Introduction aux liaisons séries filaires UART, I2C, et SPI (et détail de leurs protocoles de communication)

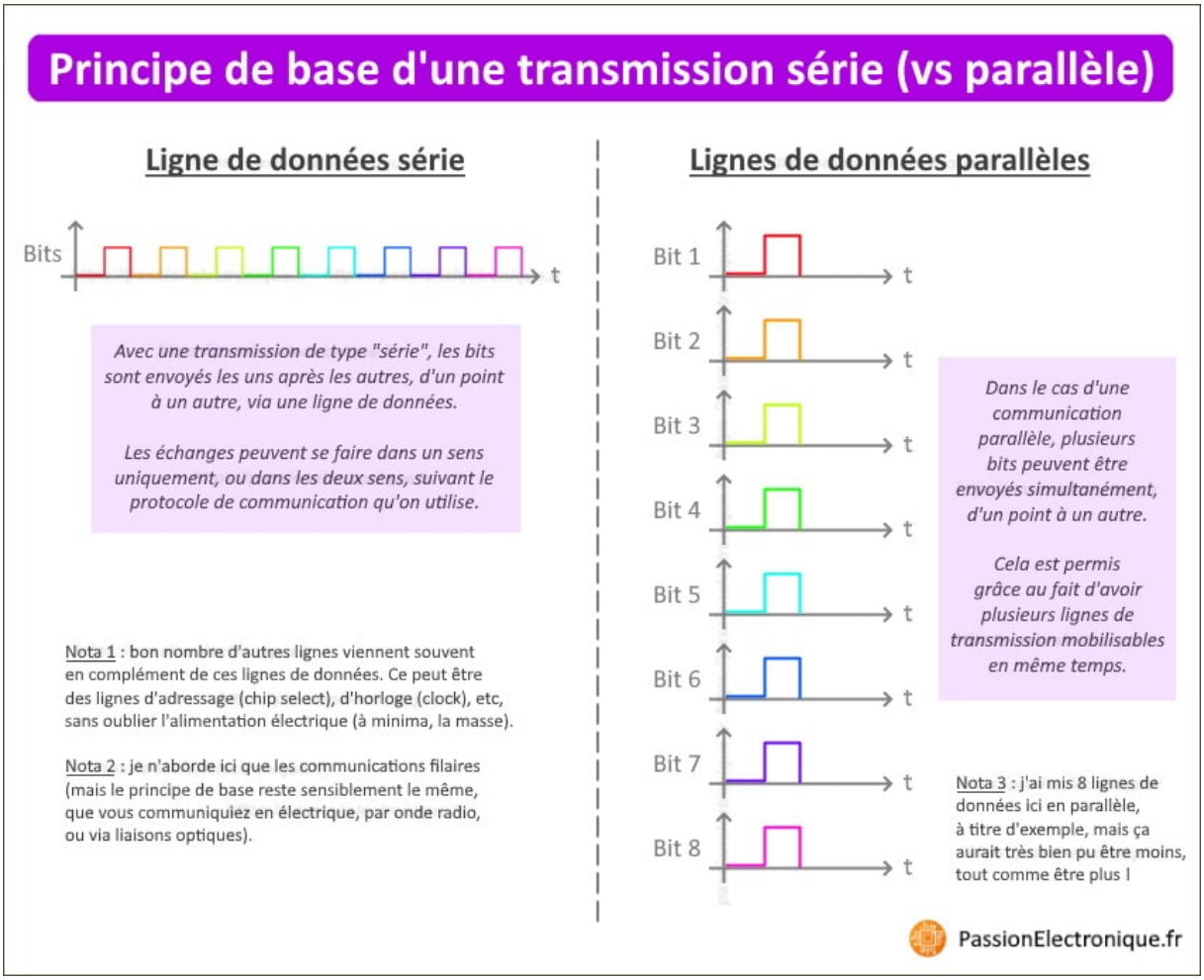
UART, I2C, SPI, … ces acronymes vous parlent sûrement, car on les retrouve souvent, en électronique ! Comme vous le savez certainement déjà, ce sont **différents types de communication série filaire**. Mais en pratique, savez-vous comment ils se câblent ? comment ils fonctionnent ? et à quel débit on peut les utiliser ?

C'est en tout cas ce que je vous propose de découvrir aujourd'hui, au travers de cet **article sur les différents types de liaisons séries filaires**. Par contre, pour rester digeste, je me limiterai ici à présenter uniquement les « principaux » protocoles de communication série, que nous rencontrons couramment de nos jours, sur nos petites cartes à microcontrôleur (comme l'Arduino, par exemple). Alors en avant !

Sommaire

Généralités

Par définition, **une liaison série est un moyen de relier deux éléments (ou plus), afin que ceux-ci puissent échanger des informations, et ce, en utilisant un minimum de fils**.



La liaison série UART

Le protocol UART (de l'anglais « Universal Asynchronous Receiver Transmitter »). Il permet à deux éléments (pas plus), de communiquer ensemble, via une liaison série filaire.

