SPI et Raspberry Pi*

Christophe BLAESS

La communication suivant le protocole SPI est très rapide, car les fréquences sont élevées et le dialogue totalement bidirectionnel « full-duplex ». Ce type de communication sert typiquement pour établir un lien entre un processeur et des périphériques (capteurs, etc.) mais on peut aussi l'utiliser pour dialoguer avec un micro-contrôleur comme nous le ferons dans l'article suivant.

SPI

Le protocole **SPI** (*Serial Peripheral Interface*) implémente une liaison série synchrone entre un maître et un esclave. Lorsqu'un seul esclave est employé, trois signaux seulement (outre la masse) sont nécessaires.

Le maître produit une horloge (signal **SCLK**) envoyé à l'esclave. Sur certaines transitions de cette horloge, l'esclave lira (sur le signal nommé **MOSI** – *Master Out Slave In*) ou écrira (sur le signal nommé **MISO** – *Master In Slave Out*) des données. Il existe plusieurs dénominations suivant les fabricants de matériel pour décrire ces signaux. Il est recommandé d'utiliser la notation **MISO** / **MOSI** (la plus répandue) car elle supprime toute ambiguïté : la broche **MOSI** d'un maître doit toujours être reliée à la broche **MOSI** d'un esclave, et de même pour leurs broches **MISO**.

Si plusieurs esclaves doivent être reliés au même hôte, il pourront être branchés en parallèle (toutes les broches **MISO** reliées entre elles, et toutes les broches **MOSI** également) mais il faudra un signal supplémentaire (**CS** – *Chip Select*) pour chacun d'eux afin de choisir avec lequel la communication est établie à un moment donné.

On peut noter qu'il existe également un schéma de connexion multi-esclaves nommé « *daisy chain* » où ils se transmettent les données en série, la sortie **MISO** de chaque esclave (sauf le dernier de la chaîne) étant reliée à l'entrée **MOSI** du suivant. Ceci sort du propos de notre article.

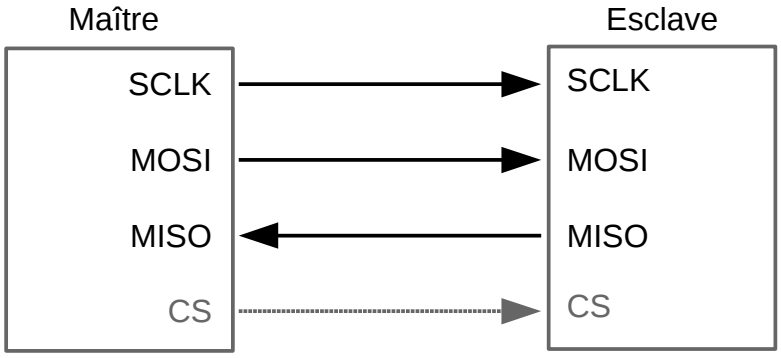

Fig. 1 – Connexions entre maître et esclave.

Le noyau Linux *vanilla* n'implémente que le côté maître du protocole, même si le support de la partie esclave existe sous forme de *patches* indépendants que l'on peut ajouter à des drivers de contrôleurs SPI bien spécifiques.

Signal d'horloge

Le signal d'horloge **SCLK** est essentiel car c'est lui qui cadence tous les échanges. Il est produit par le maître pour séquencer les transmissions de bits sur les lignes **MOSI** et **MISO**. Le protocole **SPI** étant un standard de fait, sans véritable norme sous-jacente, plusieurs modes de fonctionnement sont tolérés pour l'utilisation du signal d'horloge.

Il existe essentiellement quatre modes possibles. Pour que deux composants puissent dialoguer en SPI, il est donc indispensable de commencer par préciser quel mode nous souhaitons utiliser. La plupart des composants évolués sont capables d'utiliser les quatre modes SPI classiques. Le problème est que suivant les constructeurs, les modes ne sont pas toujours numérotés de manière identique. Nous verrons d'ailleurs une différence entre la notation utilisée par Linux et celle employée dans la documentation du micro-

* Cet article est paru dans Gnu/Linux Magazine Hors Série numéro 75 « *Raspberry Pi Avancé* » en novembre 2014, disponible sur `https://boutique.ed-diamond.com/anciens-numeros/789-gnulinux-magazine-hs-75.html`

contrôleur Texas Instrument MSP430.

Pour déterminer le mode SPI, il est nécessaire de prendre en considération deux paramètres : la polarité (nommée habituellement **POL**) et la phase de l'horloge (nommée **PHA**). En les notant sous forme binaire nous pourrons obtenir la valeur des quatre modes possibles.

