer avec celui qu'il a reçu. S'ils sont identiques, le récepteur valide la transmission sinon il en déduit que celle-ci s'est mal passée. C'est un moyen très rudimentaire de contrôle : en effet, un nombre pair d'erreurs sur les bits de données n'est pas détectable. Cela dit, la plupart du temps, il y a au pire un bit mal transmis, par conséquent le taux de détection d'erreurs est très bon en regard de la simplicité de la technique.

Remarque : quand on transmet un grand nombre d'octets on préfère ajouter en fin de transmission un ou plusieurs octets[4] constituant une **somme de contrôle** (*checksum*). À la réception, un bout de code régénère chaque *checksum* et le compare à celui qu'il a reçu pour déduire si la transmission s'est faite correctement ou pas. C'est typiquement le cas d'Ethernet.

3.2.4 Notation de la configuration

Nous l'avons bien compris, pour pouvoir communiquer, les deux systèmes connectés doivent partager les mêmes paramètres de communication. Ces paramètres sont souvent notés de façon compacte. Par exemple :

« 57600,8,N,1 » signifie 57600 bit/s, 8 bits de données, pas de parité (N = *None*), 1 bit de stop.

Les autres valeurs possibles pour le paramètre de parité sont : O = *Odd* (impair) ou E = *Even* (pair).

Remarque : S'il est utilisé, le contrôle de flux (étudié un peu plus loin) doit aussi être précisé.

3.2.5 Exercice

- ✓ Représenter le chronogramme du signal logique présent sur la broche Tx d'une UART émettant le caractère 'r'. Cette UART est configuré avec « 9600,8,N,1 ». Combien de temps a pris cette transmission ?
- ✓ Mêmes questions avec les paramètres « 57600,8,E,1 ».

[4] Pour sa génération, chercher sur Internet « CRC », c'est-à-dire « Contrôle de Redondance Cyclique ».

3.2.6 Contrôle de flux

3.2.6.1 Propos

Pour illustrer le propos du **contrôle de flux**, considérons la **Figure 11** avec un échange de Système 1 vers Système 2. Supposons que Système 1 envoie des octets à un rythme soutenu. Système 2 les range dans une mémoire tampon au fur et à mesure de leur réception afin de les traiter plus tard car il a une tâche plus importante à exécuter. Quand la mémoire tampon est presque pleine, Système 2 doit le signaler à Système 1 afin que ce dernier cesse d'envoyer des octets. Quand Système 2 aura vidé suffisamment la mémoire tampon, la transmission pourra reprendre et il le fera savoir à Système1. Dans la situation décrite, on dit que Système 2 **contrôle le flux**.

Le contrôle de flux est optionnel. Si on choisit de l'activer, il est soit « logiciel » soit « matériel ». Les deux systèmes doivent bien sûr s'entendre sur le même type de contrôle de flux (ou son absence).

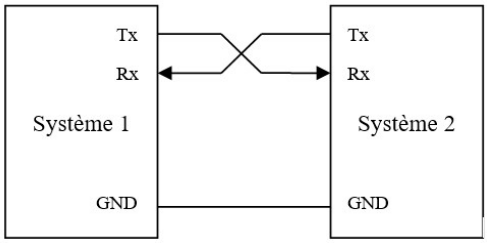

Figure 11. Échange de Système 1 vers Système 2

3.2.6.2 Contrôle de flux logiciel

Pour autoriser ou suspendre la transmission d'octets, le plus simple est de convenir d'un code spécial pour l'autorisation et d'un autre pour la suspension. Ces codes se nomment XON et XOFF, ils correspondent aux codes ASCII non imprimables de valeurs décimales 17 et 19. Quand Système 2 veut suspendre l'émission, il envoie le caractère XOFF, quand il souhaite qu'elle reprenne, il envoie le caractère XON. Sous un émulateur de terminal (décrit à la partie **3.2.7**), on provoque XON et XOFF au clavier par CTRL+Q et CTRL+S.

3.2.6.3 Contrôle de flux matériel

Le contrôle de flux matériel nécessite d'utiliser deux broches supplémentaires (une pour Système 1, une pour Système 2). Comme elles ne sont pas encore connues, cette technique sera abordée plus loin, à la partie **3.4.6**.

3.2.7 Qu'est-ce qu'un émulateur de terminal ?

Un terminal est un poste informatique dénué de toute intelligence. Il comporte un **écran texte** pour afficher des caractères reçus et un **clavier** pour en envoyer. Il y en avait beaucoup dans les usines pour consulter les stocks, dans les bibliothèques pour savoir si un livre était disponible. La communication avec l'ordinateur central disposant de l'information se faisait souvent par une liaison RS232. Le minitel était un terminal à 1200 bauds associé à un modem.