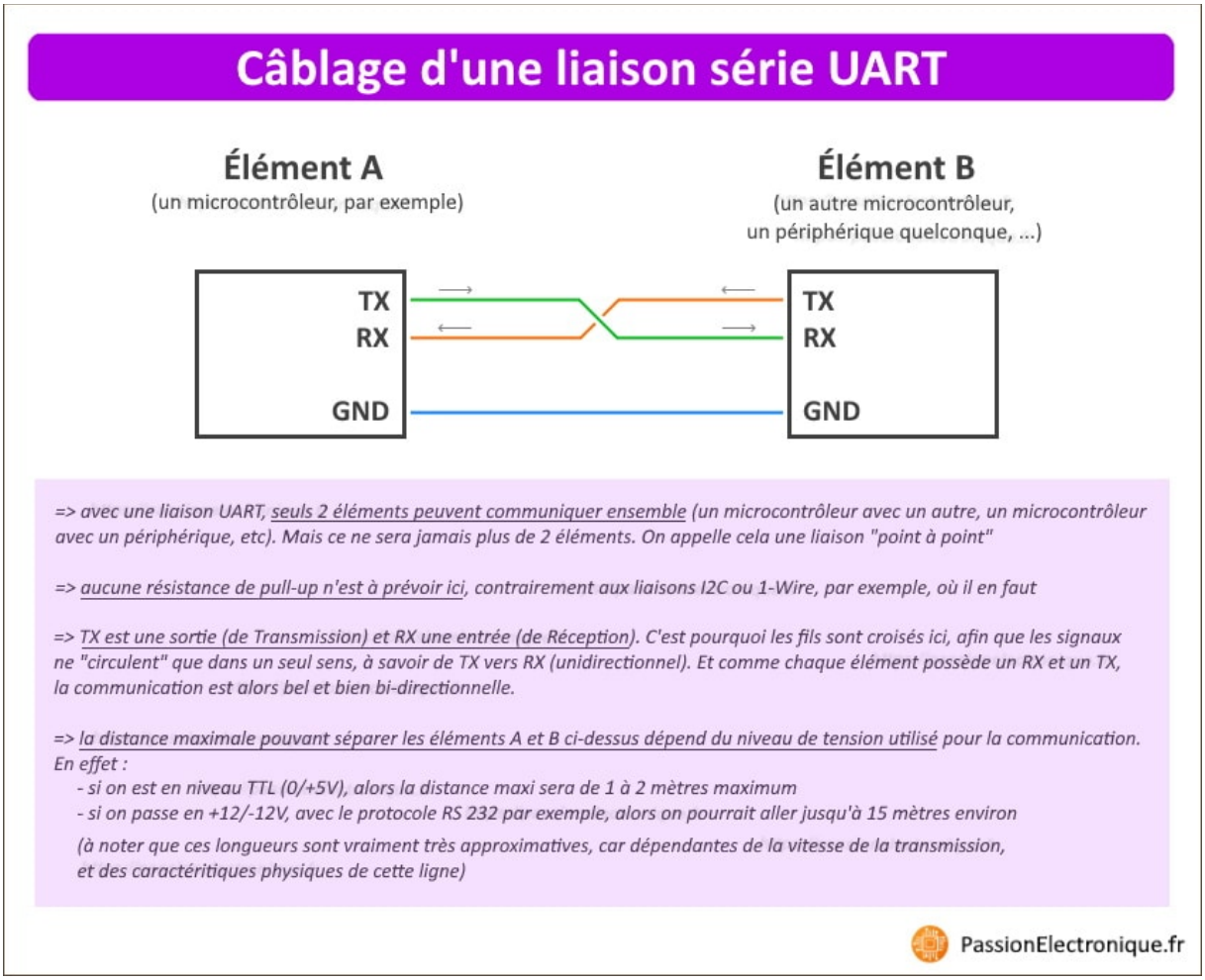
La liaison UART est ce qu'on appelle une liaison point à point, en ce sens qu'elle ne permet de relier que 2 éléments, et deux éléments seulement.

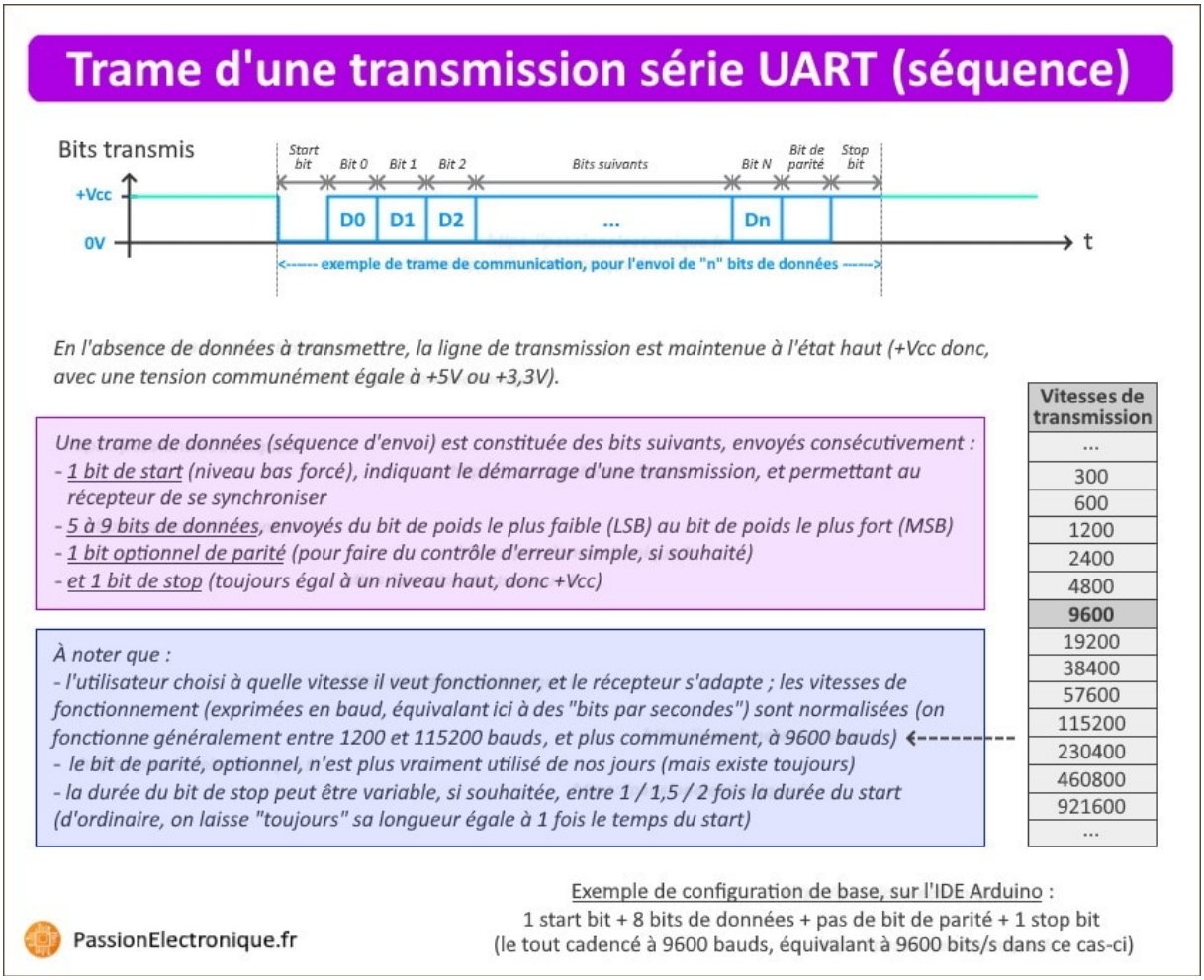
En fait, chaque élément dispose de :