Lorsque aucun échange ne se déroule, le signal **SCLK** est au repos. Il peut être au niveau haut ou au niveau bas. C'est ce que l'on nomme la **polarité d'horloge**. Dans un mode de fonctionnement avec une horloge à polarité positive (**POL=1**), **SCLK** est au repos au niveau haut, et descend au niveau bas pour indiquer un créneau actif. Inversement, dans un mode à polarité négative (**POL=0**) **SCLK** est au niveau bas au repos, et monte au niveau haut en début de créneau.

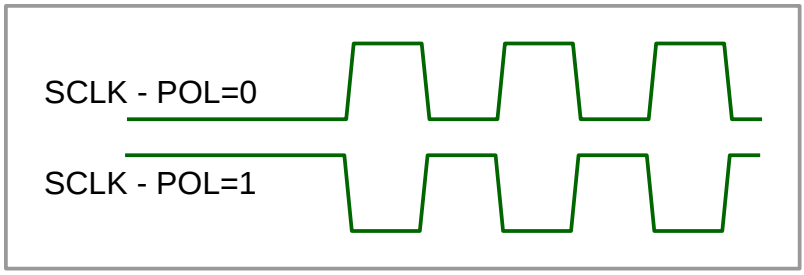

Fig. 2 – Polarité d'horloge.

En résumé, si **POL=0**, le premier front du signal SCLK est un front montant et le second un front descendant. Inversement, dans le cas où **POL=1**, le premier front est descendant et le second montant.

Le second paramètre pour spécifier le mode de communication SPI utilisé est la **phase d'horloge** (**PHA**) et indique sur quels fronts de celle-ci on cadencera les communication MISO et MOSI.

Lorsque la phase **PHA=1**, les signaux seront établis en sortie (du côté *Slave Out* ou *Master Out*) lors du premier front et lus en entrée (du côté *Master In* ou *Slave In*) sur le second front.

Inversement dans le cas où la phase **PHA=0** les signaux seront lus en entrée sur le premier front et établis en sortie sur le second front.

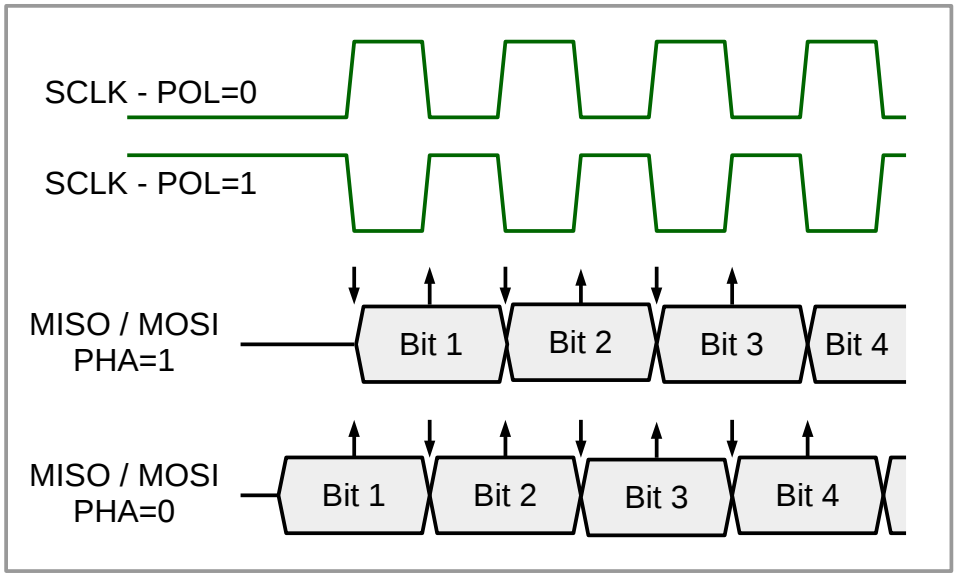

Fig. 3 – Polarité et phase d'horloge.

Sur la figure 3, on a représenté par une flèche vers le bas le moment où le signal est écrit en sortie et par une flèche vers le haut l'instant où il est lu en entrée.

On remarque que lorsque **PHA=0**, le périphérique esclave doit écrire sa sortie sur la ligne **MISO** avant le premier front d'horloge. Pour cela il doit déclencher son écriture à l'activation du signal *Chip Select* qui précède toujours le premier front d'horloge, ceci au prix d'un câblage supplémentaire. Pour économiser les lignes de communication et garder un modèle le plus simple possible nous utiliserons dans la suite de cet article et dans le suivant une configuration avec **PHA=1**.

Pour simplifier la mise en œuvre des communications SPI, on regroupe généralement les composantes de polarité et de phase en une seule valeur : le **mode** SPI. Cette notation est bien répandue et relativement

standardisée, mais elle n'est pas employée par tous les constructeurs.