On ne vend plus de terminaux aujourd'hui mais on peut les émuler sur PC par des logiciels gratuits comme **GtkTerm** et **Minicom** sous Linux ou **HyperTerminal**, **TeraTerm** et **Putty** sous Windows. Tous les paramètres de la liaison qui viennent d'être expliqués dans cette partie **3** sont configurables dans un émulateur de terminal comme en témoigne la **Figure 12**.

Figure 12. Configuration d'une interface RS232 de PC sous l'émulateur de terminal GtkTerm

Les émulateurs de terminal permettent en outre d'envoyer sur la liaison série des paquets d'octets préparés au sein d'un fichier.

3.3 Aspects électriques

Nous sommes amenés ici à distinguer deux cas :

- Soit le PC communique avec le microcontrôleur avec une vraie RS232 (prise DB9 en sortie de PC). C'est cela dit de moins en moins le cas en 2020.

- Soit le PC n'a que des ports USB et dans ce cas, un circuit adaptateur est utilisé pour qu'il communique avec le microcontrôleur. C'est le cas le plus fréquent.

3.3.1 Le PC communique avec une vraie RS232

Contre toute attente, sur les broches Tx et Rx d'une UART RS232 :

- Le niveau logique 1 est matérialisé par une tension entre -8 et -12 V.
- Le niveau logique 0 est matérialisé par une tension entre +8 et +12 V.

La différence entre un 0 et un 1 est donc d'au moins 16 V, ce qui diminue la sensibilité aux parasites et permet de plus grandes longueurs de liaison.

Cela dit, beaucoup de systèmes (microcontrôleurs par exemple) sont alimentés avec des tensions de +5 V ou de +3.3V. Ils ne peuvent donc générer les niveaux électriques requis par le standard RS232. Pour ces systèmes, un 0 logique correspond à peu près à 0 V et un 1 logique correspond à peu près à leur tension d'alimentation. Un PC équipé de vrais « ports COM » gère par contre de vrais niveaux de tension RS232 sur les broches Tx et Rx de ses UART RS232.

Si on veut faire communiquer un PC via son interface RS232 avec un microcontrôleur alimenté en +5 V, il y a donc lieu d'adapter les niveaux de tensions comme indiqué sur la **Figure 13**. Sur celle-ci, A1 transforme le +12 V en 0 V (0 logique) et le -12 V en +5 V (1 logique). Quant à A2, il transforme le 0 V en +12 V (0 logique) et le +5 V en -12 V (1 logique).

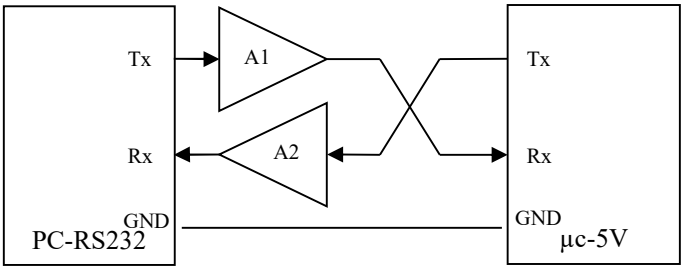

Figure 13. Adaptation des niveaux de tensions entre UART RS232 d'un PC et interface série d'un microcontrôleur 5V

Le circuit MAX232 (de Maxim Integrated) comporte deux adaptateurs de type A1 et deux adaptateurs de type A2. Il a été vendu à des millions d'exemplaires car il est capable de générer le -12 V et le +12 V tout en étant alimenté en +5 V. Il fonctionne avec la technique de pompe de charge et nécessite donc quelques condensateurs extérieurs. Il est décliné en tout un tas de variantes : pour le 3.3 V, sans nécessité de condensateurs extérieurs, etc.

3.3.2 Le PC communique par un de ses ports USB

En 2020, peu de PC ont encore une interface RS232 mais ils ont tous un port USB disponible. On ne conserve de RS232 que l'aspect définition de la trame, mais pas les niveaux de tension, trop contraignants. Une puce FTDI FT232RL (ou CP2102, CH340, ATmega8U2, etc.) encapsule/décapsule les trames série dans/depuis de l'USB. Les niveaux de tensions sont ceux de l'USB du PC (5 V) et ceux du microcontrôleur (3.3 V ou 5 V). La **Figure 14** montre deux de ces puces insérées dans des montages.

