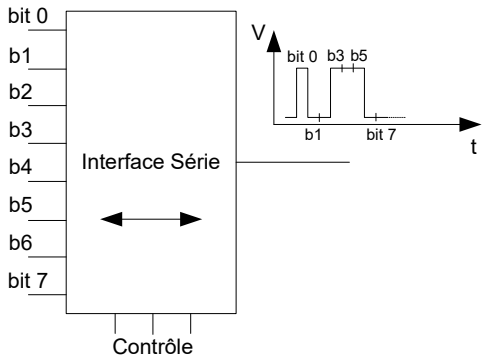
GIF-3002 Interfaces Séries

Ce document présente trois interfaces séries communes dans le monde des microcontrôleurs : le UART, le SPI et l'I2C.

Les interfaces séries communiquent les bits d'un mot parallèle un par un :

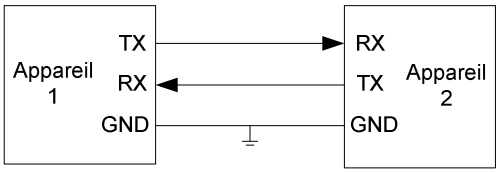


1.1 UART

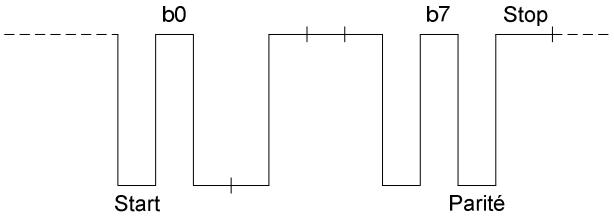
Le UART, Universal Asynchronuous Receiver Transmitter, est un protocole de communication série ayant vu le jour dans les années 1960. Même s'il est très très vieux, il est encore abondamment utilisé de nos en raison de sa simplicité. La quasi-totalité des microcontrôleurs intègrent au moins 1 UART, quand ce n'est pas 2, 3 ou même 4.

Le contrôleur d'UART gère habituellement 3 lignes : TX, RX et GND. TX permet de transmettre, en série, un mot généralement de 8 bits. RX permet de recevoir un mot et GND est le potentiel de référence (0V). La transmission est donc possible dans les deux sens simultanément (Full-Duplex).

Plusieurs contrôleurs d'UART gèrent plus de lignes pour être compatibles avec le RS232 décrit ci-dessous.



L'exemple qui suit illustre la transmission d'un octet entre 2 appareils (le UART est point-à-point) :



Lorsqu'il n'y a pas de données transmises, la ligne est à "1" (3.3V ou 5V). Pour signaler le début de la transmission et synchroniser le receveur avec le transmetteur, un premier "0" precede toujours les données (bit de depart, Start bit). Après ce "0", les données sont transmises du bit le moins significatif au bit le plus significatif (0xB5 est transmis dans l'exemple). Finalement, un bit de parité optionel permet de détecter des erreurs dans l'octet transmis (parité impaire dans l'exemple). Enfin, la ligne est toujours remise à "1" pendant un temps minimum avant de retransmettre un nouvel octet: il s'agit du bit de fin (Stop bit).

La durée d'un bit ainsi que plusieurs autres paramètres (présence et nature de la parité, longueur du bit de stop, nombre de bits dans le mot…) de la transmission est prédéterminée: le transmetteur et le receveur sont programmés individuellement afin d'opérer à la même vitesse.

1.1.1 Contrôleur de UART

Par abus de langage, le UART est un circuit logique, intégré dans le microcontrôleur, qui met les octets en série. Ce circuit logique contient plusieurs registres :

- Un registre pour transmettre un (ou plusieurs) octet(s)
- Un registre pour contenir le(s) dernier(s) octet(s)
- Un registre d'état de la transmission et de la réception
- Un registre pour déterminer la fréquence de communication
- Un ou plusieurs registres pour la parité, la durée du bit de stop, le nombre de bits par mot.
- Des registres pour configurer les lignes de contrôle du RS232 (DTR, RTS, CTS…) si le contrôleur de UART le supporte.