- une entrée pour la réception de données, notée RX
- une sortie pour la transmission de données, notée TX

U**ne liaison filaire UART comporte 2 fils de communication, dont le premier permet de faire circuler les données dans un sens (TX → RX), et dont le second permet de faire circuler les données dans l'autre sens (RX ← TX)**.

En image, voici ce que cela donne :





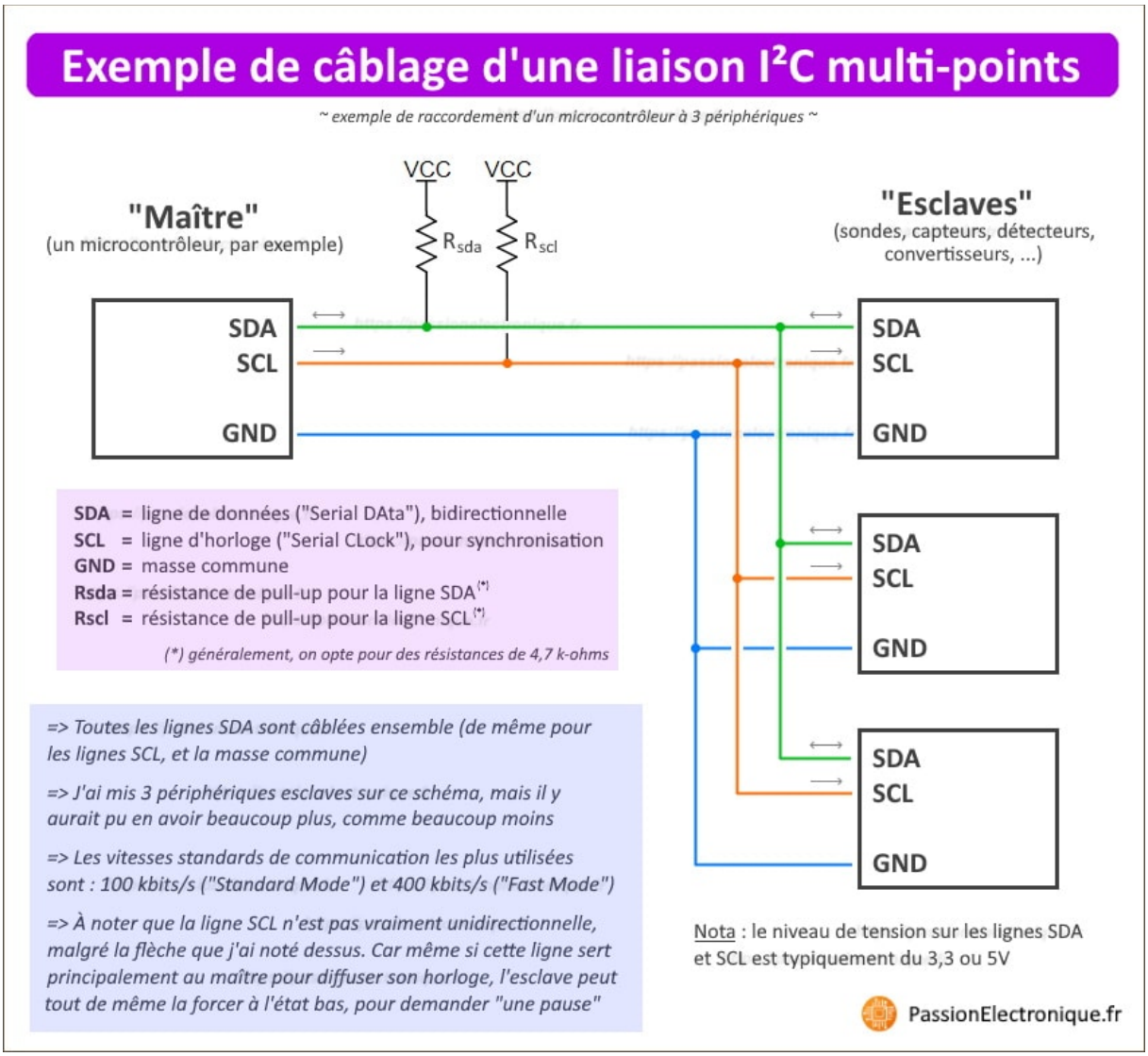
Par contre, **il est possible de communiquer sur de plus grandes distances, en utilisant par exemple le protocole RS-232 (« calqué » sur le UART)**. Ceci permet d'aller jusqu'à :