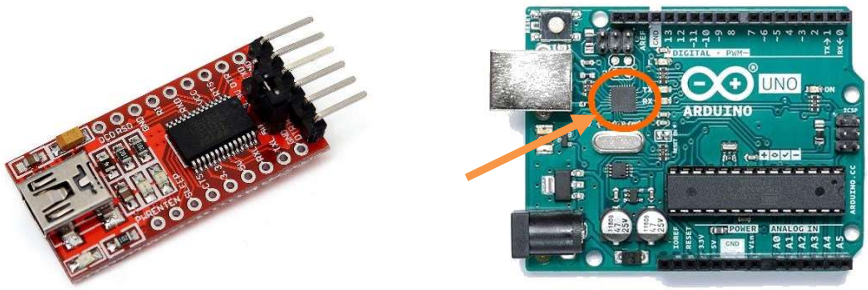

Figure 14. Interfaçage USB-Série – à gauche : séparé (puce FTDI) / à droite : intégré dans un carte Arduino Uno (puce ATmega)

3.4 Aspects mécaniques, brochage et liaisons possibles

Remarque : Dans ce qui suit, on étudie **pour mémoire** certains aspects du véritable standard RS232 (protocole complet, niveaux de tensions, mécanique).

3.4.1 Aspects mécaniques et brochage

Une interface RS232 se présentait mécaniquement sous la forme d'une prise DB25 mâle. IBM, avec son PC AT en a fait une adaptation vers une prise DB9 mâle comme sur la photo de la **Figure 15**.

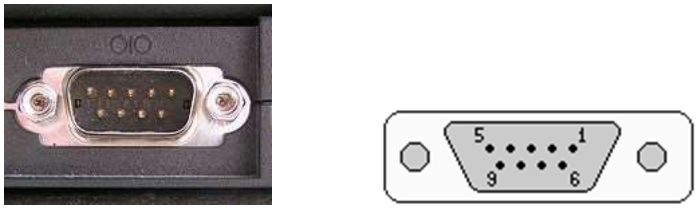

Figure 15. Prise DB9 mâle pour interface RS232

Le brochage est donné dans le **Tableau 2**.

Broche n°	Nom	Entrée ou Sortie ?	
		DTE	DCE
1	DCD	E	S
2	Rx	E	S
3	Tx	S	E
4	DTR	S	E
5	GND		
6	DSR	E	S
7	RTS	S	E
8	CTS	E	S
9	RI	E	S

Tableau 2. Brochage de la prise DB9 mâle d'une interface RS232

Dans ce tableau, on voit qu'il y a beaucoup plus de broches que les trois que l'on avait évoquées jusqu'à présent. On voit aussi apparaître les acronymes **DTE** et **DCE**. Cela tient aux origines de la liaison RS232. Elle servait initialement à connecter un ***Data Terminal Equipment***[5] (par exemple un ordinateur, capable d'initier une communication avec un DCE) avec un ***Data Communication Equipment***[6] (un périphérique du type modem, incapable d'initier une communication avec un DTE). Les détails de connexion sont donnés dans les parties suivantes.

Les interfaces RS232 avec toutes les broches sont dites « **complètes** », c'est le cas de celles des PC. Au niveau des microcontrôleurs, on trouve plus souvent les interfaces à trois broches déjà présentées (Rx, Tx, GND).

3.4.2 Liaison DTE-DCE

Un DTE et un DCE sont connectés comme sur la **Figure 16**. Remarquons bien qu'une broche d'un nom donné sur un DTE est connectée à la broche de même nom sur le DCE.

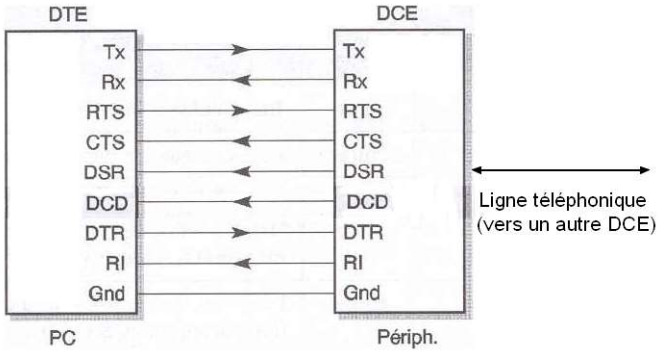

Figure 16. Liaison entre un DTE et un DCE

Sans rentrer dans les détails, voici la description de l'utilité des **six broches supplémentaires**. Sa lecture permet de se faire une idée du protocole de communication entre un PC et un modem via RS232 :