POL	PHA	Mode SPI
0	0	0
0	1	1
1	0	2
1	1	3

MISO et MOSI

Une fois la configuration de **SCLK** réalisée, une seconde surprise attend le programmeur qui découvre le protocole SPI. Il s'agit d'une communication bidirectionnelle *full-duplex*. Autrement dit, à chaque fois que le maître envoie un bit vers un esclave, il en reçoit un en retour en provenance de cet esclave.

L'image qui me vient à l'esprit est celle d'une chaîne de vélo : à chaque maillon qui part vers le pignon de la roue arrière (chaque bit émis sur le signal **MOSI**), un maillon revient sur le plateau du pédalier (un bit est reçu sur **MISO**).

Lorsqu'on voudra faire une simple opération de lecture sur un périphérique, il faut donc être conscient que celui-ci recevra également des données de notre part. De même le schéma classique « écriture d'une commande » « lecture du résultat » « écriture de la commande suivante », etc. doit être modifié pour prendre en considération le fait que la lecture du premier résultat s'accompagnera dèjà de l'écriture de la seconde commande.

SPI sur Raspberry Pi

Le Raspberry Pi propose deux ports SPI accessibles sur son connecteur d'extension P1.

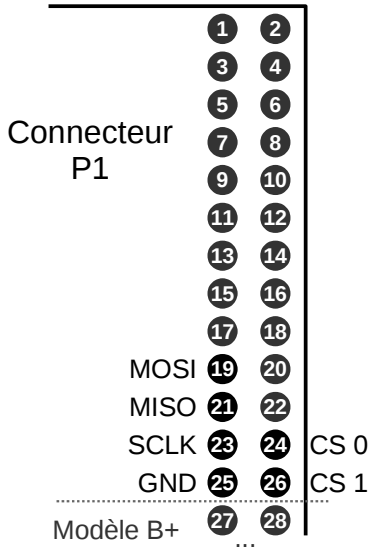

Fig 4 – Connecteur d'extension du Raspberry Pi.

Les broches **MOSI** (19), **MISO** (21) et **SCLK** (23) sont reliées directement à celles des composants esclaves. Chaque broche **CS0** (24) ou **CS1** (26) ne peut être connectée qu'à l'entrée **CS** d'un seul esclave.

Le contrôleur SPI est intégré dans le *system-on-chip* Broadcom 2835 (de la famille 2708) au cœur du Raspberry Pi. Il faut donc charger dans le noyau le module capable de le piloter. Celui-ci se nomme **spi-bcm2708.ko**. Par défaut les distributions Linux comme la Raspbian empêchent son chargement au démarrage car il ne concerne qu'un nombre très réduit d'utilisateurs et occupe des ressources système (de la mémoire notamment).

Si l'on veut charger le module dynamiquement, il suffit pour cela de saisir :

```
pi@raspberrypi:~$ sudo modprobe spi-bcm2708
```

Si on préfère que le module soit chargé automatiquement au démarrage, il faut éditer le fichier **/etc/modprobe.d/raspi-blacklist.conf** et commenter la ligne (en la précédant d'un dièse '**#**')

```
blacklist  spi-bcm2708
```

Pour accéder aux interfaces SPI depuis l'espace utilisateur, le noyau nous fournit des points d'entrées sous forme de fichiers spéciaux dans **`/dev`**. Ceci nécessite le chargement d'un second module particulier :

```
pi@raspberrypi:~$ sudo  modprobe  spidev
```

Dès le chargement de ce module, les fichiers spéciaux suivants apparaissent :

```
pi@raspberrypi:~$ ls  -l  /dev/spi*
crw-rw---T 1 root spi 153, 0 Aug 11 20:26 /dev/spidev0.0
crw-rw---T 1 root spi 153, 1 Aug 11 20:26 /dev/spidev0.1
```

Naturellement, le fichier **`/dev/spidev0.0`** correspond au périphérique esclave sélectionné par la broche **`CS0`** et le fichier **`/dev/spidev0.1`** à celui correspondant à la broche **`CS1`**.

Pour configurer les paramètres de communication sur un port SPI, le noyau met à notre disposition un appel système **`ioctl()`**. Pour la communication proprement dite, on peut combiner des appels système **`read()`** et **`write()`**, mais cela ne permet pas de tirer pleinement parti du transfert *full-duplex*. En pratique on préférera un appel **`ioctl()`** qui permet de communiquer en s'appuyant sur des structures **`spi_ioc_transfer`** décrites dans le fichier **`<linux/spi/spidev.h>`**.