- 2 m environ, pour du 56 kbits/s
- 3,5 m environ, pour du 38400 bits/s
- 7,5 m environ, pour du 19200 bits/s
- 15 m environ, pour du 9600 bits/s (c'est la valeur la plus couramment usitée)
- 30 m environ, pour du 4800 bits/s
- 60 m environ, pour du 2400 bits/s

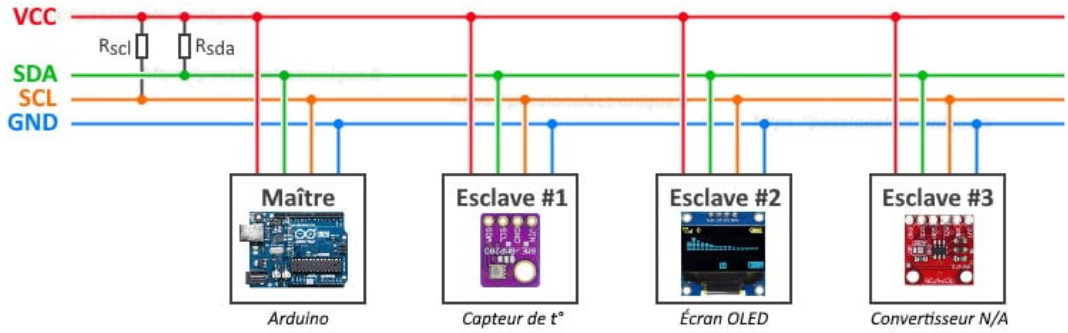
La liaison série I2C

Ce type de communication série permet l'échange de données séries filaires, notamment entre un « maître » (un microcontrôleur, par exemple) et des « esclaves » (des périphériques). On appelle également « bus I2C » l'ensemble des liens constituant l'interconnexion du « maître » avec tous ses « esclaves ».

De nos jours, **le bus I2C est devenu un moyen de communication très prisé**, dès lors qu'on cherche à interfacer un microcontrôleur avec des périphériques, tels que des :



Autre façon de réprésenter le bus I²C

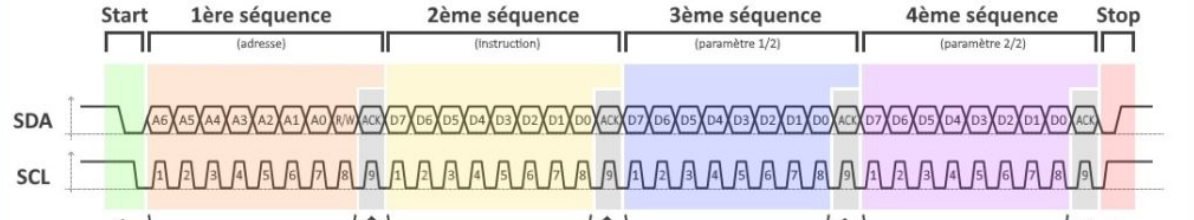


Nota : exemple de raccordement i2c d'un Arduino Uno avec 3 périphériques (un capteur de température/hygrométrie, un écran OLED, et un convertisseur DAC)

PassionElectronique.fr

Exemple "d'écriture" sur bus i2c (maître vers esclave, donc)

~ exemple d'envoi d'une commande maître vers un esclave donné, avec 2 octets de données supplémentaires (des paramètres, par exemple) ~



❷ *Le maître communique l'adresse de l'esclave à qui il compte s'adressser (bits A6 à A0), et fixe R/W à l'état bas (0 volt), car il "écrit" sur le bus I2C*

❹ *Le maître envoie ensuite sa commande, codée sur 8 bits (et attend un "accusé réception" de la part de l'esclave)*

❻ *Le maître envoie à présent un 1er octet de données (par exemple)*

❽ *Le maître envoie ensuite un 2ème octet de données, par exemple*

❿ *Le maître arrête son horloge sur SCL. Puis les lignes SDA et SCL reviennent à l'état haut.*

❶ *Le maître force la ligne SDA à l'état bas, pour indiquer qu'il souhaite s'adresser à quelqu'un. Puis il lance son horloge sur SCL, en commençant par la mettre à l'état bas.*

❸ *L'esclave se trouvant à cette adresse répond en forçant la ligne SDA à l'état bas (nota : si la ligne SDA reste à l'état haut, alors le maître saura que l'esclave n'est pas connecté, ou qu'il a rencontré un soucis).*

❺ *L'esclave accuse bonne réception de la commande, en forçant la ligne SDA à l'état bas.*

❼ *L'esclave accuse pareillement la bonne réception des données, en forçant à l'état bas la ligne SDA.*

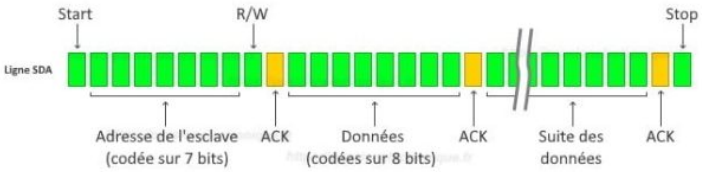
❾ *L'esclave accuse encore bonne réception des données reçues.*

Nota 1 : on aurait très bien pu avoir que deux séquences seulement ici (sans complément d'information, donc). En fait, dites vous qu'il ne s'agit là que d'un exemple parmi tant d'autres.
Nota 2 : des pauses auraient pu s'insérer entre les séquences, si l'esclave l'avait demandé ; en pratique, il aurait fait cela en abaissant et maintenant la ligne SCL au niveau bas, le temps de la pause voulue.