- DSR (*Data Set Ready*, modem prêt) est au niveau actif quand le DCE est alimenté et raccordé à une ligne téléphonique ;
- DTR (*Data Terminal Ready*, ordinateur prêt) est actif quand le DTE est prêt ;
- RTS (*Request To Send*, demande d'émission) est activé par le DTE lorsqu'il veut envoyer des données ;
- CTS (*Clear To Send*, prêt à émettre) est activé par le DCE lorsqu'il a établi la liaison et est prêt à recevoir du DTE les données à transmettre sur la ligne téléphonique ;
- DCD (*Data Carrier Detect*, porteuse détectée) est activé par le DCE lorsqu'il reçoit une porteuse provenant d'un autre DCE ;
- RI (*Ring Indicator*, indicateur d'appel) est activé par le DCE lorsqu'il reçoit un signal de sonnerie.

[5] En français, DTE = ETTD = Equipement Terminal de Traitement de Données.
[6] En français, DCE = ETCD = Equipement Terminal de Circuit de Données.

Remarque : un niveau actif sur ces 6 broches correspond à + 12 V et un niveau inactif à -12 V.

3.4.3 Liaison DTE-DTE complète, dite « Null Modem complète »

Aujourd'hui, la liaison série asynchrone sert surtout à faire communiquer deux DTE entre eux, par exemple deux PC ou un PC et un microcontrôleur. Dans ce cas, on utilise la liaison ***Null Modem***[7] de la **Figure 17**. Elle correspond à certains « câbles croisés série » que l'on pouvait acheter dans le commerce pour échanger des fichiers par exemple.

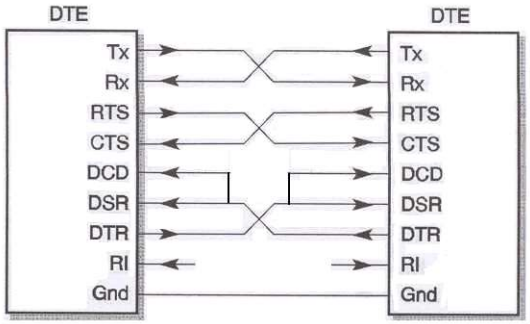

Figure 17. Liaison DTE-DTE complète (Null Modem complète)

3.4.4 Liaison DTE-DTE simplifié, dite « Null Modem 3 fils »

Il est possible de faire communiquer deux DTE par une liaison *Null Modem* simplifiée dite « 3 fils ». Les broches supplémentaires ne doivent *a priori* pas être laissées en l'air (sauf RI), la liaison ne fonctionnerait pas. La **Figure 18** illustre une possibilité pour ce type de liaison. Il existe au moins une variante de ce câblage *Null Modem*.

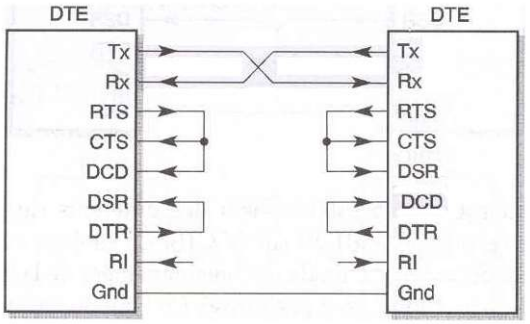

Figure 18. Liaison DTE-DTE simplifiée (Null Modem 3 fils)

3.4.5 Liaison entre un DTE à interface complète et un DTE à interface trois broches

Un « petit DTE » (un microcontrôleur par exemple) ne possède souvent qu'une interface série à trois broches. Une possibilité pour l'interfacer avec un DTE à interface complète est donnée **Figure 19** (à gauche). Sur celle-ci, le point d'interrogation est présent pour s'interroger sur la nécessité d'une adaptation des niveaux de tension. Côté DTE complet, les broches supplémentaires ne doivent *a priori* pas être laissées en l'air (sauf RI), la liaison ne fonctionnerait pas. À noter qu'il existe une variante de ce câblage : Par exemple, sur la carte EasyPic v7 de MikroElektronika, on a les connexions DCD-DTR-DSR et RTS-CTS comme en témoigne la **Figure 19** (à droite).