L'exercice 7.6 illustre le fonctionnement d'un contrôleur d'UART.

1.1.2 RS232

Voir SMI_C7_RS232.pdf.

1.1.3 RS485

Le RS485 est décrit dans SMI_C7_RS232.pdf, mais les figures suivantes illustrent le lien de communication :

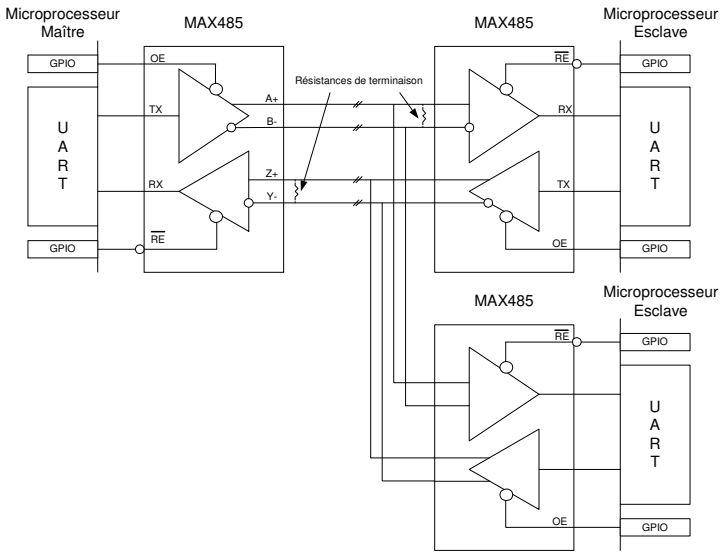


Figure 1 - RS485, FULL-DUPLEX

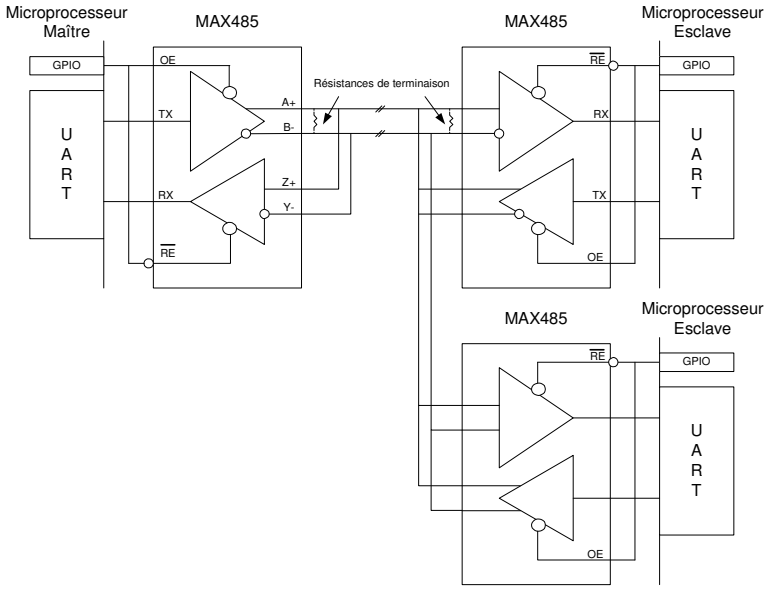


Figure 2 - RS485, HALF-DUPLEX

Voici quelques points additionnels (par rapport à SMI_C7_RS232.pdf) utiles lorsqu'on implémente un lien RS485 :

- Il y a un Receive Enable sur presque tous les transceivers RS485. Le Receive Enable permet d'ôter l'écho lorsqu'on est en half-duplex.
- Lorsque les distances de communications sont grandes, il faut des résistances de terminaison de ligne. La valeur de ces résistances dépend de l'impédance du fils utilisé pour la communication
- Le mode half-duplex est préféré au mode full-duplex parce qu'il faut 2 fils plutôt que 4!!!