PassionElectronique.fr

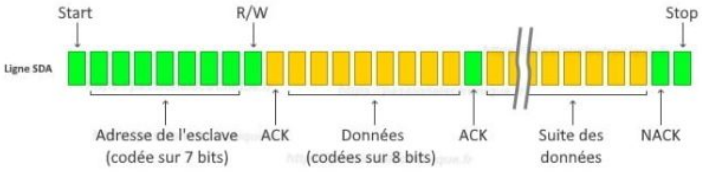
Lecture/écriture sur le bus I2C : qui fait quoi ?

ÉCRITURE (le maître envoie un "ordre" à l'esclave)



Remarques :
-> le R/W=0 ici, car on effectue une "écriture" sur le bus I2C
-> le premier ACK=0, lorsque l'esclave répond "présent" au maître
-> le ou les ACK suivants valent 0, à chaque fois que l'esclave accuse "bonne réception" des données envoyées par le maître
-> à noter que ces données peuvent très bien se limiter à un seul octet (pas nécessairement plusieurs)

LECTURE (le maître "questionne" l'esclave, qui lui retourne une réponse)



Remarques :
-> le R/W=1 ici, car on effectue une "lecture" sur le bus I2C
-> le premier ACK est fait pas l'esclave (il le met à "0" pour dire qu'il est bien présent)
-> le ou les ACK suivants sont faits par le maître, qui met la ligne SDA à 0 à chaque octet reçu de l'esclave (à noter que lorsque le maître n'attend plus de données de l'esclave, il émet au final un NACK au lieu d'un ACK, c'est à dire un "1" au lieu d'un "0")