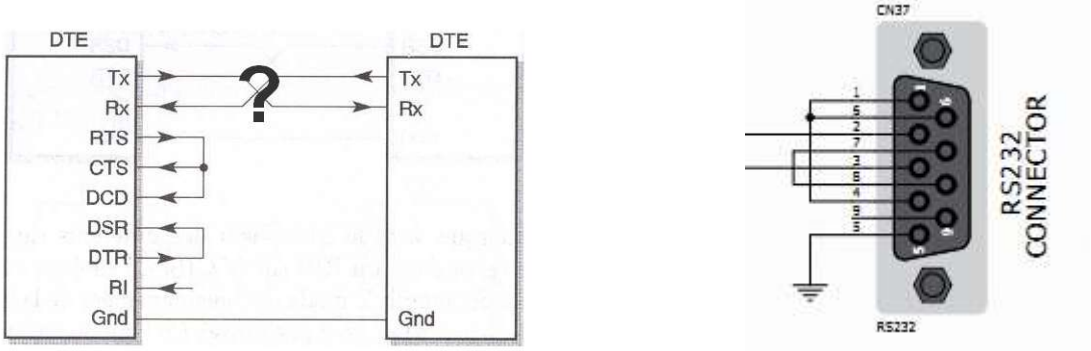

Figure 19. Liaison DTE interface complète - DTE interface 3 broches et variante sur EasyPic v7

[7] « *Null Modem* » doit être lu comme « Il n'y a plus besoin de deux modems pour faire communiquer deux DTE entre eux. ».

3.4.6 Contrôle de flux matériel, suite et fin

Maintenant que nous avons connaissance des broches supplémentaires présentes dans une interface complète (RS232), revenons au contrôle de flux matériel, qui avait été envisagé à la partie **3.2.6.3**.

Si on considère un échange de Système 1 vers Système 2, ce dernier peut signaler à Système 1 de suspendre l'émission en mettant un niveau logique 0 sur sa broche RTS. Système 1 en prend connaissance sur sa broche CTS. Quand l'émission peut reprendre, Système 2 active sa broche RTS. La **Figure 20** montre le câblage employé. À noter que le schéma de la **Figure 17** permet le contrôle de flux matériel mais pas celui de la **Figure 18** ni celui de la **Figure 19**.

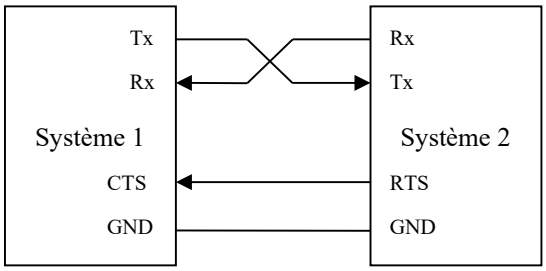

Figure 20. Contrôle de flux matériel

4 Table des codes ASCII

Le jeu de caractères codés **ASCII** (*American Standard Code for Information Interchange*, c'est-à-dire « Code américain normalisé pour l'échange d'information »), dont la table se trouve **Figure 21**, est la norme de codage de caractères en informatique la plus connue, la plus ancienne et la plus largement compatible. ASCII contient les caractères nécessaires pour écrire en anglais.

L'ASCII définit 128 caractères numérotés de 0 à 127 et codés en binaire de 0000000 à 1111111. Sept bits suffisent donc pour représenter un caractère codé en ASCII. Toutefois, les ordinateurs travaillant presque tous sur huit bits (un octet) depuis les années 1970, chaque caractère d'un texte en ASCII est stocké dans un octet dont le huitième bit est 0.

Les caractères de numéro 0 à 31 et le 127 ne sont pas affichables ; ils correspondent à des commandes de contrôle de terminal informatique. Citons le numéro 10, **LF** (*Line Feed* = saut de ligne = **`\n`** en langage C), le numéro 13, **CR** (*Carriage Return* = retour en début de ligne = **`\r`** en langage C). L'utilité des numéros 17 (XON) et 19 (XOFF) a été abordée à la partie **3.2.6.2 Contrôle de flux logiciel**.

Le caractère numéro 32 est l'espace. Les autres caractères sont les chiffres, les lettres latines majuscules et minuscules et quelques symboles comme la ponctuation ou les accolades, parenthèses, etc.

Extension : de nombreuses normes de codage de caractères ont repris les codes ASCII et ajouté d'autres caractères pour les codes supérieurs à 127. En effet, beaucoup de codes étendent l'ASCII en utilisant le huitième bit pour définir des caractères numérotés de 128 à 255. La norme ISO/CEI 8859 fournit des extensions pour diverses langues. Par exemple, l'ISO 8859-1, aussi appelée *Latin-1*, étend l'ASCII avec les caractères accentués utiles aux langues originaires d'Europe occidentale comme le français ou l'allemand (*Source :* Wikipédia).

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Figure 21. Table des codes ASCII