- Un truc avancé: il est souvent possible de relié la pin TX du UART à la broche OE (output enable) du transceiver et de mettre la broche TX à 0 pour éviter d'avoir à gérer le signal OE.

1.2 Serial Peripheral Interface

Le SPI est une interface série créée par Motorola (Freescale) pour interconnecter les composantes d'un système périphérique avec un minimum de fils. Le SPI a les caractéristiques suivantes:

- ***Master-Slave*** : Sur un bus SPI, il y a un maître qui initie toutes les communications et plusieurs slave qui ne transmettre que lorsque demandé par le maître.
- ***FULL-DUPLEX*** : La communication entre le maître et un esclave peut se faire simultanément dans les deux sens.
- ***Série*** : Les bits d'un mot sont transmis un à un…
- ***Synchrone*** : Une horloge indique le moment de transmettre un bit et le moment d'échantillonner un bit.

Le SPI n'est pas un protocole établi: il s'agit d'une couche de communication physique qui a été adoptée par de nombreux manufacturiers et qui s'est répandue universellement.

Dans un bus SPI, il y a au moins ***quatre fils*** : SCLK, MOSI, MISO et SS :

- SCLK, Serial CLocK, est l'horloge du bus.
- MOSI, Master Out Slave In, est le fils qui véhicule les données du maître aux esclaves
- MISO, Master In Slave Out, est le fils qui véhicule les données des esclaves au maître.
- SS (actif LOW), Slave Select, est le chip qui permet d'activer ou désactiver l'esclave qui communiquera avec le maître. Lorsqu'il y a plusieurs esclaves, il y a plusieurs lignes de sélection.

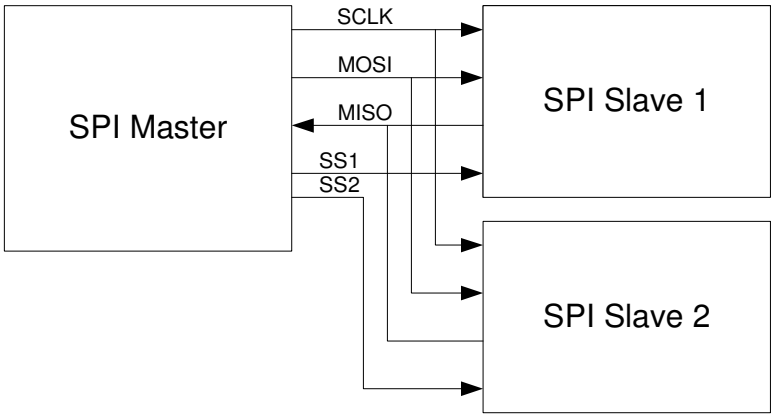


Figure 3 - Bus de communication SPI

Un transfert SPI s'effectue de la manière suivante :

- Le maître active la ligne SS de l'esclave avec lequel il veut parler.
- Le maître génère le signal d'horloge pour 8 bits
    - Pendant une montée ou une descente, le maître et l'esclave mettent une donnée sur MOSI et MISO respectivement.
    - Pendant la descente ou la montée qui suit, le maître et l'esclave lisent la donnée sur MISO et MOSI respectivement.
- Le maître désactive la ligne SS de l'esclave avec lequel il veut parler.