= maître *(état imposé par le maître)*

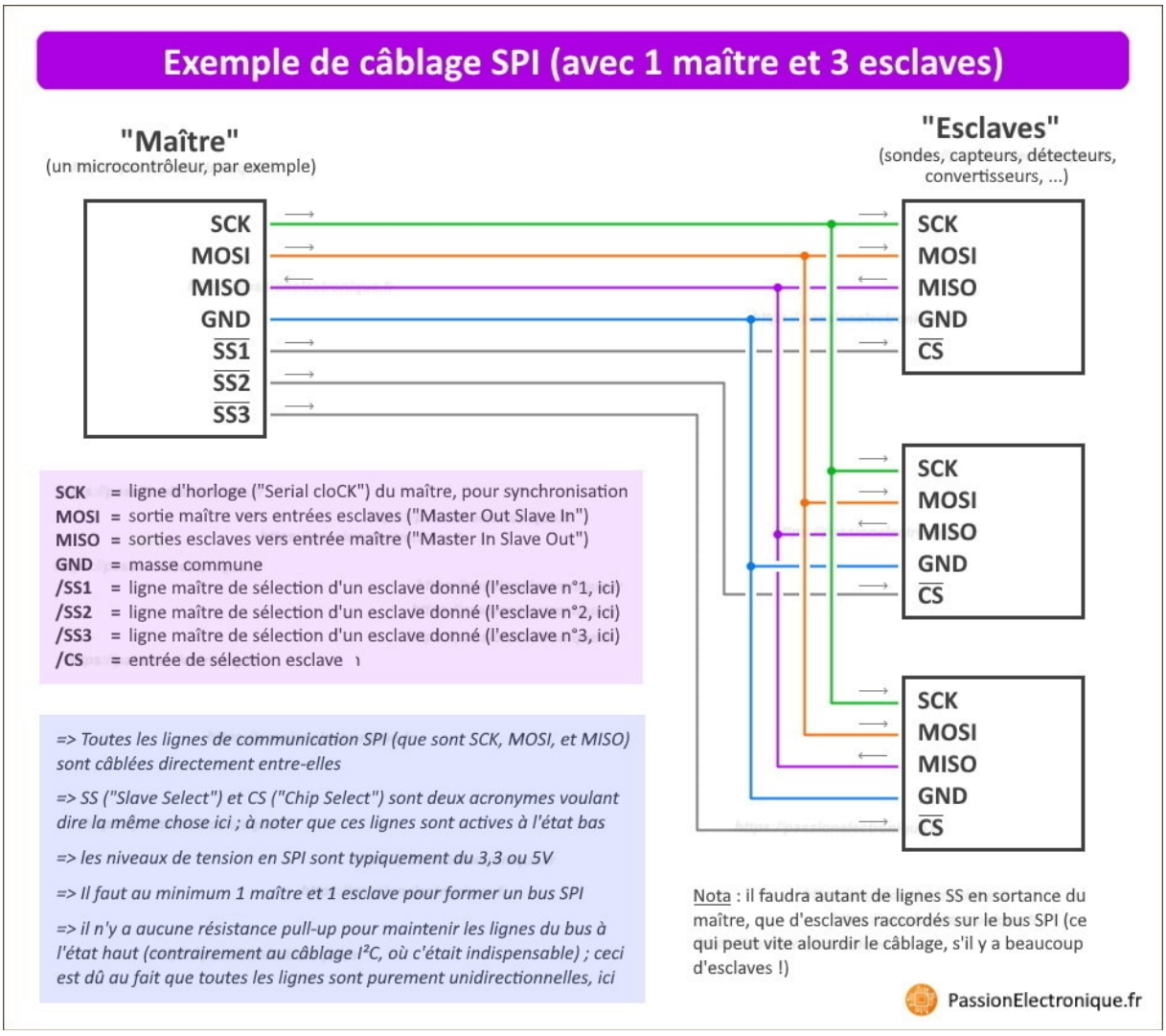
= esclave *(état imposé par l'esclave)*

PassionElectronique.fr

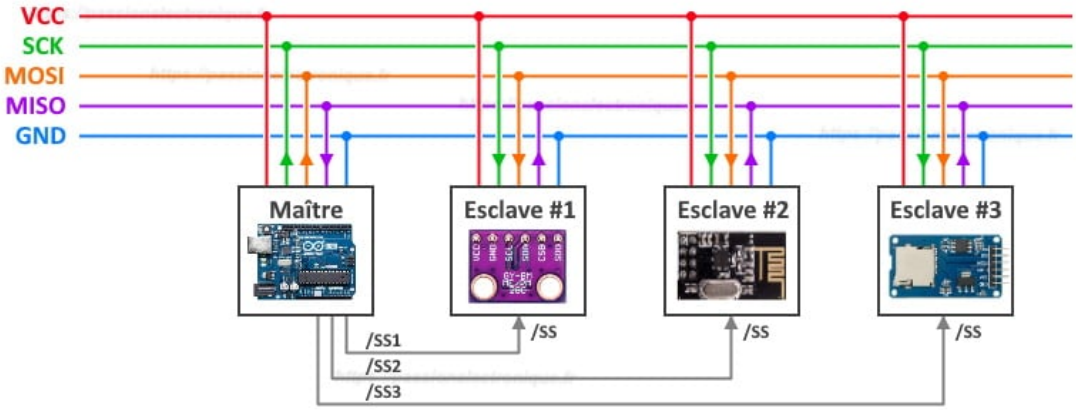
La liaison série SPI

Il s'agit d'une liaison maître/esclaves, avec, généralement :

- un microcontrôleur comme maître
- et des périphériques (capteurs, sondes, convertisseurs, …) comme esclaves



Autre façon de réprésenter le bus SPI

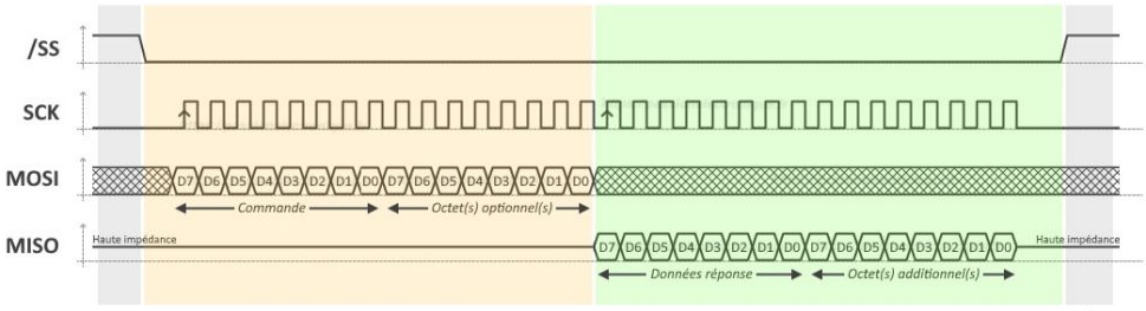


Nota : exemple de raccordement SPI d'un Arduino Uno avec 3 périphériques (un capteur de température/hygrométrie, un émetteur/récepteur radio, et un lecteur de carte SD)

PassionElectronique.fr

Exemple de "lecture" sur le bus SPI

*Dans cet exemple, le maître questionne un esclave donné, et **LIT** la réponse en retour (d'où le terme "lecture SPI")*



❶ *Pour commencer, le maître abaisse la ligne /SS, reliée à l'esclave à qui il souhaite parler.*

❷ *Ensuite, le maître génère une horloge sur la ligne SCK, et transmet bit à bit sa commande via la ligne MOSI. Ici, la commande fait 8 bits, donc 1 octet (ce qui est standard). À noter que plusieurs de ces octets peuvent être émis, à la suite les uns des autres.*

❸ *Une fois le message du maître intégralement reçu par l'esclave, ce dernier répond avec 1 ou plusieurs "paquets" de données (faisant 8 bits chacun, le plus couramment)*

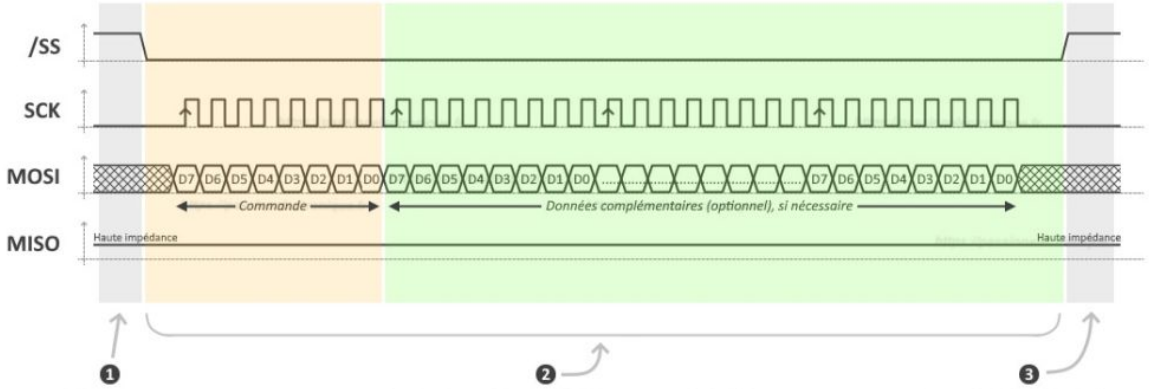
❹ *Enfin, une fois que le maître a reçu tous les "octets" qu'il attendait, il arrête son horloge sur SCK, et remet la ligne /SS à l'état haut.*

Remarques :
- dans cet exemple, tous les "blocs" de données font 8 bits (mais ce ne sera pas toujours le cas)
- dans cet exemple, chaque bit de donnée est "validé" sur front montant d'horloge SCK (là aussi, c'est un paramètre qui peut changer)
- il n'y a aucun "ACKnowlegment" (accusé de réception, si vous préférez) de la part de l'esclave ; ce qui fait que le maître pourrait très bien parler dans le vide sans arrêt, sans même le savoir
- en dehors de toute transmission, la ligne MISO est "déconnectée" ; elle est donc à haute impédance (aussi noté "High Z", en anglais)