Projet spi-tools

La programmation *full-duplex* en C avec les **`ioctl()`** est parfois un peu complexe. Pour établir rapidement un lien SPI avec un périphérique depuis un script *shell* par exemple, je vous propose d'utiliser un petit package que j'ai développé récemment, et qui permet de simplifier la configuration et la communication bidirectionnelle. Il s'agit d'un projet libre nommé **`spi-tools`**. Il n'est pas encore intégré dans les distributions, et sera donc téléchargé et compilé sur le Raspberry Pi. Il n'y a pas de dépendances particulières, la compilation est très simple.

Commençons par télécharger le projet :

```
pi@raspberrypi:~$ git  clone  https://github.com/cpb-/spi-tools.git
Cloning into 'spi-tools'...
remote: Counting objects: 53, done.
remote: Total 53 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (53/53), done.
```

Puis on le compile très simplement ainsi

```
pi@raspberrypi:~$ cd spi-tools/
pi@raspberrypi:~/spi-tools$ make
cc -Wall -DVERSION=\"0.3.0\"    spi-config.c   -o spi-config
cc -Wall -DVERSION=\"0.3.0\"    spi-pipe.c   -o spi-pipe
```

Le projet évolue encore, la version que vous téléchargerez sera probablement différente de celle ci-dessus.

Par défaut les exécutables sont installés dans **`/usr/sbin`**, ce qui réclame les droits *root* (toutefois il est possible de modifier ce répertoire en remplissant la variable d'environnement **`INSTALL_DIR`** avant le **`make install`**)

```
pi@raspberrypi:~/spi-tools$ sudo make install
install spi-config spi-pipe  "/usr/sbin/"
```

Le premier outil est **`spi-config`**, qui permet de consulter ou de modifier les paramètres de communication sur un port SPI.

```
pi@raspberrypi:~/spi-tools$ spi-config
spi-config: no device specified (use option -h for help).
```

Demandons de l'aide pour voir la liste des options :

```
pi@raspberrypi:~/spi-tools$ spi-config -h
```

```
usage: spi-config options...
 options:
  -d --device=<dev>  use the given spi-dev character device.
  -q --query         print the current configuration.
  -m --mode=[0-3]    use the selected spi mode.
          0: low iddle level, sample on leading edge
          1: low iddle level, sample on trailing edge
          2: high iddle level, sample on leading edge
          3: high iddle level, sample on trailing edge
  -l --lsb={0,1}     LSB first (1) or MSB first (0)
  -b --bits=[7...]   bits per word
  -s --speed=<int>   set the speed in Hz
  -h --help          this screen
  -v --version       display the version number
```

L'option **`-q`** permet donc de consulter la configuration actuelle, et l'option **`-d`** de préciser le port concerné.

```
pi@raspberrypi:~/spi-tools$ spi-config  -q  -d /dev/spidev0.0
/dev/spidev0.0: mode=0, lsb=0, bits=8, speed=500000
```

Que signifient ces paramètres ?

- **`mode`** : il s'agit bien sûr du mode SPI. En consultant le tableau plus haut, nous voyons que **`POL=0`** et **`PHA=0`**.

- **`lsb`** : les communications en SPI se font généralement en transmettant le bit de poids fort en premier. En indiquant **`lsb=1`** on peut inverser le sens de transmission des octets.

- **`bits`** : le nombre de bits par caractère.

-**`speed`** : la vitesse de transmission (fréquence du signal d'horloge) indiqué en Hz. Les communications SPI se font généralement à des vitesses plutôt élevées (plusieurs MHz), ce qui limite la distance entre les équipements pour éviter les effets parasites sur les signaux. Ici, l'horloge est configurée pour une fréquence de 500 kHz.

Comme précisé plus haut, nous allons choisir un mode SPI ou la phase de l'horloge est telle que l'écriture des données **`MISO`** et **`MOSI`** se fasse sur le premier front. Par exemple le mode 1.

```
pi@raspberrypi:~$ spi-config  -d /dev/spidev0.0  -m 1
pi@raspberrypi:~$ spi-config  -d /dev/spidev0.0  -q
/dev/spidev0.0: mode=1, lsb=0, bits=8, speed=500000
```

Le second outil du package, **`spi-pipe`**, permet de dialoguer en *full-duplex* avec un périphérique, en redirigeant son entrée standard et sa sortie standard vers le port SPI indiqué. Nous en verrons des exemples d'utilisation dans le prochain article.

Conclusion

Le Raspberry Pi nous permet de dialoguer facilement avec un ou deux périphériques en utilisant le protocole SPI, par l'intermédiaire des ports accessibles directement sur ses broches d'extensions. Dans le prochain article nous allons établir une communication avec un microcontrôleur.

Christophe Blaess (christophe@blaess.fr)
www.blaess.fr/christophe