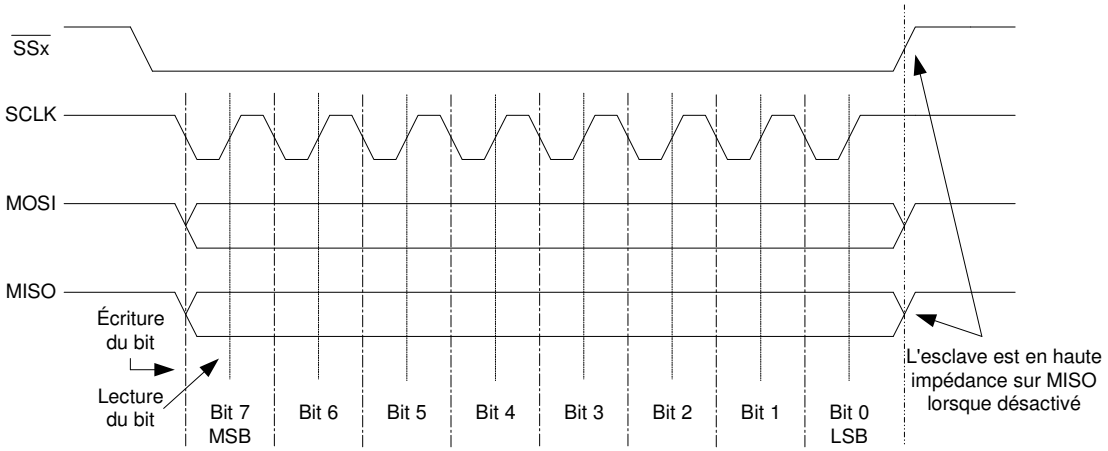


Figure 4 – Exemple de diagramme temporel d'une communication SPI

Dans la plupart des microcontrôleurs, un module de SPI reçoit les octets à transmettre sur le bus SPI et sérialise ces octets. Ce module reçoit aussi les octets et il met les données en parallèle, sur un octet. En mode maître, le module SPI génère le signal d'horloge en fonction de registres de configuration. Par ailleurs, la ligne de sélection du slave est souvent un GPIO indépendant. En mode slave, le microprocesseur attend d'être activé et attend un signal d'horloge avant de répondre au maître.

Si on veut on recevoir des données en mode maître, il faut transmettre des données pour activer l'horloge, même si ce que l'on transmet n'a pas de sens…

1.2.1 Modes SPI

Plusieurs modes SPI existent. Dans ces modes, la phase (CPHA) et la polarité (CPOL) du signal d'horloge (SCK ou SCLK) changent. La première donnée est-elle mise avant le premier coup d'horloge (phase)? La ligne d'horloge est-elle haute (HIGH) ou LOW au début de la transmission d'un octet (polarité)? La figure et la table suivantes décrivent les quatre modes SPI possibles :

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

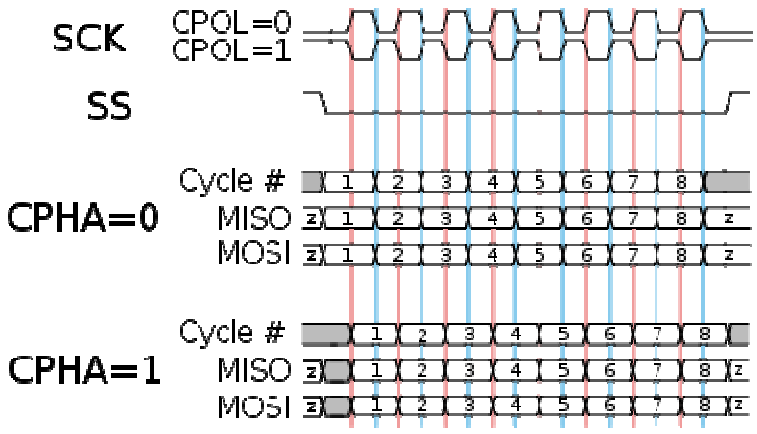


Figure 5 – Modes SPI (License GFDL)

1.3 Inter-Integrated Circuit (I2C)

L'I2C a été inventé au début des années 1980 par Philips (maintenant NXP), également pour interconnecter les composantes d'un système périphérique avec un minimum de fils. L'I2C a les caractéristiques suivantes:

- ***Master-Slave*** : Sur un bus SPI, il y a un maître qui initie toutes les communications et plusieurs slave qui ne transmettre que lorsque demandé par le maître.
- ***HALF-DUPLEX*** : La communication entre le maître et un esclave peut se faire alternativement dans les deux sens.
- ***Série*** : Les bits d'un mot sont transmis un à un…
- ***Synchrone*** : Une horloge indique le moment de transmettre un bit et le moment d'échantillonner un bit.
- ***Supporte plusieurs maîtres, l'arbitration se faisant avec des sorties open-collector.***

Comme le SPI, il n'y a pas de norme officielle pour le I2C. Cependant, le protocole est mieux établi que le SPI : il n'existe qu'une version d'I2C! Il s'agit d'un protocole qui s'est répandu dans les années 1990 par Philips et d'autres gros joueurs de l'industrie.

Il y a seulement deux fils sur le bus I2C : SCL (Serial CLock) et SDA (Serial Data). Ces deux fils relient plusieurs périphériques comme suit :

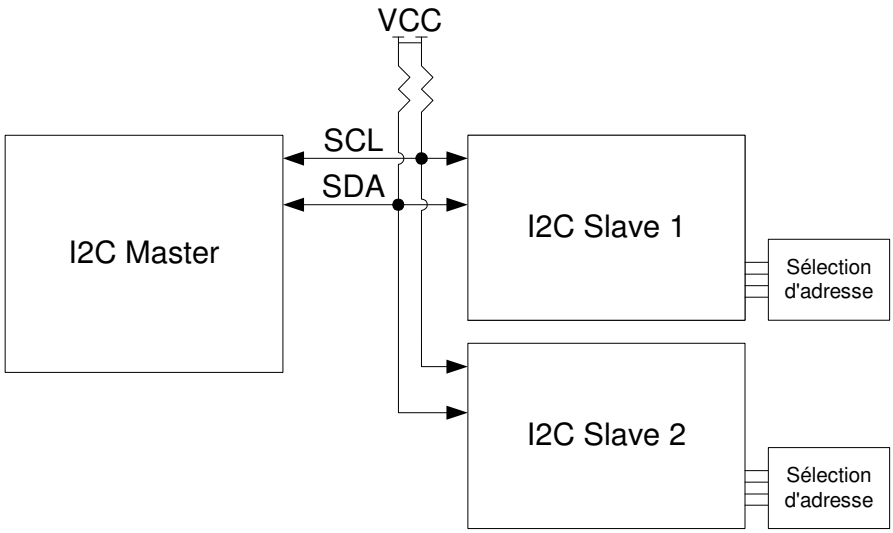


Figure 6 - Bus de communication I2C

Selon le protocole, un maître initie toujours les communications. Les esclaves répondent en fonction de la direction indiquée par le maître et l'adresse de l'esclave qui doit répondre est également indiquée par le maître. Un appareil I2C peut avoir quatre rôle : Maître qui transmet, Maître qui reçoit, Esclave qui transmet et Esclave qui reçoit.

Chaque message I2C débute par une séquence de bits prédéfinis, transmis par le maître : un Start bit (SDA activé –LOW- avant le premier coup d'horloge), 7 bits de d'adresse (on peut déduire que le nombre maximum d'appareil sur le bus est 128!) et un bit de direction. L'esclave, dont l'adresse préétablie se retrouve sur le bus, acquitte le signal.

Dans le protocole I2C, chaque mot a 9 bits : 8 bits de données envoyés par le transmetteur et 1 bit d'acquittement retourné par le receveur. Le bit d'acquittement sert à indiquer si le receveur accepte ou n'accepte pas la communication.

Lorsque la communication dans le sens décrit est terminée, le maître du bus peut redécoller la communication dans un autre sens (Start après Start) ou arrêter la communication avec un bit de Stop (SDA désactivé –HIGH- après le dernier coup d'horloge).