PassionElectronique.fr

Exemple "d'écriture" sur le bus SPI

*Dans cet exemple, le maître envoie un ordre à l'esclave, en **ÉCRIVANT** ses bits de consigne sur la ligne MOSI (d'où le terme "écriture SPI")*



❶ *Le maître commence par mettre au niveau bas la ligne /SS, reliée à l'esclave à qui il souhaite envoyer des ordres*

❷ *Le maître envoie ensuite 1 ou plusieurs paquets de données, représentant l'ordre à exécuter, accompagné des éventuels paramètres complétant celui-ci. Ces "paquets" font classiquement 8 bits (donc un octet), sauf exception. Le tout est cadencé par une horloge, présente sur la ligne SCK, générée par le maître.*

❸ *Pour finir, le maître arrête son horloge sur SCK, et remet la ligne /SS au niveau haut, afin de mettre un terme à sa communication avec cet esclave là.*

Remarques :
- dans cet exemple, tous les "blocs" de données font 8 bits (attention, ce n'est pas forcément systématique)
- dans cet exemple, chaque bit de donnée est "validé" sur front montant d'horloge SCK (là aussi, c'est un paramètre qui peut changer)
- il n'y a aucun "ACKnowlegment" (accusé de réception, si vous préférez) de la part de l'esclave ; ce qui fait que le maître pourrait très bien parler dans le vide sans arrêt, sans même le savoir
- comme le maître ne fait que parler ici, l'esclave n'aura pas de réponse à retourner ; la ligne MISO sera donc déconnectée (haute impédance)

PassionElectronique.fr

Différentes notations des entrées/sorties SPI (exemple avec 1 maître et 2 esclaves)

Schéma avec SCK/MOSI/MISO

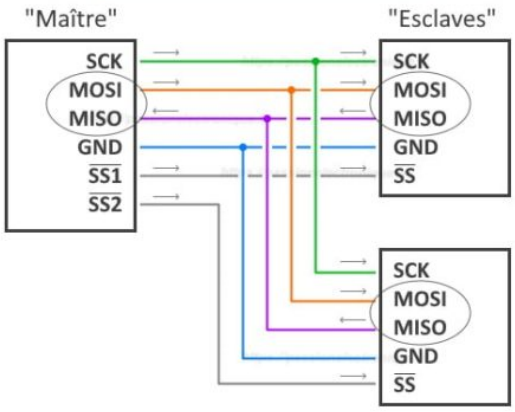
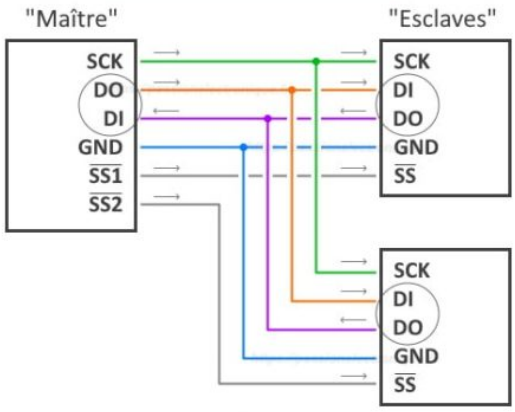


Schéma avec SCK/DO/DI



Nota : ces 2 schémas sont strictements identiques, même si les libellés diffèrent

En fait :
- **MOSI** = Master Out / Slave In = Data OUT du maître / Data IN de l'esclave = **DO** du côté du maître / **DI** du côté de l'esclave
- **MISO** = Master In / Slave Out = Data IN du maître / Data OUT de l'esclave = **DI** du côté du maître / **DO** du côté de l'esclave

Avec :
- **DO** (data output) = **SO** (serial output) = **SDO** (serial data out)
- **DI** (data input) = **SI** (serial input) = **SDI** (serial data in)
- **SS** (slave select) = **CS** (chip select)
- et **SCK** = **SCLK** = **SCL** (tout ça veut dire là aussi la même chose, à savoir "serial clock", donc "horloge série")

PassionElectronique.fr

Les autres protocoles de communication série filaires

- au bus CAN, très prisé dans le milieu automobile, qui permet avec 2 fils seulement (hors alimentation) de relier 20 à 30 éléments ensemble (selon si on fonctionne à 125 kbits/s, ou 1 Mbits/s) ; sans parler des longueurs de communication qui sont vraiment appréciables : environ 30m à 1 Mbits/s, et jusqu'à 5 km à 10 kbits/s ! Ce type de liaison filaire série est d'ailleurs assez facile à mettre à œuvre, notamment avec nos arduino, et aux mini shields « tout équipés » MCP2515 / TJA1050.

- au bus 1-Wire, popularisé avec les sondes de température DS18B20 de chez Dallas Semiconductor ; ici, un seul fil suffit, et la masse ! Difficile de faire plus simple, niveau câblage, bien que des fois, il faille rajouter le +Vcc ! Et c'est d'autant plus simple à mettre en œuvre qu'il existe de nombreuses librairies gérant cela (car à coder, ce serait fastidieux !). Du reste, une liaison 1-Wire est bidirectionnelle, fonctionne en semi-duplex (un seul sens à la fois, au niveau de la transmission), nécessite une résistance de pull-up, et fonctionne à 16 kbits/s environ, en standard (ce qui, au passage, permet d'aller sur « d'assez longues » distances)

Tableau comparatif UART / I2C / SPI (synthèse)