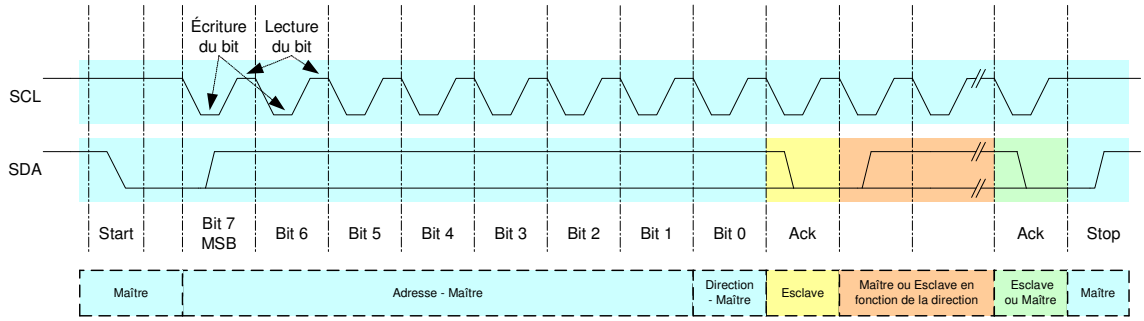


Figure 7 - Diagramme temporel d'une communication I2C

1.3.1 La couche physique

Les appareils connectés sur l'I2C contrôlent les deux lignes avec des drains ouverts (open drain ou open collector). Lorsqu'un appareil impose un « 0 », il relie la ligne au ground. Lorsqu'un appareil impose un « 1 », il laisse la ligne flottante.

Comme les « 1 » laissent la ligne flottante, il faut mettre des pull-up sur les deux lignes I2C (la valeur des pull-up dépend de la vitesse de transfert, mais 2K est un bon ordre de grandeur).

Les collisions sur une ligne avec des sorties Open Drain ou Open Collector n'ont pas de conséquences désastreuses : si un appareil impose un « 0 » alors qu'un autre appareil impose un « 1 », le « 0 » reliera la ligne au ground et le « 1 » n'aura pas d'effet, la sortie étant flottante. Un appareil détectera une collision sur la ligne s'il transmet un « 1 » et que la ligne reste « 0 »…

La possibilité d'imposer un « 0 » et celle de détecter des collisions à l'intérieur d'un bit permet au bus I2C d'avoir plusieurs caractéristiques intéressantes :

1- Le bus supporte plusieurs maîtres de communication potentiels (un seul à la fois). L'arbitration du bus est faite avec la ligne SDA : si un maître lit un « 0 » pendant qu'il transmet un « 1 », il cesse la transmission.

2- Un esclave peut signaler qu'il n'est pas prêt à recevoir des données en étirant l'horloge : l'esclave impose un « 0 » sur la ligne et le maître ne peut plus basculer l'horloge…

1.4 Controller Area Network (CAN)

Le CAN a été créé dans les années 1980 pour les automobiles (release officielle en 1986 par la Society of Automotive Engineers [SAE]). Il s'agit d'un protocole de communication série, half-duplex, multipoint, avec transmission en mode différentiel et utilisant des états dominants et récessifs pour encoder les bits (comme les sorties open-drain du I2C). Contrairement au UART ou au SPI, la norme ISO 11898, qui décrit le CAN, définit la transmissions de trames complètes plutôt que la transmission d'octets uniques.

Le CAN a été créé pour contrôler les différents éléments du système électrique d'une voiture. Il est robuste et résistant aux fautes.

Les éléments et figures de https://en.wikipedia.org/wiki/CAN_bus seront présentés en classe.

1.5 Universal Serial Bus (USB)

Voir SMI_C7_USB.zip. L'information sur le USB est disponible à titre de référence et n'est pas à l'examen.

1.6 Autres interfaces séries

Il existe plusieurs autres interfaces séries qui ne sont pas décrites à l'intérieur de ces notes de cours. Voici des références pour votre information :

- Ethernet
- Inter-Integrated Sound (I2S): https://en.wikipedia.org/wiki/I%C2%B2S
- DALI: http://www.dali-ag.org/ et http://sitelec.org/cours/abati/flash/dali.htm
- 1-Wire : https://en.wikipedia.org/wiki/1-Wire
- JTAG et Serial Wire Debug : voir autres cours.
- Autres

1.7 Interfaces séries et microcontrôleurs

La plupart des microcontrôleurs modernes supportent Ethernet et le USB. Ces deux interfaces séries sont à part des autres interfaces parce qu'elles sont plus complexes et beaucoup plus rapides. Il y a souvent un bus spécial pour celles-ci. Il y a aussi toute une infrastructure de transfert de données par DMA qui accompagne les contrôleurs Ethernet et USB. Enfin, les broches servant au USB et à Ethernet sont souvent des broches dédiées à cet usage ou n'ayant pas beaucoup d'autre rôle possible.

Les autres interfaces séries sont parfois fusionnées à l'intérieur d'un unique module matériel : un USART (Universal Synchronous/Asynchronous Receiver/Transmitter) ou équivalent. Le programmeur écrit des registres du USART pour le transformer en

contrôleur de UART, SPI, I2C, I2S, CAN ou autre… Les microcontrôleurs modernes supportent au moins quatre interfaces séries, mais c'est souvent beaucoup plus!

Enfin, on retrouve dans plusieurs cas un circuit intégré d'interface entre le microcontrôleur et le bus de communication. Ce circuit adapte les niveaux de tensions, implémente la communication différentielle, protège le microcontrôleur et plus.

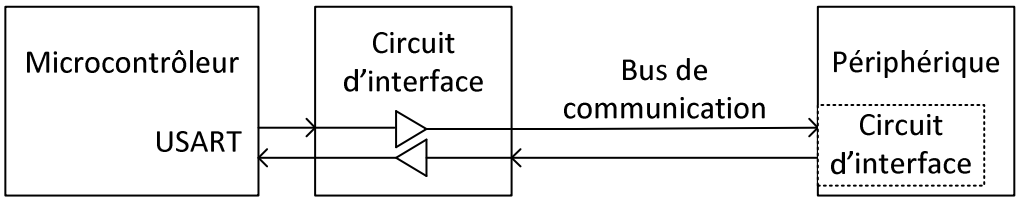


Figure 8 – Lien de communication série typique

1.8 Quelle interface série choisir?

Il y a beaucoup d'interfaces séries disponibles et ces interfaces ont toutes leurs forces et leurs faiblesses. Si on compare le RS232, le SPI, l'I2C ou le CAN, ces interfaces ont toutes leurs champs d'application.

Le RS232 est le plus lent et le plus vieux de ces protocoles. Cependant, il s'agit d'un protocole simple, dûment éprouvé, et qui permet de franchir de longues distances, surtout lorsqu'on utilise du RS485. Le RS232 ou le RS485 sont encore énormément utilisés dans des applications ou la vitesse de communication n'est pas importante (entre 10Kbps et 100Kbps), mais où le signal doit parcourir de longues distances ou être robuste au bruit (dans plusieurs milieux industriels).

Le SPI et l'I2C sont utilisés dans des applications similaires : des communications sur de courtes distances (souvent sur le même PCB) où les octets sont transférés à une vitesse de quelques centaines de kilooctets. Ces bus sont moins rapides que l'USB, mais beaucoup, beaucoup, plus simples. Le SPI est plus rapide que l'I2C (disons ~4Mbps versus 400Kbps), mais il requiert plus de broches et plus de fils… Les deux protocoles sont vulnérables au bruit et ne peuvent pas être utilisés sur de longues distances.

Le CAN sert surtout dans les voitures, mais aussi dans plusieurs autres domaines du fait de sa robustesse. Le CAN se retrouve dans de nombreuses industries comme le RS485. Il est plus rapide (environ 500kbps) que le RS485 (maximum 115200bps) et il permet plusieurs maîtres sur le bus, mais il parcourt de moins grandes distances.