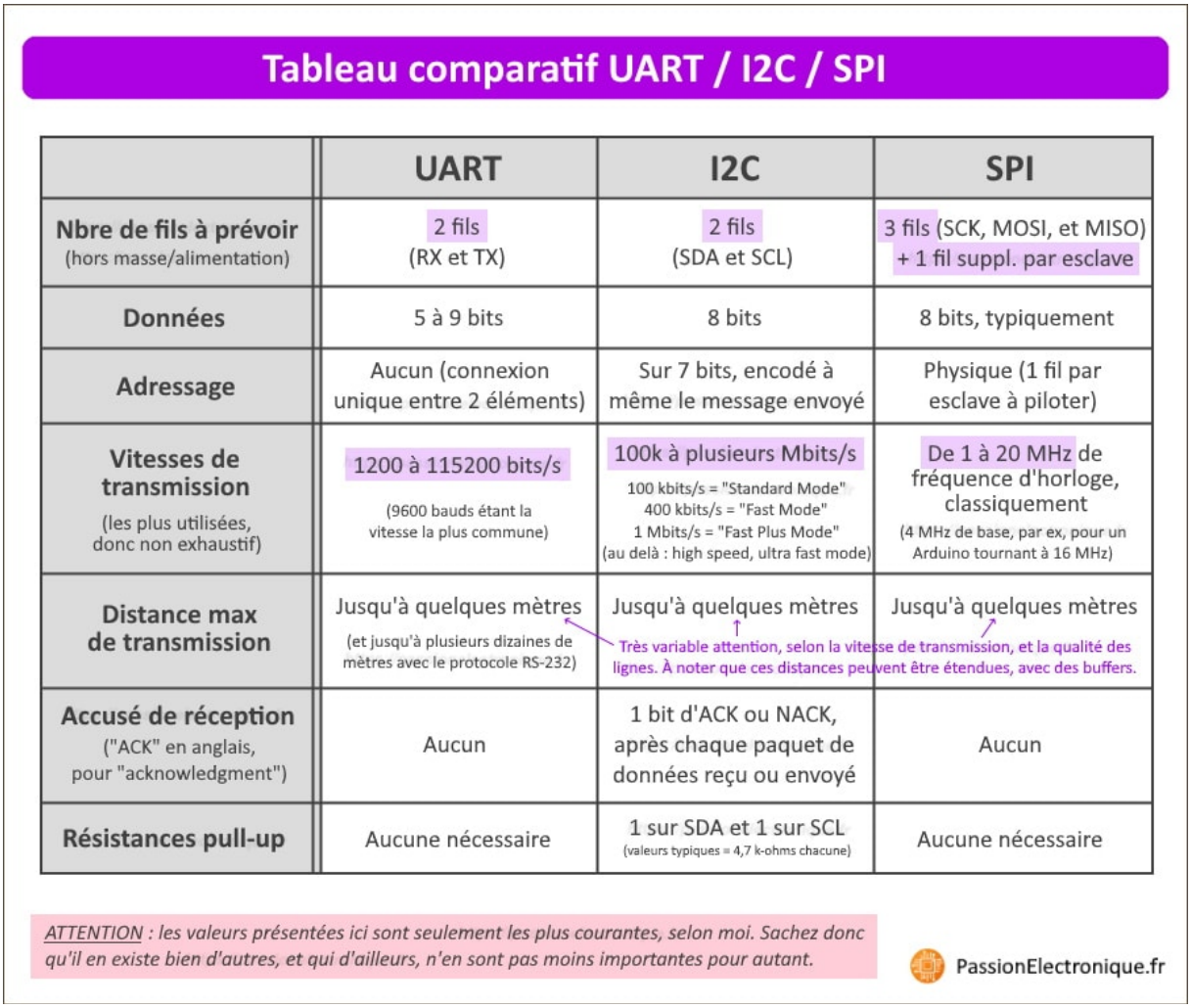


Tableau comparatif UART / I2C / SPI

	UART	I2C	SPI
Nbre de fils à prévoir (hors masse/alimentation)	2 fils (RX et TX)	2 fils (SDA et SCL)	3 fils (SCK, MOSI, et MISO) + 1 fil suppl. par esclave
Données	5 à 9 bits	8 bits	8 bits, typiquement
Adressage	Aucun (connexion unique entre 2 éléments)	Sur 7 bits, encodé à même le message envoyé	Physique (1 fil par esclave à piloter)
Vitesses de transmission (les plus utilisées, donc non exhaustif)	1200 à 115200 bits/s (9600 bauds étant la vitesse la plus commune)	100k à plusieurs Mbits/s 100 kbits/s = "Standard Mode" 400 kbits/s = "Fast Mode" 1 Mbits/s = "Fast Plus Mode" (au delà : high speed, ultra fast mode)	De 1 à 20 MHz de fréquence d'horloge, classiquement (4 MHz de base, par ex, pour un Arduino tournant à 16 MHz)
Distance max de transmission	Jusqu'à quelques mètres (et jusqu'à plusieurs dizaines de mètres avec le protocole RS-232)	Jusqu'à quelques mètres	Jusqu'à quelques mètres
Accusé de réception ("ACK" en anglais, pour "acknowledgment")	Aucun	1 bit d'ACK ou NACK, après chaque paquet de données reçu ou envoyé	Aucun
Résistances pull-up	Aucune nécessaire	1 sur SDA et 1 sur SCL (valeurs typiques = 4,7 k-ohms chacune)	Aucune nécessaire

Très variable attention, selon la vitesse de transmission, et la qualité des lignes. À noter que ces distances peuvent être étendues, avec des buffers.

ATTENTION : les valeurs présentées ici sont seulement les plus courantes, selon moi. Sachez donc qu'il en existe bien d'autres, et qui d'ailleurs, n'en sont pas moins importantes pour autant.

PassionElectronique.fr